Dissertation on

"Image Recognition Based HTML Generator"

Submitted in partial fulfillment of the requirements for the award of degree of

Bachelor of Technology
in
Computer Science & Engineering

Submitted by:

Kunal Paliwal	**01FB16ECS172**
Mayank S Rao	**01FB16ECS199**
Prerna Rao V	**01FB16ECS276**

Under the guidance of

Internal Guide
Ms. Vidhu Rojit
Adjunct Professor,
PES University

January – May 2020

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)

100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

'Image Recognition Based HTML Generator'

is a bonafide work carried out by

Kunal Paliwal	**01FB16ECS172**
Mayank S Rao	**01FB16ECS199**
Prerna Rao V	**01FB16ECS276**

In partial fulfilment for the completion of eighth semester project work in the Program of Study Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period Jan. 2020 – May. 2020. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 8th semester academic requirements in respect of project work.

Signature	Signature	Signature
Ms. Vidhu Rojit	Dr. Shylaja S S	Dr. B K Keshavan
Adjunct Professor	Chairperson	Dean of Faculty

External Viva

Name of the Examiners   **Signature with Date**

1. _____   _____

2. _____   _____

_____

DECLARATION

We hereby declare that the project entitled **"Image Recognition Based HTML Generator"** has been carried out by us under the guidance of Ms. Vidhu Rojit and submitted in partial fulfillment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester January – May 2020. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.
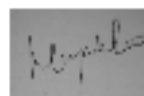
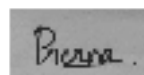|		
01FB16ECS172	**Kunal Paliwal**	
01FB16ECS199	**Mayank S Rao**	
01FB16ECS276	**Prerna Rao V**	

_____

ACKNOWLEDGEMENT

_____

ABSTRACT

This project provides a service to generate HTML pages from pure drawings. The motive is to build an easy and open source method to generate pages without spending time on positioning and basic CSS. It aims to help designers show prototypes to clients in real-time for a faster approval; to help teachers draw sample pages to be generated, and focus solely on functionality; and for any areas where only a basic responsive frontend is required.

To do this, the basics of wireframing and their style of representing HTML elements on paper were considered. Machine Learning and Image Processing techniques were used to parse, process and predict the images. The core of this project centers around the Support Vector Machine. With this, drawings are predicted at a fairly high accuracy and use coordinate geometry and HTML / CSS to generate the pages required. These pages are fully downloadable and usable HTML pages.

_____

_____

TABLE OF CONTENTS

_____

―――――――

―――――――

_____

_____

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

The creative design process for a website begins with collaboration where designers share ideas. These ideas describe the basic structure of a web page. The basic structure of a webpage is most often a mockup, a sketch, or a wireframe.

The same stands for academia as well, for someone teaching or learning how to use front-end technologies, they usually use skeletal web-pages or sketches.

The aim of this project/product is to speed up this process through the use of Image Recognition, to build an application that understands and recognizes specific elements or objects, then translates that understanding to HTML code.

The coding process begins with wireframes/images drawn on paper, which would be instantly converted to HTML5 code and would help designers give instant feedback, or save time in the academic area.

Multiple different designs could also be instantly tested and shown to customers, allowing for faster development.

2. PROBLEM DEFINITION

This project is a web-based application aimed to help the end-users quickly convert their wireframe sketches into working downloadable HTML5 code ready for use. The inspiration from this project is drawn from Microsoft's sketch2code beta service which offers the same functionality to convert wireframes to HTML5 code.

The project follows the following approach:

- Upload the hand-drawn/wireframe sketch to the UI.
- The image is pre-processed i.e. all the shapes are cropped, gaussian pass filters and canny edge filters are applied to remove noise from the input image.
- The cropped objects are then passed to the SVM model which classifies it as a specific HTML element.
- Using this information, HTML code is written to a file using an HTML factory built with Python classes and inheritance, and the elements are positioned as drawn in the image with a small padding margin.
- The output page is displayed on the UI and basic CSS can be added to the page, ready for immediate download and use.

The scope of the project is limited to 4 common HTML elements namely, input, image, button and checkbox. The application will provide functionality to:

- Upload hand-drawn images on paper or on a system to the application
- Convert these images to HTML web pages
- Preview the generated HTML page
- Modify basic styling of the elements of the page
- Download the source code to the previewed page

_____

3.   LITERATURE SURVEY

3.1.   Object Detection with Deep Learning

3.1.1.   Introduction

This paper explores methods to accurately locate objects contained in each image (object detection), which consists of many sub tasks like face, pedestrian and skeleton detection. It explores the different methods that can be applied to achieve object detection for the required use case and the most optimal approach to help solve the problem. It explains the factors that affect the learning systems and how they affect the object detection.

3.1.2.   Approaches:

- Region proposal based framework
- Experimental evaluation
- Classification based framework

3.2.   Scale-Invariant Convolutional Neural Network

3.2.1.   Introduction

The idea initially was to be able to pass any image and redeem a useful output or detected object from it. This would eliminate the need for pre-processing and would speed up the process and enable faster testing and prototyping with heavier focus being on the Machine learning aspect of the project. For this a SiCNN or the Scale-Invariant Convolutional Neural Network was considered. While traditional methods involve using augmentation and scale jittering, the SiCNN has the ability to detect and extract multi-scale features and classify

_____

them. The SiCNN basically divides an image into a multiple column architecture where each column represents or looks at a different scale and considers the same parameters during scale transformation.

This model would ideally fix any issues we had without having to deal with an exorbitant amount of data along with creating various scales and sizes of images. The SiCNN also helps in reducing model size and provides output with excellent accuracy and is extremely robust at multiple scales of variation

3.2.2.   Model Architecture
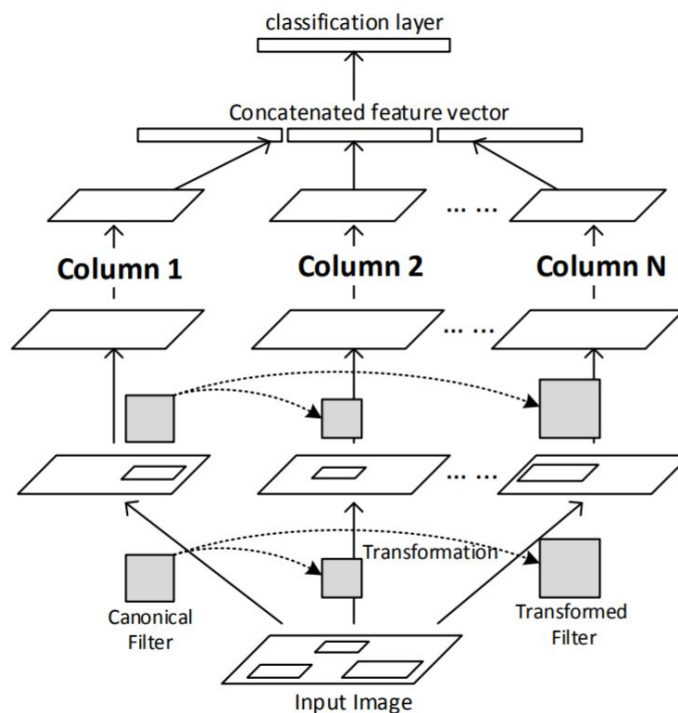
Image 3.1: SiCNN Model Architecture

3.2.3.   Results and Conclusions

Of course, as is to be expected, quality comes at a cost and the same is true here. The paper points out that scale invariance increases cost linearly with respect to

the number of columns and hence the requirement of incremental learning methods was necessary.

Model	Error on CIFAR-10	Error on scaled CIFAR-10	cost
CNN	17.33%	24.82%	1
SiCNN	14.22%	18.83%	6
SiCNN, `inc-1`	14.71%	20.10%	3.5
SiCNN, `inc-2`	16.06%	23.24%	$1 + \epsilon$

Table 3.1. Error rate and cost

The SiCNN not only takes into account the varying scale of the images it wishes to classify or learn from but also incorporates flipping of images into the model. This means that images are flipped along both the X-axis and Y-axis and hence provide a more robust performance over general simple CNNs. These optimization techniques are both well received within the community and are complemented well with other optimization techniques. From the results of the paper it is possible to glean that the model has in fact learnt features of different scales from different models. The SiCNN can act in any area that a traditional CNN can and hence it provides not only more leeway with regards to scale and size of images, but also provides a more robust and error free network.

3.3.   You Only Look Once (YOLO CNN)

3.3.1.   Introduction

Considering speed is of the essence, both in training and prediction, using a more advanced version of a CNN or R-CNN called the YOLO CNN was considered. This CNN is used even in OpenCV due to being much faster than even a Fast R-CNN.

This paper proposes an object detection approach where it is framed as an algorithm to spatially separate associated class probabilities and bounding boxes.

The YOLO model runs through images in real-time at 45-46 frames per second. Even though YOLO makes more localisation errors in comparison to other detectors, it is much less likely to detect the image wrong. It learns general ideas of objects and hence performs better than other methods.

3.3.2. Limitations

As YOLO puts in place tight spatial constraints on predicting bounding boxes, it puts a cap on the number of close by objects that the model is able to predict. It also suffers to accustom to objects in new configurations or aspect ratio. Errors in small and large boxes are treated the same, this causes scaling to suffer greatly.

3.3.3. Conclusion

YOLO can be constructed and trained directly on complete images. Unlike approaches based on classification, this model performs at a much higher rate thanks to its training loss function. It also generalises and performs very well on new domains. This makes it the choice for applications that require fast object detection.

3.4. sketch2code: paper mockup to generated website

3.4.1. Introduction

In this paper[3], author Alexander Robinson presents two approaches that automate the process of converting wireframes to basic front end code. One approach uses computer vision techniques, and the other uses the application of deep semantic segmentation networks. The paper also introduces a new approach that evaluates by automatically synthesizing sketches. This paper

finds that deep learning approaches are more efficient than computer vision techniques, and are more promising directions for future research.

3.4.2. Related Work

This paper highlighted two gaps that were found in this domain. One, that there were very few Deep Learning methods in this domain of research. Two, that there was no sketch to code solution designed yet.

It looks into two main categories - low fidelity images and high fidelity images.

On looking into existing applications in the domain, it found that -

- Low fidelity image applications (SILK/MOBIDEV) successfully used Computer Vision to detect contours, corners, lines, and edges to classify elements drawn digitally into predefined components. But the time considerations to extend to new elements and the post-processing of detected components were extremely difficult. Positioning, too, was found to be hard to manipulate.

- High fidelity image applications (REMAUI/pix2code) used Deep Learning methods as well as CV techniques to translate wireframes or screenshots into code. REMAUI proved very difficult to add new elements to, and pix2code only worked with its RNN based synthesized data set; it did not generalize to real-world examples.

