**PEN2CODE – AI GENERATED WEBPAGES FROM HAND-DRAWN MOCKUPS**



**PROJECT SUPERVISOR**

Mr. Syed Zain Ul-Hassan

**PROJECT CO-SUPERVISOR**

Ms. Kaamla Salman

**PROJECT TEAM**

Saman Khan K190354

Anusha Saad K190281

Hermain Qadir K191517

**SUBMISSION DATE**

May 02, 2023

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES KARACHI CAMPUS

**Table of Contents**

[Declaration of Authorship](#_vk0avx3xnadg) 3

[Document Information](#_dscxjusrr8om) 4

[Abstract](#_c23l4b9zlkwp) 5

[**This page was intentionally left blank**](#_8yz18jt4nrea) **6**

[**1.0 Introduction**](#_o282gn34i5v9) **7**

[1.1 Purpose of Document](#_n078l6ywacma) 7

[1.2 Intended Audience](#_mphlogj4afc8) 7

[1.3 Scope](#_5llcg0r5b57a) 7

[1.4 Need for Product](#_qenuyjpapy9x) 8

[1.5 Benefits to Users](#_rpfc0d1lk8q6) 8

[**2.0 Related Wrok**](#_pxb3qq2nr3l2)  **9**

[2.1 Heuristic Methodologies](#_yq4wpo4ploz5) 9

[2.2 End-to-End Methodologies](#_4o5qd444d7t7) 10

[2.3 Object Detection Methodologies](#_fyto25ao3e2s) 12

[2.4 Data-Driven Methodologies](#_2ppw8e8s7juy) 13

[2.5 Structural-Information Based Methodologies 13](#_cegoj9e9znnk)

3.0 Proposed Approach 18

[**4.0 Implementation**](#_cey8zwjbosiy) **19**

[4.1 Datasets](#_42zvnpqtrn7m) 19

[4.2 Model](#_xi7ckvz7tbnq) Training 20

[4.2.1 Preprocessing of Images](#_mqth4d9t7csm) 20

[4.2.2 CNN Model](#_dm3p02awgsua) 21

[4.2.3 GRU](#_ce0y1lumrvfi) 23

4.3 Web Application Developmentod 25

4.3.1 Front-End Development 25

4.3.2 Back-End Development 26

[**5.0 Model Testing and Evaluation**](#_yd1xhi9gvqhr) **27**

5.1 Initial Testing of Web Application Sketches 27

[**6.0 Results**](#_gv72jzioxs2c) **and Discussion 29**

[**7.0 Conclusion**](#_4tvjxby841ix)**s and Future Work 29**

[**8.0 References**](#_kxcq0ccozvis) **31**

### Declaration of Authorship

We, Saman/Anusha/Hermain, declare that this report titled, Pen2Code – AI Generated Webpages from Hand-drawn Mockups and the work presented in it are our own. We confirm that:

* This work was done wholly or mainly while in candidature for a degree at this University.
* Where we have consulted the published work of others, this is always clearly attributed.
* Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this report is entirely our own work.
* We have acknowledged all main sources of help.
* Where the report is based on work done by us jointly with others, we have made clear exactly what was done by others and what we have contributed ourselves.

### 

### Document Information

|  |  |
| --- | --- |
| **Category** | **Information** |
| Customer | FAST - NUCES |
| Project Title | Pen2Code – AI Genrated Webpages from Hand-drawn Mockups |
| Document | FYP-II Report |
| Status | Completed |
| Authors | Saman Khan, Anusha Saad, Hermain Qadir |
| Approvers | Mr. Zain-ul-Hassan, Ms. Kaamla Salman |
| Issue Date | May 02, 2023 |

### Abstract

A software development life cycle (SDLC) usually begins from UI designing. The UI designer designs sample webpages using tools like Figma. The same are approved by clients, albeit with changes. The front-end developer is then responsible to convert the designs to functional webpages. The process of front-end development takes up to two months in any project. At the end, the result is not the same as the one envisioned by the client. This follows with loops of discussion between the stakeholders, UI designer, and developer. The product does not meet the requirements of clients.

SDLC is needlessly stretched in front-end development phase. The tasks become redundant. With advancement in AI, it is used to automate repetitive tasks, several solutions have been made to automate front-end development. Our solution aims to make front-end development a task that can be done within minutes, so the time saved can be used elsewhere, like API integration and software testing. The user will draw a mockup of webpage (can be a webpage or a mobile screen), upload it on the website, and the system will output a HTML and CSS file for the same. The user will be able to further to apply themes to the generated webpages and choose images from the suggested image. Our product bridges the existing gap by CNN architecture Resnet34 and Resnet101 on the backend.

## *This page was intentionally left blank*

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 1.0 Introduction

### 1.1 Purpose of Document

This progress report provides a detailed overview of the work of our final year project – I. Moreover, it gives a complete description of the work that we have done in the duration of FYP-I from September to January.

### 1.2 Intended Audience

The audience of this document are supervisors as this project is being implemented under their guidance. The current and future developers working on automating aspects of software development.

### 1.3 Scope

The scope and main functionalities of this project is the following. The application which will develop in this project will provide the user-friendly interface that will be fast, efficient, and easy to use. This application will allow users to upload images of hand-drawn mockups. The backend mechanism of the system will detect and extract features from the images to classify particular HTML elements like text box, button, text label etc. The extracted elements/features will be tested on an already trained model and then recognized. The identified elements will be converted into a hierarchy to produce a HTML document (with CSS styling). The .html file will be visible on the user’s screen which he/she can then download or modify by selecting from the given themes.

### 1.4 Motivation

In Software Development Life Cycle (SDLC), the prototype design being converted into UI code takes a lot of time because usually the website design does not align with the stakeholder’s vision. Not only does this waste the time of UI developers, it also makes the tasks redundant. The confusion created in exchanging designs back-and-forth makes front-end development a very tedious task. Not only does this waste time, it can also result in loss of potential clients. With advancements in the field of Artificial Intelligence, several tasks are no longer done manually. The same cannot be said about web development. Our project has primary goal of reducing loop cycles between designers and developers and save developers from redundant work.

### 1.5 Benefits to Users

Pen2Code is a product that will provide significant assistance to developers. It will automate initial process of development to some extent; hence, giving developers the time and leniency to focus on more important and detail-oriented aspects of software development, such as testing and building database. It will reduce loop cycles that exist between product manager, UI/UX designer, and developer and ensure that the mobile/web application is coded the right way from the get go. It will also pave way for developers to code application that match stakeholder’s vision without having to go back-and-forth and correcting the code a million times.

