# Image to Code: Automatic HTML Code Generation from Images using Deep Learning based Model

Shreya Shrotriya *IIT2020501* | Himanshu Bhawnani *IIB2020035* | Mantraraj Gotecha *IIT2020269* | Shreyans Jain *IIT2020238* | Ishaan Kaustav *IIT2020261*

***Abstract*—In this research study, a unique machine learning algorithm is used to automatically synthesise HTML code from input photos. A convolutional neural network (CNN) and a recurrent neural network (RNN) are the two primary parts of the proposed system. The RNN generates the appropriate HTML code based on the recovered features after the CNN extracts the pertinent features from the input pictures. CNN is made to understand the visual elements that are important for web page design. A collection of feature maps that accurately represent the key elements of the input image make up CNN's output.**

**The RNN then constructs the HTML code using the learnt features after receiving the feature maps. A sequence-to-sequence model is used to train the RNN, with the output sequence being the matching HTML code, and the input sequence being the collection of feature maps produced by the CNN.**

## I. Introduction

Modern commerce and communication both depend on web development. For businesses to draw in and keep customers, websites must be both aesthetically pleasing and useful. However, designing web sites that react quickly to these objectives is an extremely demanding effort. Graphic designers, software professionals, end-users, business authorities, and others engaged in a variety of fields must collaborate to create web sites. Typically, the process begins with the mock-up design of the user interface by graphic designers or mock-up artists, either on paper or in a graphic editing programme, in accordance with the demands of the institution. Software professionals build code for the web pages based on these drafts.
Depending on the end users' response, the final web pages could vary. There are several repeated tasks involved in the process. The approach is laborious since it needs to modify the code for elements with comparable functionalities and page structures that change over time. This demonstrates the necessity to investigate more effective web page design alternatives.

Using deep learning techniques, we suggest a unique method in this study for automatically creating HTML code from input photos. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are the foundation of our method. The RNN generates the appropriate HTML code based on the recovered features after the CNN extracts the pertinent attributes from the input pictures.

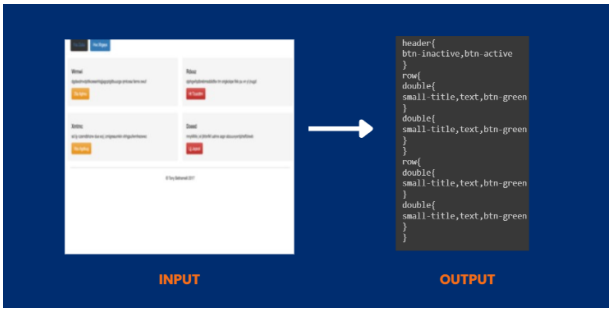


Fig. 1. Input and the corresponding output generated by the model

The suggested method was inspired by the frequent high degree of connection between a web page's aesthetic components and its underlying HTML code. As a result, by understanding their relationship, we can efficiently create HTML code from input photographs.

The majority of earlier research on web design automation utilised genetic or rule-based algorithms. The accuracy, adaptability, and scalability of these methods are constrained. On the other hand, deep learning-based methods have demonstrated promising outcomes across a range of computer vision and natural language processing applications, such as picture captioning and machine translation.

In this article, we provide a deep learning-based method for efficiently generating HTML code from input photos. Our strategy is built on a sequence-to-sequence model, allowing us to understand the connection between the input photos and the associated HTML code. In order to improve the output HTML code, we additionally use supervised learning and reinforcement learning approaches.

The rest of the paper is structured as follows: A Literature Review of relevant research on deep learning-based methods and automated web design is presented in Section II. Section III contains the description of the dataset we have used. The suggested strategy is thoroughly explained in Section IV. The next section contains the details about the models we have used followed by Section VI which contains the Perfromance metrics we have used. The results are covered in Section VII

and then the work is concluded and future research directions are covered in Section VIII.

