Universidad Autónoma de San Luis Potosí
Faculty of Engineering

Automatic web code generation from a design of a website image using machine learning.

THESIS

To obtain the degree of:

Master's in Computer Engineering

Presented by:

Gerardo Franco Delgado

Advisor:

Dr. Juan Carlos Cuevas Tello

San Luis Potosí, SLP, México
June 2024

Universidad Autónoma de San Luis Potosí
Facultad de Ingeniería
Centro de investigación y Estudios de Posgrado

Generación automática de código a partir del diseño de una imagen de un sitio web utilizando aprendizaje automático.

TESIS

Para obtener el grado de:

Maestría en Ingeniería de la Computación

Presenta:

Gerardo Franco Delgado

Asesor:

Dr. Juan Carlos Cuevas Tello

San Luis Potosí, SLP, México
Junio 2024

UASLP
Universidad Autónoma de San Luis Potosí

FACULTAD DE INGENIERÍA

14 de diciembre de 2023

ING. GERARDO FRANCO DELGADO
P R E S E N T E.

En atención a su solicitud de Temario, presentada por el **Dr. Juan Carlos Cuevas Tello,** Asesor de la Tesis que desarrollará Usted, con el objeto de obtener el Grado de **Maestro en Ingeniería de la Computación**, me es grato comunicarle que en la sesión del H. Consejo Técnico Consultivo celebrada el día 14 de diciembre del presente, fue aprobado el Temario propuesto:

TEMARIO:

"Generación automática de código a partir del diseño de una imagen de un sitio web utilizando aprendizaje automático"

1. Introducción
2. Creación del conjunto de datos: Adquisición de imágenes, preprocesamiento y etiquetado
3. Detección de elementos web mediante redes neuronales convolucionales: Reconocimiento de componentes en el diseño web
4. Modelo propuesto para la generación automatizada de código web
5. Diseño e implementación de un sistema de evaluación de similitud de imágenes
6. Cuantificación del éxito: Resultados y métricas de rendimiento
7. Conclusiones

Referencias

Apéndices

"MODOS ET CUNCTARUM RERUM MENSURAS AUDEBO"

A T E N T A M E N T E

DR. EMILIO JORGE GONZÁLEZ GALVÁN
DIRECTOR.

UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSI
FACULTAD DE INGENIERIA
DIRECCION

Copia. Archivo.
*etn.

www.uaslp.mx

Av. Manuel Nava 8
Zona Universitaria • CP 78290
San Luis Potosí, S.L.P.
tel. (444) 826 2330 al 39
fax (444) 826 2336

Automatic web code generation from a design of a website image using machine learning.

Abstract

This thesis addresses the challenge of automated code generation (ACG) in Artificial Intelligence (AI), specifically converting website images into HTML5 and CSS3 code using Machine Learning (ML). The main goal is to enhance web development workflows, minimizing manual coding by training Convolutional Neural Networks (CNNs) to detect and classify web elements from design images, and then generating corresponding HTML5 and CSS3 code. The hypothesis suggests that (i) CNNs combined with ML techniques can effectively generate web code, and (ii) an evaluation system using image similarity metrics can quantitatively assess the accuracy of the generated code. The methodology involves training a CNN for object detection, utilizing these classifications to produce web code. The evaluation system compares the auto-generated web pages with the original designs to validate accuracy. The research shows significant progress, with the system producing web code that closely mirrors input designs, achieving high accuracy and detail. Challenges such as detecting complex web elements and ensuring code robustness suggest areas for future research. The thesis highlights the feasibility and advantages of AI-driven automated web development, paving the way for future innovations in this field.

Generación automática de código a partir del diseño de una imagen de un sitio web utilizando aprendizaje automático.

Resumen

Esta tesis aborda el desafío de la generación automática de código (ACG) en el ámbito de la Inteligencia Artificial (IA), específicamente la conversión de imágenes de sitios web en código HTML5 y CSS3 utilizando Machine Learning (ML). El objetivo principal es mejorar los flujos de trabajo de desarrollo web, minimizando la codificación manual mediante el entrenamiento de Redes Neuronales Convolucionales (CNN) para detectar y clasificar elementos web a partir de imágenes de diseño, generando así el código HTML5 y CSS3 correspondiente. La hipótesis plantea que (i) las CNN combinadas con técnicas de ML pueden generar código web de manera efectiva, y (ii) un sistema de evaluación que utilice métricas de similitud de imagen puede cuantificar la precisión del código generado. La metodología implica entrenar una CNN para la detección de objetos y usar estas clasificaciones para producir el código web. El sistema de evaluación compara las páginas web autogeneradas con los diseños originales para validar su precisión. La investigación muestra avances significativos, con el sistema generando código web que refleja fielmente los diseños de entrada, logrando alta precisión y detalle. Los desafíos, como la detección de elementos web complejos y la robustez del código, sugieren áreas para futuras investigaciones. La tesis destaca la viabilidad y las ventajas del desarrollo web automatizado por IA, abriendo camino a futuras innovaciones en este campo.

Contenido

Contents

Listing of figures

List of Tables

Acknowledgments

I would like to express my deepest gratitude to my family and my wife for their support and encouragement throughout the course of my studies.

I am thankful to my thesis advisor, Dr. Juan Carlos Cuevas Tello, for his invaluable guidance, and expertise. His insights and feedback have been crucial in shaping this research. I also extend my appreciation to the entire thesis committee for their constructive comments and suggestions that significantly enriched my work.

Special thanks go to Octavio Israel Rentería Vidales, whose assistance in the system evaluation process was indispensable. His expertise and dedication played a pivotal role in the successful completion of this project.

1 Introduction

1.1 Preamble

This thesis addresses the challenge of automated code generation (ACG) within the domain of Artificial Intelligence (AI), specifically focusing on converting website images into HTML5 and CSS3 code. The primary objective is to develop a service capable of automating this conversion process using Machine Learning (ML). The motivation behind this research stems from the growing need to streamline web development workflows, reducing the time and effort required to manually code website layouts from design mockups.

The manual translation of design to code is both time-consuming and prone to errors. This inefficiency underscores the necessity for an automated solution that can accurately and efficiently generate web code from visual designs. To address this, the thesis hypothesizes that: (i) HTML5 and CSS3 code can be generated through the application of Convolutional Neural Networks (CNNs) for object detection, combined with ML techniques in computer vision and inference algorithms; and (ii) an evaluation system can be developed to measure the similarity between the generated code and the original design, providing a quantitative assessment metric. The methodology involves training a CNN to identify and classify web elements from images of website designs. This is followed by using these classifications to generate corresponding HTML5 and CSS3 code.

The second part of the solution introduces an evaluation system that employs image similarity metrics to compare the auto-generated web pages with their original designs, thereby validating the accuracy and fidelity of the generated code. The results of this research demonstrate significant advancements in the field of ACG. The proposed system successfully generates web code that closely mirrors the input designs, achieving high levels of accuracy and detail. The evaluation system further validates the effectiveness of the generated code, offering a reliable metric for assessing the performance of the ACG process. These outcomes highlight the potential for integrating such automated systems into standard web development practices, ultimately enhancing efficiency and reducing manual coding efforts.

During the research, several challenges were encountered, such as accurately detecting complex web elements and ensuring the generated code's robustness. These challenges indicate areas for future research to improve the system's performance and capabilities. Addressing these issues will further enhance the efficiency and reliability of automated web code generation. In conclusion, this thesis contributes to the field of AI-driven web development by providing a approach to automating the conversion of design images to web code. The research findings underscore the feasibility and benefits of such automation, setting the stage for future developments and applications in this area.

1.1.1 Hypothesis

In the rapidly evolving field of web development, the demand for efficient and accurate methods to convert design mockups into functional web pages is increasing. Traditional manual coding is not only time-consuming but also prone to errors, necessitating the need for automated solutions. This thesis explores the potential of Automated Code Generation (ACG) through the application of Artificial Intelligence (AI) and Machine Learning (ML), specifically focusing on generating HTML5 and CSS3 code from website design images.

The core objective of this research is to validate the hypothesis that visual web designs can be accurately converted into functional code using advanced ML techniques. This hypothesis is explored through the application of Convolutional Neural Networks (CNNs) for object detection, along with computer vision methodologies and inference algorithms. Additionally, the thesis introduces an evaluation system employing autoencoders to quantitatively assess the similarity between the generated code and the original design, thereby validating the accuracy and effectiveness of the proposed approach.

This research is based on the following hypotheses:

1. HTML5 and CSS3 code can be generated automatically from website design images using Convolutional Neural Networks (CNNs) for object detection combined with ML techniques in computer vision and inference algorithms.

2. An evaluation system using autoencoders can be developed to measure the similarity between the generated code and the original design, providing a quantitative assessment metric.

1.1.2 System Input and Results

Input

The system takes design images as input. These images are preprocessed and analyzed using Convolutional Neural Networks (CNNs) to detect and classify web elements. As described in Section 4.1.1, the input consists of the main webpage design image (Figure: 1.1) and the images included in that design (Figure: 1.2).

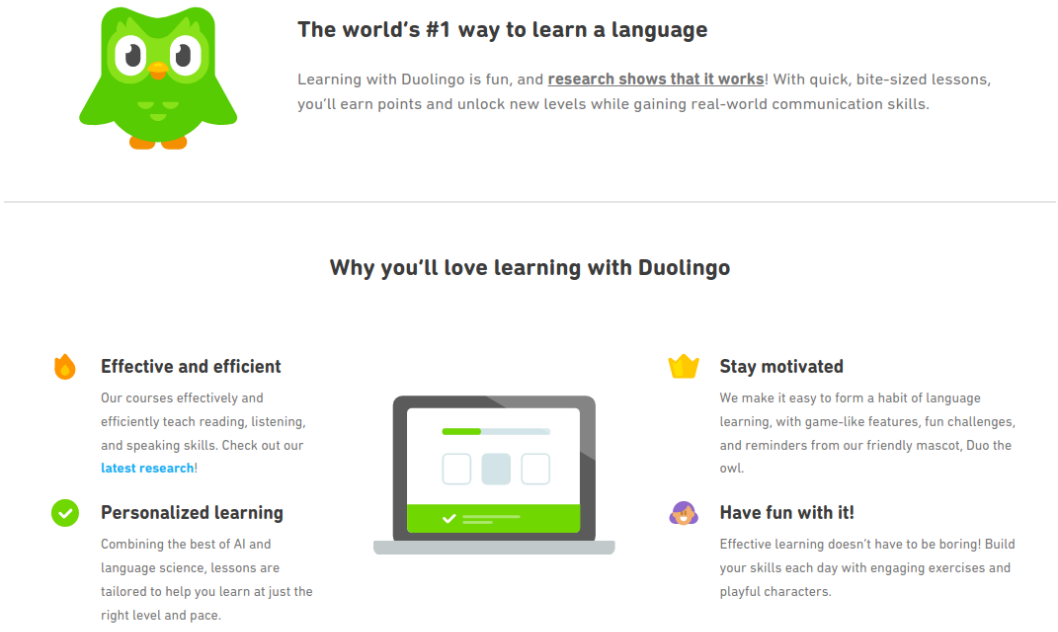


Figure 1.1: Webpage design image. Duolingo website image design as input to the system.

Figure 1.2: Collection of all images included in the main webpage design. This figure illustrates each image element that is part of the overall webpage layout, showcasing the individual components that contribute to the design.

The following diagram (Figure: 1.3) illustrates the overall process of automatic web code generation from a design image:

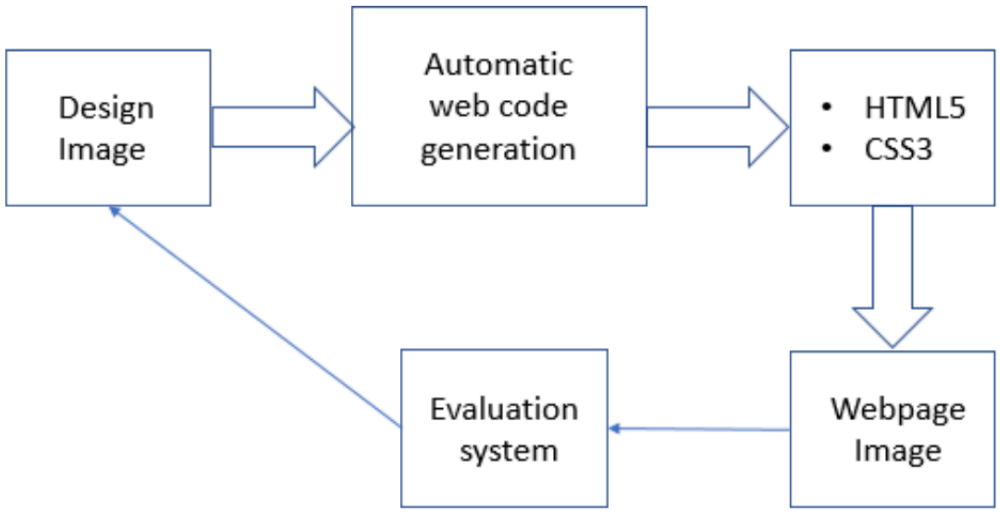


Figure 1.3: Automatic Web Code Generation Process. This diagram illustrates the process of automatic web code generation from a design image. The flow begins with the design image, which is analyzed using Convolutional Neural Networks (CNNs) to detect and classify web elements. These elements are then converted into HTML5 and CSS3 code. The generated code is rendered into a webpage image and evaluated by a system that compares the webpage image with the original design to validate accuracy and fidelity.

Explanation of the Diagram

1. **Design Image:** This represents the input to the system, which is an image of the website design.

2. **Automatic Web Code Generation:** Using the design image, the system employs machine learning techniques, specifically CNNs, to identify and classify various web elements. These elements are then converted into HTML5 and CSS3 code.

3. **HTML5 and CSS3 Code:** The output from the automatic web code generation process is the HTML5 and CSS3 code that represents the design and layout of the webpage.

4. **Webpage Image:** This is the visual representation of the generated HTML5 and CSS3 code, showing how the webpage will look when rendered in a browser.

5. **Evaluation System:** This subsystem evaluates the generated code by comparing the webpage image with the original design image. It uses image similarity metrics to assess the accuracy and fidelity of the generated code.

Results

The outcome of the code generation process is the creation of HTML5 (HTML code: 4.1.2) and CSS3 code (CSS code: 4.1.2). This code represents the system's output, encapsulating the visual elements and layout defined by the original design image. This generated code is then rendered into an image (Figure: 1.4), allowing the visual representation of the design to be viewed and evaluated. This rendering provides a clear depiction of how the generated code translates the initial design into a functional and styled web page.

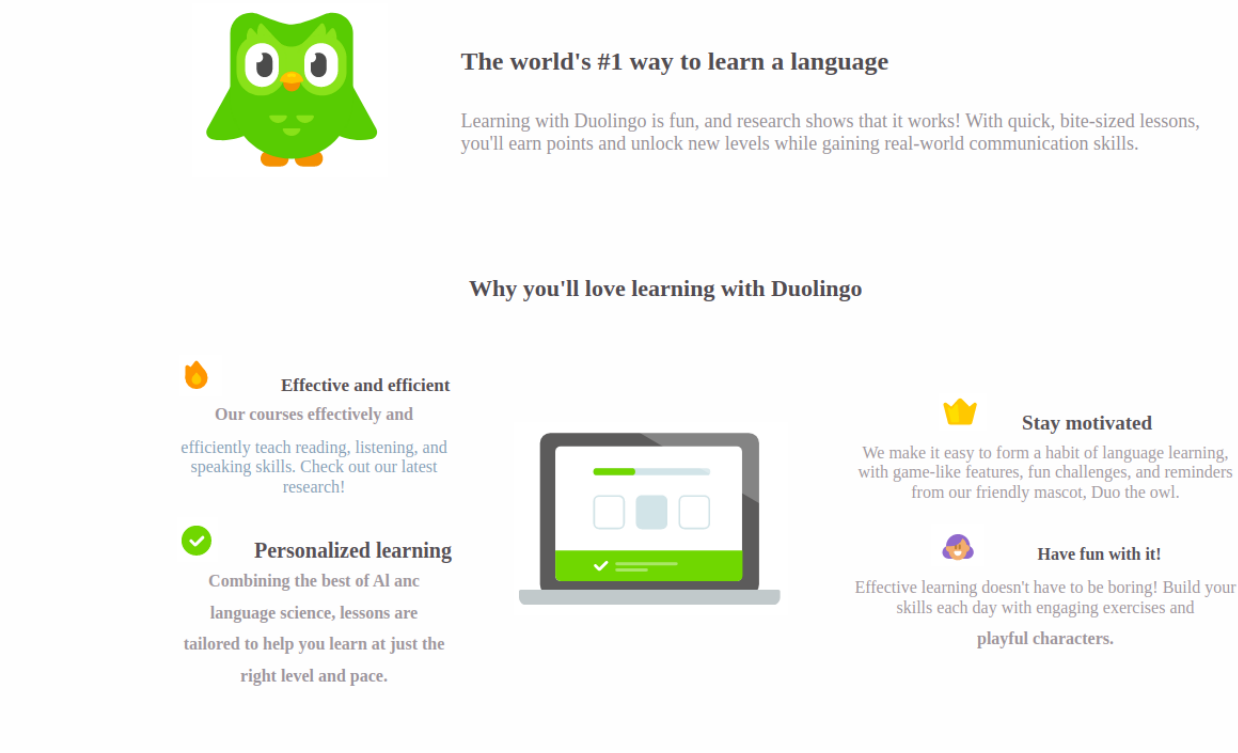


Figure 1.4: Webpage generated by the system. Result of the HTML5 and CSS3 code automated generated and rendered into an image.

1.2 Introduction

In an era where the lines between technology and daily life continue to blur, the demand for software solutions is growing at an unprecedented rate [2]. This increasing demand has pressured software development methodologies to evolve, driving the pursuit of efficiency, agility, and accuracy [3]. One significant advancement in this field is the concept of Automated Code Generation (ACG) [4]. ACG refers to the use of tools and methodologies that allow for expedited software application development through automation. This innovation has the potential to revolutionize the software development landscape by significantly reducing manual coding requirements, thus leading to a decrease in development time, cost, and human error [4].

The roots of ACG can be traced back to the concept of 'code reuse', which aimed to improve efficiency by reducing redundant programming efforts [5]. However, ACG takes this concept to a new level by automating the code production process itself, largely bypassing the need for human intervention [4]. This technology is built on sophisticated algorithms that interpret design models and transform them into functional code [6].

In recent years, Artificial Intelligence (AI) applications have appeared to assist programmers by suggesting code, intending to free up developers to spend less time coding repetitive patterns and more time focusing on what matters [7].

Research in this area has increased since GPT-3 demonstrated the ability to produce automatic code from a text paragraph [8]. GPT-3, an autoregressive language model with 175 billion parameters, achieves robust performance on many NLP datasets, including translation, question-answering, and cloze tasks. Furthermore, GPT-3 can generate samples of news articles which human evaluators have difficulty distinguishing from articles written by humans [8]. OpenAI recently released Codex, a successor of GPT-3, but with significant improvements that allow it to generate lines of code from a text description, thereby beginning a new era of AI services designed to assist developers in software development [9]. The primary distinction between GPT-3 and OpenAI Codex is that while GPT-3's expertise lies in generating natural language in response to natural language prompts, OpenAI Codex not only retains much of GPT-3's natural language understanding but also generates working code, enabling computers to better understand human intent [9].

Despite the progress made by these research efforts, it was not until GitHub released GitHub Copilot that programmers gained widespread access to these innovative tools and technologies, democratizing the use of AI tools for code automation and thus aiding developers in optimizing their time [10]. GitHub Copilot uses OpenAI Codex to suggest code and entire functions in real-time.

The AI service is most proficient in Python, but it also excels in over a dozen languages, including JavaScript, Go, Perl, PHP, Ruby, Swift, and TypeScript [10].

In addition to these developments, there have been other research initiatives for generative programming tools for various programming languages and areas, such as Web Development, where the goal is to generate front-end code using diverse AI techniques [11]. Automatic Web Code Generation (AWCG) refers to the specific application of these principles to the field of web development. Essentially, it involves using software tools and algorithms to automatically generate the code needed for creating web-based applications [11]. AWCG has the potential to dramatically accelerate the web development process, quickly generating boilerplate code, setup files, and structures common to many web applications. This allows developers to concentrate on unique features and business logic, standardizing the code to facilitate reading, maintaining, and debugging, and consequently reducing the likelihood of errors [11].

However, like any technology, AWCG comes with its own set of challenges. Ensuring the quality and security of automatically generated code, dealing with complex or unique requirements that may not be easily automated [12]. Despite these challenges, AWCG represents a significant leap forward in the field of web development, offering the potential to enhance productivity, improve quality, and reduce development time, leading to more efficient and effective web application development [12].

This thesis focuses on automatic web code generation and the design of a new service intended to enhance the efficiency of frontend developers. Recognizing that the styling of a website can be both complex and time-consuming, it is evident that all websites, irrespective of their size, depend on HTML and CSS. As a result, the primary objective of this thesis is to introduce a service that markedly reduces the time required by front-end developers, thus elevating productivity within the field of website development. Additionally, an essential element of this study is the introduction of a unique evaluation system that has been built.

By developing an evaluation technique to measure the similarity between images, this system offers a metric to evaluate the generated code in relation to the original design. This assessment determines the degree of resemblance between the code's outcome and the original source, signifying a notable breakthrough in the domain.

1.3 The Significance of the Study

The pursuit of automated code generation (ACG) represents a transformative frontier within the field of Artificial Intelligence (AI) and Machine Learning (ML). This thesis, with the focus on generating HTML5 and CSS3 code automatically from website images, stands at the intersection of several vital domains, including computer vision, object detection, algorithmic inference, and evaluation metrics. The importance of this study can be explained by these aspects:

- Innovation in Web Development: By utilizing Convolutional Neural Networks (CNN) and other ML techniques, the thesis introduces an innovative method for automatic HTML5 and CSS3 code generation. This has the potential to revolutionize web development, enabling rapid prototyping and reducing the time and effort required in hand-coding.
- Advancement in Computer Vision and Object Detection: The thesis methodology contributes to the growing body of research in computer vision and object detection. By applying these techniques to the domain of web development, it opens new avenues for exploration and practical application.
- Enhancing Evaluation Techniques: Developing an evaluation system to determine the similarity between images is a significant contribution to the field. By devising metrics to quantitatively measure the accuracy of the generated code, the study contributes to the broader landscape of evaluation methods within AI and ML.

1.4 Scope, Limitations and Delimitations

The automation of HTML5 and CSS3 code generation is a cutting-edge area of research and practice, promising significant advancements in efficiency and accuracy in web development. The state of the art in automated HTML5 and CSS3 code generation leverages advanced algorithms, machine learning, and artificial intelligence techniques to create high-quality, efficient, and maintainable web code.

These tools and techniques aim to eliminate the repetitive tasks often associated with writing HTML5 and CSS3 code, allowing developers to focus more on designing innovative and functional web applications.

This thesis will not cover services as GPT3/GPT4, because it is out of the scope of the system that it is desired to build from scratch. Instead, the thesis will investigate other research that employs Machine Learning (ML) algorithms for automatic code generation. This document will detail various ML algorithms and discuss the contemporary advancements in code generation.

Furthermore, it will explain the system developed from the ground up during this research for front-end code generation and introduce the evaluation system, designed to facilitate comparisons with alternative methodologies. The limitations of this study primarily stem from the inherent challenges in automating HTML5 and CSS3 code generation:

- Complexity of Web Design: Web design can often involve complex and unique elements that are difficult to standardize or automate. The inability of automated systems to handle this level of complexity and uniqueness can limit their applicability.
- Quality of Generated Code: Automatically generated code might not always adhere to best practices for coding and can sometimes be more verbose or less efficient than hand-written code.
- Security Vulnerabilities: Automation tools may unintentionally introduce security vulnerabilities into the generated code, particularly if these tools are not continually updated and improved.

Delimitations of this study are factors that restrict the scope and define the boundaries of the research. These delimitations have been chosen to ensure the research remains feasible and focused:

- Focus on HTML5 and CSS3: The study will only consider the automation of HTML5 and CSS3 code generation. Other web technologies, such as JavaScript or backend languages, will not be included.
- Exclusion of Certain Use Cases: Certain specialized or uncommon use cases of HTML5 and CSS3 may be excluded from the study, in order to focus on the most common and broadly applicable scenarios.

1.5 Research Hypothesis

As mentioned in preamble (Subsection: 1.1.1) the thesis is based on two main hypotheses, i) Generate HTML5 and CSS3 code using a Convolutional Neuronal Network (CNN) for object detection along with ML techniques with computer vision, and algorithms for inference in order to build the website; ii) Determine the feasibility to develop an evaluation system to determine the similarity between images to obtain an evaluation metric.

For the first activity, the primary hypothesis centers on determining the feasibility of identifying web elements in a webpage design image using Convolutional Neural Networks (CNN). If feasible, the goal becomes to create a graph data structure, extract features and styling information from the webpage

image using machine learning and computer vision algorithms, all with the objective of generating HTML5 and CSS3 code.

The secondary activity, potentially the most important contribution of this thesis, focuses on developing an evaluation system. Every research study and paper often introduces a distinct metric, such as the accuracy of an object detection model or tailored metrics to evaluate the precision of neural network models. Although conventional metrics like the F1 score can measure the accuracy of a CNN, contrasting the outcomes of various code generations presents intricacies. Due to this, the evaluation system centers on the final code produced. This system translates the code into an image and juxtaposes it with its original design. Subsequently, a similarity metric is introduced using autoencoders to ascertain the resemblance between the resultant image (the rendered HTML5 and CSS3 code) and the original webpage image design.

This system facilitates comparisons across all research and developed works, with the goal of pinpointing those with the top similarity scores. However, it is essential to note that this evaluation system only measures the resemblance of the results. Evaluating the quality of the generated code falls outside the system's scope.

1.6 State of art (Summary)

For the state of the art of the thesis, given the extensive points that need to be addressed, such as datasets, training models, CNNs, inference algorithms, computer vision techniques, evaluation systems, graph theory, etc. Each chapter will have its own state of the art section to detail in a better way the concept that is intended to be addressed. At present, this provides a concise overview of the research related to automated code generation.

Automated Code Generation (ACG) has been an area of interest in Artificial Intelligence (AI) for several decades, with applications ranging from generating software programs to web page design. Recent advancements in Machine Learning (ML) techniques and Convolutional Neural Networks (CNNs) have enabled the development of more robust and accurate ACG systems.

A noteworthy study in the field is "pix2code: Generating Code from a Graphical User Interface Screenshot" by Beltramelli [13]. This work proposed a neural network architecture that can translate GUI screenshot images into code. It employs a deep learning model that combines a CNN for visual feature extraction and a Recurrent Neural Network (RNN) for generating relevant code. While this study focused on code generation for mobile applications, the concepts and techniques presented could potentially have applications in generating HTML5 and CSS3 from website images.

DeepCoder is another relevant example in the field of code generation using ML. In the paper "DeepCoder: Learning to Write Programs" [14] describe a system that employs ML techniques to generate code programs from high-level descriptions. DeepCoder utilizes a technique called "program synthesis" where, given input/output examples, a code that performs the desired function is generated. While its focus is more on generating software programs, the principles of "program synthesis" can be useful in understanding how ACG systems can "learn" to write code more effectively.

In the field of image similarity analysis and evaluation, the paper "A Comprehensive Survey on Cross-modal Retrieval" [15], presents an exhaustive review of existing methods for multimodal information retrieval. This might be useful in developing an evaluation system for determining the similarity between website images and the generated HTML5 and CSS3 code.

Research in ACG has markedly progressed in recent years, showcasing more advanced and efficacious techniques for automatic code generation. However, there are still avenues to explore, particularly in the area of generating HTML5 and CSS3 code from website images, the central theme of this thesis.

1.7 Justification of the Technical Approaches

The selection of technical approaches in this thesis is rooted in both the current state of the art in automated code generation and the unique requirements of the proposed project.

1. **Machine Learning (ML) Algorithms:** Machine learning has increasingly become an indispensable tool in various applications of artificial intelligence. For this thesis, ML is employed due to its demonstrated effectiveness in pattern recognition, prediction, and decision-making processes, key aspects in code generation.

2. **Convolutional Neural Networks (CNNs):** CNNs are chosen as a critical component of the approach due to their superior performance in image recognition tasks. CNNs are especially effective in identifying and classifying visual features, which are crucial in translating website images into HTML5 and CSS3 code.

3. **Graph Theory:** Graph theory provides a powerful solution for representing the structural elements of a webpage. Nodes can represent HTML elements, and edges can denote the relationships between these elements. This approach allows for a more systematic and logical translation from an image to code.

4. **Inference Algorithms:** Inference algorithms play a significant role in the project due to their ability to produce logical consequences based on a given set of facts or premises. In the context

of automated code generation, these algorithms can generate the structural elements of the code based on the identified visual features of a website image.

5. **Computer Vision Techniques:** These techniques are instrumental in this project as they provide the ability to extract meaningful features from images.

6. **Evaluation Systems using Autoencoders:** An evaluation system is necessary to measure the performance of the generated code and ensure it accurately represents the website image. By assessing the similarity between the generated HTML5 and CSS3 code and the original website image, the code generation process can be continuously refined and improved.

1.7.1 Proposed Technical Concepts

- **RFCN-ResNet101 model implementation.** Region-based Fully Convolutional Networks (R-FCN) operate on the principle of applying a fully convolutional network over the entire image to compute the convolutional features and then employ position-sensitive score maps to determine the class and bounding box for each object. ResNet101, a deep residual network with 101 layers, is employed as the backbone for R-FCN, providing deep feature extraction capabilities and computational efficiency.

- **Computer Vision Techniques.**

  - **Template Matching:** Utilized to recognize and catalog images embedded within the website's design. This involves precise image manipulation, such as resizing operations to match the scale of the images as they appear in the design.

- **Machine Learning Techniques.**

  - **Optical Character Recognition (OCR):** Services provided by AWS and GCP are used to extract textual content from the design. This phase involves advanced pre-processing to filter out text that is not pertinent to the web code generation process, such as text within images or symbolic representations.
  - **K-Means Clustering:** Applied to classify titles into appropriate heading tags (h1-h6) based on their size and other characteristics. This clustering approach facilitates an organized and systematic assignment of heading tags, aligning with the hierarchical structure of the web content.

- **Inference Algorithms.** Inference algorithms are used to interpret the detected elements and generate the corresponding HTML5 and CSS3 code. These algorithms are crucial for ensuring that the generated code accurately reflects the structure and styling of the original design.

- **Evaluation System: Autoencoders.** An evaluation system using autoencoders is developed to assess the quality of the generated code. By comparing the rendered web pages from the generated code with the original design images, the system provides a similarity metric that quantifies the accuracy and fidelity of the generated code.

1.8 Methodology

This section outlines the methodology employed in this research to develop an automated system for generating HTML5 and CSS3 code from website design images. The methodology encompasses the creation of the dataset, the training of the Convolutional Neural Network (CNN) model, the generation of code, and the evaluation of the system's performance.

1.8.1 Dataset Creation: Image Acquisition, Preprocessing, and Labeling

The dataset creation process is important for training and validating the machine learning model. This process involves several steps:

Image Acquisition

A total of 10,000 images were collected from various websites to ensure a diverse representation of web design elements. These images serve as the primary data source for training the CNN model.

Preprocessing

Preprocessing steps were undertaken to prepare the images for annotation and model training:

- **Resizing**: All images were resized to a uniform dimension to standardize the input for the CNN model.

LabelImg

Images were manually labeled using the LabelImg tool. The following web elements were annotated: headers, footers, buttons, navigation bars, sections, divs, and other structural components. Annotations were saved in the PASCAL VOC format, providing XML files that contain the bounding box coordinates and labels for each element in the images.

Segmented Dataset for Section and Div Detection

To improve the detection of structurally complex web elements, a segmented filter was applied to the original images. This process involved:

- **Segmentation**: Images were segmented to highlight sections and divs, reducing noise from non-structural elements.
- **Enhanced Annotation**: The segmented images were then re-annotated to focus on these structural elements, ensuring a more accurate detection model.

1.8.2 Web Element Detection via Convolutional Neural Networks (CNN)

The detection of web elements was achieved using a Convolutional Neural Network (CNN) model. The steps involved in this process are as follows:

Model Selection: RFCN-ResNet101

The Region-based Fully Convolutional Network (RFCN) with a ResNet101 backbone was chosen for its balance of accuracy and computational efficiency. This model is particularly effective in extracting deep features and accurately detecting objects in images.

Training the Model

The annotated dataset was split into training and validation sets. The training process involved:

- **Hyperparameter Tuning**: Optimal hyperparameters, including learning rate, batch size, and number of epochs, were determined through grid search and cross-validation.
- **Framework**: TensorFlow was used as the primary framework for implementing and training the CNN model.

Evaluation of the Model

The performance of the trained model was evaluated using standard object detection metrics:

- **Mean Average Precision (mAP)**: This metric was used to measure the accuracy of the model in detecting and classifying web elements.
- **Intersection over Union (IoU)**: IoU was calculated to assess the overlap between the predicted bounding boxes and the ground truth annotations.

1.8.3 Automated Web Code Generation

The generation of HTML5 and CSS3 code from the detected web elements involved several steps:

Inference Algorithms

Inference algorithms were developed to convert the detected web elements into structured HTML5 and CSS3 code. These algorithms interpret the detected elements and generate the corresponding code snippets.

Subsystems for Code Generation

The system was divided into several subsystems, each responsible for a specific aspect of code generation:

- **Element Detection Subsystem**: Identifies and classifies web elements in the design image.
- **Code Generation Subsystem**: Generates the HTML structure and CSS styles based on the detected elements.
- **Correction and Enhancement Subsystem**: Refines the generated code to ensure adherence to web development best practices.

1.8.4 Evaluation System

An evaluation system was designed to measure the similarity between the generated code and the original design:

Autoencoders for Image Similarity

Autoencoders were employed to compare the rendered web pages from the generated code with the original design images. The autoencoders encode the images into a lower-dimensional space and then reconstruct them to measure the reconstruction error, which serves as a similarity metric.

1.8.5 Validation of the Model

The final validation of the system involved testing with new, unseen images of web designs:

- **Generalization Testing**: The system was evaluated on a separate test set to ensure it can generalize well to new web designs.
- **Performance Analysis**: The accuracy, efficiency, and robustness of the generated code were analyzed and documented.

If data is carefully prepared, a company may need far less of it than they think. With the right data, companies with just a few dozen examples or few hundred examples can have A.I. systems that work as well as those built by consumer internet giants that have billions of examples.

Andrew NG.

2

Dataset creation: Image Acquisition, Preprocessing and Labeling

2.1 Introduction

A dataset is an organized and structured collection of data, typically assembled for analysis, inference, or training predictive models in various fields such as machine learning, statistics, data mining, and more. Datasets serve as the fundamental building blocks upon which all data-driven analysis is built, comprising individual units of information that, when viewed as a whole, can provide a broad, comprehensive understanding of the subject under study. In a world increasingly driven by data, datasets have become invaluable resources. In the context of machine learning and artificial intelligence, datasets are used to train and validate models, playing a vital role in determining their subsequent performance and accuracy.

However, a dataset's effectiveness is not just about quantity but rather depends significantly on its quality. High-quality datasets are rich in diversity and free from biases and errors, producing robust, reliable, and generalizable machine learning models.

Poorly constructed or low-quality datasets, on the other hand, can lead to models that underperform, are biased, or are incapable of accurately predicting or analyzing real-world scenarios. In this Chapter it is explained the developed work for searching, generating, and labeling images to have a dataset that could produces an efficient model with the enough accuracy to detect and identify web elements, with the aim that the system could creates a data tree structure using the recognized elements and extract the information to XML code generation.

2.2 State of the art

2.2.1 Definition, Types, and Sources of Datasets

A dataset is defined as a collection of structured or unstructured data points [16]. These collections can come in different formats such as tables, matrices, or even multidimensional arrays. Data can be categorized into different types, such as nominal, ordinal, interval, and ratio [17]. Furthermore, datasets can be characterized by their source, either primary (original data collected by the researcher) or secondary (data collected by someone other than the researcher) [18].

Datasets are also classified into structured and unstructured datasets. Structured datasets have a high degree of organization and a predefined data model, such as relational databases, whereas unstructured datasets lack a specific form or model, such as text, video, or social media data. [19]. Primary datasets are typically obtained from scientific experiments, surveys, or any other data collected firsthand [20]. Secondary datasets, meanwhile, are obtained from existing sources such as government databases, data aggregators, or even online repositories. Online platforms such as Kaggle, UCI Machine Learning Repository, Google's Dataset Search, and governmental portals provide a rich source of secondary datasets that cover a wide array of topics and sectors [21] [22].

Public datasets from these sources have become an invaluable resource for researchers, especially in the machine learning and AI communities. They have been essential in driving advances in fields such as image recognition [23], natural language processing [24], and autonomous vehicles [25].

2.2.2 Importance of Datasets in Research and Machine Learning

Datasets are indispensable in both traditional research settings and in contemporary fields such as machine learning and AI. In traditional research, datasets are used for hypothesis testing, correlation studies, and observational studies [26]. This allows for a wide range of analyses to be conducted, from simple descriptive statistics to complex inferential statistics.

In machine learning area, datasets serve as the foundation for training and validating models. They are used to teach machine learning algorithms how to recognize patterns, make predictions, or perform tasks without explicit programming [27]. By learning from a dataset, a machine learning model can generalize its learning to new, unseen data, allowing it to make accurate predictions or decisions in a variety of contexts [28].

Large and diverse datasets can help to improve the performance of machine learning models by exposing them to a wider range of scenarios, thereby enhancing their ability to generalize [29]. This has been demonstrated in numerous studies across different domains, such as computer vision [30], natural language processing [31] and healthcare.

2.2.3 The Quality of Datasets and Its Impact on Models

Accuracy refers to whether the data correctly represents the real-world phenomena it is supposed to capture, and if all necessary data points have been included. Consistency deals with the uniformity of the data across the dataset. Timeliness refers to how recent and therefore relevant the data is. Lastly, relevance focuses on whether the data addresses the specific questions [32].

A poor quality dataset can lead to several issues, including overfitting, underfitting, and bias. Overfitting occurs when a model learns to perform very well on the training data but fails to generalize to unseen data, often due to excessive noise or outliers in the dataset [33]. Underfitting, on the other hand, happens when the model is too simple to capture the underlying structure of the data. Bias refers where the model unfairly favors certain outcomes over others. This could be due to an unrepresentative sample or exclusion of certain groups in the dataset [34]. The recent advancements in machine learning emphasize the crucial role of high-quality data in achieving representation and equitable outcomes [35].

2.2.4 Data Preprocessing and Cleaning

Before using datasets to train machine learning models, they often require preprocessing and cleaning to improve their quality and make them suitable for analysis. Data preprocessing is a crucial step that involves a series of operations aimed at transforming raw data into an understandable format [36]. Preprocessing includes dealing with missing values, noise, and outliers, which can skew the performance of a model if not appropriately handled [37].

Missing values can be addressed using methods like listwise removal, mean replacement, or specific algorithms designed for incomplete data[38]. Noise in the data can be reduced through smoothing techniques, while outliers can be detected and handled using statistical methods [39]. Another vital step in data preprocessing is feature scaling or normalization, which ensures that all features have a similar scale and prevents certain features from dominating others [40].

Moreover, the process of feature selection and extraction is important for dimensionality reduction, removing irrelevant or redundant attributes, and improving the interpretability of the data [41]. This can significantly enhance the performance of machine learning models and make the models less complex and easier to understand.

2.3 Dataset creation

In this research, two distinct datasets were developed, both derived from the same initial pool of 10,000 images. These images were captured from a wide array of websites, ensuring a diverse representation of web design elements. The following subsections (2.3.1, 2.3.2) detail the composition and intended use of each dataset.

2.3.1 First Dataset: Web Element Detection

The first dataset consists of 10,000 unaltered website images (Example Figure: 2.1). This dataset's primary purpose is to train the Convolutional Neural Network (CNN) model to identify and classify basic web elements such as headers, buttons, and navigation bars. The diversity of this dataset is crucial for developing a robust model capable of recognizing various web elements across different web designs.

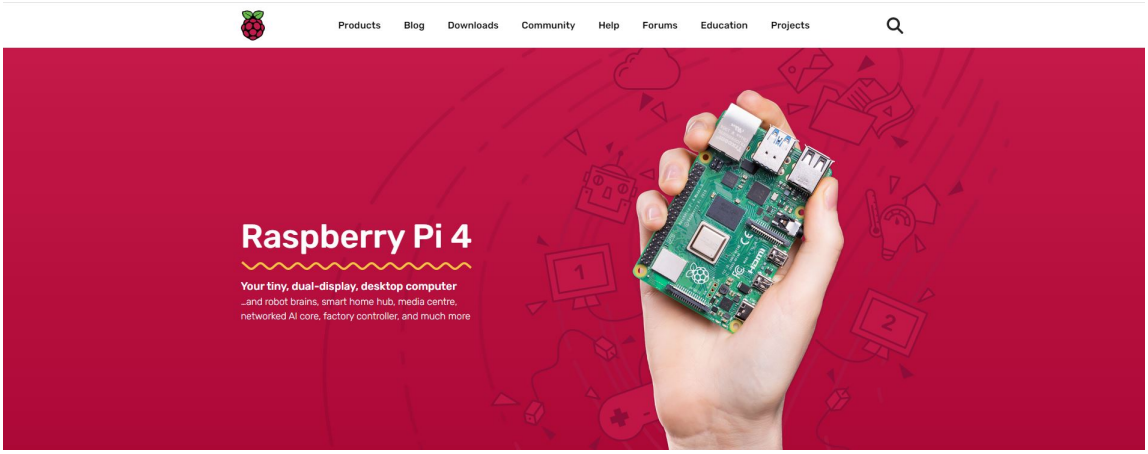


Figure 2.1: Example image for the first dataset. This dataset consists of unaltered website images.

2.3.2 Second Dataset: Segmented Dataset for Section and Div Detection

The second dataset is derived from the same 10,000 images used in the first dataset. To enhance the model's ability to detect more structurally complex web elements like sections and divs, a segmented filter was applied to each image (Example Figure: 2.2). This filtering process highlights structural divisions within the webpage by segmenting the image into distinct sections based on visual cues and boundaries. This segmentation aims to reduce the noise introduced by non-structural elements such as images and intricate backgrounds, which can obscure the more fundamental divisions of a webpage's layout.

The application of this segmented dataset is specifically improving the detection accuracy of div and section elements within a webpage. By training the CNN model with these segmented images, the system is better equipped to recognize and delineate these elements, crucial for generating structured and semantically meaningful HTML code.

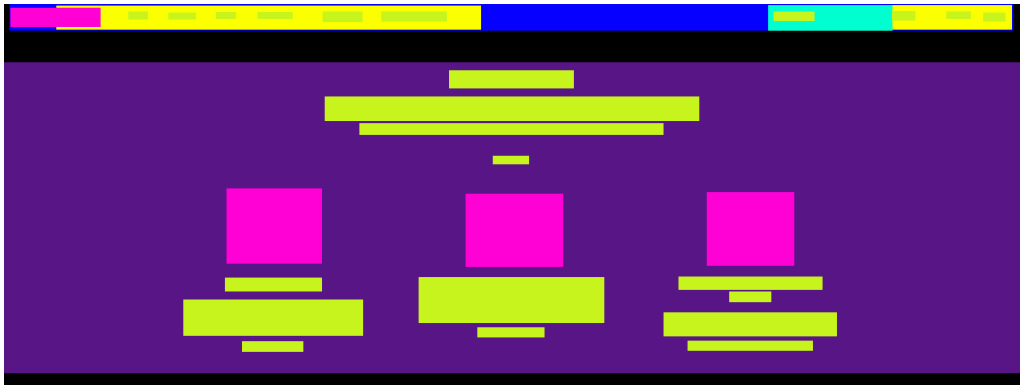


Figure 2.2: Example image for the second dataset. This dataset consists of images with segmented filter.

2.3.3 LabelImg

LabelImg [42], an open-source graphical image annotation tool, was employed for labeling. This tool is widely used in computer vision research for annotating images with object bounding boxes. It is favored for its user-friendly interface (Figure: 2.3) and compatibility with both Windows and Linux platforms. LabelImg facilitates manual annotation of images, allowing users to draw rectangles around objects of interest, which are then classified into predefined categories. This tool supports annotations in the PASCAL VOC format, essential for compatibility with many computer vision models and frameworks.

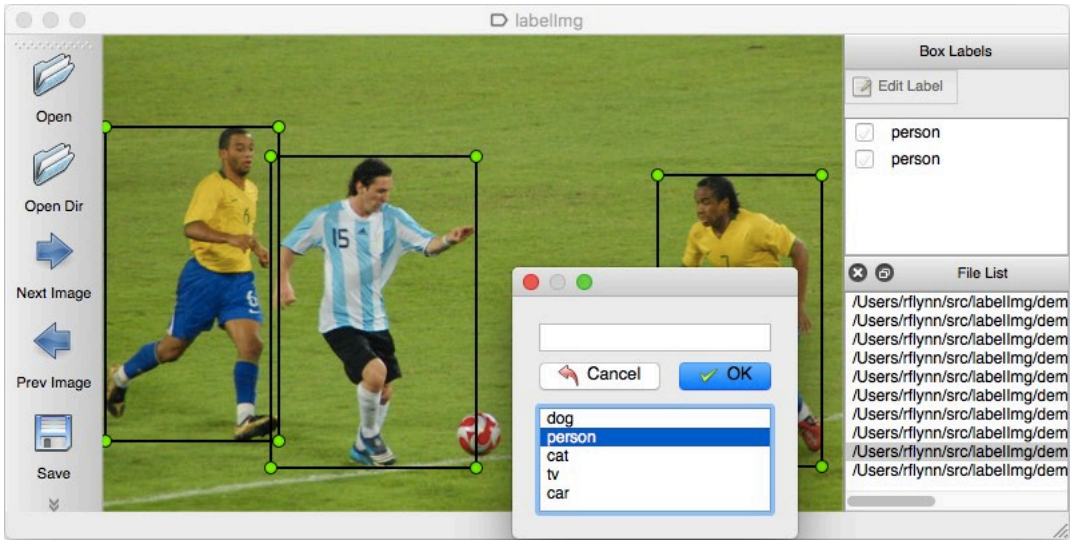


Figure 2.3: LabelImg software: Interface for labeling process.

The PASCAL VOC format uses an XML-based annotation schema (Figure: 2.4) to store the object annotation information. For each image, an XML file is created, containing the necessary metadata about the image and the bounding box annotations for all objects present in the image. Specifically, the metadata includes details like the image filename, and size. The bounding box annotations, meanwhile, describe the object class, the bounding box coordinates, and whether the object is truncated, difficult, or occluded.

```
<annotation>
  <folder>pkg_6_200</folder>
  <filename>ix2code_ds_1.png</filename>
  <path>ix2code_ds_1.png</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>1320</width>
    <height>743</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>section</name>
    <pose>Unspecified</pose>
    <truncated>1</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>2</xmin>
      <ymin>111</ymin>
      <xmax>1316</xmax>
      <ymax>743</ymax>
    </bndbox>
  </object>
```

Figure 2.4: PASCAL VOC format: Example of the XML code that was generated by the LabelImg software after an image was labeled.

One of the distinctive features of the PASCAL VOC format is its emphasis on detailed, accurate, and comprehensive annotations. The format encourages manual annotation, often resulting in high-quality, reliable datasets.

However, there are also a few limitations to consider. The format is quite verbose and not as straightforward as other popular formats, such as COCO or YOLO. Furthermore, the PASCAL VOC format does not directly support the annotation of rotated bounding boxes, making it less suitable for certain tasks such as aerial image analysis or document layout understanding. Despite these limitations, the PASCAL VOC format remains a popular choice due to its detailed nature and wide support in various computer vision libraries and tools.

2.3.4 Dataset Process Creation

Initially, before the establishment of the two distinct datasets described earlier, there was a single dataset composed with more than 10,000 website images. This primary dataset included sixteen of the most common web elements, which are essential for constructing webpages using HTML5. Elements for the first dataset version (Table: 2.1):

Table 2.1: Summary of Web Elements (Fist Dataset Version).

header	9035	footer	2948
section	17315	nav	10339
button	17232	rounded_button	3512
circle_button	6595	square_button	5363
arrow_icon	1835	horizontal_div	36617
vertical_div	27226	search_bar	4524
input_form	7794	image	3026
background_image	2359	form	2663

As an integral part of the research, It was encountered the critical need for data preprocessing. The dataset contained numerous images; however, some were mislabeled or poorly represented due to their small size. To overcome these challenges and ensure the reliability of the analysis, It was developed a series of Python scripts to correct and improve the dataset.

Resolving dataset issues

To address the issue of mislabeled images, a script was designed (Appendix-B: B.4). During the process of generating the TF records, this script checks to ensure that all labels are recognized. These labels are preloaded from a txt file containing all the required labels. If an anomaly is detected, the script displays the name of the image along with the unrecognized label, such as "heder" (Example Figure: 2.5). This output then facilitates manual identification and correction of the image labels, allowing for accurate label adjustment.

Also it was created a python script to handle images that were misrepresented due to their smaller dimensions (Appendix-B: B.5). These images were problematic because their limited resolution produced issues during the feature extraction process. The program identifies such images by comparing their dimensions against a predetermined threshold value (Example Figure: 2.5). Once identified, these cases were reviewed individually to decide whether to discard the instances. In the same line, a specialized script was developed to check each label item, guaranteeing it satisfies the minimum dimension criteria (Appendix-B: B.6).

Figure 2.5: Dataset validations scripts.

Lastly, with the cleaned and properly formatted data, It was possible to proceeded to conduct an extensive data analysis. Using pandas library was possible to provide descriptive statistics of the cleaned dataset, including the number of instances for each class, the distribution of image sizes, and other relevant information. Also using libraries like NumPy for numerical computations and Matplotlib for visualization provided valuable insights into the structure and characteristics of the data.

2.3.5 Balancing the Dataset for Optimal Machine Learning Performance

After the data is cleaned and organized, the focus shifts towards balancing the dataset, an important step for ensuring optimal machine learning performance. The primary objective is to detect labels associated with a sparse number of elements, which could hinder accurate detection due to their underrepresentation. The aim is to balance the dataset by ensuring a uniform distribution of elements. This balance is crucial because it prevents the model from being biased towards more frequent labels, enhances the model's ability to generalize across different types of web elements, and improves overall detection accuracy. Ensuring that each label has a similar number of examples aids in creating a robust dataset that contributes to more effective and equitable machine learning outcomes. Advantages for maintaining a balanced distribution:

- **Bias Reduction:** A balanced dataset helps prevent the model from developing a bias towards more frequently occurring elements. Bias in training data can lead the model to perform well on similar types of data but poorly on data types that are underrepresented.
- **Improved Generalization:** By ensuring that no single type of web element dominates the dataset, the model can learn to recognize a variety of elements, which enhances its ability to generalize from training data to real-world applications.
- **Enhanced Model Accuracy:** Consistent exposure to a balanced number of elements across different images helps the model learn important features without being overwhelmed by noise or anomalies, leading to more accurate detections and classifications.

A pandas function was employed to convert PASCAL Visual Object Classes (VOC) files into Comma Separated Values (CSV) format (Figure 2.6), with the primary intention of efficiently analyzing and deciphering the data encapsulated within the files.

```
def csvFromXML(name_xml):
    tree = xml.ElementTree(file=name_xml)
    root = tree.getroot()

    lines = []
    filename = root.find('filename').text

    row = ""
    for element in root.iter("object"):
        label = element.find('name').text
        box = element.find('bndbox')
        xmin = int(box.find('xmin').text)
        ymin = int(box.find('ymin').text)
        xmax = int(box.find('xmax').text)
        ymax = int(box.find('ymax').text)
        width = int(xmax) - int(xmin)
        height = int(ymax) - int(ymin)
        area = width * height
        row = f'{filename},{label},{xmin},{ymin},{xmax},{ymax},{width},{height},{area}\n'
        lines.append(row)
    return lines
```

Figure 2.6: Code to convert PASCAL VOC files into CSV file.

After converting the VOC files to CSV, performing data analysis and computation became possible. The transformation facilitated a thorough review of various aspects of the dataset, specifically the quantity of web elements contained within each image. To balance the elements, efforts were made to supplement the dataset with additional images that contain those elements which are least represented. This approach ensures a more uniform distribution of elements across the dataset, enhancing the robustness and accuracy of the analyses derived from this data.

After refining the dataset and obtaining preliminary results, which will be detailed in a subsequent section (Section: 3), challenges were encountered regarding the quality of the Convolutional Neural Network (CNN) model. The primary issue was the model's inability to accurately identify the bounding boxes for 'div' and 'section' web elements. It was concluded that certain elements, such as images, introduced noise, thus disrupting the detection of patterns for 'divs' and 'sections'.

To overcome these challenges, a new script was created (Appendix-B: B.7) to convert the first dataset (the website images) into a segmented dataset. In this enhanced dataset, each element was identified by a unique color code based on information from the original dataset. The aim was to make the patterns of element groupings more evident to the model, thereby improving its predictive performance for 'div' and 'section' elements (Figure: 2.7).

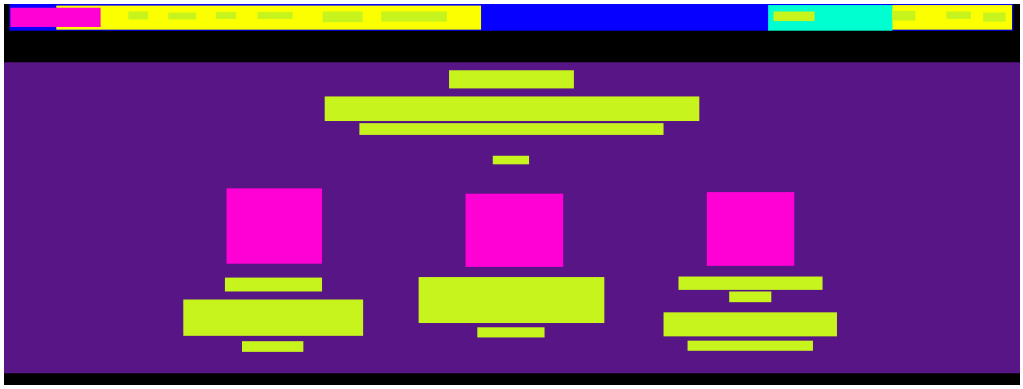


Figure 2.7: Segmented image example.

This process resulted in the two datasets mentioned earlier (Figure: 2.8). The first dataset, which contains the web elements and the second one designed specifically to facilitate more effective detection of 'section' and 'div' groups.

Figure 2.8: On the left, an image of the webpage design is displayed, representative of the first dataset, which includes the primary web elements. On the right, an illustration of the segmented dataset is shown, comprising labels for 'divs' and 'sections'. The use of this segmented dataset aims to enhance the accuracy of the detection algorithm.

The process of data balancing was pivotal in elevating the quality of the datasets and, in turn, boosting the precision and dependability of the model's results. By achieving a balanced distribution across all classes, potential biases in the model were diminished, yielding results that are both more generalizable and trustworthy.

By blending the techniques of oversampling minority classes and undersampling majority ones (Unbalanced Dataset Example Figure 2.9), a more balanced data distribution was achieved (Figures: 2.10, 2.11). During this process, caution was exercised to prevent the oversampling from adding undue noise and to ensure undersampling did not discard essential information.

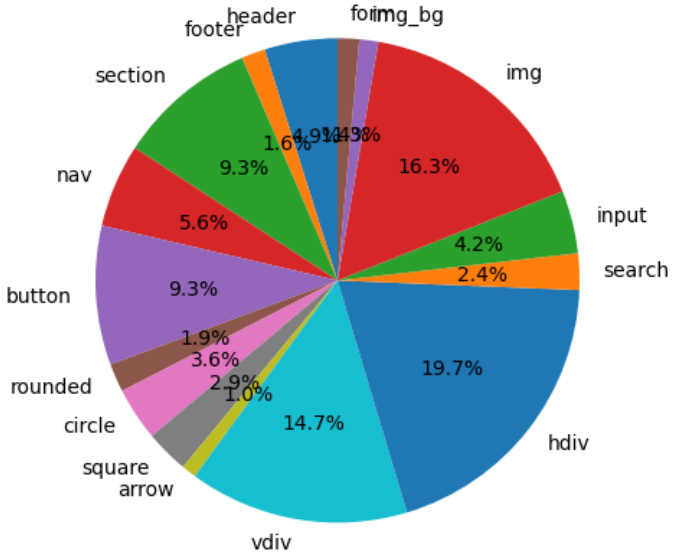


Figure 2.9: Dataset pie chart [first version]. An illustration of the data's imbalance.

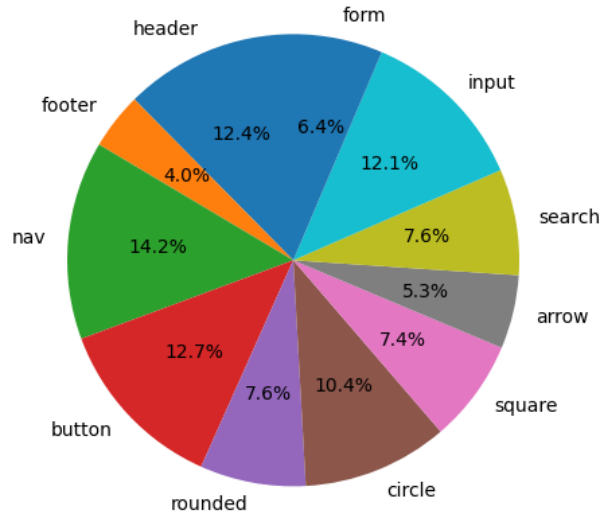


Figure 2.10: Final distribution for web elements dataset (pie chart).

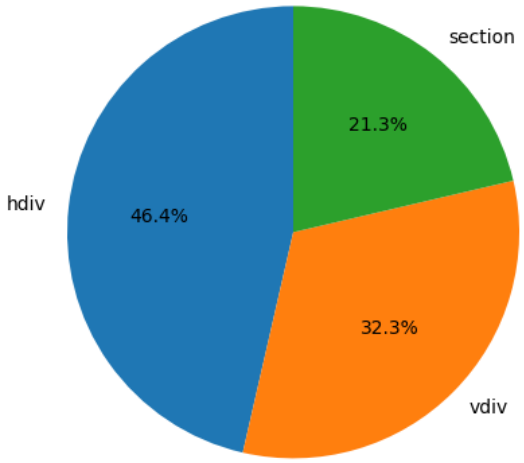


Figure 2.11: Final distribution for divs and sections in segmented dataset (pie chart).

The predominant elements still hold a considerable presence, but their overwhelming influence was considerably reduced. This facilitated a more robust analysis, as the machine learning models were now able to learn from a more diverse and representative set of data.

In summary, by rectifying the dataset imbalances, these measures not only fortified the accuracy and dependability of the results but also underscored the paramount importance of data preprocessing in the domains of machine learning and data analysis.

At the end the dataset for the primary web elements is detailed in Table 2.2, while the segmented dataset, which focuses on 'divs' and 'sections' elements, is outlined in Table 2.3.

Table 2.2: Number of each label in the Web Elements Dataset.

header	9035	footer	2948
nav	10339	button	9226
rounded_button	5512	circle_button	7595
square_button	5363	arrow_icon	3835
search_bar	5524	input_form	8794
form	4663		

Table 2.3: Number of each label in the Segmented Dataset.

hdiv	37663	vdiv	26171
section	17313		

2.3.6 Model Challenges and Improvements

Initial Model Performance Issues

The first version of the model, trained on the initial dataset, exhibited several significant issues that impacted its performance. The primary challenges included:

1. **Inaccurate Bounding Box Predictions:** The model struggled to accurately identify and delineate bounding boxes for certain web elements, particularly div and section elements. This inaccuracy was attributed to the presence of noise from other elements like images and intricate backgrounds, which obscured the structural features of the webpage layout.

2. **Overfitting and Underfitting:** The model showed signs of overfitting, where it performed exceptionally well on the training data but failed to generalize to new, unseen data. Conversely, underfitting occurred in certain cases, indicating that the model was too simplistic to capture the underlying patterns of the dataset.

3. **Imbalance in Dataset:** The initial dataset contained an uneven distribution of web elements, leading to a biased model that favored more frequently occurring elements while performing poorly on less common ones.

4. **High Variance in Image Quality:** The variance in image resolution and quality caused inconsistent feature extraction, resulting in unreliable predictions.

Solutions Implemented

To resolve these issues, several strategies were implemented, focusing on dataset improvement and model optimization:

1. **Noise Reduction through Segmentation:** To address the issue of noisy images, a segmentation filter was applied to the dataset, creating a second, segmented dataset. This filter highlighted structural divisions within the webpage by segmenting images into distinct sections based on visual cues and boundaries. The segmented dataset allowed the model to focus on the fundamental layout elements without being distracted by non-structural content.

2. **Data Preprocessing and Cleaning:** Scripts were developed to identify and correct mislabeled images and to handle images that were misrepresented due to their small size. By ensuring all images met a minimum dimension criteria and correcting any labeling errors, the quality and reliability of the dataset were significantly improved.

3. **Balancing the Dataset:** To mitigate bias, techniques such as oversampling minority classes and undersampling majority ones were employed, achieving a more balanced distribution of web elements. This ensured that each class was adequately represented, allowing the model to learn from a diverse and representative set of data.

4. **Validation and Error Checking:** Additional validation scripts were created to verify the integrity of the dataset. These scripts ensured that all elements adhered to the minimum dimension requirements and that there were no anomalies in the labels, further enhancing the dataset's quality.

Results and Evaluation

The application of these solutions resulted in significant improvements in the model's performance:

- **Improved Accuracy:** The accuracy of bounding box predictions for div and section elements improved considerably, as the model could now better recognize these structures within the segmented images.

- **Reduced Bias:** The balanced dataset led to more equitable performance across all web elements, reducing the bias towards frequently occurring elements.
- **Enhanced Generalization:** The use of advanced training techniques allowed the model to generalize better to new data, minimizing the risk of overfitting.

Figures 2.12 and 2.13 showcase examples of model predictions before and after the improvements. The comparison highlights the increased precision and accuracy in identifying web elements, demonstrating the effectiveness of the implemented solutions.

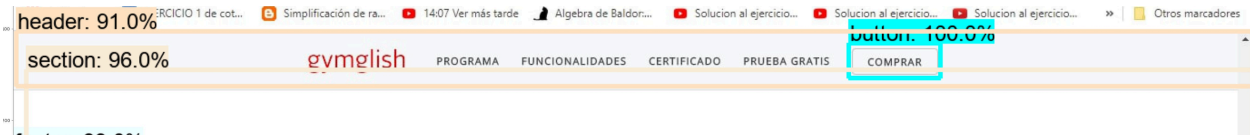


Figure 2.12: Model Performance Before Improvements: Issues with 'navbar' elements detection.

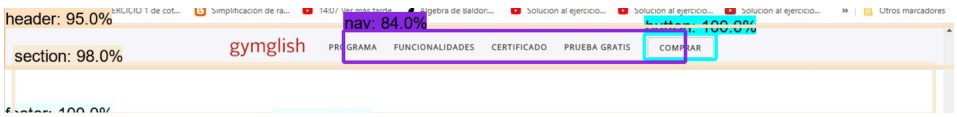


Figure 2.13: Model Performance After Improvements: Enhanced generalization and improved elements detection.

In conclusion, the combination of noise reduction, data preprocessing, balancing techniques, and advanced training methods played a crucial role in resolving the initial model challenges, leading to a more robust and accurate web element detection system.

We are teaching computers to see, so that they can understand the world at a much larger scale than we, as humans, have ever been able to do

Dr. Fei-Fei Li

3

Web Element Detection via Convolutional Neural Networks: Recognizing Components in Web Design

3.1 Introduction

In the area of deep learning and computer vision, Convolutional Neural Networks (CNNs) stand as one of the most transformative architectures. Their development has catalyzed a revolution in tasks ranging from image and video recognition to medical image analysis. Unlike traditional machine learning techniques that might see images as a linear array of pixels, CNNs recognize hierarchical patterns, thereby imitating the way the human visual cortex interprets visual data.

The fundamental difference between CNNs and other neural network architectures lies in the initial layers where the network learns to filter inputs for useful information. These layers employ convolutional operations, from which the network derives its name. Through convolution, the network identifies patterns such as edges, textures, and more complex structures.

Pooling layers interspersed within the network play a vital role in reducing the spatial dimensions of the data, ensuring computational efficiency and allowing higher layers to recognize more generalized features. Fully connected layers at the end of a typical CNN architecture assimilate these features to make predictions or categorizations, depending on the task at hand.

The power of CNNs lies not just in their architectural design, but also in their ability to learn optimized feature representations directly from the data, eliminating the need for manual feature extraction that dominated traditional computer vision techniques. The backpropagation algorithm, coupled with optimization techniques like gradient descent, ensures that CNNs continually refine these features to improve task performance.

This chapter focuses on the application of Convolutional Neural Networks (CNNs) for object detection. A comprehensive overview of various CNN architectures is provided, aiding in the selection of the most appropriate model for a given project. The mechanics of CNNs are explored in detail, followed by a discussion on the selected model. Additionally, this chapter introduces custom training scripts that utilize the TensorFlow Object Detection Framework. The culmination of these discussions is evident in the results achieved for web element detection.

3.2 State of art

3.2.1 Background and Early Techniques

Before the rise of deep learning, traditional computer vision techniques formed the foundation of object detection. Among the earliest and most effective techniques, the Scale-Invariant Feature Transform (SIFT) [43] aimed at extracting key points from images that were invariant to changes in scale, rotation, and partially to changes in illumination and affine projection. The Histogram of Oriented Gradients (HOG) [44] was another groundbreaking technique, focusing on capturing the distribution of intensity gradients or edge directions to effectively detect objects, especially pedestrians, in images.

Parallel to these methods, the Viola-Jones object detection framework [45] made strides by introducing a rapid object detection scheme using a boosted cascade of simple features. This cascade structure allowed the algorithm to discard non-faces quickly, thus focusing its computational power on more promising regions of the image. Many of these early techniques were paired with machine learning classifiers, especially Support Vector Machines (SVM) [46], which worked by finding hyperplanes in a multidimensional space to segregate different classes of objects.

However, as promising as these early techniques were, they were often painstakingly handcrafted and heavily reliant on human intuition. They showed limitations when confronted with variability in object appearances, diverse backgrounds, and real-world conditions. This underlined the need for more adaptive, data-driven methods, setting the stage for the deep learning revolution that would redefine the domain of object detection [47].

3.2.2 Rise of Convolutional Neural Networks

The limitations of traditional methods spurred the exploration of neural networks, more specifically, Convolutional Neural Networks (CNNs), for computer vision tasks. CNNs can be traced back to the neocognitron introduced by Kunihiko Fukushima in 1980 [48]. However, it was only in the late 1990s and 2000s when datasets grew and computational capabilities improved, that CNNs began showing their potential.

Unlike traditional methods, CNNs automatically and adaptively learn spatial hierarchies of features from images. This hierarchal feature learning is achieved by stacking multiple convolutional layers which can learn patterns of increasing complexity [47].

The main components of CNNs include:

- Convolutional Layers: The layer responsible for the convolutional operation where filters slide over the input data to produce a feature map, effectively transforming the input data to a form that makes it easier to understand.
- Pooling Layers: These layers perform a down-sampling operation, reducing the spatial size of the representation and in turn, the computational complexity.
- Fully Connected Layers: After several convolutional and pooling layers, the high-level reasoning is performed via fully connected layers.

CNNs ability to learn intricate patterns without any explicit feature engineering made them outperform traditional methods. Yann LeCun's LeNet-5 [49] was one of the pioneering architectures that demonstrated the capabilities of CNNs, particularly for handwriting recognition.

From the foundational LeNet-5, CNN architectures have evolved substantially, pushing the boundaries of what is possible in computer vision. AlexNet [30], introduced in 2012, was a deeper and much more intricate architecture that achieved a breakthrough in the ImageNet Large Scale Visual Recognition Challenge.

This success was followed by other architectures like VGGNet [50], which increased the depth of networks; GoogLeNet or Inception [51], which introduced a new way to expand the size of networks without adding more parameters; and ResNet [52], which revolutionized training deeper networks by using "skip connections".

These architectures, while initially designed for image classification, have foundational designs that are adapted for object detection, segmentation, and other complex vision tasks. The rapid pace of research in this area implies that the state of the art is a moving target, but the consensus remains: CNNs have transformed the landscape of computer vision, making tasks previously considered difficult, now achievable.

3.2.3 Challenges and Limitations of CNNs

Despite the groundbreaking success and performance improvements of CNNs in computer vision tasks, they are not without challenges and limitations.