Another crucial benefit of Pen2Code is that it can also be used by someone who does not know how to code. This eases the responsibility on front-end developers so they can focus more on adding functionalities and complex features that can be not be automated otherwise. Anyone looking to get a basic web page built can use Pen2Code easily.

Pen2code will be an easy-to-use solution. All a user has to do is capture a picture of hand-drawn sketch from his or her mobile phone, upload image on the web application. The system will generate a corresponding HTML file that the user can download and then use.

**2.0 Related Work**

Artificial Intelligence has automated many tasks across various industries. One such example is the use of AI in automating development of websites or even mobile applications. The field of automating development is not new and much work has been done in it. In this field of study, several techniques and algorithms have been applied and those techniques can be categorized into five distinct approaches. The following sections detail thoroughly on the methodologies employed by researchers. Subsection 2.1 details on methodology that works on algorithms and heuristics created to segment different UI elements. Subsection 2.2 lists the work done by using deep learning; where images are fed to a neural network and the output is the source code. Subsection 2.3 lists all techniques that used object detection to determine the bounding box the UI elements. Lastly, subsection 2.4 discusses the work done in this field that depends on the large amount of data to generate platform-specific code.

**2.1 Heuristic Based Methodologies**

Heuristic-based methodologies identify HTML elements from an already-designed user interface. This technique performs segmentation iteratively to detect and extract elements until all the useful elements that the website is made up of are identified. The identified elements are then used to generate the HTML code.

In [1], information is gathered in a top-down fashion. Elements are extracted by edge detection using Canny and limits are removed by applying median blur. However, this degrades the text to some extent, so morphological operations are applied to dilate the pixels and therefore make the text clearer than before. Contour detection is used to compute the bounding box for each element. The elements are extracted through pre-processing and outline detection; and then segmented into separate images. These images are classified by a UI predictor. The combined results from predictor, bounding box and features are used to build a JSON format file that stores the hierarchy. On the other hand, Huang et al. [2] performs more or less the same steps but in a bottom-to-top fashion. [2] uses Random Forest for the classification. The leaf nodes are classified first using Random Forest and the inner nodes then proceed in bottom-up fashion to add children tags or properties accordingly. In case of any missing element, heuristic methods are used to refine the hierarchy of HTML elements. At the end, CSS is also used to reflect the visual properties.

In [3] and [4], OCR is used along with the methodologies above. First, OCR detects all the textual components from the image. This does result in false positives which are filtered using heuristics. From the OCR detected text, bounding boxes are determined to further figure out the font size and family. Meanwhile, simultaneously, computer vision techniques are applied to prepare the hierarchy of all elements. At the end, OCR and CV combined produces results that are then used in Android project to produce the android studio code output [3]. OCR analysis detects keywords which are further used to detect elements. For instance, a rectangle with a verb like ‘click here’ or ‘submit’ suggests that it is a button element [4].

**2.2 End to End Methodologies**

The techniques in this section rely heavily on deep neural networks (DNNs) to transform mockup into code.

Beltramelli [5] devised a solution that takes UI pictures or design as sketches and gives the relevant output. The model has three neural networks, a Convolutional Neural Network and two LSTMs. The CNN processes the input image and outputs a corresponding vision-encoded vector of that image. The first LSTM, of 128 cells, gives an intermediate representation (DSL) language-encoded vector. The vision-encoded and language-encoded vectors pass through a second LSTM decoder (that consists of two stacks, each 512 cells long) that learns how the objects are related to their DSL tokens. Using Deep Learning, the code produced is 77 percent accurate. In [5], some DSL context is needed and a DSL token with a ‘start’ attribute has to be initialized to give a valid result. Zhu et al. [6] improves on [5] by using token LSTM. The implementation des not need an initial context and performs well without it. It performs end-to-end training by using just the UI images and DSL code. [7] works in a similar fashion; however, the vision-encoded vector is fed to an LSTM which gives an intermediate representation vector. Next, the output of the CNN model with hidden state is passed to an attention model.

Yanbin Liu, Qidi Hu, Kunxian Shu in [8] use an approach that improvises on pix2code [5], which consists of CNN and LSTM, by replacing by CNN and BiLSTM. This approach uses a stack of two BiLSTM (128 cells each). And the decoder consists of a stack of BiLSTM with 512 cells each. The approach improved the pix2code framework as the accuracy reached 85%.

Ellis et al. [9] focuses on hand-drawn sketches unlike those above. During processing, the hand-drawn image is fed to a multi-layer perceptron and an attention mechanism that filters relevant features. The rendering and original image become input to a CNN (as a two-channel image) which outputs the LaTeX code of the drawn elements. Finally, program synthesis engine structures the code to formulate a hierarchy.

Sketch2code [10] is the most recent implementation of mid-fidelity wireframes. It is developed by Microsoft AI. It utilizes ANN to normalise the wireframe, i.e. learns the association that links the wireframe to the image in dataset. After contour detection outlines the header and footer of the page, segmentation is done to detect and classify elements. At last, Domain Specific Language (DSL), JSON tree-like structure, code is generated.

Once the Input has been provided it is time to extract useful information from the wireframe, but first, the image will have to be pre-processed, which will consist of converting the image to grayscale, noise reduction and edge detection, and other techniques to make the HTML elements detection process easier in the Pattern recognition stage. This will be accomplished through various Image processing libraries in Python. In the image analysis part Python libraries such as Python Imaging Library (PIL) can be used which contains the ImageOps module with convenient image processing operations that can be used on the image [11]. OpenCV (Open Source Computer Vision Library) will be utilized to process the wireframe sketch and to detect the label, shapes and symbols and text. Tesseract OCR can be used to detect and extract text from the wireframe image and accommodate it in the HTML code output.

After getting input image file in png format from user input, image is converted into the array format. The 3\*3 rectangle kernel is created outside the object to detect it by applying counter detection algorithm so components are detected. Afterwards the catched elements are cropped and then it transferred to next stage which is the convolution neural network model. CNN model has several layer for filtering such as convolution layers with 4\*4 kernel & then there is max pooling process with 2\*2 kernels which is used for the extraction. Factorization is done by using BILSTM layer for the correlation of extraction. After that pool full connected layer and last one is the dropout layer, all the layers working together to train the model. This whole process is done by directly importing keras library [12].

Convolutional Neural Network helps to analyze visuals. In the Computer Vision model, CNN is used to perform an unsupervised feature which is learned by mapping an input image to a learned fixed-length vector, thus this acts as an encoder. Pixel values of an image are normalized before giving to CNN, hence the initial output vector is generated [13].