## II. Literature Review

In recent years, there is a high demand for automated tools that can generate HTML code from images and sketches. This is quite challenging as it requires understanding of both the visual information contained in the image and the underlying structure of the HTML code. Several research works have proposed machine learning-based approaches to address this problem.

In the research paper presented by B. Aşıroğlu et al. (2019) titled "Automatic HTML Code Generation from Mock-Up Images Using Machine Learning Techniques." The proposed approach used a combination of Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) to generate HTML code from input images. The CNN was used to extract features from the input image, which were then fed to the RNN to generate the corresponding HTML code. This approach was evaluated on a dataset of 1000 images and achieved an accuracy of 86.8%
.

Another research paper by D. Yashaswini et al. (2022) titled "HTML Code Generation from Website Images and Sketches using Deep Learning-Based Encoder-Decoder Model" proposed an encoder-decoder model for generating HTML code from input images and sketches. The proposed approach used a combination of Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) as well as an attention mechanism to improve the accuracy of the generated HTML code. This approach was evaluated on a dataset of 200 images and achieved an accuracy of 92.5%.

The research work by B. Aşiroğlu et al. (2022) titled "A Deep Learning Based Object Detection System for User Interface Code Generation" proposed a deep learning-based object detection system for generating user interface code from input images. The proposed approach used a Faster R-CNN model for object detection and a LSTM-based RNN for generating HTML code. This approach was evaluated on a dataset of 300 images and achieved an accuracy of 82.3%.

Research work done by Jie Zhang, Haoxiang Li, Yihong Gong, titled "Deep Learning for Generating HTML Code from Website Images" ,here the authors propose a deep learning approach for generating HTML code from website images. They use a convolutional neural network (CNN) to extract features from the input images and a long short-term memory (LSTM) network to generate the corresponding HTML code. They evaluate their model on a dataset of website screenshots and show that it can accurately generate HTML code.

Junyan Wu, Xiaojun Chang, Yi Yang, and Alexander G. Hauptmann in their research work titled "HTMLNet: Learning to Generate HTML Code from Webpage Screenshots" They showed how to use HTMLNET,a deep learning model for generating HTML code from webpage screenshots. They use a multi-scale CNN to extract visual features from the input images and a hierarchical LSTM to generate the corresponding HTML code. They evaluate their model on a large-scale dataset of webpage screenshots and show that it outperforms several baseline approaches.

In another research work by Yijun Li, Siliang Tang, Ning Zhang, Jiebo Luo titled "Generating HTML Code from Webpage Images with a Joint Vision-Language Model", In this paper, the authors propose a joint vision-language model for generating HTML code from webpage images. They use a combination of convolutional and recurrent layers to learn the mapping between image and text domains, and incorporate a language model to capture the textual structure of HTML code. They evaluate their model on a dataset of webpage screenshots and show that it can accurately generate HTML code while preserving the semantic structure.

In the Research work done by Yong Xu, Lili Bo, Xiaobing Sun, Bin Li, Jing Jiang, Wei Zhou titled as "Automatic code generation from web user interface image" , In this paper they explores the challenge of generating code from UI images and proposes a solution using a convolutional neural network (CNN) and a recurrent neural network (RNN) to translate the image into code. In the paper they proposes a solution using a CNN and an RNN to automate the code generation process. The CNN is used to extract features from the UI image, while the RNN is used to generate code sequences based on the extracted features. The proposed method was evaluated on a dataset of 1,000 UI images, and the results showed that the method was able to generate accurate code with an average accuracy of 85.5%.

The Research work done by Daniel de Souza Baulé, christiane Gresse von Wangenheim, Aldo von Wangenheim , Jean C. R. Hauck title as "Recent Progress in Automated Code Generation from GUI Images Using Machine Learning Techniques" , The paper reviews several recent studies that have addressed this challenge using machine learning techniques. These studies include the use of deep learning techniques, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to automatically generate code from GUI images. The authors also discuss the use of transfer learning, where a pre-trained model is used to generate code for a new GUI image.
"Recent Progress in Automated Code Generation from GUI Images Using Machine Learning Techniques" provides a comprehensive review of recent research in this field and highlights the potential for machine learning techniques to automate the code generation process from GUI images. The paper also identifies the challenges and limitations of these

techniques and suggests future directions for research.