- Computational Costs: As the depth and complexity of CNNs increased with architectures like VGGNet and ResNet, so did the computational requirements. Training these deep networks demands vast amounts of computational power, often requiring multiple GPUs or TPUs and extended periods [53].
- Overfitting: While deep architectures can learn intricate patterns, they also have a higher propensity to overfit, especially when training data is limited. Techniques such as dropout [54], data augmentation, and regularization have been proposed to combat this.
- Model Interpretability: CNNs are often criticized as being "black boxes" because, even though they can make accurate predictions, it is hard to decipher exactly what these models have learned or how they make decisions [55].
- Adversarial Vulnerabilities: Recent studies have highlighted that CNNs can be highly vulnerable to adversarial attacks. Slight, often imperceptible perturbations to the input can lead a CNN to misclassify it, posing concerns for real-world deployments in critical applications [56].
- Transfer Learning Issues: While transfer learning has allowed pre-trained CNN models to be adapted for different tasks with limited data, it is not always guaranteed that features from one domain (e.g., ImageNet) are optimal for another domain [57].
- Bias and Fairness: There is growing evidence that models, including CNNs, can inherit and even amplify biases present in the data they are trained on. This has serious implications, especially when these models are used in sensitive applications like facial recognition [58].

Addressing these challenges is the focal point of much contemporary research. It requires a holistic approach, balancing architectural improvements, better training methodologies, and robust evaluation mechanisms.

3.2.4 Image Classification vs. Object Detection:

Image classification and object detection are foundational tasks with distinct objectives and challenges. Image classification focuses on categorizing an entire image into a class label, emphasizing recognizing global features and patterns. In contrast, object detection not only classifies multiple objects within an image but also pinpoints their spatial locations with bounding boxes, requiring the model to recognize both global and local features and understand spatial hierarchies and relationships. Image classification finds its applications in scenarios where the overall context is essential, like photo tagging or categorizing scenes. On the other hand, object detection is pivotal in contexts such as surveillance, robotics, and autonomous driving, where the spatial locations of specific objects are crucial.

One of the challenges common to both tasks is the viewpoint variation, where objects may appear different when viewed from various angles. However, object detection faces additional unique challenges. Objects in images can appear in various sizes, and their appearance can significantly differ based on their distance from the camera, making scale and size variations a significant challenge for object detection. Furthermore, occlusions, where objects might be partially obscured by other objects, pose a considerable challenge for object detection, whereas in image classification, the broader context can sometimes compensate for such issues. The computational complexity for object detection is generally higher because it involves not only classifying objects but also determining their spatial boundaries. Additionally, in datasets used for object detection, there can be a class imbalance where certain classes have fewer annotations compared to others, potentially skewing the model's performance.

In terms of the state of the art, image classification has seen advancements with architectures like ResNet [59], which introduced residual connections, EfficientNet [60] that scales deep neural networks, and the Vision Transformer (ViT) [61] that uses transformers for image classification. For object detection, significant contributions include Faster R-CNN [62], which integrated the Region Proposal Network with Fast R-CNN for quicker object detection, YOLO [63], emphasizing real-time detection, and EfficientDet [64], balancing accuracy with computational efficiency.

3.2.5 Residual Networks (ResNet)

Residual Networks (ResNet) have transformed the landscape of deep learning in computer vision [52]. The architecture pioneers the use of shortcut or residual connections to facilitate the training of much deeper networks without the network succumbing to the vanishing gradient problem.

The primary innovation in ResNet lies in its residual connections. Instead of stacking layers in a traditional sequential manner, each layer adds its output to the input, effectively learning the residual function. Mathematically, if x is the input and $F(x)$ is the output of a layer, then the subsequent layer receives $x + F(x)$ as input. This paradigm shift allows gradients to propagate more freely through the network.

ResNet-50 and ResNet-101 are among the most popular variants, distinguished by their depth. ResNet-50, with its 50 layers, strikes a balance between computational efficiency and performance, making it a preferred choice in many applications. ResNet-101, with its additional depth, often delivers superior accuracy on more intricate tasks.

On the benchmark ImageNet dataset, both ResNet-50 and ResNet-101 showed a significant reduction in top-1 and top-5 error rates compared to their predecessors. This prowess is not restricted to image classification; these models have been foundational in object detection frameworks like Faster R-CNN and in semantic segmentation. The applicability of ResNet architectures extends to transfer learning, where models pre-trained on ImageNet have expedited training and improved performance on distinct tasks.

3.2.6 Region Proposal Networks (RPNs)

Region Proposal Networks (RPNs) have revolutionized the efficiency and accuracy of object detection in modern convolutional neural networks (CNNs). In traditional object detection methods, a vast number of possible object regions were assessed, making the process computationally intensive. RPNs address this by introducing a mechanism to automatically propose candidate object regions and reduce the number of proposals, while maintaining a high detection rate [65].

The RPN operates by sliding a small network over the feature map output from a preceding CNN layer. This small network then outputs scores reflecting the likelihood of the object's presence for multiple predefined bounding boxes (anchors) of various sizes and aspect ratios at each sliding window location. Besides predicting object probabilities, the RPN also outputs bounding box regressors to adjust these anchors closer to the actual object boundaries [65].

Integration of RPNs into object detection models, such as Faster R-CNN, has resulted in a unified, end-to-end trainable network. This eradicates the need for an external region proposal method, accelerating both training and testing phases [52]. Furthermore, RPNs can be seamlessly integrated with popular CNN architectures like VGG or ResNet, further enhancing the generalizability and robustness of object detection models [52].

Faster R-CNN

Faster R-CNN can be perceived as a unified network that merges the RPN and Fast R-CNN object detection into a single, end-to-end trainable model. Its architecture comprises two main components: the RPN for generating region proposals and the Fast R-CNN detector to classify these proposals into object categories and refine their bounding boxes [65].

One of the significant achievements of Faster R-CNN is its real-time operation, making it applicable for a plethora of real-world scenarios. Furthermore, it is easily integrable with various deep learning backbones like VGG and ResNet, enhancing its versatility and object detection prowess [52].

3.2.7 Transfer learning using COCO dataset

Transfer learning has been foundational in deep learning, particularly when applied to tasks where data is limited or the computational capability required for training from scratch is prohibitive. A model like Faster R-CNN has significantly benefited from such techniques, with the COCO dataset being a go-to choice for initial pre-training [66]. The rich diversity and extensive annotations of the COCO dataset offer a broad understanding of generic object patterns, which can later be fine-tuned to more task-specific datasets or domains.

One of the main reasons for the success of transfer learning, especially with the COCO dataset, is the hierarchical feature representation in deep networks. Layers in these networks capture different levels of abstraction, from basic edge features in the initial layers to more complex patterns and object parts in deeper layers. Pre-training on a vast dataset like COCO allows a model to learn a wide variety of features, which are often generalizable across various tasks. Subsequent fine-tuning then adapts these features to more specific requirements [67].

In the context of Faster R-CNN, using weights pre-trained on COCO has shown to boost performance in various scenarios, be it detecting objects in medical images, aerial images, or even niche datasets. The advantages are not just in terms of accuracy; models converge faster, and training becomes more stable, thereby reducing the chances of overfitting when the target dataset is small [68].

While transfer learning, particularly from COCO, offers several advantages, there are challenges and considerations. One should be wary of domain gaps. If the source (like COCO) and target datasets are very dissimilar, naive transfer might not be ideal. In such cases, domain adaptation techniques or more advanced transfer learning methods might be necessary [69].

3.2.8 Evaluation metrics for Object Detection Models

Evaluating the performance of object detection models necessitates a combination of metrics that assess both the accuracy of classifications and the localization of objects within images. Precision and recall serve as foundational metrics, with precision indicating the accuracy of detected objects and recall demonstrating the model's coverage of actual objects in the image [70]. Derived from these is the Average Precision (AP) which, when computed over all object classes, is denoted as mean Average Precision (mAP). This metric performance across varying detection thresholds, proving to be integral in the object detection community [71]. Another crucial metric is the Intersection over Union (IoU), which measures the overlap accuracy between predicted and ground-truth bounding boxes [72]. Balancing precision and recall, the F1 Score is particularly invaluable for datasets with imbalanced classes [73]. Recent developments like the Object Detection Average Precision (ODAP) have been introduced to cater to challenges posed by images with multiple object instances, encapsulating both object localization and classification [74].

3.3 In-depth Analysis of Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for tasks that involve spatial hierarchies, such as image recognition. CNNs are inspired by the visual system's structure, particularly the organization of the visual cortex [75].

3.3.1 Key Steps in Convolutional Neural Network Processing

- **Convolution Operation:** In this operation, a filter slides over the input image and multiplies its values by the original pixel values in the image. This process is performed across the entire image, producing a feature map [49].
- **Pooling:** This step involves reducing the spatial dimensions of the feature map. For instance, max pooling selects the maximum value from a patch of an image and uses it to represent that region [76].

- **Flattening:** After several convolutional and pooling operations, the high-level features extracted are flattened into a single vector, which is then fed into the fully connected layers.
- **Classification:** In the final layer, the CNN often uses a softmax activation function to distribute probabilities across various classes, determining the input image's class [30]. For object detection tasks, additional layers and mechanisms are introduced to not only classify but also to localize the objects within the image.

3.3.2 Architecture of CNNs

The architecture of CNNs is fundamentally composed of layers, and each layer is responsible for extracting different features from the input (Figure: 3.1).

- **Input Layer:** This is where the image is fed into the network. Images are represented as matrices where each cell value in the matrix indicates a pixel's intensity.
- **Convolutional Layer:** The primary purpose of this layer is to extract features from the input image. Filters/kernels are used to slide over the image and perform the convolution operation, producing feature maps [27].
- **Activation Layer:** Post convolution, an activation function like the Rectified Linear Unit (ReLU) is applied to introduce non-linearity into the model [77].
- **Pooling Layer:** Pooling layers are used to reduce spatial dimensions, effectively down-sampling the feature maps while retaining crucial information. Max pooling and average pooling are commonly used methods.
- **Fully Connected Layer:** Towards the end of the network, fully connected layers are used to flatten the feature maps and make decisions based on the extracted features.

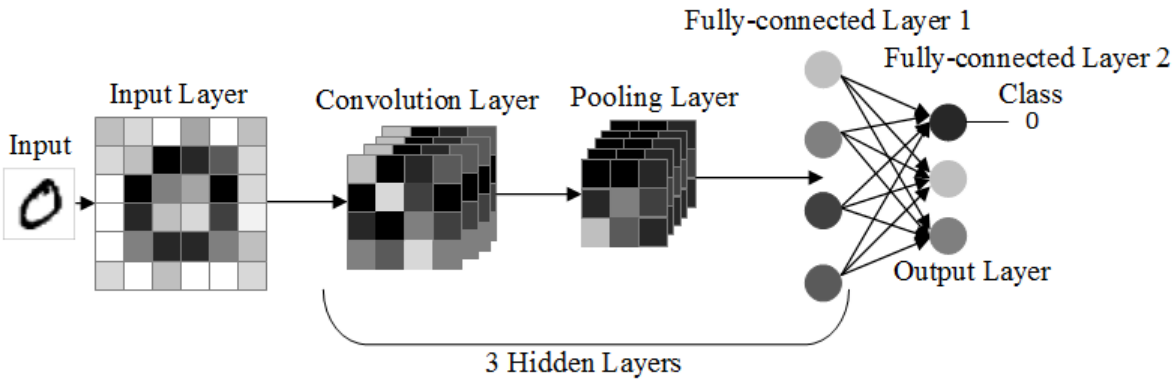


Figure 3.1: Convolution Neural Network Architecture

Input layer

The beginning of any CNN-based image processing task involves feeding an image into the network. The input layer (leftmost layer) represents the input image into the CNN. This input image serves as the foundation on which successive convolutional, pooling, and fully connected layers operate. In digital computing, an image is represented as a matrix of pixel values. A pixel (short for "picture element") is the smallest addressable unit of an image.

Grayscale vs. Color Image

- **Grayscale Image:** A grayscale image has only one channel where each pixel value indicates the shade of gray. The pixel values typically range from 0 (black) to 255 (white).

$$\text{Pixel Value} = [0 - 255]$$

- **Color Image:** Most standard color images use the Red-Green-Blue (RGB) color model and have three channels. Each channel represents the intensity of red, green, and blue in the image, respectively. For each channel, pixel values typically range from 0 (no color presence) to 255 (full color presence).

$$\text{Pixel Value} = \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Image Dimensions:
The dimensions of an image, in the context of a CNN, are represented as Height, Width, Channels.

- Height and Width represent the number of pixels in the vertical and horizontal dimensions of the image, respectively.

- Channels represent the depth of the image. A grayscale image has a depth of 1, while an RGB image has a depth of 3.

For instance, an RGB image of 224x224 pixels will have dimensions 224,224,3.

Normalization:
Before feeding the image into a CNN, it is common practice to normalize the pixel values. Normalizing ensures that each input parameter (pixel, in this case) has a similar data distribution, making convergence faster while training the model. Two common normalization methods are:

1. **Scaling between [0,1]:** By dividing each pixel value by 255 (since 8-bit pixel values range from 0 to 255), the image values are scaled between 0 and 1.

$$\text{Normalized Pixel Value} = \frac{\text{Original Pixel Value}}{255}$$

2. **Mean subtraction:** This involves subtracting the mean pixel value from each pixel in the image, which centers the data around zero. It is particularly useful when using activation functions that are centered around zero.

Convolutional and Activation layers

The convolutional layer is the core building block of a CNN. It performs the most critical computation of the network, namely the convolution operation. This layer scans the input (which could be the raw image or the output of a previous layer) to detect features like edges, corners, textures, and more.

Convolution Operation:

1. **Filters/Kernels:**
    - Filters, also known as kernels, are small, learnable weight matrices whose dimensions are much smaller than the input image.
    - A typical size might be 3 x 3 or 5 x 5.
    - During the convolution operation, these filters slide or convolve around the input image to produce a feature map or convolutional output.
2. **Feature Map:**
    - The feature map is the output of one filter applied to the previous layer.
    - A convolutional layer can contain multiple filters, each producing its own feature map. These maps are stacked depth-wise to produce the final output.
3. **Convolution Operation Mechanism:**
    - At each position of the filter on the input, the convolution involves multiplying the values of the filter with the original pixel values in the image. These products are summed up, and the result forms a single pixel in the output feature map (Figure: 3.2). This process is then repeated across the entire image.

$$\text{Output Pixel Value} = \sum(\text{Filter Values} \times \text{Image Values})$$

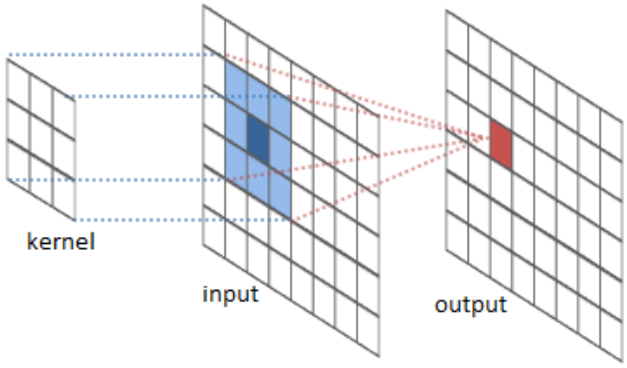


Figure 3.2: Convolution Representation

Stride and Padding:

1. **Stride:**
    - Stride defines how much the filter should move when sliding over the input image.
    - A stride of 1 means the filter moves one pixel at a time, while a stride of 2 means it jumps every second pixel (Example Figure: 3.3). Larger strides result in smaller feature maps.

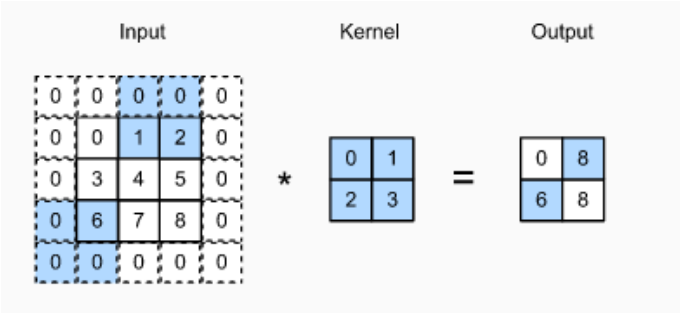


Figure 3.3: Example of cross-correlation with strides of 3 and 2 for height and width, respectively.

2. **Padding:**
    - Padding refers to adding extra pixels around the border of the input image (Example Figure: 3.4).
    - This ensures that the spatial dimensions (i.e., width and height) of the output feature map can be controlled or maintained.
    - Two common types of padding are:
        - Valid Padding: No padding (output size < input size).
        - Same Padding: Padding such that output size = input size when using a stride of 1.

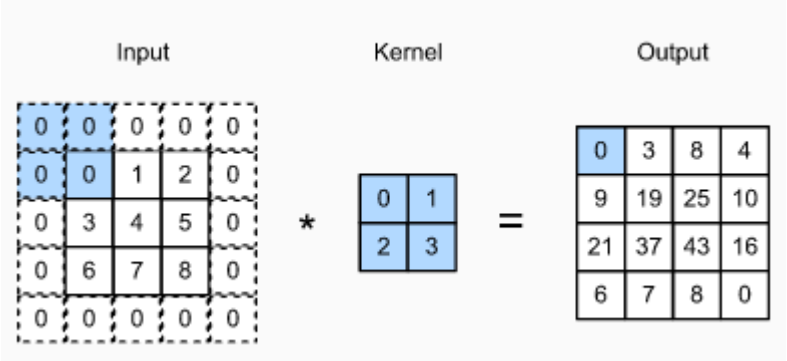


Figure 3.4: Example of two-dimensional cross-correlation with padding.

Non-linearity (Activation Function):
After the convolution operation, it is common to pass the result through a non-linear activation function like the Rectified Linear Unit (ReLU). This introduces non-linearity into the model, which is essential for the network to learn from error and adjust its weights.

In neural networks, an activation function is a mathematical operation applied to the output of a neuron or layer. It introduces non-linearity to the model, allowing the network to represent more complex functions and make decisions based on a broader set of information. Without activation functions, a neural network, regardless of its depth, would act as a linear regressor, rendering it incapable of approximating intricate patterns in data.

In the context of Convolutional Neural Networks (CNNs), activation functions play a pivotal role. CNNs, primarily employed for image processing tasks, leverage multiple convolutional layers to automatically and adaptively learn spatial hierarchies of features. Activation functions help ensure the transformation of raw pixel values and subsequent feature maps into representations that can be useful for a downstream task, such as image classification or object detection.

In CNNs, the primary objective of activation functions is to add depth to the learning capability, enabling the network to understand intricate patterns and details from the input images. They act as a gate, deciding which information should progress further down the network and which should be discarded. Activation functions, therefore, contribute significantly to the robustness and performance of the CNN.

1. **Rectified Linear Unit (ReLU):**

$$f(x) = \max(0, x)$$

   - **Description:** ReLU is the most commonly used activation function in CNNs. It introduces non-linearity into the model without affecting the receptive fields of convolutional layers. The function returns x if x is positive, else it returns zero.
   - **Advantages:** Helps mitigate the vanishing gradient problem, computationally efficient.
   - **Disadvantages:** Can suffer from the "dying ReLU" problem, where neurons can sometimes get stuck during training and always output zero.

2. **Leaky Rectified Linear Unit (Leaky ReLU):**

$$f(x) = \begin{cases} x & \text{if } x \text{ is positive} \\ \alpha x & \text{else} \end{cases}$$

   where α is a small positive constant.

   - **Description:** This is a variation of the ReLU designed to address the dying ReLU problem. It allows a small, non-zero gradient when x is negative.
   - **Advantages:** Reduces the dying ReLU problem.
   - **Disadvantages:** Performance is sometimes inconsistent across different datasets compared to ReLU.

3. **Parametric ReLU (PReLU):**
   Similar to Leaky ReLU, but α is learned from the data rather than being pre-defined.

   - **Description:** Allows the network to learn the value of α for each neuron.
   - **Advantages:** Offers more flexibility and can often outperform ReLU and Leaky ReLU.
   - **Disadvantages:** Can be more prone to overfitting on smaller datasets.

4. **Exponential Linear Unit (ELU):**

$$f(x) = \begin{cases} x & \text{if } x \text{ is positive} \\ \alpha(\exp(x) - 1) & \text{else} \end{cases}$$

- **Description:** Tries to make the mean activations closer to zero which speeds up learning. It takes on negative values when $x < 0$, unlike ReLU.
- **Advantages:** Helps mitigate the vanishing gradient problem, mean activations are closer to zero.
- **Disadvantages:** Computationally more expensive due to the exponential function.

5. **Swish:**

$$f(x) = x \times \sigma(x)$$

where σ is the sigmoid function.

- **Description:** A self-gated activation function. It tends to outperform ReLU on deeper models.
- **Advantages:** Tends to perform well in practice, especially for deeper networks.
- **Disadvantages:** Computationally more expensive than ReLU.

6. **Mish:**

$$f(x) = x \times \tanh(\ln(1 + \exp(x)))$$

- **Description:** A relatively newer activation function that has shown good performance in some applications.
- **Advantages:** Can offer improved performance in certain cases over ReLU.
- **Disadvantages:** Computationally more intensive.

Pooling layer

The pooling layer serves as a downsampling operation that reduces the spatial dimensions of the feature maps, thereby decreasing the computational requirements for subsequent layers and also providing a form of translational invariance. Pooling layers are usually inserted between successive convolutional layers in a CNN.

Max Pooling

Definition: Max pooling is a subsampling method where the input is partitioned into a set of non-overlapping rectangles, and for each such sub-region, the maximum value is chosen as the representative.

How it works: Given a feature map, a window (often called a pooling window or kernel) of a fixed size $k \times k$ slides over the feature map with a certain stride s. At each step, the maximum value within the window is selected and forms the new pooled feature map (Example Figure: 3.5).

Example: If consider a 2×2 max pooling operation on the following matrix:

$$\begin{matrix} 1 & 3 \\ 4 & 2 \end{matrix}$$

The output would be 4 since that is the maximum value among these four numbers.

Benefits:

- **Reduction in spatial dimensions:** By reducing the size of the feature maps, it reduces the number of parameters and computations in the network, which can lead to a decrease in overfitting.
- **Invariant to small translations:** A slight shift in the input might not change the output of max pooling because it focuses on the most prominent feature in a local region.

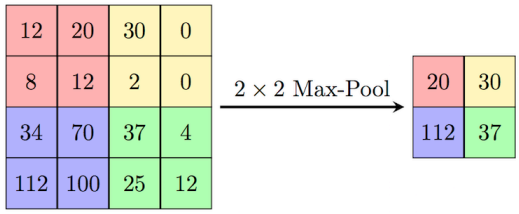


Figure 3.5: Example Max Pooling.

Average Pooling

Definition: Average pooling, as the name suggests, involves taking the average of the elements in the pooling window.

How it works: Given a feature map, a window of a fixed size $k \times k$ slides over the feature map with a certain stride s. At each step, the average value within the window is computed and used to form the new pooled feature map (Example Figure: 3.6).

Example: Using a 2×2 average pooling operation on the same matrix:

$$\begin{matrix} 1 & 3 \\ 4 & 2 \end{matrix}$$

The output would be $\frac{1+3+4+2}{4} = 2.5$.

Benefits:

- **Retains more information:** Unlike max pooling, which only retains the maximum value, average pooling considers all values in the window, so it retains more information from the original input.
- **Smoother down-sampling:** By taking averages, it ensures that feature maps are down-sampled smoothly, potentially retaining more nuanced features.

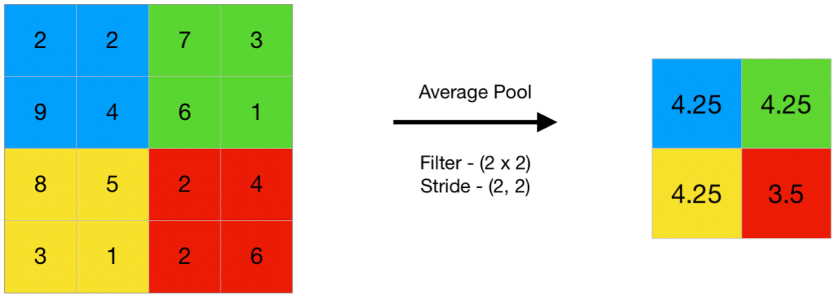


Figure 3.6: Example Average Pooling.

Function of the Pooling Layer:

1. **Dimensionality Reduction:** Pooling reduces the spatial dimensions (height and width) of the input feature map. This is beneficial because it decreases the amount of parameters and computations in the network, which helps in preventing overfitting and also reduces computational costs.

2. **Translational Invariance:** By discarding some of the spatial information, pooling layers provide a form of translational invariance. This means that even if an object in the image is slightly shifted, the pooled feature map remains largely unchanged, allowing the network to recognize objects irrespective of their exact position in the image.

3. **Hierarchical Feature Learning:** Each layer tends to learn increasingly abstract features. By using pooling layers, the network is forced to represent information in a condensed form, facilitating this hierarchical learning process. For instance, while early layers might detect edges, subsequent layers might detect patterns made up of edges, and even further layers might detect objects made up of these patterns.

Configuration of Pooling Layers:

- **Pool Size:** This refers to the size of the pooling filter (e.g., 2×2). A larger pool size results in a more aggressive downsampling.
- **Stride:** The stride controls how much the pooling window is shifted by on each step. A stride of 2 with a 2x2 pool size would mean no overlap, whereas a stride of 1 would mean the pooling windows overlap.

Fully connected layer

After feature extraction and spatial reduction using convolutional, activation, and pooling layers, the Fully Connected (FC) layer serves as the final step where high-level reasoning happens. In essence, FC layers are the traditional artificial neural network (ANN) layers where every neuron is connected to every other neuron in the previous and subsequent layers.

1. **Structural Context**: Convolutional and pooling operations in CNNs produce output tensors that have spatial dimensions (height and width) along with depth (corresponding to the number of feature maps or channels). For instance, a convolutional layer can output a tensor of shape $H \times W \times D$, where H is the height, W is the width, and D is the depth of the feature map.

2. **Transition to Fully Connected Layers**: Fully connected layers, as the name suggests, have neurons that are connected to all activations in the previous layer. However, they expect their input data to be in a flat, vectorized form rather than a multi-dimensional tensor. Here is where flattening comes into play.

3. **Flattening Operation**: Flattening is a preprocessing step that transforms the structure of the data from a multi-dimensional tensor into a one-dimensional vector. This transformation is necessary when transitioning from convolutional and pooling layers in a CNN to fully connected

(dense) layers. Flattening involves taking each row of each channel of the tensor and lining them up into a single vector. The order can vary, but a common approach is to iterate over the depth (channels) and then serialize each 2D feature map row by row.

For a tensor of shape $H \times W \times D$, after flattening, the resulting vector will have a size of $H \times W \times D$.

As a simple example, consider a tensor of shape $2 \times 2 \times 2$ (two channels, each of 2x2 size):

$$\left[\begin{bmatrix} a & b \\ c & d \end{bmatrix}, \begin{bmatrix} e & f \\ g & h \end{bmatrix}\right]$$

After flattening, the resulting vector would be:

$$[a, b, c, d, e, f, g, h]$$

Flattening bridges the gap between the spatial processing layers (convolutional and pooling) and the decision-making layers (fully connected). It ensures that spatial hierarchies and patterns detected by convolutional layers are made available in a format that can be ingested by dense layers for final classification or regression tasks (Figure: 3.7).

While flattening is a straightforward operation, it is essential to ensure the sequential ordering remains consistent during both the forward and backward passes in training. Any inconsistency can lead to misalignment of learned weights and activations, leading to incorrect model training.

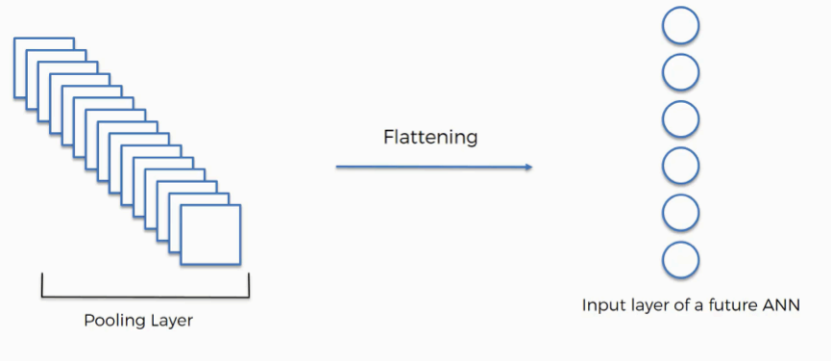


Figure 3.7: Example Flattening process.

Classification in CNNs

After processing through convolutional layers, activation functions, pooling layers, and the fully connected layers, the CNN provides a final output that corresponds to class scores. These scores represent the probability that a given input belongs to each of the possible classes. The task of translating these scores into meaningful class predictions is called classification.

The last fully connected layer produces outputs for each potential class. These outputs, often referred to as 'logits', are raw scores representing the model's confidence that the input belongs to each respective class. These scores are the groundwork for generating probabilities and making a final prediction.

An activation function like softmax function is a vital component in classification. It takes the logits from the last fully connected layer and converts them into probabilities. Once we have the probabilities for each class (from the softmax function), the class with the highest probability is taken as the model's prediction. In the training phase, the predicted probabilities are compared to the true labels using a loss function, often the categorical cross-entropy for classification problems.

Importance of Classification in CNNs:

1. **Decision Point:** Classification is the culmination of all the previous steps in a CNN. All the feature extraction and processing in earlier layers lead to this point where a final decision is made.
2. **Evaluating Performance:** The accuracy of the classification, along with other metrics like precision, recall, and F1-score, gives a measure of the model's performance.
3. **Feedback Loop:** The difference between predicted and actual labels (the loss) serves as feedback, guiding the optimization of the network's weights during training.

Activation Functions for Classification Output Layers

Neural networks process information and produce outputs that can be used for various tasks such as regression, clustering, and classification. In the context of classification, a specific subset of activation functions is tailored to produce outputs suitable for class probabilities or decisions. The choice of the output activation function is crucial since it ensures that the network outputs can be interpreted in the context of the given classification task.

1. **Sigmoid:**

$$f(x) = \frac{1}{1 + \exp(-x)}$$

   - **Description:** Sigmoid maps the input values into the range between 0 and 1.
   - **Advantages:** Smooth gradient, output values bound between 0 and 1.
   - **Disadvantages:** Suffers from the vanishing gradient problem, especially for input values that are very positive or very negative.

2. **Tanh (Hyperbolic Tangent):**

$$f(x) = \tanh(x) = \frac{2}{1 + \exp(-2x)} - 1$$

   - **Description:** Tanh is similar to the sigmoid but maps input values into the range between -1 and 1.
   - **Advantages:** Output is zero-centered, meaning negative and positive input values are roughly in the same range.
   - **Disadvantages:** Like the sigmoid, it can suffer from the vanishing gradient problem.

3. **Softmax**
   For input vector $\mathbf{z}$ of length K, the softmax function is defined for each element z_i as:

$$f(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}$$

   - **Description**: The softmax function generalizes the sigmoid function to multi-class problems. It converts the raw output scores (often called logits) from the network into a normalized probability distribution over the classes.
   - **Used**: Typically used for the output layer in multi-class classification problems.

Object Detection in CNNs

Object detection is an advanced application of convolutional neural networks (CNNs) that not only classifies individual objects in an image but also provides spatial localization for each detected object, typically in the form of bounding boxes. Unlike image classification, where the output is a single label or class, object detection aims to identify multiple objects and their locations within the same image. Modern object detection methods, such as Faster R-CNN, employ a Region Proposal Network (RPN) to suggest potential bounding boxes that might contain objects. This significantly reduces the number of candidates that need to be evaluated, making the detection process more efficient.

An RPN is a neural network that scans an image for potential bounding boxes that might contain objects. Its primary objective is not to classify these objects but to provide regions (proposals) where objects might be found. These proposed regions are then passed onto a more robust classifier for further processing. RPNs function by sliding a small window (often referred to as an "anchor") over the feature map produced by a preceding CNN layer. For each position of this window, the RPN predicts both:

1. The likelihood that the window contains an object.
2. Coordinates to adjust the window to better fit the object.

Anchors in RPNs are essentially starting reference boxes with different scales and aspect ratios. Given the varying sizes and shapes of objects in real-world images, using multiple anchors at each spatial position in the feature map allows the RPN to detect objects of different sizes and shapes.

Before the advent of RPNs, object detection methods like Fast R-CNN utilized selective search or other mechanisms to generate region proposals. These methods were often time-consuming and not end-to-end trainable. RPNs, being part of the neural network, can be jointly trained with the classifier and regressor, ensuring a more harmonious learning process and better performance. This integration leads to the Faster R-CNN architecture, which is both efficient and effective for object detection tasks.

After the RPN produces region proposals, these regions are then processed further using a detection network. The role of this subsequent network is to classify the object within each proposed region and refine the bounding box coordinates. The synergy between the RPN and the detection network ensures that objects are both accurately localized and classified.

3.3.3 CCNN in Practice: A Step-by-step Example for Object Detection

This section illustrate the operations of a CNN in object detection by using a representative example: determining the bounding box and classifying the object within an image.

1. **Input:** Consider a color image with dimensions 64x64 pixels. This can be represented as a 64x64x3 tensor due to the three RGB channels, with each tensor value specifying the pixel intensity ranging between 0 to 255.

2. **Convolutional Operation**

    - **Definition: Filters/Kernels:** A filter, also known as a kernel, is a matrix of smaller dimensions utilized to traverse the input image, aiming to extract specific features.
    - **Mechanism:** For illustrative purposes, a 3x3 filter can be applied to detect rudimentary features such as edges. Through convolution, this filter traverses the 64x64 image, producing a feature map that delineates the regions where the features are identified.

3. **Activation Operation:** Subsequent to the convolution operation, the resultant feature map may encompass values beyond the desired range or negative values. To address this, an activation function, such as the Rectified Linear Unit (ReLU), is employed. The function primarily nullifies negative values while maintaining the positive ones.

4. **Pooling Operation:**

    - **Definition: Down-sampling:** The pooling operation aims to decrease the spatial dimensions of the feature map whilst preserving salient information.
    - **Mechanism:** For instance, a 2x2 max-pooling approach evaluates 2x2 regions of the feature map, retaining the maximal value from each region. This method effectively compresses the spatial dimensions of the feature map.

5. **Region Proposal Network (RPN):** Within the context of object detection, the RPN is imperative for suggesting plausible object regions. Utilizing anchor boxes with diversified sizes and aspect ratios, the RPN forecasts the likelihood of object presence in tandem with fine-tuning the anchor box dimensions to encase the object optimally.

6. **RoI Pooling:** The Region of Interest (RoI) pooling process standardizes the sizes of the proposed object regions. This ensures uniformity in size, facilitating processing by subsequent layers.

7. **Fully Connected Layer:** Post the RoI pooling, the regions are flattened and directed through fully connected layers. These layers are instrumental in ascertaining the object's class and fine-tuning the bounding box coordinates for accurate object localization.

8. **Classification and Bounding Box Regression:** Outputs from the fully connected layers bifurcate into two segments: one dedicated to object classification and the other to refining the bounding box coordinates. A softmax activation function computes a probability distribution over possible classes. Concurrently, the bounding box regression refines the object's positional coordinates.

3.4 Selecting the Optimal Architecture for Object Detection in Project

3.4.1 Justification

In the context of Convolutional Neural Networks (CNNs), the choice of architecture plays a pivotal role in determining the overall performance of the system. While real-time detection systems such as YOLO (You Only Look Once) [78] offer impressive speed and decent accuracy, they are fundamentally designed for tasks that prioritize low-latency detection.

On the contrary, for many applications, the highest possible accuracy is the paramount criterion, even if it entails more processing time. In such cases, architectures like ResNet [52] become an attractive choice for several reasons:

1. **Deep Layers for Enhanced Feature Learning:** ResNet models, particularly deeper variants like ResNet-101, have multiple layers which facilitate intricate feature extraction. This deep architecture allows the network to capture and learn from a wide range of features from the input data, enhancing its capacity to generalize well on diverse datasets.

2. **Residual Connections:** ResNet's introduction of skip (or residual) connections ensures that gradients can propagate deeper into the network without the vanishing gradient problem. This characteristic enables the training of very deep networks, which can be essential for capturing intricate patterns and achieving higher accuracy.

3. **Trade-off Between Speed and Accuracy:** While YOLO focuses on rapid object detection, it sometimes compromises on accuracy to achieve real-time performance. In contrast, ResNet offers a balance that leans more towards accuracy, even if it requires additional computational time.

4. **Versatility:** ResNet models have been pre-trained on vast datasets like ImageNet [23], making them versatile and suitable for a wide range of tasks, including both classification and detection. The rich feature representations learned from such training can be highly beneficial when aiming for top-tier accuracy.

In conclusion, for scenarios where the utmost accuracy is desired and a slight delay in processing time is acceptable, the ResNet architecture stands out as an optimal choice over real-time detection systems like YOLO.

3.5 In-depth Analysis of RFCN-ResNet101

RFCN-ResNet101 is an implementation of the RFCN (Region-based Fully Convolutional Networks) object detection approach, where ResNet-101 is used as the backbone architecture for feature extraction.

3.5.1 ResNet-101

The traditional neural networks often degrade in performance when they are deepened, a problem which is not only because of overfitting. A deep learning framework called Residual Networks (ResNets) to address this issue. By introducing shortcut connections that bypass one or more layers, ResNets are designed to solve the degradation problem. When these residual networks are increased in depth, they achieve more accurate results [52].

In the subsequent sub-sections, the ResNet architecture will be examined in depth, with an emphasis on its design and functional features. The primary reference for this exploration is the paper "Deep Residual Learning for Image Recognition" by He et al. [52]. All subsequent references and discussions derive from this publication. The paper's motivation, methodology, and pivotal results will be explored to comprehend the significance of ResNet. The overarching aim is to shed light on the profound influence ResNet has exerted on the deep learning domain and to highlight the rationale behind its selection as the architectural choice for this thesis project.

Introduction and Motivation Behind Deep Residual Learning

In the domain of deep learning, the network depth is crucial for achieving good performance. This is because deep networks can represent complex functions with a smaller number of parameters compared to shallow networks. However, as networks grow deeper, they often face two main challenges: vanishing/exploding gradients and the degradation problem.

Degradation Problem

The degradation problem is particularly intriguing. Traditionally, adding more layers to a neural network was believed to only pose the risk of overfitting, especially when there is insufficient data. However, He et al. observed that deepening the networks led to a higher training error, indicating that the problem is not just overfitting. This raised the question: if added layers are meant to make the function more expressive, then why does a deeper model – which should only provide at least as good a performance as a shallower counterpart – result in higher training error?

Vanishing/Exploding Gradients

This is another challenge where, in very deep networks, gradients can become too small (vanish) or too large (explode) as they are propagated backward through the layers. This makes the networks very hard to train, as the layers in the beginning (in case of vanishing gradients) or at the end (in case of exploding gradients) learn very slowly or destabilize the learning, respectively.

Deep Residual Learning Hypothesis

Given these challenges, the authors hypothesized that direct paths between earlier and later layers might alleviate these problems. They posited that if deep networks can be seen as a composition of multiple nonlinear transformations, having direct paths (or shortcuts) could assist in directly learning the identity function over some transformations, which could aid optimization.

To tackle the degradation problem and potentially help with the vanishing/exploding gradients issue, the authors introduced the concept of residual learning. Instead of asking the network to learn an underlying mapping directly, they asked it to learn the residual or difference between the underlying mapping and the input. This fundamental shift in perspective became the basis for the Deep Residual Networks.

Residual Learning: Concept and Formulation

In conventional deep networks, each layer attempts to transform its input feature maps into a set of output feature maps, potentially aiming to learn the desired underlying mapping of the data. In contrast, with residual learning, a layer learns the residual or difference between its input and the desired output (Figure: 3.8).

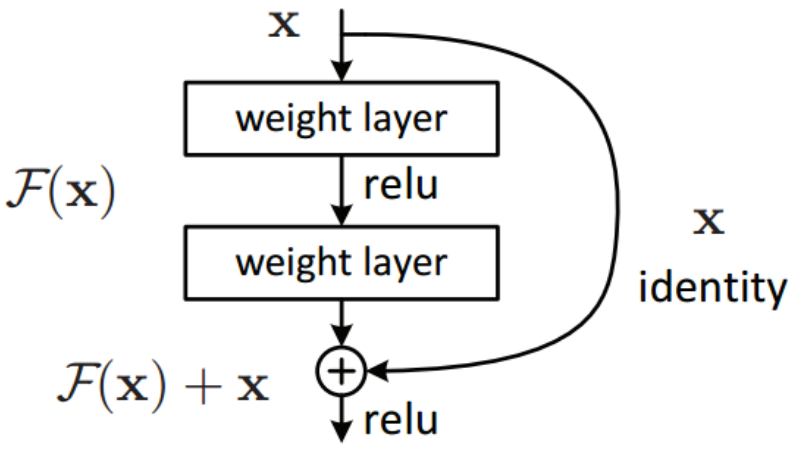


Figure 3.8: Residual learning: a building block

$H(x)$ represent the desired underlying mapping targeted for a stack of layers to learn. Rather than approximating this function directly, residual learning seeks to model the residual function:

$$F(x) = H(x) - x \tag{3.1}$$

From this, the original function can be derived by reconfiguring the equation to:

$$H(x) = F(x) + x \tag{3.2}$$

In this context, $F(x)$ denotes the output from the stack of layers, while x signifies their input. The sum $F(x) + x$ is achieved via a shortcut connection that bypasses the layers.

Shortcut Connections

One of the main innovations of resnet model is the introduction of shortcut connections, the proposal model forward the input feature map directly to a later layer in the network without any modification. These connections do not introduce any additional parameters or computational complexity. They skip one or more layers and then combine their input with the output of the layers they skipped.

Benefits of Shortcut Connections:

1. **Easier Optimization:** Shortcut connections mitigate the degradation problem by providing direct paths for the information to flow. As a result, even if the stacked layers (or a subset of them) are not contributing positively to the learning process, the overall network performance does not degrade thanks to the identity mappings provided by shortcut connections.

2. **Addressing the Vanishing Gradient:** These shortcuts also provide direct paths for gradient flow during backpropagation, which can help to some extent in alleviating the vanishing gradient problem seen in deep networks. The gradients have more direct paths to earlier layers, potentially making training more stable and efficient (Figure: 3.9).

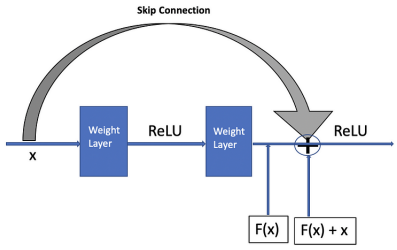


Figure 3.9: Example of shortcut connection

In essence, residual learning reformulates the layers as learning residuals with reference to the layer inputs, instead of learning unreferenced functions. This important change in learning strategy, aided by shortcut connections, becomes the foundation for constructing deep and efficient networks.

ResNet Architecture: Design Principles and Implementation

Residual Networks, or ResNets, were born from the principle of residual learning. Their design primarily revolves around building blocks that employ shortcut connections to facilitate the learning of residual functions. For most of the datasets of images with dimensions smaller than 224×224, a simple building block is utilized. This block comprises two layers of 3×3 convolutions. The shortcut connection skips these two layers. If the dimensions of the input and output are different, a linear projection by shortcuts is performed to match the dimensions.

For deeper networks, especially those used in the ImageNet challenge with larger image dimensions, a bottleneck design was adopted to reduce computational complexity. In this design (Figure: 3.10), a building block is made up of three layers: a 1×1, a 3×3, and another 1×1 convolution. The 1×1 convolutions are responsible for reducing and then restoring the dimensions, which makes the 3×3 convolution the bottleneck with fewer input-output channels.

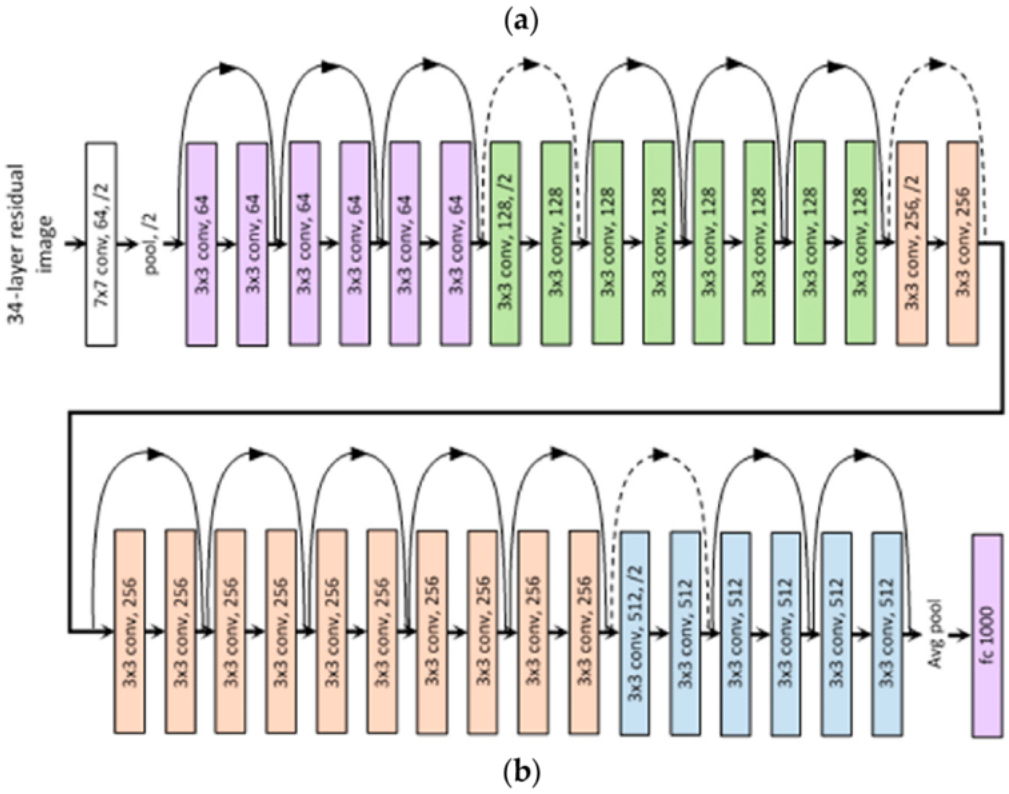


Figure 3.10: Example of resnet architecture

Network Depth and Training

One of the significant advantages of ResNets is their capability to be very deep without facing the issues traditionally associated with deep networks. In their work, He et al. trained ResNets that were much deeper than previous architectures, with models comprising 50, 101, and 152 layers. The deeper models, such as the 152-layer one, were achieved by increasing the number of bottleneck blocks.

Batch normalization is applied after each convolution and before activation. The Rectified Linear Unit (ReLU) is the activation function used throughout the network. Instead of pooling layers, ResNets utilize convolutions with a stride of 2 to down-sample feature maps. This design choice reduces the number of hyperparameters and keeps the network more uniform. The end of the network uses global average pooling to reduce the spatial dimensions, followed by a fully connected layer and softmax for classification.

Achievements on ImageNet

ResNets achieved remarkable performance on the ImageNet Large Scale Visual Recognition Challenge:

1. **ResNet-34:** A ResNet with 34 layers, using the basic building block, was first introduced. It outperformed its deeper plain counterpart, showcasing the efficacy of residual connections.
2. **ResNet-50, ResNet-101, and ResNet-152:** Using the bottleneck architecture, these deeper variants of ResNets were introduced. ResNet-152, with 152 layers, was the deepest network trained by the authors. It achieved an impressive top-1 error rate of 22.44% and a top-5 error rate of 6.71% on ImageNet's validation set, setting a new state-of-the-art.
3. **Ensemble Performance:** An ensemble of multiple ResNets further reduced the top-5 error to 3.57%, winning the 1st place in the ILSVRC 2015 classification task.

The success of ResNets signifies more than just a breakthrough in image classification performance. It highlights a paradigm shift in how deep learning models are perceived and designed. By introducing residual connections, the degradation problem, which had long limited the depth of neural networks, was substantially mitigated.

Depth in neural networks was traditionally linked with both increased representational power and heightened optimization challenges. ResNets have illustrated that with the appropriate architectural designs, the pitfalls associated with increased depth can be effectively managed, allowing for networks that were previously deemed infeasible. The principles behind ResNets have found utility beyond the realm of image classification. Their impact is evident across a spectrum of applications, including object detection, semantic segmentation, and even tasks outside the visual domain. This demonstrates the robustness and versatility of the residual learning paradigm.

Challenges and Considerations

1. **Computational Demand:** Deeper networks, even with the benefits of ResNets, demand more computational power. This can be a constraint, especially when deploying models in resource-limited environments or real-time applications.

2. **Model Interpretability:** With increased depth, understanding the internal workings of the model becomes challenging. As these networks become more intricate, ensuring their transparency and interpretability is crucial, especially for critical applications.

3. **Interactions with Other Architectural Innovations:** While ResNets provide a solution to the degradation problem, how they synergize or potentially conflict with other novel neural network designs remains an area of research and exploration.

3.5.2 RFCN (Region-based Fully Convolutional Networks)

Region-based Fully Convolutional Networks (R-FCN), proposed by Dai et al. [79], aim to integrate the merits of both region-based object detection methods and fully convolutional networks. These networks provide a fully convolutional end-to-end solution to object detection while retaining the accuracy benefits associated with methods that use regions. RFCN presents an approach for accurate and efficient object detection using region-based, fully convolutional networks. The core innovation lies in the proposal of position-sensitive score maps which aim to balance between translation-invariance for image classification and translation-variance for object detection [80].

Traditional region-based detectors applied a per-region subnetwork multiple times, which was computationally expensive. In contrast, the presented region-based detector is almost entirely convolutional, sharing most of the computation across the whole image. This was achieved by leveraging fully convolutional image classifier backbones, like Residual Networks (ResNets).

The decomposition of traditional object detection networks into two subnetworks, separated by the Region-of-Interest (RoI) pooling layer, led to a design that had a shared convolutional subnetwork and an RoI-wise subnetwork. However, newer image classification networks like ResNets are fully convolutional. Directly applying these networks for object detection led to suboptimal results. A major challenge was balancing the translation invariance required for image classification and the translation variance necessary for object detection. To address this, the authors introduced the Region-based Fully Convolutional Network (R-FCN) [80]. This network uses position-sensitive score maps, built using specialized convolutional layers, to encode spatial information necessary for object detection. The final architecture is trained end-to-end and is entirely convolutional, sharing computation across the entire image [80].

The approach presented in the paper R-FCN: Object Detection via Region-based Fully Convolutional Networks [80] follows a two-stage object detection strategy: (1) region proposal and (2) region classification. The Region Proposal Network (RPN) is used for extracting candidate regions, and the R-FCN is designed to classify these regions into object categories or background (Figure: 3.11). In the R-FCN, all learnable layers are convolutional and are computed on the entire image. The final convolutional layer produces position-sensitive score maps for each category. The R-FCN concludes with a position-sensitive RoI pooling layer that aggregates outputs and generates scores for each region [80].

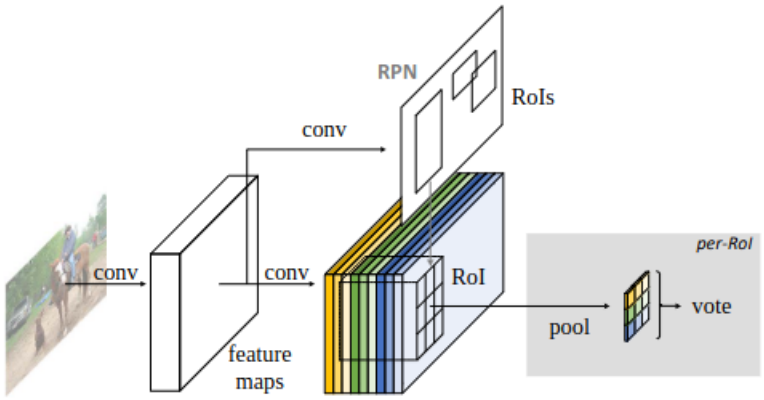


Figure 3.11: Overall architecture of R-FCN. A Region Proposal Network (RPN) [1] proposes candidate RoIs, which are then applied on the score maps. All learnable weight layers are convolutional and arecomputed on the entire image; the per-RoI computational cost is negligible

The R-FCN architecture comprises three primary components:

1. **Backbone Network:** Typically a deep Convolutional Neural Network (CNN) is used to extract feature maps from the input image. Popular choices for this backbone include ResNet [52] and VGGNet [50].

2. **Region Proposal Network (RPN):** Upon the feature maps produced by the backbone network, an RPN generates region proposals that potentially contain objects. The RPN is fully convolutional, meaning that it can operate on any input size and produce proposals at multiple scales and aspect ratios.

3. **Position-Sensitive Score Maps:** Instead of using fully connected layers to produce classification scores and bounding box regressors for each region proposal as in Fast R-CNN [68], R-FCN utilizes position-sensitive score maps. These are a set of 2D maps that correlate with relative spatial positions within the region proposals. The classification and localization processes are performed using these maps, enabling the network to be fully convolutional.

Working Principle

- The feature maps from the backbone network feed into the RPN to produce region proposals.
- These proposals are then mapped onto the position-sensitive score maps. For each object category, a fixed set of position-sensitive maps is generated. These maps encode both the spatial position and the object category.
- The final object detection scores are obtained by pooling from these position-sensitive maps according to the relative positions of pixels within each region proposal. This pooling operation allows the network to determine which parts of the region proposal correspond to which parts of an object, thereby facilitating accurate object detection.

Advantages of R-FCN

- **Computation Efficiency:** Since the R-FCN is fully convolutional, it shares computation across the entire image, resulting in efficient inference, especially for images with multiple objects.
- **Accuracy:** Despite being efficient, R-FCN retains high levels of accuracy, rivalling or even surpassing the performance of other state-of-the-art object detectors.

3.5.3 RFCN-ResNet101

Region-based Fully Convolutional Networks (R-FCN) operate on the principle of applying a fully convolutional network over the entire image to compute the convolutional features and then employ position-sensitive score maps to determine the class and bounding box for each object. The R-FCN architecture effectively eliminates the time-consuming process of region proposal generation, making object detection faster without sacrificing accuracy.

ResNet101, a deep residual network with 101 layers, is employed as the backbone for R-FCN. It is an extended version of the original ResNet and has demonstrated competitive performance in image classification tasks. Residual networks address the vanishing gradient problem by introducing skip connections, also known as shortcut connections, which allow the network to learn identity functions for certain layers. This enables the training of very deep networks without significant degradation in performance.

When combined, RFCN-ResNet101 offers a powerful tool for object detection. ResNet101 provides the deep feature extraction capabilities, while R-FCN offers a computationally efficient approach to object detection. The fusion of these two architectures allows for accurate and efficient object detection in various applications.

To implement the RFCN-ResNet101, one can utilize pre-trained models available in popular deep learning frameworks. For instance, OpenVINO offers a pre-trained RFCN-ResNet101 model optimized for the TensorFlow framework, which can be employed in various computer vision applications [81].

3.6 Training the Dataset with ResNet-101 via TensorFlow Object Detection Framework

3.6.1 Advanced Machine Learning Frameworks for Efficient Training of Convolutional Neural Networks (CNNs)

Introduction to TensorFlow and its Relevance in Object Detection

TensorFlow is an open-source machine learning framework developed by Google Brain Team. Its name is derived from the operations which neural networks perform on multidimensional data arrays, known as tensors. Since its initial release in 2015, TensorFlow has become one of the most popular and widely used machine learning libraries.

Several reasons make TensorFlow a go-to framework for object detection:

- **Versatility and Scalability:** TensorFlow can run on multiple CPUs, GPUs, and even mobile devices. Its distributed computing capabilities make it perfect for large-scale projects, like training an object detection model on vast datasets.
- **High-Level APIs:** TensorFlow provides high-level APIs such as Keras, making it easier for developers to create neural networks without delving too deep into the underlying mathematics.
- **Rich Ecosystem:** TensorFlow's ecosystem is filled with numerous tools and extensions, designed explicitly for tasks like object detection, image recognition, and more. This rich ecosystem allows developers to leverage pre-trained models, fine-tune them, or start training from scratch.
- **Visualization with TensorBoard:** One of the standout features of TensorFlow is TensorBoard, a visualization tool that allows researchers and developers to monitor training processes, examine learned embeddings, and more.

3.6.2 Understanding the TensorFlow Object Detection Framework

The TensorFlow Object Detection Framework is a collection of tools and libraries designed to aid in the development, training, and deployment of object detection models. Given the complexity of object detection tasks, this framework provides a structured approach, making it more approachable and efficient.

Core Components of the Framework

- **Model Builders:** Streamlines the process of constructing different detection models and defining the computations needed for each step.
- **Input/Output Pipelines:** These manage data input, preprocessing, and output. The input pipelines handle tasks such as reading data from disk, resizing and augmenting images, etc.
- **Loss Definitions:** Given that the objective of training is to minimize a loss function, the framework provides several loss definitions tailored for object detection.
- **Post-Processing Operations:** After a model predicts object locations and classes, these predictions need post-processing to be usable. This includes tasks such as non-maximum suppression to eliminate redundant bounding box predictions.
- **Evaluation Metrics:** It provides tools for assessing the performance of an object detection model, such as mAP (mean average precision).

Configuring the Framework for Custom Object Detection

1. **Dataset Preparation:** Gather images and annotate them with bounding boxes and labels.
2. **Data Conversion:** Convert the dataset into the TFRecord format, which is optimized for TensorFlow.
3. **Model Configuration:** Choose a pre-trained model from the TensorFlow 1 Detection Model Zoo as a starting point and configure its hyperparameters for the specific task.
4. **Training and Evaluation:** Use the framework's tools to train the model on custom dataset, and then evaluate its performance using the provided metrics.

TensorFlow Detection Model Zoo (ResNet-101: Architecture and Significance)

The TensorFlow 1 Detection Model Zoo is a collection of pre-trained models specifically developed for object detection tasks. These models, curated by TensorFlow developers and the community, represent a range of architectures optimized for different use-cases, from lightweight models suitable for edge devices to more robust ones designed for accuracy on high-performance computing setups.

Among the models available in the Model Zoo, ResNet-101 stands out due to its deep architecture and capability to capture intricate patterns. In the context of object detection, variants of ResNet-101, like rfcn_resnet101_coco, have been pre-trained on the COCO dataset, making them ideal for tasks demanding high accuracy. As mentioned before the core idea behind ResNet is the use of "skip connections" or "residual connections" that bypass one or more layers. This innovative approach addressed the vanishing gradient problem prevalent in deeper neural networks, allowing the successful training of networks with hundreds (or even thousands) of layers.

Key Features of ResNet-101

1. **Depth:** As the name suggests, ResNet-101 consists of 101 layers. This depth allows the model to capture intricate patterns and hierarchies in the input data.

2. **Residual Blocks:** The building blocks of ResNet. Each block typically consists of three convolutional layers (1x1, 3x3, and 1x1 convolutions) with Batch Normalization and ReLU activation. The residual connection skips these layers and adds the input directly to the output, enhancing the gradient flow.

3. **Global Average Pooling:** Instead of using fully connected layers at the end, ResNets often use global average pooling to reduce spatial dimensions before the final classification layer. This reduces the total number of parameters and mitigates overfitting.

4. **Bottleneck Design:** To reduce the number of parameters and computations, ResNet-101 employs a "bottleneck" design, especially in its deeper configurations.

In object detection tasks, ResNet-101 is often used as a backbone or feature extractor. The deep architecture ensures a rich representation of input images, capturing both low-level features (like edges) and high-level semantics (like object parts). When combined with frameworks like Faster R-CNN or R-FCN, ResNet-101 provides the foundation upon which region proposals are made and subsequently classified.

The rfcn_resnet101_coco: Specifications and Performance Metrics

The 'rfcn_resnet101_coco' model is a fusion of the ResNet-101 architecture with the R-FCN (Region-based Fully Convolutional Network) object detection mechanism, pre-trained on the COCO dataset. This combination allows for both deep feature extraction and efficient object detection, making it a popular choice for tasks demanding precision.

R-FCN stands for Region-based Fully Convolutional Network. Unlike traditional object detection mechanisms that might rely on dedicated layers for predicting bounding boxes and class scores, R-FCN operates in a fully convolutional fashion. It utilizes position-sensitive score maps to produce detections, ensuring both high accuracy and computational efficiency.

Key Specifications of rfcn_resnet101_coco

1. **Backbone:** The primary feature extractor is ResNet-101, which captures rich hierarchies of features from input images.
2. **Position-Sensitive Score Maps:** The model uses these maps to generate object scores and bounding box regressions for each region of interest, allowing for efficient and accurate object detection.
3. **Training Dataset:** Pre-trained on the COCO dataset, this model is well-versed in detecting a diverse range of object classes, making it a versatile choice for various applications.
4. **Fully Convolutional Design:** This design ensures that the model can process images of varying sizes, allowing for flexibility during inference.

3.6.3 Training CNN model for web elements detection

Dataset Gathering and Annotation

For a task like generating HTML5 and CSS3 code from website images, a representative dataset is crucial. This dataset should encompass a variety of web design elements, layouts, and styles.

1. **Training Set:** A majority (e.g., 70%) of the dataset is allocated here. This portion will be used to train the model.
2. **Validation Set:** Approximately 15% of the data can serve as the validation set. It is used to tune hyperparameters and prevent overfitting by providing feedback during training.
3. **Test Set:** The remaining 15% should be reserved as the test set to evaluate the model's performance on unseen data.

Creating TFRecords

TensorFlow uses a special binary file format called TFRecord for training. These files store data in a serialized format, allowing for efficient storage and improved I/O.

Conversion Process:
The code developed for create TFRecords (B.1) is designed to transform a dataset into the TFRecords format, which is an established format employed by TensorFlow. This transformation is particularly focused on preparing data for object detection training on custom datasets.

Label Maps in TensorFlow

Label maps, in the context of TensorFlow and object detection tasks, serve as a bridge between the numeric labels used during model training and the human-readable class names. In essence, they provide a mapping between the numeric IDs that a model might predict and the actual string representation of the class.

For object detection tasks, TensorFlow expects label maps to be defined in the Protocol Buffer (protobuf) format. Define Each Item: For every class in the custom dataset, define an item in the label map. Each item will have:

- **id:** An integer representing the class (starting from 1).
- **name:** A string that is the human-readable name of the class.

Example:

```
item {
  id: 1
  name: 'header'
}

item {
  id: 2
  name: 'footer'
}
```

Initializing with Pre-trained Weights

Leveraging pre-trained weights is a cornerstone of efficient model training, especially when dealing with large and complex architectures.

Advantages:

- **Performance:** Starting with a model that is already trained on a vast dataset (like ImageNet) can lead to better generalization, especially with limited data.
- **Speed:** Converge faster, saving time and computational resources.
- **Avoid Overfitting:** With an established base, it is less likely the model will overfit on a smaller dataset.

Checkpoint Files:
TensorFlow uses .ckpt files to store weights. When initializing training, you can specify the path to these checkpoint files to begin training from that point.

Running the Training Script

Once the TFRecords and label map files were created, also is going to be necessary Choose a Pre-Trained Model, for this thesis it was selected "rfcn_resnet101_coco.config" (B.2).

To train the model, execute the object detection runner script that was created (B.3). In the object detection runner script, a TensorFlow-based pipeline is established for the purpose of running inference on a pre-trained object detection model. Initially, the script imports necessary Python libraries, including TensorFlow, NumPy, Pandas, and utilities from the TensorFlow Object Detection API. The system path is appended to include the parent directory, which allows the script to access modules from the object_detection folder. A TensorFlow session is configured to manage GPU memory allocation, ensuring efficient memory usage during inference.

The script loads the frozen TensorFlow model into memory and reads the label map, which associates class IDs with class names. It defines utility functions such as load_image_into_numpy_array for image processing, _flatten for list flattening, and _get_images for retrieving image file paths from the specified directory. The core functionality is encapsulated in the run_inference_for_single_image function, which executes the model's inference operation on an image input, provided as a NumPy array. This function processes input tensors, retrieves the necessary output tensors (detections, scores, classes, boxes, and optionally masks), and runs the TensorFlow session to obtain the inference results.

The inference loop iterates over each image in the test directory, loading images, expanding their dimensions to match the model's input requirements, and running inference through the defined function. After obtaining the detection results, the script uses the visualization_utils from the Object Detection API to draw bounding boxes and labels on the images, indicating detected objects. The visualization applies a minimum score threshold to display only confident detections.

Each processed image is then plotted with matplotlib's pyplot module, showcasing the detections, and saved to a designated results directory. The image is also displayed inline if the script is executed in an IPython environment or a Jupyter notebook. The script concludes by outputting the path of the saved visualized image, providing a reference to the processed results.

This runner script is designed to be a streamlined and user-friendly interface for applying TensorFlow's Object Detection capabilities to a given set of images, simplifying the process of visualizing model predictions.

The script is typically invoked as follows:

```
python model_main_tf2.py \
    --model_dir=/path/to/model_directory   \
    --pipeline_config_path=/path/to/pipeline.config
```

Monitor the training process through TensorBoard:

```
tensorboard --logdir=/path/to/model_directory
```

Once training is complete, export the trained model:

```
python exporter_main_v2.py
    --input_type image_tensor \
    --pipeline_config_path /path/to/your/model/pipeline.config \
    --trained_checkpoint_dir /path/to/your/model/checkpoint \
    --output_directory /path/to/exported_model_directory
```

This script exports the model in a format that can be used for inference.

3.7 Metrics for Evaluating Object Detection Models

When developing object detection models, evaluating their performance is crucial. Unlike classification tasks, object detection requires evaluating both the model's localization accuracy (how well it identifies object locations) and its classification accuracy (how well it identifies the object's class). The section below provides an overview of the key metrics employed in evaluating object detection models.

3.7.1 Intersection over Union (IoU)

Intersection over Union (IoU) is one of the fundamental metrics in object detection for assessing the accuracy of an object detector on a specific dataset. This metric, while simple in its construction, plays a pivotal role in the way we measure and benchmark the efficacy of detection algorithms.

IoU is defined as the ratio of the area of the intersection to the area of the union of two bounding boxes:

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}} \tag{3.3}$$

Where:

- **Area of Intersection:** The overlapping region between the ground-truth bounding box and the predicted bounding box.
- **Area of Union:** The combined area covered by both the ground-truth and predicted bounding boxes, without double-counting the intersection.

Imagine two bounding boxes:

- **Box A (Ground Truth):** The true location of the object, annotated by a human.
- **Box B (Prediction):** The predicted location of the object by a detection algorithm.

The overlapping area between Box A and Box B is the intersection. Everything encompassed by Box A and Box B combined, minus this overlapping area, constitutes the union. The value of IoU ranges between 0 and 1:

- **IoU = 0:** No overlap between the predicted and ground-truth boxes.
- **IoU = 1:** Perfect overlap, meaning the predicted box fits the ground-truth box precisely.

Values between 0 and 1 indicate partial overlap. For instance, an IoU value of 0.7 suggests a substantial overlap but not a perfect match. In object detection tasks, an IoU threshold is set to determine whether a prediction is valid:

- **Common Threshold:** An IoU value greater than 0.5 is often considered a "good" detection. If the IoU exceeds this threshold, the predicted box is typically deemed to have successfully detected the object (Figure: 3.12).

- **Varied Thresholds:** Depending on the application, this threshold can be adjusted. For precision-sensitive tasks, such as medical imaging, a higher threshold like 0.75 might be used.

Figure 3.12: Intersection over Union Representation.

Limitations and Considerations

- **Uniform Threshold:** While a standard threshold (e.g., 0.5) simplifies evaluations, it might not always be ideal for every scenario. Some applications may demand tighter or looser overlaps for a detection to be considered correct.

- **Non-maximum Suppression (NMS):** In object detection, models often produce multiple boxes for a single object. NMS is a post-processing technique that picks the best box based on IoU and confidence scores. It retains the box with the highest confidence and suppresses others with IoU values exceeding a certain threshold.

- **IoU vs. GIoU (Generalized IoU):** Traditional IoU does not consider the scenario where two boxes have no overlap. In such cases, GIoU is a modification that considers the smallest enclosing box containing both boxes and adjusts the metric to account for this "missed" area.

3.7.2 Precision and Recall

Precision and recall are two foundational metrics in the world of machine learning and statistics, often used to evaluate the performance of classifiers, especially in situations where classes are imbalanced.

Definitions

- **Precision:** (also known as the positive predictive value) quantifies the number of correct positive predictions made by the classifier relative to the total number of positives it claims:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{3.4}$$

- **Recall:** (also known as sensitivity, hit rate, or true positive rate) calculates the number of correct positive predictions made by the classifier relative to the actual number of positives in the dataset:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{3.5}$$

Consider a dataset where objects are categorized as either "Cat" or "Not Cat":

- **True Positives (TP):** Images correctly predicted as "Cat."
- **False Positives (FP):** Images incorrectly predicted as "Cat."
- **True Negatives (TN):** Images correctly predicted as "Not Cat."
- **False Negatives (FN):** Images incorrectly predicted as "Not Cat."

Precision examines the proportion of actual cats in the predicted "Cat" group, while recall measures the proportion of predicted cats in the actual "Cat" group.