**2.3 Object Detection Methodologies**

Object detection algorithms are very useful in this area of research as they identify the bounding boxes of detected objects.

Suleri et al. [14] worked on a solution to detect the drawn mockup’s elements using object detection. The low-fidelity level wireframes depict the sketches in their pure form i.e. drawn by a pen or pencil. In the first stage, a dataset was generated manually. By involving a total of 350 participants consisting of front-end developers and UI/UX designers 5,096 sketches were produced that made use of 19 UI elements. As a result, an annotated dataset was generated with 125,000 images. The system in [14] had three distinguished interactions; low-fidelity wireframes, mid-fidelity wire-frames, and high-fidelity wireframes. The medium fidelity wireframes give the representation by an object detector, so the user can control conversion process and change any properties as desired. Lastly, in the high-fidelity wireframes, user can additionally choose themes to generate the final code.

YOLO, an already existing network, transfers learning to train on learning features of different UI elements [15]. The YOLO detects elements and gives the confidence level for each. This output is finally accumulated in a hierarchy which generates code for any target platform (could by Web, iOS etc.).

[16] utilizes a sufficiently different technique than the ones mentioned in this section. The approach has two main steps; layout detection and element recognition. UI element recognition is done by Faster R-CNN. However, layout detection was not as simple. To cater to any disconnected edges, computer vision techniques like slope filtering (for slopes that are not aligned on the x-axis or y-axis) are used. A correspondence line algorithm detects the layouts and keeps repeating it until all layout options have been exhausted. This project [17] uses simple object detection that takes a low-fidelity hand-drawn image of a web page as input. This input undergoes morphological transformations and contour detection to perform object cropping that leads to the detection of individual elements of the sketched image. By using deep learning, their algorithm identifies whether the detected element is a button, text, image, or any other component. Then, a corresponding HTML page is constructed.

**2.4 Data-Driven Methodologies**

One significant research work has been undertaken to make use of this methodology. [18] relies heavily on the data generated and used in the pipeline. Firstly, detection is performed in two ways: Once detection is done, the detected elements proceed to the classification stage. Elements are classified using a trained CNN. Lastly, the assembly stage begins. To determine UI, K-Nearest Neighbor algorithm is used where the UI is compared to a large dataset of already-extracted UI layouts. Overall, the methodology aims to find patterns from real-world mobile applications to create code which a developer would realistically aim to achieve. Additionally, it also performs some CSS styling to the code like background color, font size, font style etc.

**2.5 Structural Information Based Methodologies**

The methodologies in this section do not necessarily output a UI code. Instead, a structural analysis is done to organize the semantic and hierarchical relationship between the UI elements. Once the semantic relationship has been identified, it can be used to further work on any semantic errors spotted.

Using the DOM code, a visual representation of a mockups is created where bounding boxes of each element are colour coded. A text-box has a different colour boundary than a bounding box for image, so it essentially differentiates between a text-box and image first. Followed by this, component instances are classified by using a technique that maximizes the result using the repetitions of each component and their encapsulated instances found. The DOM from the webpage is used to built a tree of semantic analysis. In the last iteration with maximum modularization potential, the output is fed to a visual matching stage which works based on unsupervised learning. The unsupervised clustering process identifies the UI elements by cosine distances. At last, a UI Generation Phase is carried when all UI elements are identified. However, this phase only generates an intermediate result or output which can be further translated into a framework of choice [19].

This work is targeted only for mobile applications and makes use of knowledge of the hierarchical view to understand the semantic layout [20]. Heuristics are used to classify the components. The UI hierarchy infers the class ancestors of all components. Final classification is performed using a CNN, after which any anomalies present are detected [20].

The table below lists the strengths and weaknesses of each proposed solution:

|  |  |  |
| --- | --- | --- |
| **Paper #** | **Strengths/Pros** | **Weaknesses/Cons** |
| 1 | High accuracy of 90.2% but only on trained icons. | The variation in gradient negatively affects contour detection |
| 2 | The result is compared to actual label therefore accuracy is found and improved at each step. The final validation accuracy of 90.2% is achieved. | While identifying the text regions there are some cases when icons are falsely identified as text. |
| 3 | The approach identified the elements and their coordinates perfectly. | Half layers frozen due to limited memory. Less data to train |
| 4 | Removes ambiguities in UI widgets by relating UI widgets representations and their ambiguities | Only works for 9 specific UI elements |
| 5 | Provides higher accuracy in image recognition problems | Small training dataset with little variation |
| 6 | Gaussian functions help reduce the noise | The model is difficult to train |
| 7 | Context not needed as input to the network because the training strategy is more accessible. | The model is difficult to train given it’s how more complex and has several stages. |
| 8 | Better accuracy than pix2code with LSTM decoder | BiLSTM are prone to overfitting often |
| 9 | Can correct errors made by the deep network, measure similarity between drawings by use of similar high-level geometric structures, and extrapolate drawings. | Only produces LaTeX style figures; needs more work to be able to reproduce it to the theme of code-generation. |
| 10 | Colour detection happens alongside. | The wireframe generated varies from the original dataset. Some have little variation while others differ a lot from the sketch |
| 11 | Performs styling using CSS frameworks | Does not always produce the output that matches the drawn wireframe |
| 12 | High accuracy of 93% | Data trained on very few UI elements & BiLSTM is prone to overfitting |
| 13 | Language model achieves token level language modelling so techniques like word2vec that are expensive in terms of computing are eliminated. | Less accuracy than what could be achieved by instead using RNN, LSTM. |
| 14 | Objects can be detected at any given positions and produces a large dataset for future work | Low average recall of only 72.7%; can be improved further. |
| 15 | Adopts the object detection approach using DNN rather than image captioning based CNN and RNN or synthetic knowledge-based approaches which detects instances of real-world objects of a certain class. | Training data only consists of 50 sketches. |
| 16 | Unlike other studies that detect conventional GUI, this paper uses computer vision and deep running together. | There were some objects with poor accuracy and recall. |
| 17 | Computer vision techniques are utilized to process hand-drawn mock-ups, and then deep learning approaches are employed to construct the system which produced accuracy of 96% on train data. | Overfitting can be seen as accuracy produced on the test data is 96% whereas the test accuracy is only 73%. |
| 18 | Considers how websites in reality appear to be designed. | There is usage of multiple screens so developer will have to do manual work to establish a concrete navigation flow. |
| 19 | The general code can be modified in further work to produce code for a desired platform. | No platform specific code produced |
| 20 | Semantic identification can generate more robust generative models of UI. | Only works for mobile apps. UI screenshots are a necessity in inputs. |

### 3.0 Proposed Methodology

### Following the line of related work, our proposed approach is to implement the model with a custom web application with possible improvements to previous models.