Post this research, it was found that there was no research or attempt to transform wireframes on paper directly to code

This paper attempts to apply Deep Learning techniques on low fidelity images. The following techniques were looked into:

3.4.2.1. COMPUTER VISION TECHNIQUES

- Image Denoising: They chose a median blur rather than a Gaussian blur or a denoising autoencoder, so that the technique was fast as well as edge-preserving.

- Colour Detection: By setting threshold values and recoloring the image to RGB.
- Edge Detection: They chose the Canny edge detection algorithm rather than the Sobel or Richer Convolution features
- Segmentation: They chose to go forward with structural based segmentation, rather than heavy contour detection or detection and approximation techniques.
- Text Detection: They used an SWT approach for text detection. This only detects (does not recognize) handwritten text.

3.4.2.2. MACHINE LEARNING TECHNIQUES

- Multilayer Perceptrons: They are a class of feed-forward ANNs that are used to distinguish data that is not linearly separable. However, there are multiple Machine Learning implementations and approaches that work similarly to MLPs.

  The paper mentions that Support Vector Machines can aid in such multi-classification problems, by using binary classification SVMs in parallel.

- Semantic Segmentation: This paper chose to work with Deeplabv3+ because of its top performance. It performed this well due to its use of dilated convolutions and ASPP, for robust segmentation at multiple scales.

3.4.3. Approaches

The two goals of this paper were to create an application to translate wireframes to code and to compare different to achieve the best performance. The two approaches are described later, below.

To create the dataset, the chosen method in this paper was to collect existing websites/webpages and auto sketch wireframes of them. This technique was most convenient to build a large dataset and to avoid mistakes due to human errors/opinions.

Since wireframes have a small element set and are style invariant as compared to HTML, all the elements were normalized into broader element

classes, and styling removed, and the images were cropped and converted to black and white.

These images were then used as input for the two approaches:

- Classical Computer Vision
- Deep Learning Segmentation

3.4.4. Evaluation and Conclusion

The approaches were evaluated on the following criteria:

- Classification of elements and containers

  Deep learning performed better in element classification and detection, while Computer Vision worked better on containers.

- Similarity between provided input and generated output

  The Deep Learning techniques showed a more consistent performance than the Computer Vision approaches

- Generalisation to unseen examples or inputs

  Computer Vision mildly outperformed Deep Learning methods.

The results indicated that deep learning solutions could provide a higher performance and can be used to build consumer applications.

_____

4. PROJECT REQUIREMENTS SPECIFICATION

4.1. Scope

- This project/product is a web application intended to generate HTML pages with rendered elements and downloadable source code for further additions.
- UI will be provided to upload a picture of the hand-drawn image to be converted to an HTML page.
- On upload, there will be an option to convert the image to source code.
- The HTML page will then be generated and will be available to view on the screen.
- A button to download the source code will be provided as well.
- The product will be able to generate basic elements such as - images, input text fields, checkboxes, buttons.
- It will not provide the functionality to generate CSS.
- Scaling may not be as accurate as required by the user due to hand-drawn images.

4.2. Gantt Chart

_____

_____
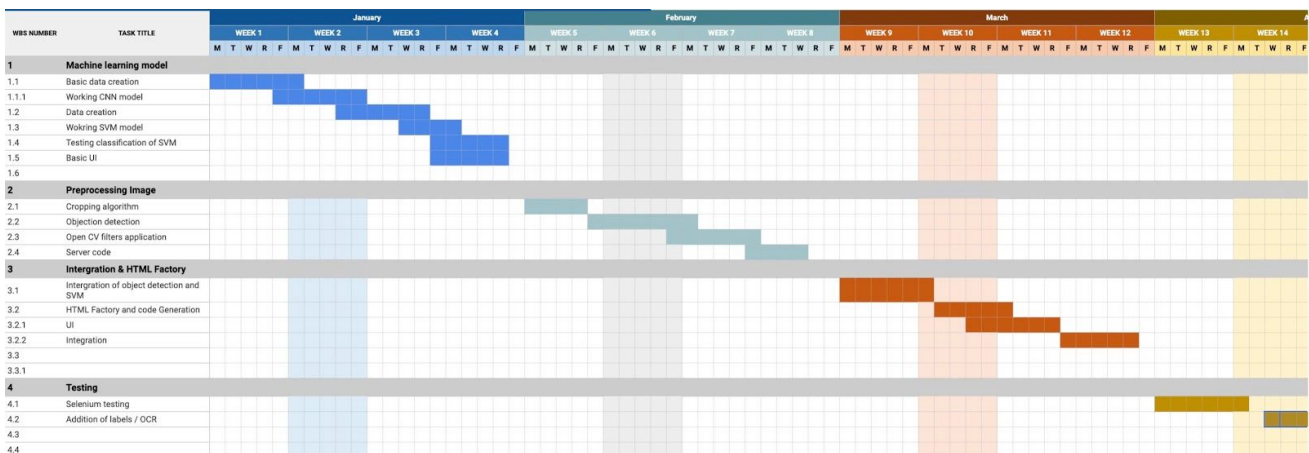
4.3. Product Perspective

4.3.1. User Characteristics

- **Developers**: Front-end developers waste time when typing out basic code, when a quick prototype can be generated and modified from there instead. It allows backend developers to focus on backend code with a simple frontend.

- **Designers**: UI designers can use the application to be able to easily prototype Web Pages to show the clients samples of how a real page would look like. If the client doesn't like the page structure after coding is done, it wastes the designer's time. This would simplify the process.

- **Academics**: Students and teachers can use the application so that websites can be created instantly or copied from the board without having to do tedious HTML coding every time. Not all students are good designers or know HTML, CSS. This will enable them to create a basic page for enough functionality without wasting time. It also allows teachers to quickly modify pages and frontend to focus more on backend code.

4.3.2. Risks

- Severe lack of data, i.e, all data has to be manually created by hand.
- Positioning will be tricky to maintain as drawings don't give an accurate representation of position.
- Lack of papers or source material to read up on for a clearer understanding. The majority of the project is dependent on personal innovation.

_____

4.4. Requirements List

4.4.1. Module / Scenario 1

Reqmt #	Requirement
CRS – 1	UI should be easy to manage and intuitively usable

4.4.2. Module / Scenario 2

Reqmt #	Requirement
CRS – 2	Users should be allowed to upload a hand-drawn image from a local machine using the provided UI.

4.4.3. Module / Scenario 3

Reqmt #	Requirement
CRS – 3	The preview of the generated page should be displayed on the UI so that the user can verify/change the image if necessary.

4.4.4. Module / Scenario 4

Reqmt #	Requirement
CRS – 4	Objects and diagrams should be detected accurately around 80% of the time.

4.4.5. Module / Scenario 5

Reqmt #	Requirement
CRS – 5	The returned file should follow clear HTML syntax with necessary formatting.

_____

_____

5.   SYSTEM REQUIREMENTS SPECIFICATION

5.1.   Functional Requirements

5.1.1.   UI Page

The application shall allow a user to upload and download an image. It should also be able to perform all the actions required by the user.

5.2.   Non-Functional Requirements

5.2.1.   Dependencies

- The application requires a system with Python 3.6
- Some packages as mentioned below would be required to run the models.

5.2.2.   Assumptions

- All aspects of this application will meet the user's requirements
- The input provided by the user matches the minimum requirements of the application.

5.3.   External Interface Requirements

5.3.1.   Hardware Requirements

- Apart from a 32/64-bit architecture machine, there are no specific hardware requirements as such to run the product. At most, a decent computer would be required to launch a server and run the model, however, this isn't compulsory.
- Only PCs will be supported. Due to Android and iOS file systems being slightly problematic to work with, along with the fact that this project is written in Python -- which is not supported natively by either Android or iOS.

_____

Any supporting framework to allow Python to run would most likely hamper the quality of the tool.

5.3.2. Software Requirements

- Filesystems for file operations
  - NTFS, FAT32, exFAT, ext3, APFS
  - Windows 7 onwards, Ubuntu 16.04 onwards, Yosemite onwards for OSX

- Visual Studio Code

  VSCode is an IDE by Microsoft to help build GUI, console, web and mobile applications, cloud services, etc.

- Google Colab

  It is a free cloud service which also allows a free GPU.

- Python 3.6
  - Flask and its dependencies run on Python
  - 3.6 only, as it is the current most stable release
  - Python2 is no longer supported by the Python foundation and is not supported

- Packages/Libraries used:
  - Scikit-learn

    It is a machine learning library that features algorithms like SVM, k-neighbours, etc.
  - Keras (2.0.0 +)

    Supplement module that runs on Tensorflow and abstracts and agglomerates functions from the Tensorflow library for ease of access
  - TensorFlow(2.0)

It can be used to create large-scale neural networks with many layers and is used for classification, prediction, etc.

- NumPy

  It is a Python library that supports large multidimensional arrays, in addition to functions to operate on these values.

  It is used in this project to store and work with images.

- Matplotlib

  It is a library for Python, NumPy used for embedding plots into applications.

- PIL

  PIL or Pillow (formerly), is a Python Imaging Library that provides support for saving different image formats

- OpenCV

  It is a computer vision library used to do image related operations

- Pickle

  It is a module used to serialize and deserialize object structures in Python

- OS

  It is a Python module that enables functionality dependent on the operating system.

- Random

  It is a library that is used to pick a random number from the provided range of values.

- HTML5/CSS3/ES6
  - Compatible with Firefox and Chrome
  - Edge browsers may have an issue running the applications
  - Internet Explorer and Mini browsers such as Opera are not supported

5.3.3. Communication Interfaces

- Communication is possible over Localhost if deployed as a downloadable tool
- If there is scope for the application to be deployed on Cloud, it will be deployed over EC2 instances with Docker containers running multiple instances of the server code.
- This would be accessible over the web with some associated IP addresses.
- If run as a tool, the server will run on the system as a daemon process on some associated port
- If on the server, the client has to request a webpage and can follow through from there.
- Over the web, there would be a slight delay, whereas on the localhost it would be instantaneous.

5.4. User Interfaces

- Front end software: HTML, CSS, Javascript, Bootstrap
- Back end software: Python
- Framework: Flask

5.5. Help

Git-hub repository to provide specific installation guides and developer support.