False Positive Rate (FPR) and False Negative Rate (FNR)

The classification problems in machine learning requires a granular understanding of different error types. Two such critical error metrics are the False Positive Rate (FPR) and the False Negative Rate (FNR). They are especially important when discerning a model's performance in situations where specific types of classification errors may have more severe consequences than others.

- **False Positive Rate (FPR):**

$$\text{FPR} = \frac{FP}{FP + TN} \tag{3.6}$$

  Where:

  - TN is the number of True Negatives.

  FPR calculates the proportion of negative instances that are incorrectly classified as positive.

- **False Negative Rate (FNR):**

$$\text{FNR} = \frac{FN}{FN + TP} \tag{3.7}$$

  Where:

  - TP is the number of True Positives.

  FNR computes the proportion of positive instances that are mistakenly classified as negative.

Interpretation and Implications:

- **Higher FPR:** Suggests that a significant proportion of negative instances are incorrectly classified. It might indicate an overly aggressive model that predicts positives too frequently.
- **Higher FNR:** Indicates that a large fraction of actual positives are missed by the model. This might suggest a conservative model that hesitates to predict positives.

Trade-off Between Precision and Recall

In many situations, there is an inverse relationship between precision and recall:

- **Increasing Precision:** This often reduces recall as the model becomes more conservative, making fewer positive predictions to avoid making mistakes.
- **Increasing Recall:** This often reduces precision as the model tries to capture as many positives as possible, risking more false positive errors in the process.

3.7.3 Average Precision (AP) and mean Average Precision (mAP)

Average Precision (AP) and mean Average Precision (mAP) are crucial metrics in many domains, notably in object detection and information retrieval. They provide an aggregated measure of a model's performance across various levels of precision and recall.

Average Precision (AP)

Average Precision (AP) is a critical metric in object detection. While Precision and Recall offer insights at a particular decision threshold, AP provides a summarized measure of the model's performance across all possible thresholds, offering a more holistic view.

Average Precision (AP) essentially computes the area under the Precision-Recall curve, providing a single-figure summary of the curve and, by extension, the classifier's performance over all thresholds.

$$AP = \sum_{n} (R_n - R_{n-1}) P_n \tag{3.8}$$

Where:

- P_n is the precision at the n^{th} threshold.
- R_n is the recall at the n^{th} threshold.

Interpretation of AP Values

- **AP = 1.0:** Indicates perfect precision and recall across all thresholds, an ideal scenario.
- **AP = 0.0:** Indicates that the model has failed to correctly predict the positive class across all thresholds.

Values between 0 and 1 give a sense of the model's performance, with higher values indicating better alignment with the ground truth.

mean Average Precision (mAP)

mean Average Precision (mAP) is a metric, particularly within the domains of object detection. It serves as a single-figure measure, encapsulating a model's performance across multiple classes or over various Intersection over Union (IoU) thresholds.

The concept of mAP builds upon Average Precision (AP). While AP gives a summarized measure of model performance for a single class across all possible decision thresholds, mAP extends this idea to multiple classes:

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{3.9}$$

Where:

- N is the total number of classes.
- AP_i is the average precision for the i^{th} class.

mAP in Object Detection:
mAP is especially prevalent in object detection, where models predict bounding boxes around objects of interest:

- For each predicted bounding box, an IoU (Intersection over Union) value is computed with respect to the ground truth boxes.
- Predictions are typically considered correct if the IoU surpasses a specific threshold (often 0.5). However, to capture performance nuances at various levels of overlap, mAP is often computed across a range of IoU thresholds, e.g., 0.5 to 0.95 in steps of 0.05.
- The final mAP score represents the mean AP calculated over all these IoU thresholds, providing a holistic view of the detector's performance.

Benefits of Using mAP
mAP is especially prevalent in object detection, where models predict bounding boxes around objects of interest:

- **Holistic View:** By averaging AP over multiple classes or IoU thresholds, mAP offers a broader perspective on a model's strengths and weaknesses.
- **Standardized Benchmarking:** mAP provides a standardized metric that allows for objective comparison between different models on shared datasets, making it popular in challenges and competitions.

3.7.4 F1 Score

The F1 Score emerges as a metric, especially when dealing with datasets exhibiting class imbalance. Providing a balance between precision and recall, the F1 Score encapsulates the harmonic trade-off between these two metrics, offering a more holistic measure of a model's accuracy.

Unlike the arithmetic mean, which can be skewed by extreme values, the harmonic mean tends to be closer to the smaller value between precision and recall, ensuring that both metrics are given equal importance.

$$F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{3.10}$$

Interpretation

- **F1 Score = 1:** Indicates perfect precision and recall. An ideal yet rarely achieved scenario.
- **F1 Score = 0:** Reflects that either the precision or the recall is zero, signaling a completely ineffective model.

3.7.5 Selecting the Optimal Evaluation Metric: A Justification for mAP in Web Element Object Detection Models

For object detection tasks, particularly with respect to web elements where accurate localization is as important as classification, mean Average Precision (mAP) is generally the better evaluation metric. Here are justifications for selecting mAP as an evaluation metric for object detection of web elements:

1. **Localization and Classification:** mAP accounts for both the classification accuracy and the spatial precision of the bounding boxes. It considers the Intersection over Union (IoU) between the predicted bounding boxes and the ground truth, which is crucial for object detection where the precise location of web elements is important.
2. **Multiple Thresholds:** mAP evaluates the model's performance across various IoU thresholds. This provides a more comprehensive assessment since web elements may vary greatly in size and shape, and a single threshold might not be representative of overall performance.
3. **Handling Multiple Classes:** Websites consist of multiple types of elements like buttons, text fields, images, etc. mAP evaluates the average precision for each class and then computes the mean of these values, providing a balanced metric that fairly evaluates all classes, irrespective of their frequency.
4. **Standard in Research:** mAP is a standard metric in object detection and is used in leading competitions and benchmarks like the Pascal VOC challenge, COCO detection challenge, etc. Using mAP allows your work to be directly comparable to state-of-the-art methods and standards in the field.
5. **Robust to Imbalance:** Web pages may have an imbalanced distribution of elements (e.g., many more text blocks than images). mAP is robust to such imbalances because it computes the average precision per class before taking the mean across classes.
6. **Comprehensive Performance Evaluation:** Because mAP integrates over the entire precision-recall curve, it provides a single figure of merit that reflects the model's ability to detect objects at different levels of confidence, which is beneficial when tuning the trade-off between precision and recall.

3.8 Detection Outcomes for Web Elements Model

In evaluating the performance of the object detection model for web elements, the Mean Average Precision (mAP) serves as the primary metric. Average Precision (AP) computes the average precision value for recall value over the interval [0, 1], offering a single-figure measure of quality across recall levels. It consolidates the precision-recall curve into a single value representing the area under the curve, effectively balancing both precision (the proportion of true positives among the detections) and recall (the proportion of actual positives correctly identified by the model).

The mAP extends this concept across multiple classes or categories, calculating the mean AP over all classes. It encapsulates the overall effectiveness of the model across these varied elements, providing a comprehensive picture of its detection capabilities. The selection of mAP as a benchmark metric is predicated on its widespread acceptance and use within the domain of object detection algorithms. Given its comprehensive nature, it facilitates a detailed evaluation of the model's performance.

The next tables delineates the performance results obtained from the object detection model under study. The following exposition details the empirical findings, tabulated and analyzed in the context of mean Average Precision (mAP) scores—a metric that quantifies the balance between precision and recall in the detection process. The data presented is the culmination of a series of tests designed to evaluate the model's ability to accurately identify and locate web elements.

Evaluative Outcomes of Web Element Recognition via Object Detection Model Using Mean Average Precision (mAP) Scores (Table: 3.1):

Table 3.1: Detection Outcomes for Web Elements Using mAP Scores

Web Element Label	mAP Score
Header	0.982421
Footer	0.915664
Navigation Bar	0.749839
Button	0.912689
Rounded Button	0.959868
Circle Button	0.815862
Square Button	0.722323
Arrow Icon	0.388160
Search Bar	0.717337
Input Form	0.837717
Form	0.856018

Evaluative Outcomes for Segmented Images of Web Element Recognition via Object Detection Model Using Mean Average Precision (mAP) Scores (Table: 3.2):

Table 3.2: Detection Outcomes for Web Elements in Segmented Images Using mAP Scores

Web Element Label	mAP Score
Horizontal div element (hdiv)	0.656950
Vertical div element (vdiv)	0.716463
Section	0.919900

The images presented below (Figures: 3.13, 3.14, 3.15, 3.16, 3.17 and 3.18) depict the outcomes obtained from the object detection model. These visual representations illustrate the model's performance in recognizing and identifying various web elements, offering a clear and empirical demonstration of its capabilities.

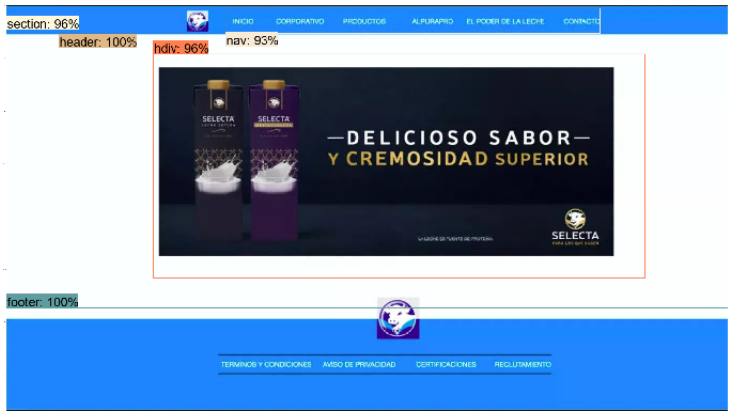


Figure 3.13: Example 1. Evaluation of the Object Detection Model's Performance in Localizing Web Elements.

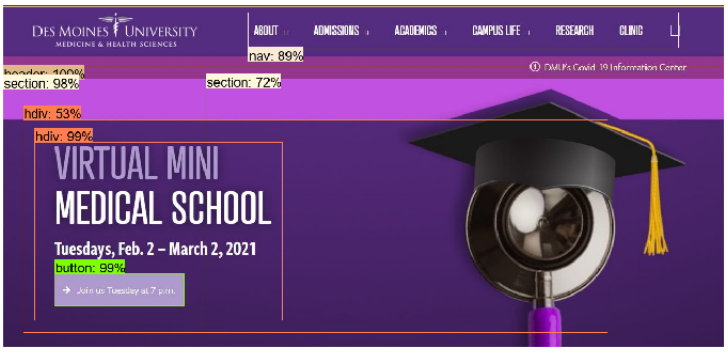


Figure 3.14: Example 2. Evaluation of the Object Detection Model's Performance in Localizing Web Elements.

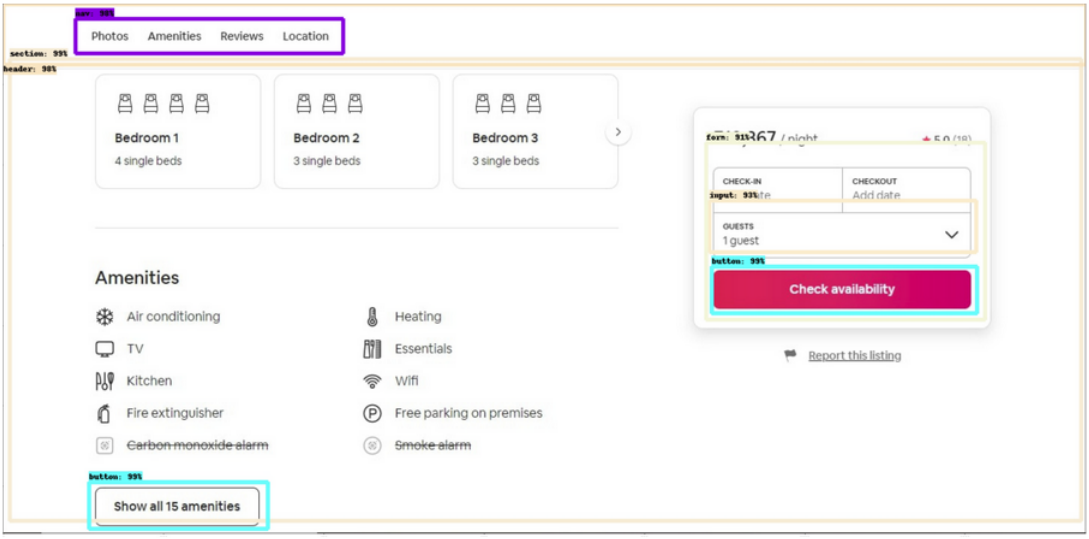


Figure 3.15: Example 3. Evaluation of the Object Detection Model's Performance in Localizing Web Elements.

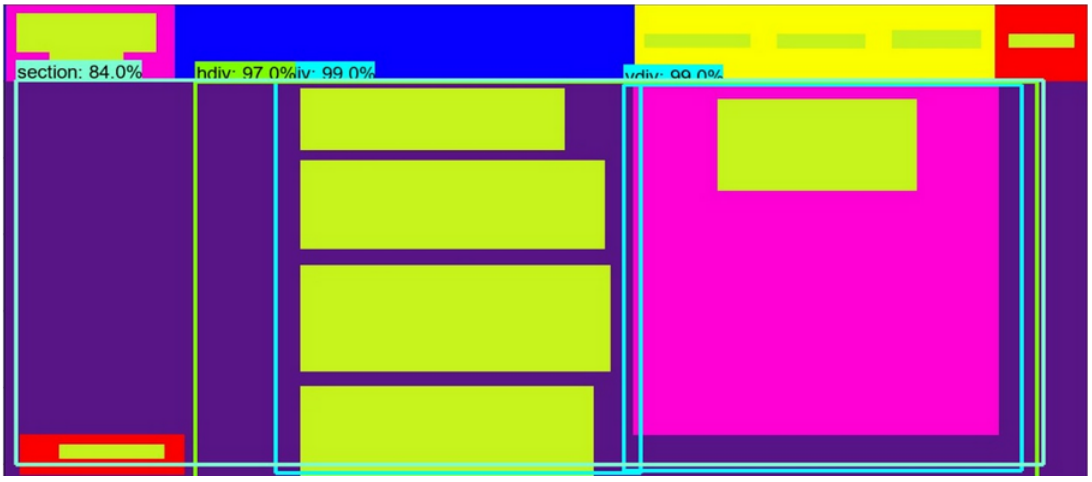


Figure 3.16: Example 4. Evaluation of the Object Detection Model's Performance in Localizing Web Elements.

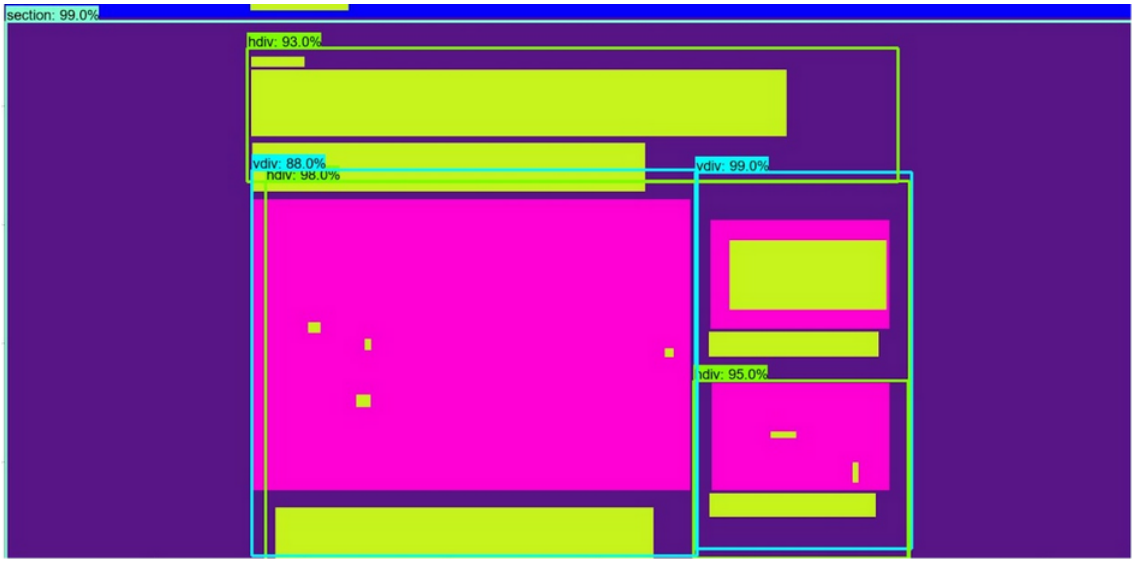


Figure 3.17: Example 5. Evaluation of the Object Detection Model's Performance in Localizing Web Elements.

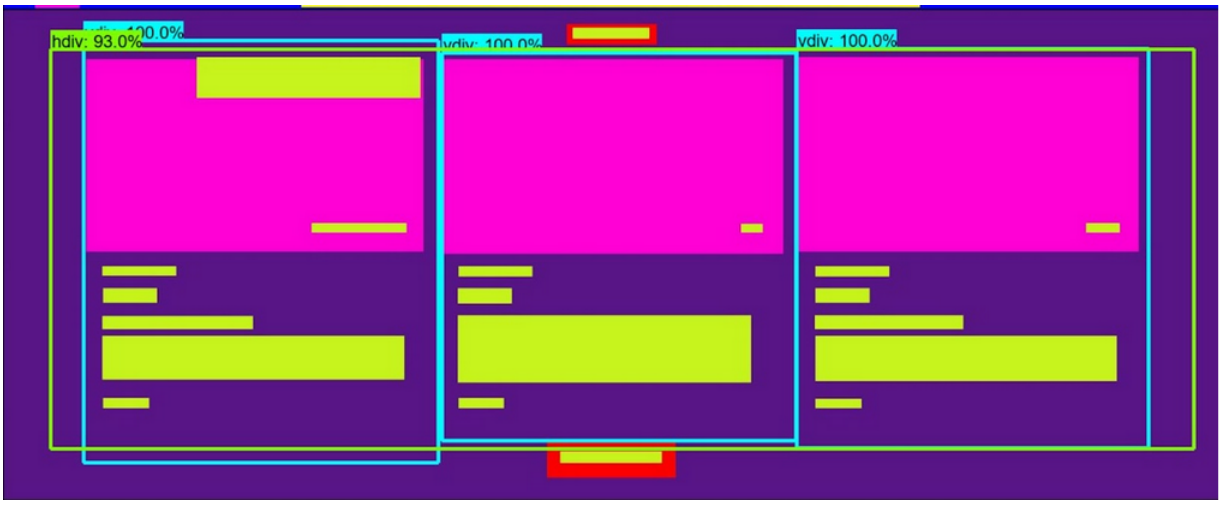


Figure 3.18: Example 6. Evaluation of the Object Detection Model's Performance in Localizing Web Elements.

AI-powered code generation has the potential to revolutionize software development, making it faster, more efficient, and more accurate.

Software Testing Lead.

4

Proposed Model for Automated Web Code Generation

4.1 Introduction

This chapter presents an in-depth examination of the methodological framework designed to transform webpage design images into structured HTML5 and CSS3 code. The proposed model is a composite of crafted steps aimed at automating the transformation process. The system is designed to recognize and convert visual elements of web page designs directly into the equivalent code structures, ensuring both semantic accuracy and stylistic fidelity. The chapter will explain the framework process to systematically decodes the elements of design and reconstructs them in a code format that adheres to web standards.

4.1.1 Automated Code Generation: System Entry

The initiation of the automated transformation process begins with a webpage design image (Figure: 4.1). This image serves as the input to the system and is a visual representation of the desired final webpage. It encapsulates all graphical elements, layout specifications, and design features intended for the final product.

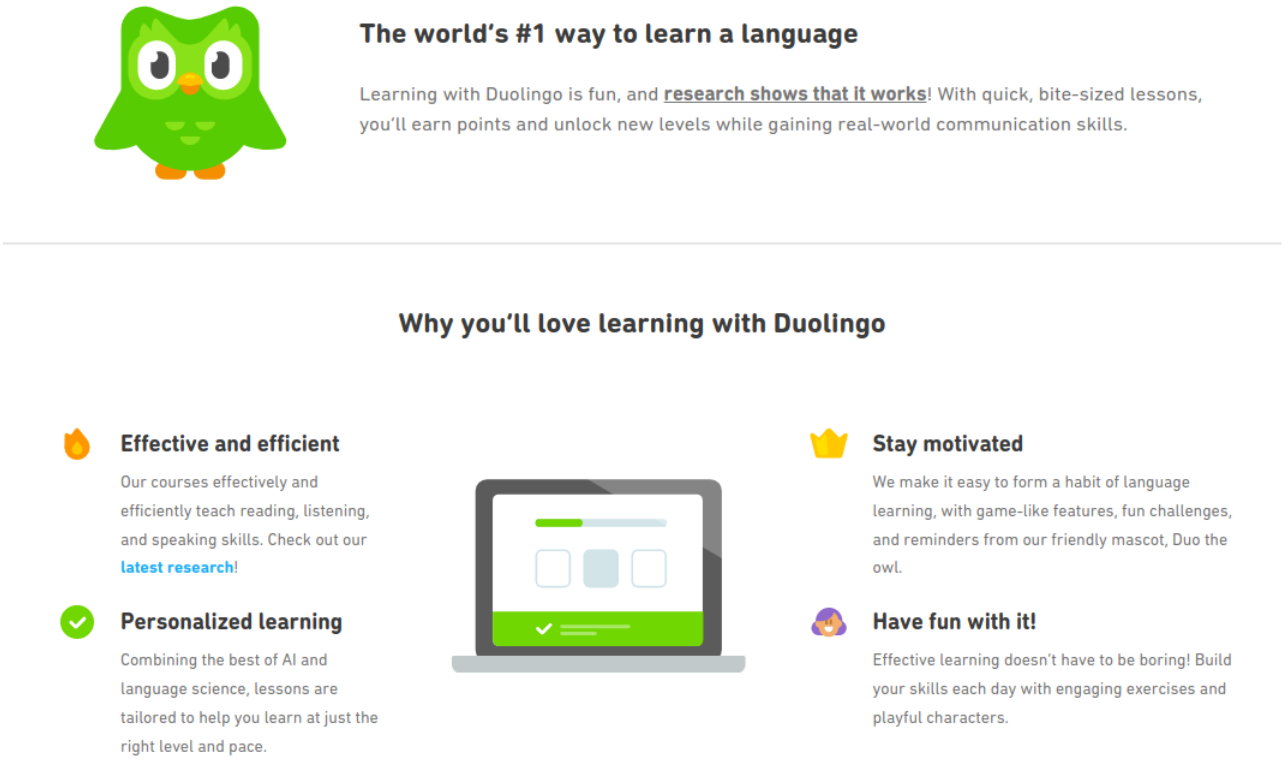


Figure 4.1: Webpage design image. Duolingo website image design as input to the system.

4.1.2 Results for Automated Code Generation System:

The outcome of the code generation process is the creation of HTML5 and CSS3 code. This code represents the system's output, encapsulating the visual elements and layout defined by the original design image. For visual representation, the generated code is rendered into an image, allowing the evaluation of how well the system translates the initial design into functional web page.

HTML5 code generated by the system:

```
<!DOCTYPE html>
<html lang="en">
<!-- This is the <head> tag, where all meta data is defined. -->
    <head>
        <meta charset="utf-8"/>
        <meta content="width=device-width, initial-scale=1, shrink-to-fit=no" name="viewport"/>
        <link href="normalize.css" rel="stylesheet"/>
        <link href="styles_22.css" rel="stylesheet"/>
        <title>22</title>
    </head>
<!-- This is the <body> tag, where the whole webpage is allocated. -->
    <body>
<!-- This is the <section> tag, corresponding to "section_2". -->
        <section id="section_2">
```

```
            <div id="div_10" row="True">
                <div id="div_12" row="True">
                    <div css_pattern="div_12" id="div-image0">
                        <img class="div-image0__img__image0" css_pattern="div-image0" src="images/upload_test22duolingoimage1.png"/>
                    </div>
                </div>
                <div css_pattern="div_10" id="div_10-col_1" row="False">
                    <h1 class="div_10-col_1__h1__text0" css_pattern="div_10-col_1">The world's #1 way to learn a language</h1>
                    <p class="div_10-col_1__p__text1" css_pattern="div_10-col_1">
                        Learning with Duolingo is fun, and research shows that it works! With quick,
                        bite-sized lessons, you'll earn points and unlock new levels while gaining
                        real-world communication skills.
                    </p>
                </div>
            </div>
        </section>
<!-- This is the <section> tag, corresponding to "section_3". -->
        <section id="section_3">
            <div css_pattern="section_3" id="div-section_3" row="True">
                <h2 class="div-section_3__h2__text10" css_pattern="div-section_3">Why you'll love learning with Duolingo</h2>
            </div>
            <div id="div_13" row="True">
                <div id="div_11" row="True">
                    <div id="div_7" row="False">
                        <div css_pattern="div_7" id="div_7-row_0" row="True">
                            <div css_pattern="div_7-row_0" id="div-image1">
                                <img class="div-image1__img__image1" css_pattern="div-image1" src="images/upload_test22duolingoimage5.png"/>
                            </div>
                            <h2 class="div_7-row_0__h2__text2" css_pattern="div_7-row_0">Effective and efficient</h2>
                        </div>
                        <h2 class="div_7__h2__text3" css_pattern="div_7">Our courses effectively and</h2>
                        <p class="div_7__p__text4" css_pattern="div_7">
                            efficiently teach reading, listening, and speaking skills. Check out our latest research!
                        </p>
                        <div css_pattern="div_7" id="div_7-row_3" row="True">
                            <div css_pattern="div_7-row_3" id="div-image2">
                                <img class="div-image2__img__image2" css_pattern="div-image2" src="images/upload_test22duolingoimage3.png"/>
                            </div>
                            <h2 class="div_7-row_3__h2__text5" css_pattern="div_7-row_3">Personalized learning</h2>
                        </div>
                        <h2 class="div_7__h2__text6" css_pattern="div_7">Combining the best of Al anc</h2>
                        <h2 class="div_7__h2__text7" css_pattern="div_7">language science, lessons are</h2>
                        <h2 class="div_7__h2__text8" css_pattern="div_7">tailored to help you learn at just the</h2>
                        <h2 class="div_7__h2__text9" css_pattern="div_7">right level and pace.</h2>
                    </div>
                    <div id="div_8" row="True">
                        <div css_pattern="div_8" id="div-image4">
                            <img class="div-image4__img__image4" css_pattern="div-image4" src="images/upload_test22duolingoimage2.png"/>
                        </div>
                    </div>
                </div>
                <div id="div_9" row="False">
                    <div css_pattern="div_9" id="div_9-row_0" row="True">
                        <div css_pattern="div_9-row_0" id="div-image3">
                            <img class="div-image3__img__image3" css_pattern="div-image3" src="images/upload_test22duolingoimage4.png"/>
                        </div>
                        <h2 class="div_9-row_0__h2__text11" css_pattern="div_9-row_0">Stay motivated</h2>
                    </div>
                    <p class="div_9__p__text12" css_pattern="div_9">
                        We make it easy to form a habit of language learning,
                        with game-like features, fun challenges, and reminders from
                        our friendly mascot, Duo the owl.
                    </p>
                    <div css_pattern="div_9" id="div_9-row_2" row="True">
                        <div css_pattern="div_9-row_2" id="div-image5">
                            <img class="div-image5__img__image5" css_pattern="div-image5" src="images/upload_test22duolingoimage6.png"/>
                        </div>
                        <h2 class="div_9-row_2__h2__text13" css_pattern="div_9-row_2">Have fun with it!</h2>
                    </div>
                    <p class="div_9__p__text14" css_pattern="div_9">
                        Effective learning doesn't have to be boring! Build your skills each day with engaging exercises and
                    </p>
```

```
                <h2 class="div_9__h2__text15" css_pattern="div_9">playful characters.</h2>
            </div>
        </div>
    </section>
</body>
</html>
```

CSS3 CODE GENERATED BY THE SYSTEM:

```
/*******************************/
/* Beginning of common styles. */
/*******************************/
/* Beginning of styles for block body. */
/* Beginning of styles for block button. */
button {
    display: block;
}
/* Beginning of styles for block div-image0. */
#div-image0 {
    margin-right: 5px;
}
.div-image0__img__image0 {
    height: 164px;
    width: 185px;
}
/* Beginning of styles for block div-image1. */
.div-image1__img__image1 {
    height: 39px;
    width: 41px;
}
/* Beginning of styles for block div-image2. */
.div-image2__img__image2 {
    height: 44px;
    width: 40px;
}
/* Beginning of styles for block div-image3. */
.div-image3__img__image3 {
    height: 37px;
    width: 46px;
}
/* Beginning of styles for block div-image4. */
#div-image4 {
    margin-right: 5px;
}
.div-image4__img__image4 {
    height: 182px;
    width: 257px;
}
/* Beginning of styles for block div-image5. */
.div-image5__img__image5 {
    height: 40px;
    width: 51px;
}
/* Beginning of styles for block div-section_3. */
#div-section_3 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    width: 436px;
}
.div-section_3__h2__text10 {
    color: rgb(80, 75, 81);
    font-size: 22px;
    margin-bottom: 0em;
    margin-right: 5px;
}
/* Beginning of styles for block div_10. */
#div_10 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    width: 963px;
}
/* Beginning of styles for block div_10-col_1. */
#div_10-col_1 {
    align-items: flex-start;
    display: flex;
    flex-direction: column;
    justify-content: center;
    margin-right: 5px;
    text-align: left;
    width: 702px;
}
.div_10-col_1__h1__text0 {
    color: rgb(81, 76, 80);
    font-size: 24px;
    margin-bottom: 0em;
}
.div_10-col_1__p__text1 {
    color: rgb(150, 144, 150);
    font-size: 18px;
    margin-bottom: 0em;
    margin-top: 32px;
}
/* Beginning of styles for block div_11. */
#div_11 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-left: 5px;
    width: 623px;
}
/* Beginning of styles for block div_12. */
#div_12 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-left: 3px;
    width: 215px;
}
/* Beginning of styles for block div_13. */
#div_13 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-top: 50px;
    width: 1025px;
}
/* Beginning of styles for block div_7. */
#div_7 {
    align-items: center;
    display: flex;
    flex-direction: column;
    justify-content: center;
```

```
    margin-left: 5px;
    text-align: center;
    width: 279px;
}
.div_7__h2__text3 {
    color: rgb(159, 150, 157);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 4px;
}
.div_7__h2__text6 {
    color: rgb(160, 152, 158);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 4px;
}
.div_7__h2__text7 {
    color: rgb(158, 151, 157);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 11px;
}
.div_7__h2__text8 {
    color: rgb(160, 152, 158);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 11px;
}
.div_7__h2__text9 {
    color: rgb(156, 149, 157);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 12px;
}
.div_7__p__text4 {
    color: rgb(136, 161, 183);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 12px;
}
/* Beginning of styles for block div_7-row_0. */
#div_7-row_0 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-right: 0px;
    width: 255px;
}
.div_7-row_0__h2__text2 {
    color: rgb(86, 79, 84);
    font-size: 17px;
    margin-bottom: 0em;
}
/* Beginning of styles for block div_7-row_3. */
#div_7-row_3 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-right: 0px;
    margin-top: 18px;
    width: 258px;
}
.div_7-row_3__h2__text5 {
    color: rgb(86, 80, 86);
    font-size: 20px;
    margin-bottom: 0em;
}
/* Beginning of styles for block div_8. */
#div_8 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-right: 5px;
    width: 287px;
}
/* Beginning of styles for block div_9. */
#div_9 {
    align-items: center;
    display: flex;
    flex-direction: column;
    justify-content: center;
    margin-right: 5px;
    text-align: center;
    width: 371px;
}
.div_9__h2__text15 {
    color: rgb(158, 149, 155);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 11px;
}
.div_9__p__text12 {
    color: rgb(161, 153, 159);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 6px;
}
.div_9__p__text14 {
    color: rgb(160, 151, 157);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 7px;
}
/* Beginning of styles for block div_9-row_0. */
#div_9-row_0 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-right: 0px;
    width: 200px;
}
.div_9-row_0__h2__text11 {
    color: rgb(87, 82, 87);
    font-size: 19px;
    margin-bottom: 0em;
}
/* Beginning of styles for block div_9-row_2. */
#div_9-row_2 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-right: 0px;
    margin-top: 20px;
    width: 217px;
}
.div_9-row_2__h2__text13 {
    color: rgb(85, 80, 88);
    font-size: 16px;
    margin-bottom: 0em;
}
/* Beginning of styles for block section_2. */
#section_2 {
    align-items: center;
    background-color: rgb(254, 254, 254);
    display: flex;
    justify-content: center;
    padding: 9px 0px;
    width: 100%;
}
```

```
/* Beginning of styles for block section_3. */
#section_3 {
    align-items: center;
    background-color: rgb(254, 254, 254);
    display: flex;
    flex-direction: column;
    justify-content: center;
    padding: 59px 0px;
    text-align: center;
    width: 100%;
}
/*****************************/
/* Beginning of media styles. */
/*****************************/
/* Beginning of styles for 480px width. */
@media(max-width: 480px) {
    #div-section_3 {
        width: 100%;
    }
    #div_12 {
        width: 100%;
    }
    #div_7 {
        width: 100%;
    }
    #div_7-row_0 {
        width: 100%;
    }
    #div_7-row_3 {
        width: 100%;
    }
    #div_8 {
        width: 100%;
    }
    #div_9 {
        width: 100%;
    }
    #div_9-row_0 {
        width: 100%;
    }
    #div_9-row_2 {
        width: 100%;
    }
}
/* Beginning of styles for 800px width. */
@media(max-width: 800px) {
    #div_10-col_1 {
        width: 100%;
    }
    #div_11 {
        width: 100%;
    }
}
/* Beginning of styles for 1200px width. */
@media(max-width: 1200px) {
    #div_10 {
        width: 100%;
    }
    #div_13 {
        width: 100%;
    }
}
```

4.1.3 Comparative Analysis: Original Webpage Design vs. Automatically Generated Code

Upon generating the HTML5 and CSS3 code, the system renders the code into an image (Figure: 4.2) to facilitate a comparative analysis with the original design input. This rendered image is a visual representation of the code's interpretation of the initial design. The comparison between the original webpage design image and the image rendered from the generated code is crucial in evaluating the effectiveness of the code generation process. It serves as a benchmark for assessing the fidelity of the conversion.

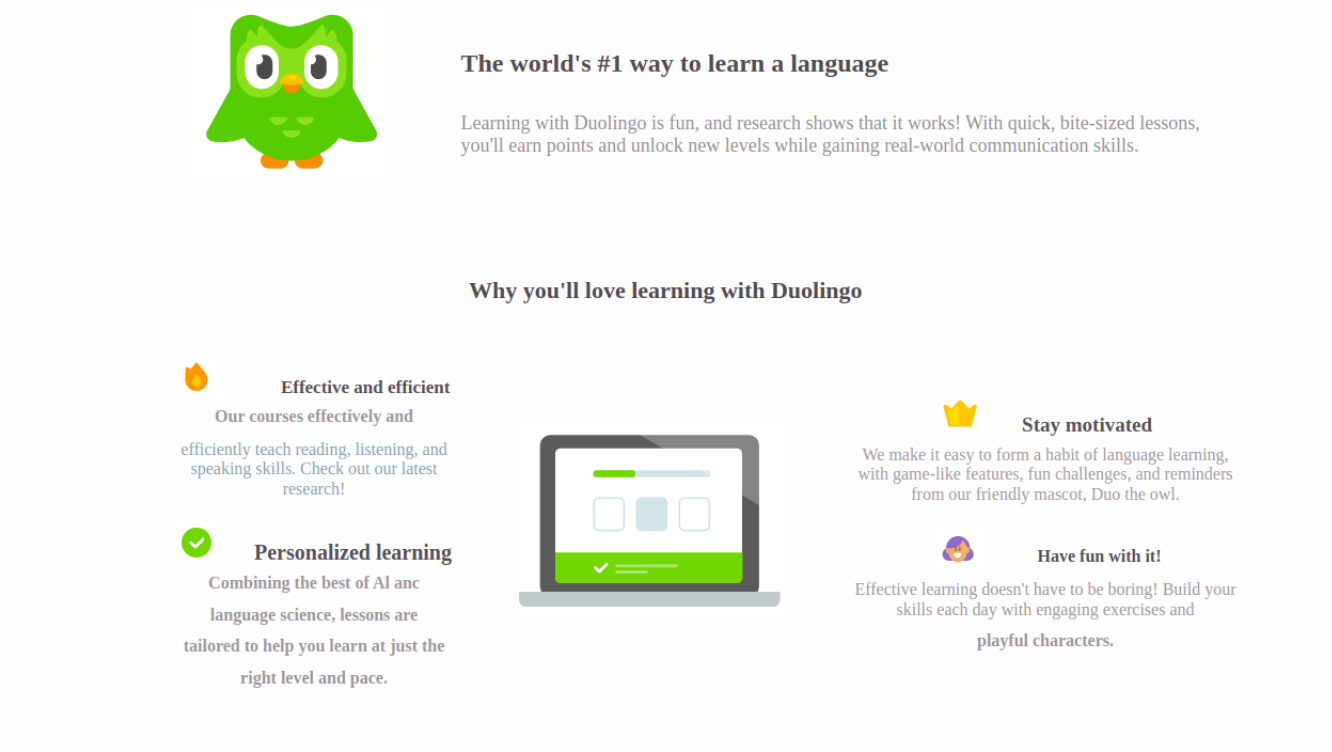


Figure 4.2: Webpage generated by the system. Result of the HTML5 and CSS3 code automated generated and rendered into an image.

The images presented side-by-side (Figure: 4.3) depict a comparative analysis between an original webpage design and the resulting webpage constructed from the generated HTML5 and CSS3 code by the automated system.

On the right side is the original design, which exhibits a clear and professional layout typical of modern web pages, with a balance of text, graphical elements, and other components.

On the left side, the image displays the output as rendered from the code produced by the automated system. This interpretation closely mirrors the original design, maintaining the structural layout and styling. The image, header text, and supporting sections are replicated, suggesting that the automated code generation has effectively interpreted and translated the visual design elements from the original image. The colors, alignment, and distribution of space appear to be consistent with the original intent of the design.

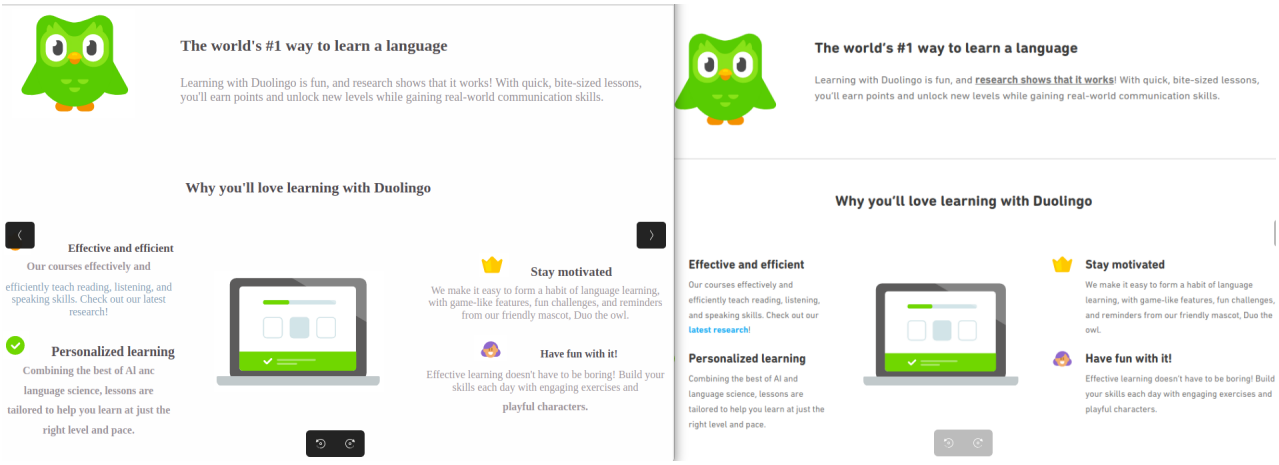


Figure 4.3: Original Design vs Code Generated. Comparison of Automated Web Code Generation Results: The left image shows the webpage rendered from the HTML5 and CSS3 code generated by the proposed system, while the right image displays the original webpage design. This side-by-side view demonstrates the system's ability to replicate design elements with high accuracy.

This juxtaposition illustrates the proficiency of the first version of the proposed model to generate automatic web code. The similarity in the rendered output denotes a successful initial step in automating web code generation, showing promise for future refinements and enhancements of the system.

4.2 Proposed Model For Automated Web Code Generation

The preceding chapter described the creation of a Convolutional Neural Network (CNN) model (Chapter: 3) adapted for the detection of web elements. This model serves as the foundational tool for the automated generation of web code from a given website image design. The elements that the CNN is trained to recognize and classify include: headers, footers, navigation bars (navs), buttons, search icons, input fields, forms, sections, and generic divisions (divs). Once the CNN model has been established and trained, it can be employed to run an object detection script. This script meticulously identifies and localizes the aforementioned web elements, capturing their properties and hierarchical relationships in a structured dictionary format.

To identify images that are inside of the webpage design, the method proceeds to incorporate computer vision techniques using the OpenCV library for image analysis. Specifically, it employs template matching strategies to recognize and catalog images embedded within the website's design. This step requires precise image manipulation, including resizing operations to match the scale of the images as they appear in the design.

Furthermore, the research exploits Optical Character Recognition (OCR) services provided by AWS and GCP to extract textual content from the design. This phase necessitates advanced pre-processing to filter out text that is not pertinent to the web code generation process, such as text within images or symbolic representations. The extracted text is then intelligently grouped according to its contextual relevance to various web elements like buttons, navigation bars, and footers.

Following the meticulous extraction and categorization process, the information is then transformed into node elements. These nodes are strategically placed within a graph tree structure. This structured representation enables more intuitive manipulation of parent and child relationships, essential for the accurate reconstruction of the web design in code form. To refine the quality of the generated web code and the tree structure, the model applies sophisticated inference algorithms. These algorithms are designed to rectify any inconsistencies and enhance the structure, thus optimizing the accuracy of the output.

Subsequently, the model initiates a style extraction phase. Using machine learning techniques, it discerns various stylistic attributes of the web elements, such as colors, margins, padding, dimensions, and positioning. This phase also involves the application of media queries to ensure the responsive adaptation of the layout across different screen sizes.

In the final step of the model, the tree structure is synthesized into XML code, which is then converted into HTML5 code. This code is annotated with essential attributes like classes and IDs, adhering to the Block, Element, Modifier (BEM) methodology. Concurrently, a JSON file is generated, encapsulating the stylistic attributes extracted earlier. This file serves as the blueprint for creating the corresponding CSS3 file, which is formatted to align with standard styling conventions.

In summary, the proposed model is comprised of an eight-fold subsystem architecture (Figure: 4.4), each designated to execute a specific role in the automated web code generation process:

1. The initial subsystem is tasked with deploying the Convolutional Neural Network model utilizing TensorFlow's object detection API. It systematically identifies various web elements within the design image, serving as the entry point for the subsequent data transformation process.

2. The second subsystem is responsible for transforming the raw output from the object detection model into a structured dictionary format, enriching it with valuable data points crucial for the ensuing steps.

3. The third subsystem employs computer vision techniques, specifically template matching, to accurately locate and identify pre-defined images within the website design. This process not only detects each image but also adjusts their dimensions and positioning to fit seamlessly into the webpage structure.

4. The fourth subsystem leverages Optical Character Recognition (OCR) capabilities provided by Google Cloud Platform (GCP) and Amazon Web Services (AWS) to extract textual content from the design. This subsystem ensures the conversion of visual text elements into digital format for incorporation into the web code.

5. In the fifth step, the gathered data, now augmented with text and images, is organized into a hierarchical tree structure. This allows for a systematic arrangement of web elements and their properties, constituting the core structure of the website's architectural design.

6. The sixth subsystem employs advanced inference algorithms to rectify and refine the placement of web elements. These algorithms enhance the tree structure by performing tasks such as aligning elements, sorting them within containers, accommodating orphan elements by creating new containers, and reordering siblings to ensure logical consistency.

7. The seventh subsystem is dedicated to the extraction of the styles properties. Utilizing techniques such as K-Means clustering, it discerns background colors, margins, positioning, and employs CSS flexbox principles to arrange elements effectively.

8. The final subsystem functions as the web code generator. It synthesizes the tree structure into XML code and simultaneously produces a JSON file detailing stylistic attributes. These files play a critical role in the building process of the HTML5 and CSS3 code generation.

This comprehensive, multi-subsystem approach ensures the automated generation of web code is both efficient and accurate, encapsulating the complexity of website design in a structured and machine-understandable format.

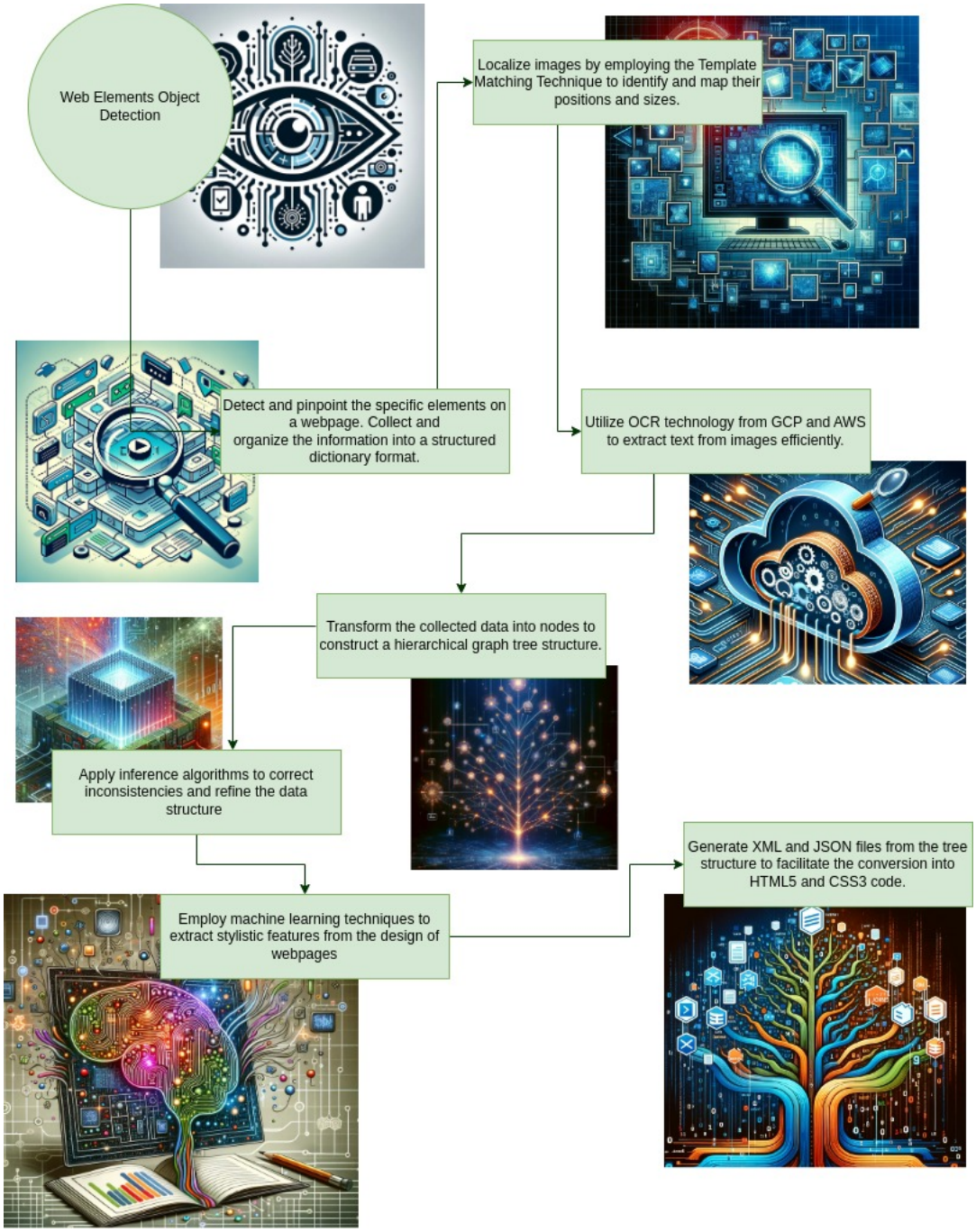


Figure 4.4: Proposed model flowchart. The proposed model is comprised of an eight-fold subsystem architecture.

In summary, this model orchestrates a sequence of advanced technological processes to bridge the gap between visual web design and its functional web code counterpart. Through a combination of machine learning, computer vision, and algorithmic inference, it aims to deliver an automated, accurate, and efficient code generation service that could contribute the field of web development.

4.3 State of art

The state of the art in automatic web code generation has been evolving with significant contributions from deep learning techniques. One of the notable advancements in this area is the SkCoder, which is a sketch-based code generation approach. SkCoder operates by mimicking the behavior of developers when they reuse code. It begins by retrieving a similar code snippet to a given natural language requirement, extracting the relevant parts as a "code sketch," and then editing this sketch to produce the final desired code. This approach allows the model to understand "how to write" by providing a pattern from the sketch, and "what to write" from the natural language requirements [82].

The effectiveness of SkCoder has been validated through extensive experiments. It has been tested on two public datasets and a new dataset called AixBench-L, which consists of 200k real natural language-code pairs. SkCoder has shown to outperform the state-of-the-art by a significant margin, improving upon other models by up to 120.1%. These experiments also included an decomposition study that highlighted the importance of each component in the SkCoder system [82].

Furthermore, the field has seen other significant developments, such as the introduction of Code Llama, a large language model specifically designed for coding tasks. This model is capable of generating code and natural language about code from both code and natural language prompts [83]. Another approach, LILO, focuses on learning interpretable libraries by compressing and documenting code, addressing the key aspect of refactoring in software development [84].

CodeGen is another open large language model for code that is contributed to the advancement of program synthesis. While the prevalence of large language models has advanced program synthesis, access to these models has been limited by training resources and data. CodeGen represents efforts to democratize access to these powerful models [85].

For code documentation, GPT-3's Codex has shown promise, achieving an overall BLEU score of 20.6 across six different programming languages, which is an 11.2% improvement over earlier state-of-the-art techniques [86].

In addition to the aforementioned models, the foundational work in the domain of automatic web code generation was discussed earlier in the document (Section: 1.6), highlighting systems such as Microsoft's Sketch2Code and Pix2Code. These systems have also made substantial contributions to the field and represent important research in the web code generation technologies.

4.4 Translating visual elements to code

4.4.1 Element Detection Subsystem

Chapter 3 provided a detailed explanation of the Convolutional Neural Network (CNN) model adapted for the detection of web elements. This chapter describe the practical application of this model. The primary function of the model is to identify various web elements within a webpage and extract crucial information about each detected element.

The data extracted for each element includes spatial coordinates, dimensions, and categorical identification, specifically formatted as:

[xmin, ymin, xmax, ymax, width, height, tag, label] .

This detailed information forms the foundation for subsequent analytical and visualization processes.

Upon processing a webpage, the CNN model scans and identifies distinct web elements. For each detected element, the model extracts a set of coordinates: xmin and ymin (representing the top-left corner of the bounding box), and xmax and ymax (indicating the bottom-right corner). Additionally, it computes the width and height of the bounding box, providing a comprehensive understanding of each element's spatial occupation on the page.

In this process, each detected web element is categorized with both a tag and a label. The tag corresponds to the type of HTML element, such as button, nav, or img, offering a clear indication of the element's structural role within the webpage. The label, in contrast, corresponds to the specific identification given by the CNN model, denoting the unique characteristic or category of the element as recognized in the analysis (Example Figures: 4.5, 4.6 and 4.7).

This dual categorization approach provides a comprehensive understanding of the webpage, highlighting not only the structural components but also the specific categories of elements as detected by the model.

Post extraction, the system generates multiple images, each consolidating specific types of web elements as identified by their labels. For instance, one image will display all detected navigation bars, another all headers, and so forth.

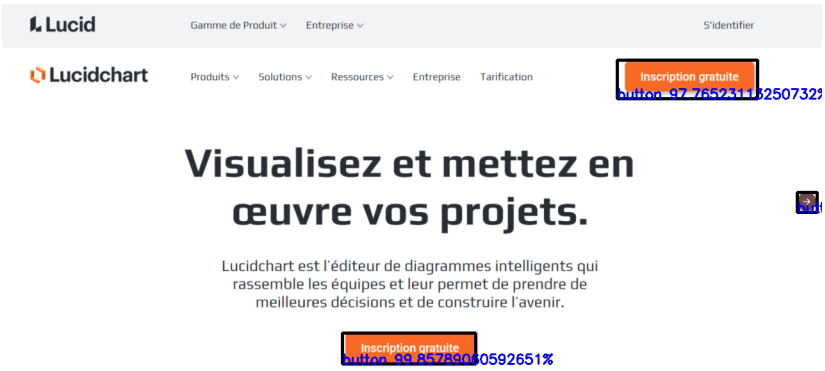


Figure 4.5: Illustration of Raw Results for Button Detection. This image showcases the comprehensive detection of all buttons within a webpage design.

Figure 4.6: Illustration of Raw Results for NavBar Detection. This image illustrates the effective identification and comprehensive detection of all navigation bar elements in a webpage design.

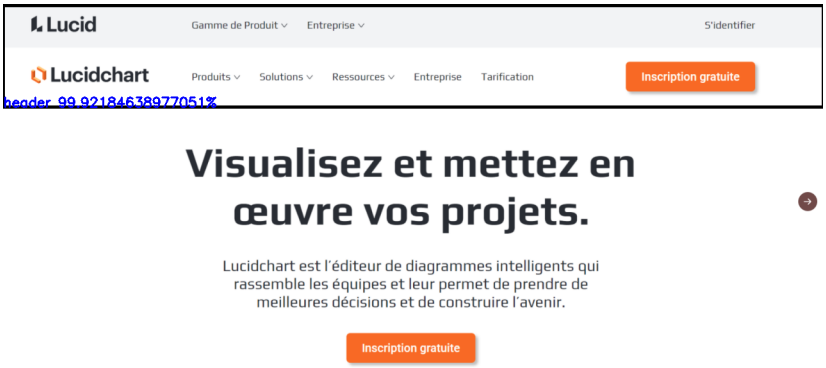


Figure 4.7: Illustration of Raw Results for Header Detection. This image illustrates the effective identification and comprehensive detection of all header elements in a webpage design.

In the pursuit of detecting structural web elements like sections and divs, image segmentation plays an important role. This technique involves partitioning a digital image (Figure: 4.8) into multiple segments or sets of pixels to simplify and/or change the representation of an image into something more meaningful and easier to analyze. By applying image segmentation, the system effectively reduces background noise and irrelevant details, which might otherwise interfere with the accurate identification of containers on a webpage (Figure: 4.9).

Segmenting the image refines the focus on essential structural elements by isolating them from less relevant content. This targeted approach significantly improves the detection process for containers such as sections and divs. With segmentation, the system can discern and classify these elements with greater precision, leading to a more accurate reconstruction of the webpage's layout when analyzing the original image.

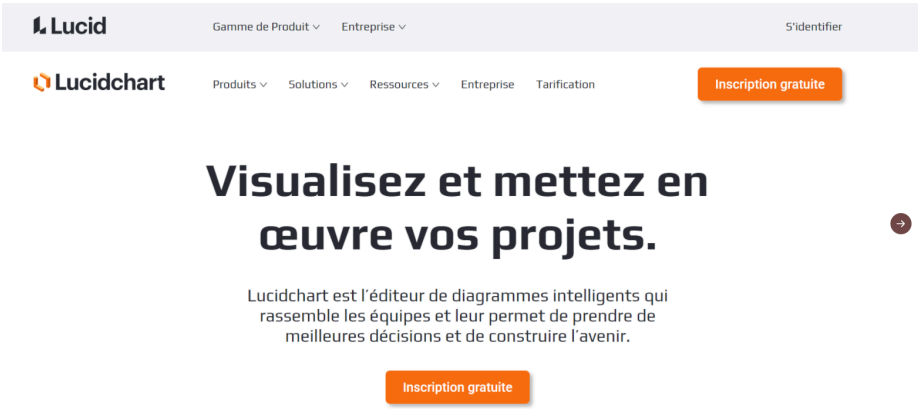


Figure 4.8: Example of an webpage desing

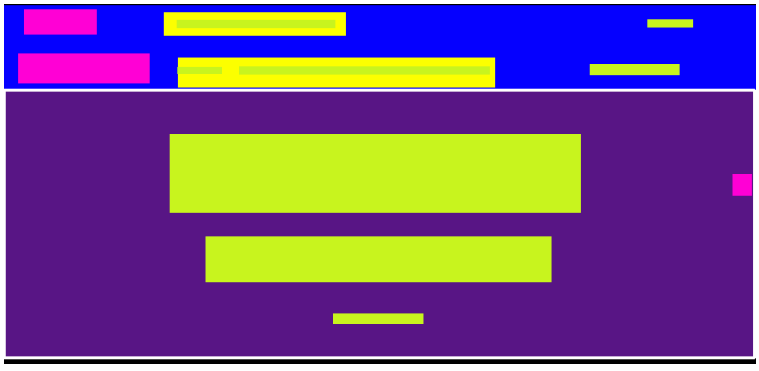


Figure 4.9: Segmented image representation of webpage design figure: 4.8

It is important to note that the image localization subsystem is a prerequisite for producing the segmented images necessary for this phase of detection. This subsystem is responsible for identifying the images from the webpage that will subsequently required on the segmentation process. It is also worth mentioning that the tasks of the localization system do not necessarily have to be executed in a linear sequence; they can be processed in parallel with other tasks, such as text extraction. This parallelization of tasks allows for a more efficient workflow, where different subsystems work concurrently until all necessary preprocessing has been completed for the final detection and analysis.

In the final configuration, the system employs a dual-model approach. The first model is trained to identify and classify a variety of web elements. This includes discerning buttons, navigation bars, and other components that are crucial. The second CNN model is specially trained using segmented images. This adapted approach is focused on detecting containers, such as sections and divs (Figure: 4.10), which are fundamental for understanding the structural hierarchy and layout of a webpage.

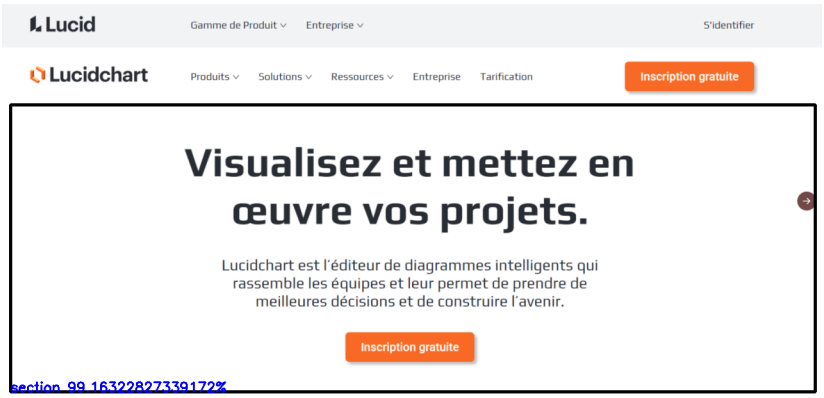


Figure 4.10: Visualization of Sections Detection on Original Webpage Design. This image delineates the detection of sections as identified by the segmentation model. To enhance clarity and context, the maps sections are transposed on the original webpage image, employing rectangles to distinctly represent the location and boundaries of each detected section.

4.4.2 Data Structuring Subsystem

At the end, the raw data extracted from TensorFlow's object detection process consists of an array that includes the following information for each detected element:

1. **ymin:** The minimum Y coordinate of the bounding box, representing the top edge of the box.
2. **ymax:** The maximum Y coordinate of the bounding box, representing the bottom edge of the box.
3. **xmin:** The minimum X coordinate of the bounding box, representing the left edge of the box.

4. **xmax:** The maximum X coordinate of the bounding box, representing the right edge of the box.

5. **width:** The width of the bounding box.

6. **height:** he height of the bounding box.

7. **(box_to_score_map[box]*100):** The confidence score for the detected object, expressed as a percentage. This value is derived from the model's confidence in its detection, with a higher percentage indicating greater certainty.

8. **cn[counter_for]:** The label assigned to the detected object, which identifies the type of web element that has been recognized by the model.

Each array in the dataset corresponds to a detected web element, having essential details like spatial coordinates, dimensions within the image, the confidence level, and the category label assigned by the object detection model. These details are very important for subsequent processing steps, including filtering operations that rely on confidence thresholds.

The system verifies each element against a minimum confidence level, predetermined by the model's threshold, to ensure reliable detection accuracy. Upon confirming the element's validity, the data is systematically organized into a structured format, typically a list of lists, containing information in the following order: [xmin, ymin, xmax, ymax, width, height, tag, label]. This structured arrangement facilitates easier analysis and manipulation of the data for further applications.

The code B.8, as presented in the appendix, encapsulates all the operational logic of the web elements detection system. It outlines the initialization of the model, the processing of the webpage images, and the execution of the detection algorithm. The script also details the post-processing steps, where the raw data from TensorFlow's object detection output is filtered and organized, ensuring that only relevant and accurately identified elements are included in the final analysis.

4.4.3 Image Localization Subsystem

Template matching is a technique in computer vision used for finding a sub-image (template) in a larger image (main image). This method is particularly effective in scenarios where the template needs to be located precisely within the main image. The process involves sliding the template image over the main image (as in 2D convolution) and comparing the template with the patch of the main image under the template image.

How Template Matching Works

- **Grayscale Conversion:** Initially, both the main image and the template are converted to grayscale. This simplification reduces computational complexity and enhances the effectiveness of the matching process.
- **Sliding Window:** The template is slid over the main image, and at each position, a similarity measure is calculated.
- **Similarity Measures:** Several methods can be employed to assess the similarity between the template and the current patch of the main image, such as Cross-Correlation, Square Differences, and Correlation Coefficient.
- **Match Location:** The position where the highest similarity is found (or the lowest difference, depending on the method used) indicates the probable location of the template in the main image.

In OpenCV, template matching is implemented using the function cv2.matchTemplate(). This function returns a grayscale image where each pixel denotes how much does the neighbourhood of that pixel match with the template.

Image Search and Size Optimization System

1. **Search Process Using OpenCV Template Matching**
    - **Initial Search:** The system begins by applying template matching using OpenCV to search for instances of the sub-images (templates) within the main image. This is done by sliding each template across the main image and computing a similarity score at each position.
    - **Multi-Scale Search:** Since traditional template matching is scale-sensitive, the system must consider variations in the size of the templates as they appear in the main design.
2. **Size Optimization Through Iterative Reduction**
    - **Iterative Scaling:** To handle the scale variance, the system employs an iterative approach where the size of each template is reduced by 5% in each iteration.
    - **Best Fit Determination:** In every iteration, the template matching process is reapplied. The system records the similarity scores for each scaled version of the template.
    - **Optimal Size Selection:** The iteration that yields the highest similarity score indicates the best fit, i.e., the size at which the template most closely matches a portion of the main image.

3. **Key Aspects of the System**

   - **Efficiency in Localization:** By iteratively scaling the templates, the system efficiently localizes sub-images within the main image, even when the sizes in the original design differ.
   - **Precision of Matching:** This method enhances the precision of matching by adjusting for size discrepancies, ensuring that the templates are accurately identified within the main image.
   - **Automated Scaling and Matching:** The process is automated, making the system robust and scalable for different templates and main images.

Results and Evaluation

The primary input for the system is the original web design image (Figure: 4.11), which is a composite that includes various individual elements layered to form the final design. For the purposes of this demonstration, two key elements were used: a smartphone image (Figure 4.12a) and a logo image labeled 'Bacon' (Figure: 4.12b).

Figure 4.11: Example of the webpage design image.

Figure 4.12: Images embedded within the webpage design layout.

(a) Smartphone Image Incorporated in Webpage Design: 4.11

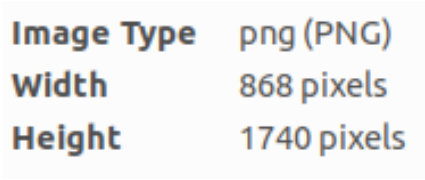
(b) Logo Image Incorporated in Webpage Design: 4.11

The system leverages OpenCV's template matching algorithm to search for the presence of these individual images within the main design. The algorithm computes a match by sliding the template image over the main design and quantifying the similarity at each position. Given the scale-sensitive nature of template matching, the system incorporates an iterative scaling technique, systematically reducing the size of the template images by 5% during each iteration to refine the search. The resulting output, as demonstrated by the final image (Figure: 4.13), showcases the system's efficacy.

Figure 4.13: Results of Image Detection Using Template Matching Technique. The individual elements — the smartphone and the logo — are accurately positioned within the web design template. This precision is evidenced by the congruence of size and location in relation to the encompassing elements of the design.

The original smartphone image, with a width of 868 pixels and a height of 1740 pixels (Figure 4.14a), experienced a considerable size modification. This adjustment was necessary to fit the image within the constraints of the web design's template, ensuring a responsive and visually consistent end result. As indicated by the stylesheet snippet (Figure: 4.14b), the original dimensions of the image, as required by the web design, were modified to a height of 271 pixels and a width of 135 pixels. This resizing is essential to align the visual representation in the code with the webpage design.

Figure 4.14: Adjusting image dimensions to match webpage requirements.



(a) Original Dimensions of Image. Figure: 4.12a

```
.div-image3__img__image3 {
    height: 271px;
    width: 135px;
}
```

(b) Results of Validating the Image Size with CSS3 Code for Accurate Webpage Insertion.

The image localization subsystem's accurately determined the necessary scaling to transition from the original image size to the dimensions specified in the stylesheet, showcasing the precision of the algorithm employed.

Finally the following example (Figure: 4.15) serves as an additional demonstration of the image localization subsystem's capabilities within the automated web code generation framework. It exemplifies the subsystem's precision and reliability in identifying, positioning, and resizing images to match the predetermined design criteria of a website.

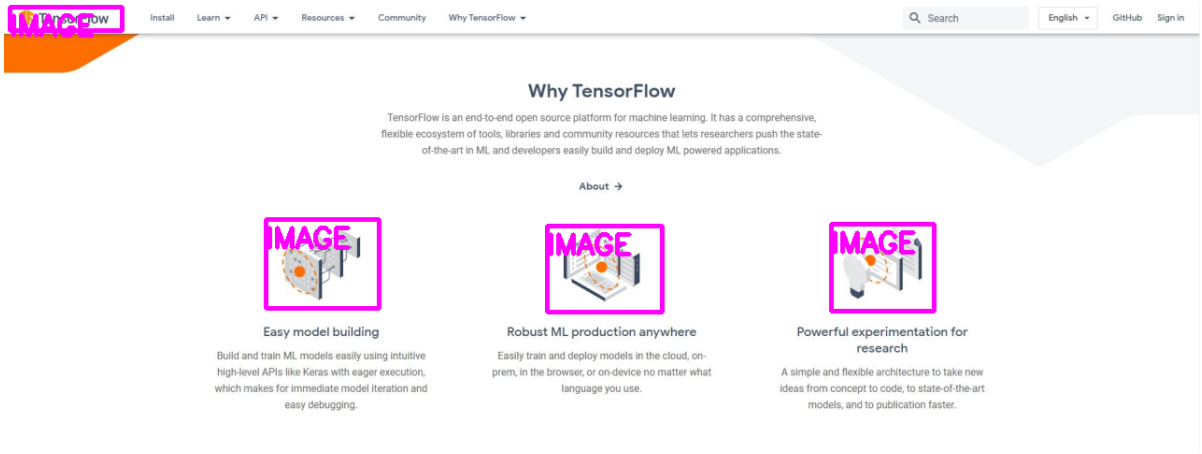


Figure 4.15: Additional Example of the results of Image Localization Subsystem.

4.4.4 Text Extraction Subsystem

Optical Character Recognition, commonly known as OCR, is a technology in the field of machine learning and computer vision that enables the conversion of different types of documents, such as scanned paper documents, PDF files, or images captured by a digital camera, into editable and searchable data. When applied to web design, OCR technology becomes an indispensable tool for extracting textual content directly from website images.

Understanding OCR: From Image to Text

The process begins with preprocessing, where the quality of the input image is enhanced. This step is all about making the text as clear as possible. Techniques like reducing background noise and adjusting the contrast are used. This step is important because clearer text makes the following steps more accurate. Next, the OCR system identifies where the text is on the image. This is known as text detection. It involves breaking down the image into segments, identifying which parts contain text and which do not. By focusing only on the text parts, the system avoids getting confused by other elements in the image like pictures or decorations.

Once the text regions are identified, the system moves onto character recognition. Here, the OCR software examines each character in these regions. It uses a database of character patterns or advanced machine learning models to recognize and identify each character. These models have learned from a vast amount of data, which helps them recognize a wide range of fonts and handwriting styles. After the characters are identified, the system enters the post-processing phase. This stage is all about cleaning up and refining the output. The software might check spelling, understand the context, and even use specialized dictionaries to correct any mistakes. This step is crucial for ensuring that the final text is not just accurate but also makes sense. Finally, the OCR process concludes with the output stage. Here, the recognized text is converted into a digital format that can be edited and searched.