### The dataset of generated websites includes DSL tokens for each website as well as the images needed to create the website. The dataset will be divided into three parts: training, validation, and testing.

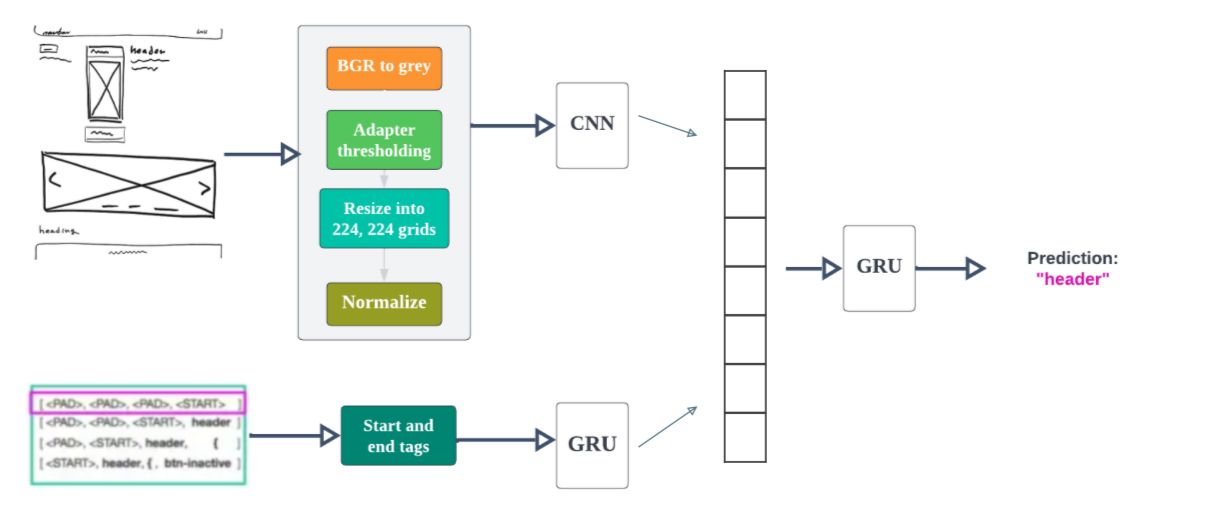
### To extract image features from the source photos, a Convolutional Neural Network (CNN) will be trained. To increase its performance, the CNN is pre-trained on a large dataset, such as ImageNet, and fine-tuned on a dataset of website images. This model will be used to encode images that will be fed into the language model.

### To encode sequences of source code tokens, a Gated Recurrent Unit (GRU) should be trained. The language model accepts DSL token sequences and uses information from the computer vision model to anticipate the next token in the series. The model will be taught using either a teacher forcing technique, in which the model's own predictions are utilized as input during training, or a sampling-based approach, in which the model's own predictions are used as input during training.

### A second GRU would be trained to decode the language model's output and predict the next token in the sequence. The result from the previous two processes is sent into this model, which generates the DSL tokens for the website. The decoder model will be trained similarly to the language model.

### At inference time, the image undergoes initial processing through the CNN network, and the initial sequence is provided as input to the language model. The model's prediction for the next token in the sequence is then added to the current input sequence and fed into the model as a new input sequence at each step. This is continued until the model predicts an END> token or the procedure reaches a predetermined token limit per document. A compiler can be used to transform the generated DSL tokens into HTML.

### After the model generates a set of predicted tokens, a compiler is used to convert the DSL tokens into HTML, which can be rendered in any browser. The HTML output can be inspected directly to determine the quality of the generated websites.



### 4.0 Implementation

**4.1 Dataset**

We have worked on two different datasets, one for web applications and other for mobile applications.

The dataset for web applications has been obtained from [21]. [21] started with an open-source dataset from [5], which consists of 1,750 screenshots of synthetically generated websites and their relevant source code. The dataset was of GUI screenshots, so it was converted to appear as hand-drawn sketches. This was done by using tools like [OpenCV](https://opencv.org/) and the [PIL library](https://pillow.readthedocs.io/en/stable/) in python to make modifications to each image such as grayscale transformations and contour detection.

Ultimately, the CSS stylesheet was modified of the original websites, performing a series of operations:

* Changed the border radius of elements on the page to curve the corners of buttons and divs
* Adjusted the thickness of borders to mimic drawn sketches, and added drop shadows
* Changed the font to one that looks like handwriting.
* Augmentation of images by adding skews, shifts, and rotations to mimic the variability in actual drawn sketches.

The dataset for mobile applications is obtained from [22]. The process of creating dataset begins by creation a DSL (domain-specific language) dictionary in which a block of code is mapped to a single token to simplify our problem. The generator creates unique random pages using the DSL dictionary and applying a set of rules to output DSL files which are then mapped to realistic looking web-pages using a DSL compiler. The resulting web-pages are rendered with a special CSS file using PhantomJS engine. Finally, simple object detection is applied on the rendered web-pages to detect all the different elements within this page, and create a matching sketch for each web-page; the sketch is generated by placing an actual hand sketch for the detected element, the sketched element is chosen randomly from a set of images provided for this element.

**4.2 Model Training**

**4.2.1 Preprocessing**

In the very stage, images are pre-processed. For pre-processing, several techniques are applied.

**Grayscaling** is the process of converting an image from other color spaces e.g. RGB, CMYK, HSV, etc. to shades of gray. It varies between complete black and complete white. Its main importance is dimension reduction for example, In RGB images there are three color channels and three dimensions while grayscale images are single-dimensional and hence that reduces model complexity for example consider training neural articles on RGB images of 10x10x3 pixels. The input layer will have 300 input nodes. On the other hand, the same neural network will need only 100 input nodes for grayscale images.

**Adaptive thresholding** is the method where the threshold value is calculated for smaller regions and therefore, there will be different threshold values for different regions. We have used OpenCV to perform Adaptive threshold operation on an image using the method *adaptiveThreshold()*. Adaptive thresholding is used to separate desirable foreground image objects from the background based on the difference in pixel intensities of each region. It produces a much cleaner segmentation result.

**Image resizing** is necessary when you need to increase or decrease the total number of pixels, whereas remapping can occur when you are correcting for lens distortion or rotating an image. Zooming refers to increase the quantity of pixels, so that when you zoom an image, you will see more detail. We have resized the image into 224x224 as we are using Resnet architecture.