_____

6.   HIGH-LEVEL DESIGN

6.1.   System Design/Architecture
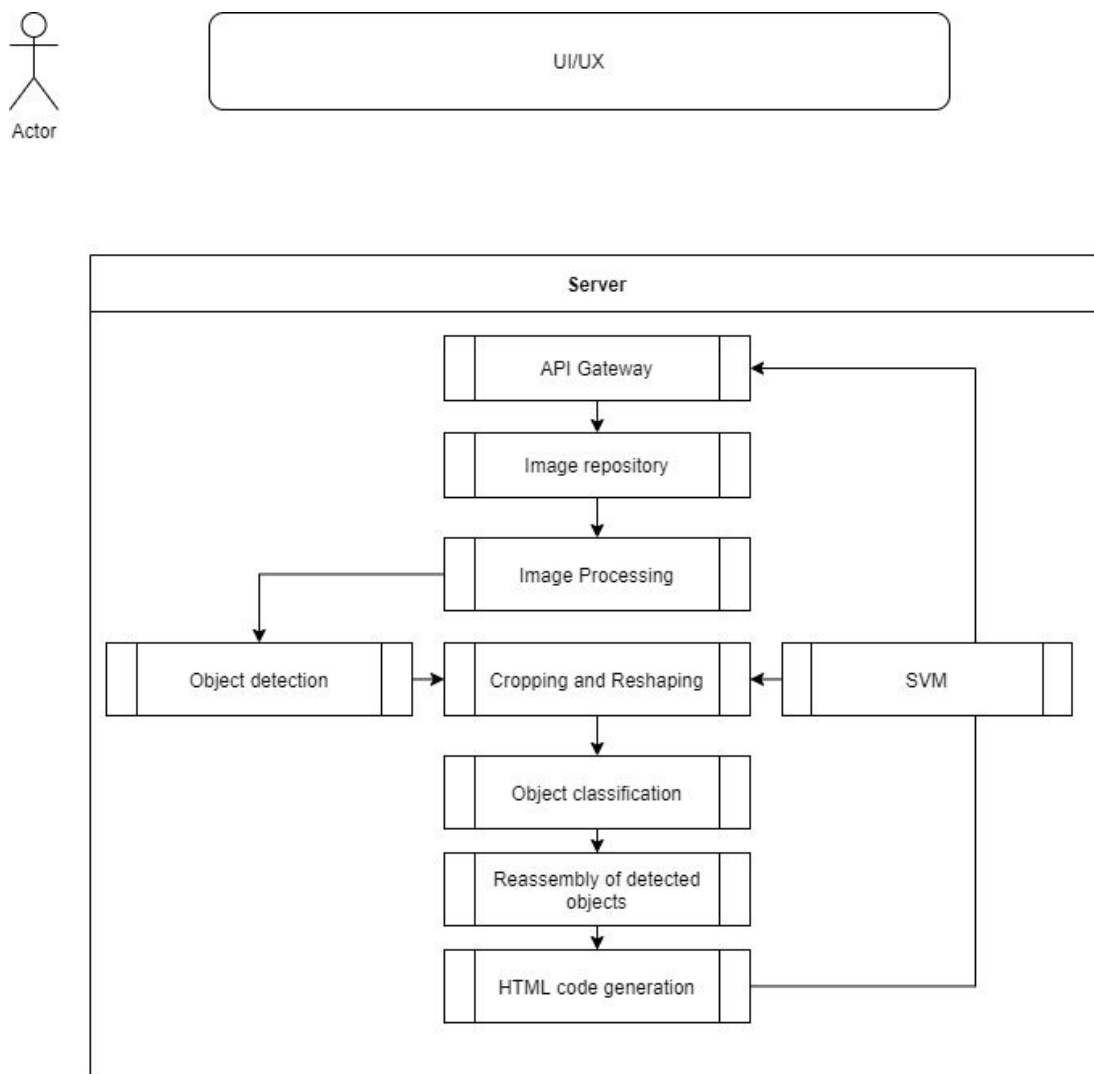
Image 6.1: System Architecture for this application

The UI design has the following features:

- Input image upload
- Input/Output preview
- Facility for conversion of input and generation of HTML output
- Facility to modify basic styling of rendered elements
- Facility to download output

6.2. Design Constraints, Assumptions, and Dependencies

6.2.1. Assumptions

- Lines are drawn fairly straight
- Picture is taken in a well lit environment
- The image is not already close-cropped and has some padding around it
- Elements drawn have an area of at least 1cm sq
- Horizontal images provide better performance

6.2.2. Limitations

- Elements have to have a minimum margin and size to be detected properly and for ideal cropping
- Images must be uploaded in .jpg format

6.2.3. Obstacles/Risks

- Severe lack of data, i.e, all data has to be manually created by hand.
- Positioning will be tricky to maintain as drawings don't give an accurate representation of position.
- Lack of accuracy during the prediction of element
- Custom tags will require large amounts of data as well
- Lack of papers or source material to read up on for a clearer understanding.
- The majority of the project is dependent on personal innovation.

6.2.4. Dependencies

- Python = 3.6

6.3. Design Description
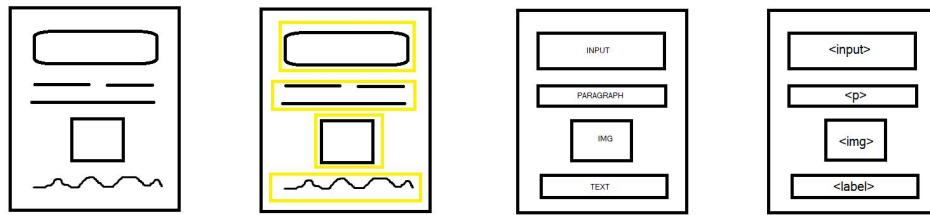
Image 6.2: (a) Hand-drawn image (b) Object Detection (c) Classify into tags (d) Generated HTML Page

- HTML tags are encoded using special symbols (Eg.: A box with an X, to indicate an image)

- Input is provided in the form of a drawing of a webpage with these encoded symbols/tags.

- Each individual element in the image is then identified and cropped out using the object detection module.

- Each detected object is passed through the SVM to identify and then classify it into an HTML tag type.

- Each classified element/tag is converted to HTML code, and an HTML page is generated.

- All elements are positioned and aligned as accurately as possible to match the input image.

- After applying styling, the final output page is generated.