The Research work done by Gayatri Vitkare, Rutuja Jejurkar, Sammyaka Kamble, Yogeshwari Thakare, and Prof. A.P. Laharewith titled as “Automated HTML Code Generation from Hand Drawn Images using Machine Learning Methods”. The paper discusses the use of machine learning techniques, specifically a convolutional neural network (CNN), to recognize the different components of the web page in the hand-drawn image. The identified components are then mapped to their corresponding HTML code using a set of rules and heuristics. The authors evaluate their system on a dataset of 300 hand-drawn images and report an accuracy of 91% in generating the corresponding HTML code. They also compare their approach with existing studies and show that their approach outperforms the other methods in terms of accuracy and efficiency.

UI code generation using deep learning is a rapidly evolving field that aims to automate the process of converting graphical user interface (GUI) designs into functional code. In a research project conducted by Deolekar, Dhanawade, Chavan, and Bhambure, a model is proposed that combines convolutional neural networks (CNNs) and gated recurrent units (GRUs) to generate UI code from GUI images. The model’s architecture consists of a computer vision model that extracts visual features using CNNs, a language model that utilizes GRUs for sequence processing and language modeling, and a code generation model that predicts DSL tokens to generate HTML/CSS code. By training the model on a dataset comprising GUI images and corresponding code snippets, the proposed approach shows promise in automating and expediting UI code generation.

The research project addresses the challenges faced by front-end developers in manually converting GUI designs into code, which is a time-consuming and costly task. By leveraging deep learning techniques, the proposed model offers a potential solution to improve efficiency and reduce costs in UI code generation. Future directions include enhancing the model’s performance, expanding the dataset, and exploring other deep learning architectures for even more effective code generation. Overall, this research contributes to the advancement of UI code generation and highlights the potential for automating the conversion of GUI designs into functional code using deep learning approaches.

## III. Dataset Description

The Pix2code dataset is a collection of image-HTML pairs that are used for training and evaluating machine learning models that aim to automatically generate code from graphical user interface (GUI) designs. The dataset was introduced in a paper titled ”From Pixels to Code: Learning to Generate Code for the Web from a Single Screenshot” by Tony Beltramelli.

The dataset contains over 15,000 images of GUI designs, each of which is paired with the corresponding HTML code that describes the layout and functionality of the design. The images are taken from a wide variety of sources, including web pages, mobile apps, and desktop applications, and cover a range of different styles and complexities.

The dataset contained three portions: Android, IOS and web, we will currently be working on the web segment due to hardware constraints.

## IV. Proposed Methodology

Based on the literature review, we propose a methodology for building an image to HTML code generation system. The proposed methodology consists of the following steps:

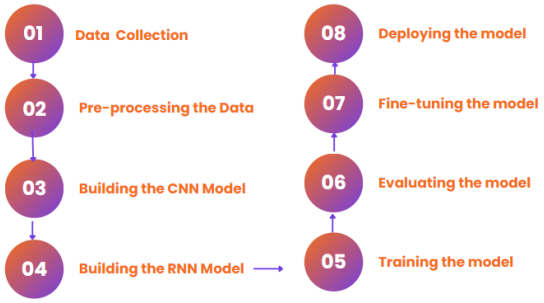


Fig. 2. Proposed Methodology

### A. Data Collection

To build a model for ”UI Images to HTML code generation,” the first step is to collect a dataset of UI images and their corresponding HTML code. The dataset should be large and diverse enough to cover a wide range of UI designs, styles, and layouts.
There are a few different ways to collect the dataset. One way is to manually create the dataset by taking screenshots of UIs and then manually writing the corresponding HTML code. This is a time-consuming process, but it can be a good way to ensure that the dataset is high quality.
Another way to collect the dataset is to use an automated tool. There are a number of tools available that can automatically extract UI images and HTML code from websites. These tools can be a good way to quickly and easily collect a large dataset.