**Image normalization** ensures optimal comparisons across data acquisition methods and texture instances. The normalization of pixel values (intensity) is recommended for imaging modalities that do not correspond to absolute physical quantities. Data normalization is an important step which ensures that each input parameter (pixel, in this case) has a similar data distribution. This makes convergence faster while training the network.

def resize\_img(png\_file\_path):

        img\_rgb = cv2.imread(png\_file\_path)

        img\_grey = cv2.cvtColor(img\_rgb, cv2.COLOR\_BGR2GRAY)

        img\_adapted = cv2.adaptiveThreshold(img\_grey, 255, cv2.ADAPTIVE\_THRESH\_MEAN\_C,cv2.THRESH\_BINARY, 101, 9)

        img\_stacked = np.repeat(img\_adapted[...,None],3,axis=2)

        resized = cv2.resize(img\_stacked, (224,224), interpolation=cv2.INTER\_AREA)

        bg\_img = 255 \* np.ones(shape=(224,224,3))

        bg\_img[0:224, 0:224,:] = resized

        bg\_img /= 255

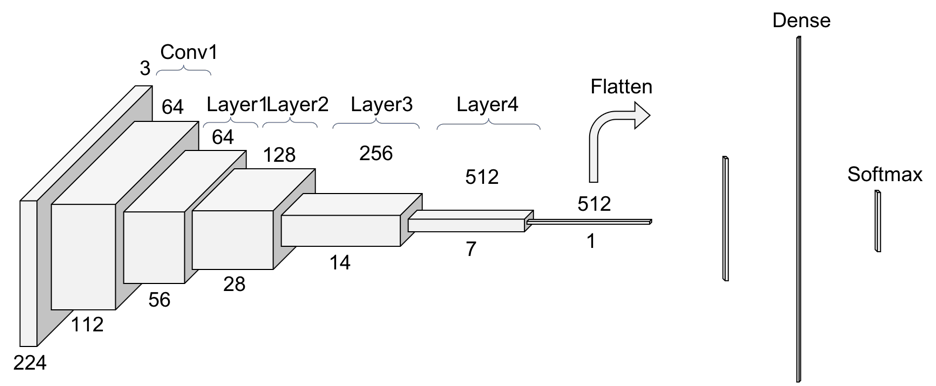
    bg\_img = np.rollaxis(bg\_img, 2, 0)

        return bg\_img

**Fig 1.0 Preprocessing code**

**4.2.2 CNN**

We have used the concept of transfer learning by using pretrained Resnet architectures. For extracting features from images of web application hand-drawn sketches, we have used resnet34. For doing the same with images of mobile applications, resnet101 architecture has been used where we have deleted the last fully-connected layer of the pre-trained model and added a few new layers to make the model fit our project requirements.

**Fig 2.0 Basic ResNet Architecture**

Resnet34 is a state-of-the-art image classification model, structured as a 34 layer convolutional neural network and defined in "Deep Residual Learning for Image Recognition". Restnet34 is pre-trained on the ImageNet dataset which contains 100,000+ images across 200 different classes.

However, RestNet is different from traditional neural networks in the sense that it takes residuals from each layer and uses them in the subsequent connected layers (similar to residual neural networks used for text prediction). With ResNets, the gradients can flow directly through the skip connections backwards from later layers to initial filters.

class EncoderCNN(nn.Module):

    def \_\_init\_\_(self, embed\_size):

        """Load the pretrained ResNet-152 and replace top fc layer."""

        super(EncoderCNN, self).\_\_init\_\_()

        resnet = models.resnet152(weights='ResNet152\_Weights.DEFAULT')

        modules = list(resnet.children())[:-1]      # delete the last fc layer.

        self.resnet = nn.Sequential(\*modules)

        self.linear = nn.Linear(resnet.fc.in\_features, embed\_size)

        self.bn = nn.BatchNorm1d(embed\_size, momentum=0.01)

        self.init\_weights()

    def init\_weights(self):

        """Initialize the weights."""

        self.linear.weight.data.normal\_(0.0, 0.02)

        self.linear.bias.data.fill\_(0)

    def forward(self, images):

        """Extract the image feature vectors."""

        features = self.resnet(images)

        features = Variable(features.data)

        features = features.view(features.size(0), -1)

        if images.shape[0] < 2:

            features = self.linear(features)

            return features

        features = self.bn(self.linear(features))

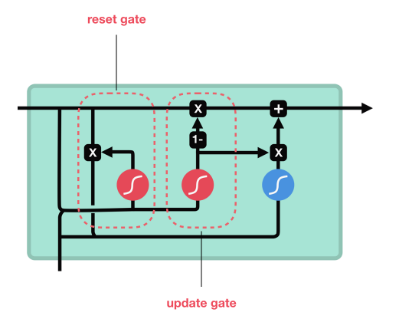
        return features # Bxembed\_size

**Fig 3.0 CNN Encoder Implementation**

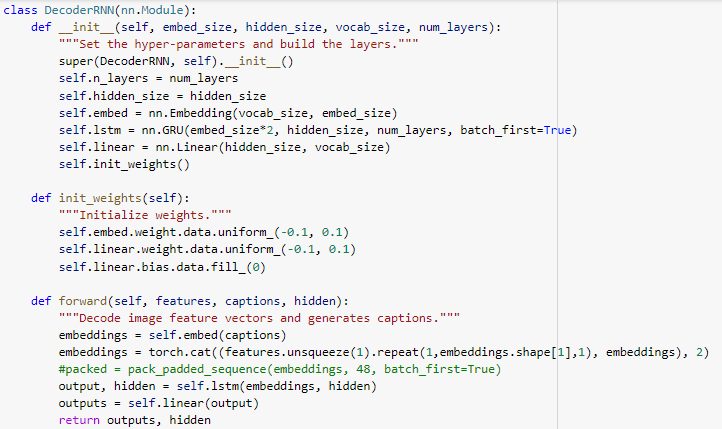
**4.2.3 GRU**

The Gated Recurrent Unit (GRU) is a type of Recurrent Neural Network (RNN) that, in certain cases, has advantages over long short term memory (LSTM). GRU uses less memory and is faster than LSTM, however, LSTM is more accurate when using datasets with longer sequences.

Also, GRUs address the vanishing gradient problem (values used to update network weights) from which vanilla recurrent neural networks suffer. If the grading shrinks over time as it back propagates, it may become too small to affect learning, thus making the neural net untrainable.

**  
Fig 3.0 GRU Architecture**

### Model training begins with features of image extracted using Resnet and text features are extracted using GRU. Followed by this, the GRU layer concatenates image features and text features and outputs a prediction.