6.3.1.   Master Class Diagram
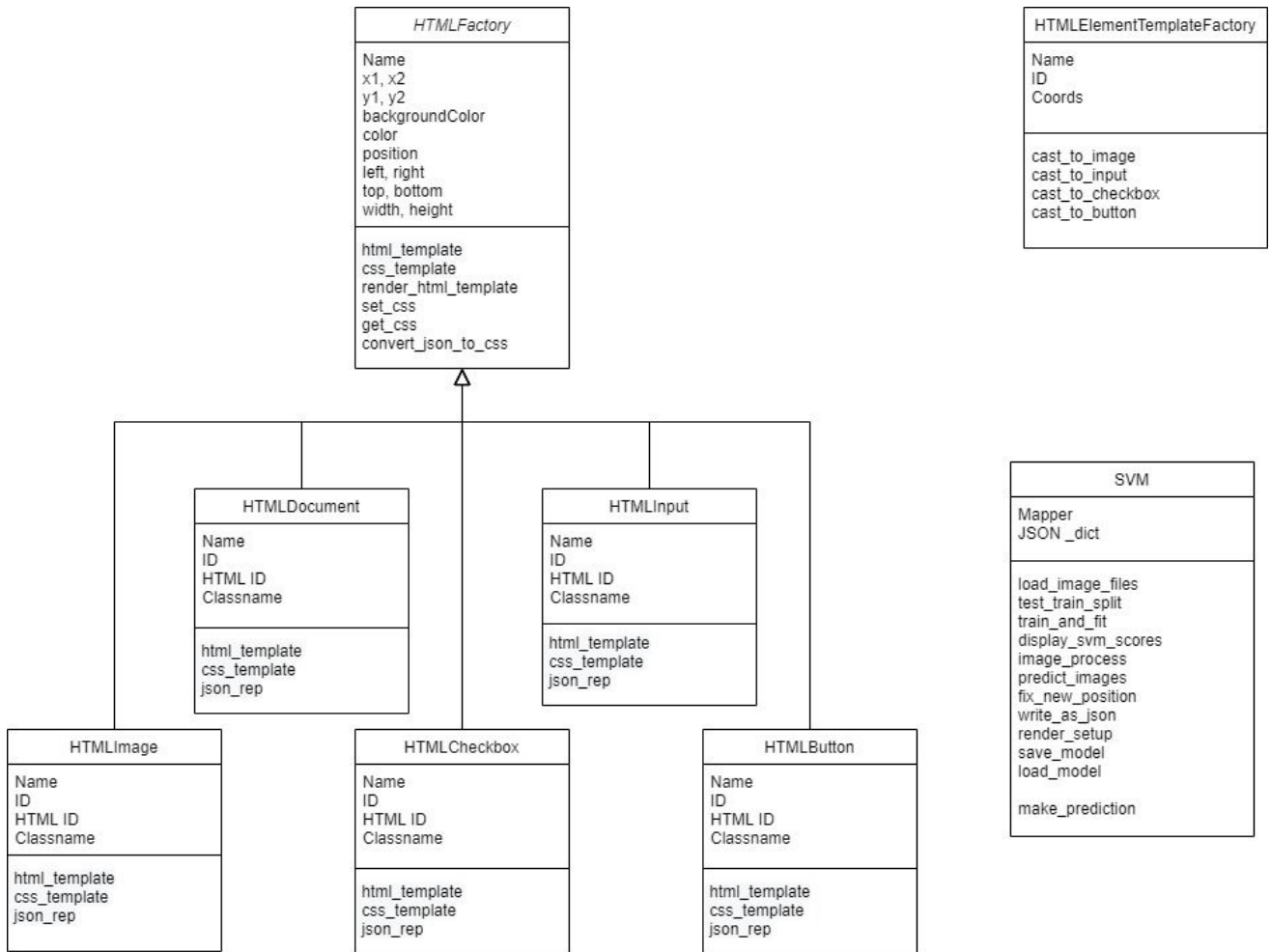
Image 6.3: Master Class Diagram

6.3.2.   Master Use Case Diagram
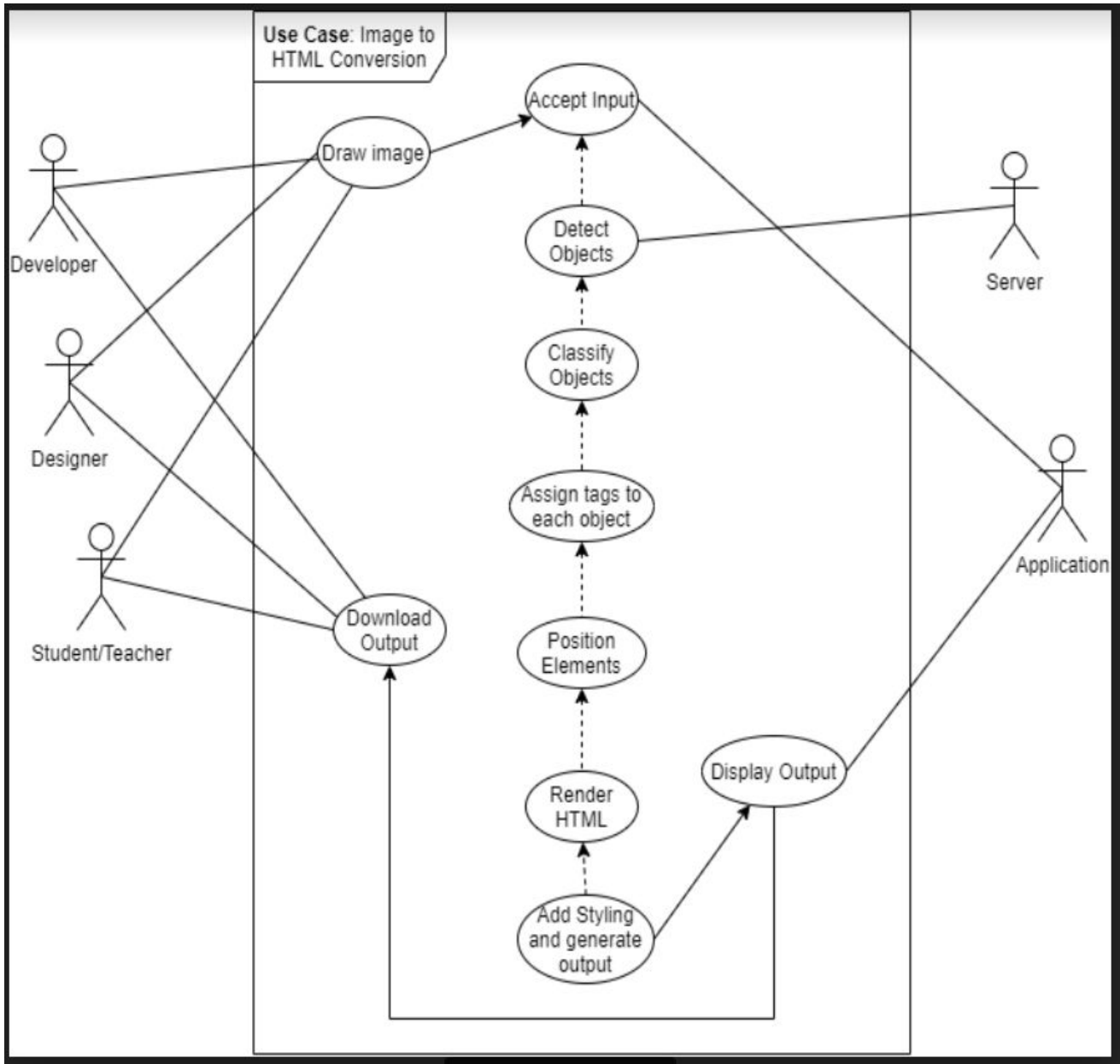
Image 6.4: Master Use Case Diagram

Description:

Use Case Item	Description
Draw Image	Done to provide a hand-drawn image as input to the application
Accept Input	To accept input into the application and send it to the server
Detect objects	To detect individual elements on the page
Classify objects	To identify and classify each detected element into one of the predefined classes
Assign tags to each object	To assign the corresponding HTML tag to each classified element
Generate/Render HTML	To render all the detected elements in HTML
Position on output page	To align and position all items on HTML page as accurately as possible
Add styling to generated page	To add CSS to the HTML page and display the updated styles
Display/download output	To preview the output page and download final code

Table 6.1: Use Case Description

6.4. Sequence Diagram
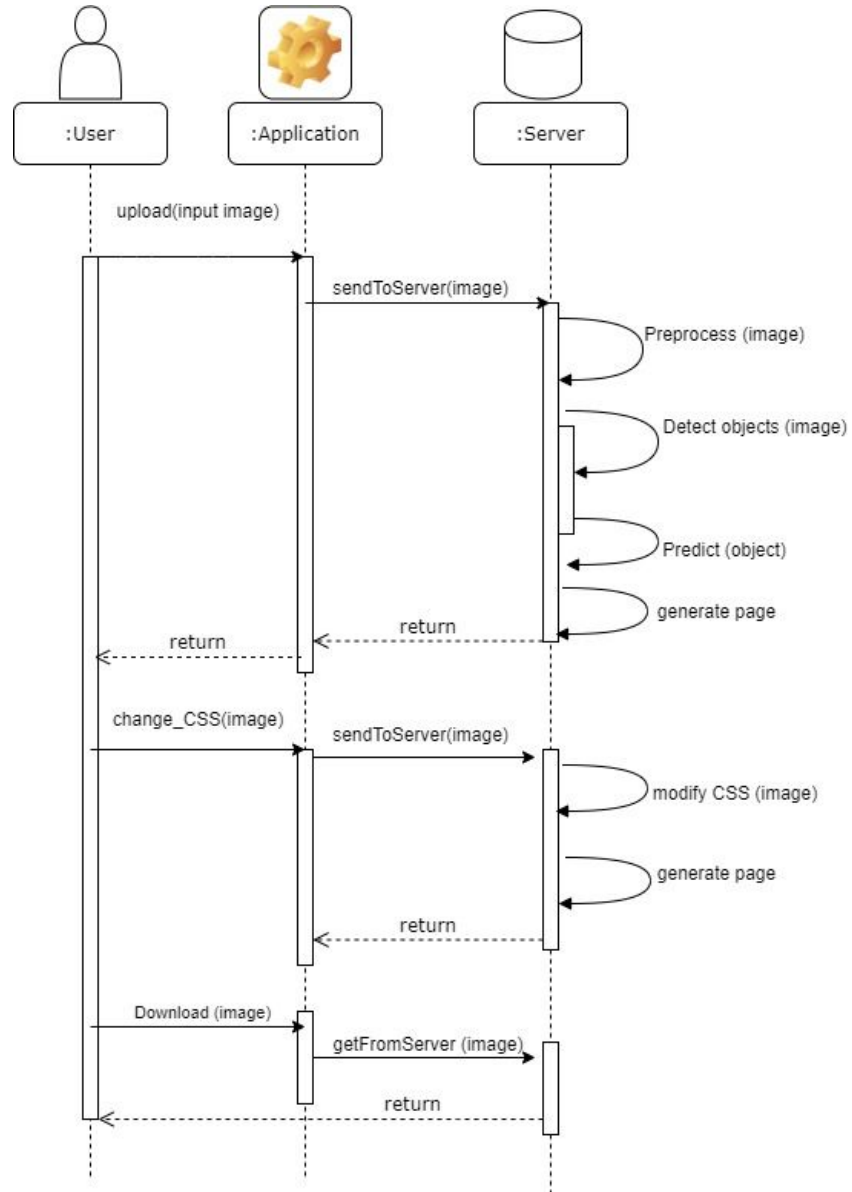
Image 6.5: Master Sequence Diagram

_____

6.5. Modules

6.5.1. Object Detection with OpenCV

The input to this module is the hand-drawn input image. This module has been designed to go through the entire input image, and then separate it into its constituent elements (or pieces). Each element detected in the image is cropped and made suitable for the next module, i.e, the SVM or Support Vector Classifier. The output of this module is SVC compatible detected objects.

The pieces are decided by the Canny edge detection and contour detection facilities provided by OpenCV.

6.5.2. SVM with Sklearn

The de facto module for building and running SVMs is the Sklearn module. Combined with Skimage submodules, the entire module allows us to build the SVM. the SVM is basically a class built out of Sklearn provided methods.

The input to this module is the detected objects/ elements from the object detection module. This module has been designed to classify each element on the input image into one of the predefined HTML tags. It classifies each element and then labels them. This is stored as metadata and sent to the next module, i.e, the HTML Factory module.

6.5.3. HTML Factory

The input to this module is the metadata created by the SVC module. This module has been designed to create the final HTML page. It renders, positions, and styles all the detected and classified elements. The output of this module is the final HTML page which can be accessed by the user.

_____

6.5.4. Server

This module is the entry point of the application. It provides functionality for the user to upload an image, and then sends the generated HTML page - with a preview and facility to modify the style, and download the source code.

6.6. User Interface

The UI is easy to manage and to understand. All functions of the application are visible on the single screen below.
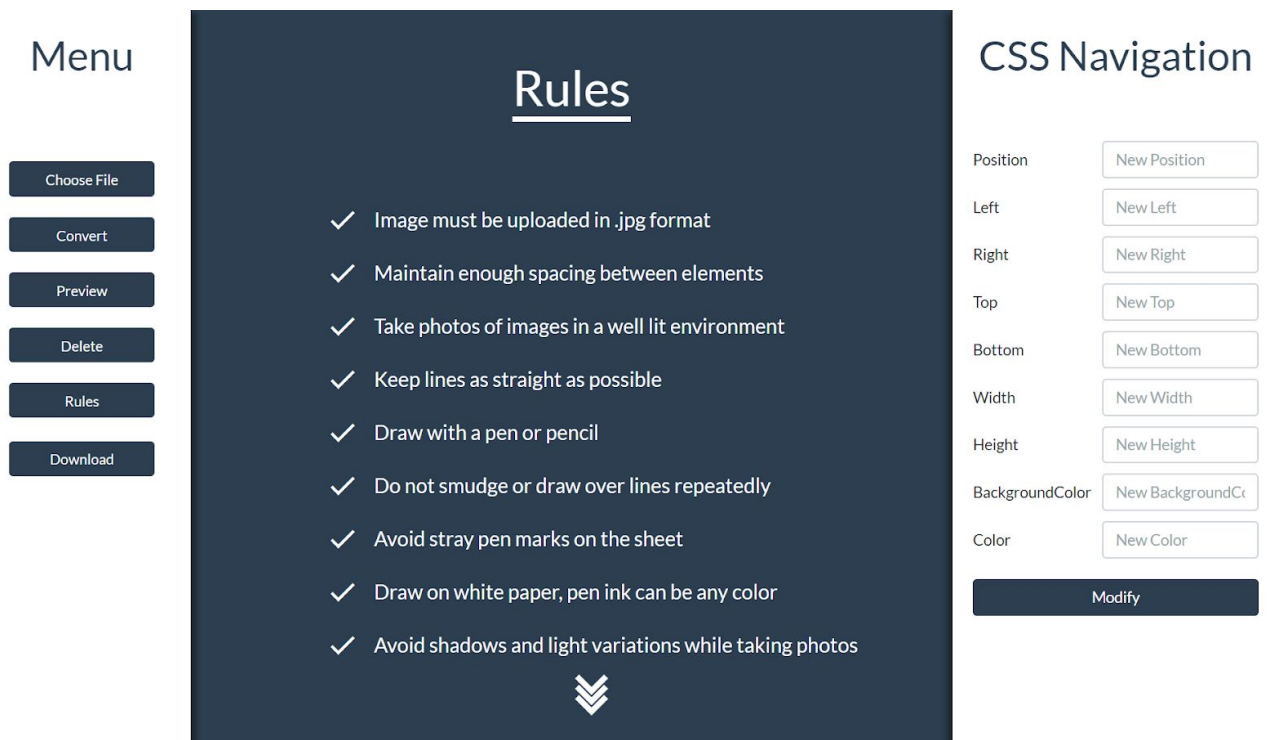
Image 6.6: Main User Interface

6.7. Help

Git-hub repository to provide specific installation guides and developer support.