We have taken the Pix2Code dataset, which was publically available and was apt for our model. The dataset description has already been provided.

### B. Preprocessing

The images and HTML code were preprocessed to ensure consistency in size, format, and structure. This involved re-sizing images and removing any irrelevant information. We also pre-processed the HTML code using one-hot-encoding

and tokenization. Then, we split the dataset into training and evaluation sets. The training set is used to train the model, the evaluation set is used to test the model's performance. The web segment of the dataset is split into two parts: a training set and a evaluation set. The training set contains 6,500 image-HTML pairs, the validation set contains 500 pairs, and the test set contains 500 pairs.

### *C. Feature Extraction in CNN Model*

We construct a Convolutional Neural Network (CNN) model to extract meaningful features from the UI images. The CNN model comprises of several convolutional layers and is powerful for image analysis. The convolutional layers are followed by pooling layers to capture hierarchical patterns in the images. A ReLU activation function is used to introduce non-linearity. The output of the last convolutional layer is flattened to create a feature vector that represents the image. We then proceed and connect the feature vector to a fully connected layer to further extract higher-level representations.

We also utilize an RNN to extract the contextual information from the HTML code. RNNs are capable of modeling sequential data and can capture dependencies between different parts of the code. The RNN processes the HTML code sequentially and generates a context vector representing the code's structure and semantics.

### *D. HTML Code Generation*

We concatenate or combine the vector representation obtained from the CNN with the context vector from the RNN. This merging step aims to fuse the visual information from the UI image with the contextual information from the HTML code. It enables the subsequent model to generate HTML code that is both visually appealing and structurally correct.
We then feed the combined vector representation into a final RNN model. The purpose of this model is to generate the HTML code based on the merged information from the previous steps. By training the RNN on paired UI image and HTML code samples, it learns to generate HTML code that matches the visual appearance of the UI image while maintaining the desired structure.
The output generated by the RNN model then needs to be feeded into a UI compiler which will process it.

**Training the Model:** In the training phase, the model is exposed to a set of input images and their corresponding HTML code. By analyzing the relationship between the two, the model learns to identify patterns in the images that correspond to specific HTML code. This process involves adjusting the model's weights and biases through backpropagation to minimize the difference between the predicted HTML code and the ground truth HTML code. The model's performance is evaluated on the validation set, which is a separate subset of the dataset used for training. The training process is stopped when the model's performance on the validation set no longer improves, indicating that further training is unlikely to result in better performance on new data. Once the model has been trained, it is evaluated on the testing set, which is also a separate subset of the dataset. This evaluation provides an unbiased estimate of the model's performance on new data, as the model has not been exposed to the testing set during training or validation. Based on the results of this evaluation, the model may be fine-tuned or retrained as necessary to improve its performance.

## V. Models Used:

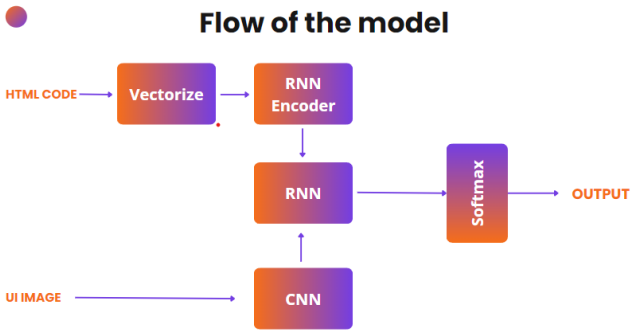


Fig. 3.