**Fig 4.0 CNN Decoder Implementation**

**4.3 Web Application Development**

**4.3.1 Front-End Development**

We developed the front-end of the web application of Pen2Code. This involved the following tasks:

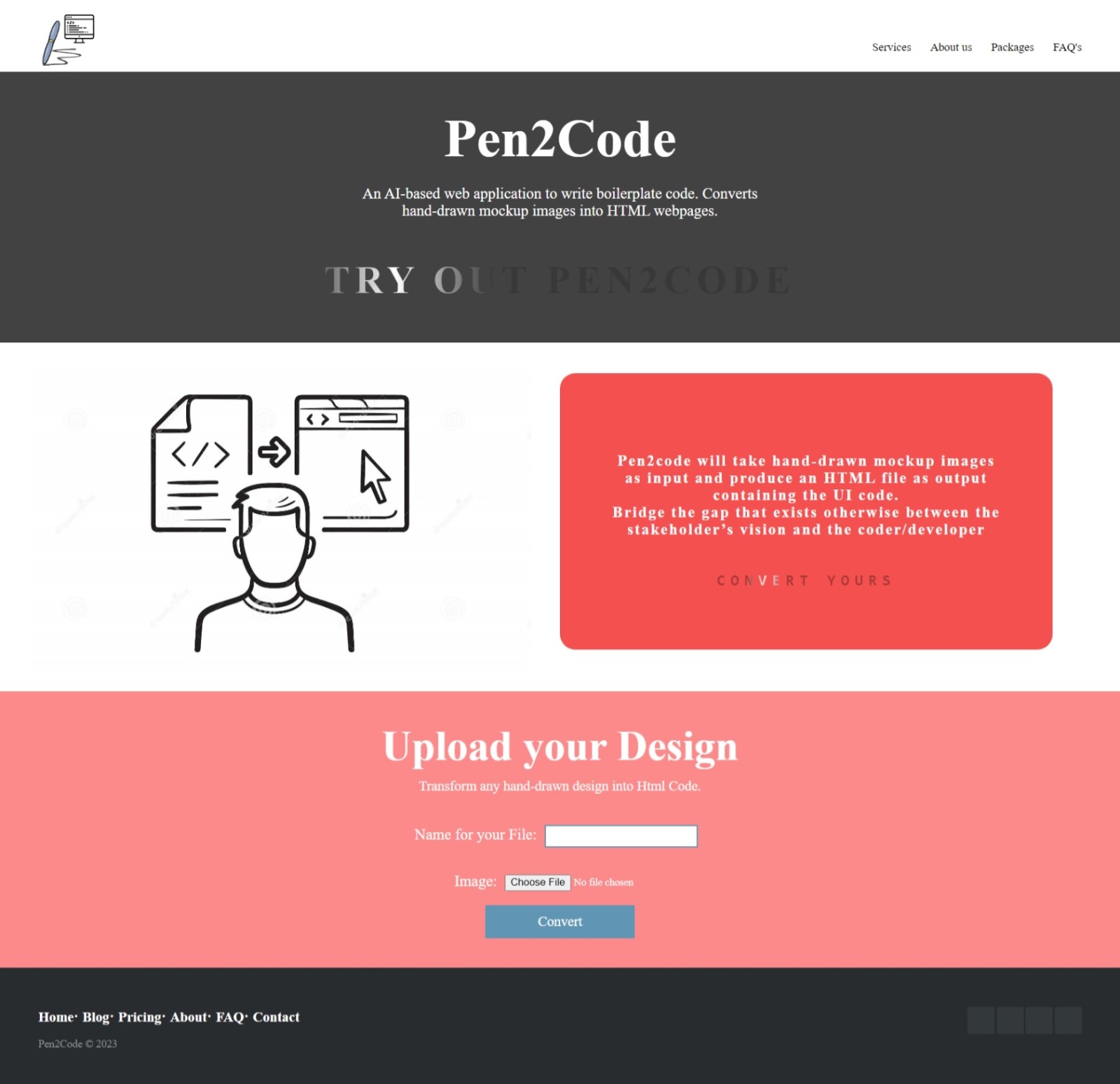
* Designing the UI/UX using HTML/CSS
* Creating the layout and structure of each webpage of the website
* Integrating various textual and visual elements

The technologies used to develop the front-end are:

* Hyper Text Markup Language
* Cascading Style Sheets
* JavaScript
* React

While developing the front-end of the website, we aimed to achieve the following goals:

1. **Cross-browser compatibility:** Different web browsers render HTML, CSS, and JavaScript in slightly different ways, which can result in inconsistent behavior across different browsers. This can lead to layout issues, broken functionality, and other unexpected behavior.
2. **Responsive design:** The front end of a website must be optimized for various screen sizes and devices, including desktops, tablets, and smartphones. Failure to properly design for different screen sizes can result in a poor user experience, with elements that are too small to read or too large to fit on the screen.
3. **Performance optimization:** A website's front end must be optimized for fast load times and smooth performance, which can be challenging when dealing with large images, videos, and other media. Poorly optimized code or assets can lead to slow load times, which can result in a high bounce rate and lost visitors.

****

**Fig 5.0 Home Page of Web Application**

**4.3.2 Back-End Development**

The second milestone of the project is the development of the back end of the web application, which is based on the Django language. The back-end is the part of the application that processes and stores data, and provides the core functionality that drives the user interface. This milestone involves designing and implementing the server-side logic, creating models for the application's data structures, and defining the endpoints for the application API. Django provides a powerful and flexible foundation for the back-end development, with features such as ORM (Object-Relational Mapping), built-in authentication, and scalable architecture. Once the back end is complete, it will be integrated with the front end to create a fully functional and responsive web application.

In Django, the rest framework is used to send pictures from the front-end to the back-end for further model processing.

In our project, there is no utilization of Database Management Systems like SQL or MongoDB because our data is 1500 .jpg images and their corresponding .GUI files. Therefore, we make use of unstructured data hence no need of database query languages.

While coding the back-end of website, we had our main focus on testing and debugging. As with any software development project, testing and debugging are critical to ensuring the quality and reliability of the back end. However, testing and debugging a back-end system can be more complex and challenging than testing a front end system, due to the intricacies of the underlying code and data structures.

### 5.0 Model Evaluation and Testing

**5.1 Initial Testing of Web Application Sketches**

In the initial testing of our model and web application, we uploaded a few hand-drawn web page mockup test images from the front-end of the application and the output was checked against the appearance of HTML file.

The proposed methodology for creating websites using a model architecture similar to image captioning received a BLEU score of 0.90, indicating that the created websites are quite close to what a person would have produced given the identical input.