_____

6.8.   Alternate Design Approach

This section shall describe the design approaches that were considered. The limitations and advantages of each approach shall be explained briefly.

- The initial approach was to use a Neural Network to classify the detected elements. Though this would have provided a higher accuracy than an SVC, it was skipped because of the lack of data. The NN was not learning enough with the limited data that had been created, and creating/building data was proving to be extremely tedious with minimum success in training the NN. With lesser data, SVMs were guaranteed to provide a better answer at the cost of accuracy. Scaling an SVC would also be easier than scaling a NN. Since the images were going to be very basic and classes limited, an SVC would be extremely fast and easy to train.

- Coming to a decision on what symbols would be used to represent each different element was a slightly long process. It proved to be slightly tricky as it had to be ensured that the symbols made sense logically/intuitively and that no symbol was too similar to the others in a way that would confuse the classifier.

  This led to multiple ideas, and a lot of trial and error until finally deciding on a set of symbols for the elements.

- Some alternate approaches that were considered for providing CSS to the generated page. One approach was to accept colored images as input, but it was chosen not to go forward with this approach as it would be extremely difficult to maintain styling through the preprocessing, and would greatly increase the number of training parameters for classification. Another approach was to provide a fixed basic styling to every generated web page, but it was chosen not to continue this as it reduced functionality and client-side flexibility.

_____

6.9. Reusability Considerations

This application can be installed on different systems and is reusable.

As future improvements, to access the application from anywhere, it can be hosted on a central server.

All HTML elements are objects that inherit from a base class and aim to provide maximum reusability and cohesion.

7. LOW LEVEL DESIGN

7.1. Design Description

7.1.1. object detection

7.1.1.1. Description

● Data Members

Data Type	Data Name	Access Modifiers	Initial Value	Description
integer	id_gen	Public	0	Generated ID
array	HTML_objects	Public	None	
array	img	Public	None	Input image
array	cnts	Public	None	Detected contours
file	dbfile	Public	None	To store metadata
array	coords	Public	None	Coordinates of object

Table 7.1: Data Members 1

● Methods

1. Create_crops

   ○ Purpose: To apply preprocessing steps to the image and store original coordinates

   ○ Input: uploaded image from user

   ○ Output: Processed and cropped objects and image

   ○ Parameters: uploaded image, objects in image, coordinates

2. split_images( )

   ○ Purpose: To accept the image and split it into its constituent parts

   ○ Input: uploaded image from user

   ○ Output: Array of images of detected objects

   ○ Parameters: None

_____

7.1.2. Server

7.1.2.1. Description

7.1.2.2. Data members

Data Type	Data Name	Access Modifiers	Initial Value	Description
String	json_str	Public	None	Obtains JSON data from client

Table 7.2: Data Members 2

7.1.2.3. Methods

- get_file
  - Purpose: Upload input image to convert
  - Input: Image convert to HTML code
  - Output: None
  - Parameters: None
- get_page
  - Purpose: Preview the HTML output file
  - Input: None
  - Output: HTML file opened in browser
  - Parameters: None
- download
  - Purpose: Get useable version of file
  - Input: None
  - Output: HTML file
  - Parameters: None
- delete_page
  - Purpose: Clear the generated preview
  - Input: None
  - Output: UI refresh

_____

_____

- ○ Parameters: None
  - ● modify_css
    - ○ Purpose: Add basic css
    - ○ Input: User input for changes to attributes
    - ○ Output: Updated page with CSS modifications
    - ○ Parameters: None
  - ● root
    - ○ Purpose: Render home page
    - ○ Input: None
    - ○ Output: Homepage of the web app
    - ○ Parameters: None

7.1.3. SVM

7.1.3.1. Description

7.1.3.2. Data Members

Data Type	Data Name	Access Modifiers	Initial Value	Description
String	file_name	Public	None	Name of the file to be read
List	HTMLobject_list	Public	None	List of objects that store all the objects created by the HTMLFactory
Class	svm	Public		Trained model
HashMap	mapper	Private	{0:"Button" ,1:"Checkbox",2:"Image" ,3:"Input", 4:"Video"}	Maps SVM output to object

List<Map>	param_grid	Private	Too large to add, but contains C value, gamma value and kernel for SVM	Provides parameters for GridSearch to run
List	X_train,X_test,Y_train,Y_test	Private	[ ]	Training splits and testing splits
Tuple	dimension	Private	(64,64)	Dimension required for SVM training

Table 7.3: Data Members 3

7.1.3.3. Methods

● predict_images

   ● Purpose: Image classification

   ● Input: object

   ● Output: Classification labels

   ● Parameters: Object

● make_prediction

   ● Purpose: Reload model in case of corruption

   ● Input: Object

   ● Output: Image classification

   ● Parameters: None

7.1.4. HTMLFactory

7.1.4.1. HTMLFactory

● Description

● Data members

Data Type	Data Name	Access Modifiers	Initial Value	Description
integer	prev_left	Private	None	Old values for left

integer	prev_top	Private	None	Old values for top

Table 7.4: Data Members 4

- Methods
    - attach_new_<attribute>
        - Purpose: Add attribute setting to element
        - Input: HTML element , attribute value
        - Output: HTML element
        - Parameters: object, integer
    - view_coordinates
        - Purpose: View coordinates of HTML element
        - Input: None
        - Output: Co-ordinates
        - Parameters: Object
    - set_<css_attribute>
        - Purpose: Alter CSS attribute
        - Input: HTML Element, value
        - Output: CSS refresh
        - Parameters: Object, value
    - render_HTML_template
        - Purpose: render HTML
        - Input: HTML & CSS file to render
        - Output: HTML page
        - Parameters: object

7.1.4.2.   HTMLElementTemplateFactory

- Description
- Data members

Data Type	Data Name	Access Modifiers	Initial Value	Description
string	name	Private	None	Name of the element
string	id	Private	None	ID of the element
tuple	coords	Private	None	Position of the element

Table 7.5: Data Members 5

- Methods: cast_to_<element>
  - Purpose: place HTML elements on page
  - Input: Element, type of element
  - Output: New element
  - Parameters: Object, id, className

7.1.4.3.   HTMLDocument

- Description
- Data members

Data Type	Data Name	Access Modifiers	Initial Value	Description
string	name	Private	None	Name of the element
string	id	Private	None	ID of the element
tuple	coords	Private	None	Position of the element

Table 7.6: Data Members 6

- Methods: constructor
  - Purpose: initialise object of the class
  - Input: object attributes if any
  - Output: Object
  - Parameters: Attributes

7.1.4.4.   HTML<Element>

- Description

● Data members

Data Type	Data Name	Access Modifiers	Initial Value	Description
integer	id	Private	None	ID of the element
string	HTMLid	Private	None	HTML ID of the element
string	classname	Private	None	HTML class of the element
string	name	Private	None	Element name

Table 7.7: Data Members 7

● Methods

   ○ json_rep

     ● Purpose: Represent attributes in JSON format

     ● Input: HTML element

     ● Output: JSON object

     ● Parameters: Object

7.1.5.   Resets.py

● Description

● Data members

Data Type	Data Name	Access Modifiers	Initial Value	Description
string	<file>_paths	Public	None	Relative paths to all files in the server

Table 7.8: Data Members 8

● Methods

   ○ Reset_header

     ● Purpose: reset HTML headers

     ● Input: HTML header code

     ● Output: HTML file

     ● Parameters: None

- ○ server_reset
  - Purpose: reset flask server
  - Input: Conent.html, index.css, element structure, metadata
  - Output: Server reset
  - Parameters: None
- ○ rewrite_css
  - Purpose: update CSS params from user
  - Input: user input
  - Output: index.css
  - Parameters: data
- ○ download_file
  - Purpose: download generated HTML file
  - Input: None
  - Output: HTML file
  - Parameters: None

8. IMPLEMENTATION AND PSEUDOCODE

8.1. Data Creation

All data had to be created manually by the team. Each of us drew approximately 300 to 400 samples for each element. Each sample was further rotated through right angles, and flipped, to increase the size of the dataset by six times.

8.2. Flask Server

First, custom made modules for object detection and classification are imported

```python
from flask import Flask, request, render_template, jsonify
from modules.objectdetection import split_images
from modules.SVM import make_prediction
import os
from PIL import Image
import json
from modules.resets import
server_reset,reset_header,rewrite_css,download_file


app = Flask(__name__, static_url_path='')


app.config['UPLOAD_FOLDER'] = 'sketches'
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0
```

Default route to call index.html, i.e, the main page of the UI with basic styles

```python
@app.route('/')
def root():
    return render_template('index.html')


@app.route('/getcss', methods=['GET'])
def get_css():
    with open(os.path.join("metadata", "element_structure.json"), 'r') as f:
        return json.load(f)
```

To update the generated page with the new CSS

```python
@app.route('/modify', methods=['POST'])
def modify_css():
    json_str = request.get_json()
    rewrite_css(json_str)
    return render_template("generatedpage.html")
```

To clear/delete the uploaded image

```python
@app.route('/delete', methods=['GET'])
def delete_page():
    for name, value in globals().copy().items():
        print(name, value)
    reset_header()
    server_reset()
    return "Page was cleared"
```

To clear any existing data, and accept the uploaded input image

```python
@app.route('/sendfile', methods=['POST'])
def get_file():
    server_reset()
    latestfile = request.files['uploaded-file']
    full_filename = os.path.join(app.config['UPLOAD_FOLDER'], "newimage.jpg")
    latestfile.save(full_filename)
    split_images()
    make_prediction()
    return 'File uploaded successfully'
```

To return a preview of the generated HTML page

```python
@app.route('/generatedpage')
def get_page():
    return render_template("generatedpage.html")
```

To download the final version of the HTML file

```python
@app.route("/download")
def download():
    download_file()
    return render_template("generatedpage.html")



if __name__ == "__main__":
    server_reset()
    app.run(debug=True)
```