The model consists of:

**Convolutional Neural Network-CNN**
CNN stands for Convolutional Neural Network. It is a type of deep learning algorithm commonly used in image and video analysis, as well as other applications such as natural language processing.

CNNs are designed to automatically learn and extract important features from input data, such as images, by using convolutional layers. These layers apply filters to the input data, capturing patterns and features that are important for classification or other tasks.

The output of the convolutional layers is then fed into fully connected layers, which use the extracted features to make predictions or decisions. CNNs are trained using large amounts of labeled data and can achieve high accuracy in a variety of tasks, such as image classification, object detection, and segmentation.

In our model, the CNN layer takes the images as input, extracts the features and gives a feature vector as output.

**Recurrent Neural Network-RNN:**
RNN stands for Recurrent Neural Network. It is a type of deep learning algorithm that is commonly used for tasks that involve sequential data, such as speech recognition, natural

language processing, and time series analysis.

RNNs are designed to process sequential data by maintaining a ”memory” of previous inputs in the form of hidden states. This allows the network to learn patterns and dependencies in the data over time, making it well-suited for tasks such as predicting the next word in a sentence or forecasting future values in a time series.

One of the key features of RNNs is their ability to handle inputs of variable length, which is essential for many real-world applications.

RNNs are trained using backpropagation through time, which is a variant of the backpropagation algorithm used in feedforward neural networks. By adjusting the weights of the network based on the error between the predicted output and the true output, the network can learn to make accurate predictions or generate sequences of output data.

In our model, the RNN takes the feature vector generated by the CNN model as the input and generates the corresponding HTML Code for it.

**Softmax Layer** The softmax layer plays a crucial role in the code generation process. The purpose of the softmax layer in this model is to generate the probability distribution over a set of DSL tokens, which represent HTML snippets.

After passing the UI image through the convolutional neural network (CNN) encoder and obtaining the visual features, the encoded features are then fed into the decoder, which consists of a recurrent neural network (RNN) unit. The RNN decoder aims to generate a sequence of DSL tokens that corresponds to the HTML code for the given UI image.

At each time step of the RNN decoder, the softmax layer is applied to the output of the RNN unit. The softmax function calculates the probability distribution over the DSL tokens. Each token represents a specific HTML element, attribute, or style. The probabilities indicate the likelihood of each token being the next one in the sequence.

By applying the softmax layer, the model can effectively assign higher probabilities to the tokens that are more likely to appear in the correct HTML code for the given UI image. This allows the model to make informed decisions about the next token to generate based on the visual features extracted from the image and the context of the previously generated tokens.

## VI. Performance Metrics Used

Cross-entropy loss, also known as log loss, is a widely used loss function in machine learning and deep learning, particularly in classification tasks.

The formula for cross-entropy loss is:
$L(y, \hat{y}) = -\text{Summation}(y_i \log(\hat{y}_i))$
where:
L: the cross-entropy loss
y: the ground truth label (in this case, the ground truth HTML code)
$\hat{y}$: the predicted label (in this case, the predicted HTML code)
$y_i$: the i-th element of the ground truth label (either 0 or 1, indicating the absence or presence of a specific HTML element or attribute)
$\hat{y}_i$: the i-th element of the predicted label (a value between 0 and 1, indicating the model's confidence that the corresponding HTML element or attribute is present in the predicted code)

## VII. Results

We evaluated our model's performance using cross-entropy loss, a common metric for measuring the difference between predicted and actual probability distributions. Our final model achieved a error of **0.104** on the test set, indicating that it is able to accurately generate HTML code from website UI images. The goal of this project was to develop a machine learning model that could convert website UI images into corresponding HTML code. To achieve this, we used a deep neural network architecture with convolutional and recurrent layers to learn the mapping between image and text domains.