Choosing an External OCR Service Over Building a Custom Model

Developing a machine learning model for text detection from scratch is a substantial challenge. The primary difficulty is the requirement for extensive data, with thousands of annotated images needed to effectively train a model. Additionally, algorithm development and optimization are necessary to achieve reliable accuracy. Given these factors, the decision in this project was to utilize established external services like Google Cloud Platform (GCP) and Amazon Web Services (AWS). This approach leverages advanced technologies to process and achieve optimal results in text extraction.

The initial strategy involved integrating both GCP and AWS, aiming to capitalize on the strengths of each service. However, this integration led to unexpected complications, such as bugs and inconsistencies in the text extraction process. To address these issues and ensure stability and accuracy, an evaluation of both services was conducted. The assessment focused on factors such as accuracy, consistency, and overall performance in OCR tasks. Based on the evaluations, Google Cloud Platform, with its Cloud Vision API, was chosen as the primary tool for OCR needs. GCP's Cloud Vision demonstrated superior performance in accurately extracting text from a wide range of images, aligning well with the project's requirements. Additionally, Amazon Web Services with its Textract package was integrated as a backup service. This setup ensures that an alternative is available in case of any interruptions or issues with the primary service. This dual-service approach enhances the robustness of the system and provides flexibility in handling diverse OCR challenges.

The use of these established platforms also offers scalability, an essential feature for projects that may need to handle varying volumes of data. They are designed to process large quantities of images, making them suitable for projects with high workloads or those that may scale over time. Another significant benefit is the reduction in development time and costs. By leveraging these ready-made services, the project avoids the extensive time required for building and training a custom OCR model. GCP and AWS continuously update their services, incorporating the latest advancements in machine learning and OCR technology, which means the project inherently stays at the forefront of technological advancements without additional investment in research and development.

Addressing Challenges in Text Extraction Using External OCR Services

Despite the high accuracy of external OCR services like Google Cloud Platform and Amazon Web Services, several challenges appear in the process of text extraction from website images. One notable issue was the occasional detection of strange symbols or noise within the images (Figures 4.16 and 4.17). This phenomenon, often a result of the complex textures or patterns in the design, required additional filtering mechanisms to ensure only relevant textual content was extracted.

Figure 4.16: Illustration of Symbol and Noise Identification in Text Extraction via OCR Technology.

Figure 4.17: Illustration of Symbol and Noise Identification in Text Extraction via OCR Technology.

Another complication involved differentiating between textual content that was part of the website's design and text or symbols embedded within pictures (Figure 4.18). To maintain the integrity of the web design in the code generation process, it became necessary to implement strategies to ignore text detected within pictures. This selective extraction ensured that only pertinent text elements were included in the final web code output.

Figure 4.18: Visual Representation of Text Within Pictures.

Moreover, the project faced the complexity of categorizing text blocks into appropriate web elements. Determining whether a text block was a title or a paragraph, and further, if it was a title, assigning the correct heading level (from H1 to H6) presented a unique challenge. This classification was important for preserving the hierarchical structure of the original web design in the generated code, thereby enhancing the website's Search Engine Optimization (SEO).

To categorize text as either belonging to a paragraph or a title, the GCP Cloud API was employed (a similar approach is feasible with AWS). This API is designed to extract text blocks, which are pre-trained to group logically. Here, titles are identified as blocks containing fewer than a specified number of characters. Subsequently, all detected text is divided into titles and paragraphs using the Text Analyzer Script (Appendix-B: B.9). Following this separation, the Title Analyzer (Appendix-B: B.10) is then utilized to assign the appropriate HTML heading tag (h1-h6) to each title element. To classify titles into appropriate heading tags, the K-Means algorithm was applied to cluster titles based on their size. K-Means is a widely-used algorithm in data analysis for clustering, which means it groups data points into distinct categories based on their characteristics. The 'K' in K-Means represents the number of clusters to be formed. The process involves assigning data points to the nearest cluster center and iteratively adjusting these centers until the most optimal grouping is achieved. This algorithm is efficient in identifying patterns and similarities among data points.

In the context of title classification in web content, K-Means can be applied to group titles based on certain features like font size, position, or formatting. For instance, titles with similar characteristics can be clustered together, aiding in determining their hierarchical significance in the web page's structure. By using K-Means, titles can be systematically classified into different heading levels (such as h1, h2, etc.), reflecting their relative importance and role in the overall organization of the content.

At the end, the process involved creating five distinct groups. The h1 tag was exclusively assigned to the most notable title, identified as the one with the lowest 'ymin' value. The remaining titles were then grouped into five categories, corresponding to h2-h5 classifications (Figure: 4.19). This clustering approach facilitated an organized and systematic assignment of heading tags, aligning with the hierarchical structure of the web content.

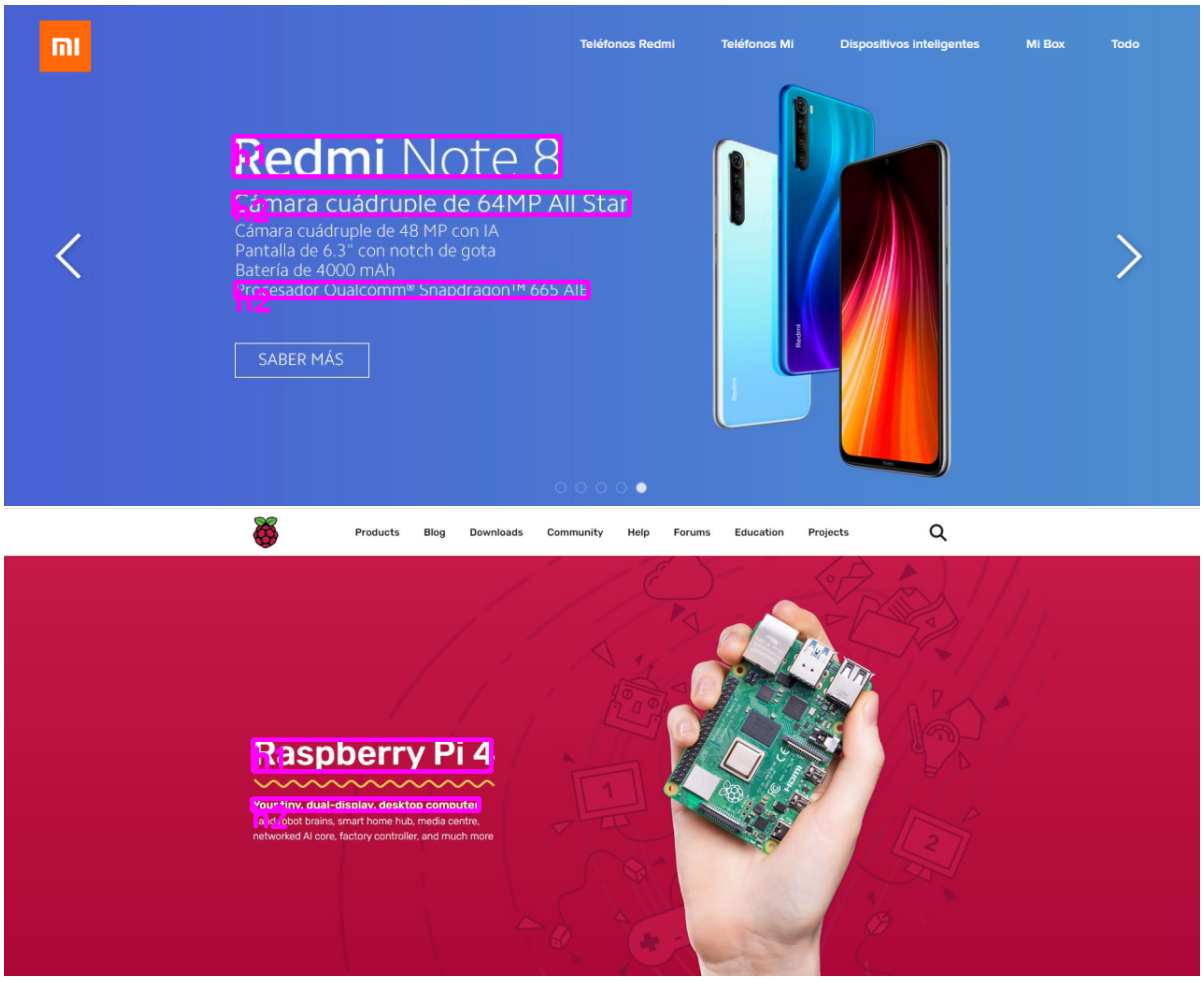


Figure 4.19: Illustration of Heading Level Assignment for Titles (h1-h6).

The additional challenge addressed in this research involved the precise mapping of text to specific web components such as buttons, navigation bars, headers, and other interactive elements. This mapping is critical as it not only involves the recognition of text but also the discernment of its contextual relevance and functional role within the website's architecture. Ensuring the accurate association of text with its corresponding elements is fundamental to preserving the operational integrity and user experience of the site (Figure 4.20).

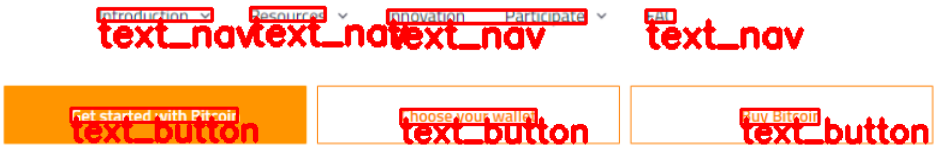


Figure 4.20: Illustration Mapping Text to Web Elements.

Text Extraction Subsystem Results

The results demonstrate the OCR service's proficiency in recognizing text across varying fonts, sizes, and styles, including its adaptability to diverse contrast levels and complex backgrounds. The following figures (Figures: 4.21, 4.22 and 4.23) are dedicated to displaying the outcomes of employing Optical Character Recognition (OCR) services in the extraction of textual content from web-based images. These images provide a visual results of the OCR's performance, demonstrating its accuracy in text recognition and the effectiveness of its integration into web design.

Figure 4.21: Text Extraction Subsystem Results. Example 1.

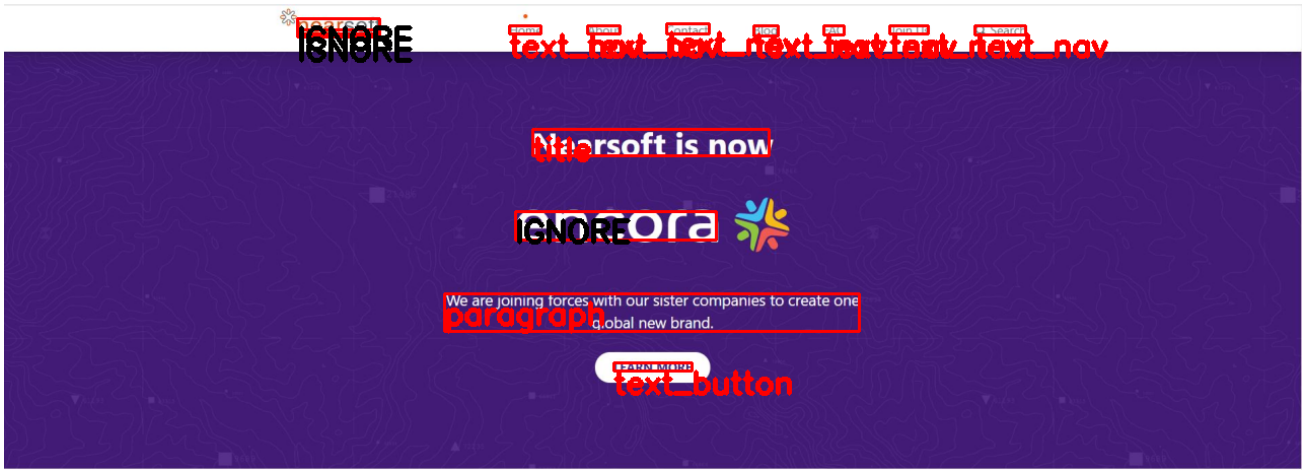


Figure 4.22: Text Extraction Subsystem Results. Example 2.

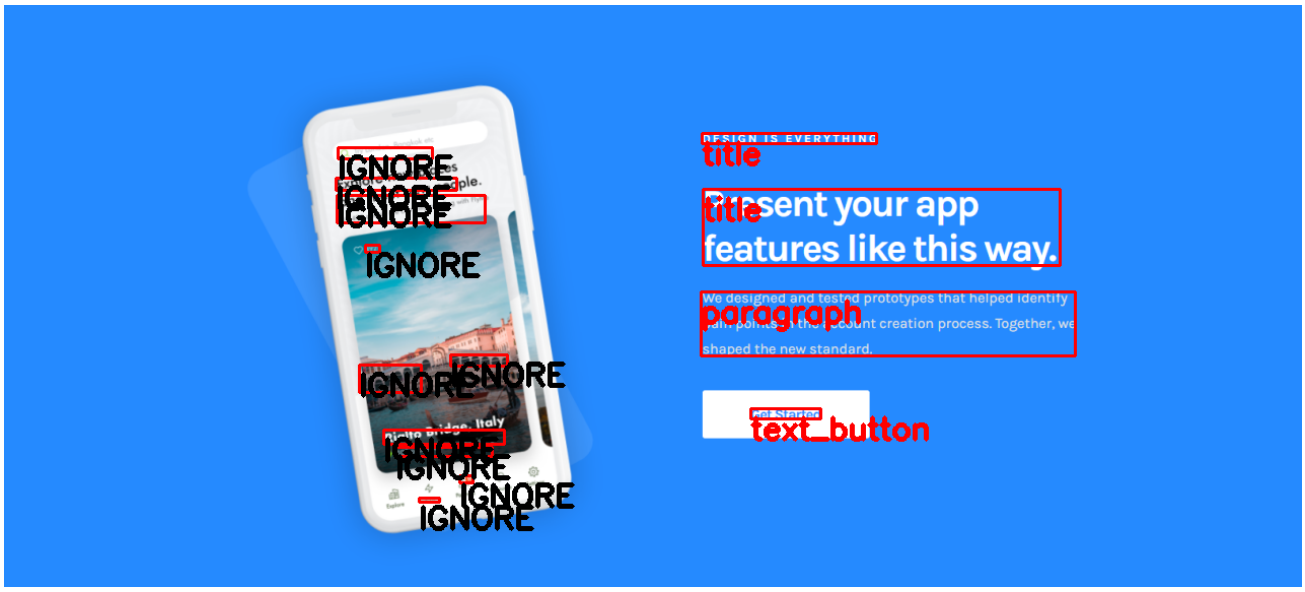


Figure 4.23: Text Extraction Subsystem Results. Example 3.

4.4.5 Tree Construction Subsystem

In the progression of the project, a graph generator was developed, incorporating a 'Node' class to transform the way data is processed. Rather than utilizing a conventional list structure such as [xmin, ymin, xmax, ymax, width, height, tag, label], the data is now encapsulated within a Node class, formatted as Element key Node(xmin, ymin, xmax, ymax, width, height, tag, label). This restructured data is then systematically stored in a dictionary.

This new system architecture allows for a hierarchical organization where each web component, such as buttons and headers, is associated with a unique 'Node' that represents its spatial and attribute data.

The structure is exemplified as follows:

```
{
    'images': {
        'image0': <graph_generator.node.ImageNode object at 0x7f057ba89208>,
        'image1': <graph_generator.node.ImageNode object at 0x7f0579cf24a8>,
        'image2': <graph_generator.node.ImageNode object at 0x7f0579cf2780>,
        'image3':<graph_generator.node.ImageNode object at 0x7f0579cf2438>,
        'image4': <graph_generator.node.ImageNode object at 0x7f0579cf22b0>
    },
    'buttons': {
        'button0':<graph_generator.node.Node object at 0x7f057ba5e400>,
        'button1': <graph_generator.node.Node object at 0x7f0579d75438>
    },
    'headers': {
        'header1': <graph_generator.node.Node object at 0x7f05725ed128>
    }
}
```

This approach greatly simplifies the manipulation of web elements, enabling straightforward access to all components like buttons, headers, footers, and navigation items. It facilitates the dynamic assembly and modification of the web design's tree structure, making the conversion to XML more efficient and enhancing the capability to append, remove, and manage the relationships between parent, children, and sibling elements within the tree.

Prior to the creation of a node element within the system, a series of validations are executed to ensure the structural integrity of the web design hierarchy. One such validation involves checking for nested elements under the same label. For instance, if a header is found within another header, the system identifies this redundancy and retains only the element with the larger area. By eliminating nested duplicates, the system effectively reduces redundancy, thereby significantly enhancing the accuracy of the web design representation. This validation is crucial for maintaining a clean and accurate tree structure, which is essential for the subsequent conversion process into XML format.

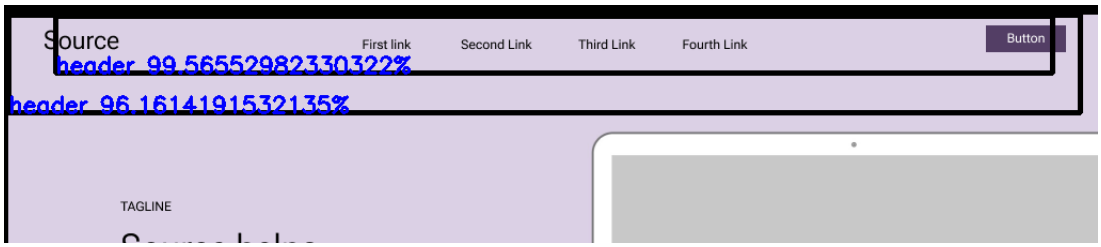


Figure 4.24: Example of Header Redundancy. Here is necessary eliminating the nested elements.

Upon detecting two headers as indicated in Figure 4.24, the system retrieves the raw data structured as follows:

```
[0.0028250187169760466, 0.0773322731256485, 0.043722447007894516, 0.9553057551383972, 1024, 700, 99.56552982330322, 'header']
[0.00018005371384788305, 0.1307305246591568, 0.0, 0.9804819822311401, 1024, 700, 96.1614191532135, 'header']
```

This data is then transformed into a node-based structure for integration into the tree graph, as such:

```
"headers": {
  "header0": "<graph_generator.node.Node object at 0x7f0579d75668>",
  "header1": "<graph_generator.node.Node object at 0x7f05725ed128>"
}
```

Subsequently, the system employs a review function to detect and resolve instances of duplicate nested elements. Through this function, the system retains only the element with the larger area, effectively removing any nested duplicates. The refined tree graph structure, free from redundancies, ultimately contains:

```
"headers": {
  "header1": "<graph_generator.node.Node object at 0x7f05725ed128>"
}
```

To structure the nodes into a tree graph to XML format, the system utilizes the xml.dom.minidom module in Python, often abbreviated as md. This library provides a wide array of functionalities for manipulating nodes within the tree. The library helps to handle operations like:

- The appendChild(newChild) method is employed to add a new child node to the end of the current list of children of a particular node. If the child node already exists elsewhere in the tree, it is first removed from its original location before being appended.
- The insertBefore(newChild, refChild) function allows for the insertion of a new child node before an existing child node. This method requires that the refChild must be an existing child of the node in question, or it raises a ValueError. The newChild is then inserted at the designated position or, if refChild is None, it is added at the end of the list.
- The removeChild(oldChild) method is used to delete a child node from the tree. This operation is only successful if oldChild is indeed a child of the node, and it returns the oldChild upon successful removal. It is recommended to call the unlink() method on the oldChild if it will no longer be used, to ensure proper cleanup.
- The replaceChild(newChild, oldChild) method is utilized when a node needs to be replaced with another. This method also checks that oldChild is a child of the node and raises a ValueError if this is not the case.

These methods provide a robust framework for editing the tree structure, allowing for the addition, removal, and rearrangement of nodes, which is essential for converting the tree structure into well-formed XML and for making modifications that reflect the structure of the web design accurately.

In addition to the aforementioned methods for manipulating nodes, the xml.dom.minidom (md) library is instrumental in retrieving node relationships and properties within the tree structure. It provides the capability to access sibling nodes, discern parent and child relationships, and navigate the node hierarchy. This functionality is crucial for understanding the tree structure and for the precise placement of elements within the XML. Moreover, the md library includes methods to obtain and manipulate the attributes of nodes, allowing for a detailed customization of node properties. It also offers features to compare nodes, facilitating the examination of their equivalence or similarity within the tree.

Header Element Integration in XML Structure

Following the transformation of web elements into nodes, these nodes can be systematically integrated into the XML tree. The initial step involves the integration of the 'header' element and all associated child elements, such as navigation bars, text links, logos, and titles.

An example of this insertion within the XML structure is demonstrated by the following code snippet:

```
xml_gen.appendTo(
    "div",
    "header",
    attributes={
        "id": parent_key,
        "css_pattern": header_key
    },
    parent_id=class_header.key
)
```

This code facilitates the straightforward insertion of nodes into the XML structure and simplifies adding new nodes and changing current ones in the XML code. The append of the 'header' element into the XML structure results in a representation of the webpage's design (Figure: 4.25).

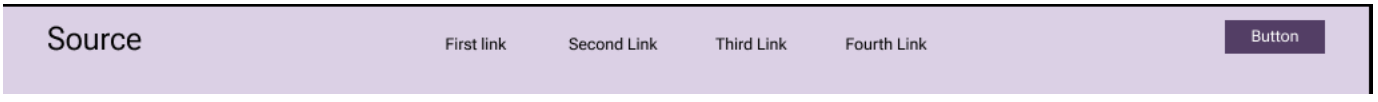


Figure 4.25: Header extracted from an example of a webpage design.

The resulting XML structure based on the example of the header element picture (Figure: 4.25) is as follows:

```
<body>
  <header id="header1">
    <input class="header__checkbox" id="header__checkbox" type="checkbox"/>
    <label class="icon_menu" for="header__checkbox">
      <i class="fas␣fa-align-justify"> </i>
    </label>
    <h3 class="header__h3__txtheader0-0" css_pattern="header">
        Source
    </h3>
    <nav class="nav0" css_pattern="header1">
      <ul class="nav0__ul" css_pattern="nav0">
        <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item0">
          <a class="nav-link␣nav0__a__txtnav0-4" css_pattern="nav0" href="#">
            First link
          </a>
        </li>
        <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item1">
          <a class="nav-link␣nav0__a__txtnav0-10" css_pattern="nav0" href="#">
            Second Link
          </a>
        </li>
        <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item2">
          <a class="nav-link␣nav0__a__txtnav0-14" css_pattern="nav0" href="#">
            Third Link
          </a>
        </li>
        <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item3">
          <a class="nav-link␣nav0__a__txtnav0-15" css_pattern="nav0" href="#">
            Fourth Link
          </a>
        </li>
      </ul>
    </nav>
    <button class="header__button__button0" css_pattern="header">Button</button>
  </header>
</body>
```

This structure clearly defines the ‘header’ as the container for the topmost content of the webpage, including an input for a menu checkbox, a label with a menu icon, a heading for the website’s title or section name, a navigation block with a list of links, and a button. Each element is given specific classes and identifiers, which not only facilitate styling with CSS but also help in identifying these elements for further manipulations or content insertion.

Strategic Insertion of Container Elements in XML Code

The process begins once the raw data of web elements is effectively converted into nodes within this structure. This conversion sets the stage for identifying and locating various container elements such as sections, divs, and footers. A key focus of this step is the insertion of container elements that have been identified by the Convolutional Neural Network (CNN) model. It is critical to note that the insertion process is selective; the XML code will only incorporate container elements ignoring web elements like texts, buttons, or images. This selective approach ensures that the XML structure remains sorted and focused on the essential structural elements of the webpage.

To illustrate this feature, a webpage design was selected (Figure: 4.26), and the CNN model trained to detect section and div elements was employed to identify that containers.

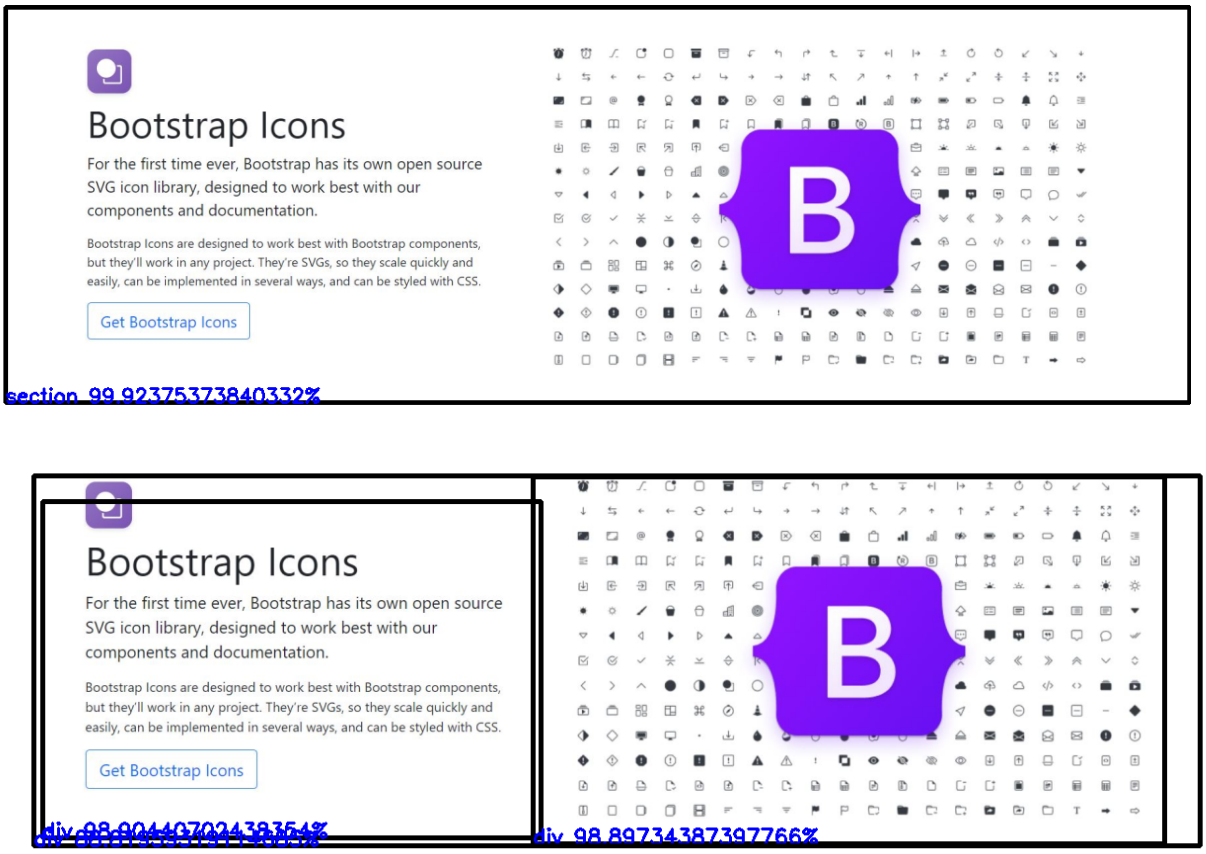


Figure 4.26: Section and Div Container Detection Visualization. The CNN model's output highlighting identified section and div elements.

The container insertion logic effectively organizes the webpage's structure. It accurately positions each container, identifying parent and child relationships, and prepares them to contain the webpage's content. The code provided below exemplifies the precise conversion of visually detected elements (Figure: 4.26) into the XML tree structure.

```
<body>
  <section id="section_0">
    <div id="div_5">
      <div id="div_4"/>
      <div id="div_3"/>
    </div>
  </section>
</body>
```

Enhancing Webpage Structure: Implementing Main Section

There are a few cases where a main section is absent, and to continue with the analyses an adjustment to the webpage's structure is required. Consider the following initial XML code as an example:

```
<body>
    <div id="div_5">
        <div id="div_4"/>
        <div id="div_3"/>
    </div>
</body>
```

In certain scenarios where the main section is missing, it becomes necessary to modify the webpage's structure for continued analysis. The following initial XML code serves as an illustrative example:

1. Only 'section' elements should be direct children of the 'body' node.
2. All web elements must be enclosed within a 'div' container.
3. Any container without children should be removed.

To align with these rules, a main 'section' will be inserted on the XML tree structure:

```
<body>
  <section id="main_section">
    <div id="div_5">
      <div id="div_4"/>
```

```
      <div id="div_3"/>
    </div>
  </section>
</body>
```

This adjustment guarantees adherence to established guidelines and enhances the overall architecture of the webpage.

Insertion of Web Elements:

After establishing the webpage's structure and inserting the necessary containers, the next step is to add web elements within each 'div' element. The process involves going through each 'div' element in an order sorted from smallest to largest area. This sorting is crucial to ensure that each element within a 'div' is inserted once and marked as allocated. This approach prevents duplication, ensuring elements are only inserted in the smallest containing 'div' and not repeated in larger 'divs' that also encompass the same elements. Here is an example to illustrate this, initial structure:

```
<body>
  <section id="main_section">
    <div id="div_5">
      <div id="div_4"/>
      <div id="div_3"/>
    </div>
  </section>
</body>
```

Result After Element Insertion:

```
<body>
  <section id="section_1">
    <div id="div_5">
      <div deepest="True" id="div_4">
        <img class="div_4__img__image0" css_pattern="div_4" src="images/upload_test13discordimage11.png"/>
      </div>
      <div deepest="True" id="div_3">
        <p class="div_3__p__text0" css_pattern="div_3">
            An invite-only place with plenty of room to talk
        </p>
        <p class="div_3__p__text1" css_pattern="div_3">
            Discord servers are organized into topic-based channels where you can collaborate,
            share, and just talk about your day without clogging up a group chat.
        </p>
      </div>
    </div>
  </section>
</body>
```

This outcome demonstrates the successful implementation of the sorting and insertion strategy, ensuring efficient organization and avoidance of redundant content. The XML code generated is the outcome of the analysis performed on the web page design (Figure: 4.27).

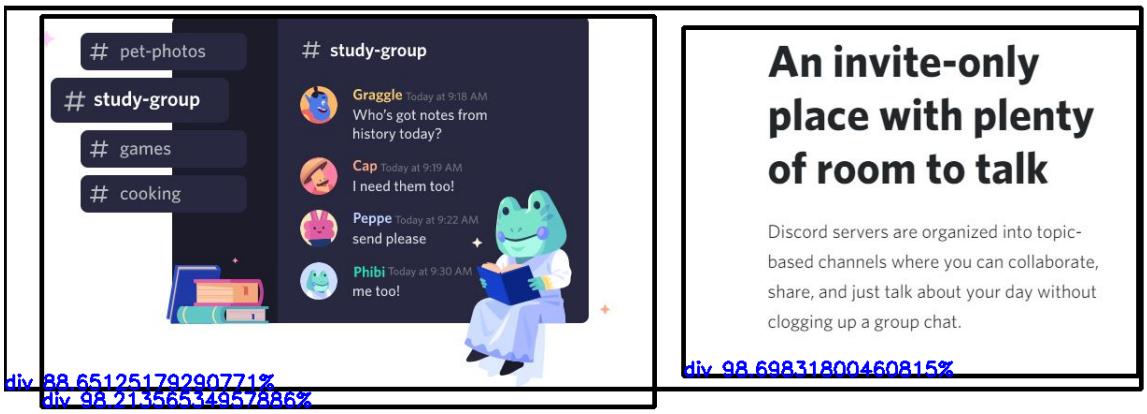


Figure 4.27: Webpage Design Example. This illustration demonstrates the application of element insertion techniques.

4.5 Inference and Correction Subsystem: Enhancing Auto-generated Code Quality

4.5.1 Alignment of Web Elements

In the web elements insertion process, it is not uncommon for elements and containers to be mistakenly inserted in an incorrect sequence. To address this issue, a correctness algorithm has been developed to ensure that the insertion of web elements is organized in the proper order. This algorithm is particularly effective in scenarios where elements that are siblings need to be reordered within the same hierarchical level. It systematically sorts nodes that share the same parent, thereby maintaining the intended structural integrity.

Consider the following scenario as an illustrative example (Figure: 4.28): a div container encompasses three other divs (labeled as div_8, div_7, div_6). However, these nested divs are inserted in a wrong sequence. The algorithm rectifies this by reordering the nodes, thus producing the correct arrangement of elements. This ensures that the overall structure and flow of the web page align with the intended design and functionality. The figure 4.28 illustrates this scenario.

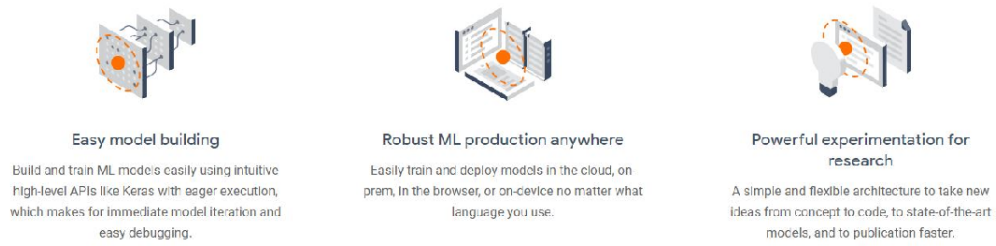


Figure 4.28: Webpage Design Example.

The XML structured obtained from web elements insertion algorithm is the following code:

```
<section id="section_1">
    <div id="div_11">
      <div deepest="True" id="div_9">
        <img class="div_9__img__image3" css_pattern="div_9" src="images/upload_test19tensorflowimage31.png"/>
        <h2 class="div_9__h2__text13" css_pattern="div_9">
            Powerful experimentation for research
        </h2>
        <p class="div_9__p__text14" css_pattern="div_9">
            A simple and flexible architecture to take new ideas from concept
            to code, to state-of-the-art models, and to publication faster.
        </p>
      </div>
     <div deepest="True" id="div_7">
          <img class="div_7__img__image0" css_pattern="div_7" src="images/upload_test19tensorflowimage21.png"/>
          <h2 class="div_7__h2__text10" css_pattern="div_7">
            Robust ML production anywhere
          </h2>
          <p class="div_7__p__text11" css_pattern="div_7">
            Easily train and deploy models in the cloud, on- prem, in the browser,
            or on-device no matter what language you use,
          </p>
        </div>
        <div deepest="True" id="div_8">
          <img class="div_8__img__image2" css_pattern="div_8" src="images/upload_test19tensorflowimage11.png"/>
          <h2 class="div_8__h2__text7" css_pattern="div_8">Easy model building</h2>
          <p class="div_8__p__text8" css_pattern="div_8">
            Build and train ML models easily using intuitive high-level APIs like
            Keras with eager execution, which makes for immediate model iteration
            and easy debugging.
          </p>
        </div>
    </div>
</section>
```

Applying this alignment algorithm could potentially resolve the issues, resulting in the following XML Tree structure:

```
<section id="section_1">
    <div id="div_11">
        <div deepest="True" id="div_8">
          <img class="div_8__img__image2" css_pattern="div_8" src="images/upload_test19tensorflowimage11.png"/>
          <h2 class="div_8__h2__text7" css_pattern="div_8">Easy model building</h2>
          <p class="div_8__p__text8" css_pattern="div_8">
            Build and train ML models easily using intuitive high-level APIs like
            Keras with eager execution, which makes for immediate model iteration
            and easy debugging.
          </p>
        </div>
        <div deepest="True" id="div_7">
          <img class="div_7__img__image0" css_pattern="div_7" src="images/upload_test19tensorflowimage21.png"/>
          <h2 class="div_7__h2__text10" css_pattern="div_7">
            Robust ML production anywhere
```

```
          </h2>
          <p class="div_7__p__text11" css_pattern="div_7">
            Easily train and deploy models in the cloud, on- prem, in the browser,
            or on-device no matter what language you use,
          </p>
        </div>
        <div deepest="True" id="div_9">
            <img class="div_9__img__image3" css_pattern="div_9" src="images/upload_test19tensorflowimage31.png"/>
            <h2 class="div_9__h2__text13" css_pattern="div_9">
                Powerful experimentation for research
            </h2>
            <p class="div_9__p__text14" css_pattern="div_9">
              A simple and flexible architecture to take new ideas from concept to
              code, to state-of-the-art models, and to publication faster.
            </p>
        </div>
    </div>
</section>
```

4.5.2 Eliminating Empty Containers

During some instances, the Machine Learning (ML) model may erroneously identify noise in 'div' elements, which are essentially empty and do not contain any nested child elements. To ensure the integrity and quality of the XML structure, it becomes necessary to remove these empty 'div' elements.

Consider the following XML example:

```
<section>
    <div deepest="True" id="div_5">
        <img class="div_5__img__image0" css_pattern="div_5" src="IMAGEPATH"/>
    </div>
    <div id="div_4" />
</section>
```

Upon applying the proposed method, the revised XML tree structure will be:

```
<section>
    <div deepest="True" id="div_5">
        <img class="div_5__img__image0" css_pattern="div_5" src="IMAGEPATH"/>
    </div>
</section>
```

This approach ensures a cleaner, more efficient XML structure by removing elements that serve no functional purpose.

4.5.3 Handling Orphan Elements in Web Element Insertion:

In the web element insertion process, the system assigns elements to their respective 'div' containers. However, there are instances where certain elements lack a 'div' as their direct parent, existing outside any established container. In such scenarios, the algorithm is required to determine if these isolated elements belong to any specific section within the XML structure. If an element does not associate with any existing section, the algorithm should create a new section and incorporate the orphan element into this new section or into an appropriate existing one.

It is essential to note that, under normal circumstances, elements should not have a 'section' as their direct parent. For the purposes of this procedure, placing the element directly within a 'section' is temporarily permissible. Later inference algorithms will address and correct this placement. The main goal at this stage is to ensure the inclusion of all elements, particularly those without a 'div' parent, in the XML tree structure. This method ensures a complete and accurate representation of web elements in XML.

To Illustrate the process the Figure 4.29 reveals that certain text elements are not encapsulated within any 'div' container.

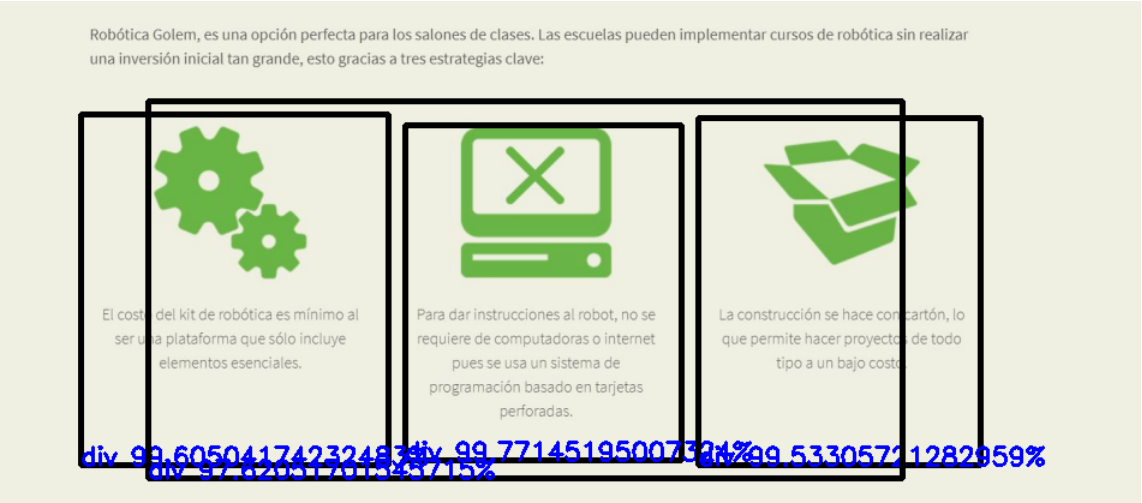


Figure 4.29: Addressing Unrecognized Elements in Web Structures.

Initially, the web element insertion results in a code structure as follows:

```
<section id="section_1">
    <div id="div_8">
      <div deepest="True" id="div_6">
        <img class="div_6__img__image2" css_pattern="div_6" src="images/upload_test16golemimage11.png"/>
        <p class="div_6__p__text1" css_pattern="div_6">
            El costo del kit de robótica es mínimo al ser una plataforma que sólo
            incluye elementos esenciales.
        </p>
      </div>
```

```
      <div deepest="True" id="div_7">
        <img class="div_7__img__image1" css_pattern="div_7" src="images/upload_test16golemimage31.png"/>
        <p class="div_7__p__text3" css_pattern="div_7">
            La construcción se hace con cartón, lo que permite hacer proyectos de
            todo tipo a un bajo costo.
        </p>
      </div>
      <div deepest="True" id="div_5">
        <img class="div_5__img__image0" css_pattern="div_5" src="images/upload_test16golemimage21.png"/>
        <p class="div_5__p__text2" css_pattern="div_5">
            Para dar instrucciones al robot, no se requiere de computadoras o internet
            pues se usa un sistema de programación basado en tarjetas perforadas.
        </p>
      </div>
    </div>
</section>
```

However, it is observed that the 'div' elements are misaligned. By applying an alignment algorithm, the order of node elements is corrected, resulting in an improved structure:

```
<section id="section_1">
    <div id="div_8">
      <div deepest="True" id="div_6">
        <img class="div_6__img__image2" css_pattern="div_6" src="images/upload_test16golemimage11.png"/>
        <p class="div_6__p__text1" css_pattern="div_6">
            El costo del kit de robótica es mínimo al ser una plataforma que sólo
            incluye elementos esenciales.
        </p>
      </div>
      <div deepest="True" id="div_5">
        <img class="div_5__img__image0" css_pattern="div_5" src="images/upload_test16golemimage21.png"/>
        <p class="div_5__p__text2" css_pattern="div_5">
            Para dar instrucciones al robot, no se requiere de computadoras o internet
            pues se usa un sistema de programación basado en tarjetas perforadas.
        </p>
      </div>
      <div deepest="True" id="div_7">
        <img class="div_7__img__image1" css_pattern="div_7" src="images/upload_test16golemimage31.png"/>
        <p class="div_7__p__text3" css_pattern="div_7">
            La construcción se hace con cartón, lo que permite hacer proyectos de
            todo tipo a un bajo costo.
        </p>
      </div>
    </div>
  </section>
```

Finally, implementing the orphan element insertion method addresses text elements not originally contained within any 'div'. The system integrates these elements by creating new sections or assigning them to appropriate existing sections, thereby enhancing the XML structure:

```
<section id="section_1">
    <p>
        Róbotica Golem, es una opción perfecta para los salones de clases. Las escuelas
        pueden implementar cursos de robótica sin realizar una inversión tan grande,
        esto gracias a tres estrategias clave:
    </p>
    <div id="div_8">
      <div deepest="True" id="div_6">
        <img class="div_6__img__image2" css_pattern="div_6" src="images/upload_test16golemimage11.png"/>
        <p class="div_6__p__text1" css_pattern="div_6">
            El costo del kit de robótica es mínimo al ser una plataforma que sólo
            incluye elementos esenciales.
        </p>
      </div>
      <div deepest="True" id="div_5">
        <img class="div_5__img__image0" css_pattern="div_5" src="images/upload_test16golemimage21.png"/>
        <p class="div_5__p__text2" css_pattern="div_5">
            Para dar instrucciones al robot, no se requiere de computadoras o internet
            pues se usa un sistema de programación basado en tarjetas perforadas.
```

```
      </p>
    </div>
    <div deepest="True" id="div_7">
      <img class="div_7__img__image1" css_pattern="div_7" src="images/upload_test16golemimage31.png"/>
      <p class="div_7__p__text3" css_pattern="div_7">
          La construcción se hace con cartón, lo que permite hacer proyectos de
          todo tipo a un bajo costo.
      </p>
    </div>
  </div>
</section>
```

4.5.4 Ensuring Structural Consistency

As mentioned before the system's output is required to adhere to three key structural rules to ensure consistency in the XML tree structure:

1. Only 'section' elements should be direct children of the 'body' node.

2. All web elements must be enclosed within a 'div' container.

3. Any container without children should be removed.

This segment of the system validates that the XML code complies with these rules. There may be instances where the insertion of orphan elements disrupts the intended structure. However, the objective of this process is to confirm that the XML code maintains the correct structure. Consider the following example where a 'p' element was introduced into the structure as part of the orphan handling method:

```
<section id="section_1">
    <p>
        Róbotica Golem, es una opción perfecta para los salones de clases. Las escuelas
        pueden implementar cursos de robótica sin realizar una inversión tan grande,
        esto gracias a tres estrategias clave:
    </p>
    <div id="div_8">
      <div deepest="True" id="div_6">
        <img class="div_6__img__image2" css_pattern="div_6" src="images/upload_test16golemimage11.png"/>
        <p class="div_6__p__text1" css_pattern="div_6">
            El costo del kit de robótica es mínimo al ser una plataforma que sólo
            incluye elementos esenciales.
        </p>
      </div>
      <div deepest="True" id="div_5">
        <img class="div_5__img__image0" css_pattern="div_5" src="images/upload_test16golemimage21.png"/>
        <p class="div_5__p__text2" css_pattern="div_5">
            Para dar instrucciones al robot, no se requiere de computadoras o internet
            pues se usa un sistema de programación basado en tarjetas perforadas.
        </p>
      </div>
      <div deepest="True" id="div_7">
        <img class="div_7__img__image1" css_pattern="div_7" src="images/upload_test16golemimage31.png"/>
        <p class="div_7__p__text3" css_pattern="div_7">
            La construcción se hace con cartón, lo que permite hacer proyectos de
            todo tipo a un bajo costo.
        </p>
      </div>
    </div>
  </section>
```

Post-application of the consistency checker method, the structure is modified to align with the stipulated rules:

```
<section id="section_1">
  <div id="new_div_1">
    <p>
        Róbotica Golem, es una opción perfecta para los salones de clases...
    </p>
  </div>
  <div id="div_8">
    <!-- Nested div elements -->
  </div>
</section>
```

In this revised structure, the previously orphaned 'p' element is now appropriately enclosed within a 'div' container ('new_div_1'), ensuring compliance with the second rule. This process effectively corrects any deviations from the established structural rules, thereby maintaining the integrity and consistency of the XML code.

4.5.5 Generating Dedicated 'div' Containers for Images

This step involves creating a separate 'div' container for each image within the XML tree structure. The aim for assigning a distinct 'div' to each image is to facilitate better relationship handling and styling. By enclosing images in their own containers, the system can apply styles to these containers, enhancing the practicality of layout and positioning of the images.

Previously, the XML structure might have looked like this:

```
<div deepest="True" id="div_7" row="False">
    <img class="div_7__img__image1" css_pattern="div_7" src="IMG_PATH"/>
    <p class="div_7__p__text3" css_pattern="div_7">
        La construcción se hace con cartón...
    </p>
</div>
```

After implementing this method, the structure is modified as follows:

```
<div deepest="True" id="div_7" row="False">
    <div css_pattern="div_7" id="div-image1">
        <img class="div-image1__img__image1" css_pattern="div-image1"
            src="IMG_PATH"/>
    </div>
    <p class="div_7__p__text3" css_pattern="div_7">
        La construcción se hace con cartón...
    </p>
</div>
```

4.5.6 Optimizing Web Layouts: Strategic Div Container Insertion for Flexbox Use.

One of the most complex tasks for the system involved correctly positioning web elements. This task required not only inserting web elements in the appropriate order but also ensuring they were placed within suitable containers. Additionally, the structure needed to be compatible with a flexible box (flexbox) layout to facilitate the creation of a flexible grid layout. This process demanded attention to the hierarchical arrangement of elements and their respective containers. The goal was to establish a structure that could efficiently leverage flexbox properties. Flexbox provides a more efficient way to lay out, align, and distribute space among items in a container, even when their size is unknown or dynamic.

The system had to ensure that the elements were not only sequentially correct but also nested within the right containers. These containers then had to be organized in a way that aligns with the principles of flexbox, allowing for a responsive and adaptable grid layout.

Flexbox, formally known as the Flexible Box Module, is a contemporary layout model in CSS3. It is designed to improve the way designers align and distribute space among items within a container, especially when their sizes are dynamic or unknown. This layout mode is an efficient tool for developing responsive web designs, as it ensures that elements behave predictably when the page layout must accommodate different screen sizes and display devices. Flexbox enables the creation of flexible grids through a series of container and item-specific properties. When an element is designated as a Flexbox container (using display: flex;), it activates Flexbox properties for its child elements. These properties include justify-content for horizontal alignment, align-items for vertical alignment, and align-self for individual item alignment.

Flexbox presents several advantages over traditional grid systems. Firstly, it simplifies the process of creating one-dimensional layouts, either as rows or columns, making it particularly useful for UI components and smaller-scale layouts. Its superior alignment and space distribution capabilities are especially beneficial when dealing with dynamically sized or unknown item dimensions. Flexbox also enhances responsive design, allowing items to adjust automatically to various screen sizes without requiring explicit width or height values.

While specific flexbox functionalities such as 'justify-content' and 'align-items' are addressed in the styles extraction subsystem (Section: 4.6), a critical aspect under consideration is the 'flex-direction' property (Figure: 4.30). This property, which determines the layout direction of flex items as either a row or a column. The system must analyze the arrangement of the inserted elements to infer whether they should be treated as a row or a column. This analysis is essential to ensure that the layout accurately reflects the intended design and functional requirements.

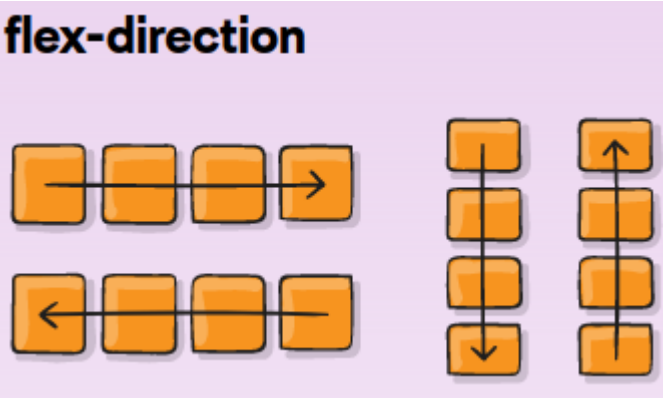


Figure 4.30: Flexbox: flex-direction property.

Developing a logic to integrate new 'div' elements within the XML structure was a significant challenge. The correct placement of elements within appropriate containers is crucial; otherwise, the rendered HTML and CSS could lead to unsatisfactory visual outcomes. The inference method does more than just determine which 'div' should have a 'row' or 'column' flex property; it also evaluates the existing structure. It then proceeds to create new 'div' containers where necessary, ensuring that the application of the flexbox layout is not only possible but also visually coherent and functionally effective. This method is integral to producing a clean, organized, and professionally styled web layout.

The following image (Figure: 4.31) illustrates the application of this inference method, in the example provided, the red rectangle illustrates a 'div' container that needs to be created. This container will enable the application of a 'flex-direction' property with a value of 'column'. This structural setup is crucial for ensuring that the content within the red rectangle is displayed correctly, adhering to the design that requires a stacked, columnar layout.

Creating such a ‘div’ container allows for greater control over the presentation and responsiveness of the content on various screen sizes, as it will maintain the vertical order regardless of the width of the display area.

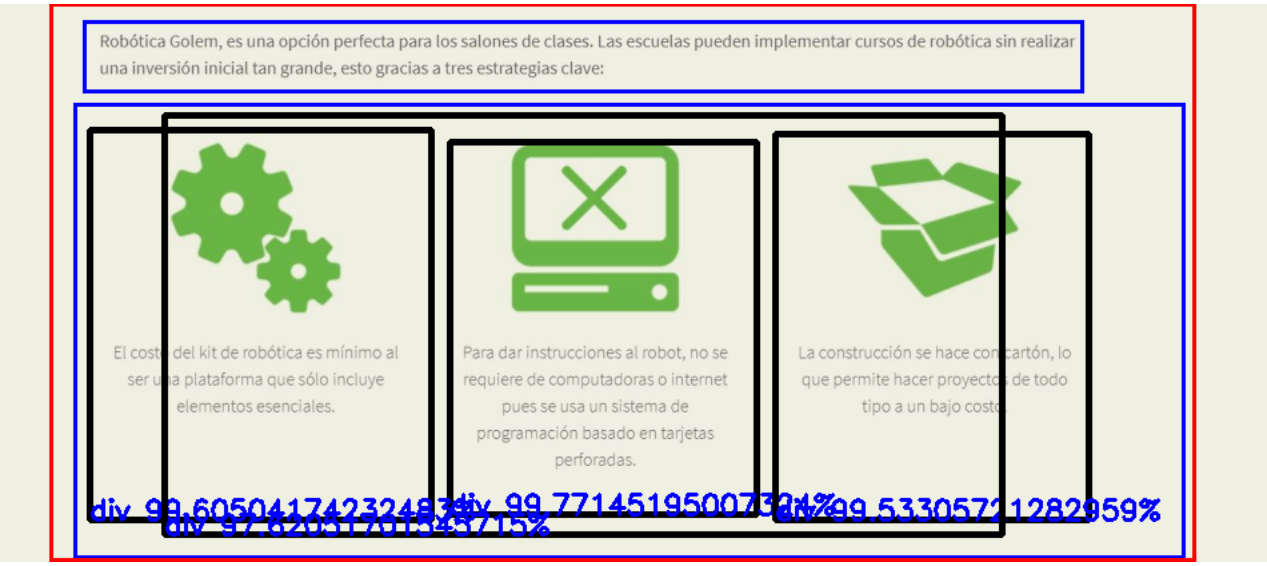


Figure 4.31: Illustration of New ‘div’ Container Implementation for Column Flex Direction.

The application of the inference method aims to refine the XML code structure, as demonstrated below. Notably, the ‘new-column-div-1’ is introduced to systematically organize the child ‘div’ elements. Furthermore, each ‘div’ element is equipped with an attribute named ‘row’. This attribute is important for the system to discern whether to apply a ‘row’ or ‘column’ flex direction to a particular ‘div’. This setup greatly simplifies the subsequent extraction of styles, thereby facilitating the generation of the corresponding CSS3 styles.

```
<section id="section_1">
  <div id="new-column-div-1" row="False">
    <div id="new_div_1" row="False">
        <p>
          Róbotica Golem, es una opción perfecta para ...
        </p>
    </div>
    <div id="div_8" row="True">
      <!-- Nested div elements -->
    </div>
  </div>
</section>
```

Here is another example, in the displayed image (Figure: 4.32), the red rectangles demonstrate the need for inserting new 'div' elements to conform to the flexbox layout design. While the primary 'div' container possesses a 'row' attribute, suggesting a horizontal arrangement, the child elements are currently positioned in a column format. To reposition the buttons and links—labeled "Información" and "Comprar" into a row, the introduction of new 'div' containers for each button-link set is essential. These newly created 'div' containers will realign the button and link pairs into a horizontal row, in compliance with flexbox design rules, thus preserving the intended visual and functional structure of the webpage.

Figure 4.32: Illustration of New 'div' Containers Implementation for Row Flex Direction.

The results are demonstrated in the following XML tree structure:

```
<section id="section_1">
  <div id="div-1" row="True">
    <!-- First child div with Laser printer details -->
    <div id="div-2" row="False">
      <div id="div-2-image">
        <img src="path_to_laser_printer_image"/>
      </div>
      <p>Laser</p>
      <div id="new-row-div-1" row="True">
        <button>Información<button>
        <a href="link_to_purchase">Comprar</a>
      </div>
    </div>
    <!-- Second child div with Ink and toner details -->
    <div id="div-3" row="False">
      <div id="div-3-image">
```

```
        <img src="path_to_ink_toner_image"/>
      </div>
      <p>Tinta y tóner</p>
      <div id="new-row-div-2" row="True">
        <button>Información<button>
        <a href="link_to_purchase_ink">Comprar</a>
      </div>
    </div>
    <!-- Third child div with large format printers details -->
    <div id="div-4" row="False">
      <div id="div-4-image">
        <img src="path_to_large_format_printer_image"/>
      </div>
      <p>Impresoras de gran formato</p>
      <div id="new-row-div-3" row="True">
        <button>Información<button>
        <a href="link_to_purchase_large_format">Comprar</a>
      </div>
    </div>
  </div>
</section>
```

In this revised code, div-2, div-3, and div-4 represent the individual children within the main container div-1. Each of these child div elements contains an image, a descriptive paragraph, and a new div for the 'Información' and 'Comprar' buttons. These inner div elements have the row="True" attribute to lay out the buttons in a row, conforming to the flexbox design indicated in the image (Figure: 4.32).

4.5.7 Adjusting Container Dimensions

Following the implementation of the previously mentioned methods into the XML tree structure, it is imperative to perform a reordering function. This step is crucial to verify and confirm that all web elements have been correctly integrated within the code.

Subsequently, the system proceeds to redefine the dimensions of all containers. The objective of this refinement is to ensure that each container precisely fits the elements it encloses. This precise adjustment facilitates the extraction of style attributes such as positioning, margins, and paddings.

The process implements a thorough examination of each element within every container. By calculating the necessary coordinates, the system can optimally resize the containers, reducing their original dimensions where possible. This leads to a cleaner, more defined structure that significantly aids the style extraction subsystem in its operations.

The following image (Figure: 4.33) illustrates that the initial dimensions of all elements include more space than necessary.

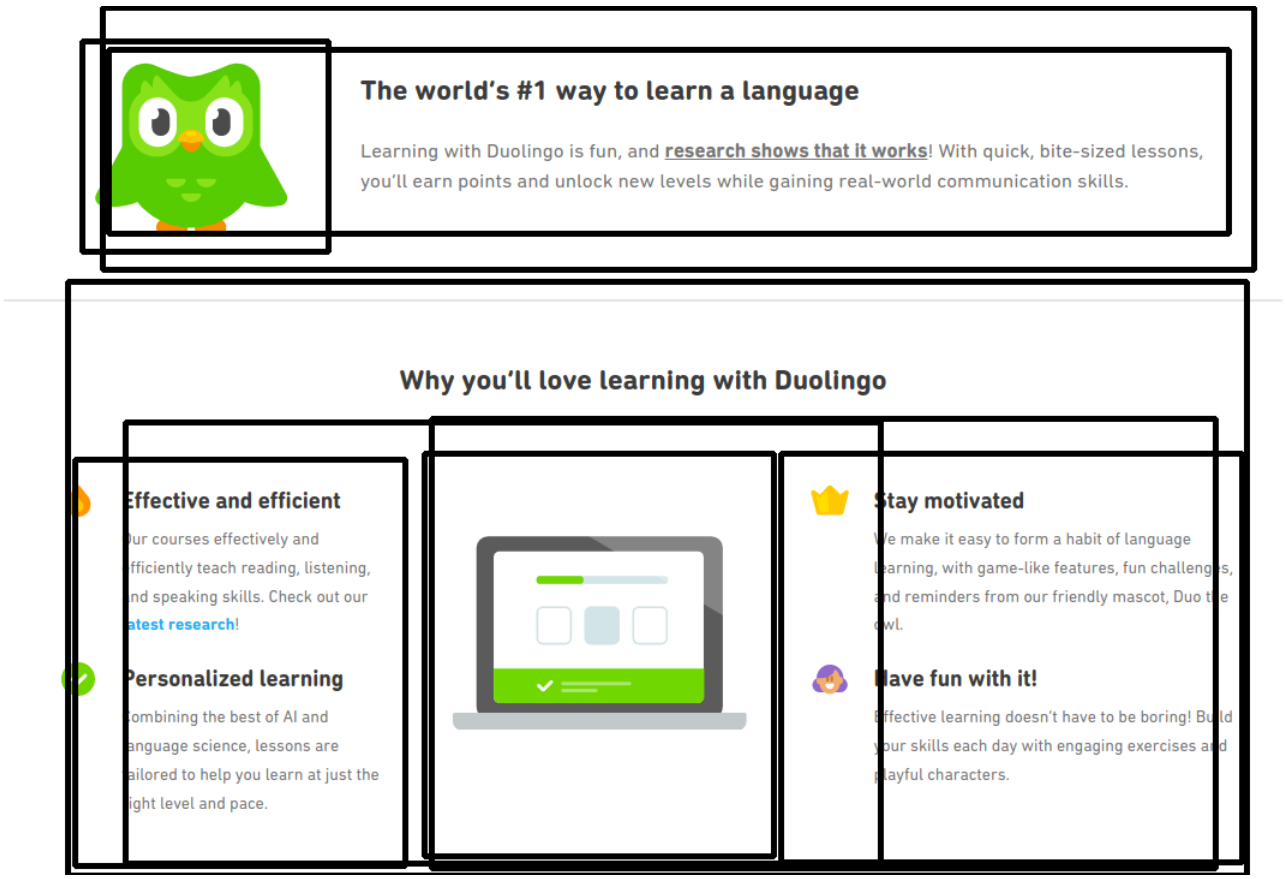


Figure 4.33: Illustration of New 'div' Containers Implementation for Row Flex Direction.

However, the system at this point where all web elements are identified and located, there is an opportunity to fine-tune the raw data coordinates. This refinement process involves adjusting the dimensions of the containers to more accurately reflect the content they enclose. The result image (Figure: 4.34) exemplifies the improved delineation of containers, where the boundaries are recalibrated to closely fit around the elements, eliminating the noise and excess space from the original detection. This leads to a more precise and clean representation of the web layout.

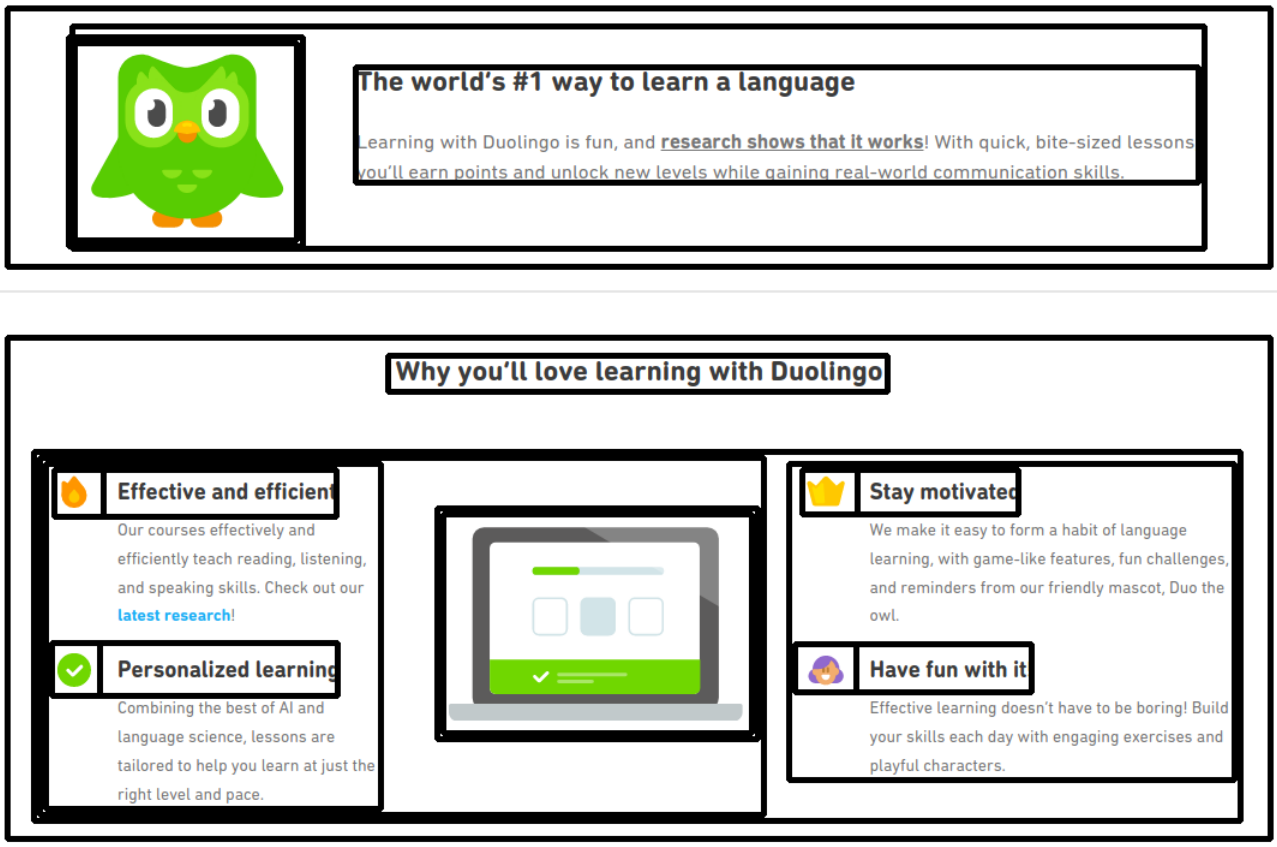


Figure 4.34: Illustration of New 'div' Containers Implementation for Row Flex Direction.

4.5.8 Contributions of the Inference and Correction Subsystem

The Inference and Correction Subsystem is the most important component of the proposed model for automated web code generation. Its contributions are crucial in ensuring the accuracy and reliability of the generated HTML and CSS code. This subsystem performs several key functions that enhance the overall performance and quality of the automated code generation process:

1. **Error Detection and Correction:** The subsystem identifies and corrects errors in the initial code generation, ensuring that the output is free from syntactical and logical mistakes.

2. **Optimization of Code Structure:** By refining the structure of the generated code, the subsystem enhances readability and maintainability. This includes improving the hierarchy of HTML elements and optimizing CSS for better performance and compatibility across different web browsers.

3. **Alignment with Design Intent:** The subsystem ensures that the generated code accurately reflects the original design intent. This involves aligning the visual elements and layout specified in the input image.

4. **Enhancement of Model Predictions:** The subsystem applies advanced inference techniques to enhance the predictions made by the convolutional neural network (CNN). This results in more precise identification and classification of web elements, leading to higher quality code generation.

In summary, the Inference and Correction Subsystem is vital for achieving the high standards of accuracy, performance, and design fidelity required for automated web code generation. Its contributions ensure that the final product is robust, reliable, and true to the original design.

4.6 Styles Extractor Subsystem: Deriving CSS3 Attributes from Webpage Image Design

4.6.1 Style Extraction Using BEM Nomenclature

The style extractor subsystem employs BEM (Block, Element, Modifier) nomenclature for CSS components to enhance clarity in style application. This naming convention facilitates the association of style rules with corresponding ID and class attributes defined within the XML code.

The BEM methodology segments a webpage into three primary components: Blocks, Elements, and Modifiers. A 'Block' refers to a standalone entity that is meaningful on its own, like a header, footer, or a menu. An 'Element' is a part of a Block that performs a specific function, such as a button in a form block. A 'Modifier' is a flag on a Block or an Element used to change appearance, behavior, or state. In practice, BEM employs a specific naming convention.

BEM minimizes the risk of cascading style conflicts. By using unique and specific class names, BEM ensures that styles do not unintentionally affect unrelated parts of the website. This specificity is crucial in maintaining a consistent look and feel across a website. BEM is a powerful and efficient methodology for organizing and maintaining CSS3 code. Its emphasis on readability, reusability, and specificity makes it a superior choice for both small and large-scale web development projects.

Consider the following examples where BEM nomenclature is utilized:

```
#div_10 {
    align-items: flex-start;
    display: flex;
    flex-direction: column;
    justify-content: center;
    text-align: left;
}
```

```
.div_10__h1__text1 {
    color: rgb(63, 62, 52);
    font-size: 41px;
}
```

Through the implementation of a BEM Generator class (Appendix-B: B.11), the subsystem dynamically appends styles for blocks, elements, and modifiers. This class makes it easier to add styles to CSS.

To create a CSS block, the createBlock method is used. This method requires specific parameters like container's key and styles. An example of usage is:

```
bem_gen.createBlock(
    div_key,
    prefix="#",
    styles={
        "width": f`{class_div.width}px',
␣␣␣␣␣␣␣␣"display":␣"flex",
␣␣␣␣␣␣␣␣"flex-direction":␣"column",
␣␣␣␣␣␣␣␣"justify-content":␣"center",
␣␣␣␣␣␣␣␣"align-items":␣f`{align_items}',
        "text-align": f`{text_align}'
␣␣␣␣}
)
```

Elements can be added using the createElement method. Here is an example:

```
bem_gen.createElement(
    "header",
    "checkbox",
    styles={
        "display": "none",
    }
)
```

Modifiers are added using the createModifier method. This method allows the addition of more specific styles to blocks. For example:

```
bem_gen.createModifier(
    block_key,
    class_image.tag,
    key,
    styles={
        "width": f`{class_image.width}px',
␣␣␣␣␣␣␣␣"height":␣f`{class_image.height}px'
    }
)
```

This approach not only simplifies the process of style application but also maintains a standardized format across the generated CSS code, ensuring consistency.

4.6.2 Enhancing Style Extraction for Web Containers (Footers, Sections, Headers and Divs)

The process of extracting styles for various web containers, such as footers, sections, headers, and individual 'div' elements, is consolidated into a singular, efficient method. The designated method is responsible for accurately identifying and extracting the detailed specifications of each container component, ensuring a comprehensive representation of the container's styling in the generated code.