8.3.   Object Detection Algorithm

```python
import cv2
from PIL import Image
import os
import json
import pickle
from . import HTMLFactory


id_gen = 0
HTML_objects = []
```

Detecting and cropping objects

```python
def create_crops(img, page, coords=None):

    global id_gen,HTML_objects

    if page:
        image = cv2.imread(img)
    else:
        image = cv2.imread(img)
        image = image[coords[1]:coords[3], coords[0]:coords[2]]
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (3, 3), 0)
    canny = cv2.Canny(blurred, 120, 255, 1)
```

```
    cnts = cv2.findContours(canny, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    cnts = cnts[0] if len(cnts) == 2 else cnts[1]
    for c in cnts:
        peri = cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, 0.015 * peri, True)
        if len(approx) == 4:
            x, y, w, h = cv2.boundingRect(approx)
            cv2.rectangle(image, (x, y), (x+w, y+h), (36, 255, 12), 2)
            advanced_coords = [x, y, x+w, y+h]
            if page:
                im = Image.open(os.path.join(os.path.join(
                    os.path.dirname(__file__), '..', 'sketches',
'newimage.jpg')))
                im1 = im.crop(advanced_coords)
                im1.save(os.path.join(os.path.join(os.path.dirname(
                    __file__), '..', 'samples', '0.jpg')))
                HTML_objects.append(HTMLFactory.HTMLDocument(
                    advanced_coords, str(id_gen)))
                id_gen += 1
                return advanced_coords

            else:
                im = Image.open(os.path.join(os.path.join(
                    os.path.dirname(__file__), '..', 'samples', '0.jpg')))
                im1 = im.crop([advanced_coords[0], advanced_coords[1],
                        advanced_coords[2]+40, advanced_coords[3]+40])
                print(advanced_coords)
                newsize = (200, 200)
                HTML_objects.append(HTMLFactory.HTMLElementTemplateFactory(
                    str(id_gen), advanced_coords, str(id_gen)))
                im1 = im1.resize(newsize)
                im1.save(os.path.join(os.path.join(os.path.dirname(
                    __file__), '..', 'samples', str(id_gen)+'.jpg')))
                id_gen += 1

    dbfile = open(os.path.join(os.path.join(
```

```
        os.path.dirname(__file__), '..', 'metadata', 'metadata.pkl')), 'wb')
    pickle.dump(HTML_objects, dbfile)
    dbfile.close()
    id_gen = 0
    HTML_objects = []
```

Preprocessing the input image

```
def preprocessor():
    col = Image.open(os.path.join(os.path.join(
        os.path.dirname(__file__), '..', 'sketches'), "newimage.jpg"))
    gray = col.convert('L')
    bw = gray.point(lambda x: 0 if x < 128 else 255, '1')
    # eroded = cv2.erode(bw,kernel,iterations = 1)
    # opened = cv2.morphologyEx(bw, cv2.MORPH_OPEN, kernel)
    # cv2.imwrite('eroded_image.jpg', eroded)
    # cv2.imwrite('opened_image.jpg', opened)
    bw.save(os.path.join(os.path.join(os.path.dirname(
        __file__), '..', 'sketches'), "newimage.jpg"))
```

Splitting and saving cropped images

```
def split_images():
    preprocessor()
    coords = create_crops(os.path.join(os.path.join(
        os.path.dirname(__file__), '..', 'sketches', 'newimage.jpg')), True)
    coords = [coords[0]+20, coords[1]+20, coords[2]-20, coords[3]-20]
    create_crops(os.path.join(os.path.join(os.path.dirname(
        __file__), '..', 'sketches', 'newimage.jpg')), False, coords)
```

8.4.   Training the Support Vector Classifier

```
import pickle
from pathlib import Path
import matplotlib.pyplot as plt
import numpy as np
```

```python
from sklearn import svm as sksvm, metrics, datasets
from sklearn.utils import Bunch
from sklearn.model_selection import GridSearchCV, train_test_split
import skimage
from skimage.io import imread
from skimage.transform import resize
import cv2
from PIL import Image
import os
import time
from colorama import Fore, Back, Style
import pickle
from . import HTMLFactory
import statistics
import json


class SVMfactory:

    mapper = {0: "Button", 1: "Checkbox", 2: "Image", 3: "Input", 4: "Video"}
    JSON_dict = {}
```

```python
    def __init__(self, loading=False,
img_dir=os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'HTMLElements'))):

        self.STATIC_PATH = None

        if not loading:
            self.image_dataset = self.load_image_files(img_dir)
            print(Fore.GREEN + "Images have been loaded")
        self.param_grid = [
            {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
            {'C': [1, 10, 100, 1000], 'gamma': [
                0.001, 0.0001], 'kernel': ['rbf']},
        ]

        self.X_train = self.X_test = self.y_train = self.y_test = []

        self.y_pred = self.clf = None
```

```python
        self.dimension = (64, 64)
        self.HTML_element_list = []


    def load_image_files(self, container_path, dimension=(64, 64)):
        image_dir = Path(container_path)
        folders = [directory for directory in image_dir.iterdir()
                   if directory.is_dir()]
        categories = [fo.name for fo in folders]
        descr = "A image classification dataset"
        images = []
        flat_data = []
        target = []
        for i, direc in enumerate(folders):
            for file in direc.iterdir():
                img = skimage.io.imread(file)
                img_resized = resize(
                    img, dimension, anti_aliasing=True, mode='reflect')
                flat_data.append(img_resized.flatten())
                images.append(img_resized)
                target.append(i)
        flat_data = np.array(flat_data)
        target = np.array(target)
        images = np.array(images)


        return Bunch(data=flat_data,
                     target=target,
                     target_names=categories,
                     images=images,
                     DESCR=descr)

    def test_train_split(self, test_size=0.3, random_state=109):


        self.X_train, self.X_test, self.y_train, self.y_test =
train_test_split(self.image_dataset.data,

self.image_dataset.target,

test_size=test_size,
```

```python
random_state=random_state)

        print("Test train split has been completed =>")
        print("Training parameters = %s" % (len(self.X_train)))
        print("Testing parameters = %s" % (len(self.X_test)))

    def train_and_fit(self):
        print("Model will now begin training. This could take a while")
        start_time = time.time()
        svc = sksvm.SVC()
        self.clf = GridSearchCV(svc, self.param_grid)
        self.clf.fit(self.X_train, self.y_train)
        self.y_pred = self.clf.predict(self.X_test)
        print("SVM training is completed. \n Time elapsed =>",
                (time.time() - start_time))

    def display_svm_scores(self):

        print("Classification report for - \n{}:\n{}\n".format(self.clf,
metrics.classification_report(self.y_test, self.y_pred)))

    def image_process(self, image):

        hsv_image = cv2.imread(image, 1)  # pretend its HSV
        #noise_img = sp_noise(hsv_image,0.2)
        rgbimg = cv2.cvtColor(hsv_image, cv2.COLOR_HSV2RGB)
        image_gray = cv2.cvtColor(rgbimg, cv2.COLOR_BGR2GRAY)
        _, threshold = cv2.threshold(image_gray, 127, 255, 0)
        plt.imshow(image_gray, cmap='gray')

        contours, hierarchy = cv2.findContours(
            threshold, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

        left_arr = []
        right_arr = []
        top_arr = []
        bottom_arr = []
```

```python
        for cnt in contours:
            leftmost = tuple(cnt[cnt[:, :, 0].argmin()][0])
            rightmost = tuple(cnt[cnt[:, :, 0].argmax()][0])
            topmost = tuple(cnt[cnt[:, :, 1].argmin()][0])
            bottommost = tuple(cnt[cnt[:, :, 1].argmax()][0])
            left_arr.append(leftmost)
            right_arr.append(rightmost)
            top_arr.append(topmost)
            bottom_arr.append(bottommost)


        left_arr.sort()
        right_arr.sort(reverse=True)
        top_arr.sort(key=lambda x: x[1])
        bottom_arr.sort(reverse=True, key=lambda x: x[1])


        left_arr.pop(0)
        right_arr.pop(0)
        top_arr.pop(0)
        bottom_arr.pop(0)


        advanced_coords = (left_arr[0][0]-10, top_arr[0]
                            [1]-10, right_arr[0][0]+10, bottom_arr[0][1]+10)
        im = Image.open(image)
        im1 = im.crop(advanced_coords)
        newsize = (200, 200)

        im1 = im1.resize(newsize)
        os.remove(image)
        im1.save(image)


    def predict_images(self):
        file_name = []
        HTMLobject_list = []
        samp_data = []
        fp = open(os.path.join(os.path.join(
            os.path.dirname(__file__), '..', 'metadata', 'metadata.pkl')),
"rb")
        HTMLobject_list = pickle.load(fp)
        parent = HTMLobject_list.pop(0)
```

```python
        if direction == "width":
            range_blocker = [((i.w-i.x1), i)
                                for i in self.HTML_element_list]
        elif direction == "height":
            range_blocker = [((i.h-i.y1), i)
                                for i in self.HTML_element_list]
        elif direction == "top":
            range_blocker = [((i.top_offset), i)
                                for i in self.HTML_element_list]
        elif direction == "left":
            range_blocker = [((i.x1), i)
                                for i in self.HTML_element_list]


        range_blocker.sort(key=lambda x: x[0])
        clusters = []
        temp_clust = []
```

```python
        for i in range(1, len(range_blocker)):
            if (range_blocker[i][0] - range_blocker[i-1][0]) < threshold:
                temp_clust.append(range_blocker[i])
                temp_clust.append(range_blocker[i-1])
            else:
                if len(temp_clust) > 0:
                    clusters.append(temp_clust)
                    temp_clust = []
                    temp_clust.append(range_blocker[i])
                else:
                    clusters.append([range_blocker[i-1]])
                    temp_clust.append(range_blocker[i])


    clusters.append(temp_clust)
    clusters = [i for i in clusters if len(i) > 0]
    clusters = [i[0] for i in clusters]
    cnt = 0
    for i in self.HTML_element_list:


        min_diff = 100000
        min_obj = None
        min_w = None
        for j in clusters:
            if direction == "width":
                if abs(j[0] - (i.w-i.x1)) < min_diff:
                    min_diff = abs(j[0] - (i.w-i.x1))
                    min_obj = j[1]
                    min_w = j[0]
            elif direction == "height":

                if abs(j[0] - (i.h-i.y1)) < min_diff:
                    min_diff = abs(j[0] - (i.h-i.y1))
                    min_obj = j[1]
                    min_w = j[0]
            elif direction == "top":
                if abs(j[0] - (i.top_offset)) < min_diff:
                    min_diff = abs(j[0] - (i.top_offset))
                    min_obj = j[1]
```