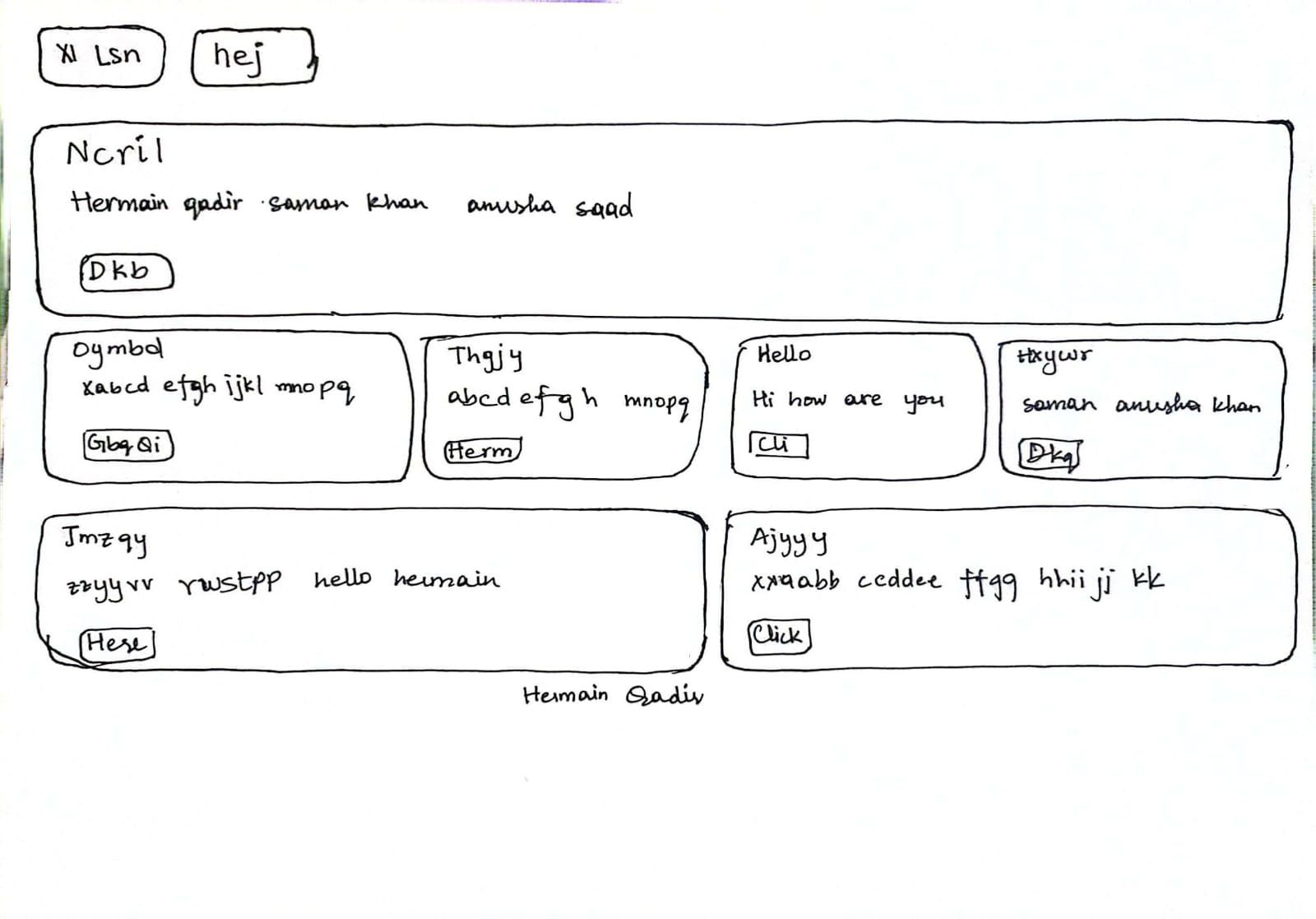
**6.0 Results and Discussion**

The proposed methodology for creating websites using a model architecture similar to image captioning received a BLEU score of 0.90, indicating that the created websites are quite close to what a person would have produced given the identical input.

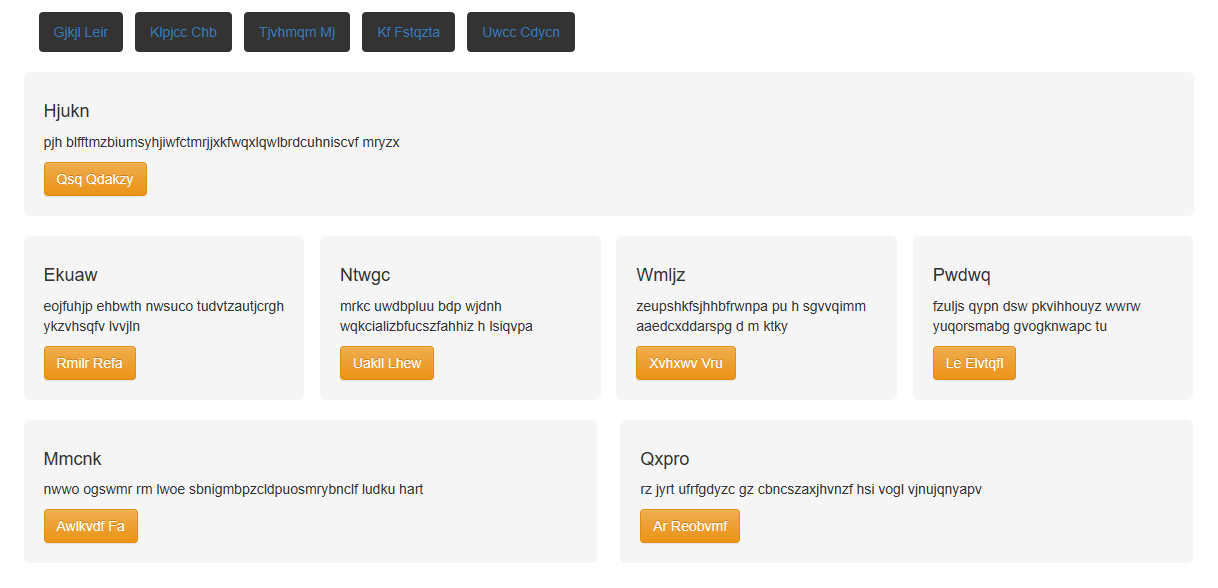
The model, however, has several drawbacks. Because the model was trained on a vocabulary of only 16 elements, it can't predict tokens that aren't in the data. To increase the model's vocabulary, one next step could be to construct further website instances employing more features such as photos, dropdown menus, and forms. In addition, there is a lot of variety in actual production websites that the current dataset does not fully cover. To overcome this, we propose scraping actual websites in order to generate a training dataset that is more representative of heterogeneity in production websites. The HTML/CSS code, as well as screenshots of the site's content, should be included in this dataset.