For instance, to determine the background color of the container, a CSS Helper Extractor class (Appendix-B: B.14) is employed. This class leverages the KMeans algorithm to isolate the container's dominant color, which is typically used for the background, and the second most predominant color, potentially applied as the text color. A critical step in this process is the exclusion of images within the container to prevent their influence on the background color analysis.

Furthermore, the Header, Footer, Section and Div Analyzer class is designed to perform operations on elements, extracting essential style information such as margins, paddings, as well as width and height attributes. This enables a comprehensive understanding and reconstruction of the container's layout and stylistic features.

The extraction of background colors for containers plays a critical role, but equally important is the implementation of flexbox properties to mirror the visual appearance of the original webpage design. The system simplifies this process by utilizing a pre-existing method that detects the flex direction of containers. If the determined flex direction is 'column', then 'flex-direction: column' is explicitly set within the styles. Otherwise, the default 'row' direction is assumed, and no additional style attribute is necessary.

After establishing the flex direction, the system can then proceed to assess and apply the appropriate 'justify-content' property to fine-tune the layout alignment as per the design specifications.

One of the key properties in Flexbox is justify-content. This property is used to align and distribute space among items within a Flex container along the main axis (horizontally for a row, vertically for a column). It plays a significant role in controlling the spacing and alignment of items when the container has extra space beyond that required to display its items. The justify-content property offers several values to control the distribution of space:

- **flex-start**: This aligns items to the start of the container.
- **flex-end**: It aligns items to the end of the container.

- **center**: This centers items in the container.
- **space-between**: It distributes items evenly, ensuring that the first item is on the start line and the last item is on the end line.
- **space-around**: Items are distributed evenly with equal space around them.
- **space-evenly**: Items are distributed so that the spacing between any two items (and the space to the edges) is equal.

The justify-content property is particularly useful for creating flexible and responsive layouts. It allows developers to easily control the alignment and spacing of elements within a Flex container, making it invaluable for both simple and complex web layouts. The ability to distribute space dynamically means that layouts created with Flexbox easily adapt to different screen sizes, enhancing the responsiveness of web designs. Moreover, the simplicity and precision of justify-content enable developers to create visually appealing and user-friendly interfaces without the need for complex calculations or additional markup. This simplifies the development process and leads to cleaner, more maintainable code.

Within the CSS Analyzer class (Appendix-B: B.12), a method named getJustifyContent has been crafted. This function is crucial in determining the 'justify-content' property for each container, a specification deduced from the alignment and distribution of child elements within the container. This method analyzes the positioning of child elements to ascertain the optimal setting for the 'justify-content' CSS property, ensuring that the layout corresponds accurately with the intended design of the web elements.

Another essential feature is determining alignment of container items, the CSS Analyzer (Appendix-B: B.12) plays an important role in identifying the 'align-items' property for each container. The align-items property in Flexbox is used to align Flex items vertically within a Flex container when the flex direction is a row, or horizontally when it is a column. This property is crucial for controlling the vertical alignment of items in a row or the horizontal alignment in a column, particularly when the items have different sizes or when there is extra space in the container.

The `align-items` property provides several values for alignment:

- **stretch**: This is the default value. It stretches the Flex items to fill the container along the cross axis.
- **flex-start**: Aligns items to the start of the container along the cross axis.
- **flex-end**: Aligns items to the end of the container.
- **center**: Centers items along the cross axis.

- **baseline**: Aligns items based on their baselines.

The align-items property is particularly advantageous for creating consistent and uniform layouts. It allows for precise alignment of items, regardless of their size, ensuring a clean and orderly presentation. This is especially useful in responsive web design, where the layout must adapt to different screen sizes and orientations. Moreover, the simplicity of the align-items property enables developers to achieve complex alignments with minimal code, enhancing the efficiency of the development process.

Similar to the process of extracting 'justify-content', this feature's value is ascertained by the vertical alignment and ordering of the container's child elements. Utilizing the logical framework established for 'justify-content', the CSS Analyzer (Appendix-B: B.12) evaluates the distribution and orientation of the children within the container to accurately determine the appropriate 'align-items' setting. This ensures that the vertical alignment of the content matches the intended design.

The culmination of the style extraction process can produce results such as the following CSS definitions:

```
#div-section_3 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    width: 436px;
}
```

Alternatively, for a column within a container, the styles might appear as:

```
#div_10-col_1 {
    align-items: flex-start;
    display: flex;
    flex-direction: column;
    justify-content: center;
    margin-right: 5px;
    text-align: left;
    width: 702px;
}
```

These CSS rules define the visual structure and layout for each container, aligning them with the design of the webpage.

4.6.3 Enhancing Style Extraction for Web Elements

After defining container styles, the next step involves styling a variety of web elements, including paragraphs, headings, images, buttons, links, and more. Each of these elements has unique characteristics and requirements, making a one-size-fits-all approach to styling impractical. To address this, the CSS Feature Extractor (Appendix-B: B.13) plays a crucial role. It identifies common style properties that apply across different web elements, such as width, height, and margins. This analysis helps create a consistent and effective styling strategy, ensuring that the specific needs of each element are addressed, while preserving the cohesive visual of the website's design.

For example when it comes to text-specific styles, such as font size and color, a separate Text Analyzer is employed. This specialized tool meticulously extracts and applies these attributes to ensure that each text element on the web page accurately reflects the intended design. This focused approach guarantees the integrity and coherence of the website's visual presentation.

The following CSS code is an example of applying CSS styles to different elements, like texts, images, buttons, and navigation items.

```
.div_7__h2__text3 {
    color: rgb(159, 150, 157);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 4px;
}

.div-image3__img__image3 {
    height: 37px;
    width: 46px;
}

.div_4-row_0__button__button0 {
    background-color: rgb(255, 254, 254);
    border-radius: 0px;
    color: rgb(255, 156, 68);
    font-size: 16px;
    height: 59px;
    padding: -25px -105px;
    width: 283px;
}

.nav0__ul {
    align-items: center;
    display: flex;
    justify-content: space-between;
    list-style: none;
    width: 100%;
}
```

4.6.4 Integration of Normalize.css into the System

The use of a CSS reset is a standard practice in web development to ensure that browser-specific styling does not interfere with the intended design of a webpage. To address this issue, the system incorporates Normalize.css, a modern alternative to traditional CSS resets. This integration is aimed at enhancing the consistency of the webpage's appearance across different browsers, ensuring that the design and layout remain unaffected by varying browser defaults. The adoption of Normalize.css is a strategic step to enhance the robustness and cross-browser compatibility of the system's generated code."

One of the main advantages of using Normalize.css is the enhanced cross-browser consistency it offers. This simplifies the process of developing and testing websites across different browsers and devices, as it minimizes the amount of browser-specific hacks or adjustments required. Moreover, Normalize.css contributes to better usability. It preserves standard behaviors such as the clickable nature of buttons and ensures that elements like form fields are easily readable and accessible.

To implement Normalize.css (Appendix-B: B.15), it is usually linked at the beginning of the CSS file or included directly within the project's stylesheets. This sets the stage for further custom styling, allowing developers to design their layouts and typography without worrying about browser default styles.

In summary, Normalize.css is a powerful tool to normalizing styles, as opposed to resetting them, provides a more practical and efficient foundation for website styling.

4.7 Analyzing the Outcomes: Results from the Proposed Model and Review of the Generated Code

This section presents a comparative analysis of the outcomes from the proposed model. It includes pairs of images for each case study: the upper image demonstrates the original webpage design, while the lower image displays the result of the code rendered by the system (Figures: 4.35, 4.36, 4.37, 4.38 and 4.39). This visual comparison facilitates a thorough evaluation of the code generation's effectiveness in replicating the intended design and highlights areas requiring enhancement as well as those of notable success.

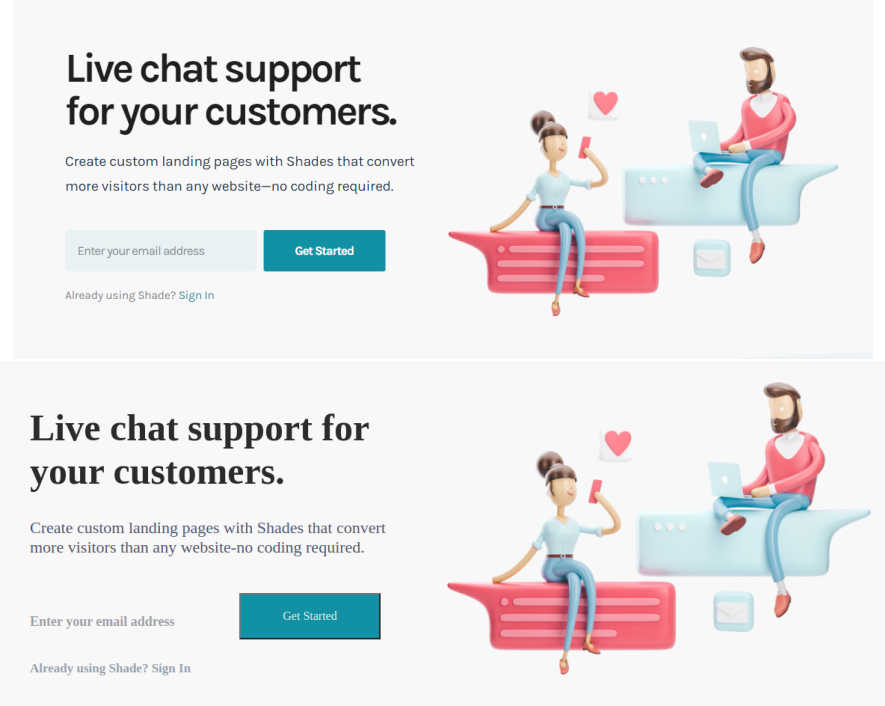


Figure 4.35: Example 1. Evaluating the Proposed Model For Automated Web Code Generation. Comparison of the Original Design (Upper Image) and the Rendered Output from the System-Generated Code (Lower Image)

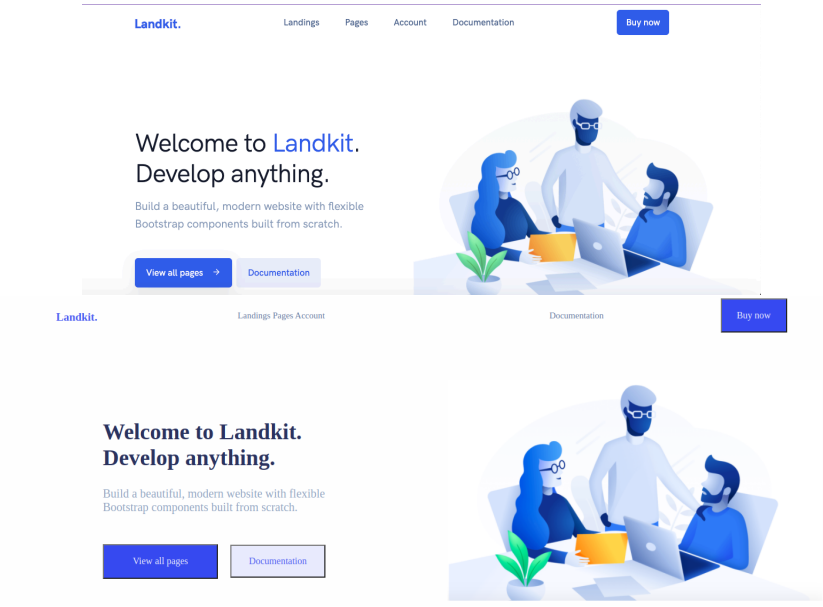


Figure 4.36: Example 2. Evaluating the Proposed Model For Automated Web Code Generation. Comparison of the Original Design (Upper Image) and the Rendered Output from the System-Generated Code (Lower Image)

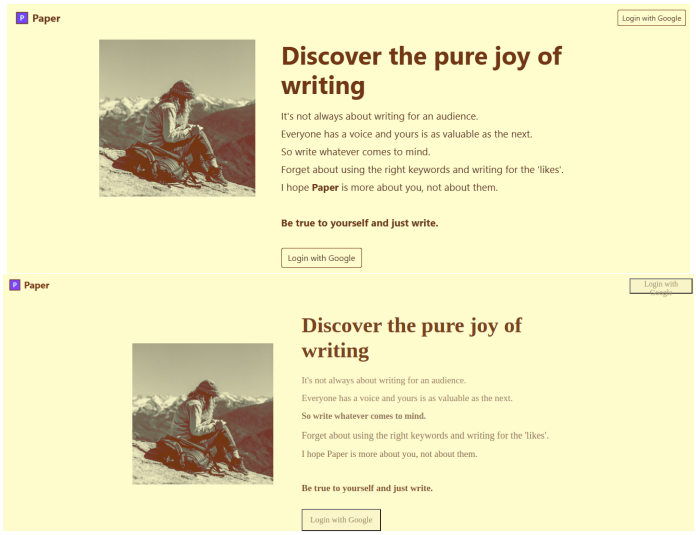


Figure 4.37: Example 3. Evaluating the Proposed Model For Automated Web Code Generation. Comparison of the Original Design (Upper Image) and the Rendered Output from the System-Generated Code (Lower Image)

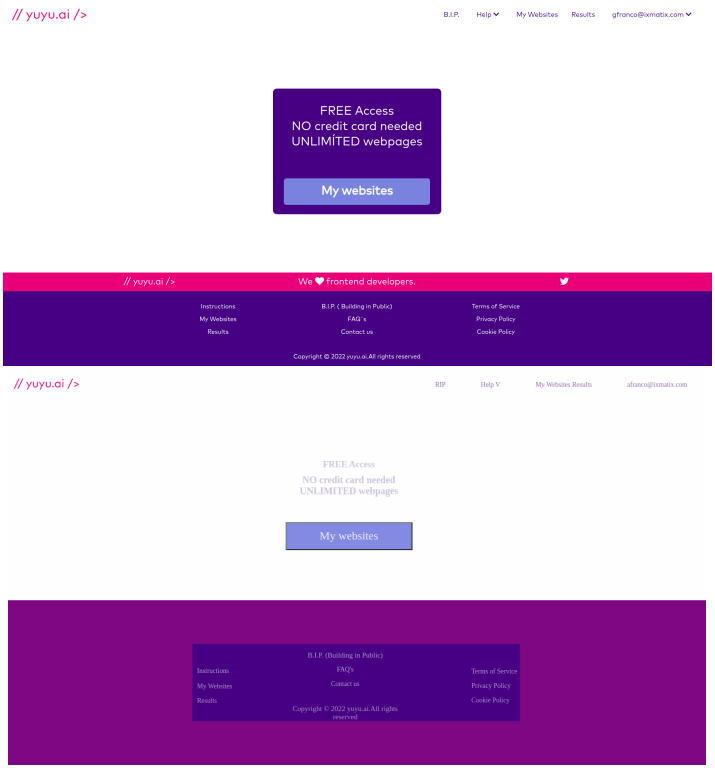


Figure 4.38: Example 4. Evaluating the Proposed Model For Automated Web Code Generation. Comparison of the Original Design (Upper Image) and the Rendered Output from the System-Generated Code (Lower Image)

Now additionally the following example (Figure: 4.39), also illustrates the original web design and the corresponding result image, but it includes the HTML5 and CSS3 code generated by the system.

Figure 4.39: Example 5. Evaluating the Proposed Model For Automated Web Code Generation. Comparison of the Original Design (Upper Image) and the Rendered Output from the System-Generated Code (Lower Image)

HTML5 code generated:

```
<!DOCTYPE html>
<html lang="en">
<!-- This is the <head> tag, where all meta data is defined. -->
    <head>
        <meta charset="utf-8"/>
        <meta content="width=device-width, initial-scale=1, shrink-to-fit=no" name="viewport"/>
        <link href="normalize.css" rel="stylesheet"/>
        <link href="styles_2.css" rel="stylesheet"/>
        <title>2</title>
    </head>
<!-- This is the <body> tag, where the whole webpage is allocated. -->
    <body>
<!-- This is the <head> tag, where all meta data is defined. -->
```

```
        <header id="header0">
            <input class="header__checkbox" id="header__checkbox" type="checkbox"/>
            <label class="icon_menu" for="header__checkbox">
                <i class="fas␣fa-align-justify"> </i>
            </label>
            <div id="header0-div-image1">
                <img class="logo␣header__img__image1" css_pattern="header" id="image1" src="images/upload_test2openknowledgelogo11.png"/>
            </div>
            <nav class="nav0" css_pattern="header0">
                <ul class="nav0__ul" css_pattern="nav0">
                    <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item0">
                        <a class="nav-link␣nav0__a__txtnav0-0" css_pattern="nav0" href="#">
                            Places Datasets Download Insights Methodology About Help
                        </a>
                    </li>
                </ul>
            </nav>
            <div id="header0-div-image0">
                <img class="logo␣header__img__image0" css_pattern="header" id="image0" src="images/upload_test2openknowledgelogo21.png"/>
            </div>
        </header>
<!-- This is the <section> tag, corresponding to "section_1". -->
        <section id="section_1">
            <div id="div_12" row="False">
                <div id="div_10" row="False">
                    <h1 class="div_10__h1__text1" css_pattern="div_10">TRACKING THE STATE OF OPEN GOVERNMENT</h1>
                </div>
                <div id="div_9" row="False">
                    <h2 class="div_9__h2__text2" css_pattern="div_9">DATA</h2>
                    <p class="div_9__p__text3" css_pattern="div_9">
                        The Global Open Data Index provides the most
                        comprehensive snapshot available of the state of open
                        government data publication
                    </p>
                </div>
                <div id="div_7" row="True">
                    <div id="div_11" row="False">
                        <div css_pattern="div_11" id="div_11-row_0" row="True">
                            <h2 class="div_11-row_0__h2__text5" css_pattern="div_11-row_0">Compare countries</h2>
                            <h2 class="div_11-row_0__h2__text8" css_pattern="div_11-row_0">Discuss findings</h2>
                        </div>
                        <h2 class="div_11__h2__text6" css_pattern="div_11">Ranked table and map views</h2>
                        <div css_pattern="div_11" id="div_11-row_2" row="True">
                            <h2 class="div_11-row_2__h2__text7" css_pattern="div_11-row_2">of participating countries.</h2>
                            <h2 class="div_11-row_2__h2__text9" css_pattern="div_11-row_2">Discuss your findings in our forum.</h2>
                        </div>
                    </div>
                    <div id="div_8" row="False">
                        <h2 class="div_8__h2__text10" css_pattern="div_8">Get the insights</h2>
                        <h2 class="div_8__h2__text11" css_pattern="div_8">Insights from local and</h2>
                        <h2 class="div_8__h2__text12" css_pattern="div_8">thematic perspectives.</h2>
                    </div>
                </div>
            </div>
        </section>
    </body>
</html>
```

CSS3 code generated:

```
/*******************************/
/* Beginning of common styles. */
/*******************************/

/* Beginning of styles for block body. */


/* Beginning of styles for block button. */
button {
    display: block;
}


/* Beginning of styles for block div_10. */
#div_10 {
    align-items: flex-start;
    display: flex;
    flex-direction: column;
    justify-content: center;
    text-align: left;
    width: 1181px;
}


.div_10__h1__text1 {
    color: rgb(63, 62, 52);
    font-size: 41px;
    margin-bottom: 0em;
}


/* Beginning of styles for block div_11. */
#div_11 {
    align-items: flex-start;
    display: flex;
    flex-direction: column;
    justify-content: center;
    text-align: left;
    width: 482px;
}


.div_11__h2__text6 {
    color: rgb(116, 108, 64);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 7px;
}


/* Beginning of styles for block div_11-row_0. */
#div_11-row_0 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-right: 0px;
    width: 448px;
}


.div_11-row_0__h2__text5 {
    color: rgb(102, 96, 58);
    font-size: 19px;
    margin-bottom: 0em;
}


.div_11-row_0__h2__text8 {
    color: rgb(92, 89, 58);
    font-size: 17px;
    margin-bottom: 0em;
}


/* Beginning of styles for block div_11-row_2. */
#div_11-row_2 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-right: 0px;
    margin-top: 12px;
    width: 492px;
}


.div_11-row_2__h2__text7 {
    color: rgb(127, 116, 64);
    font-size: 16px;
    margin-bottom: 0em;
}

.div_11-row_2__h2__text9 {
    color: rgb(124, 116, 64);
    font-size: 16px;
    margin-bottom: 0em;
}


/* Beginning of styles for block div_12. */
#div_12 {
    align-items: center;
    display: flex;
    flex-direction: column;
    justify-content: center;
    text-align: center;
    width: 1191px;
}


/* Beginning of styles for block div_7. */
#div_7 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-around;
    margin-top: 9px;
    width: 914px;
}


/* Beginning of styles for block div_8. */
#div_8 {
    align-items: flex-start;
    display: flex;
    flex-direction: column;
    justify-content: center;
    text-align: left;
    width: 154px;
}


.div_8__h2__text10 {
    color: rgb(94, 91, 58);
```

```
    font-size: 18px;
    margin-bottom: 0em;
}

.div_8__h2__text11 {
    color: rgb(118, 109, 64);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 10px;
}

.div_8__h2__text12 {
    color: rgb(129, 120, 64);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 3px;
}


/* Beginning of styles for block div_9. */
#div_9 {
    align-items: center;
    display: flex;
    flex-direction: column;
    justify-content: center;
    margin-top: 13px;
    text-align: center;
    width: 844px;
}


.div_9__h2__text2 {
    color: rgb(62, 61, 52);
    font-size: 40px;
    margin-bottom: 0em;
}


.div_9__p__text3 {
    color: rgb(91, 86, 57);
    font-size: 26px;
    margin-bottom: 0em;
    margin-top: 51px;
}


/* Beginning of styles for block header. */

.header__checkbox {
    display: none;
}


.header__img__image0 {
    object-fit: cover;
    width: 166px;
}

.header__img__image1 {
    object-fit: cover;
    width: 237px;
}


/* Beginning of styles for block header0. */
#header0 {
    align-items: center;
    background-color: rgb(232, 206, 69);
    display: flex;
    justify-content: space-around;
    padding: 1px 0px;
    width: 100%;
}


/* Beginning of styles for block icon_menu. */
.icon_menu {
    cursor: pointer;
    display: none;
    font-size: 25px;
}


/* Beginning of styles for block nav0. */
.nav0 {
    align-items: center;
    display: flex;
    justify-content: center;
    width: 617px;
}


.nav0__a__txtnav0-0 {
    color: rgb(114, 105, 58);
    font-size: 17px;
    margin-bottom: 0.5em;
    margin-top: 0.5em;
    text-decoration: none;
}

.nav0__ul {
    align-items: center;
    display: flex;
    justify-content: space-between;
    list-style: none;
    width: 100%;
}


/* Beginning of styles for block section_1. */
#section_1 {
    align-items: center;
    background-color: rgb(234, 209, 73);
    display: flex;
    justify-content: center;
    padding: 84px 0px;
    width: 100%;
}


/*****************************/
/* Beginning of media styles. */
/*****************************/

/* Beginning of styles for 480px width. */
@media(max-width: 480px) {
    #div_11-row_0 {
        width: 100%;
    }
    #div_11-row_2 {
        width: 100%;
    }
    #div_8 {
        width: 100%;
    }
}

/* Beginning of styles for 800px width. */
@media(max-width: 800px) {
    #div_11 {
        width: 100%;
    }
}
```

```
/* Beginning of styles for 1200px width. */
@media(max-width: 1200px) {
    #div_10 {
        width: 100%;
    }
    #div_12 {
        width: 100%;
    }
    #div_7 {
        width: 100%;
    }
    #div_9 {
        width: 100%;
    }
    #header0 {
        flex-direction: row-reverse;
        justify-content: space-between;
    }
    #header0-div-image0 {
        margin-left: 15px;
    }
    #header0-div-image1 {
        margin-left: 15px;
    }
    .header__checkbox:checked ~ .nav0 {
        display: flex;
        flex-direction: column;
    }
    .icon_menu {
        display: flex;
        margin-right: 15px;
    }
    .nav0 {
        display: none;
        position: absolute;
        width: 100%;
        top: 81px;
    }
    .nav0__ul {
        flex-direction: column;
        background-color: rgb(232, 206, 68);
    }
}
```

More detailed results and their evaluation will be presented in Quantifying Success: Results and Performance Metrics (Chapter: 6). Subsequently, a comprehensive discussion and conclusions based on these results will be featured in Conclusions (Chapter: 7).

5

Design and Implementation of an Image Similarity Evaluation System

5.1 Introduction

This study introduces an evaluation system that seeks to compare the visual accuracy of web pages generated automatically by a proposed system against their original designs. By converting the automatically generated code into an image and comparing it side-by-side with the original design image, it becomes possible to quantitatively measure the similarity between the two. Leveraging the power of autoencoders, this evaluation metric uses the cosine distance to determine the likeness. A result nearer to zero suggests that the code generated by the machine accurately reflects a web page that closely resembles the initial design.

In object detection, metrics such as the mAP score are fundamental for evaluating the performance of CNN models. However, for assessing the quality of systems that produce code automatically, the mAP metric is inadequate. It does not serve as an indicator of the performance of systems that generate web code, and it does not allow for a comparison of the efficacy of different code-generation methodologies. Such projects necessitate a dedicated metric to ascertain how closely the generated web code aligns with an original image design.

Traditional metrics, like the mAP score, cannot capture the relationships between web code and its visual design counterpart. This highlights the importance of a specialized system that evaluates the congruence between generated HTML5 and CSS3 code and the original design image. With a customized measurement set in place, it becomes possible to assess the effectiveness of a system's website generation capability and provide a corresponding score. This metric is crucial when introducing new features or adjustments, ensuring that proposed changes genuinely enhance system performance rather than degrade it. However, while this specific metric proficiently measures the congruence between code and design, it does not provide perspective on the inherent quality or optimization of the generated code.

One significant benefit of this evaluation system is its ability to compare various techniques that convert web design images into code. Regardless of the method employed, if a technique produces web code, it can be rendered into an image. This allows for a direct comparison to determine how similar the resulting image is to the original design source, providing a clear metric to gauge which system's output aligns most closely with the intended design.

5.2 State of art

Evaluating image similarity has been a focal point in computer vision for several years. Initially, methods such as SIFT (Scale-Invariant Feature Transform) [87] and HOG (Histogram of Oriented Gradients) [88] were prominent. SIFT is a technique that identifies and describes local features in images. It is particularly valuable because of its scale-invariant nature, which means it could detect features regardless of their size in the image. The method involves detecting keypoints (or interest points) in scale-space, assigning them orientations, and then generating descriptors based on local gradient histograms around these keypoints.

HOG, on the other hand, is a feature descriptor used primarily in object detection. The technique works by dividing an image into small regions, called cells, and then for each cell, a histogram of gradient directions (or orientations) is computed. These histograms are then normalized, which gives the descriptor its robustness to variations in illumination and shadowing. While both SIFT and HOG were revolutionary in their time, capturing local image descriptors effectively, they laid the groundwork for subsequent techniques. However, their performance in diverse scenarios highlighted the limitations of handcrafted features, pointing to the need for more adaptive and robust methods.

As deep learning began to reshape the landscape of computer vision, Convolutional Neural Networks (CNNs) emerged as the primary tool for image similarity tasks.

Pioneering architectures like the Siamese network [89] utilized two parallel networks to compare two distinct images. The twin networks shared the same parameters and weights, and their output feature vectors were combined to determine the similarity between the input images. The Triplet network [90], building upon this concept, introduced a third component: it operated on an anchor image, a positive sample (similar to the anchor), and a negative sample (different from the anchor). The objective was to make the anchor closer to the positive sample and farther from the negative sample in the feature space, thus enhancing the robustness of similarity comparisons.

The momentum of deep learning in image similarity grew as researchers extracted feature representations from intermediate layers of CNNs, achieving unprecedented results on benchmarks. However, the journey of deep learning in this domain has not been without challenges. Issues like lighting variations, pose differences, and occlusions persist. Addressing these, the research community has gravitated towards integrating attention mechanisms [91] and transformers [92], aiming to refine the representation learning process further.

Alongside the rapid progression of deep learning, traditional methods have maintained their significance. Consider the Root Mean Square Error (RMSE). RMSE quantifies the difference between two images by computing the square root of the average of squared differences between corresponding pixels. It serves as a straightforward and tangible metric, offering insights into pixel-by-pixel discrepancies. However, a notable limitation of RMSE is that it treats all deviations equally, often overlooking nuances that are evident to human observers. Thus, while RMSE is proficient at identifying raw pixel differences, it might not consistently align with human visual interpretation. Transitioning to perceptual metrics, the Structural Similarity Index Measure (SSIM) [93] has garnered attention. Unlike methods that focus merely on pixel values, SSIM delves into the inherent structures and patterns within images. It quantifies image quality by examining changes in structural information, luminance (brightness), and texture contrast. These elements are more in sync with how humans perceive and interpret images, making SSIM a more holistic metric.

Additionally, the Feature-based Similarity Index (FSIM) [94] further advances the concept of perceptual similarity. FSIM synergizes two primary components: phase congruency and image gradient magnitude. Phase congruency pertains to the alignment of features in an image across different scales, while image gradient magnitude refers to the rate of change in image intensity. By combining these elements, FSIM provides a robust assessment of similarity, retaining its efficacy even when images undergo distortions such as blurring or noise. Exploring a more theoretical angle, the Information Theoretic-based Statistic Similarity Measure (ISSM) taps into the realms of information theory. Instead of relying on direct pixel values or features, ISSM evaluates the shared information content between images. This measure can be particularly effective when the similarities are subtle or buried in complex patterns.

In recent years, autoencoders have emerged as a powerful tool in the domain of unsupervised learning, particularly in the area of dimensionality reduction and feature extraction [95]. Autoencoders are neural network architectures designed to encode input data into a compressed representation and then decode it back to its original form. The advantage of this approach lies in the encoded layer, which captures the most critical features of the input data. For image similarity tasks, the potential of autoencoders is vast [96]. By training an autoencoder on a dataset of images, the encoded layer can serve as a compressed representation of each image. When comparing two images, their respective encoded representations can be used to compute distances in this compressed space. This methodology offers an advantage in terms of computational efficiency, as distances are calculated in a reduced-dimensional space compared to the original image space [97]. Moreover, the encoded data, being a distilled representation, often captures the essence of the image, making it a robust metric for similarity.

Several variations of autoencoders have been proposed to enhance their performance and adaptability. One such adaptation is the Sparse Autoencoder [98], which, distinct from its traditional counterparts, operates under a unique principle of activating a limited set of neurons in its hidden layers. By imposing this sparsity constraint, the network not only ensures a prioritized representation of information but also enhances the interpretability of the encoded data, making the learned features more distinct and resistant to overfitting. Diverging from the deterministic nature of typical autoencoders, Variational Autoencoders (VAEs) [99] embrace a probabilistic approach. They model both the input data and the ensuing encoded representations as distributions. This distinctive trait allows VAEs to stand out in generating data by sampling from the learned distributions, rendering them invaluable for tasks that demand the coherent generation of new samples, such as in image synthesis.

In contrast, Denoising Autoencoders [100] prioritize robustness. Trained on intentionally corrupted versions of input data, their objective is the meticulous reconstruction of the pristine, unaltered data. This deliberate exposure to noise mandates the network to discern and capture the intrinsic patterns of the data, rather than superficial or noisy elements. Consequently, this form of autoencoder exhibits enhanced resilience to varied data corruptions, fortifying its generalization capabilities and providing a potent form of regularization in scenarios prone to data perturbations.

The use of autoencoders for image similarity is still an active area of research, with ongoing efforts to optimize their architectures, training methods, and applications. Their ability to capture the essential features of images in a compact representation positions them as a promising tool for future advancements in image similarity assessment.

5.2.1 Evaluation Systems in Pix2Code and Sketch2Code

Pix2Code Evaluation Metrics:
Pix2Code by Tony Beltramelli [13] employs the BLEU (Bilingual Evaluation Understudy) [101] score to evaluate the quality of the generated code. The BLEU score, originally developed for evaluating machine translation, compares the generated code with a reference code by analyzing the n-gram precision. An n-gram is a contiguous sequence of n items from a given sample of text or code. For example, a unigram (1-gram) consists of single items, a bigram (2-gram) consists of pairs of items, and a trigram (3-gram) consists of triplets of items. In the context of BLEU, these items are typically words or tokens. The BLEU score measures how many n-grams in the generated code appear in the reference code, providing a score between 0 and 1, where a higher score indicates better alignment with the reference.

Sketch2Code Evaluation Metrics:
Sketch2Code by Microsoft [102] evaluates the generated HTML and CSS code by rendering the output and performing a pixel-by-pixel comparison with the original hand-drawn design [71]. This approach ensures that the visual layout and styling closely match the intended design. Additionally, Sketch2Code uses Intersection over Union (IoU) to measure object detection accuracy, evaluating the overlap between detected elements and ground truth.

5.3 Similarity Evaluation System Proposal

5.3.1 Hypothesis and System Definition

Given the unique capability of autoencoders to compress and distill critical features of images into reduced-dimensional spaces, it is hypothesized that utilizing autoencoder-derived representations will yield superior performance in image similarity evaluation tasks. Specifically, these representations are expected to capture inherent patterns, structures, and nuances in images, allowing for more effective similarity measures compared to traditional pixel-based methods. This approach is anticipated to be particularly effective for comparing images with specific differences or those that vary in structural or contextual aspects, offering a robust metric that aligns closely with human visual interpretation.

To facilitate this assessment, the system proposed in this thesis converts the rendered output of the generated HTML5 and CSS3 code into an image. Importantly, this rendered image is created at the same resolution as the original design image, ensuring a consistent basis for comparison. There are multiple methodologies to achieve this image conversion.

For the scope of this research, the wkhtmltoimage tool was employed. wkhtmltoimage is a command line tool that renders HTML content into raster image formats. By feeding the generated web code into this tool, an accurate visual representation of the rendered webpage is obtained, encapsulating all the design elements, layout structures, and color schemes.

Determining similarity between two images, especially in the context of web design, is a multifaceted challenge. Factors such as shapes, colors, layouts, and even typography play roles in this assessment. While traditional techniques provide some means to compute similarity, they often fall short in capturing the nuanced differences that are perceptually significant. To address this challenge, this research proposes the use of autoencoders, a type of neural network architecture, for image similarity assessment. Autoencoders are traditionally designed to compress input data into a lower-dimensional representation (encoding) and then reconstruct it back to its original form (decoding). However, the core hypothesis of this research is that the encoded representation alone can serve as a robust measure of image content and structure.

By training the autoencoder on a dataset of webpage design examples, it learns to generate encoded representations, or embeddings, of these designs. The underlying theory is that these embeddings, when visualized in a multidimensional space, will cluster similar designs close together. Consequently, when comparing the embeddings of two images, their proximity in this space can indicate their similarity. A distance close to 0 would imply high similarity, while a larger distance would indicate differences.

Such an autoencoder-based evaluation system offers an approach to assessing the efficacy of automated web code generation tools. By converting both the original design and the rendered output of the generated code into images, and subsequently into embeddings, a quantitative metric can be derived. This measure serves as a direct evaluation, indicating how faithfully the generated code replicates the original design.

5.3.2 Comparison of BLEU and Pixel-by-Pixel Methods with the Proposed Autoencoder-Based Evaluation System

The evaluation systems used by Pix2Code and Sketch2Code focus on direct comparisons between generated code and reference designs, using BLEU scores and pixel-by-pixel comparisons, respectively. While effective, these methods have limitations in capturing the perceptual similarity and structural fidelity of complex web designs.

Advantages of Autoencoder-Based Evaluation:

1. **Perceptual Accuracy:** Captures the perceptual similarity between the generated code and the original design more accurately by focusing on essential features and structures.

2. **Robustness:** By using embeddings in a latent space, it is less sensitive to minor variations in design that do not affect the overall structure and functionality of the web page.

3. **Comprehensive Evaluation:** Provides a quantitative similarity metric that encompasses both visual and structural fidelity, offering a more holistic evaluation of the generated code.

While Pix2Code and Sketch2Code use a combination of automated metrics and manual inspection to evaluate the quality of generated code, the proposed autoencoder-based evaluation system offers a more advanced and perceptually accurate method for assessing similarity. This system ensures that the generated code visually matches the original design and maintains structural integrity and functionality, providing a significant improvement in evaluating automated web code generation systems. Additionally, the proposed system does not require having the exact code for the image design you want to convert, which focuses more on the structure and allows for an easier creation of a dataset for testing. However, it should be noted that this method does not allow for checking the quality of the generated code.

5.3.3 Problem definition

The primary challenge that this evaluation system addresses relates to the quantitative evaluation of similarity between the rendered image generated from automated HTML5 and CSS3 code and the original design image. While traditional methods might use pixel-by-pixel comparisons or other basic techniques [103], they often fall short in capturing the nuanced differences crucial in web design. Furthermore, these methods can be computationally demanding and may not correspond closely to the human perceptual understanding of similarity [104]. To address this challenge, a reliable system is needed that can accurately compare two images: the original design and the outcome produced by the web code. This system should identify and measure differences in a manner that provides clear guidance for improvements [105]. Such a system becomes essential for evaluating tools that automatically generate web code [106], highlighting their strengths and areas for enhancement.

By leveraging autoencoders, a type of neural network well-suited for feature extraction and dimensionality reduction [107], this research aims to develop a system that can capture the essence of each image in a compressed representation. The encoded data from the autoencoder can then be used to compute a similarity percentage between the original and rendered images [108]. This approach promises a balance between computational efficiency and a perceptually meaningful evaluation metric.

5.3.4 Autoencoders Technique Justification for Image Similarity Evaluation

Autoencoders, a specialized type of neural network, have witnessed significant adoption in various domains, notably in feature extraction, dimensionality reduction, and data compression. Given their proven prowess in these areas, their applicability in image similarity assessment is grounded on several key attributes:

1. **Feature Compression:** Autoencoders are intrinsically designed to condense complex data into a compact representation. This means that high-dimensional images can be transformed into a lower-dimensional space, emphasizing only the most relevant and discerning features, effectively eliminating redundancies.

2. **Learning Hierarchical Features:** Neural networks, including autoencoders, inherently extract features in a hierarchical manner. Lower layers typically capture basic elements such as edges and textures, while deeper layers discern patterns, objects, and contextual information. This layered abstraction resonates with how humans perceive images, starting from granular details and progressively recognizing complex structures.

3. **Generalizability:** Autoencoders can be trained on vast datasets, allowing them to generalize well across a range of images. This means that the learned representations are not overly specialized to a narrow set of images but can provide meaningful metrics across diverse visual content.

4. **Perceptual Similarity:** Traditional metrics, like pixel-based comparisons, often do not account for perceptual nuances. Autoencoders, given their ability to capture hierarchical features, are more attuned to discern patterns and structures that are perceptually significant, producing similarity metrics that align more closely with human interpretation.

5. **Adaptability:** Variants of autoencoders, such as Variational Autoencoders (VAEs) and Denoising Autoencoders, offer refined capabilities. For instance, VAEs capture probabilistic representations, while Denoising Autoencoders are adept at handling noisy data. These adaptable architectures ensure that the autoencoder-based approach remains flexible, catering to diverse requirements in image similarity assessment.

5.3.5 In-depth Analysis of Autoencoders

Basic Architecture

An autoencoder consists of two main parts: the encoder and the decoder. The encoder compresses the input data into a lower-dimensional representation, while the decoder works to reconstruct the original data from this compressed form. [109]:

- **Encoder:** This part compresses the input into a lower-dimensional representation.
- **Decoder:** Starting from the encoding, it reconstructs the input.

The primary goal during the training of an autoencoder is to minimize the reconstruction error – the difference between the original input and its reconstructed version. This ensures that the compressed representation retains as much information as possible from the original data.

The choice of loss function is crucial in guiding the autoencoder during training. Mean squared error is a common choice for continuous data, while binary cross-entropy can be used for binary data. The loss function quantifies the difference between the original and reconstructed data.

The entire autoencoder is trained to minimize the difference between the input and its reconstructed output. An autoencoder is a special type of neural network that is trained to copy its input to its output. For example, given an image of a handwritten digit, an autoencoder first encodes the image into a lower dimensional latent representation, then decodes the latent representation back to an image (Figure: 5.1). An autoencoder learns to compress the data while minimizing the reconstruction error [110].

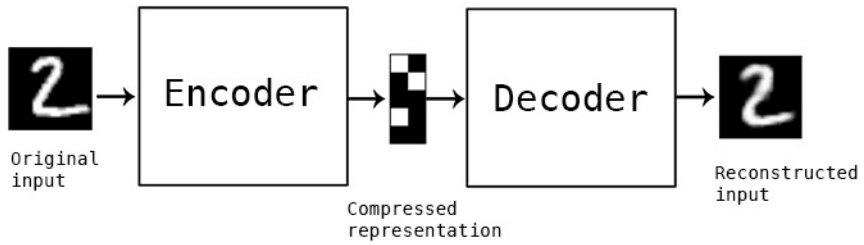


Figure 5.1: Autoencoder schema

Encoder:
The encoder (Figure: 5.2) is one of the two main components of an autoencoder, with the other being the decoder. It plays a critical role in the dimensionality reduction process. Essentially, the encoder is responsible for translating the input data into a lower-dimensional representation, often referred to as the latent space or code. The encoder typically consists of multiple layers, similar to the architecture of a deep neural network. The input layer receives data in its original high-dimensional form. As data progresses through subsequent layers, the dimensionality gets reduced. This reduction is achieved using various types of layers, such as dense (fully connected), convolutional, or recurrent layers, depending on the nature and type of input data.

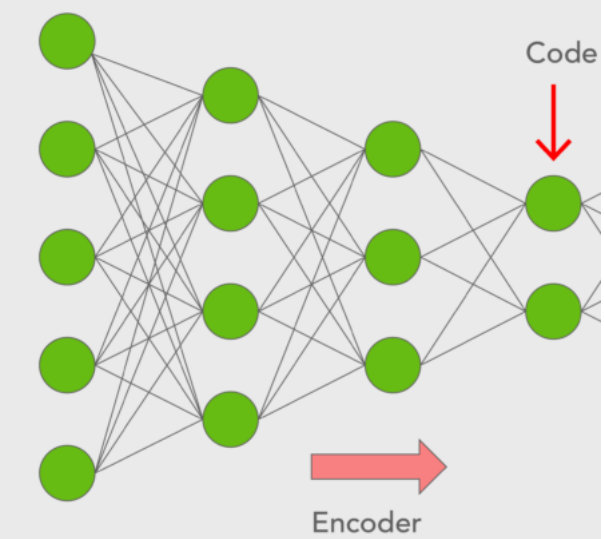


Figure 5.2: Encoder structure.

Within the encoder's architecture, neurons utilize activation functions to introduce non-linearity into the network. Common activation functions used are the Rectified Linear Unit (ReLU), sigmoid, and hyperbolic tangent (tanh). These functions ensure that the encoder can capture complex, non-linear relationships in the data. The final layer of the encoder produces the latent space representation of the input data. This compact representation retains essential information while discarding redundancies and noise. The dimensionality of the latent space is chosen based on the desired level of compression and the specific application's requirements.

The encoder's primary objective is to ensure that the compressed representation in the latent space can be effectively decompressed by the decoder to produce an output closely resembling the original input. This means the encoder must learn to retain critical features and patterns in the data during the compression process. During the training phase, the encoder and the decoder work in tandem. The loss function, often the mean squared error or a similar metric, measures the difference between the original input and the reconstructed output. By minimizing this error, the encoder learns the most efficient way to represent data in the latent space.

While the encoder's primary role is within the autoencoder framework, the compressed representations it produces can have standalone applications. For instance, these representations can be used for anomaly detection, data visualization, or as features for other machine learning models.

Embeddings:
Embeddings refer to the compact representation of data in the latent space. These embeddings provide a reduced-dimensional perspective on data while capturing its essential features and patterns. The embeddings generated by autoencoders are continuous, dense vectors. This is in contrast to one-hot encoded vectors which are sparse and high-dimensional. The dense nature of embeddings allows them to effectively capture the nuances and relationships inherent in the data.

The encoder's last layer is responsible for generating these embeddings. After processing the input data through various layers and transformations, the encoder culminates its task by compressing this information into the embedding representation.

The decoder uses the embeddings as its input. It aims to reconstruct the original data from this compact representation (Figure: 5.3). The fidelity of the reconstructed data to the original input acts as a testament to the quality and richness of the embeddings.

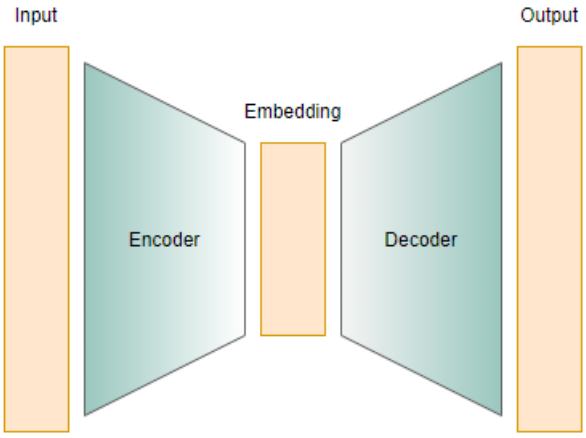


Figure 5.3: Embedding representation in autoencoder architecture.

Embeddings can be visualized using techniques like t-SNE or PCA. Visualization aids in understanding the relationships and clusters within data. For example, similar data points would cluster together in the embedded space, revealing patterns that might be hard to discern in the original high-dimensional space. In specific domains, especially text and images, the embeddings might capture semantic information. For instance, in text-based autoencoders, the embeddings might place synonyms close together, thereby revealing the semantic relationship between words or phrases. A noteworthy property of embeddings in autoencoders is their ability to act as noise filters. The autoencoder aims to capture the core patterns, often sidelining noise or outliers. Thus, the embeddings often represent a cleaner, noise-reduced version of the data.

Decorder:
The decoder is the counterpart to the encoder within the autoencoder framework. Following the dimensionality reduction executed by the encoder, the decoder's primary role is to reconstruct the original data from its compact representation in the latent space. Mirroring the encoder's design, the decoder is also composed of multiple layers (Figure: 5.4). However, it operates in the reverse direction. Beginning with the compressed representation as input, the decoder gradually upscales the data through its layers, ultimately aiming to produce an output that closely matches the original high-dimensional input.

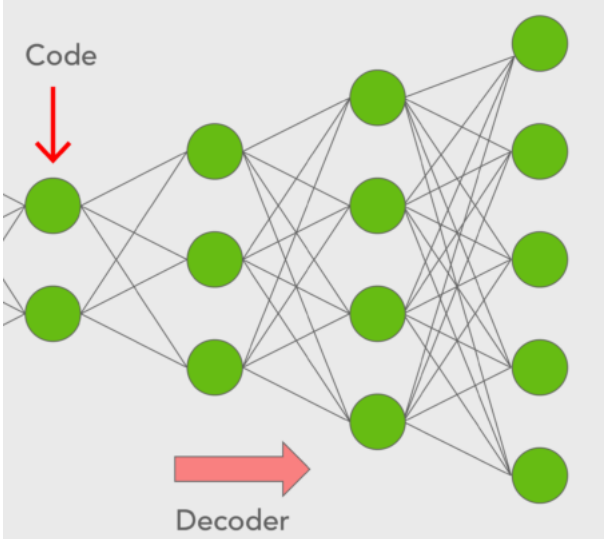


Figure 5.4: Decoder structure.

Just as in the encoder, activation functions are pivotal in the decoder to capture non-linear relationships. While the Rectified Linear Unit (ReLU) remains a popular choice, the final layer might employ the sigmoid or tanh activation functions, especially if the input data values are normalized between specific ranges. The decoder starts its operation in the latent space, taking the embeddings as input. Through a series of transformations, it attempts to reverse the compression, unraveling the data to its original structure and dimensionality.

The accuracy of the decoder's output is gauged using a loss function, which measures the discrepancy between the reconstructed data and the original input. Common choices include the mean squared error for continuous data and the binary cross-entropy for binary data. Minimizing this loss ensures that the decoder captures the essence of the original data. Reconstructing original data from a compact representation is not trivial. The decoder must navigate the information loss resulting from the encoding process. Striking a balance between data compression and accurate reconstruction is paramount for a well-functioning decoder.

While reconstruction remains the primary application, decoders can also be used for generative tasks. Given a point in the latent space, possibly not derived from any original input, the decoder can generate new data instances. This property is fundamental in generative models like Variational Autoencoders. The performance of the decoder is intricately tied to the encoder. A well-trained encoder producing meaningful embeddings will naturally aid the decoder in generating accurate reconstructions. Conversely, poor embeddings can hamper the decoder's performance, irrespective of its architectural prowess.

Applications in Image Similarity

Autoencoders, owing to their capacity to learn compact and meaningful representations of data, have found extensive applications in the area of image similarity. One of the primary utilities of autoencoders in image similarity is feature extraction. By training an autoencoder on a dataset of images, it learns to capture the most salient and distinguishing features of those images in its latent space [111]. These features can then serve as the basis for various similarity metrics, such as cosine similarity or Euclidean distance, providing a means to quantify the similarity between images based on their most essential characteristics.

In scenarios where the objective is to identify images that deviate from a norm, autoencoders can be invaluable. By training on a dataset of "normal" images, the autoencoder learns a typical representation. When a deviant or anomalous image is passed through, the reconstruction error tends to be higher, indicating its dissimilarity from the norm [112]. This principle is frequently used in industries like manufacturing to detect defects in products.

Autoencoders play a pivotal role in content-based image retrieval systems [113]. When a query image is presented, its encoding can be compared with the encodings of images in a database. Images with the most similar encodings to the query image are then retrieved. This approach ensures that the retrieved images are perceptually and contextually similar to the query image. While traditional image similarity techniques, like SSIM or MSE, operate in the pixel domain, incorporating autoencoder-based features can enhance their effectiveness [114]. By combining pixel-level metrics with feature-level representations from autoencoders, a more holistic measure of similarity can be achieved, aligning better with human perception.

5.3.6 Building the Similarity Evaluation System

Training involves feeding the autoencoder input data and setting the target output as the same input data [115]. The network adjusts its weights based on the difference between its output and the actual input. The process of preparing data, especially in the context of deep learning, is a foundational step that ensures models are trained and evaluated on the right information. The developed code for the system (Appendix-B: B.17) begins by collecting file paths from two distinct directories: /training/ and /testing/. It is a common practice to split data into training and testing sets, allowing models to learn from one subset and be evaluated on another. This partition aids in determining how well the model might perform on unseen data, improving its ability to generalize.

```
training_files = []
testing_files = []
for file_path in sorted(glob('./training/*')):
    training_files.append(file_path)

for file_path in sorted(glob('./testing/*')):
    testing_files.append(file_path)

training = np.array(training_files)
testing = np.array(testing_files)
print(training, testing)
```

After storing the file paths, the code defines the target dimensions for the images: a height (H) and width (W) of 128 pixels, with a channel (C) count of 3 to accommodate RGB images (Figure: 5.5). These uniform dimensions are vital for neural networks, as they guarantee consistent input shapes. Moreover, a 'results/' directory path is earmarked for storing any potential outputs or visualizations generated from the dataset.

The training phase is where models learn to generalize and make predictions. The journey begins with the initialization of the training and validation samples. It is crucial to discern the volume of data available for each phase. The code initializes tensors for both training (train_inputs) and validation (val_inputs) datasets, ensuring they are ready for the subsequent steps. Deep learning often grapples with large volumes of data. Processing all data simultaneously is computationally expensive and inefficient. Enter batch processing technique solves this where data is processed in small chunks or batches.

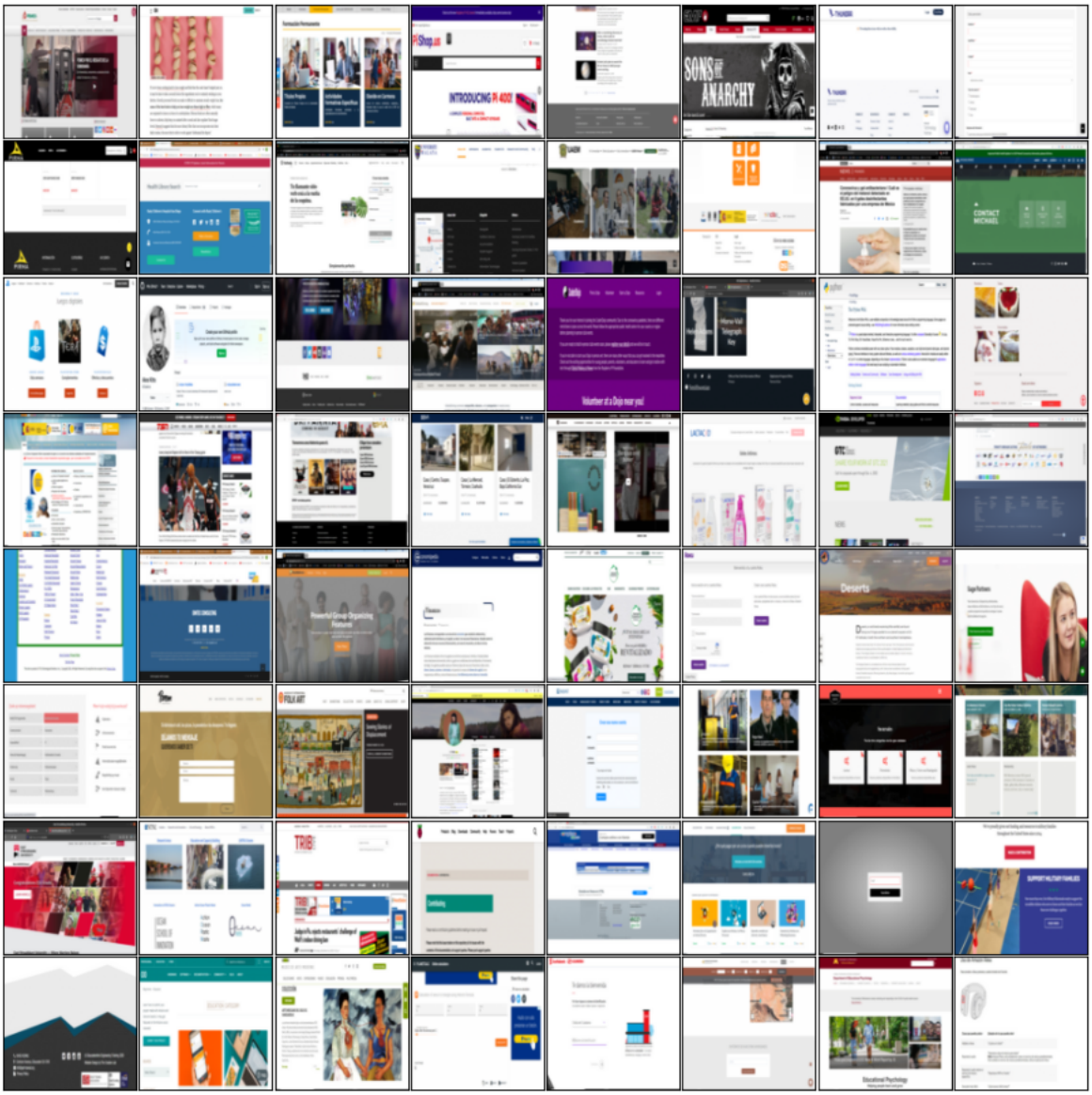


Figure 5.5: Random sampling of dataset for training the autoencoder.

The system specifies a BATCH_SIZE of 64, implying that 64 samples are processed concurrently in each iteration. Furthermore, data streaming is optimized using TensorFlow's tf.data API. This ensures efficient data loading, shuffling, and batching. The BUFFER_SIZE set to 128 aids in the shuffling of data, ensuring that the model is not exposed to data in the same order in every epoch, promoting better generalization.

```
BATCH_SIZE = 64
BUFFER_SIZE = 128

TRAIN_LENGTH = train_samples - valid_samples
STEPS_PER_EPOCH = TRAIN_LENGTH // BATCH_SIZE
train_dataset = train_dataset.cache()\
    .shuffle(BUFFER_SIZE)\
    .batch(BATCH_SIZE)\
    .repeat()
train_dataset = train_dataset.prefetch(
    buffer_size=tf.data.experimental.AUTOTUNE)
val_dataset = valid_dataset.batch(BATCH_SIZE)

VAL_SUBSPLITS = 1
VALIDATION_STEPS = valid_samples//BATCH_SIZE//VAL_SUBSPLITS

from model_autoencoder import Autoencoder
model = Autoencoder((H,W,C), latent_dim=3)
model.build((None, H, W, C))
model.summary()
```

After that, the autoencoder model is instantiated (Appendix-B: B.16). An autoencoder consists of two main components: an encoder that compresses the input and a decoder that reconstructs it (Figure: 5.6).

```
Model: "autoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 3)                 192195

 sequential_1 (Sequential)   (None, 120, 120, 3)       286659

=================================================================
Total params: 478854 (1.83 MB)
Trainable params: 478854 (1.83 MB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 5.6: Autoencoder model used for evaluation system.

The learning process of the model is influenced by several factors, including the optimization algorithm, loss function, and evaluation metrics. The model is compiled using the adam optimizer and the Mean Squared Error (mse) as the loss function, which is a common choice for regression problems. For effective training, callbacks are included. These serve as checkpoints, saving the weights of the model,

monitoring performance metrics, and adjusting the learning rate in situations where the progress of the model stagnates. The training is conducted for 150 epochs, with each epoch being a full forward and backward pass of all training samples.

```
model.compile(optimizer='adam', loss='mae', metrics = ['mse'])

EPOCHS = 150
Path(save_dir).mkdir(parents=True, exist_ok=True)

model_history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=EPOCHS,
    steps_per_epoch=STEPS_PER_EPOCH,
    callbacks=callbacks
)
```

After the training phase, assessing the model's performance is essential. With the help of the pandas library, the training history of the model is transformed into a dataframe and then visualized. The resulting plot displays the loss and mean squared error for both the training and validation datasets over the epochs. Significant drops in loss and a consistent pattern suggest effective learning.

An important aspect of machine learning involves not only training a model but also choosing its best iteration. Considering the fluctuation in performance over epochs, the code identifies the model with the optimal weights (lowest validation loss) using the glob function (Figure: 5.7). These weights are then loaded into the model for subsequent predictions.

```
history = pd.DataFrame(data=model_history.history)
history.to_csv(csv_file,index=False)
history.plot(figsize=(15,10))

best_model = sorted(glob(F"{save_dir}/*.hdf5"))[-1]
model.load_weights(best_model)
```

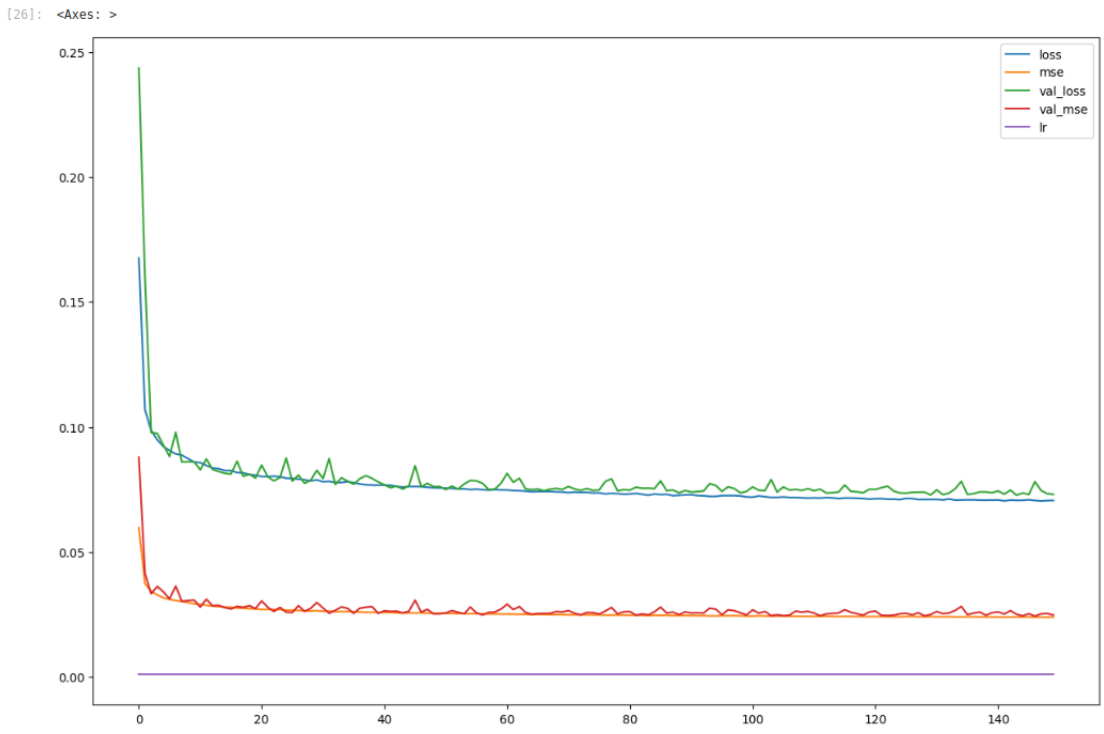


Figure 5.7: Choosing the best autoencoder model.

To properly evaluate the neural network model, it is necessary to load and prepare custom test data. The system begins by fetching files from a specified test directory, then converting these files into an array format for easy manipulation. The primary goal of this phase is to extract meaningful embeddings from the test images using the encoder component of the trained model. These embeddings represent compressed versions of the input and are substantial in the subsequent similarity evaluation. The code iterates through the test dataset, feeds the data into the encoder, and collects the resulting embeddings.

```
embeddings=[]

for i, (X, _) in enumerate(custom_dataset):
    X_batch = np.expand_dims(X, axis=0)  # Add the batch dimension
    Y = model.encoder.predict(X_batch)
    embeddings.append(Y)
    if i > 4:
        break

embeddings = np.vstack(embeddings)
```

Once the embeddings are generated, the following task is to measure the similarity between them. In the context of embeddings, the concept of "distance" refers to a measure of how different or similar two points (embeddings) are in a multi-dimensional space. Different distance metrics can be used to compute this measure, each with its own advantages and applications. The choice of distance metric is crucial, as it directly influences the evaluation of similarity. Commonly used distance metrics include Euclidean distance and cosine distance, each with its own advantages and applications. This section will provide an in-depth explanation of these metrics and justify the selection of cosine distance for this study.

5.3.7 Distance Metrics

An embedding is a representation of data in a continuous vector space where each dimension captures some aspect of the data's characteristics. In this multi-dimensional space, similar data points will have embeddings that are close to each other. The hypothesis is that if two embeddings are in near locations within this space, they are likely to be similar. This is because the distance between embeddings reflects the similarity in their characteristics. Therefore, measuring the distance between embeddings can provide insights into how similar the underlying data points are.

Cosine Distance:
Cosine distance measures the cosine of the angle between two vectors, focusing on their orientation rather than their magnitude. Cosine similarity S is defined as:

$$S(\mathbf{p}, \mathbf{q}) = \frac{\mathbf{p} \cdot \mathbf{q}}{\|\mathbf{p}\|\|\mathbf{q}\|}$$

where $\mathbf{p} \cdot \mathbf{q}$ is the dot product of the vectors, and $\|\mathbf{p}\|$ and $\|\mathbf{q}\|$ are their magnitudes. Cosine distance d is then given by:

$$d(\mathbf{p}, \mathbf{q}) = 1 - S(\mathbf{p}, \mathbf{q})$$

Captures the orientation of the vectors, which is often more relevant in high-dimensional spaces.

5.3.8 Justification for Choosing Cosine Distance

In the context of comparing neural network embeddings, the cosine distance metric is particularly well-suited for several reasons:

1. **High-Dimensional Data:** Neural network embeddings typically exist in high-dimensional spaces where the orientation of vectors is often more significant than their magnitude. Cosine distance

effectively captures this orientation, making it a more appropriate measure of similarity.

2. **Magnitude Invariance:** Cosine distance ensures that the similarity measure focuses solely on the direction of the vectors, providing a more meaningful comparison in many scenarios.

Given the nature of neural network embeddings and the requirements of this evaluation system, cosine distance was chosen as the preferred metric. It offers a robust measure of similarity that accounts for the orientation of high-dimensional vectors, ensuring a more accurate and meaningful comparison of embeddings. By leveraging cosine distance, the evaluation system can more effectively assess the similarity between generated and reference images, ultimately enhancing the reliability and interpretability of the results.

The following code snippet demonstrates the calculation of the cosine distance matrix for the generated embeddings:

```
val = distance.cdist(embeddings, embeddings, metric='cosine')
print(val.shape)
d1 = val[0][1]
d2 = val[2][5]
print(d1, d2)
```

This matrix provides insights into which images are most alike based on their embeddings, facilitating a comprehensive evaluation of the similarity between the generated and reference images.

5.4 Performance Metrics and Benchmarks

In the scope of this research, autoencoders were deployed for the feature extraction from images, serving as a preliminary step for the evaluation of image similarity. Upon obtaining the reduced-dimensional embeddings via the encoder, the similarity between images was quantified using the cosine distance metric. This particular metric was chosen as it evaluates the cosine of the angle between two vectors, which effectively ignores the magnitude of the vectors, thus focusing only on the directional similarity.

A pairwise comparison of all image embeddings was executed using the cdist function from the “scipy spatial distance” library. This function computes the cosine distance between every pair of the multivariate observations represented in the embedding matrix. The output from this computation, a square matrix where each element (i, j) represents the distance between the i-th and j-th embeddings, was then examined.

```
image_a = 1
image_b = 2
evaluation_images = []
evaluation_images.append(images[image_a])
evaluation_images.append(images[image_b])

canvas, axis = multi_plots(evaluation_images, rows=1, cols=2, scale=10)

val = distance.cdist(embeddings, embeddings, metric='cosine')
d = val[image_a][image_b]
print(f"The Distance between IMAGE: {image_a} and IMAGE: {image_b} is: {d}")
```

In order to assess the performance of the image similarity evaluation system, a set of eight images (Figure: 5.8) pertaining to web page designs and their corresponding code rendered results were selected.

Figure 5.8: Choosing images for evaluation system testing.

The images chosen for this assessment are described as follows:

- Image 1 represents the original design of webpage A.
- Image 2 is an identical copy of Image 1, serving as a control to validate the sensitivity of the system to identical inputs.
- Image 3 is the original design of webpage B, distinct from webpage A.
- Image 4 illustrates the original design of webpage C, introducing further variability.

- Image 5 depicts the original design of webpage D.
- Image 6 presents the code-rendered result of webpage B, providing a basis for comparison with Image 3.
- Image 7 shows the code-rendered result of webpage C, to be compared with Image 4.
- Image 8 displays the code-rendered result of webpage D, offering a comparative view against Image 5.

The selection was strategically composed to validate the system's efficacy in differentiating between identical, similar, and dissimilar images. Given the identical nature of Images 1 and 2 (Figure: 5.9), the similarity evaluation system is expected to yield a cosine distance metric very close to 0, indicating no discernible difference.

Figure 5.9: Evaluation System Result: A distance of 0.0. Image 1 compared with Image 2.

In contrast, a pair such as Images 1 and 7 should manifest a significantly greater cosine distance (Figure: 5.10), reflecting their dissimilarity as they represent different web pages.

Figure 5.10: Evaluation System Result: A distance of 1.9938. Image 1 compared with Image 7.

Moreover, the system's ability to recognize the likeness between designs and their code-rendered counterparts is examined through pairs such as Images 4 vs 7 (Figure: 5.11) and Images 3 vs 6 (Figure: 5.12). While not identical, these images share a high degree of similarity due to their common origin, which should be indicated by a cosine distance metric that is low but may not necessarily approach zero, acknowledging the subtle differences that may arise from the rendering process.

Figure 5.11: Evaluation System Result: A distance of 0.1020. Image 4 compared with Image 7.

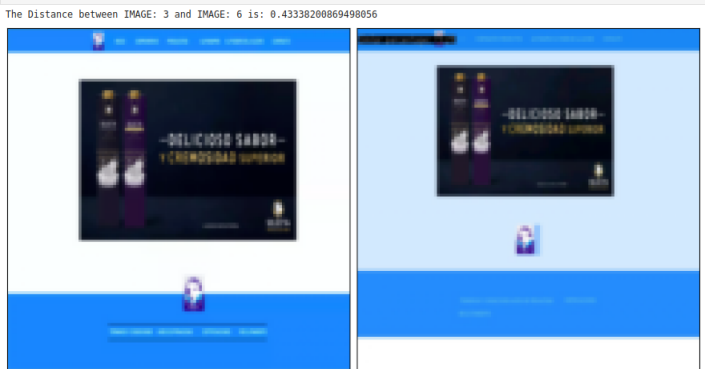


Figure 5.12: Evaluation System Result: A distance of 0.4333. Image 3 compared with Image 6.

The results of these comparisons are anticipated to demonstrate the robustness of the similarity evaluation mechanism, especially in discerning varying levels of likeness among the images, which range from identical to similar to distinct.

For a comprehensive visualization of the similarities, a heatmap was generated using the Seaborn library's heatmap function (Figure: 5.13). This allowed for the entire matrix of distances to be displayed in a color-coded format, facilitating an intuitive grasp of the relative similarities between all pairs of images. The color scheme was intentionally selected to have a diverging palette, the color palette was centered to effectively represent the spectrum of values encountered within the dataset.