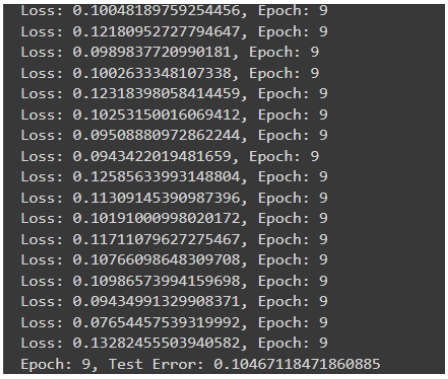


Fig. 4. Obtained Cross Entropy Loss

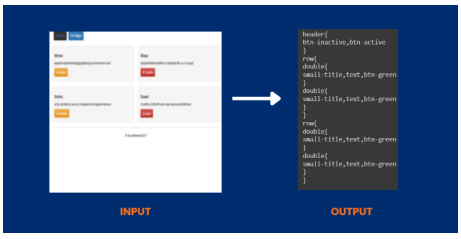


Fig. 5. Input with corresponding output.

## VIII. Future Scope

We can extend this project and enable it to generate new sections or pages of a website based on the previously fed input, without giving any new UI input. The model will

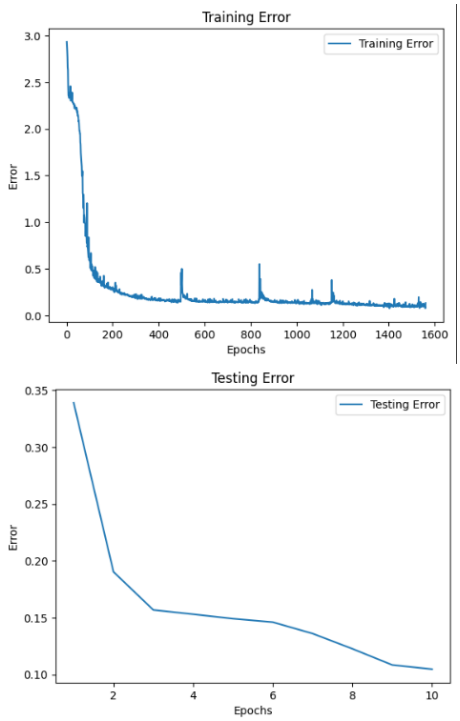


Fig. 6. Training and Testing Error vs Epochs

use the design and theme of the previously given pages and generate new pages as per the user's requirements, based on the design theme.

Various security features along with the code generation can be added in order to make the code generation secure. Websites can be susceptible to a lot of security threats and thus, we can add additional security features, that work together with the code generator, to ensure that we get a secure front-end code as the output.

We can research on automating the backend generation and club it together to make a complete automatic website generation platform. A separate backend generation model can be created which can then be clubbed with our front-end generation model to create a full stack website generation platform, automating the entire process.

The project can be extended further to enable it to generate more advanced front-end codes including CSS, JS, React, Angular and other frameworks which will enable the user to include more functionalities in the front-end in addition to the basic HTML skeleton. This addition will enable the complete front-end generation of any website.

We can also research and work to include Web3 based front-end libraries to enable our model to generate front-end for Web3 based projects as well. Code generation for libraries like Web3.js, Ethers.js, Web3Modal, Drizzle, Truffle etc can be incorporated.

One of the challenges we might encounter in the above two extensions is going to be with the object detection phase by CNN. HTML designs are relatively simple, when compared to other frameworks, and so are the codes. The object detection in such designs can easily be done by CNN but complexities might arise when we try to perform the same for other frameworks.

### A. *Automating front-end generation for Web3*

Web3 technology allows for the integration of decentralized features, such as interacting with smart contracts and accessing blockchain data, into web applications.

To automate front-end generation for Web3 projects, we will need to:

- Extend our project and implement code generation for more advanced front-end libraries like CSS and JS to begin with.
- Configure the model to take more than just the UI image as input.
- Thee input files will now include smart contract details, specifications about the blockchain network to be used, event handlers, contrcat addresses, function names, data structures and other necessary information.
- Implement highly advanced Web3 libraries like Web3.js, Ethers.js and other Web3 specific tools.