_____

```python
                    min_w = j[0]
            elif direction == "left":
                if abs(j[0] - (i.x1)) < min_diff:
                    min_diff = abs(j[0] - (i.x1))
                    min_obj = j[1]
                    min_w = j[0]
        cnt += 1


        if direction == "width":
            i.attach_new_width(min_w)
        elif direction == "height":
            i.attach_new_height(min_w)
        elif direction == "top":
            i.attach_new_top(min_obj)
        elif direction == "left":
            i.attach_new_left(min_obj)


    def write_as_json(self):


        with open(os.path.abspath(os.path.join(os.path.dirname(
                __file__), '..', 'metadata', 'element_structure.json')), 'w')
as f:

            json.dump(SVMfactory.JSON_dict, f)
        SVMfactory.JSON_dict = {}


    def render_setup(self, parent):
        self.fix_new_position("width", 50)
        self.fix_new_position("height", 20)
        self.fix_new_position("top", 20)
        self.fix_new_position("left", 20)
        for i in self.HTML_element_list:
            i.set_css(parent)
            SVMfactory.JSON_dict.update(i.json_rep())
            i.render_HTML_template()
        self.write_as_json()
```

```python
def save_model(self):
    try:
        with open(os.path.abspath(os.path.join(os.path.dirname(__file__),
'..', 'models', 'SVM.pkl')), 'wb') as fid:
            pickle.dump(self.clf, fid)
            print("Model has been saved")
    except Exception as e:
        print("Error occured")
        print(e)


def load_model(self):
    with open(os.path.join(os.path.join(os.path.dirname(__file__), '..',
'models', 'SVM.pkl')), 'rb') as f:
        self.clf = pickle.load(f)
        print(Fore.GREEN + "Model has been loaded")
```

8.5.   Support Vector Classifier

```python
def make_prediction():
    try:
        svm = SVMfactory(True)
        svm.load_model()
    except:
        print("Error occurred, model does not exist or is damaged. Preparing
new model...")
        svm = SVMfactory()
        svm.test_train_split()
        svm.train_and_fit()
        svm.display_svm_scores()
        svm.save_model()
        make_prediction()

    print("Model was loaded successfully")
    svm.predict_images()
```

8.6.   Html Factory

```python
import os
import abc
import json


class HTMLFactory(object):

    prev_left = None
    prev_top = None


    def __init__(self, coords):

        self.name = None
        self.x1, self.y1, self.w, self.h = coords
        self.x2 = self.x1+self.w
        self.y2 = self.y1+self.h
        self.top_offset = self.y1
        self.backgroundColor = "auto"
        self.color = "auto"
        self.position = "absolute"
        self.left = "auto"
        self.right = "auto"
        self.top = "auto"
        self.bottom = "auto"
        self.width = "auto"
        self.height = "auto"
        self.document = None


    def attach_new_width(self, width):
        self.w = width


    def attach_new_height(self, height):
        self.h = height


    def attach_new_top(self, obj):
        self.top_offset = obj.top_offset
```

```python
    def attach_new_left(self, obj):
        self.x1 = obj.x1


    def view_coordinates(self):
        return "W : %d" % (self.w)


    def set_backgroundColor(self, backgroundColor):
        self.backgroundColor = backgroundColor


    def set_color(self, color):
        self.color = color


    def set_position(self, position):
        self.position = position


    def set_left(self, left):
        self.left = left


    def set_top(self, top):
        self.top = top


    def set_bottom(self, bottom):
        self.bottom = bottom


    def set_width(self, width):
        self.width = width



def set_height(self, height):
    self.height = height

@abc.abstractmethod
def html_template(self):
    pass

@abc.abstractmethod
def css_template(self):
    pass
```

```python
    def render_HTML_template(self):
        fp = open(os.path.abspath(os.path.join(os.path.dirname(
            __file__), '..', 'templates', 'content.html')), "a+")
        fp.write(self.html_template())
        fp.close()
        fp = open(os.path.abspath(os.path.join(os.path.dirname(
            __file__), '..', 'static', 'styles',    'index.css')), "a+")
        fp.write(self.css_template().replace(" ", "").replace("\n", ""))
        fp.close()


    def set_css(self, parent):
        self.width = str(int((int(self.w) / int(parent.w))*100) - 3) + "%"
        self.height = str(int(((int(self.h) / int(parent.h))*100)) - 2) + "%"
        self.left = str(int((int(self.x1) / (int(parent.w))) * 100) + 10) +
"%"

        self.top = str(
            int((int(self.top_offset) / int(parent.h)) * 100) + 8) + "%"


    def get_css(self):
        return dict(self.css_template().replace(" ", "").replace("\n",
"").replace("{", ":{"))


    def convert_json_to_css(self):
        pass
```

Returning the HTML Templates

```python
class HTMLElementTemplateFactory:

    def __init__(self, name, coords, id):
        self.name = name
        self.id = id
        self.coords = coords


    def cast_to_image(self, HTMLid, className):
        return HTMLImage(HTMLid, className, self.coords, self.id, self.name)


    def cast_to_input(self, HTMLid, className):
```

```python
        return HTMLInput(HTMLid, className, self.coords, self.id, self.name)

    def cast_to_checkbox(self, HTMLid, className):
        return HTMLCheckBox(HTMLid, className, self.coords, self.id,
self.name)

    def cast_to_button(self, HTMLid, className):
        return HTMLButton(HTMLid, className, self.coords, self.id, self.name)

    def cast_to_video(self, HTMLid, className):
        return HTMLButton(HTMLid, className, self.coords, self.id, self.name)
```

Child Classes of HTML Factory

```python
class HTMLDocument(HTMLFactory):

    def __init__(self, coords, id):
        self.id = id
        self.x1, self.y1, self.w, self.h = coords
        self.x2 = self.x1+self.w
        self.y2 = self.y1+self.h
        HTMLFactory.__init__(self, coords)


class HTMLInput(HTMLFactory):

    def __init__(self, HTMLid, className, coords, id, name):

        self.id = id
        self.HTMLid = HTMLid
        self.className = className
        self.name = name
        HTMLFactory.__init__(self, coords)

    def html_template(self):
        return '''
                <div class="''' + self.className + '''" >
```

_____

```python
                    <input type="text"   id="input''' + self.HTMLid + '''"
class="form-control" placeholder="Default input">
                </div>
                '''


    def css_template(self):
        # return "#{}\{ position : {}; left : {}; right : {}; top : {};
bottom : {}; width : {}; height : {}; backgroundColor : {}; color : {}
\}".format(self.HTMLid,self.position,self.left,self.right,self.top,self.botto
m,self.width,self.height,self.backgroundColor,self.color)
        return f'''#input{self.HTMLid}{{
            position : {self.position};
            left : {self.left};
            right : {self.right};
            top : {self.top};
            bottom : {self.bottom};
            width : {self.width};
            height : {self.height};
            background-color : {self.backgroundColor};
            color : {self.color};
        }}\n'''



    def json_rep(self):
        return {"#input"+self.HTMLid:
                {
                    "position": self.position,
                    "left": self.left,
                    "right": self.right,
                    "top": self.top,
                    "bottom": self.bottom,
                    "width": self.width,
                    "height": self.height,
                    "backgroundColor": self.backgroundColor,
                    "color": self.color,
                }
                }
```

_____

```python
class HTMLImage(HTMLFactory):

    def __init__(self, HTMLid, className, coords, id, name):
        self.name = name
        self.id = id
        self.HTMLid = HTMLid
        self.className = className


        HTMLFactory.__init__(self, coords)


    def html_template(self):
        return '''
                <div class="container">
                    <img class="''' + self.className + '''" src={{
url_for('static', filename = 'images/placeholder.png') }} id="image''' +
self.HTMLid + '''" >
                </div>
                '''


    def css_template(self):
        # return '''#{}{ position : {}; left : {}; right : {}; top : {};
bottom : {}; width : {}; height : {}; backgroundColor : {}; color : {}
}'''.format(self.HTMLid,self.position,self.left,self.right,self.top,self.bott
om,self.width,self.height,self.,self.color)
        return f'''#image{self.HTMLid}{{
            position : {self.position};
            left : {self.left};
            right : {self.right};
            top : {self.top};
            bottom : {self.bottom};
            width : {self.width};
            height : {self.height};
            background-color : {self.backgroundColor};
            color : {self.color};
        }}\n'''


    def json_rep(self):
        return {"#image"+self.HTMLid:

                {
```

```python
                "position": self.position,
                "left": self.left,
                "right": self.right,
                "top": self.top,
                "bottom": self.bottom,
                "width": self.width,
                "height": self.height,
                "backgroundColor": self.backgroundColor,
                "color": self.color,
            }
        }


class HTMLCheckBox(HTMLFactory):

    def __init__(self, HTMLid, className, coords, id, name):
        self.name = name
        self.id = id
        self.HTMLid = HTMLid
        self.className = className


        HTMLFactory.__init__(self, coords)



    def html_template(self):
        return '''
                <div class="''' + self.className + '''" >
                    <input type="checkbox"   id="checkbox''' + self.HTMLid +
'''" placeholder="Default input">
                </div>
                '''

    def css_template(self):
        # return "#{}\{ position : {}; left : {}; right : {}; top : {};
bottom : {}; width : {}; height : {}; backgroundColor : {}; color : {}
\}".format(self.HTMLid,self.position,self.left,self.right,self.top,self.botto
m,self.width,self.height,self.backgroundColor,self.color)
        return f'''#checkbox{self.HTMLid}{{
            position : {self.position};
```

```
            left : {self.left};
            right : {self.right};
            top : {self.top};
            bottom : {self.bottom};
            width : {self.width};
            height : {self.height};
            background-color : {self.backgroundColor};
            color : {self.color};
        }}\n'''

    def json_rep(self):
        return {"#checkbox"+self.HTMLid:
                {
                    "position": self.position,
                    "left": self.left,
                    "right": self.right,
                    "top": self.top,
                    "bottom": self.bottom,
                    "width": self.width,
                    "height": self.height,
                    "backgroundColor": self.backgroundColor,
                    "color": self.color,
                }
                }


class HTMLButton(HTMLFactory):

    def __init__(self, HTMLid, className, coords, id, name):
        self.name = name
        self.id = id
        self.HTMLid = HTMLid
        self.className = className

        HTMLFactory.__init__(self, coords)

    def html_template(self):
        return '''
```