```
plt.figure(figsize=(10, 8))
sns.heatmap(val, cmap='RdBu_r', center=0)
plt.show()
```

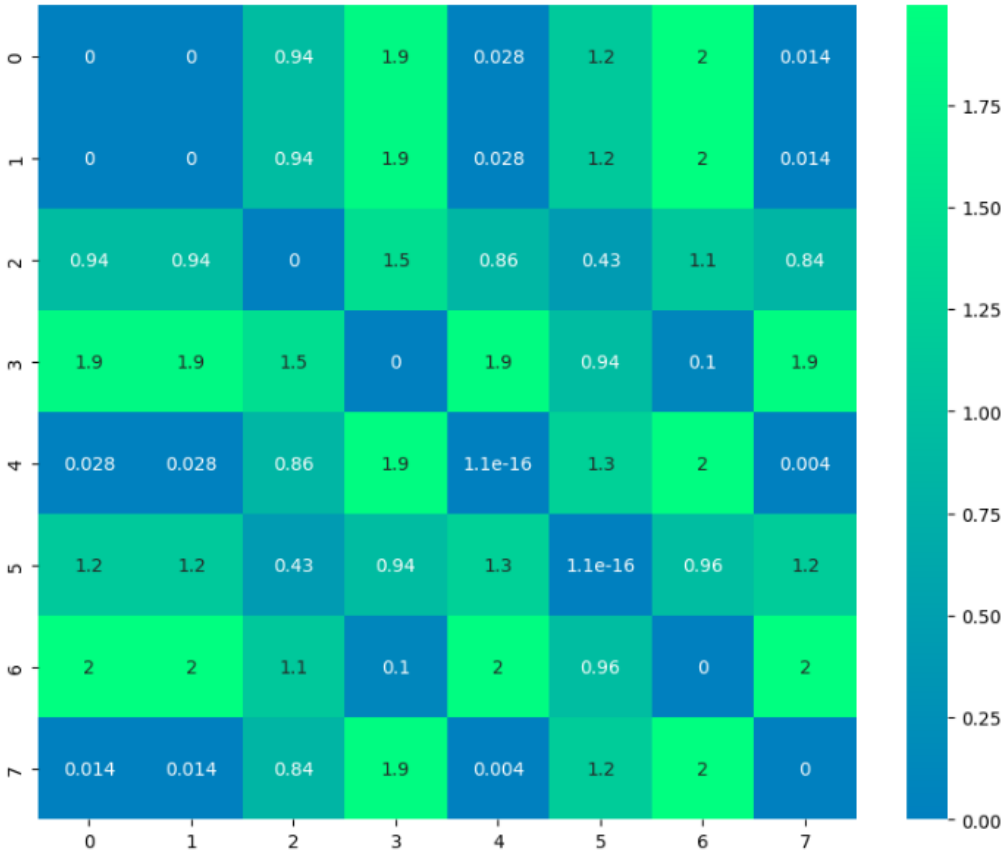


Figure 5.13: Evaluation System Result: A distance of 0.4333. Image 3 compared with Image 6.

The heatmap (Figure: 5.13) represents the similarity in structure and element localization between different web pages. Despite the distinct nature of each page, the visualization highlights the regions of high congruence, denoted by the blue colors, and regions of lesser similarity, denoted by green colors. It can be observed that certain areas maintain consistent similarity across both pages, as evidenced by the concentration of non-zero values in corresponding cells of the matrix. This consistency suggests that while the content may vary, the underlying structure and the spatial distribution of elements have a significant resemblance. This pattern of similarity is critical in understanding the design paradigms that persist across different web pages and how these may influence user interaction and experience

Upon meticulous examination of the heatmap generated from the comparative analysis, it was observed that the similarity index between Image 1 and Image 5 (Figure: 5.14) is notably high (closer to 0). This high degree of similarity underscores a critical limitation in the evaluation system. The heatmap serves as a visual representation of structural congruity, revealing a strong alignment between the two web pages in terms of layout and design. However, this result also highlights a significant challenge: the system's current inability to distinguish between different yet visually similar webpage designs effectively.

Figure 5.14: Evaluation System Result: A distance of 0.0278. Image 1 compared with Image 5.

The quantitative analysis reveals a minimal distance of approximately 0.0278 between Image 1 and Image 5 (Figure: 5.14), suggesting a high degree of structural similarity. This proximity indicates that, despite differences in content and thematic presentations, the foundational layout patterns of both pages are nearly identical. This result validates the ability of the system to identify similar structural features but also highlights a critical vulnerability: the system's sensitivity to subtle variations in design is insufficient.

This issue is particularly pronounced in cases where the web pages represent similar structures, revealing a significant area for enhancement. A better approach could involve combining cosine distance (for angle similarity) and Euclidean distance (for magnitude in high-dimensional embeddings). This combined metric could reinforce the evaluation of structural patterns, ensuring a more robust differentiation between designs that are similar but not identical.

Additionally, integrating pixel-to-pixel comparison techniques could allow for the evaluation of stylistic elements like colors, fonts, and other visual details not currently captured by the structural similarity metrics. This method would permit a more comprehensive analysis of both the functional and aspects of web pages, providing deeper insights into their similarities and differences.

In conclusion, while the use of autoencoders for feature extraction and cosine distance as a similarity metric has proven effective, these findings underscore the need for a more sophisticated approach. The current study exposes the limitations of the evaluation system, particularly in differentiating web pages with close structural similarities. Future research should focus on refining the similarity metrics to enhance their precision and expand their applicability across diverse web design scenarios, ultimately leading to a more capable and versatile tool for template detection, thematic categorization, and design consistency analysis.

6

Quantifying Success: Results and Performance Metrics

Evaluating the success of automated web code generation requires comparing the original design with the output produced by the generated code. This chapter focuses on this comparison, employing both visual and structural examinations to quantify the conversion results.

6.1 Introduction

The initial stage of comparison is visual—presenting side-by-side images of the original web page design alongside the website rendered from the generated HTML5 and CSS3 code. These images form the foundation for a qualitative assessment, wherein the similarities in layout, color schemes, typography, and responsive elements can be assessed.

Subsequent to the visual evaluation, attention is directed towards a structural comparison. This describes a detailed result of the Document Object Model (DOM) of the rendered page. Parsing the elements of the DOM tree allows for an assessment of the hierarchical organization of elements, the nesting of tags, and the application of CSS styling rules.

Concluding the evaluation process, the assessment utilizes the proposed evaluation system to determine the image similarity between the original design and the resulting rendered webpage. Employing autoencoders, as delineated in Chapter 5, facilitates the computation of a metric that quantifies the degree of similarity between the two images. This metric serves as an objective indicator of the system's performance in replicating the design with high fidelity. Through this methodical approach, the evaluation system provides a robust framework for measuring the effectiveness of the code generation in capturing the intended styles and functional attributes of the original webpage design.

6.2 Evaluating the Success of Automated Web Code Generation. Example 1.

6.2.1 Visual comparison

The first stage of comparison involves a visual examination of the generated output against the original design. This stage presents side-by-side images of the original webpage design and the website rendered from the generated HTML5 and CSS3 code. These images form the foundation for a qualitative assessment, where similarities in layout, color schemes, typography, and responsive elements can be evaluated. It helps to identify any discrepancies or areas for improvement in the visual fidelity of the generated code.

This section will exhibit an example of a webpage design alongside the output of the automated code generation system, providing a practical illustration to evaluate the general results (Figure: 6.1). The upper image delineates the original webpage design which will be used as the benchmark for comparison. The lower image reveals the outcome of the web code generated by the system. A general inspection reveals that the structural essence and layout of the rendered webpage closely align with the original design.

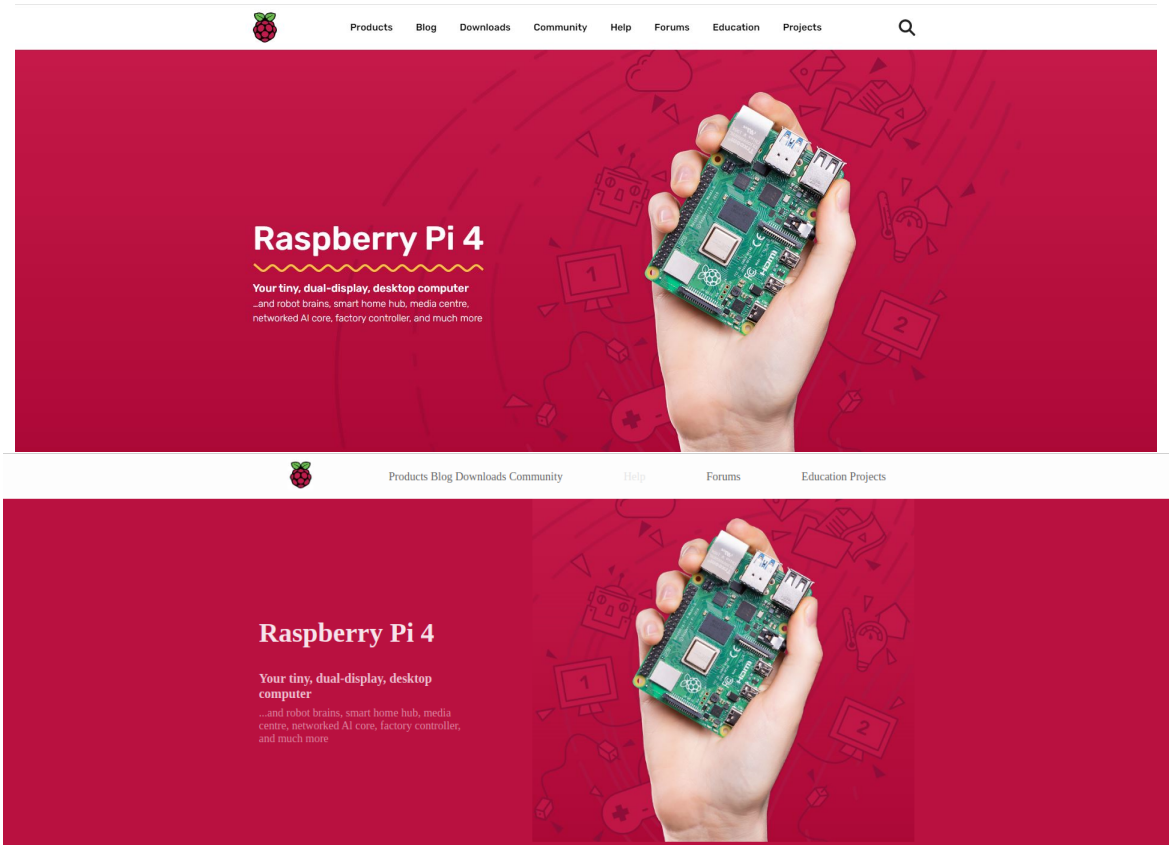


Figure 6.1: Result of the Proposed Model For Automated Web Code Generation. Comparison of the Original Design (Upper Image) and the Rendered Output from the System-Generated Code (Lower Image)

6.2.2 Structural comparison

Upon reviewing the result of the automated code generation, it is observed that the system accurately detected and appropriately structured the header of the webpage, incorporating the correct positioning of the logo image (Figure: 6.2).

Figure 6.2: Result Header And Logo Image Detection.

```
<header id="header0">
    <div id="header0-div-image1">
        <img id="image1"  class="logo header__img__image1" src="IMG_PATH"/>
    </div>
    ...
</header
```

Furthermore, the textual elements, presumably menu items, have been identified and assigned within the navigation bar that the system detected (Figure: 6.3). The precise detection and correct structuring of these elements within the generated code are important for maintaining the design integrity and navigational functionality of the original webpage design.

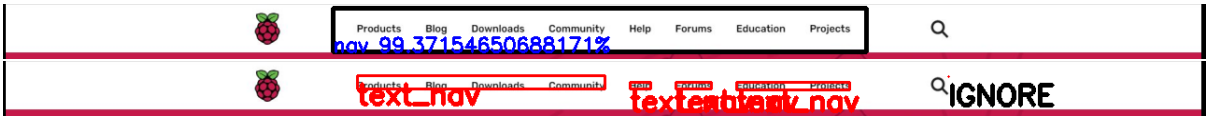


Figure 6.3: Result Navbar and Text detection.

```
<header id="header0">
    ...
    <nav class="nav0" css_pattern="header0">
        <ul class="nav0__ul" css_pattern="nav0">
            <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item0">
                <a class="nav-link␣nav0__a__txtnav0-0" css_pattern="nav0" href="#">
                    Products Blog Downloads Community
                </a>
            </li>
            <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item1">
                <a class="nav-link␣nav0__a__txtnav0-4" css_pattern="nav0" href="#">Help</a>
            </li>
            <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item2">
                <a class="nav-link␣nav0__a__txtnav0-0" css_pattern="nav0" href="#">Forums</a>
            </li>
            <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item3">
                <a class="nav-link␣nav0__a__txtnav0-0" css_pattern="nav0" href="#">Education Projects</a>
            </li>
        </ul>
    </nav>
</header>
```

In the generated code, a notable area for improvement is the text analysis for navigation items. The code snippet presented suggests that the text “Products Blog Downloads Community” has been erroneously grouped as a single navigation item (<li> element), rather than being split into individual items:

```
<li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item0">
    <a class="nav-link␣nav0__a__txtnav0-0" css_pattern="nav0" href="#">
        Products Blog Downloads Community
    </a>
</li>
```

Ideally, each of these words should represent a separate navigational link within the header. This misinterpretation can be attributed to the Optical Character Recognition (OCR) service from Google Cloud Platform (GCP) not segregating the text into distinct boxes during the analysis phase, leading to a fusion of separate menu items into a single link.

Adjusting the OCR parameters or refining the post-processing algorithms to correctly identify and separate text into individual navigable elements would enhance the accuracy of the generated code, ensuring each menu item is correctly placed within its own <li> element, reflecting the structure of the original website design.

For constructing the XML tree requires the assignment of web elements to their respective containers. The following examples illustrate how the system has effectively recognized web elements such as texts (Figure: 6.5) and images (Figure: 6.4) within the structure of this particular webpage.

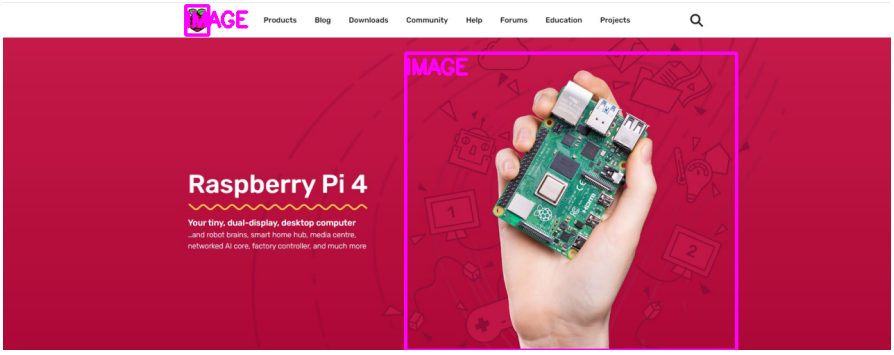


Figure 6.4: Resultant Detection of Web Elements: Images.

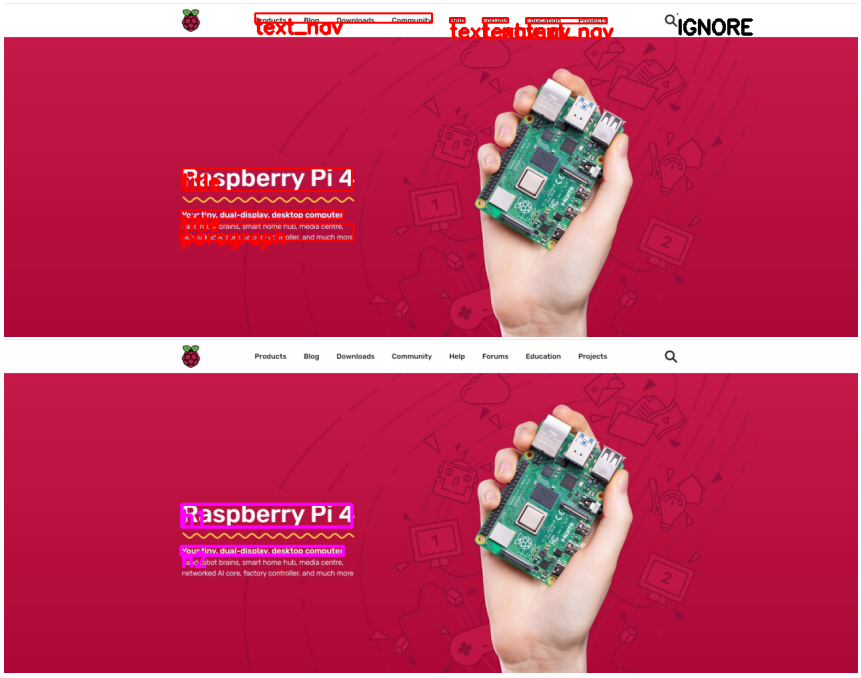


Figure 6.5: Resultant Detection of Web Elements: Text and Titles. This picture specifically illustrates the successful identification of textual content and titles within the webpage by the detection system, showcasing the precision of the element recognition process.

Once web elements are detected, segmentation of the webpage can be undertaken (Figure: 6.6), which defines the various sections and div containers. This process results in a structured layout similar to image figure: 6.8, where the web containers are distinctly identified.

Figure 6.6: Illustration of Webpage Segmentation: This figure demonstrates the application of segmentation on the webpage after the successful detection of web elements.

The segmentation process is crucial for the application of a Convolutional Neural Network (CNN) model, which is particularly defined to identify 'section' and 'div' elements within a webpage (Figure: 6.7). This targeted focus of the CNN ensures precise recognition and classification of these structural components, which are fundamental to the semantic layout of the web design.

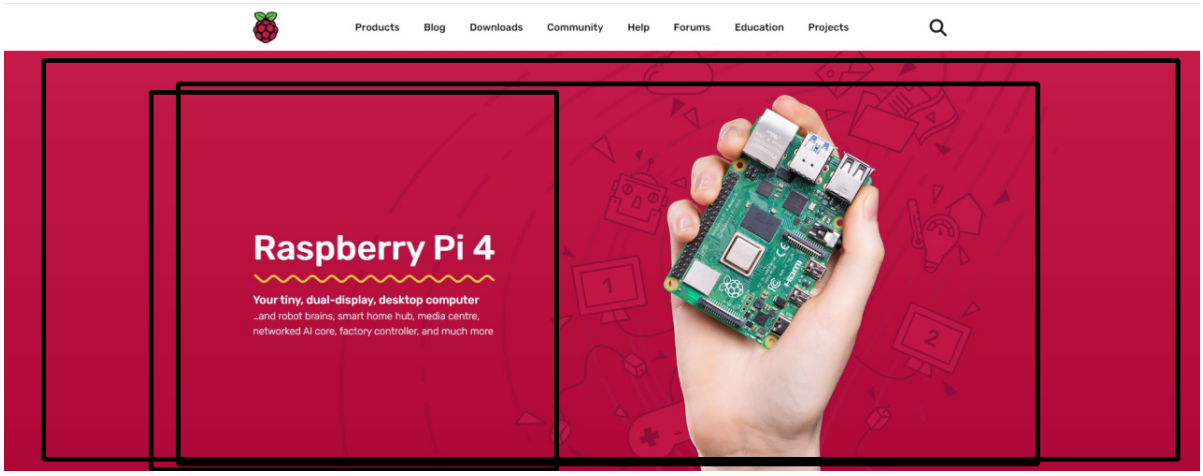


Figure 6.7: Visualization of Detected Containers: Section and Div Elements. This image showcases the output of the container detection process, highlighting how the system identifies various 'section' and 'div' elements within the webpage's structure.

Following the initial placement of web elements into containers and the proposal of an XML tree structure by the system, it becomes necessary to execute the inference algorithm subsystem (Figure: 6.8). This step is important for enhancing the quality of the generated code. It involves fine-tuning container

sizes and refining the XML tree structure to ensure its validity and efficiency. Such improvements are crucial in achieving a more precise representation of the intended layout and design of the webpage within the code.

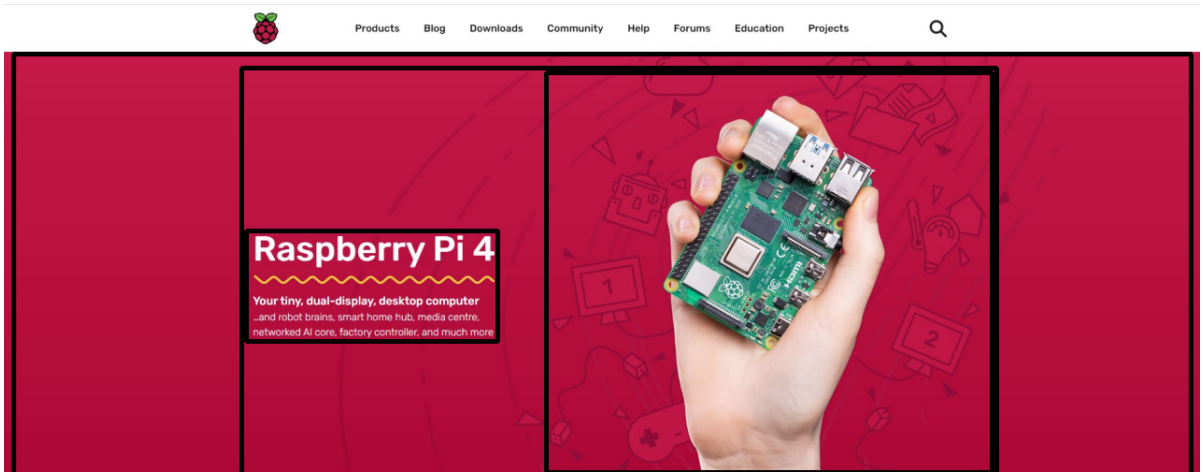


Figure 6.8: Enhanced Container Structure Post-Inference Algorithm Application:

```
<section id="section_1">
    <div id="div_4" row="True">
        <div id="div_3" row="False">
            <h1 class="div_3__h1__text1" css_pattern="div_3">
                Raspberry Pi 4
            </h1>
            <h2 class="div_3__h2__text2" css_pattern="div_3">
                Your tiny, dual-display, desktop computer
            </h2>
            <p class="div_3__p__text3" css_pattern="div_3">
                ...and robot brains, smart home hub, media centre,
                networked Al core, factory controller, and much more
            </p>
        </div>
        <div css_pattern="div_4" id="div-image0">
            <img class="div-image0__img__image0" css_pattern="div-image0" src="IMG_PATH"/>
        </div>
    </div>
</section>
```

The generated code snippet provided demonstrates a structured approach to web layout using CSS Flexbox. A section element is introduced, encapsulating a main div with a Flexbox display set to a row configuration. This serves as a container for two child div elements: the first is assigned a Flexbox column direction to organize web elements vertically, accommodating elements such as headers (<h1> and <h2>) and paragraphs (<p>) for textual content. The second child div is designated for image inclusion. The application of Flexbox properties ensures a responsive and orderly layout, reflecting the design's intent for both textual and visual content alignment.

Finally the styles feature extractor subsystem comes into play, enabling the generation of CSS3 code. This subsystem analyzes the design elements, translating the visual attributes into corresponding CSS3 styles. This final step goal is ensuring that the style aspects of the webpage, such as layout, color schemes, and typography, are accurately reflected in the generated stylesheet, thereby bringing the visual design to life in the rendered webpage.

```
/******************************/
/* Beginning of common styles. */
/******************************/

/* Beginning of styles for block body. */


/* Beginning of styles for block button. */
button {
    display: block;
}


/* Beginning of styles for block div-image0. */
#div-image0 {
    margin-right: 5px;
}


.div-image0__img__image0 {
    height: 427px;
    width: 477px;
}


/* Beginning of styles for block div_3. */
#div_3 {
    align-items: flex-start;
    display: flex;
    flex-direction: column;
    justify-content: center;
    margin-left: 5px;
    text-align: left;
    width: 269px;
}


.div_3__h1__text1 {
    color: rgb(247, 228, 235);
    font-size: 34px;
    margin-bottom: 0em;
}


.div_3__h2__text2 {
    color: rgb(238, 194, 210);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 29px;
}


.div_3__p__text3 {
    color: rgb(222, 138, 164);
    font-size: 14px;
    margin-bottom: 0em;
    margin-top: 7px;
}


/* Beginning of styles for block div_4. */
#div_4 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    width: 828px;
}


/* Beginning of styles for block header. */

.header__checkbox {
    display: none;
}


.header__img__image1 {
    margin-right: 54px;
    object-fit: cover;
    width: 33px;
}


/* Beginning of styles for block header0. */
#header0 {
    align-items: center;
    background-color: rgb(253, 253, 253);
    display: flex;
    justify-content: center;
    padding: 3px 0px;
    width: 100%;
}


/* Beginning of styles for block icon_menu. */
.icon_menu {
    cursor: pointer;
    display: none;
    font-size: 25px;
}


/* Beginning of styles for block nav0. */
.nav0 {
    align-items: center;
    display: flex;
    justify-content: center;
    width: 661px;
}


.nav0__a__txtnav0-0 {
    color: rgb(95, 95, 95);
    font-size: 14px;
    margin-bottom: 0.5em;
    margin-top: 0.5em;
    text-decoration: none;
```

```
}

.nav0__a__txtnav0-4 {
    color: rgb(230, 230, 230);
    font-size: 14px;
    margin-bottom: 0.5em;
    margin-top: 0.5em;
    text-decoration: none;
}

.nav0__ul {
    align-items: center;
    display: flex;
    justify-content: space-between;
    list-style: none;
    width: 100%;
}


/* Beginning of styles for block section_1. */
#section_1 {
    align-items: center;
    background-color: rgb(185, 17, 64);
    display: flex;
    justify-content: center;
    padding: 0px 0px;
    width: 100%;
}


/*****************************/
/* Beginning of media styles. */
/*****************************/

/* Beginning of styles for 480px width. */
@media(max-width: 480px) {
    #div_3 {
        width: 100%;
    }
}

/* Beginning of styles for 800px width. */
@media(max-width: 800px) {
}

/* Beginning of styles for 1200px width. */
@media(max-width: 1200px) {
    #div_4 {
        width: 100%;
    }
    #header0 {
        flex-direction: row-reverse;
        justify-content: space-between;
    }
    #header0-div-image1 {
        margin-left: 15px;
    }
    .header__checkbox:checked ~ .nav0 {
        display: flex;
        flex-direction: column;
    }
    .icon_menu {
        display: flex;
        margin-right: 15px;
    }
    .nav0 {
        display: none;
        position: absolute;
        width: 100%;
        top: 55px;
    }
    .nav0__ul {
        flex-direction: column;
        background-color: rgb(253, 253, 253);
    }
}
```

6.2.3 Evaluation Process

Utilizing the image similarity evaluation system outlined in Chapter 5, the performance can be quantitatively measured by comparing the original design with the rendered image of the webpage generated from the HTML5 and CSS3 code. In this process, a similarity metric is computed, yielding a value of 0.06467 (Figure: 6.9). This metric operates on a scale where a value closer to 0 indicates a higher degree of similarity between the generated webpage and the original design.

Therefore, a value of 0.06467 suggests a substantial degree of resemblance, underscoring the effectiveness of the system in replicating the original design. This metric serves as a concrete indicator of the system's proficiency, demonstrating that the results achieved possess a high level of fidelity to the original design.

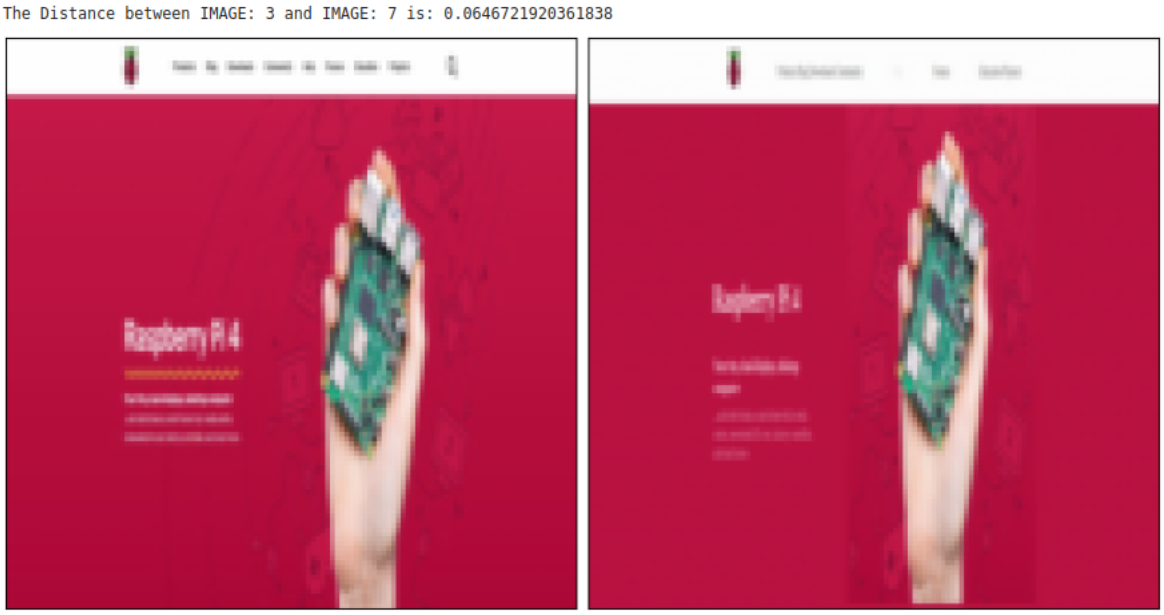


Figure 6.9: Application of Evaluation Metrics via the Proposed Evaluation System: Highlighted here is the computed score, representing a distance metric of 0.0646721920361838, which quantifies the similarity between the generated output and the original design.

6.3 Results from the Automatic Web Code Generation and Evaluation System.

In addition to the outcome already demonstrated, this section includes additional examples showcasing the performance of the system. Accompanying each example is the respective evaluation metric and code, providing a quantifiable measure of the system's effectiveness in each scenario. These examples serve to further illustrate the capabilities and areas for improvement of the web code generation model, offering a more general perspective on its overall performance.

In this section, three additional examples (Examples 2, 3, and 4) are presented to further demonstrate the system's performance. These examples are not explored in as much detail as the first example but provide a general overview of the results and their corresponding evaluation metrics. Including these additional examples showcases the consistency and reliability of the web code generation model across different scenarios, highlighting both its strengths and areas for improvement.

6.3.1 Webpage Analysis: TensorFlow Landing Page. Example 2.

Example 2 showcases the main page design alongside the results of the rendered code generated by the system (Figure: 6.10). This comparison allows for a detailed evaluation of the system's ability to accurately replicate the visual and structural elements of the webpage.

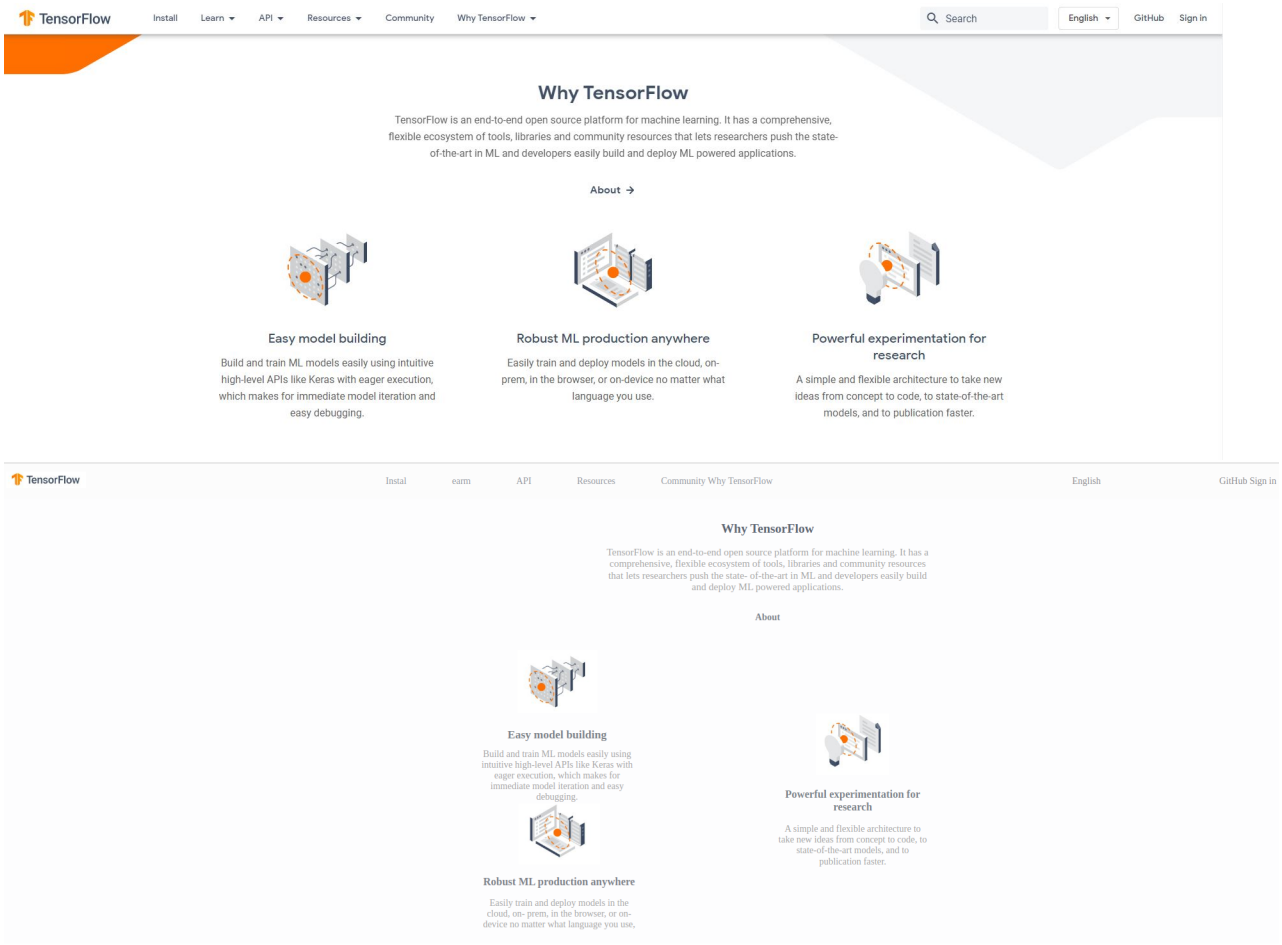


Figure 6.10: Illustrative Example of Outcomes from the Automatic Web Code Generation Model: This image showcases a specific instance of the results produced by the model, exemplifying its capability in translating web designs into code.

HTML5 code:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8"/>
        <meta content="width=device-width, initial-scale=1/>
        <link href="normalize.css" rel="stylesheet"/>
        <link href="styles_19.css" rel="stylesheet"/>
        <title>19</title>
```

```
␣␣␣␣</head>
␣␣␣␣<body>
␣␣␣␣␣␣␣␣<header␣id="header0">
␣␣␣␣␣␣␣␣␣␣␣␣<input␣class="header__checkbox"␣id="header__checkbox"␣type="checkbox"/>
␣␣␣␣␣␣␣␣␣␣␣␣<label␣class="icon_menu"␣for="header__checkbox">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<i␣class="fas fa-align-justify">␣</i>
␣␣␣␣␣␣␣␣␣␣␣␣</label>
␣␣␣␣␣␣␣␣␣␣␣␣<div␣id="header0-div-image1">
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<img␣class="logo header__img__image1"␣css_pattern="header"␣id="image1"␣src="IMG_PATH/>
            </div>
            <nav class="nav0" css_pattern="header0">
                <ul class="nav0__ul" css_pattern="nav0">
                    <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item0">
                        <a class="nav-link␣nav0__a__txtnav0-0" css_pattern="nav0" href="#">Instal</a>
                    </li>
                    <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item1">
                        <a class="nav-link␣nav0__a__txtnav0-0" css_pattern="nav0" href="#">earm</a>
                    </li>
                    <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item2">
                        <a class="nav-link␣nav0__a__txtnav0-0" css_pattern="nav0" href="#">API</a>
                    </li>
                    <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item3">
                        <a class="nav-link␣nav0__a__txtnav0-0" css_pattern="nav0" href="#">Resources</a>
                    </li>
                    <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item4">
                        <a class="nav-link␣nav0__a__txtnav0-0" css_pattern="nav0" href="#">Community Why TensorFlow</a>
                    </li>
                </ul>
            </nav>
            <nav class="nav1" css_pattern="header0">
                <ul class="nav1__ul" css_pattern="nav1">
                    <li class="nav-item␣nav1__li" css_pattern="nav1" id="nav1-item0">
                        <a class="nav-link␣nav1__a__txtnav1-15" css_pattern="nav1" href="#">English</a>
                    </li>
                    <li class="nav-item␣nav1__li" css_pattern="nav1" id="nav1-item1">
                        <a class="nav-link␣nav1__a__txtnav1-15" css_pattern="nav1" href="#">GitHub Sign in</a>
                    </li>
                </ul>
            </nav>
        </header>
<!-- This is the <section> tag, corresponding to "section_1". -->
        <section id="section_1">
            <div id="div_11" row="False">
                <div id="div_12" row="False">
                    <h2 class="div_12__h2__text5" css_pattern="div_12">Why TensorFlow</h2>
                    <p class="div_12__p__text6" css_pattern="div_12">
                        TensorFlow is an end-to-end open source platform for
                        machine learning. It has a comprehensive, flexible ecosystem
                        of tools, libraries and community resources that lets
                        researchers push the state- of-the-art in ML and developers
                        easily build and deploy ML powered applications.
                    </p>
                    <h2 class="div_12__h2__text9" css_pattern="div_12">About</h2>
                </div>
                <div css_pattern="div_11" id="div_11-row_1" row="True">
                    <div class="" css_pattern="div_11-row_1" id="div_10" row="True">
                        <div id="div_8" row="False">
                            <div css_pattern="div_8" id="div-image2">
                                <img class="div-image2__img__image2" css_pattern="div-image2" src="IMG_PATH"/>
                            </div>
                            <h2 class="div_8__h2__text7" css_pattern="div_8">Easy model building</h2>
                            <p class="div_8__p__text8" css_pattern="div_8">
                                Build and train ML models easily using intuitive
                                high-level APIs like Keras with eager execution,
                                which makes for immediate model iteration and easy debugging.
                            </p>
                        </div>
                        <div id="div_7" row="False">
                            <div css_pattern="div_7" id="div-image0">
                                <img class="div-image0__img__image0" css_pattern="div-image0" src="IMG_PATH"/>
                            </div>
                            <h2 class="div_7__h2__text10" css_pattern="div_7">
```

```
                        Robust ML production anywhere
                    </h2>
                    <p class="div_7__p__text11" css_pattern="div_7">
                        Easily train and deploy models in the cloud,
                        on- prem, in the browser, or on-device no matter
                        what language you use,
                    </p>
                </div>
            </div>
            <div class="" css_pattern="div_11-row_1" id="div_9" row="False">
                <div css_pattern="div_9" id="div-image3">
                    <img class="div-image3__img__image3" css_pattern="div-image3" src="IMG_PATH"/>
                </div>
                <h2 class="div_9__h2__text13" css_pattern="div_9">
                    Powerful experimentation for research
                </h2>
                <p class="div_9__p__text14" css_pattern="div_9">
                    A simple and flexible architecture to take new
                    ideas from concept to code, to state-of-the-art
                    models, and to publication faster.
                </p>
            </div>
        </div>
    </div>
</section>
</body>
</html>
```

CSS3 CODE:

```
/*******************************/
/* Beginning of common styles. */
/*******************************/

/* Beginning of styles for block body. */


/* Beginning of styles for block button. */
button {
    display: block;
}


/* Beginning of styles for block div-image0. */


.div-image0__img__image0 {
    height: 92px;
    width: 123px;
}


/* Beginning of styles for block div-image2. */


.div-image2__img__image2 {
    height: 95px;
    width: 121px;
}


/* Beginning of styles for block div-image3. */


.div-image3__img__image3 {
    height: 93px;
    width: 110px;
}
```

```
/* Beginning of styles for block div_10. */
#div_10 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    width: 274px;
}


/* Beginning of styles for block div_11. */
#div_11 {
    align-items: flex-end;
    display: flex;
    flex-direction: column;
    justify-content: center;
    text-align: right;
    width: 858px;
}


/* Beginning of styles for block div_11-row_1. */
#div_11-row_1 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-right: 0px;
    margin-top: 40px;
    width: 676px;
}


/* Beginning of styles for block div_12. */
#div_12 {
    align-items: center;
    display: flex;
    flex-direction: column;
```

```
    justify-content: center;
    text-align: center;
    width: 483px;
}


.div_12__h2__text5 {
    color: rgb(94, 102, 115);
    font-size: 19px;
    margin-bottom: 0em;
}

.div_12__h2__text9 {
    color: rgb(127, 133, 141);
    font-size: 14px;
    margin-bottom: 0em;
    margin-top: 29px;
}


.div_12__p__text6 {
    color: rgb(166, 166, 166);
    font-size: 15px;
    margin-bottom: 0em;
    margin-top: 16px;
}


/* Beginning of styles for block div_7. */
#div_7 {
    align-items: center;
    display: flex;
    flex-direction: column;
    justify-content: center;
    margin-right: 5px;
    text-align: center;
    width: 244px;
}


.div_7__h2__text10 {
    color: rgb(127, 131, 139);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 13px;
}


.div_7__p__text11 {
    color: rgb(167, 167, 167);
    font-size: 14px;
    margin-bottom: 0em;
    margin-top: 15px;
}


/* Beginning of styles for block div_8. */
#div_8 {
    align-items: center;
    display: flex;
    flex-direction: column;
    justify-content: center;
    text-align: center;
    width: 238px;
}


.div_8__h2__text7 {
    color: rgb(126, 131, 138);
    font-size: 17px;
    margin-bottom: 0em;
    margin-top: 19px;
}


.div_8__p__text8 {
    color: rgb(166, 165, 166);
    font-size: 14px;
    margin-bottom: 0em;
    margin-top: 11px;
}


/* Beginning of styles for block div_9. */
#div_9 {
    align-items: center;
    display: flex;
    flex-direction: column;
    justify-content: center;
    text-align: center;
    width: 228px;
}


.div_9__h2__text13 {
    color: rgb(127, 133, 140);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 16px;
}


.div_9__p__text14 {
    color: rgb(166, 166, 166);
    font-size: 14px;
    margin-bottom: 0em;
    margin-top: 16px;
}


/* Beginning of styles for block header. */

.header__checkbox {
    display: none;
}


.header__img__image1 {
    margin-left: 6px;
    object-fit: cover;
    width: 120px;
}


/* Beginning of styles for block header0. */
#header0 {
    align-items: center;
    background-color: rgb(252, 252, 252);
    display: flex;
    justify-content: space-between;
    padding: 1px 0px;
    width: 100%;
}


/* Beginning of styles for block icon_menu. */
.icon_menu {
    cursor: pointer;
    display: none;
    font-size: 25px;
}


/* Beginning of styles for block nav0. */
```

```
.nav0 {
    align-items: center;
    display: flex;
    justify-content: center;
    width: 620px;
}


.nav0__a__txtnav0-0 {
    color: rgb(150, 154, 162);
    font-size: 14px;
    margin-bottom: 0.5em;
    margin-top: 0.5em;
    text-decoration: none;
}

.nav0__ul {
    align-items: center;
    display: flex;
    justify-content: space-between;
    list-style: none;
    width: 100%;
}


/* Beginning of styles for block nav1. */
.nav1 {
    align-items: center;
    display: flex;
    justify-content: center;
    margin-right: 10px;
    width: 346px;
}


.nav1__a__txtnav1-15 {
    color: rgb(141, 141, 145);
    font-size: 14px;
    margin-bottom: 0.5em;
    margin-top: 0.5em;
    text-decoration: none;
}

.nav1__ul {
    align-items: center;
    display: flex;
    justify-content: space-between;
    list-style: none;
    width: 100%;
}


/* Beginning of styles for block section_1. */
#section_1 {
    align-items: center;
    background-color: rgb(253, 253, 254);
    display: flex;
    justify-content: center;
    padding: 19px 0px;
    width: 100%;
}


/*****************************/
/* Beginning of media styles. */
/*****************************/

/* Beginning of styles for 480px width. */
@media(max-width: 480px) {
    #div_10 {
        width: 100%;
    }
    #div_7 {
        width: 100%;
    }
    #div_8 {
        width: 100%;
    }
    #div_9 {
        width: 100%;
    }
}

/* Beginning of styles for 800px width. */
@media(max-width: 800px) {
    #div_11-row_1 {
        width: 100%;
    }
    #div_12 {
        width: 100%;
    }
}

/* Beginning of styles for 1200px width. */
@media(max-width: 1200px) {
    #div_11 {
        width: 100%;
    }
    #header0 {
        flex-direction: row-reverse;
        justify-content: space-between;
    }
    #header0-div-image1 {
        margin-left: 15px;
    }
    .header__checkbox:checked ~ .nav0 {
        display: flex;
        flex-direction: column;
    }
    .header__checkbox:checked ~ .nav1 {
        display: flex;
        flex-direction: column;
    }
    .icon_menu {
        display: flex;
        margin-right: 15px;
    }
    .nav0 {
        display: none;
        position: absolute;
        width: 100%;
        top: 36px;
    }
    .nav0__ul {
        flex-direction: column;
        background-color: rgb(253, 253, 253);
    }
    .nav1 {
        display: none;
        position: absolute;
        width: 100%;
        top: 37px;
    }
    .nav1__ul {
        flex-direction: column;
        background-color: rgb(250, 250, 251);
    }
}
```

Assessment with the Proposed Evaluation System

The evaluation metric for this example (Figure: 6.11) provides a quantifiable measure of the system's effectiveness in reproducing the intended design.

Figure 6.11: Application of Evaluation Metrics via the Proposed Evaluation System: Highlighted here is the computed score, representing a distance metric of 0.2929388, which quantifies the similarity between the generated output and the original design.

6.3.2 Webpage Analysis: Lucichart Landing Page. Example 3.

Example 3 presents the primary page design together with the rendered code output produced by the system (Figure: 6.12). This side-by-side comparison facilitates a thorough assessment of the system's capability to faithfully reproduce the visual and structural components of the original webpage.

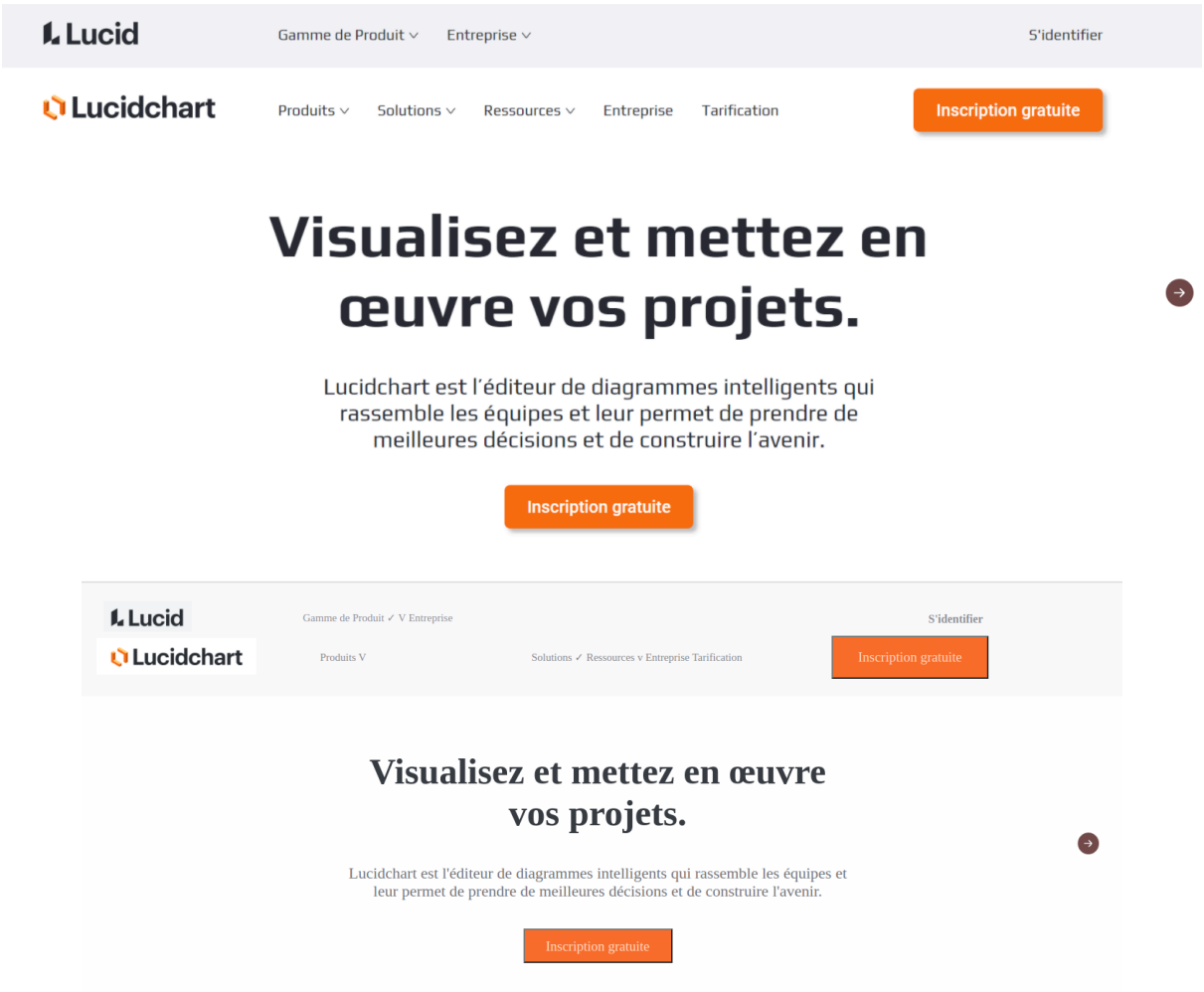


Figure 6.12: Illustrative Example of Outcomes from the Automatic Web Code Generation Model: This image showcases a specific instance of the results produced by the model, exemplifying its capability in translating web designs into code.

HTML5 CODE:

```
<!DOCTYPE html>
<html lang="en">
<!-- This is the <head> tag, where all meta data is defined. -->
    <head>
        <meta charset="utf-8"/>
        <meta content="width=device-width, initial-scale=1, shrink-to-fit=no" name="viewport"/>
        <link href="normalize.css" rel="stylesheet"/>
        <link href="styles_26.css" rel="stylesheet"/>
        <title>26</title>
    </head>
<!-- This is the <body> tag, where the whole webpage is allocated. -->
    <body>
<!-- This is the <head> tag, where all meta data is defined. -->
        <header id="header0">
            <input class="header__checkbox" id="header__checkbox" type="checkbox"/>
            <label class="icon_menu" for="header__checkbox">
```

```
                <i class="fas␣fa-align-justify"> </i>
            </label>
            <div css_pattern="header0" id="header__row_0-div">
                <div id="header__row_0-div-div-image2">
                    <img class="logo␣header__img__image2" css_pattern="header" id="image2" src="images/upload_test25lucidimage1.png"/>
                </div>
                <nav class="nav1" css_pattern="header__row_0-div">
                    <ul class="nav1__ul" css_pattern="nav1">
                        <li class="nav-item␣nav1__li" css_pattern="nav1" id="nav1-item0">
                            <a class="nav-link␣nav1__a__txtnav1-0" css_pattern="nav1" href="#">Gamme de Produit  V Entreprise</a>
                        </li>
                    </ul>
                </nav>
                <h3 class="header__h3__txtheader0-6" css_pattern="header">S'identifier</h3>
            </div>
            <div css_pattern="header0" id="header__row_1-div">
                <div id="header__row_1-div-div-image0">
                    <img class="logo␣header__img__image0" css_pattern="header" id="image0" src="images/upload_test25lucidimage2.png"/>
                </div>
                <nav class="nav0" css_pattern="header__row_1-div">
                    <ul class="nav0__ul" css_pattern="nav0">
                        <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item0">
                            <a class="nav-link␣nav0__a__txtnav0-1" css_pattern="nav0" href="#">Produits V</a>
                        </li>
                        <li class="nav-item␣nav0__li" css_pattern="nav0" id="nav0-item1">
                            <a class="nav-link␣nav0__a__txtnav0-1" css_pattern="nav0" href="#">
                                Solutions  Ressources v Entreprise Tarification
                            </a>
                        </li>
                    </ul>
                </nav>
                <button class="header__button__button1" css_pattern="header">Inscription gratuite</button>
            </div>
        </header>
<!-- This is the <section> tag, corresponding to "section_1". -->
        <section id="section_1">
            <div css_pattern="section_1" id="div_row_section_1_row_0" row="True">
                <div css_pattern="div_row_section_1_row_0" id="div_1" row="False">
                    <h1 class="div_1__h1__text3" css_pattern="div_1">
                        Visualisez et mettez en œuvre vos projets.
                    </h1>
                    <p class="div_1__p__text4" css_pattern="div_1">
                        Lucidchart est l'éditeur de diagrammes intelligents
                        qui rassemble les équipes et leur permet de prendre de
                        meilleures décisions et de construire l'avenir.
                    </p>
                    <button class="div_1__button__button0" css_pattern="div_1">Inscription gratuite</button>
                </div>
                <div css_pattern="div_row_section_1_row_0" id="div-section_1" row="True">
                    <div css_pattern="div-section_1" id="div-image1">
                        <img class="div-image1__img__image1" css_pattern="div-image1" src="images/upload_test25lucidimage3.png"/>
                    </div>
                </div>
            </div>
        </section>
    </body>
</html>
```

CSS3 CODE:

```
/*******************************/
/* Beginning of common styles. */
/*******************************/

/* Beginning of styles for block body. */


/* Beginning of styles for block button. */
button {
    display: block;
}


/* Beginning of styles for block div-image1. */
#div-image1 {
    margin-right: 5px;
}
```

```
.div-image1__img__image1 {
    height: 36px;
    width: 32px;
}


/* Beginning of styles for block div-section_1. */
#div-section_1 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    margin-right: 3px;
    width: 62px;
}


/* Beginning of styles for block div_1. */
#div_1 {
    align-items: center;
    display: flex;
    flex-direction: column;
    justify-content: center;
    margin-left: 5px;
    margin-right: 249px;
    text-align: center;
    width: 709px;
}


.div_1__button__button0 {
    background-color: rgb(247, 108, 42);
    border-radius: 0px;
    color: rgb(253, 220, 202);
    font-size: 19px;
    height: 48px;
    margin-top: 40px;
    padding: -14px -29px;
    width: 208px;
}


.div_1__h1__text3 {
    color: rgb(54, 58, 65);
    font-size: 51px;
    margin-bottom: 0em;
}


.div_1__p__text4 {
    color: rgb(109, 110, 116);
    font-size: 22px;
    margin-bottom: 0em;
    margin-top: 41px;
}


/* Beginning of styles for block div_row_section_1_row_0. */
#div_row_section_1_row_0 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    width: 1093px;
}


/* Beginning of styles for block header. */


.header__button__button1 {
    background-color: rgb(248, 108, 41);
    border-radius: 0px;
    color: rgb(253, 215, 196);
    font-size: 19px;
    height: 60px;
    margin-left: 125px;
    padding: -15px -37px;
    width: 219px;
}

.header__checkbox {
    display: none;
}


.header__h3__txtheader0-6 {
    color: rgb(139, 141, 148);
    font-size: 16px;
    margin-bottom: 0em;
    margin-left: 514px;
    margin-top: 4px;
}


.header__img__image0 {
    margin-left: 24px;
    object-fit: cover;
    width: 223px;
}

.header__img__image2 {
    margin-left: 34px;
    object-fit: cover;
    width: 123px;
}


/* Beginning of styles for block header0. */
#header0 {
    background-color: rgb(248, 248, 249);
    padding: 24px 0px;
    width: 100%;
}


/* Beginning of styles for block header__row_0-div. */
#header__row_0-div {
    align-items: center;
    display: flex;
    justify-content: flex-start;
    width: 100%;
}


/* Beginning of styles for block header__row_1-div. */
#header__row_1-div {
    align-items: center;
    display: flex;
    justify-content: flex-start;
    width: 100%;
}


/* Beginning of styles for block icon_menu. */
.icon_menu {
    cursor: pointer;
    display: none;
    font-size: 25px;
}


/* Beginning of styles for block nav0. */
```

```
.nav0 {
    align-items: center;
    display: flex;
    justify-content: center;
    margin-left: 49px;
    width: 629px;
}


.nav0__a__txtnav0-1 {
    color: rgb(133, 134, 140);
    font-size: 15px;
    margin-bottom: 0.5em;
    margin-top: 0.5em;
    text-decoration: none;
}

.nav0__ul {
    align-items: center;
    display: flex;
    justify-content: space-between;
    list-style: none;
    width: 100%;
}


/* Beginning of styles for block nav1. */
.nav1 {
    align-items: center;
    display: flex;
    justify-content: center;
    margin-left: 115px;
    width: 399px;
}


.nav1__a__txtnav1-0 {
    color: rgb(142, 143, 148);
    font-size: 15px;
    margin-bottom: 0.5em;
    margin-top: 0.5em;
    text-decoration: none;
}

.nav1__ul {
    align-items: center;
    display: flex;
    justify-content: space-between;
    list-style: none;
    width: 100%;
}


/* Beginning of styles for block section_1. */
#section_1 {
    align-items: center;
    background-color: rgb(254, 254, 254);
    display: flex;
    justify-content: flex-end;
    padding: 41px 0px;
    width: 100%;
}


/*****************************/
/* Beginning of media styles. */
/*****************************/

/* Beginning of styles for 480px width. */
@media(max-width: 480px) {
    #div-section_1 {
        width: 100%;
    }
}

/* Beginning of styles for 800px width. */
@media(max-width: 800px) {
    #div_1 {
        width: 100%;
    }
}

/* Beginning of styles for 1200px width. */
@media(max-width: 1200px) {
    #div_row_section_1_row_0 {
        width: 100%;
    }
    #header0 {
        flex-direction: row-reverse;
        justify-content: space-between;
    }
    #header__row_0-div-div-image2 {
        margin-left: 15px;
    }
    #header__row_1-div-div-image0 {
        margin-left: 15px;
    }
    .header__checkbox:checked ~ .nav0 {
        display: flex;
        flex-direction: column;
    }
    .header__checkbox:checked ~ .nav1 {
        display: flex;
        flex-direction: column;
    }
    .icon_menu {
        display: flex;
        margin-right: 15px;
    }
    .nav0 {
        display: none;
        position: absolute;
        width: 100%;
        top: 146px;
    }
    .nav0__ul {
        flex-direction: column;
        background-color: rgb(253, 253, 253);
    }
    .nav1 {
        display: none;
        position: absolute;
        width: 100%;
        top: 58px;
    }
    .nav1__ul {
        flex-direction: column;
        background-color: rgb(240, 241, 243);
    }
}
```

Assessment with the Proposed Evaluation System

The evaluation metric for this example (Figure: 6.13) provides a quantifiable measure of the system's effectiveness in reproducing the intended design.

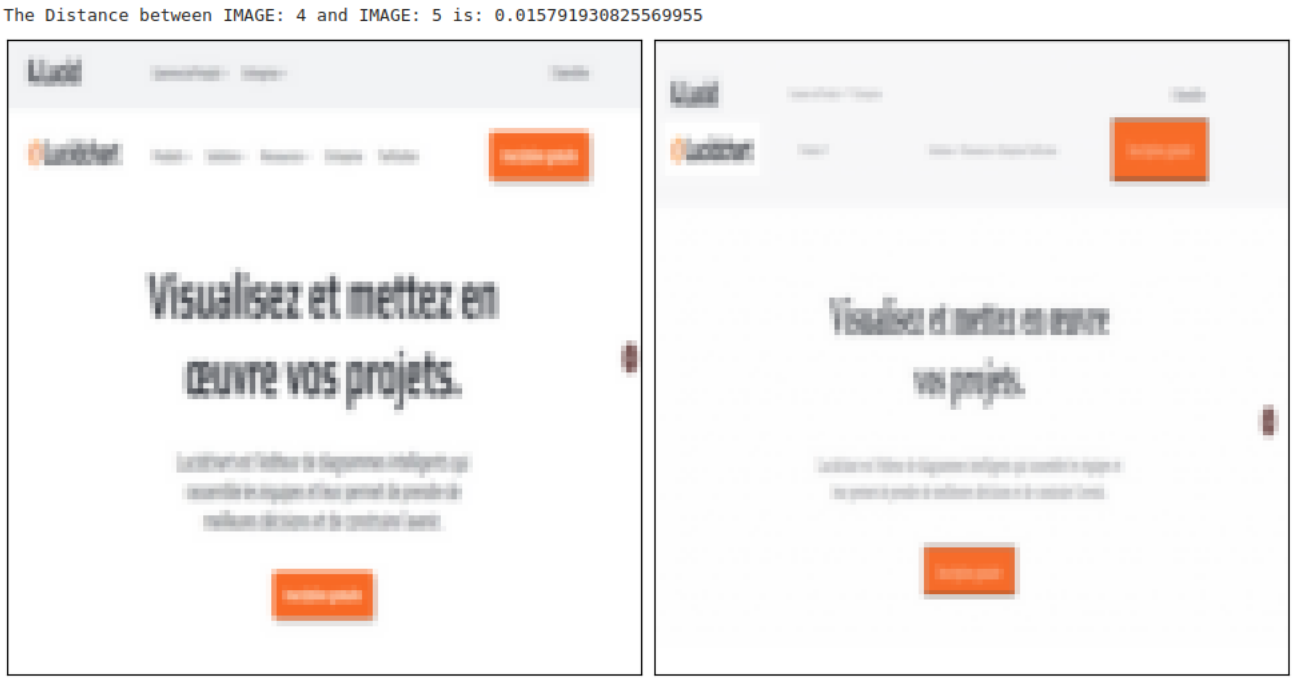


Figure 6.13: Application of Evaluation Metrics via the Proposed Evaluation System: Highlighted here is the computed score, representing a distance metric of 0.01579193, which quantifies the similarity between the generated output and the original design.

6.3.3 Webpage Analysis: YoutubeMusic Landing Page. Example 4.

Example 4 displays the initial page design alongside the rendered code generated by the system (Figure: 6.14). This juxtaposition enables a comprehensive evaluation of the system's proficiency in accurately mirroring the visual and structural aspects of the original webpage.

Figure 6.14: Illustrative Example of Outcomes from the Automatic Web Code Generation Model: This image showcases a specific instance of the results produced by the model, exemplifying its capability in translating web designs into code.

HTML5 code:

```
<!DOCTYPE html>
<html lang="en">
<!-- This is the <head> tag, where all meta data is defined. -->
    <head>
        <meta charset="utf-8"/>
        <meta content="width=device-width, initial-scale=1, shrink-to-fit=no" name="viewport"/>
        <link href="normalize.css" rel="stylesheet"/>
        <link href="styles_33.css" rel="stylesheet"/>
        <title>33</title>
    </head>
<!-- This is the <body> tag, where the whole webpage is allocated. -->
    <body>
<!-- This is the <section> tag, corresponding to "section_3". -->
        <section id="section_3">
            <div id="div_11" row="True">
                <div id="div_8" row="False">
                    <div css_pattern="div_8" id="div-image0">
                        <img class="div-image0__img__image0" css_pattern="div-image0" src="images/IMG_PATH.png"/>
                    </div>
                    <h2 class="div_8__h2__text0" css_pattern="div_8">App de música de YouTube</h2>
                    <p class="div_8__p__text1" css_pattern="div_8">
                        Un nuevo servicio de música con una app y experiencia
                        para ordenadores totalmente reinventadas.
                    </p>
                </div>
                <div id="div_7" row="False">
                    <div css_pattern="div_7" id="div-image2">
                        <img class="div-image2__img__image2" css_pattern="div-image2" src="IMG_PATH"/>
                    </div>
                    <h3 class="div_7__h3__text2" css_pattern="div_7">
                        Encuentra tu música
                    </h3>
                    <p class="div_7__p__text3" css_pattern="div_7">
                        Encuentra fácilmente álbumes, actuaciones en directo
                        y remixes buscando letras o describiendo la canción.
                    </p>
                </div>
                <div id="div_9" row="False">
                    <div css_pattern="div_9" id="div-image3">
                        <img class="div-image3__img__image3" css_pattern="div-image3" src="IMG_PATH"/>
                    </div>
                    <h3 class="div_9__h3__text6" css_pattern="div_9">Descubre con</h3>
                    <h3 class="div_9__h3__text7" css_pattern="div_9">facilidad</h3>
                    <p class="div_9__p__text8" css_pattern="div_9">
                        Escucha nueva música sugerida según tus gustos
                        y de las tendencias de tu comunidad.
                    </p>
                </div>
            </div>
        </section>
<!-- This is the <section> tag, corresponding to "section_2". -->
        <section id="section_2">
            <div id="div_12" row="True">
                <div id="div_10" row="False">
                    <h1 class="div_10__h1__text4" css_pattern="div_10">
                        Un nuevo servicio de música en streaming
                    </h1>
                    <p class="div_10__p__text5" css_pattern="div_10">
                        YouTube Music es una app y experiencia para
                        ordenadores totalmente reinventadas dándote
                        acceso a álbumes oficiales, listas de reproducción,
                        sencillos y mucho más.
                    </p>
                </div>
                <div css_pattern="div_12" id="div-image1">
                    <img class="div-image1__img__image1" css_pattern="div-image1" src="IMG_PATH"/>
                </div>
            </div>
        </section>
```

```
    </body>
</html>
```

CSS3 CODE:

```
/*******************************/
/* Beginning of common styles. */
/*******************************/

/* Beginning of styles for block body. */


/* Beginning of styles for block button. */
button {
    display: block;
}


/* Beginning of styles for block div-image0. */


.div-image0__img__image0 {
    height: 96px;
    width: 98px;
}


/* Beginning of styles for block div-image1. */
#div-image1 {
    margin-right: 5px;
}


.div-image1__img__image1 {
    height: 390px;
    width: 556px;
}


/* Beginning of styles for block div-image2. */


.div-image2__img__image2 {
    height: 89px;
    width: 90px;
}


/* Beginning of styles for block div-image3. */


.div-image3__img__image3 {
    height: 92px;
    width: 96px;
}


/* Beginning of styles for block div_10. */
#div_10 {
    align-items: flex-start;
    display: flex;
    flex-direction: column;
    justify-content: center;
    margin-left: 5px;
    text-align: left;
    width: 497px;
}


.div_10__h1__text4 {
    color: rgb(98, 98, 98);
    font-size: 37px;
    margin-bottom: 0em;
}


.div_10__p__text5 {
    color: rgb(103, 102, 103);
    font-size: 19px;
    margin-bottom: 0em;
    margin-top: 32px;
}


/* Beginning of styles for block div_11. */
#div_11 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    width: 999px;
}


/* Beginning of styles for block div_12. */
#div_12 {
    align-items: center;
    display: flex;
    flex-wrap: wrap;
    justify-content: space-between;
    width: 1100px;
}


/* Beginning of styles for block div_7. */
#div_7 {
    align-items: center;
    display: flex;
    flex-direction: column;
    justify-content: center;
    text-align: center;
    width: 314px;
}


.div_7__h3__text2 {
    color: rgb(93, 91, 91);
    font-size: 24px;
    margin-bottom: 0em;
    margin-top: 19px;
}


.div_7__p__text3 {
    color: rgb(147, 147, 147);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 17px;
}


/* Beginning of styles for block div_8. */
#div_8 {
```

```
    align-items: center;
    display: flex;
    flex-direction: column;
    justify-content: center;
    margin-left: 5px;
    text-align: center;
    width: 310px;
}


.div_8__h2__text0 {
    color: rgb(98, 96, 97);
    font-size: 27px;
    margin-bottom: 0em;
    margin-top: 13px;
}


.div_8__p__text1 {
    color: rgb(146, 146, 146);
    font-size: 16px;
    margin-bottom: 0em;
    margin-top: 14px;
}


/* Beginning of styles for block div_9. */
#div_9 {
    align-items: center;
    display: flex;
    flex-direction: column;
    justify-content: center;
    margin-right: 5px;
    text-align: center;
    width: 306px;
}


.div_9__h3__text6 {
    color: rgb(93, 91, 91);
    font-size: 25px;
    margin-bottom: 0em;
    margin-top: 15px;
}

.div_9__h3__text7 {
    color: rgb(94, 94, 94);
    font-size: 25px;
    margin-bottom: 0em;
    margin-top: 14px;
}


.div_9__p__text8 {
    color: rgb(148, 147, 148);
    font-size: 17px;
    margin-bottom: 0em;
    margin-top: 14px;
}
```

```
/* Beginning of styles for block section_2. */
#section_2 {
    align-items: center;
    background-color: rgb(253, 253, 253);
    display: flex;
    justify-content: center;
    padding: 163px 0px;
    width: 100%;
}


/* Beginning of styles for block section_3. */
#section_3 {
    align-items: center;
    background-color: rgb(249, 249, 248);
    display: flex;
    justify-content: center;
    padding: 28px 0px;
    width: 100%;
}


/*****************************/
/* Beginning of media styles. */
/*****************************/

/* Beginning of styles for 480px width. */
@media(max-width: 480px) {
    #div_7 {
        width: 100%;
    }
    #div_8 {
        width: 100%;
    }
    #div_9 {
        width: 100%;
    }
}

/* Beginning of styles for 800px width. */
@media(max-width: 800px) {
    #div_10 {
        width: 100%;
    }
}

/* Beginning of styles for 1200px width. */
@media(max-width: 1200px) {
    #div_11 {
        width: 100%;
    }
    #div_12 {
        width: 100%;
    }
}
```

Assessment with the Proposed Evaluation System

The evaluation metric for this example (Figure: 6.15) provides a quantifiable measure of the system's effectiveness in reproducing the intended design.