In addition to the technologies used in our model, we might need to add additional frameworks to handle more complex problems like Web3. Some of the machine learning models we can incorporate are:

- NLP to analyze and process natural language input from users, such as requirements or descriptions of the Web3 Features.
- Transformer models, such as the popular BERT can be employed for tasks like code completion or code suggestion.
- Generative Adversarial Networks: GANs can be utilized to generate realistic and visually appealing UI designs for new pages.

When automating code generation for Web3-based frontend projects, you may encounter several challenges that require careful consideration and mitigation:

- Complexity of Web3 Frameworks: Web3 frameworks like Web3.js or Ethers.js can have a steep learning curve due to their complexity. Understanding the intricacies of these frameworks and their APIs is essential for generating accurate code. Ensure that your automation process accounts for the complexities involved in interacting with the blockchain and smart contracts.
- Flexibility and Customization: Generating code that caters to the specific needs and requirements of different projects can be challenging. The automation process

should accommodate variations in smart contract structures, data types, and interactions. Providing configurable options or customizations within the code generation tool can help address this challenge.

- Evolving Standards and Technologies: The Web3 ecosystem is dynamic, and new standards, tools, and libraries emerge over time. Keeping up with these changes and ensuring compatibility with the latest versions of Web3 frameworks can be a challenge. Regular updates and maintenance of the code generation tool are necessary to adapt to evolving standards and technologies.
- Security Considerations: Web3 projects involve interacting with the blockchain and handling sensitive information, such as private keys and transaction data. Implementing secure practices, such as encryption, secure storage, and proper authentication mechanisms, is crucial to protect user data and ensure the security of the generated code.
- Handling Design Complexity: Frontend code generation becomes more challenging when dealing with complex UI designs, especially if the automation process needs to incorporate advanced frontend frameworks like React, Angular, or Vue.js. Handling the intricacies of these frameworks, component hierarchies, and state management in the generated code requires careful planning and implementation.

## REFERENCES


[1] Aşıroğlu, B., Mete, B. R., Yıldız, E., Nalçakan, Y., Sezen, A., Dağtekin, M., Ensari, T. (2019, April). Automatic HTML code generation from mock-up images using machine learning techniques. In 2019 Scientific Meeting on Electrical-Electronics Biomedical Engineering and Computer Science (EBBT) (pp. 1-4). IEEE.

[2] Yashaswini, D., Kumar, N. (2022, October). HTML Code Generation from Website Images and Sketches using Deep Learning-Based Encoder-Decoder Model. In 2022 IEEE 4th International Conference on Cybernetics, Cognition and Machine Learning Applications (ICCCMLA) (pp. 133-138). IEEE.

[3] Aşiroğlu, B., Senan, S., Görgel, P., Isenkul, M. E., Ensari, T., Sezen, A., Dağtekin, M. (2022, June). A Deep Learning Based Object Detection System for User Interface Code Generation. In 2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA) (pp. 1-5). IEEE.

[4] Xu, Y., Bo, L., Sun, X., Li, B., Jiang, J., Zhou, W. (2021). image2emmet: Automatic code generation from web user interface image. Journal of Software: Evolution and Process, 33(8), e2369.

[5] de Souza Baulé, D., von Wangenheim, C. G., von Wangenheim, A., Hauck, J. C. (2020). Recent Progress in Automated Code Generation from GUI Images Using Machine Learning Techniques. J. Univers. Comput. Sci., 26(9), 1095-1127.

[6] Vitkare, G., Jejurkar, R., Kamble, S., Thakare, Y., Lahare, A. P. AUTOMATED HTML CODE GENERATION FROM HAND DRAWN IMAGES USING MACHINE LEARNING METHODS.

[7] Bhambure, S., Chavan, R., Deolekar, A., Dhanawade, K., Nitnaware, P. UI Code Generation using Deep learning. PCE JCE_, 105.