Overall, the suggested methodology offers a promising approach to deep learning-based website generation. More research can be conducted to improve the model and expand its vocabulary, resulting in more diverse websites that are more similar to those found in the real world.

Input:

****

Output

****

**Fig. 4.0**

**7.0 Conclusions and Future Work**

We developed a way for creating websites using a model architecture comparable to picture captioning in this project. The model consists of a computer vision model that extracts image features from source images using a Convolutional Neural Network (CNN), a language model that encodes sequences of source code tokens, and a decoder model (also a GRU) that predicts the next token in the sequence. A compiler is then used to translate the resulting DSL tokens to HTML.

The proposed methodology received a BLEU score of 0.90, suggesting that the created webpages are reasonably similar to what a human would have produced given the identical input. However, the model has some limitations, such as a limited element vocabulary and a lack of variation in the dataset.

### Future work could concentrate on expanding the model's vocabulary to include more elements such as images, dropdown menus, and forms, as well as generating a more diverse dataset that reflects the variation in production websites. Furthermore, employing a Generative Adversarial Network (GAN) to generate realistic-looking drawn website images can aid in improving the quality of the generated websites.

### Overall, the suggested methodology offers a promising approach to deep learning-based website generation. This approach, with additional study and improvements, can be applied to a wide range of website creation activities, such as website prototyping and website personalization, and contribute to the advancement of the area of web development.

### 8.0 References

|  |  |
| --- | --- |
| 1 | R. Huang, Y. Long, and Xiangping Chen. Automaticly generating web page from a mockup. Proceedings of the International Conference on Software Engineeringand Knowledge Engineering, SEKE, 2016-Janua, 2016. |
| 2 | Saad Hassan, Manan Arya, Ujjwal Bhardwaj, and Silica Kole. Extraction and Classification of User Interface Components from an Image. International Journal ofPure and Applied Mathematics, 118(24):1–16, 2018. |
| 3 | T A Nguyen and C Csallner. Reverse Engineering Mobile Application User In terfaces with REMAUI (T). In 2015 30th IEEE/ACM International Conference onAutomated Software Engineering (ASE), pages 248–259, nov 2015. |
| 4 | Bongjae Kim, Hong Min, and Junyoung Heo. Identifying UI widgets of mobile applications from sketch images. *Journal of Engineering and Applied Sciences*, 13(6):1561–1566, 2018. |
| 5 | Beltramelli, T. (2018). *Pix2Code: Generating Code from a Graphical User Interface Screenshot*. In Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computin  g Systems, EICS. |
| 6 | Zhihao Zhu, Zhan Xue, and Zejian Yuan. Automatic Graphics Program Generation using Attention-Based Hierarchical Decoder. CoRR, abs/1810.1, 2018. |
| 7 | Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI Implementation. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE ’18, pages 665–676, New York, NY, USA, 2018. ACM. |
| 8 | Yanbin Liu, Qidi Hu, Kunxian Shu, “Improving pix2code based Bi-directional LSTM”, IEEE International Conference on Automation, Electronics and Electrical Engineering, 2018. |
| 9 | Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Joshua B Tenenbaum. Learning to Infer Graphics Programs from Hand-Drawn Images. CoRR, abs/1707.0, 2017. |
| 10 | Robinson, Alex. “*Sketch2code: Generating a website from a paper mockup”*,May 2019 |
| 11 | Naik, Harish and Raj, Rishav and Jain, Sanidhya and Srinivasa, H. and Goklani, Denish, STML (Sketch to Markup Language) (December 1, 2020). Available at SSRN: <https://ssrn.com/abstract=3740413> or [http://dx.doi.org/10.2139/ssrn.3740413](https://dx.doi.org/10.2139/ssrn.3740413) |
| 12 | Vaishnavi Kalbande, Kajal Meshram, Samiksha Somnath, Raksha Deshmukh, Mrunali Mohod, “Automatic HTML Code Generation from Mock-Up Images Using Machine Learning Techniques”, International Research Journal of Engineering and Technology (IRJET), Volume: 08 Issue: 06, June 2021. |
| 13 | Deolekar, Aditi, Kimaya Dhanawade, Rahul Chavan and Sayali Bhambure. “UI Code Generation using Deep learning.” (2021). |
| 14 | Sarah Suleri, Vinoth Pandian Sermuga Pandian, Svetlana Shishkovets, and Matthias Jarke. Eve. In Extended Abstracts of the 2019 *CHI Conference on Human Factors in Computing Systems* - CHI EA ’19, pages 1–6, New York, New York, USA, 2019. ACM Press. |
| 15 | Young-Sun Yun, Jinman Jung, Seongbae Eun, Sun-Sup So, and Junyoung Heo. Detection of GUI Elements on Sketch Images Using Object Detector Based on Deep Neural Networks. In Seong Oun Hwang, Syh Yuan Tan, and Franklin Bien, editors, Proceedings of the Sixth International Conference on Green and Human Information Technology, pages 86–90, Singapore, 2019. Springer Singapore. |
| 16 | Bada Kim, Sangmin Park, Taeyeon Won, Junyoung Heo, and Bongjae Kim. Deeplearning based web UI automatic programming. In Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems - RACS ’18, pages 64–65, New York, New York, USA, 2018. ACM Press. |
| 17 | B. A§iroglu et al., ’’Automatic HTML Code Generation from Mock-Up Images Using Machine Learning Techniques,” 2019 Scientific Meeting on Electrical-Electronics Biomedical Engineering and Computer Science (EBBT), Istanbul, Turkey, 2019, pp. 1-4, doi:10.1109/EBBT.2019.8741736. |
| 18 | Kevin Patrick Moran, Carlos Bernal-Cardenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk. Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps. IEEE Transactions on Software Engineering, 5589(May):1– 26, 2018. |
| 19 | Mohammad Bajammal, Davood Mazinanian, and Ali Mesbah. Generating reusable web components from mockups. Proceedings of the 33rd ACM/IEEE InternationalConference on Automated Software Engineering - ASE 2018, pages 601–611, 2018. |
| 20 | Thomas F. Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. Learning Design Semantics for Mobile Apps. The *31st Annual ACM Symposium on User* *Interface Software and Technology* - UIST ’18, pages 569–579, 2018 |