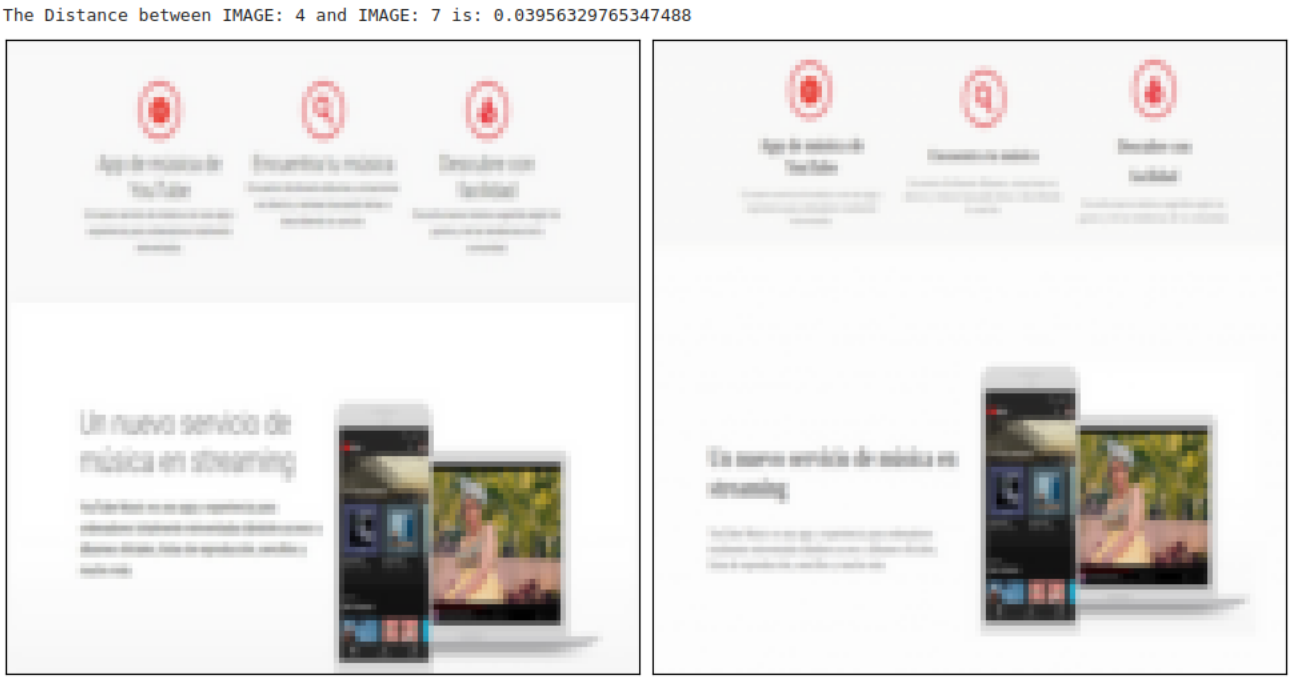


Figure 6.15: Application of Evaluation Metrics via the Proposed Evaluation System: Highlighted here is the computed score, representing a distance metric of 0.03956, which quantifies the similarity between the generated output and the original design.

7 Conclusions

7.1 Summary of Key Findings

1. **Development of an Automated System:** A system was successfully developed that translates images of website designs into functional HTML5 and CSS3 code. This system integrates technologies as Convolutional Neural Networks (CNNs) for element detection and Optical Character Recognition (OCR) for text extraction.

2. **Accuracy in Web Element Detection:** The system demonstrated a high degree of accuracy in identifying and segmenting various web elements such as headers, paragraphs, and images. This precision played a critical role in the successful generation of web code.

3. **Effective Code Generation:** The research showed that the system could effectively generate structured web code, maintaining the integrity and styles of the original design. The application of CSS Flexbox in code generation ensured responsive and orderly layouts.

4. **Evaluation System Development:** A significant achievement was the development of an evaluation system designed to measure image similarity between the original design and the generated web code. This system, utilizing encoder to produce embedding, provided an objective means to quantify the system's effectiveness in replicating design elements.

7.1.1 Evaluation of the Hypothesis and Results

The primary hypothesis supporting this research was that web design images could be effectively converted into functional HTML5 and CSS3 code using Convolutional Neural Networks (CNNs) and advanced machine learning techniques. The secondary hypothesis suggested that an evaluation system using autoencoders could be developed to measure the similarity between the generated code and the original designs.

Hypothesis 1: Conversion of Design Images to Web Code

- **Evidence of Success:** The system developed during this research has demonstrated a strong capability to translate design elements into web code with high fidelity (Results: Chapter 6).
- **Areas for Improvement:** While the system performs well with standard web designs, it struggles with more complex scenarios (Section: 4.7).

Hypothesis 2: Development of an Evaluation System

- **Evidence of Success:** The evaluation system utilizing autoencoders has successfully provided a consistent metric for assessing the similarity between the original design images and the generated web pages, indicating a strong resemblance between the two (Results: Chapter 6).
- **Areas for Improvement:** Despite the effectiveness of using autoencoders for feature extraction and cosine distance for measuring similarity, the results highlight an essential need for enhancements. The evaluation system currently shows limitations, especially in its ability to distinguish between web pages that exhibit only subtle structural differences. To address this, future enhancements should aim at refining the evaluation metrics to improve their accuracy and broaden their effectiveness in varying web design contexts (Section: 5.4).

7.2 Achieved Results

The overall results are promising and validate the feasibility of using machine learning techniques for automated web code generation. The success observed in the initial implementations suggests that with further refinement, the system could become a tool for web developers, significantly reducing the time and effort required for web design coding.

The successful results of the automatic web code generation system can be attributed to a series of methodical strategies, advanced technologies, and innovative machine learning techniques. This section details the specific processes and approaches that contributed significantly to achieving these results.

- **Convolutional Neural Networks (CNNs):** Central to the system's ability to interpret and translate website designs into functional code, CNNs were utilized to detect and classify web elements from images. These networks were trained on a vast dataset of web design images, which were labeled to identify various elements such as headers, buttons, and navigation bars. The accuracy of element detection was crucial in generating semantically correct HTML and CSS code.
- **Optical Character Recognition (OCR):** OCR technology was employed to accurately extract text from the web designs.
- **Template Matching and Machine Learning Algorithms:** Sophisticated template matching techniques ensured to identify images that are inside of the web page design. Machine learning algorithms, such as K-Means clustering, were applied to categorize and prioritize web elements, aiding in the accurate structural representation of the design.
- **Autoencoders in the Evaluation System:** The integration of an evaluation system using autoencoders provided a metric for measuring the similarity between the original designs and the generated web pages.

7.3 Limitations and Areas for Improvement

While the automatic web code generation system has demonstrated significant capabilities, it is crucial to acknowledge its limitations and identify areas for future improvement, which will guide the next phases of research and development.

- **Handling Complex Web Designs:** The system's performance varies significantly with complex web designs, where it struggles to accurately interpret and replicate intricate layouts and design elements.
  - *Area for Improvement:* Future development should focus on enhancing the system's algorithms to better manage complex designs and incorporate more advanced machine learning techniques that can adapt to diverse and dynamic design elements.
- **Text Recognition Accuracy:** While the OCR component performs adequately in most scenarios, it faces challenges with stylized fonts and busy backgrounds, resulting in less accurate text extraction.
  - *Area for Improvement:* Further refinement of OCR technology is required, potentially through the integration of more sophisticated neural network models that specialize in text recognition under varied conditions.

- **Evaluation Metrics Refinement:** The current evaluation system, while effective in measuring the structural similarity between web pages using cosine distance, may not fully capture nuanced differences in fonts, style, and element arrangement.
  - *Area for Improvement:* Incorporate a multi-faceted approach to similarity assessment by combining cosine distance with other metrics like SSIM or FSIM. Explore pixel-level comparisons or semantic similarity metrics to provide a more comprehensive evaluation. Develop metrics to assess functional correctness.

7.4 General conclusions

The research into developing and evaluating the web code generation model has yielded predominantly positive results, though these are underscored by areas necessitating further refinement. The results obtained are commendable, especially considering this is the inaugural version of the system. The model demonstrates robust capabilities in translating website designs into functional and aesthetically aligned web code, effectively leveraging a suite of advanced technologies, including Convolutional Neural Networks (CNNs), Optical Character Recognition (OCR), and sophisticated computer vision techniques like Template Matching. Additionally, it utilizes Machine Learning algorithms such as K-Means clustering, complemented by the application of autoencoders within the evaluation system.

Challenges arise, particularly when the system faces complex website designs. In these cases, the system tends to struggle with accurately interpreting and translating design elements, leading to ambiguities and inconsistencies, especially in container definitions. Such difficulties often result in 'noise' within the layout, compromising the clarity and precision of the final code structure. This indicates an urgent need for enhancement in the model's ability to handle diverse and sophisticated design elements more effectively.

Despite these challenges, it is crucial to recognize the achievements of this initial version. The system has demonstrated admirable performance across a wide range of scenarios, showcasing its versatility and effectiveness in various design contexts. This versatility lays a strong foundation for future improvements and expansions of the system.

Looking forward, the potential for further development and refinement of the system is substantial. The current challenges provide valuable insights that will guide the direction for future research and development efforts.

Enhancements in the model's design interpretation capabilities, along with advancements in managing complex layouts, will be crucial for the next stages of development. Moreover, integrating user feedback mechanisms and exploring the application of emerging AI technologies could significantly elevate the system's performance and usability.

With respect to the evaluation system designed to assess the similarity between the original web page design and the rendered output from the generated code has proven to be a valuable tool. By leveraging autoencoders and cosine distance metrics, the system offers a quantitative measure of how well the generated code replicates the original design. The results obtained from the evaluation system have been instrumental in identifying areas where the code generation process excels and areas that require further refinement. The system's ability to pinpoint discrepancies between the original and generated outputs has provided valuable insights for improving the overall performance and accuracy of the web code generation model. However, the evaluation system, in its current state, has limitations, particularly in distinguishing between web pages with subtle structural differences. This limitation underscores the need for further research and development to enhance the system's sensitivity and discriminatory power. Future work could explore incorporating additional metrics, such as a merge between Cosine and Euclidean distance or pixel-by-pixel comparisons, to complement and provide a more comprehensive evaluation of similarity.

In conclusion, while the system in its current form exhibits areas for improvement, the overall results are promising and illustrate the feasibility and potential of automated web code generation. The insights gained also highlight significant opportunities for advancing the technology further, ensuring its practical application and relevance in real-world scenarios.

References

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016.

[2] Gartner, "Top strategic technology trends for 2023," 2022.

[3] F. Research, "The state of agile software development," 2023.

[4] B. Boehm, *Software Engineering Economics*. Prentice Hall, 2021.

[5] I. Sommerville, *Software Engineering*. Pearson, 2016.

[6] R. Kazman *et al.*, *Software Architecture in Practice*. Addison-Wesley Professional, 2020.

[7] M. Research, "The future of software development: Ai-assisted coding," 2022.

[8] T. B. Brown *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[9] OpenAI, "Openai codex," *OpenAI Blog*, 2022.

[10] GitHub, "Github copilot," 2023.

[11] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 2020.

[12] X. Chen *et al.*, "Automatic web application development and its challenges," in *IEEE International Conference on Web Services (ICWS)*, 2018.

[13] T. Beltramelli, "pix2code: Generating code from a graphical user interface screenshot," *arXiv preprint arXiv:1705.07962*, 2017.

[14] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow, "Deepcoder: Learning to write programs," *International Conference on Learning Representations*, 2018.

[15] N. Rasiwasia, J. C. Pereira, E. Coviello, G. Doyle, G. R. Lanckriet, R. Levy, and N. Vasconcelos, "A comprehensive survey on cross-modal retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014.

[16] H. Wickham and G. Grolemund, *R for data science: import, tidy, transform, visualize, and model data*. O'Reilly Media, Inc., 2017.

[17] S. S. Stevens, "On the theory of scales of measurement," *Science*, vol. 103, no. 2684, pp. 677–680, 1946.

[18] T. P. Vartanian, *Secondary data analysis.* Oxford University Press, 2010.

[19] H. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi, "Big data and its technical challenges," *Communications of the ACM*, vol. 57, no. 7, pp. 86–94, 2014.

[20] C. A. Mertler and R. V. Reinhart, *Advanced and multivariate statistical methods: practical application and interpretation.* Routledge, 2016.

[21] M. Lichman, "Uci machine learning repository," 2013.

[22] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Pearson, 2022.

[23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition.* Ieee, 2009, pp. 248–255.

[24] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[25] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[26] F. L. Coolidge, *Statistics: A Gentle Introduction.* SAGE Publications, 2019.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016.

[28] C. M. Bishop, *Pattern Recognition and Machine Learning.* Springer, 2006.

[29] A. Halevy, P. Norvig, and F. Pereira, "The unreasonable effectiveness of data," *IEEE Intelligent Systems*, vol. 24, no. 2, pp. 8–12, 2009.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[31] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean, "A guide to deep learning in healthcare," *Nature Medicine*, vol. 25, no. 1, pp. 24–29, 2019.

[32] C. Batini and M. Scannapieco, *Data and Information Quality: Dimensions, Principles and Techniques.* Springer, 2016.

[33] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[34] S. Barocas and A. D. Selbst, "Big data's disparate impact," *California Law Review*, pp. 671–732, 2016.

[35] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A survey on bias and fairness in machine learning," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–35, 2021.

[36] S. García, J. Luengo, and F. Herrera, *Data Preprocessing in Data Mining.* Springer, 2016.

[37] J. Han, J. Pei, and M. Kamber, *Data Mining: Concepts and Techniques*. Elsevier, 2011.

[38] S. Van Buuren, *Flexible Imputation of Missing Data*. Chapman and Hall/CRC, 2018.

[39] P. J. Rousseeuw and M. Hubert, "Robust statistics for outlier detection," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 73–79, 2011.

[40] A. K. Jain, R. P. Duin, and J. Mao, "Statistical pattern recognition: A review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4–37, 2000.

[41] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[42] Tzutalin, "Labelimg," https://github.com/HumanSignal/labelImg, 2023, accessed: 2023-05-10.

[43] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[44] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[45] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. IEEE, 2001, pp. I–511.

[46] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[47] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[48] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[49] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[50] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, Conference Track Proceedings*, 2015.

[51] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[52] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[53] C. Sun, R. Shetty, R. Sukthankar, and R. Nevatia, "Optimizing deep cnn-based queries over video streams at scale," *arXiv preprint arXiv:1903.02503*, 2019.

[54] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[55] M. T. Ribeiro, S. Singh, and C. Guestrin, "Should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 2016, pp. 1135–1144.

[56] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd International Conference on Learning Representations, ICLR 2015, Conference Track Proceedings*, 2015.

[57] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" pp. 3320–3328, 2014.

[58] J. Buolamwini and T. Gebru, "Gender shades: Intersectional accuracy disparities in commercial gender classification," in *Conference on fairness, accountability and transparency*, 2018, pp. 77–91.

[59] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CVPR*, 2016.

[60] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *ICML*, 2019.

[61] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *ICLR*, 2021.

[62] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *NeurIPS*, 2015.

[63] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CVPR*, 2016.

[64] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," *CVPR*, 2020.

[65] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[66] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," *ECCV*, 2014.

[67] M. Huh, P. Agrawal, and A. A. Efros, "What makes imagenet good for transfer learning?" in *NeurIPS Workshops*, 2016.

[68] R. B. Girshick, "Fast r-cnn," *ICCV*, 2015.

[69] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *CVPR*, 2017.

[70] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," *Proceedings of the 23rd international conference on Machine learning*, pp. 233–240, 2006.

[71] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," in *International journal of computer vision*, vol. 88, no. 2. Springer, 2010, pp. 303–338.

[72] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 658–666.

[73] M. Sokolova and G. Lapalme, "Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation," *Australasian Joint Conference on Artificial Intelligence*, vol. 4304, pp. 1015–1021, 2006.

[74] X. Yang, T. Zhang, D. He, Z. Liu, H. Zhu, and X. Fu, "Revisiting the evaluation of object detection and proposal: Measure ap correctly," in *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 2805–2809.

[75] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[76] Y.-l. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," *ICML*, vol. 10, pp. 111–118, 2010.

[77] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," *ICML*, vol. 27, pp. 807–814, 2010.

[78] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[79] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016, pp. 379–387.

[80] ——, "R-FCN: object detection via region-based fully convolutional networks," *CoRR*, vol. abs/1605.06409, 2016. [Online]. Available: http://arxiv.org/abs/1605.06409

[81] (2023) rfcn-resnet101-coco-tf — openvino™ documentation. [Online]. Available: https://docs.openvino.ai/latest/openvino_docs_pretrained_models_rfcn_resnet101_coco_tf.html

[82] J. Li, Y. Li, G. Li, Z. Jin, Y. Hao, and X. Hu, "Skcoder: A sketch-based approach for automatic code generation," *ar5iv*, 2023. [Online]. Available: https://ar5iv.org/abs/2302.06144

[83] gabegrand, "Lilo: Learning interpretable libraries by compressing and documenting code." Papers With Code, 2023. [Online]. Available: https://paperswithcode.com/paper/lilo-learning-interpretable-libraries-by

[84] "Codegen: An open large language model for code with multi ..." arXiv preprint arXiv:2203.13474, 2022. [Online]. Available: https://arxiv.org/abs/2203.13474

[85] X. Chen, "Automatic code documentation generation using gpt-3," in *Proceedings of the ACM*. ACM, 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/nnnnnnn.nnnnnnn

[86] M. AI, "Introducing code llama, a state-of-the-art large language model for coding," Aug 2023, blog post. [Online]. Available: https://ai.meta.com/blog/code-llama

[87] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[88] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 886–893.

[89] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," in *Advances in neural information processing systems*, 1993.

[90] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[91] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, "Residual attention network for image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3156–3164.

[92] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[93] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[94] L. Zhang, L. Zhang, X. Mou, and D. Zhang, "Feature similarity index for image quality assessment," *Signal Processing: Image Communication*, vol. 26, no. 7, pp. 343–358, 2011.

[95] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[96] J. Masci, U. Meier, D. Ciresan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *Artificial Neural Networks and Machine Learning–ICANN 2011*. Springer, 2011, pp. 52–59.

[97] L. Van Der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[98] A. Ng, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, pp. 1–19, 2011.

[99] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[100] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," in *Journal of Machine Learning Research*, vol. 11, 2010, pp. 3371–3408.

[101] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2002, pp. 311–318. [Online]. Available: https://www.aclweb.org/anthology/P02-1040/

[102] Microsoft, "Sketch2code: Transforming hand-drawn designs into html code using ai," 2018, available at: https://www.microsoft.com/en-us/ai/ai-lab-sketch2code.

[103] R. Patel and M. Gupta, "A comparative study of image similarity techniques," *Journal of Computer Vision*, 2015.

[104] A. Singh and P. Malik, "Human perception in image analysis: Challenges and techniques," *Cognitive Science Journal*, 2017.

[105] M. Daniels and F. Lee, "Feedback systems in design tools: A review," *Design and Technology Review*, 2020.

[106] L. Torres and M. Hernandez, "A review of automated web code generation algorithms," *Journal of Web Development*, 2021.

[107] K. Wong and L. Chang, "Autoencoders in deep learning: An overview," *Journal of Neural Networks*, 2016.

[108] V. Nair and A. Rai, "Image similarity assessment using autoencoders," in *Proceedings of the International Conference on Image Processing*, 2018.

[109] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," in *Biological cybernetics*, vol. 59, no. 4-5, 1988, pp. 291–294.

[110] TensorFlow, "Autoencoder," 2023, accessed: 2023-10-30. [Online]. Available: https://www.tensorflow.org/tutorials/generative/autoencoder

[111] Y. Wang, B. Zhang, and Q. Tian, "Deep learning for image similarity: An end-to-end training approach," *Multimedia Tools and Applications*, vol. 75, no. 21, pp. 12 531–12 549, 2016.

[112] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 665–674, 2017.

[113] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Image retrieval: Ideas, influences, and trends of the new age," *ACM Computing Surveys (CSUR)*, vol. 40, no. 2, pp. 1–60, 2008.

[114] F. Zhang, S. Li, S. Wang, L. Ma, and K. N. Ngan, "Perceptual quality assessment of images: A survey," *Journal of Visual Communication and Image Representation*, vol. 55, pp. 130–154, 2018.

[115] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1096–1103.

A

Docker environment setup (Environment to run the project).

In the development of the system, a Docker Compose file was utilized to configure and manage containers. This approach is critical for several reasons:

1. **Portability**: By using Docker, the system becomes hardware-agnostic, allowing it to run seamlessly on any computer with Docker installed. This eliminates the ”works on my machine” problem, ensuring consistent behavior across different environments.

2. **Isolation**: Each component of the system is encapsulated within its container. This isolation prevents conflicts between components and simplifies the management of dependencies.

3. **Scalability**: Docker Compose allows for easy scaling of components. Components can be replicated or scaled down based on demand, enhancing the system's ability to handle varying workloads.

4. **Version Control and Consistency**: Containers are defined by code in the Docker Compose file. This approach enables version control for the environment, ensuring consistency across development, testing, and production stages.

5. **Rapid Deployment and Testing**: Docker's containerization enables quick setup and teardown of environments. This facilitates faster development cycles and more efficient testing processes.

6. **Resource Efficiency**: Containers share the host system's kernel and, therefore, are more lightweight and resource-efficient compared to traditional virtualization approaches.

A.1 Dockerfile Code:

```
FROM ubuntu:18.04

# LOCALTIME SETUP
ENV TZ=America/Mexico_City
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
RUN apt-get update && apt-get -y update

# PYTHON AND IX2CODE SETUP
RUN apt install -y sl
RUN apt-get install -y build-essential python3.6 python3-pip python3-dev
RUN apt install -y libsm6 libxext6
RUN apt-get -y install cmake protobuf-compiler
RUN apt-get install -y libxrender1 libfontconfig1
RUN apt-get install -y  libpq-dev
RUN apt-get install -y libopenmpi-dev
RUN apt-get install -y python3-pil python3-lxml
RUN apt-get install -y imagemagick
RUN apt-get install -y screen
RUN apt-get install -y webp
RUN apt install -y libgl1-mesa-glx
RUN apt install -y curl
RUN apt install -y git
RUN apt install -y zip unzip
RUN apt install -y nano
RUN apt install -y wget
RUN apt install -y xvfb
RUN apt-get install -y wkhtmltopdf
RUN apt install -y rustc

# ELIXIR AND PHOENIX SETUP
RUN apt-get -y update
RUN apt-get install -y --no-install-recommends locales
RUN curl -sL https://deb.nodesource.com/setup_12.x -o nodesource_setup.sh
RUN bash nodesource_setup.sh
RUN apt install -y nodejs
RUN apt install -y build-essential
RUN apt-get install -y inotify-tools
RUN wget https://packages.erlang-solutions.com/erlang-solutions_2.0_all.deb
RUN dpkg -i erlang-solutions_2.0_all.deb
RUN apt-get update
RUN apt-get install -y esl-erlang
RUN apt-get install -y elixir

RUN apt-get -y update

# USER CREATION
ARG USER=yuyu
ARG HOME=/home/$USER
ARG USER_ID=1000

ENV USER=$USER \
    HOME=$HOME

RUN groupadd $USER
RUN useradd -u $USER_ID -d $HOME -g $USER -ms /bin/bash $USER

# INSTALL PYTHON REQUIREMENTS FOR IX2CODE
USER $USER
RUN pip3 install --upgrade pip
RUN pip3 install transformers
RUN pip3 install opencv-python
RUN pip3 install botocore
RUN pip3 install sklearn
RUN pip3 install scikit-learn
RUN pip3 install scikit-image
RUN pip3 install matplotlib
RUN pip3 install mpi4py
RUN pip3 install pandas
RUN pip3 install setuptools
RUN pip3 install tensorflow==1.15.5
RUN pip3 install keras==2.1.6
RUN pip3 install syspath
RUN pip3 install lockfile
RUN pip3 install tokenizer
RUN pip3 install tokenizers
RUN pip3 install Keras==2.3.1
RUN pip3 install Keras-Applications==1.0.8
RUN pip3 install boto3
RUN pip3 install imutils
RUN pip3 install Cython
RUN pip3 install h5py==2.10.0
RUN pip3 install contextlib2
RUN pip3 install tf_slim
RUN pip3 install pillow
RUN pip3 install --upgrade opencv-python
RUN pip3 install --upgrade google-cloud-vision==0.25
RUN pip3 install simpy.io
RUN pip3 install regex
RUN pip3 install psycopg2
```

```
RUN pip3 install resnet
RUN pip3 install python-magic
RUN pip3 install pytest
RUN pip3 install fastapi
RUN pip3 install "uvicorn[standard]"
RUN pip3 install openai

# AWS SETUP
WORKDIR $HOME
RUN mkdir -p $HOME/.aws
RUN curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
RUN unzip awscliv2.zip
USER root
RUN ./aws/install
USER $USER
COPY credentials .aws
COPY config .aws


# YOLO REQUIREMENTS
COPY yolo_requirements.txt $HOME
RUN pip3 install -r $HOME/yolo_requirements.txt

# TF OBJECT DETECTION SETUP
RUN git clone https://github.com/tensorflow/models.git
RUN pip3 install slim
RUN pip3 install tf-slim
RUN cd models/research \
    && protoc object_detection/protos/*.proto --python_out=. \
    && cp object_detection/packages/tf1/setup.py . \
    && python3 -m pip install .
RUN echo "export PYTHONIOENCODING=utf-8" >> .bashrc

# Error in test
# test_create_ssd_models_from_config
#RUN python3 $HOME/models/research/object_detection/builders/model_builder_tf1_test.py

# MIX AND PHOENIX INSTALLATION
RUN mix local.hex --force
RUN mix archive.install hex phx_new 1.5.9

# WORKSPACE FOLDER
RUN mkdir -p $HOME/workspace

#UTF CONFIGURATIONS
USER root
ENV LANG=en_US.UTF-8
RUN echo $LANG UTF-8 > /etc/locale.gen
RUN locale-gen
RUN update-locale LANG=$LANG

# ADD ENV VARIABLES
ENV PYTHONPATH=$PYTHONPATH:/tensorflow/models:/tensorflow/models/slim

USER $USER
WORKDIR $HOME/workspace
```

A.2 Docker-compose file:

```
# Version of docker-compose
version: '3'

# Containers we are going to run
services:
  # Our Phoenix and ix2code container
  yuyu:
    image: ixmatix/server_ix2code_yuyu
    user: root
    stdin_open: true
    tty: true
    volumes:
      - .:/home/yuyu/workspace
    ports:
      # Mapping the port to make the Phoenix app accessible outside of the container
      - "4001:4001"
      - "3000:3000"
    depends_on:
      # The db container needs to be started before we start this container
      - db
    working_dir:
      /home/yuyu/workspace/yuyu_backend
    command:
      python3 "/home/yuyu/workspace/run_yuyu.py"

  db:
```

```
  image: postgres:11
  ports:
  - "5432:5432"
  environment:
  - POSTGRES_USER=postgres
  - POSTGRES_PASSWORD=postgres
  - POSTGRES_DB=yuyu_website_dev

pgadmin:
  container_name: pgadmin4
  image: dpage/pgadmin4
  environment:
      - PGADMIN_DEFAULT_EMAIL=admin@ixmatix.com
      - PGADMIN_DEFAULT_PASSWORD=admin
  ports:
      - "5050:80"
  depends_on:
      - db
```

B

Code Snippets.

B.1 Script for create TF Records

```
from __future__ import division, print_function, absolute_import

import os
import io
import pandas as pd
import tensorflow as tf

from PIL import Image
from object_detection.utils import dataset_util
from collections import namedtuple, OrderedDict

flags = tf.app.flags
flags.DEFINE_string('csv_input', '', 'Path␣to␣the␣CSV␣input')
flags.DEFINE_string('output_path', '', 'Path␣to␣output␣TFRecord')
flags.DEFINE_string('image_dir', '', 'Path␣to␣images')
FLAGS = flags.FLAGS

def class_text_to_int(row_label):
    if row_label == 'footer':
        return 1
    elif row_label == 'footer_bottom':
        return 2
    elif row_label == 'footer_mid':
        return 3
    elif row_label == 'footer_top':
        return 4
    else:
        return None

def split(df, group):
    data = namedtuple('data', ['filename', 'object'])
    gb = df.groupby(group)
    return [
        data(filename, gb.get_group(x))
        for filename, x in zip(gb.groups.keys(), gb.groups)
    ]

def create_tf_example(group, path):
    file_path = os.path.join(path, group.filename)
    with tf.gfile.GFile(file_path, 'rb') as fid:
        encoded_jpg = fid.read()

    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size

    filename = group.filename.encode('utf8')
```

```
    image_format = b'jpg'
    xmins, xmaxs, ymins, ymaxs = [], [], [], []
    classes_text, classes = [], []

    for _, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))

    features = {
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
        'image/filename': dataset_util.bytes_feature(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
        'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
        'image/object/class/label': dataset_util.int64_list_feature(classes),
    }

    tf_example = tf.train.Example(features=tf.train.Features(feature=features))
    return tf_example

def main(_):
    writer = tf.python_io.TFRecordWriter(FLAGS.output_path)
    path = os.path.join(FLAGS.image_dir)
    examples = pd.read_csv(FLAGS.csv_input)
    grouped = split(examples, 'filename')
    for group in grouped:
        tf_example = create_tf_example(group, path)
        writer.write(tf_example.SerializeToString())

    writer.close()
    output_path = os.path.join(os.getcwd(), FLAGS.output_path)
    print('Successfully␣created␣the␣TFRecords:␣{}'.format(output_path))

if __name__ == '__main__':
    tf.app.run()
```

B.2 RFCN_RESNET101_COCO.CONFIG

```
model {
  faster_rcnn {
    num_classes: 90
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
    feature_extractor {
      type: 'faster_rcnn_resnet101'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
        aspect_ratios: [0.5, 1.0, 2.0]
        height_stride: 16
        width_stride: 16
      }
    }
    first_stage_box_predictor_conv_hyperparams {
      op: CONV
      regularizer { l2_regularizer { weight: 0.0 } }
      initializer {
        truncated_normal_initializer { stddev: 0.01 }
      }
    }
    first_stage_nms_score_threshold: 0.0
    first_stage_nms_iou_threshold: 0.7
    first_stage_max_proposals: 300
    first_stage_localization_loss_weight: 2.0
    first_stage_objectness_loss_weight: 1.0
    second_stage_box_predictor {
      rfcn_box_predictor {
        conv_hyperparams {
          op: CONV
```

```
          regularizer { l2_regularizer { weight: 0.0 } }
          initializer {
            truncated_normal_initializer { stddev: 0.01 }
          }
        }
        crop_height: 18
        crop_width: 18
        num_spatial_bins_height: 3
        num_spatial_bins_width: 3
      }
    }
    second_stage_post_processing {
      batch_non_max_suppression {
        score_threshold: 0.0
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 300
      }
      score_converter: SOFTMAX
    }
    second_stage_localization_loss_weight: 2.0
    second_stage_classification_loss_weight: 1.0
  }
}

train_config: {
  batch_size: 1
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        manual_step_learning_rate {
          initial_learning_rate: 0.0003
          schedule { step: 900000 learning_rate: .00003 }
          schedule { step: 1200000 learning_rate: .000003 }
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  gradient_clipping_by_norm: 10.0
  fine_tune_checkpoint: "PATH_TO_BE_CONFIGURED/model.ckpt"
  from_detection_checkpoint: true
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip { }
  }
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED/mscoco_train.record-?????-of-00100"
  }
  label_map_path: "PATH_TO_BE_CONFIGURED/mscoco_label_map.pbtxt"
}

eval_config: {
  num_examples: 8000
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED/mscoco_val.record-?????-of-00010"
  }
  label_map_path: "PATH_TO_BE_CONFIGURED/mscoco_label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

B.3 Object Detection Runner Script

```
import glob
import numpy as np
import tensorflow as tf
from PIL import Image
from matplotlib import pyplot as plt

from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util
from object_detection.utils import ops as utils_ops

# Set TensorFlow logging level
tf.logging.set_verbosity(tf.logging.ERROR)
```

```
# Flags for input parameters
flags = tf.app.flags
flags.DEFINE_string('frozen_file', '', 'Path␣to␣Frozen')
flags.DEFINE_string('label_map', '', 'Path␣to␣labelmap')
flags.DEFINE_string('number_of_classes', '', 'number_of_classes')
flags.DEFINE_string('test_images_path', '', 'Path␣to␣test_images')
FLAGS = flags.FLAGS

# Model preparation variables
PATH_TO_FROZEN_GRAPH = FLAGS.frozen_file
PATH_TO_LABELS = FLAGS.label_map
NUM_CLASSES = int(FLAGS.number_of_classes)  # Convert to integer
PATH_TO_TEST_IMAGES_DIR = FLAGS.test_images_path
IMAGE_SIZE = (50, 50)


def configure_gpu():
    """Configure TensorFlow to use GPU."""
    config = tf.compat.v1.ConfigProto()
    config.gpu_options.allow_growth = True
    return tf.compat.v1.InteractiveSession(config=config)


# Create a session to manage GPUs
session = configure_gpu()


def load_frozen_model(path):
    """Load a frozen TensorFlow model into memory."""
    detection_graph = tf.Graph()
    with detection_graph.as_default():
        graph_def = tf.GraphDef()
        with tf.gfile.GFile(path, 'rb') as fid:
            serialized_graph = fid.read()
            graph_def.ParseFromString(serialized_graph)
            tf.import_graph_def(graph_def, name='')
    return detection_graph


def load_label_map(path):
    """Load the label map."""
    return label_map_util.create_category_index_from_labelmap(
        path, use_display_name=True)


def load_image_into_numpy_array(image):
    """Load an image into a numpy array."""
    (im_width, im_height) = image.size
    return np.array(image.getdata()).reshape(
        (im_height, im_width, 3)).astype(np.uint8)


def get_image_paths(path, extensions=None):
    """Retrieve image paths from directory."""
    if extensions is None:
        extensions = ['png', 'jpg', 'jpeg']
    image_paths = [
        p for ext in extensions
        for p in glob.glob(f"{path}/*.{ext}")
    ]
    return image_paths


def run_inference_for_single_image(image, graph):
    """Run inference on a single image."""
    with graph.as_default():
        with tf.Session(graph=graph) as sess:
            # Get handles to input and output tensors
            tensor_dict = get_tensor_dict_from_graph(graph)
            image_tensor = graph.get_tensor_by_name('image_tensor:0')

            # Run inference
            output_dict = sess.run(
                tensor_dict, feed_dict={image_tensor: np.expand_dims(image, 0)})

            # Extract and process outputs
            return process_output_dict(output_dict)


def get_tensor_dict_from_graph(graph):
    """Get a dictionary of tensors from the graph."""
    ops = graph.get_operations()
    all_tensor_names = {output.name for op in ops for output in op.outputs}
    tensor_dict = {
        key: graph.get_tensor_by_name(key + ':0')
        for key in [
            'num_detections', 'detection_boxes', 'detection_scores',
            'detection_classes', 'detection_masks'
        ]
```

```
        if key + ':0' in all_tensor_names
    }
    if 'detection_masks' in tensor_dict:
        tensor_dict = reframe_detection_masks(tensor_dict, graph)
    return tensor_dict


def process_output_dict(output_dict):
    """Process the TensorFlow output dictionary."""
    output_dict['num_detections'] = int(output_dict['num_detections'][0])
    output_dict['detection_classes'] = output_dict[
        'detection_classes'][0].astype(np.int64)
    output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
    output_dict['detection_scores'] = output_dict['detection_scores'][0]
    if 'detection_masks' in output_dict:
        output_dict['detection_masks'] = output_dict['detection_masks'][0]
    return output_dict


def reframe_detection_masks(tensor_dict, graph):
    """Reframe the detection masks to fit image size."""
    detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [0])
    detection_masks = tf.squeeze(tensor_dict['detection_masks'], [0])
    detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
        detection_masks, detection_boxes, graph
```

B.4 Script for labels validation

```
class GenerateClassText:
    def __init__(self):
        self.tfrecordpath = 'dataSet_generator/generate_tfrecord.py'
        self.labelspath = 'data/generate_labelmap/labels.ix2code'
        self.labels = TagsClass(path=self.labelspath).getTags()

        self._update_record_file()

    def _update_record_file(self):
        with open(self.tfrecordpath, "r") as record_file:
            record_lines = record_file.readlines()

        start_index, end_index = self._get_record_indices(record_lines)
        new_statements = self._generate_tf_record_statements()

        del record_lines[start_index:end_index]
        record_lines.insert(start_index, new_statements)

        with open(self.tfrecordpath, 'w') as f:
            for line in record_lines:
                f.write(line)

    def _get_record_indices(self, record_lines):
        start_index = next(
            (i for i, line in enumerate(record_lines)
             if "class_text_to_int" in line), None) + 1

        end_index = next(
            (i for i, line in enumerate(record_lines)
             if "None" in line), None) + 1

        return start_index, end_index

    def _generate_tf_record_statements(self):
        cleaned_labels = [
            label.strip() for label in self.labels if label.strip()
        ]

        statements = []
        for index, label in enumerate(cleaned_labels):
            if index == 0:
                statements.append(
                    f"\tif␣row_label␣==␣'{label}'␣:␣return␣{index+1}\n"
                )
            else:
                statements.append(
                    f"\telif␣row_label␣==␣'{label}'␣:␣return␣{index+1}\n"
                )

        statements.append("\telse:␣return␣None\n")
        return "\n".join(statements)
```

B.5 Image Validation Script for Dataset

The script validates the sizes of images in a dataset.

```
class ValidateImages:
    VALID_EXTENSIONS = ['png', 'jpg', 'jpeg']

    def __init__(self, path, min_height=200,
                 rejected_folder='data/error_files/rejected_images_size/'):
        self.path = path
        self.min_height = min_height
        self.rejected_folder = rejected_folder
        self.rejected_images = []

        self._prepare_rejected_folder()
        self._validate_images()

    def _prepare_rejected_folder(self):
        """Creates the directory for rejected images if it doesn't exist."""
        if not os.path.exists(self.rejected_folder):
            os.makedirs(self.rejected_folder)

    def _get_images(self):
        """Retrieve all images with valid extensions."""
        images = []
        for ext in self.VALID_EXTENSIONS:
            lower_files = glob.glob(os.path.join(self.path, f"*.{ext}"))
            upper_files = glob.glob(os.path.join(self.path, f"*.{ext.upper()}"))
            images.extend(lower_files)
            images.extend(upper_files)
        return images

    def _move_rejected(self, image_path):
        """Move rejected image and its XML (if exists) to the folder."""
        img_name = os.path.basename(image_path)
        img_ext = os.path.splitext(image_path)[1][1:]
        xml_path = image_path.replace(img_ext, 'xml')

        shutil.move(image_path, self.rejected_folder)
        if os.path.exists(xml_path):
            shutil.move(xml_path, self.rejected_folder)

    def _validate_images(self):
        """Validate image's size, move rejected ones to the folder."""
        for image_path in self._get_images():
            try:
                img = cv2.imread(image_path)
                if img is None:
                    msg = f"Couldn't␣read␣{image_path}.␣Moving␣to␣rejected."
                    print(msg)
                    self._move_rejected(image_path)
                    continue

                h, _, _ = img.shape
                if h < self.min_height:
                    base_name = os.path.basename(image_path)
                    self.rejected_images.append(base_name)
                    self._move_rejected(image_path)
            except Exception as e:
                print(f"Error␣processing␣{image_path}:␣{e}")

        print_msg = (
            f'REJECTED␣IMAGES␣ON␣{self.rejected_folder}␣TOTAL␣REJECTED:␣'
            f'[{len(self.rejected_images)}]'
        )
        print(print_msg)
```

B.6 Validate web elements in XML files

The main goal is to validate VOC PASCAL labeling files by checking each element's minimum area.

```
class Validate:
    def __init__(self, csv_files, folders):
        self.csv_files = csv_files
        self.folders = folders
        self.total_error = 0
        self.total_images = 0
        self.error_names = set()

    def validate_csv(self):
```

```
        for folder_idx, folder in enumerate(self.folders):
            with open(self.csv_files[folder_idx], 'r') as fid:
                file = csv.reader(fid, delimiter=',')
                # Skip the header row
                next(file)

                cnt = 0
                error_cnt = 0
                error = False

                for row in file:
                    if error:
                        error_cnt += 1
                        error = False

                    cnt += 1
                    name, width, height, _, xmin, ymin, xmax, ymax = (
                        row[0], int(row[1]), int(row[2]), row[3], int(row[4]),
                        int(row[5]), int(row[6]), int(row[7])
                    )
                    path = os.path.join(folder, name)
                    img = cv2.imread(path)

                    if img is None:
                        print("\033[1;31m" + 'Could not read image: ', path)
                        error = True
                        continue

                    org_height, org_width = img.shape[:2]

                    checks = [
                        (org_width != width,
                         'Width mismatch for image: {} {} != {}'
                         .format(name, width, org_width)),
                        (org_height != height,
                         'Height mismatch for image: {} {} != {}'
                         .format(name, height, org_height)),
                        (xmin > org_width,
                         f'TAG {row[3]}\n{xmin}XMIN > org_width {org_width} '
                         f'for file {name}'),
                        (xmin <= 0,
                         f'TAG {row[3]}\n{xmin}XMIN < 0 for file {name}'),
                        (xmax > org_width,
                         f'TAG {row[3]}\n{xmax}XMAX > org_width {org_width} '
                         f'for file {name}'),
                        (ymin > org_height,
                         f'TAG {row[3]}\n{ymin}YMIN > org_height {org_height} '
                         f'for file {name}'),
                        (ymin <= 0,
                         f'TAG {row[3]}\n{ymin}YMIN < 0 for file {name}'),
                        (ymax > org_height,
                         f'TAG {row[3]}\n{ymax}YMAX > org_height {org_height} '
                         f'for file {name}'),
                        (xmin >= xmax,
                         f'TAG {row[3]}\n{xmin}xmin >= xmax {xmax} '
                         f'for file {name}'),
                        (ymin >= ymax,
                         f'TAG {row[3]}\n{ymin}ymin >= ymax {ymax} '
                         f'for file {name}')
                    ]

                    for condition, message in checks:
                        if condition:
                            error = True
                            print("\033[1;31m" + message)

                    if error:
                        self.error_names.add(name)
                        print("\033[1;31m" + f'Error for file: {name}\n')

                self.total_error += error_cnt
                self.total_images += cnt

        return self.total_error, self.total_images, self.error_names
```

B.7 Script to transform the website dataset into segmented images

```
class ImageSegmentation:
    ELEMENT_COLOR = {
```

```
        'button': (0, 0, 255),
        'header': (255, 0, 5),
        'footer': (140, 140, 140),
        'nav': (0, 255, 252),
        'form': (0, 138, 255),
        'search': (208, 255, 0),
        'input': (0, 255, 57),
        'img': (213, 0, 255),
        'img_bg': (252, 17, 105),
        'section': (133, 21, 88),
        'text': (30, 244, 200),
        'vmenu': (181, 201, 255),
        'border': (255, 255, 255)
    }

    def __init__(self, coordinates, path):
        self._coordinates = coordinates
        self._path = path
        self._thickness = -1
        self._image = cv2.imread(self._path)
        self._img_h, self._img_w, _ = self._image.shape
        self._segment_image()

    def _segment_image(self):
        black_color = (0, 0, 0)
        cv2.rectangle(self._image, (0, 0),
                      (self._img_w, self._img_h),
                      black_color, self._thickness)

        for element in self._coordinates:
            color = self.ELEMENT_COLOR[element[4]]
            top_left = (element[0], element[1])
            bottom_right = (element[2], element[3])

            if "section" in element[4]:
                cv2.rectangle(self._image, top_left, bottom_right,
                              color, self._thickness)
                border_color = self.ELEMENT_COLOR['border']
                cv2.rectangle(self._image, top_left, bottom_right,
                              border_color, 3)
            else:
                cv2.rectangle(self._image, top_left, bottom_right,
                              color, self._thickness)

    def get_segmented_image(self):
        return self._image

    def get_image_proportions(self):
        return self._img_h, self._img_w
```

B.8 Web Elements Detection Code

```
# Libraries
import cv2
import numpy as np
import os
import tensorflow as tf
from config import temp
from utils import label_map_util, visualization_utils as vis_util
from logs.log import Log

# Configure environment for TensorFlow logs
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
tf.get_logger().setLevel('ERROR')

class WebElementsDetection:
    # Class attributes for logging and TensorFlow session
    _logger = Log("WebElementsDetection")
    _category_index = _sess = _image_tensor = None
    _detection_boxes = _detection_scores = _detection_classes = _num_detections = None
    _image_expanded = None

    def __init__(self, webpage_frame, web_dictionary, model_name, classes, label_file):
        self._num_classes = classes
        self._web_dictionary = web_dictionary
        self._frame = webpage_frame
        cwd_path = os.getcwd()

        # Paths to model and label map files
        model_path = model_name + "/frozen_inference_graph.pb"
        self._ckpt_path = os.path.join(cwd_path, model_path)
        label_path = "labels/" + label_file + ".pbtxt"
        self._labels_path = os.path.join(cwd_path, label_path)

        # Initialize object detection
```

```
        try:
            self.initializeObjectDetection()
            WebElementsDetection._logger.info("WebElementsDetection instance created successfully")
        except Exception as e:
            e_msg = f'Error in __init__: {type(e).__name__}: {e}'
            WebElementsDetection._logger.critical(e_msg)
            raise

    def getObjectCoordinates(self):
        try:
            output_tensors = [self._detection_boxes, self._detection_scores,
                              self._detection_classes, self._num_detections]
            boxes, scores, classes, num = self._sess.run(output_tensors,
                                                         feed_dict={self._image_tensor: self._image_expanded})

            coordinates = vis_util.return_coordinates(
                self._frame, np.squeeze(boxes), np.squeeze(classes).astype(np.int32),
                np.squeeze(scores), self._category_index,
                use_normalized_coordinates=True, line_thickness=8,
                min_score_thresh=0.10, skip_scores=True,
                percentage_labels=self._web_dictionary)

            WebElementsDetection._logger.info("getObjectCoordinates executed successfully")
            return coordinates
        except Exception as e:
            e_msg = f'Error in getObjectCoordinates: {type(e).__name__}: {e}'
            WebElementsDetection._logger.critical(e_msg)
            raise

    def initializeObjectDetection(self):
        try:
            label_map = label_map_util.load_labelmap(self._labels_path)
            categories = label_map_util.convert_label_map_to_categories(
                label_map, max_num_classes=self._num_classes, use_display_name=True)
            self._category_index = label_map_util.create_category_index(categories)

            with tf.Graph().as_default() as detection_graph:
                with tf.gfile.GFile(self._ckpt_path, "rb") as fid:
                    od_graph_def = tf.compat.v1.GraphDef()
                    od_graph_def.ParseFromString(fid.read())
                    tf.import_graph_def(od_graph_def, name="")

                self._sess = tf.compat.v1.Session(graph=detection_graph)

            # Define input and output tensors for the object detection classifier
            tensor_names = ["image_tensor:0", "detection_boxes:0",
                            "detection_scores:0", "detection_classes:0", "num_detections:0"]
            self._image_tensor, self._detection_boxes, self._detection_scores, \
            self._detection_classes, self._num_detections = \
                (detection_graph.get_tensor_by_name(name) for name in tensor_names)

            # Process image for detection
            image_rgb = cv2.cvtColor(self._frame, cv2.COLOR_BGR2RGB)
            self._image_expanded = np.expand_dims(image_rgb, axis=0)
            WebElementsDetection._logger.info("Object detection initialized successfully")
        except Exception as e:
            e_msg = f'Error in initializeObjectDetection: {type(e).__name__}: {e}'
            WebElementsDetection._logger.critical(e_msg)
            raise
```

B.9 Text Analyzer Script

```
import cv2
import io
import numpy as np
import os

from enum import Enum
from google.cloud import vision
from google.cloud.vision import types
from imutils.object_detection import non_max_suppression

from graph_generator.node import *
from logs.log import Log

#export PYTHONIOENCODING=utf-8

# FeatureType enumeration.
class FeatureType(Enum):
    PAGE = 1
    BLOCK = 2
    PARA = 3
    WORD = 4
    SYMBOL = 5
```

```
# Class to analyze the texts.
class TextAnalyzer:
    # Logging
    _logger = None

    _orig = None # Original image
    _frame = None # Image with the rectangles
    _images_inside = {}
    _buttons_dict = {}
    _headers_dict = {}
    _navs_dict = {}

    # Constructor needs all the elements that can contain texts.
    def __init__(
            self, image, images_inside,
            buttons_dict, navs_dict, headers_dict, footer_dict):
        try:
            TextAnalyzer._logger = Log("TextAnalyzer")

            self._texts = {}
            self._buttons_dict = buttons_dict
            self._headers_dict = headers_dict
            self._navs_dict = navs_dict
            self._footer_dict = footer_dict
            self._paragraphsFontSize = []
            self._orig = image
            self._frame = self._orig.copy()
            self._images_inside = images_inside.copy()

            os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = r"ServiceAccountToken.json"
        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣__init__␣-␣{e_type}:␣{e}'

            TextAnalyzer._logger.critical(e_msg)
            raise Exception(e_msg)

        TextAnalyzer._logger.info("SUCCESSFULLY␣CREATED")

    ####################
    # WRAPPER METHODS #

    # Return the text dictionary.
    def getTexts(self):
        return self._texts

    # Return the text dictionary in Nodes.
    def getTextsNodes(self, texts_dict):
        return TextNode.fromDictionary(texts_dict)

    # Return the paragraph font size.
    def getFontSizeParagraph(self):
        bounds = self._getDocumentBounds(FeatureType.WORD) # Is this useful???
        return self._paragraphsFontSize

    # Return the frame.
    def getImage(self):
        return self._frame

    ####################
    # PRIVATE METHODS #

    # Analyze all the texts to see where it corresponds or if it must
    # be ignored.
    def _textAnalyzer(self, bounds, elements_index):
        try:
            # Function definition for use only inside this method.
            #
            # This function is used to evaluate if one text is inside
            # another element in the given dictionary.
            def isTextInside(class_bound, dictionary):
                for key, class_item in dictionary.items():
                    if (class_item.has(class_bound)
                            or class_bound.has(class_item)):
                        return key

            i = 0
            for bound in bounds:
                height = int(bound.vertices[3].y - bound.vertices[0].y)
                width = int(bound.vertices[1].x - bound.vertices[0].x)

                y_max = bound.vertices[3].y
                x_max = bound.vertices[1].x
                y = bound.vertices[0].y
                x = bound.vertices[0].x

                class_bound = Node.fromBounds(x, y, ymax=y_max, xmax=x_max)

                frame_aux = self._orig[y:y_max, x:x_max]
```

```
text = self.textExtractGoogle(frame_aux)
fontSize = self._paragraphsFontSize[i]
i += 1

if text:
    if not isTextInside(class_bound, self._images_inside):
        val_button = isTextInside(class_bound, self._buttons_dict)
        val_nav = isTextInside(class_bound, self._navs_dict)
        val_header = isTextInside(class_bound, self._headers_dict)
        val_footer = isTextInside(class_bound,self._footer_dict)

        tag = "p"

        # Key will be assigned depending on element
        # text type.
        if val_button:
            label = "text_button"
            tag = ""
            # Key will be the button key.
            key = val_button

        elif val_nav:
            tag = "a"
            label = "text_nav"

            # Key will be accord to nav and the items
            # count.
            key = f'txt{val_nav}-{elements_index["text"]}'

        elif val_header:
            label = "text_header"

            # Key will be accord to header and the
            # items count.
            key = f'txt{val_header}-{elements_index["text"]}'

        elif val_footer:
            label = "text_footer"

            # Key will be accord to footer and the
            # items count.
            key = f'txt{val_footer}-{elements_index["text"]}'

        elif len(text) > 45: # The text is a paragraph:
            label = "paragraph"

            # Key will be enumerated text.
            key = f'text{elements_index["text"]}'

        else: # The text is a title
            tag = "h1"
            label = "title"

            # Key will be enumerated text.
            key = f'text{elements_index["text"]}'

        cv2.putText(
            self._frame,
            label,
            (x, y + 30),
            cv2.FONT_HERSHEY_SIMPLEX,
            1,
            (0, 0, 255),
            3
        )

        self._texts[key] = [
            x, y,
            x_max, y_max,
            width, height,
            tag, label,
            text, fontSize
        ]

        elements_index["text"] += 1

    else:
        cv2.putText(
            self._frame,
            "IGNORE",
            (x, y + 30),
            cv2.FONT_HERSHEY_SIMPLEX,
            1,
            (0, 0, 0),
            3
        )
else:
    cv2.putText(
        self._frame,
```

```
                    "IGNORE",
                    (x, y + 30),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    1,
                    (0, 0, 0),
                    3
                )

    except Exception as e:
        e_type = str(type(e))[8:-2]
        e_msg = f'IN␣_textAnalyzer␣-␣{e_type}:␣{e}'

        TextAnalyzer._logger.critical(e_msg)
        raise Exception(e_msg)

    TextAnalyzer._logger.info("TEXT␣ANALYZED␣SUCCESSFULLY")

##################
# PUBLIC METHODS #

# Erase everything drawn in frame.
def cleanFrame(self, frame_aux):
    self._frame = frame_aux.copy()

    TextAnalyzer._logger.info("IFRAME␣CLEANED")

# Google text extraction.
def textExtractGoogle(self, img):
    #export PYTHONIOENCODING=utf-8
    client = vision.ImageAnnotatorClient()
    a, b, c = img.shape
    try:
        assert a > 0 and b > 0 and c > 0, "IMAGE␣HAS␣VOID␣DIMENSION"
    except AssertionError as e:
        TextAnalyzer._logger.error(f'{e}␣:␣{img.shape}')
        return None

    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    _, encoded_image = cv2.imencode(".jpg", img)

    content = encoded_image.tobytes()
    image = vision.types.Image(content=content)

    response = client.text_detection(image=image)
    texts = response.text_annotations

    if texts:
        return texts[0].description

    TextAnalyzer._logger.info("textExtractGoogle␣successfully␣executed")

# Begin of renderDocText. #
# Get the bounds of paragaphs, draw them and send it to textAnalyzer.
def renderDocText(self, elements_index):
    try:
        #bounds = self.getDocumentBounds(FeatureType.BLOCK)
        #self.drawBoxes(bounds, (255,0,0))

        bounds = self._getDocumentBounds(FeatureType.PARA)
        self._drawBoxes(bounds, (0, 0, 255))
        self._textAnalyzer(bounds, elements_index)

        #bounds = self.getDocumentBounds(FeatureType.WORD)
        #self.drawBoxes(bounds, (0,255,0))
    except Exception as e:
        e_type = str(type(e))[8:-2]
        e_msg = f'IN␣renderDocText␣-␣{e_type}:␣{e}'

        TextAnalyzer._logger.critical(e_msg)
        raise Exception(e_msg)

    TextAnalyzer._logger.info("renderDocText␣successfully␣executed")

# Draw boxes into frame.
def _drawBoxes(self, bounds, color):
    try:
        for bound in bounds:
            height = int(bound.vertices[3].y - bound.vertices[0].y)
            width = int(bound.vertices[1].x - bound.vertices[0].x)

            cv2.rectangle(
                self._frame,
                (bound.vertices[0].x, bound.vertices[0].y),
                (bound.vertices[0].x + width, bound.vertices[0].y + height),
                color, 2
            )
    except Exception as e:
        e_msg = f'IN␣_drawBoxes␣METHOD'
        TextAnalyzer._logger.critical(e_msg)
```

```
            raise Exception(e_msg)

        TextAnalyzer._logger.info("_drawBoxes␣successfully␣executed")

    # Return a list of the bounding boxes.
    def _getDocumentBounds(self, feature):
        try:
            #export PYTHONIOENCODING=utf-8
            client = vision.ImageAnnotatorClient()
            img = cv2.cvtColor(self._orig, cv2.COLOR_BGR2RGB)

            _, encoded_image = cv2.imencode(".jpg", img)
            content = encoded_image.tobytes()

            bounds = []
            image = types.Image(content=content)

            response = client.document_text_detection(image=image)
            document = response.full_text_annotation

            # Collect specified feature bounds by enumerating all
            # document features.
            for page in document.pages:
                for block in page.blocks:
                    for paragraph in block.paragraphs:
                        count = 0
                        sum_fontSize = 0

                        for word in paragraph.words:
                            for symbol in word.symbols:

                                if (feature == FeatureType.SYMBOL):
                                    bounds.append(symbol.bounding_box)

                            if (feature == FeatureType.WORD):
                                bounds.append(word.bounding_box)

                            if (feature == FeatureType.PARA):
                                wbb_vertices = word.bounding_box.vertices
                                h_aux = [vertex.y for vertex in wbb_vertices]

                                h = h_aux[2] - h_aux[1]
                                sum_fontSize += h
                                count += 1

                        if (feature == FeatureType.PARA):
                            bounds.append(paragraph.bounding_box)
                            self._paragraphsFontSize.append(
                                int(sum_fontSize / count)
                            )

                    if (feature == FeatureType.BLOCK):
                        bounds.append(block.bounding_box)
        except Exception as e:
            e_msg = f'␣IN␣_getDocumentBounds␣METHOD'
            TextAnalyzer._logger.critical(e_msg)
            raise Exception(e_msg)

        TextAnalyzer._logger.info("_getDocumentBounds␣successfully␣executed")

        # The list `bounds` contains the coordinates of the bounding
        # boxes.
        return bounds
    # End of renderDocText. #
```

B.10 Title Analyzer Script

```
import numpy as np
import cv2
from sklearn.cluster import KMeans

from graph_generator.node import Node
from logs.log import Log

TAG = 0
HEIGHT = 1
YMIN = 2
GROUP = 3

# Class to analyze the titles.
class TitleAnalyzer():
    # Logging
    _logger = None

    # Constructor just needs the texts dictionary.
```

```
def __init__(self, frame, texts_dict):
    try:
        TitleAnalyzer._logger = Log("TitleAnalyzer")
        self._frame = frame.copy()
        self._texts_dict = texts_dict
        self._titles_dict = {}
        self._groups = []
        self._titleAssignation = {}

    except Exception as e:
        e_type = str(type(e))[8:-2]
        e_msg = f'IN␣__init__␣-␣{e_type}:␣{e}'

        TitleAnalyzer._logger.critical(e_msg)
        raise Exception(e_msg)

    TitleAnalyzer._logger.info("SUCCESSFULLY␣CREATED")

###################
# WRAPPER METHODS #

def getImage(self):
    return self._frame

##################
# PUBLIC METHODS #

# Begin of titleAssignments. #
def titleAssignments(self):
    try:
        self._getTitleElements()
        self._groups = self._groupTitles()

        if len(self._groups) <= 6:
            self._groups2TitleLabels()
        else:
            self._groupsKmeansImplementation()

        self._extraH1toH2()

        for key_t, class_title in self._texts_dict.copy().items():
            if 'h' in class_title.tag:
                title_removed = False

                for key_p, class_p in self._texts_dict.copy().items():
                    if 'p' in class_p.tag:
                        if class_p.percentageInside(class_title) > 25:
                            del self._texts_dict[class_title.key]
                            title_removed = True

                if title_removed:
                    continue

                new_tag = self._titles_dict[key_t][TAG]
                self._texts_dict[class_title.key].tag = new_tag

                # Identify title.
                cv2.rectangle(
                    self._frame,
                    (class_title.xmin, class_title.ymin),
                    (class_title.xmax, class_title.ymax),
                    (255, 0, 255),
                    4
                )

                cv2.putText(
                    self._frame,
                    new_tag,
                    (class_title.xmin, class_title.ymin + 30),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    1,
                    (255, 0, 255),
                    3
                )

    except Exception as e:
        e_type = str(type(e))[8:-2]
        e_msg = f'IN␣titleAssignments␣-␣{e_type}:␣{e}'

        TitleAnalyzer._logger.critical(e_msg)
        raise Exception(e_msg)

    TitleAnalyzer._logger.info("␣titleAssignments␣successfully␣executed")

# Method to convert h1 titles into h2.
def _extraH1toH2(self):
    try:
        h1_titles = list()
```

```
            for key, title in self._titles_dict.items():
                if title[TAG] == "h1":
                    h1_titles.append((key, title[YMIN]))

            h1_titles.sort(key=lambda t: t[1])
            first_h1 = True
            for key, _ in h1_titles:
                if not first_h1:
                    self._titles_dict[key][TAG] = "h2"

                first_h1 = False

        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣_extraH1toH2␣-␣{e_type}:␣{e}'

            TitleAnalyzer._logger.critical(e_msg)
            raise Exception(e_msg)

        TitleAnalyzer._logger.info(f'_extraH1toH2␣successfully␣executed')

    # Get all the title text elements.
    def _getTitleElements(self):
        try:
            for key, class_text in self._texts_dict.items():
                if 'h' in class_text.tag:
                    group = -1 # Initialize the title without group
                    self._titles_dict[key] = [
                        class_text.tag,
                        class_text.height,
                        class_text.ymin,
                        group
                    ]

        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣_getTitleElements␣-␣{e_type}:␣{e}'

            TitleAnalyzer._logger.critical(e_msg)
            raise Exception(e_msg)

        TitleAnalyzer._logger.info(f'GETTING␣TITLE␣ELEMENTS␣SUCCESFULLY')

    def _groups2TitleLabels(self):
        try:
            for _, title_value in self._titles_dict.items():
                group = title_value[GROUP]
                # Update title label (h1 - h6)
                title_value[TAG] = self._titleAssignation[group]

        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣_groups2TitleLabels␣-␣{e_type}:␣{e}'

            TitleAnalyzer._logger.critical(e_msg)
            raise Exception(e_msg)

        TitleAnalyzer._logger.info("_groups2TitleLabels␣successfully␣executed")

    def _groupsKmeansImplementation(self):
        try:
            groups_sorted = sorted(self._groups, reverse=True)
            X = np.array(groups_sorted).reshape((-1, 1))

            kmeans = KMeans(n_clusters=6)
            kmeans.fit(X)
            clusters = kmeans.labels_

            group_tags = [None, None, None, None, None, None]

            i = 1
            for cluster in clusters:
                if not group_tags[cluster]:
                    group_tags[cluster] = f'h{i}'
                    i += 1

            for i, element in enumerate(groups_sorted):
                index = self._groups.index(element)
                self._titleAssignation[index] = group_tags[clusters[i]]

            self._groups2TitleLabels()

        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣_groupsKmeansImplementation␣-␣{e_type}:␣{e}'

            TitleAnalyzer._logger.critical(e_msg)
            raise Exception(e_msg)
```

```
        TitleAnalyzer._logger.info("␣_groupsKmeansImplementation␣successfully␣executed")

    def _groupTitles(self):
        try:
            groups = []

            for _, title_value in self._titles_dict.items():
                belongs_to_group = False
                height_title = title_value[HEIGHT]

                for index, group in enumerate(groups):

                    if (group-15) <= height_title <= (group+15):
                        belongs_to_group = True
                        title_value[GROUP] = index
                        break

                if not belongs_to_group:
                    title_value[GROUP] = len(groups)
                    groups.append(height_title)

            groups_sorted = sorted(groups.copy(), reverse=True)
            for i, group in enumerate(groups_sorted):
                self._titleAssignation[groups.index(group)] = "h" + str(i + 1)

        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣_groupTitles␣-␣{e_type}:␣{e}'

            TitleAnalyzer._logger.critical(e_msg)
            raise Exception(e_msg)

        TitleAnalyzer._logger.info("_gropuTtitles␣successfully␣executed")
        return groups
    # End of titleAssignments. #
```

B.11 BEM Genrator Code

```
from logs.log import Log


class Styles:

    def __init__(self, key, prefix="", attributes=None):
        self._key = key
        self._prefix = prefix

        if attributes:
            self._attributes = attributes
        else:
            self._attributes = dict()

    #################
    # MAGIC METHODS #

    def __str__(self):
        if not self._key:
            raise Exception(f'KEY␣ERROR:␣{self._prefix}{self._key}')

        css = self._prefix + self._key + "␣{\n"

        for name in sorted(self._attributes.keys()):
            if "tag" in name:
                continue

            value = self._attributes[name]
            css += f'␣␣␣␣{name}:␣{value};\n'

        css += "}\n"

        return css

    ##################
    # PUBLIC METHODS #

    def appendAttribute(self, name, value):
        self._attributes[name] = value

    def appendMultipleAttributes(self, attributes):
        for name, value in attributes.items():
            self._attributes[name] = value

    def getAttribute(self, name):
        return self._attributes[name]
```

```
    def getAttributes(self):
        return self._attributes.items()

    def hasAttribute(self, attribute):
        return attribute in self._attributes

    def updatePrefix(self, prefix):
        self._prefix = prefix

    def combine(self, styles):
        if not styles:
            return

        for name, value in styles.items():
            if name not in self._attributes:
                self._attributes[name] = value

    def commonStyles(self, styles, new_key, prefix="."):
        common_styles = Styles(new_key, prefix)

        for name, value in styles.getAttributes():
            if (name in self._attributes
                    and value == self._attributes[name]):
                common_styles.appendAttribute(name, value)

        return common_styles

    def extraIndent(self):
        if not self._key:
            raise Exception(f'KEY␣ERROR:␣{self._prefix}{self._key}')

        css = f'␣␣␣␣{self._prefix}{self._key}' + "␣{\n"

        for name in sorted(self._attributes.keys()):
            if "tag" in name:
                continue

            value = self._attributes[name]
            css += f'␣␣␣␣␣␣␣␣{name}:␣{value};\n'

        css += "␣␣␣␣}\n"

        return css


class Modifier:

    def __init__(self, key, element, prefix=".", styles=None):
        self.css_name = f'{element.block.key}__{element.key}__{key}'
        self.element = element
        self.styles = Styles(
            self.css_name,
            prefix,
            styles
        )

    ##################
    # MAGIC METHODS #

    def __str__(self):
        if not self.styles.getAttributes():
            return ""

        return f'\n{self.styles}'

    ###################
    # PUBLIC METHODS #

    def combineStyles(self, styles):
        if not styles:
            return

        self.styles.combine(styles)

    def extraIndent(self):
        if not self.styles.getAttributes():
            return ""

        return f'\n{self.styles.extraIndent()}'


class Element:

    def __init__(self, key, block, prefix=".", styles=None):
        self.block = block
        self.css_name = f'{block.key}__{key}'
        self.key = key
        self.styles = Styles(
```

```
            self.css_name,
            prefix,
            styles
        )

        self.modifiers = dict()

    #################
    # MAGIC METHODS #

    def __str__(self):
        if self.styles.getAttributes():
            element_text = f'{self.styles}'
        else:
            element_text = ""

        for key in sorted(self.modifiers.keys()):
            element_text += f'{self.modifiers[key]}'

        return "\n" + element_text

    ##################
    # PUBLIC METHODS #

    def combineStyles(self, styles):
        if not styles:
            return

        self.styles.combine(styles)

    def createModifier(self, key, prefix=".", styles=None):
        if key in self.modifiers:
            self.modifiers[key].combineStyles(styles)
            self.modifiers[key].styles.updatePrefix(prefix)
        else:
            self.modifiers[key] = Modifier(key, self, prefix, styles)

    def extraIndent(self):
        if self.styles.getAttributes():
            element_text = f'{self.styles.extraIndent()}'
        else:
            element_text = ""

        for key in sorted(self.modifiers.keys()):
            element_text += f'{self.modifiers[key].extraIndent()}'

        return "\n" + element_text

    def generalize(self):
        common_styles = None

        for _, modifier in self.modifiers.items():
            if not common_styles:
                common_styles = modifier.styles
            else:
                common_styles = common_styles.commonStyles(
                    modifier.styles,
                    self.key
                )

        self.styles.combine(common_styles)


class Block:

    def __init__(self, key, prefix=".", styles=None):
        if ("header" in key
                or "div" in key):
            prefix = "#"

        self.key = key
        self.styles = Styles(
            key,
            prefix,
            styles
        )

        self.elements = dict()

    #################
    # MAGIC METHODS #

    def __str__(self):
        if self.styles.getAttributes():
            block_text = f'{self.styles}'
        else:
            block_text = ""

        for key in sorted(self.elements.keys()):
```

```
            block_text += f'/* Styles for element {key}. */\n'
            block_text += f'{self.elements[key]}'

        return block_text + "\n\n"

    ##################
    # PUBLIC METHODS #

    def combineStyles(self, styles):
        if not styles:
            return

        self.styles.combine(styles)

    def createElement(self, key, prefix=".", styles=None):
        if key in self.elements:
            self.elements[key].combineStyles(styles)
            self.elements[key].styles.updatePrefix(prefix)
        else:
            self.elements[key] = Element(key, self, prefix, styles)

    def extraIndent(self):
        if self.styles.getAttributes():
            block_text = f'{self.styles.extraIndent()}'
        else:
            block_text = ""

        for key in sorted(self.elements.keys()):
            block_text += f'{self.elements[key].extraIndent()}'

        return block_text + "\n\n"

    def generalizeElements(self):
        for _, element in self.elements.items():
            element.generalize()


class Block:

    def __init__(self, key, prefix=".", styles=None):
        if ("header" in key
                or "div" in key):
            prefix = "#"

        self.key = key
        self.styles = Styles(
            key,
            prefix,
            styles
        )

        self.elements = dict()

    ##################
    # MAGIC METHODS #

    def __str__(self):
        if self.styles.getAttributes():
            block_text = f'{self.styles}'
        else:
            block_text = ""

        for key in sorted(self.elements.keys()):
            block_text += f'{self.elements[key]}'

        return block_text + "\n\n"

    ##################
    # PUBLIC METHODS #

    def combineStyles(self, styles):
        if not styles:
            return

        self.styles.combine(styles)

    def createElement(self, key, prefix=".", styles=None):
        if key in self.elements:
            self.elements[key].combineStyles(styles)
            self.elements[key].styles.updatePrefix(prefix)
        else:
            self.elements[key] = Element(key, self, prefix, styles)

    def extraIndent(self):
        if self.styles.getAttributes():
            block_text = f'{self.styles.extraIndent()}'
        else:
            block_text = ""
```

```
        for key in sorted(self.elements.keys()):
            block_text += f'{self.elements[key].extraIndent()}'

        return block_text + "\n\n"

    def generalizeElements(self):
        for _, element in self.elements.items():
            element.generalize()


class BEMGenerator:
    # Logging
    _logger = None

    # Static attributes.
    _media = None
    _blocks = None

    def __init__(self, result_path, webpage="index"):
        try:
            BEMGenerator._logger = Log("BEMGenerator")
            self._medias = ["480px", "800px", "1200px"]

            self._result_path = result_path
            self._webpage = webpage
        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣__init__␣-␣{e_type}:␣{e}'

            BEMGenerator._logger.critical(e_msg)
            raise Exception(e_msg)

    ##################
    # PUBLIC METHODS #

    def build(self, save=True):
        try:
            css_text = "/*******************************/\n"
            css_text += "/*␣Beginning␣of␣common␣styles.␣*/\n"
            css_text += "/*******************************/\n\n"

            for key in sorted(BEMGenerator._blocks.keys()):
                block = BEMGenerator._blocks[key]
                css_text += f'/*␣Beginning␣of␣styles␣for␣block␣{key}.␣*/\n'
                css_text += f'{block}'

            css_text += "/*******************************/\n"
            css_text += "/*␣Beginning␣of␣media␣styles.␣*/\n"
            css_text += "/*******************************/\n\n"
            for media in self._medias:
                css_text += f'/*␣Beginning␣of␣styles␣for␣{media}␣width.␣*/\n'
                css_text += f'@media(max-width:␣{media})' + '␣{\n'

                for key in sorted(BEMGenerator._media[media].keys()):
                    css_text += f'␣␣␣␣{key}' + '␣{\n'

                    for key_style, style in BEMGenerator._media[media][key].items():
                        css_text += f'␣␣␣␣␣␣␣␣{key_style}:␣{style};\n'

                    css_text += '␣␣␣␣}\n'

                css_text += '}\n\n'

            if save:
                # Default name is index.css
                f = open(
                    f'{self._result_path}styles_{self._webpage}.css',
                    "wb"
                )
                f.write(css_text.encode())
                f.close()

        except Exception as e:
            e_msg = f'IN␣METHOD␣build:␣{e}'

            BEMGenerator._logger.critical(e_msg)
            raise Exception(e_msg)

        BEMGenerator._logger.info("CSS␣CODE␣CREATED")

    def createBlock(self, key, prefix=".", styles=None):
        if ("header" in key
                or "div" in key):
            prefix = "#"

        if key in BEMGenerator._blocks:
            BEMGenerator._blocks[key].combineStyles(styles)
            BEMGenerator._blocks[key].styles.updatePrefix(prefix)
        else:
```

```
            BEMGenerator._blocks[key] = Block(key, prefix, styles)

    def createElement(self,
            block_key,
            key, prefix=".", styles=None):
        try:
            block = BEMGenerator._blocks[block_key]
        except:
            self.createBlock(block_key, prefix="#")
            block = BEMGenerator._blocks[block_key]

        block.createElement(key, prefix, styles)

    def createMedia(
            self,
            media_key, block_key,
            element=None, modifier=None,
            prefix=".", styles=None):
        if element:
            if modifier:
                key = f'{prefix}{block_key}__{element}__{modifier}'
            else:
                key = f'{prefix}{block_key}__{element}'
        else:
            key = f'{prefix}{block_key}'

        BEMGenerator._media[media_key][key] = styles

    def createModifier(
            self,
            block_key, element_key,
            key, prefix=".", styles=None):

        try:
            block = BEMGenerator._blocks[block_key]
        except:
            self.createBlock(block_key)
            block = BEMGenerator._blocks[block_key]

        try:
            element = block.elements[element_key]
        except:
            self.createElement(block_key, element_key)
            element = block.elements[element_key]

        element.createModifier(key, prefix, styles)

    def generalizeElements(self):
        for _, block in BEMGenerator._blocks.items():
            block.generalizeElements()

    def initStatics(self):
        BEMGenerator._media = {
            "480px": {},
            "800px": {},
            "1200px": {}
        }

        BEMGenerator._blocks = {}
```

B.12 CSS Analyzer Code

```
from graph_generator.node import *
from logs.log import Log

class CSSAnalyzer():

    _logger = None

    def __init__(self):
        try:
            CSSAnalyzer._logger = Log("CSSAnalyzer")

        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣__init__␣-␣{e_type}:␣{e}'

            CSSAnalyzer._logger.critical(e_msg)
            raise Exception(e_msg)

        CSSAnalyzer._logger.info("SUCESSFULLY␣CREATED")

    ##################
    # STATIC METHODS #
```

```
@staticmethod
def getAlignItems(x_max_img, elements_inside, x_min_container=0):
    try:
        if not elements_inside:
            return "center"

        element = max(elements_inside, key=lambda x: x.xmin)

        if element.xmin < (x_min_container + 40):
                return "flex-start"
        else:
            if (x_max_img - element.xmax) < 30:
                return "flex-end"
            else:
                return "center"

    except Exception as e:
        e_type = str(type(e))[8:-2]
        e_msg = f'IN␣getAlignItems␣-␣{e_type}:␣{e}'

        CSSAnalyzer._logger.critical(e_msg)
        raise Exception(e_msg)

@staticmethod
def getElementMarginsForAlignItems(elements):
    try:
        element_features = {}
        elements = sorted(elements, key=lambda y: y.ymin)
        for i, element in enumerate(elements):
            ymax= element.ymax

            if i < (len(elements) - 1):
                ymin = elements[i+1].ymin

                margin_top =  abs(ymin-ymax)

                element_features[elements[i+1].key] = {
                    "tag": elements[i+1].tag,
                    "margin-top": f'{margin_top}px'
                }


    except Exception as e:
        e_msg = F'IN␣getElementMarginsForAlignItems␣METHOD'
        CSSAnalyzer._logger.critical(e_msg)
        raise Exception(e_msg)

    CSSAnalyzer._logger.info("␣getElementMarginsForAlignItems␣successfully␣executed")
    return element_features

@staticmethod
def getElementMarginsForJustifyContent(
        elements,
        x_max_container,
        justify_content,
        x_min_container=0):
    try:
        element_features = {}
        nearest_container = x_min_container
        center_nearest_container = elements[-1].xmin
        mx_nearest_container = x_max_container

        for i, element in enumerate(elements):

            if justify_content == "flex-start":
                """
                Flex-start:
                All header elements will be a margin-left
                """
                margin = element.xmin - nearest_container
                if margin < -50:
                    margin = 0

                element_features[element.key] = {
                    "tag": element.tag,
                    "margin-left": f'{margin}px'
                }
                nearest_container = element.xmax

            elif justify_content == "flex-end":
                """
                Flex-end:
                All header elements will be a margin-right
                """
                element = elements[-(i+1)]
                margin = abs(mx_nearest_container - element.xmax)
                element_features[element.key] = {
                    "tag": element.tag,
                    "margin-right": f'{margin}px'
```

```
                }
                mx_nearest_container = element.xmin

            elif justify_content == "center":
                """
                Center:
                All elements except the last one will have a margin-right
                """
                if  i > 0:
                    element = elements[-(i+1)]
                    margin = abs(center_nearest_container - element.xmax)
                    element_features[element.key] = {
                        "tag": element.tag,
                        "margin-right": f'{margin}px'
                    }
                    center_nearest_container = element.xmin

            else:
                break

        if justify_content == "space-between":

            # Space-between
            # The first element will have a margin-left
            # The last element will have a margin-right

            margin_left = abs(elements[0].xmin - x_min_container)
            margin_right = abs(x_max_container - elements[-1].xmax)

            element_features[elements[0].key] = {
                "tag": elements[0].tag,
                "margin-left": f'{margin_left}px'
            }
            element_features[elements[-1].key] = {
                "tag": elements[-1].tag,
                "margin-right": f'{margin_right}px'
            }
    except Exception as e:
        e_type = str(type(e))[8:-2]
        e_msg = f'IN␣getElementMargins␣-␣{e_type}:␣{e}'

        CSSAnalyzer._logger.critical(e_msg)
        raise Exception(e_msg)

    CSSAnalyzer._logger.info("getElementMarginsForJustifyContent␣successfully␣executed")

    return element_features

"""
It's necessary determine the justify-content of the header,
could be [flex-start, flex-end, center, space-between, space-around].
[Image link for reference]
https://cdn.discordapp.com/attachments/858054490762379324/873253503081009252/justify-content.png

1. The element closest to the left is checked if it's close to the left margin, then our options are:
A. flex-start
B. space-between

2. If condition 1 is true, then the last element with its margin to the right is
   checked, if it is close it must be used:
B. space-between

Else, it would be:
A. flex-start

3. If condition 1 is false, then our options can be:
C. flex-end
D. center
E. space-around

4. We evaluate the right margin of the last element and if it's close the answer would be:
C. flex-end

If it's not close it would be between:
D. center
E. space-around

If we have more than one element, and if element 1 that is next to it has a distance greater than X, then it is:
E. space-around

Else, it's:
D. center
"""
@staticmethod
def getJustifyContent(
        x_max_container,
        elements_inside,
        x_min_container=0):
    try:
```

```
            first_element = elements_inside[0].xmin
            last_element = elements_inside[-1].xmax

            if first_element < (x_min_container + 40):
                if (x_max_container - last_element) < 40:
                    return "space-between"
                else:
                    return "flex-start"
            else:
                if (x_max_container - last_element) < 40:
                    return "flex-end"
                else:
                    if len(elements_inside) > 1:
                        second_element = elements_inside[1].xmin
                        first_element_max = elements_inside[0].xmax
                        if second_element - first_element_max < 60:
                            return "center"
                        else:
                            return "space-around"
                    else:
                        return "center"

        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣getJustifyContentHeader␣-␣{e_type}:␣{e}'

            CSSAnalyzer._logger.critical(e_msg)
            raise Exception(e_msg)

    @staticmethod
    def getTextAlign(align_items):
        if align_items == "flex-start":
            return "left"
        elif align_items == "flex-end":
            return "right"
        else:
            return "center"
```

B.13 CSS Features Extractor Code

```
import cv2
import numpy as np
import math
import os, io

from sklearn.cluster import KMeans

from analyzers.divAnalyzer import DivAnalyzer
from analyzers.footerAnalyzer import FooterAnalyzer
from analyzers.sectionAnalyzer import SectionAnalyzer
from analyzers.headerAnalyzer import HeaderAnalyzer
from analyzers.cssAnalyzer import CSSAnalyzer
from analyzers.navAnalyzer import NavAnalyzer
from analyzers.buttonAnalyzer import ButtonAnalyzer
from web_generator.cssHelperExtractor import CSSHelperExtractor

from graph_generator.node import *
from logs.log import Log

# CSSFeaturesExtractor class to get CSS attributes of all elements.
class CSSFeaturesExtractor:
    # Increase font-size. The goal of this constant is to increment
    # the fontsizes of the texts but with the next formula:
    # fontsize += INCREASE_FONT_SIZE / fontsize
    # With this increment we ensure that bigger texts doesn't grow
    # too much.
    INCREASE_FONT_SIZE = 50

    # Reduce image-size (the goal of this constant is to avoid that several
    # images are too big in processing, to the idea it's reduce the size in pixels)
    REDUCE_IMAGE_SIZE = 50
    # Logging
    _logger = None

    # Constructor needs all the dictionaries to get all the
    # features.
    def __init__(self, image, web_componets):
        try:
            CSSFeaturesExtractor._logger = Log("CSSFeaturesExtractor")
            CSSHelperExtractor()
            self._image = image
            self._web_components = web_componets
        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣__init__␣-␣{e_type}:␣{e}'
```

```
        CSSFeaturesExtractor._logger.critical(e_msg)
        raise Exception(e_msg)

    CSSFeaturesExtractor._logger.info("SUCCESSFULLY␣CREATED")

##################
# PUBLIC METHODS #

# Get the body attributes.
def getBodySpecs(self, bem_gen, css_rules):
    try:
        styles = dict()

        if css_rules:
            value = css_rules.replace("font-family:␣", "")
            value = value.replace(";", "")
            styles["font-family"] = value

        bem_gen.createBlock(
            "body",
            prefix="",
            styles=styles
        )

        bem_gen.createBlock(
            "button",
            prefix="",
            styles={
                "display": "block"
            }
        )

    except Exception as e:
        e_type = str(type(e))[8:-2]
        e_msg = f'IN␣getBodySpecs␣-␣{e_type}:␣{e}'

        CSSFeaturesExtractor._logger.critical(e_msg)
        raise Exception(e_msg)

    CSSFeaturesExtractor._logger.info("getBodySpecs␣successfully␣executed")

# Get the button attributes and insert them in css_gen to build
# it later in the CSS file.
def getButtonSpecs(self, bem_gen, xml_gen):
    try:
        # Analyze each button in the dictionary.
        for key, class_button in self._web_components["buttons"].items():

            frame = self._image.copy()
            xmin, ymin, xmax, ymax = class_button.getCoords()
            frame = frame[ymin:ymax, xmin:xmax]

            button_cropped = ButtonAnalyzer.buttonCropped(frame)

            # Get the button style.
            if "rounded" in key:
                border_radius = "1000px"
            else:
                border_radius = "0px"

            # If contains text then analyze it too.
            textExists = False
            if key in self._web_components["texts"].keys():
                class_text = self._web_components["texts"][key]
                textExists = True

                frame_text = self._image.copy()
                xmin, ymin, xmax, ymax = class_text.getCoords()
                frame_text = frame_text[ymin:ymax, xmin:xmax]

                top_padding = class_button.ymin \
                              - class_text.ymin
                left_padding = class_button.xmin \
                               - class_text.xmin
                padding_value = f'{top_padding}px␣{left_padding}px'

            #border_color = ButtonAnalyzer.getBorderColor(button_cropped)

            graph_button = xml_gen.getElementByClassName("button", key)
            if not graph_button:
                continue
            block_key = graph_button.getAttribute("css_pattern")

            styles = dict()

            element_color = CSSHelperExtractor.getElementColor(
                button_cropped
            )
```

```
            styles["background-color"] = f'rgb{element_color}'

            #if border_color:
            #    styles["border-color"] = f'rgb{border_color}'

            if textExists:
                element_color = CSSHelperExtractor.getElementColor(
                    frame_text,
                    text_color=True
                )

                fontsize = class_text.fontsize
                fontsize += int(self.INCREASE_FONT_SIZE / fontsize)
                styles["color"] = f'rgb{element_color}'
                styles["font-size"] = f'{fontsize}px'
                styles["padding"] = padding_value

            styles["border-radius"] = border_radius
            styles["height"] = f'{class_button.height}px'
            styles["width"] = f'{class_button.width}px'

            bem_gen.createModifier(
                block_key,
                "button",
                key,
                styles=styles
            )

        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣getButtonSpecs␣-␣{e_type}:␣{e}'

            CSSFeaturesExtractor._logger.critical(e_msg)
            raise Exception(e_msg)

        CSSFeaturesExtractor._logger.info("getButtonSpecs␣successfully␣executed")

    # Styles for each HTML sections (body, header, section, footer).
    # Implement the default styles of website.
    def getDefaultSpecs(self, bem_gen):
        try:
            bem_gen.createBlock(
                "*",
                prefix="",
                styles={
                    "margin": 0,
                    "padding": 0,
                    "box-sizing": "border-box",
                    "text-decoration": "none",
                    "list-style": "none"
                }
            )
        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣getDefaultSpecs␣-␣{e_type}:␣{e}'

            CSSFeaturesExtractor._logger.critical(e_msg)
            raise Exception(e_msg)

        CSSFeaturesExtractor._logger.info("getDefaultSpecs␣successfully␣executed")

    # Get CSS divs's styles
    def getDivsSpecs(self, bem_gen, xml_gen):
        try:
            def insertDivInMedia(div_key, div_width):
                if div_width > 800:
                    bem_gen.createMedia(
                        "1200px",
                        div_key,
                        prefix="#",
                        styles={
                            "width": "100%",
                        }
                    )
                if div_width > 480 and div_width < 800:
                    bem_gen.createMedia(
                        "800px",
                        div_key,
                        prefix="#",
                        styles={
                            "width": "100%",
                        }
                    )
                if div_width < 480:
                    bem_gen.createMedia(
                        "480px",
                        div_key,
                        prefix="#",
                        styles={
```

```
                "width": "100%",
            }
        )

# Begin of getDivsSpecs.
for graph_div in xml_gen.aux_xml.getElementsByTagName("div"):
    div_key = graph_div.getAttribute("id")
    if "header" in div_key and "image" in div_key:
        bem_gen.createMedia(
            "1200px",
            div_key,
            prefix="#",
            styles={
                "margin-left": "15px",
            }
        )
    else:
        try:
            class_div = self._web_components["sections_footers_divs"][div_key]
        except:
            continue

    if graph_div.hasAttribute("row"):
        """
        Get background color current div
        """
        div_frame = self._image.copy()
        xmin, ymin, xmax, ymax = class_div.getCoords()
        div_frame = div_frame[ymin:ymax, xmin:xmax]

        img_filter_color_frame = CSSHelperExtractor.imageFilterColor(
            class_div,
            div_frame,
            [
                self._web_components["texts"],
                self._web_components["buttons"],
                self._web_components["images"]
            ]
        )

        element_color = CSSHelperExtractor.getElementColor(
            img_filter_color_frame
        )
        if CSSHelperExtractor.isBlack(element_color):
            try:
                element_color = CSSHelperExtractor.getElementColor(
                    img_filter_color_frame,
                    text_color=True
                )
            except:
                element_color = (0, 0, 0)

        """
        Get background color parent div
        """
        try:
            parent_id = graph_div.parentNode.getAttribute("id")
            assert not parent_id == ''

            if "div" in parent_id:
                divs_dict = self._web_components["divs"]
                class_parent = divs_dict[parent_id]
            else:
                containers = self._web_components["sections_footers_divs"]
                class_parent = containers[parent_id]
        except:
            continue

        parent_frame = self._image.copy()
        xmin, ymin, xmax, ymax = class_parent.getCoords()
        parent_frame = parent_frame[ymin:ymax, xmin:xmax]

        if "div" in parent_id:
            img_filter_color_frame = CSSHelperExtractor.imageFilterColor(
                class_parent,
                parent_frame,
                [
                    self._web_components["texts"],
                    self._web_components["buttons"],
                    self._web_components["images"]
                ]
            )
        else:
            img_filter_color_frame = CSSHelperExtractor.imageFilterColor(
                class_parent,
                parent_frame,
                [self._web_components["divs"]]
            )
```

```
parent_element_color = CSSHelperExtractor.getElementColor(
    img_filter_color_frame
)

if CSSHelperExtractor.isBlack(parent_element_color):
    try:
        parent_element_color = CSSHelperExtractor.getElementColor(
            img_filter_color_frame,
            text_color=True
        )
    except:
        parent_element_color = (0, 0, 0)

"""
Colors evaluation
"""
(I1R, I1G, I1B) = element_color
(I2R, I2G, I2B) = parent_element_color
try:
    d = math.sqrt((I1G-I2G)^2+(I1B-I2B)^2+(I1R-I2R)^2)
except:
    d = 0

if d > 4.5: #the div has background
    bem_gen.createBlock(
        div_key,
        prefix="#",
        styles={
            "background-color": f'rgb{element_color}'
        }
    )

"""
If the div is a row ...
"""
if graph_div.getAttribute("row") == "True":
    extra_size = 20

    # If there exists 'row' word in div key we need
    # to remove 'div-' part to use the rest as the
    # row key in elements in rows.
    if "div_row_section" in div_key:
        extra_size *= 4
        row_key = div_key.replace(f"div_row_{parent_id}_", "")
        div_elements = SectionAnalyzer.elements_in_rows[parent_id][row_key][0]
    elif "div_row_footer" in div_key:
        extra_size *= 4
        row_key = div_key.replace(f"div_row_{parent_id}_", "")
        div_elements = FooterAnalyzer.elements_in_rows[parent_id][row_key][0]
    elif "col" in div_key:
        col_key = div_key.replace(f"{parent_id}-", "")
        div_elements = DivAnalyzer.elements_in_cols[parent_id][col_key][0]
    elif "row" in div_key:
        row_key = div_key.replace(f"{parent_id}-", "")
        div_elements = DivAnalyzer.elements_in_rows[parent_id][row_key][0]
    else:
        try:
            div_elements = DivAnalyzer.elements_in_rows[div_key]["row_0"][0]
        except:
            continue

    div_elements = sorted(
        div_elements,
        key=lambda k: k.xmin
    )

    justify_content = CSSAnalyzer.getJustifyContent(
        class_div.xmax,
        div_elements,
        x_min_container=class_div.xmin
    )

    bem_gen.createBlock(
        div_key,
        prefix="#",
        styles={
            "width": f'{class_div.width␣+␣extra_size}px',
            "align-items": "center",
            "display": "flex",
            "flex-wrap": "wrap",
            "justify-content": f'{justify_content}'
        }
    )

    insertDivInMedia(div_key, class_div.width)

    margins = CSSAnalyzer.getElementMarginsForJustifyContent(
        div_elements,
```

```
            class_div.xmax,
            justify_content,
            x_min_container=class_div.xmin
        )

        for key_element, element_features in margins.items():
            if "div" in key_element:
                bem_gen.createBlock(
                    key_element,
                    styles=element_features
                )
            else:
                graph_element = xml_gen.getElementByClassName(
                    element_features["tag"],
                    key_element
                )
                css_pattern = graph_element.getAttribute(
                    "css_pattern"
                )
                bem_gen.createModifier(
                    css_pattern,
                    element_features["tag"],
                    key_element,
                    styles=element_features
                )

    else:
        div_elements = []
        for graph_child_div in graph_div.childNodes:
            # Keys from class attribute has the
            # BEM format so we need to split it
            # to get the last string as the
            # real key.

            if "div" in graph_child_div.tagName:
                key = graph_child_div.getAttribute("id")
                div_elements.append(
                    self._web_components["sections_footers_divs"][key]
                )

            elif ("h" in graph_child_div.tagName or
                "p" in graph_child_div.tagName):
                key = graph_child_div.getAttribute("class")
                key = key.split("_")[-1]
                div_elements.append(
                    self._web_components["texts"][key]
                )

            elif "button" in graph_child_div.tagName:
                key = graph_child_div.getAttribute("class")
                key = key.split("_")[-1]
                div_elements.append(
                    self._web_components["buttons"][key]
                )

            elif "img" in graph_child_div.tagName:
                key = graph_child_div.getAttribute("class")
                key = key.split("_")[-1]
                div_elements.append(
                    self._web_components["images"][key]
                )

        div_elements = sorted(
            div_elements,
            key=lambda k: k.xmin
        )

        align_items = CSSAnalyzer.getAlignItems(
            class_div.xmax,
            div_elements,
            x_min_container=class_div.xmin
        )
        text_align = CSSAnalyzer.getTextAlign(align_items)

        bem_gen.createBlock(
            div_key,
            prefix="#",
            styles={
                "width": f'{class_div.width}px',
                "display": "flex",
                "flex-direction": "column",
                "justify-content": "center",
                "align-items": f'{align_items}',
                "text-align": f'{text_align}'
            }
        )

        insertDivInMedia(div_key, class_div.width)
```

```
                    margins = CSSAnalyzer.getElementMarginsForAlignItems(
                        div_elements
                    )

                    for key_element, element_features in margins.items():
                        if "div" in key_element:
                            bem_gen.createBlock(
                                key_element,
                                styles=element_features
                            )
                        else:
                            bem_gen.createModifier(
                                class_div.key,
                                element_features["tag"],
                                key_element,
                                styles=element_features
                            )

        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣getDivsSpecs␣-␣{e_type}:␣{e}'

            CSSFeaturesExtractor._logger.critical(e_msg)
            raise Exception(e_msg)

        CSSFeaturesExtractor._logger.info("getDivsSpecs␣successfully␣executed")

    def getFooterDivsSpecs(self, bem_gen, xml_gen):
        try:
            def insertDivInMedia(div_key, div_width):
                if div_width > 800:
                    bem_gen.createMedia(
                        "1200px",
                        div_key,
                        prefix="#",
                        styles={
                            "width": "100%",
                        }
                    )
                if div_width > 480 and div_width < 800:
                    bem_gen.createMedia(
                        "800px",
                        div_key,
                        prefix="#",
                        styles={
                            "width": "100%",
                        }
                    )
                if div_width < 480:
                    bem_gen.createMedia(
                        "480px",
                        div_key,
                        prefix="#",
                        styles={
                            "width": "100%",
                        }
                    )

            for graph_div in xml_gen.aux_xml.getElementsByTagName("div"):
                div_key = graph_div.getAttribute("id")
                try:
                    class_div = self._web_components["footer_divs"][div_key]
                except:
                    continue

                if "footer" in class_div.key:
                    if graph_div.hasAttribute("row"):
                        if graph_div.getAttribute("row") == "True":

                            # If there exists 'row' word in div key we
                            # need to remove 'footer-div-' part to use
                            # the rest as the row key in elements in
                            # rows.
                            if "row" in div_key:
                                parent_id = graph_div.parentNode.getAttribute("id")
                                row_key = div_key.replace("footer-div-", "")
                                div_elements = FooterAnalyzer.elements_in_rows[parent_id][row_key][0]
                            else:
                                div_elements = FooterAnalyzer.elements_in_rows[div_key]["row_0"][0]

                            div_elements = sorted(
                                div_elements,
                                key=lambda k: k.xmin
                            )

                            justify_content = CSSAnalyzer.getJustifyContent(
                                class_div.xmax,
                                div_elements,
                                x_min_container=class_div.xmin
```

```
        )

        bem_gen.createBlock(
            div_key,
            prefix="#",
            styles={
                "width": f'{class_div.width}px',
                "align-items": "center",
                "display": "flex",
                "flex-wrap": "wrap",
                "justify-content": f'{justify_content}'
            }
        )

        insertDivInMedia(div_key, class_div.width)

        margins = CSSAnalyzer.getElementMarginsForJustifyContent(
            div_elements,
            class_div.xmax,
            justify_content,
            x_min_container=class_div.xmin
        )

        for key_element, element_features in margins.items():
            if "div" in key_element:
                bem_gen.createBlock(
                    key_element,
                    styles=element_features
                )
            else:
                bem_gen.createModifier(
                    class_div.key,
                    element_features["tag"],
                    key_element,
                    styles=element_features
                )

    else:
        div_elements = []
        for graph_child_div in graph_div.childNodes:
            # Keys from class attribute has the
            # BEM format so we need to split it
            # to get the last string as the
            # real key.

            if "div" in graph_child_div.tagName:
                key = graph_child_div.getAttribute("id")
                div_elements.append(
                    self._web_components["footer_divs"][key]
                )

            elif ("h" in graph_child_div.tagName or
                "p" in graph_child_div.tagName):
                key = graph_child_div.getAttribute("class")
                key = key.split("_")[-1]
                div_elements.append(
                    self._web_components["texts"][key]
                )

            elif "button" in graph_child_div.tagName:
                key = graph_child_div.getAttribute("class")
                key = key.split("_")[-1]
                div_elements.append(
                    self._web_components["buttons"][key]
                )

            elif "img" in graph_child_div.tagName:
                key = graph_child_div.getAttribute("class")
                key = key.split("_")[-1]
                div_elements.append(
                    self._web_components["images"][key]
                )

        div_elements = sorted(
            div_elements,
            key=lambda k: k.xmin
        )

        align_items = CSSAnalyzer.getAlignItems(
            class_div.xmax,
            div_elements,
            x_min_container=class_div.xmin
        )
        text_align = CSSAnalyzer.getTextAlign(align_items)

        bem_gen.createBlock(
            div_key,
            prefix="#",
            styles={
```

```
                            "width": f'{class_div.width}px',
                            "display": "flex",
                            "flex-direction": "column",
                            "justify-content": "center",
                            "align-items": f'{align_items}',
                            "text-align": f'{text_align}'
                        }
                    )

                    insertDivInMedia(div_key, class_div.width)

                    margins = CSSAnalyzer.getElementMarginsForAlignItems(
                        div_elements
                    )

                    for key_element, element_features in margins.items():
                        if "div" in key_element:
                            bem_gen.createBlock(
                                key_element,
                                styles=element_features
                            )
                        else:
                            bem_gen.createModifier(
                                class_div.key,
                                element_features["tag"],
                                key_element,
                                styles=element_features
                            )

        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣getFooterDivsSpecs␣-␣{e_type}:␣{e}'

            CSSFeaturesExtractor._logger.critical(e_msg)
            raise Exception(e_msg)

        CSSFeaturesExtractor._logger.info("getFooterDivsSpecs␣successfully␣executed")

    # Get CSS header's styles
    def getHeadersSpecs(self, bem_gen):
        try:
            _, width, _ = self._image.shape # Get website size

            # Analyze each header.
            for key, class_header in self._web_components["headers"].items():
                # Crop the header section of the webpage image
                frame = self._image.copy()

                xmin, ymin, xmax, ymax = class_header.getCoords()

                frame = frame[
                    ymin:ymax,
                    xmin:xmax
                ]
                # Get background color of header
                bg_color = CSSHelperExtractor.getElementColor(frame)


                rows_header = HeaderAnalyzer.elements_in_rows[key]
                # get the max ymax element inside of section
                max_ymax_element = class_header.ymin
                for _, row  in rows_header.items():
                    aux_ymax_element = row[1]
                    if aux_ymax_element > max_ymax_element:
                        max_ymax_element = aux_ymax_element

                padding_bottom = abs(class_header.height - max_ymax_element)

                # Apply CSS styles in header block
                bem_gen.createBlock(
                    key,
                    prefix="#",
                    styles={
                        "background-color": f'rgb{bg_color}',
                        "padding": f'{padding_bottom}px␣0px',
                        "width": "100%"
                    }
                )

                bem_gen.createBlock(
                    "icon_menu",
                    styles={
                        "display": "none",
                        "cursor": "pointer",
                        "font-size": "25px"
                    }
                )

                bem_gen.createMedia(
```

```
    "1200px",
    key,
    prefix="#",
    styles={
        "flex-direction": "row-reverse",
        "justify-content": "space-between"
    }
)

bem_gen.createMedia(
    "1200px",
    "icon_menu",
    styles={
        "display": "flex",
        "margin-right": "15px",
    }
)

bem_gen.createElement(
    "header",
    "checkbox",
    styles={
        "display": "none",
    }
)

# Get all header elements that are inside of each row
header_key = class_header.key
header_rows = HeaderAnalyzer.elements_in_rows[header_key]

for row_key, row_values in header_rows.items():
    header_elements = row_values[0]
    header_elements = sorted(
        header_elements,
        key=lambda k: k.xmin
    )

    # If there exists more than one row the parent key
    # will be the row_key and the '-div', otherwise
    # we just use the header key.
    if len(header_rows) > 1:
        parent_key = "header__" + row_key + "-div"
    else:
        parent_key = key

    """
    It's necessary determine the justify-content of the header,
    could be [flex-start, flex-end, center, space-between, space-around].
    [Image link for reference]
    https://cdn.discordapp.com/attachments/858054490762379324/873253503081009252/justify-content.png
    """
    # Get justify-content of header
    justify_content = CSSAnalyzer.getJustifyContent(
        width, header_elements
    )

    # Apply CSS styles in header block
    bem_gen.createBlock(
        parent_key,
        prefix="#",
        styles={
            "width": "100%",
            "display": "flex",
            "align-items": "center",
            "justify-content": f'{justify_content}'
        }
    )

    # Insert margins of each header's element.
    element_margins = CSSAnalyzer.getElementMarginsForJustifyContent(
        header_elements,
        width,
        justify_content
    )
    for key_element, element_features in element_margins.items():
        if "nav" in key_element:
            bem_gen.createBlock(
                key_element,
                styles=element_features
            )
        else:
            bem_gen.createModifier(
                class_header.tag,
                element_features["tag"],
                key_element,
                styles=element_features
            )

    # Insert CSS in header's element.
```

```
                for element in header_elements:
                    if "image" in element.key:
                        bem_gen.createModifier(
                            class_header.tag,
                            element.tag,
                            element.key,
                            styles={
                                "width": f'{element.width}px',
                                "object-fit": "cover"
                            }
                        )

                    elif "txtheader" in element.key:
                        # Crop the title that is in the header
                        frame_text = self._image.copy()
                        xmin, ymin, xmax, ymax = element.getCoords()
                        frame_text = frame_text[ymin:ymax+15, xmin:xmax+15]

                        # Get text's color of title
                        element_color = CSSHelperExtractor.getElementColor(
                            frame_text,
                            text_color=True
                        )

                        # The margin top is fixed to be 1/3 of the element.
                        margin_top = int(element.height / 3)

                        fontsize = element.fontsize
                        fontsize += int(self.INCREASE_FONT_SIZE / fontsize)
                        bem_gen.createModifier(
                            class_header.tag,
                            element.tag,
                            element.key,
                            styles={
                                "margin-top": f"{margin_top}px",
                                "margin-bottom": "0em",
                                "color": f'rgb{element_color}',
                                "font-size": f'{fontsize}px',
                            }
                        )

                    elif "nav" in element.key:
                        bem_gen.createBlock(
                            element.key,
                            styles={
                                "width": f'{element.width␣+␣90}px',
                            }
                        )

        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣getHeadersSpecs␣-␣{e_type}:␣{e}'

            CSSFeaturesExtractor._logger.critical(e_msg)
            raise Exception(e_msg)

        CSSFeaturesExtractor._logger.info("getHeadersSpecs␣successfully␣executed")

    # Get CSS nav's styles
    def getNavSpecs(self, bem_gen, xml_gen):
        try:
            # Get the image width.
            _, width, _ = self._image.shape
            for key, class_nav in self._web_components["navs"].copy().items():
                # Get the nav element from the XML graph.
                graph_nav = xml_gen.getElementByClassName(
                    "nav", class_nav.key
                )
                if not graph_nav:
                    del self._web_components["navs"][key]
                    continue

                bem_gen.createBlock(
                    key,
                    styles={
                        "width": f'{class_nav.width}px',
                        "align-items": "center",
                        "display": "flex",
                        "justify-content": "center"
                    }
                )

                bem_gen.createMedia(
                    "1200px",
                    key,
                    styles={
                        "display": "none",
                        "position": "absolute",
                        "width": "100%",
```

```
        "top": f"{class_nav.ymax␣+␣5}px"
    }
)

# Create the BEM element for the <ul> tag into the nav.
bem_gen.createElement(
    class_nav.key,
    "ul",
    styles={
        "list-style": "none",
        "width": "100%",
        "display": "flex",
        "justify-content": "space-between",
        "align-items": "center"
    }
)

frame = self._image.copy()

xmin, ymin, xmax, ymax = class_nav.getCoords()

frame = frame[
    ymin:ymax,
    xmin:xmax
]
# Get background color of header
bg_color = CSSHelperExtractor.getElementColor(frame)

bem_gen.createMedia(
    "1200px",
    class_nav.key,
    element="ul",
    styles={
        "flex-direction": "column",
        "background-color": f'rgb{bg_color}',
    }
)

bem_gen.createMedia(
    "1200px",
    "header",
    element=f"checkbox:checked␣~␣.{class_nav.key}",
    styles={
        "display": "flex",
        "flex-direction": "column",
    }
)


# Create the BEM styles for each element in <nav>.
txtnavitems = {}
for element in NavAnalyzer.elements[class_nav.key]:

    if "txt" in element.key:
        xmin, ymin, xmax, ymax = element.getCoords()

        # Get the frame cropped for text.
        frame_text = self._image.copy()
        frame_text = frame_text[ymin:ymax, xmin:xmax]

        # Get the padding right.
        padding_right = str(int(width) - int(xmax))

        # Get the parent key, because is the BEM block.
        graph_text = xml_gen.getElementByClassName(
            "a", element.key
        )
        block_key = graph_text.getAttribute("css_pattern")

        # Get the element color.
        element_color = CSSHelperExtractor.getElementColor(
            frame_text,
            text_color=True
        )

        fontsize = element.fontsize
        fontsize += int(self.INCREASE_FONT_SIZE / fontsize)
        # Get the properties of each txt nav item.
        key_txtnav = block_key + "__a__" + element.key
        txtnavitems[key_txtnav] = [
            element_color,
            fontsize
        ]

# Get the styles of each group and the assignations.
txtnav_groups, group_mapping = NavAnalyzer.mergeTxtNavStyles(
    txtnavitems
)
```

```
            # Insert the css styles of text navs elements.
            for txtnav_key, txtnav_value in txtnav_groups.items():
                # Get the block and elements keys from BEM format.
                block_key = txtnav_key.split("_")[0]
                element_key = txtnav_key.split("_")[-1]

                # Insert styles in css file.
                bem_gen.createModifier(
                    block_key,
                    "a",
                    element_key,
                    styles=txtnav_value
                )

                bem_gen.createModifier(
                    block_key,
                    "a",
                    element_key,
                    styles={
                        "text-decoration": "none",
                        "margin-top": "0.5em",
                        "margin-bottom": "0.5em",
                    }
                )

            # Now it is going to be necessary replace
            # the old class attributes with the assignation
            # of the group class attribute.
            for searching_class, new_class in group_mapping.items():
                # Replace the class of txtnavitems
                xml_gen.replaceClassName(
                    "a",
                    searching_class,
                    new_class
                )

    except Exception as e:
        e_type = str(type(e))[8:-2]
        e_msg = f'IN␣getNavSpecs␣-␣{e_type}:␣{e}'

        CSSFeaturesExtractor._logger.critical(e_msg)
        raise Exception(e_msg)

    CSSFeaturesExtractor._logger.info("getNavSpecs␣successfully␣executed")

# Get the section attributes.
def getSectionFooterSpecs(self, bem_gen, xml_gen, type):
    try:
        _, img_width, _ = self._image.shape
        for graph_section_footer in xml_gen.aux_xml.getElementsByTagName(type):
            section_footer_key = graph_section_footer.getAttribute("id")
            class_section_footer = self._web_components["sections_footers_divs"][section_footer_key]
            section_footer_frame = self._image.copy()
            xmin, ymin, xmax, ymax = class_section_footer.getCoords()
            section_footer_frame = section_footer_frame[ymin:ymax, xmin:xmax]
            img_filter_color_frame = CSSHelperExtractor.imageFilterColor(
                class_section_footer,
                section_footer_frame,
                [self._web_components["divs"]]
            )
            # NOTE: Remove after validate all coord values.
            if 0 in img_filter_color_frame.shape:
                continue

            # If the bg color is black, it will necessary check again, because maybe
            # the reason it's that there are black rectangles to cover the images.
            element_color = CSSHelperExtractor.getElementColor(img_filter_color_frame)
            # NOTE: with text_color=True it is possible to obtain
            # the second predominant color.
            if CSSHelperExtractor.isBlack(element_color):
                element_color = CSSHelperExtractor.getElementColor(
                    img_filter_color_frame,
                    text_color=True
                )

            if "section" in type:
                rows_section_footer = SectionAnalyzer.elements_in_rows[section_footer_key]
            else:
                rows_section_footer = FooterAnalyzer.elements_in_rows[section_footer_key]

            # Get padding top and bottom
            previous_sibling = graph_section_footer.previousSibling
            try:
                assert (previous_sibling.tagName == "section"
                    or previous_sibling.tagName == "header"
                    or previous_sibling.tagName == "footer")

                if (previous_sibling.tagName == "section"
                        or previous_sibling.tagName == "footer"):
```

```
        dict_name = "sections_footers_divs"
    else:
        dict_name = "headers"

    last_key = previous_sibling.getAttribute("id")
    class_last = self._web_components[dict_name][last_key]

    top_value = class_section_footer.ymin - class_last.ymax
    if top_value < -50:
        top_value = 0
    padding = top_value
except:
    ymin_value = class_section_footer.ymin
    if not previous_sibling:
        if ymin_value < 250:
            padding = ymin_value
        else:
            padding = 0
    else:
        padding = ymin_value

# End padding top and bottom

if type == "footer":
    bem_gen.createBlock(
        section_footer_key,
        prefix="#",
        styles={
            "height": f'{class_section_footer.height}px'
        }
    )

bem_gen.createBlock(
    section_footer_key,
    prefix="#",
    styles={
        "padding": f'{padding}px␣0px',
        "background-color": f'rgb{element_color}'
    }
)

if len(rows_section_footer) > 1:

    div_elements = []
    for graph_child in graph_section_footer.childNodes:
        if "img" in graph_child.tagName:
            continue
        child_key = graph_child.getAttribute("id")
        class_div = self._web_components["sections_footers_divs"][child_key]
        div_elements.append(class_div)

    div_elements = sorted(
        div_elements,
        key=lambda k: k.xmin
    )

    align_items = CSSAnalyzer.getAlignItems(
        img_width, div_elements
    )

    text_align = CSSAnalyzer.getTextAlign(align_items)

    bem_gen.createBlock(
        section_footer_key,
        prefix="#",
        styles={
            "width": "100%",
            "display": "flex",
            "flex-direction": "column",
            "justify-content": "center",
            "align-items": f'{align_items}',
            "text-align": f'{text_align}'
        }
    )

    margins = CSSAnalyzer.getElementMarginsForAlignItems(
        div_elements
    )

    for key_element, element_features in margins.items():
        if "div" in key_element:
            bem_gen.createBlock(
                key_element,
                styles=element_features
            )
elif "row_0" in rows_section_footer:
    section_row_elements = rows_section_footer["row_0"][0]
    section_row_elements = sorted(
        section_row_elements,
```

```
                    key=lambda k: k.xmin
                )

                justify_content = CSSAnalyzer.getJustifyContent(
                    img_width,
                    section_row_elements
                )

                bem_gen.createBlock(
                    section_footer_key,
                    prefix="#",
                    styles={
                        "width": "100%",
                        "display": "flex",
                        "align-items": "center",
                        "justify-content": f'{justify_content}'
                })

                element_margins = CSSAnalyzer.getElementMarginsForJustifyContent(
                    section_row_elements,
                    img_width,
                    justify_content
                )
                for key_element, element_features in element_margins.items():
                    if "div" in key_element:
                        bem_gen.createBlock(
                            key_element,
                            styles=element_features
                        )
                    else:
                        bem_gen.createModifier(
                            class_section_footer.key,
                            element_features["tag"],
                            key_element,
                            styles=element_features
                        )
        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣getSectionFooterSpecs␣-␣{e_type}:␣{e}'

            CSSFeaturesExtractor._logger.critical(e_msg)
            raise Exception(e_msg)

        CSSFeaturesExtractor._logger.info("getSectionFooterSpecs␣successfully␣executed")

#-------------------------------------------------------------------
    # NOTE:
    # It will necessary create a complete refactor from this point.

    # Get the images attributes.
    def getImagesSpecs(self, bem_gen, xml_gen):
        try:
            # Analyze each image.
            for key, class_image in self._web_components["images"].items():
                if not class_image.isAllocated():
                    continue

                graph_image = xml_gen.getElementByClassName(
                    class_image.tag, key
                )
                block_key = graph_image.getAttribute("css_pattern")

                if "header" not in block_key:
                    # If the block key is not a header, we can inference,
                    # that will be a div
                    try:
                        div_parent = self._web_components["divs"][block_key]
                        bem_gen.createModifier(
                                block_key,
                                class_image.tag,
                                key,
                                styles={
                                    "width": f'{class_image.width}px',
                                    "height": f'{class_image.height}px'
                                }
                            )
                        """
                        image_width = class_image.width
                        div_width = div_parent.width
                        if image_width > div_width:
                            bem_gen.createModifier(
                                block_key,
                                class_image.tag,
                                key,
                                styles={
                                    "width": "100%"
                                }
                            )
                        else:
```

```
                        bem_gen.createModifier(
                            block_key,
                            class_image.tag,
                            key,
                            styles={
                                "width": f'{image_width}px',
                            }
                        )
                    """
                except:
                    # If an image is inside directly in a section,
                    # it won't find the key in divs dictionary
                    e_msg = f'Image␣inside␣of␣section␣without␣div.'
                    CSSFeaturesExtractor._logger.error(e_msg)
        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣getImageSpecs␣-␣{e_type}:␣{e}'

            CSSFeaturesExtractor._logger.critical(e_msg)
            raise Exception(e_msg)

        CSSFeaturesExtractor._logger.info("getImagesSpecs␣successfully␣executed")

    def getTextFooterSpecs(self, bem_gen, xml_gen):
        try:
            _, width, _ = self._image.shape

            # Analyze each text in nav.
            for key, class_text in self._web_components["texts"].items():

                if "footer" in key:
                    frame_text = self._image.copy()
                    xmin, ymin, xmax, ymax = class_text.getCoords()
                    frame_text = frame_text[ymin:ymax, xmin:xmax]

                    # Create its StyledElement and append it to
                    # css_gen.
                    graph_text = xml_gen.getElementByClassName(
                        class_text.tag, key
                    )

                    if not graph_text:
                        continue

                    block_key = graph_text.getAttribute("css_pattern")

                    element_color = CSSHelperExtractor.getElementColor(
                        frame_text,
                        text_color=True
                    )

                    fontsize = class_text.fontsize
                    fontsize += int(self.INCREASE_FONT_SIZE / fontsize)
                    bem_gen.createModifier(
                        block_key,
                        class_text.tag,
                        key,
                        styles={
                            "color": f'rgb{element_color}',
                            "font-size": f'{fontsize}px'
                        }
                    )
        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣getTextFooterSpecs␣-␣{e_type}:␣{e}'

            CSSFeaturesExtractor._logger.critical(e_msg)
            raise Exception(e_msg)

        CSSFeaturesExtractor._logger.info("getTextFooterSpecs␣successfully␣executed")

    # Get the text attributes.
    def getTextsSpecs(self, bem_gen, xml_gen):
        try:
            # Analyze all the texts that don't belong to other
            # elements.
            for key, class_text in self._web_components["texts"].items():
                if ("h" in class_text.tag
                        or "p" in class_text.tag):
                    frame_text = self._image.copy()
                    xmin, ymin, xmax, ymax = class_text.getCoords()
                    frame_text = frame_text[ymin:ymax, xmin:xmax]

                    graph_text = xml_gen.getElementByClassName(
                        class_text.tag, key
                    )

                    if not graph_text:
                        continue
```

```
                    block_key = graph_text.getAttribute("css_pattern")

                    element_color = CSSHelperExtractor.getElementColor(
                        frame_text,
                        text_color=True
                    )

                    fontsize = class_text.fontsize
                    fontsize += int(self.INCREASE_FONT_SIZE / fontsize)
                    bem_gen.createModifier(
                        block_key,
                        class_text.tag,
                        key,
                        styles={
                            "margin-bottom": "0em",
                            "color": f'rgb{element_color}',
                            "font-size": f'{fontsize}px'
                        }
                    )

            except Exception as e:
                e_type = str(type(e))[8:-2]
                e_msg = f'IN␣getTextsSpecs␣-␣{e_type}:␣{e}'

                CSSFeaturesExtractor._logger.critical(e_msg)
                raise Exception(e_msg)

            CSSFeaturesExtractor._logger.info("getTextsSpecs␣successfully␣executed")
```

B.14 CSS Helper Extractor Code

```
import cv2
import numpy as np
from sklearn.cluster import KMeans

from logs.log import Log


class CSSHelperExtractor:
    # Logging
    _logger = None

    def __init__(self) -> None:
        try:
            CSSHelperExtractor._logger = Log("CSSHelperExtractor")

        except Exception as e:
            e_type = str(type(e))[8:-2]
            e_msg = f'IN␣__init__␣-␣{e_type}:␣{e}'

            CSSHelperExtractor._logger.critical(e_msg)
            raise Exception(e_msg)

        CSSHelperExtractor._logger.info("SUCCESSFULLY␣CREATED")

    ##################
    # STATIC METHODS #

    # Get the colors and its percentage.
    @staticmethod
    def getColorsAndPercent(cluster, centroids):
        try:
            # Get the number of different clusters, create histogram,
            # and normalize.
            labels = np.arange(0, len(np.unique(cluster.labels_)) + 1)
            (hist, _) = np.histogram(cluster.labels_, bins=labels)

            hist = hist.astype("float")
            hist /= hist.sum()

            colors = sorted([
                (percent, color) for (percent, color) in zip(hist, centroids)
            ])
            colors_rgb =[]

            for (percent, color) in colors:
                colors_rgb.append([
                    percent,
                    (int(color[0]), int(color[1]),
                    int(color[2]))
                ])

        except Exception as e:
            e_type = str(type(e))[8:-2]
```

```
        e_msg = f'IN␣getColorsAndPercent␣-␣{e_type}:␣{e}'

        CSSHelperExtractor._logger.critical(e_msg)
        raise Exception(e_msg)

    CSSHelperExtractor._logger.info("getColorsAndPercent␣successfully␣executed")

    return colors_rgb

# Get the color of a given element, it can be used to get the
# text color.
@staticmethod
def getElementColor(image, text_color=False):
    try:
        frame = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        reshape = frame.reshape((frame.shape[0] * frame.shape[1], 3))

        cluster = KMeans(n_clusters=2).fit(reshape)
        colors = CSSHelperExtractor.getColorsAndPercent(
            cluster,
            cluster.cluster_centers_
        )
    except Exception as e:
        e_type = str(type(e))[8:-2]
        e_msg = f'IN␣getElementColor␣-␣{e_type}:␣{e}'

        CSSHelperExtractor._logger.critical(e_msg)
        raise Exception(e_msg)

    CSSHelperExtractor._logger.info("getElementColor␣successfully␣executed")
    # Always the last color of the list has the biggest area.
    if text_color:
        return colors[-2][1]
    else:
        return colors[-1][1]

@staticmethod
def imageFilterColor(class_box, box_frame, list_elements):
    try:
        # Replace the webpage with white rectangle
        h, w, c = box_frame.shape
        image_filter_frame = box_frame.copy()

        for element_dictionary in list_elements:
            # Insert black rectangles for images identified
            for key, class_element in element_dictionary.items():

                if class_box.has(class_element):
                    x_min, y_min, x_max, y_max = class_element.getCoords()
                    cv2.rectangle(
                        image_filter_frame,
                        (x_min - class_box.xmin, y_min - class_box.ymin),
                        (x_max - class_box.xmin, y_max - class_box.ymin),
                        (0, 0, 0),
                        -1 # Filled rectangle color
                    )

    except Exception as e:
        e_type = str(type(e))[8:-2]
        e_msg = f'IN␣imageFilterColor␣-␣{e_type}:␣{e}'

        CSSHelperExtractor._logger.critical(e_msg)
        raise Exception(e_msg)

    CSSHelperExtractor._logger.info("imageFilterColor␣successfully␣executed")

    return image_filter_frame

@staticmethod
def isBlack(color):
    try:

        (r, g, b) = color
        if r < 20 and g < 20 and b < 20:
            return True
        else:
            return False

    except Exception as e:
        e_type = str(type(e))[8:-2]
        e_msg = f'IN␣isBlack␣-␣{e_type}:␣{e}'

        CSSHelperExtractor._logger.critical(e_msg)
        raise Exception(e_msg)
```

B.15 Normalize.css

```
/*! normalize.css v8.0.1 | MIT License | github.com/necolas/normalize.css */

/* Document
   ========================================================================== */

/**
 * 1. Correct the line height in all browsers.
 * 2. Prevent adjustments of font size after orientation changes in iOS.
 */

html {
  line-height: 1.15; /* 1 */
  -webkit-text-size-adjust: 100%; /* 2 */
}

/* Sections
   ========================================================================== */

/**
 * Remove the margin in all browsers.
 */

body {
  margin: 0;
}

/**
 * Render the `main` element consistently in IE.
 */

main {
  display: block;
}

/**
 * Correct the font size and margin on `h1` elements within `section` and
 * `article` contexts in Chrome, Firefox, and Safari.
 */

h1 {
  font-size: 2em;
  margin: 0.67em 0;
}

/* Grouping content
   ========================================================================== */

/**
 * 1. Add the correct box sizing in Firefox.
 * 2. Show the overflow in Edge and IE.
 */

hr {
  box-sizing: content-box; /* 1 */
  height: 0; /* 1 */
  overflow: visible; /* 2 */
}

/**
 * 1. Correct the inheritance and scaling of font size in all browsers.
 * 2. Correct the odd `em` font sizing in all browsers.
 */

pre {
  font-family: monospace, monospace; /* 1 */
  font-size: 1em; /* 2 */
}

/* Text-level semantics
   ========================================================================== */

/**
 * Remove the gray background on active links in IE 10.
 */

a {
  background-color: transparent;
}

/**
 * 1. Remove the bottom border in Chrome 57-
 * 2. Add the correct text decoration in Chrome, Edge, IE, Opera, and Safari.
 */

abbr[title] {
  border-bottom: none; /* 1 */
  text-decoration: underline; /* 2 */
  text-decoration: underline dotted; /* 2 */
}

/**
 * Add the correct font weight in Chrome, Edge, and Safari.
 */

b,
strong {
  font-weight: bolder;
}

/**
 * 1. Correct the inheritance and scaling of font size in all browsers.
 * 2. Correct the odd `em` font sizing in all browsers.
 */

code,
kbd,
samp {
  font-family: monospace, monospace; /* 1 */
  font-size: 1em; /* 2 */
}

/**
 * Add the correct font size in all browsers.
 */

small {
  font-size: 80%;
}

/**
 * Prevent `sub` and `sup` elements from affecting the line height in
 * all browsers.
 */

sub,
sup {
  font-size: 75%;
  line-height: 0;
  position: relative;
  vertical-align: baseline;
}

sub {
  bottom: -0.25em;
}

sup {
  top: -0.5em;
}

/* Embedded content
   ========================================================================== */

/**
 * Remove the border on images inside links in IE 10.
 */

img {
  border-style: none;
}

/* Forms
   ========================================================================== */

/**
 * 1. Change the font styles in all browsers.
 * 2. Remove the margin in Firefox and Safari.
 */

button,
input,
optgroup,
select,
textarea {
  font-family: inherit; /* 1 */
  font-size: 100%; /* 1 */
```

```
  line-height: 1.15; /* 1 */
  margin: 0; /* 2 */
}

/**
 * Show the overflow in IE.
 * 1. Show the overflow in Edge.
 */

button,
input { /* 1 */
  overflow: visible;
}

/**
 * Remove the inheritance of text transform in Edge, Firefox, and IE.
 * 1. Remove the inheritance of text transform in Firefox.
 */

button,
select { /* 1 */
  text-transform: none;
}

/**
 * Correct the inability to style clickable types in iOS and Safari.
 */

button,
[type="button"],
[type="reset"],
[type="submit"] {
  -webkit-appearance: button;
}

/**
 * Remove the inner border and padding in Firefox.
 */

button::-moz-focus-inner,
[type="button"]::-moz-focus-inner,
[type="reset"]::-moz-focus-inner,
[type="submit"]::-moz-focus-inner {
  border-style: none;
  padding: 0;
}

/**
 * Restore the focus styles unset by the previous rule.
 */

button:-moz-focusring,
[type="button"]:-moz-focusring,
[type="reset"]:-moz-focusring,
[type="submit"]:-moz-focusring {
  outline: 1px dotted ButtonText;
}

/**
 * Correct the padding in Firefox.
 */

fieldset {
  padding: 0.35em 0.75em 0.625em;
}

/**
 * 1. Correct the text wrapping in Edge and IE.
 * 2. Correct the color inheritance from `fieldset` elements in IE.
 * 3. Remove the padding so developers are not caught out when they zero out
 *    `fieldset` elements in all browsers.
 */

legend {
  box-sizing: border-box; /* 1 */
  color: inherit; /* 2 */
  display: table; /* 1 */
  max-width: 100%; /* 1 */
  padding: 0; /* 3 */
  white-space: normal; /* 1 */
}

/**
 * Add the correct vertical alignment in Chrome, Firefox, and Opera.
 */

progress {
  vertical-align: baseline;
}

/**
 * Remove the default vertical scrollbar in IE 10+.
 */

textarea {
  overflow: auto;
}

/**
 * 1. Add the correct box sizing in IE 10.
 * 2. Remove the padding in IE 10.
 */

[type="checkbox"],
[type="radio"] {
  box-sizing: border-box; /* 1 */
  padding: 0; /* 2 */
}

/**
 * Correct the cursor style of increment and decrement buttons in Chrome.
 */

[type="number"]::-webkit-inner-spin-button,
[type="number"]::-webkit-outer-spin-button {
  height: auto;
}

/**
 * 1. Correct the odd appearance in Chrome and Safari.
 * 2. Correct the outline style in Safari.
 */

[type="search"] {
  -webkit-appearance: textfield; /* 1 */
  outline-offset: -2px; /* 2 */
}

/**
 * Remove the inner padding in Chrome and Safari on macOS.
 */

[type="search"]::-webkit-search-decoration {
  -webkit-appearance: none;
}

/**
 * 1. Correct the inability to style clickable types in iOS and Safari.
 * 2. Change font properties to `inherit` in Safari.
 */

::-webkit-file-upload-button {
  -webkit-appearance: button; /* 1 */
  font: inherit; /* 2 */
}

/* Interactive
   ========================================================================== */

/*
 * Add the correct display in Edge, IE 10+, and Firefox.
 */

details {
  display: block;
}

/*
 * Add the correct display in all browsers.
 */

summary {
  display: list-item;
}

/* Misc
   ========================================================================== */

/**
 * Add the correct display in IE 10+.
 */

template {
  display: none;
}

/**
 * Add the correct display in IE 10.
```

```
 */                                                    display: none;
                                                     }
[hidden] {
```

B.16 Autoencoder Model Code

```
import tensorflow as tf


class Autoencoder(tf.keras.Model):
    def __init__(self, input_shape, latent_dim=3):
        super(Autoencoder, self).__init__()

        # Encoder
        self.encoder = tf.keras.Sequential([
            tf.keras.layers.InputLayer(
                input_shape=input_shape
            ),  # Input shape now has 3 channels
            tf.keras.layers.Conv2D(
                32, (3, 3),
                activation='relu',
                padding='same',
                strides=2
            ),
            tf.keras.layers.Conv2D(
                64, (3, 3),
                activation='relu',
                padding='same',
                strides=2
            ),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(latent_dim)
        ])

        # Decoder
        self.decoder = tf.keras.Sequential([
            tf.keras.layers.InputLayer(input_shape=(latent_dim,)),
            tf.keras.layers.Dense(
                (input_shape[0] // 4) * (input_shape[1] // 4) * 64,
                activation='relu'
            ),
            tf.keras.layers.Reshape(
                target_shape=(input_shape[0] // 4, input_shape[1] // 4, 64)
            ),
            tf.keras.layers.Conv2DTranspose(
                64, (3, 3),
                activation='relu',
                padding='same',
                strides=2
            ),
            tf.keras.layers.Conv2DTranspose(
                32, (3, 3),
                activation='relu',
                padding='same',
                strides=2
            ),
            tf.keras.layers.Conv2D(
                input_shape[2],
                (3, 3),
                activation='sigmoid',
                padding='same'
            )  # Output channels match input channels
        ])

    def call(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded

    @property
    def input_encoder(self):
        return self.encoder.input
```

B.17 Similarity Evaluation System Code

```
import numpy as np
import matplotlib.pyplot as plt

from glob import glob
```

```
import tensorflow as tf
from pathlib import Path
from skimage import io, transform, color

def multi_plots(
    images, rows=3, cols=3, scale=20, colormap="gray",
    vmin=None, vmax=None, col_titles=None, cell_texts=None,
    cell_colors=None, fontsize=14, alpha=1, rotation=0):

    maps = images
    if type(images) is np.ndarray:
        maps = [images[:,:,i] for i in range(images.shape[2])]

    size = maps[0].shape[:2]

    fsize = np.array([size[1] / rows, size[0] / cols])/10

    fsize/=fsize[0]
    fsize*=scale

    fig, axs = plt.subplots(
        nrows=rows, ncols=cols,
        figsize=(fsize[0],fsize[1]), squeeze=True,
        gridspec_kw={
                'top': 0.98,
              'bottom': 0.02,
              'left': 0.02,
              'right': 0.98,
              'wspace': 0.02,
              'hspace': 0.02
        }
    )

    axs = axs.flatten()
    if cell_colors is not None:
        lenght = len(cell_colors)
        for i,ax in enumerate(axs):
            if i < lenght:
                ax.patch.set_edgecolor(cell_colors[i])
                ax.patch.set_linewidth('5')

    lenght = len(maps)
    for i,ax in enumerate(axs):
        ax.set_xticks([])
        ax.set_yticks([])
        if i < lenght:
            ax.imshow( maps[i], colormap, vmin=vmin, vmax=vmax )
        else:
            ax.set_axis_off()

    if cell_texts is not None:
        lenght = len(cell_texts)
        for i,ax in enumerate(axs):
            if i < lenght:
                props = dict(
                    facecolor='white', alpha=alpha
                )
                ax.text(
                    0.1, 0.9, cell_texts[i],
                    fontsize=fontsize,verticalalignment='top',
                    bbox=props, rotation=rotation
                )

    if col_titles is not None:
        lenght = len(col_titles)
        for i,ax in enumerate(axs):
            if i < lenght:
                ax.set_title(col_titles[i], fontsize=32)


    return fig, axs

@tf.function
def load_image(path):
    image = tf.io.decode_jpeg(tf.io.read_file(path), channels=3)
    image = tf.cast(image, tf.float32) / 255.0
    image = tf.image.resize(image, (H, W))
    return image

from sklearn.manifold import TSNE
class saveImagesCallback(tf.keras.callbacks.Callback):
    def __init__(self, model, save_dir, dataset, fig, axs):
        super(tf.keras.callbacks.Callback, self).__init__()
        self.model = model
        self.dataset = dataset
        self.fig = fig
        self.axs = axs
        self.save_dir = save_dir
```

```
    def on_epoch_end(self, epoch, logs=None):

        print(epoch)
        for batch in self.dataset:
            embeddings = self.model.encoder(batch[0])
            preds = self.model.decoder(embeddings)
            break


        X_embedded = TSNE(
            n_components=2, init='random', perplexity=3
        ).fit_transform(embeddings)

        f, ax = plt.subplots(
            nrows=1, ncols=1,
            figsize=(10,10), squeeze=True,
            gridspec_kw={
                'top': 0.98,
              'bottom': 0.02,
              'left': 0.02,
              'right': 0.98,
              'wspace': 0.02,
              'hspace': 0.02
            }
        )


        ax.scatter(X_embedded[:,0], X_embedded[:,1])
        f.savefig(F"{self.save_dir}/embeddings_epoc_{epoch:05d}.png")
        plt.close(f)

        [ax.imshow(x) for ax,x in zip(self.axs , preds)]
        self.fig.savefig(F"{self.save_dir}/preds_epoc_{epoch:05d}.png")

def load_and_process_image(file_path):
    # Load the image from the file path, resize it, and possibly apply other preprocessing
    image = tf.io.read_file(file_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = tf.image.resize(image, [H, W])
    image = image / 255.0  # Normalize to [0,1]
    return image

training_files = []
testing_files = []
for file_path in sorted(glob('./training/*')):
    training_files.append(file_path)

for file_path in sorted(glob('./testing/*')):
    testing_files.append(file_path)

training = np.array(training_files)
testing = np.array(testing_files)
print(training, testing)

H,W,C = 240,240,3
save_dir = Path('results/')

sample = np.random.choice(training,64)
images = [transform.resize(io.imread(i),(H,W)) for i in sample]

if 'canvas' in locals():
    plt.close(canvas)
canvas,axis = multi_plots(images, rows=8, cols=8, scale=10)

train_samples = training.shape[0]
valid_samples = testing.shape[0]
train_inputs = tf.constant(training)
val_inputs = tf.constant(testing)

train_dataset = tf.data.Dataset.from_tensor_slices(train_inputs)
valid_dataset = tf.data.Dataset.from_tensor_slices(val_inputs)
train_dataset = train_dataset.map(
    lambda x: (load_and_process_image(x),
    load_and_process_image(x))
)
valid_dataset = valid_dataset.map(
    lambda x: (load_and_process_image(x),
    load_and_process_image(x))
)

BATCH_SIZE = 64
BUFFER_SIZE = 128

TRAIN_LENGTH = train_samples - valid_samples
STEPS_PER_EPOCH = TRAIN_LENGTH // BATCH_SIZE
train_dataset = train_dataset.cache().shuffle(
    BUFFER_SIZE
).batch(BATCH_SIZE).repeat()
```

```
train_dataset = train_dataset.prefetch(
    buffer_size=tf.data.experimental.AUTOTUNE
)
val_dataset = valid_dataset.batch(BATCH_SIZE)

VAL_SUBSPLITS = 1
VALIDATION_STEPS = valid_samples//BATCH_SIZE//VAL_SUBSPLITS

from model_autoencoder import Autoencoder
model = Autoencoder((H,W,C), latent_dim=3)
model.build((None, H, W, C))
model.summary()

filepath = str(save_dir)+"/simple_{epoch:04d}_{loss:.4f}.hdf5"

callbacks = [
    tf.keras.callbacks.ModelCheckpoint(
        filepath, monitor='loss', verbose=1,
        save_weights_only=True, save_best_only=True
    ),
    tf.keras.callbacks.ReduceLROnPlateau(
        monitor='loss', factor=0.1,  patience=10, min_lr=0.0001
    ),
    saveImagesCallback(model, save_dir, val_dataset, canvas, axis
)]

model.compile(optimizer='adam', loss='mae', metrics = ['mse'])
EPOCHS = 150

Path(save_dir).mkdir(parents=True, exist_ok=True)

model_history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=EPOCHS,
    steps_per_epoch=STEPS_PER_EPOCH,
    callbacks=callbacks
)

import pandas as pd
import datetime

date = datetime.datetime.now().strftime("%d_%m_%Y_%H_%M_%S")
csv_file = F"{save_dir}/history_{date}.csv"

history = pd.DataFrame(data=model_history.history)
history.to_csv(csv_file,index=False)
history.plot(figsize=(15,10))
best_model = sorted(glob(F"{save_dir}/*.hdf5"))[-1]
model.load_weights(best_model)

custom_files = []
for file_path in sorted(glob('./custom_test/*')):
    custom_files.append(file_path)
custom_entry = np.array(custom_files)
print(custom_entry)
sample = custom_entry[:8]
images = [transform.resize(io.imread(i),(H,W)) for i in sample]
custom_inputs = tf.constant(custom_entry)
custom_dataset = tf.data.Dataset.from_tensor_slices(
    (custom_inputs)
).map(load_image)
if 'canvas' in locals():
    plt.close(canvas)

canvas, axis = multi_plots(images, rows=2, cols=4, scale=10)

embeddings=[]

for i, (X, _) in enumerate(custom_dataset):
    X_batch = np.expand_dims(X, axis=0)  # Add the batch dimension
    Y = model.encoder.predict(X_batch)
    embeddings.append(Y)
    if i > 4:
        break

embeddings = np.vstack(embeddings)
from scipy.spatial import distance
val = distance.cdist(embeddings, embeddings, metric='cosine')
print(val.shape)
d1 = val[0][1]
d2 = val[2][5]
print(d1, d2)
```

Thesis presented to the Engineering Faculty of the Universidad Autónoma de San Luis Potosí to obtain the degree of Master in the subject of Computer Engineering.

Gerardo Franco Delgado
a226392@alumnos.uaslp.mx.
June, 2024