_____

```python
                <div class="''' + self.className + '''" >
                    <input type="button"    id="button''' + self.HTMLid + ''''"
class="btn btn-primary" value="Button">
                </div>
                '''


    def css_template(self):
        # return "#{}\{ position : {}; left : {}; right : {}; top : {};
bottom : {}; width : {}; height : {}; backgroundColor : {}; color : {}
\}".format(self.HTMLid,self.position,self.left,self.right,self.top,self.botto
m,self.width,self.height,self.backgroundColor,self.color)
        return f'''#button{self.HTMLid}{{
            position : {self.position};
            left : {self.left};
            right : {self.right};
            top : {self.top};
            bottom : {self.bottom};
            width : {self.width};
            height : {self.height};
            background-color : {self.backgroundColor};
            color : {self.color};
        }}\n'''



    def json_rep(self):
        return {"#button"+self.HTMLid:
                {
                    "position": self.position,
                    "left": self.left,
                    "right": self.right,
                    "top": self.top,
                    "bottom": self.bottom,
                    "width": self.width,
                    "height": self.height,
                    "backgroundColor": self.backgroundColor,
                    "color": self.color,
                }
                }
```

_____

```python
def build_elements(element, cast):

    if element == "Input":
        return cast.cast_to_input(cast.id, "form-group")
    elif element == "Image":
        return cast.cast_to_image(cast.id, "")
    elif element == "Checkbox":
        return cast.cast_to_checkbox(cast.id, "")
    elif element == "Button":
        return cast.cast_to_button(cast.id, "")
    elif element == "Video":
        return cast.cast_to_video(cast.id, "")
```

8.7. Resets

```python
import os
import json

style_path = os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'static','styles'))
template_path = os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'templates'))
metadata_path = os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'metadata'))
samples_path = os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'samples'))

def reset_header():

    fdata = '''<!DOCTYPE html>
<!--[if lt IE 7]>        <html class="no-js lt-ie9 lt-ie8 lt-ie7"> <![endif]-->
<!--[if IE 7]>          <html class="no-js lt-ie9 lt-ie8"> <![endif]-->
<!--[if IE 8]>          <html class="no-js lt-ie9"> <![endif]-->
<!--[if gt IE 8]><!-->
<html class="no-js">
<!--<![endif]-->
```

_____

```
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title></title>
    <meta http-Equiv="Cache-Control" Content="no-cache" />
    <meta http-Equiv="Pragma" Content="no-cache" />
    <meta http-Equiv="Expires" Content="0" />
    <meta name="description" content="">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link
href="https://stackpath.bootstrapcdn.com/bootswatch/4.4.1/simplex/bootstrap.m
in.css" rel="stylesheet"
integrity="sha384-cRAmF0wErT4D9dEBc36qB6pVu+KmLh516IoGWD/Gfm6FicBbyDuHgS4jmkQ
B8u1a" crossorigin="anonymous">
    <style>
        #main-container {
            position: absolute;
            top: 0px;
            left: 0px;
            right: 0px;
            bottom: 0px;
            width: 100%;
            height: 100%;
            display: flex;
            justify-content: center;
            align-items: center;
        }

        #cssplaceholder{}

    </style>
</head>

<body>
    <div id="main-container">'''
```

_____

```python
        fp = open(os.path.join(template_path,"header.html"),"w")
        fp.write(fdata)
        fp.close()

def server_reset():
        fp = open(os.path.join(template_path, "content.html"), "w")
        fp.close()
        fp = open(os.path.join(style_path, "index.css"), "w")
        fp.close()
        fp = open(os.path.join(metadata_path, "element_structure.json"), "w")
        fp.close()
        fp = open(os.path.join(metadata_path, "metadata.pkl"), "wb")
        fp.close()
        for i in os.listdir(samples_path):
            os.remove(os.path.join(samples_path, i))
```

_____

```python
def rewrite_css(data):
    fp = open(os.path.join(style_path, "index.css"), "r")
    fp_content = fp.read().replace("}", "} ").split()
    fp.close()
    reppos = -1
    for i in enumerate(fp_content):
        if data["ele"] in i[1]:
            reppos = i[0]
    ele = data["ele"]
    del data["ele"]
    fp_content[reppos] = ele+json.dumps(data).replace(",", ";").replace(
        " ", "").replace('"', '').replace("backgroundColor",
"background-color")
    with open(os.path.join(metadata_path, "element_structure.json"), 'r') as
f:
        json_css = json.load(f)
        for i in data:
            json_css[ele][i] = data[i]
    with open(os.path.join(metadata_path, "element_structure.json"), 'w') as
f:
        json.dump(json_css, f)
    fp = open(os.path.join(style_path, "index.css"), "w")
    fp.write("".join(fp_content))
    fp.close()

def download_file():
    reset_header()

    fp = open(os.path.join(template_path,"header.html"),"r")
    files = fp.readlines()
    fp.close()
    fp = open(os.path.join(style_path,"index.css"),"r")
    css = fp.read()
    fp.close()
    ind = 0
    for i in enumerate(files):
        if "cssplaceholder" in i[1]:
            ind = i[0]
            break
```

_____

```
    files[ind] =
"\t\t"+css.replace("{","{\n\t\t\t").replace(";",";\n\t\t\t").replace("}","}\n
\t\t")


    fp = open(os.path.join(template_path,"header.html"),"w")
    fp.writelines(files)
    fp.close()
```

8.8.   Testing

```python
from selenium.webdriver import Chrome
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException

driver = Chrome(executable_path='C:\Program Files (x86)\Chrome
Driver\chromedriver.exe')
# exec(open("server.py").read())
driver.get("localhost:5000")

buttons = driver.find_elements_by_tag_name('button')
inputs = driver.find_elements_by_tag_name('input')

upload = driver.find_element_by_id('file-btn')
upload.send_keys("C:\\Users\\Prerna\\Downloads\\Samples\\sample21.jpg")
slide = driver.find_element_by_id('overlay-iframe')
slide.click()

convert = driver.find_element_by_id('sbt')
convert.click()

iframe = driver.find_element_by_tag_name("iframe")
driver.switch_to.frame(iframe)
timeout = 10
try:
```

```python
    element_present = EC.presence_of_element_located((By.ID, 'input3'))
    WebDriverWait(driver, timeout).until(element_present)
except TimeoutException:
    print("Timed out waiting for page to load")
finally:
    print("Page loaded")


example_element = driver.find_element_by_tag_name('input')
example_element.click()
driver.switch_to.default_content()
modify = driver.find_element_by_id('mod-btn')
modify.click()
print('modified')


driver.implicitly_wait(60) # seconds


preview = driver.find_element_by_id('previewframe')
preview.click()
print('previewed')



driver.implicitly_wait(30) # seconds
driver.switch_to.window(driver.window_handles[0])
download = driver.find_element_by_id('down-btn')
download.click()
print('downloading')
driver.implicitly_wait(60) # seconds
print('downloaded')


driver.implicitly_wait(60) # seconds


delete = driver.find_element_by_id('del-btn')
delete.click()
print('deleted')
driver.implicitly_wait(60) # seconds
driver.quit()
```

_____

9. TESTING

9.1. Scope

- All modules of the project will be tested manually
- UI testing will be automated.

9.2. Strategies, Roles, and Responsibilities

A proactive testing strategy was followed by all team members. In this approach, the testing process is initiated alongside the development process so as to detect and fix the defects before the build is created. In doing so, bugs were detected at the earliest and fixed immediately.

9.3. Test Tools Used

The test tool used for this project is Selenium.

It is used for automated web application testing across different browsers.

It has 4 components: IDE, RC, WebDriver Grid.

The Selenium WebDriver has been chosen for testing the UI for this web application.

This is an automation framework for the web that allows test execution against different browsers. It also allows usage of a programming language to create the testing scripts.

_____

10.   RESULTS AND DISCUSSION

10.1.   Exploratory Analysis

The origins of the idea are very organic as there are few applications like this one available in the market. Outside of one or two papers, it was extremely difficult to find documentation or methodology to follow. Hence, the procedure of working on and building this product was based on intuition and innovation.

The main source of inspiration was Microsoft's own Sketch2Code. As such, it was natural to start browsing from there, however, the product is no longer displayed on the website.

Beyond that, the rest of the videos were small videos from Twitter and other social media. Therefore, as many videos and samples as possible were found and explored before trying to find a novel way of implementing it.

10.2.   Problems that led to preprocessing ideas

- The main issue associated with pre-processing was detecting the objects themselves. Since there was no dataset available and the expected route for data gathering was through crowdsourcing, the following ideas were developed.
- The first was anticipated that people would draw however they wanted, wherever they wanted. It was expected that photos of the drawings, as well as the sample data required for training, would not be perfectly aligned, or straight. Therefore, it was necessary to create a cropping algorithm that would cut out only the required image, this would also be a proof of concept that if this did work, it would be extendable to the rest of the project. Therefore, it was elected to get bounding boxes for all elements including the border of the image
  This completed the first stage of pre-processing.

_____

- The second stage involved the actual detection of the rectangular bounding boxes. The performance was impacted by lines of different color, the brightness of the surroundings and various degrees of saturation. To combat this, all images are grayscaled on upload. While this fixed a lot of the problems, the gradients still caused detection errors. A combination of the OTSU filter and binary conversion of the image into only black and white enabled us to improve the quality of detection.

- This still left the problem of actually detecting the rectangles. It was fair to assume that people could not draw straight lines perfectly, especially long lines and so the contour values had to be tweaked further to incorporate bent lines, curves and varying thicknesses. This still reached a limit where lines angled at anything nearing 30deg would not be considered. This problem would finally be addressed with the addition of Canny Edge detection which would finally improve detection tremendously.

- The last issue, while not pressing, was the degree of cropping required. Some breathing room would have to be provided to prevent the image from being close-cropped. For this, a padding of 40px was provided to images scaled to 200sqpx.

- Finally, to boost the quantity of data, all images were rotated and flipped along their axes (where possible).

10.3.   Why SVM

SVMs or Support Vector Machines' sole goal is the separation of data into classes , the process known as 'classification'. However, this doesn't explain why a CNN wouldn't perform equally well, especially since CNN is built to work exclusively with images.

The answer is very direct: data.

A CNN requires over a thousand images, closer to ten thousand images to perform efficiently, whereas an SVM can get by with a fraction of the data. A few assumptions and contrasting features were noted, as a CNN was with the limited data possessed, only to watch the CNN fail spectacularly.

The reasons for choosing the SVM over a CNN were:

1. While crowdsourcing was planned initially, this proved to be inconvenient, and the decision to stick to the less data-intensive models was taken due to the unique situation put in front of us.

2. The images were not complicated and could not be found in many orientations. A CNN is extremely special in that it can detect objects like dogs and cats with incredible accuracy even if they are contorted into various positions or angles, however, the images were extremely basic and would not vary much; if at all. Hence the SVM would not be required to "see" things in odd orientations and would not lose out here.

3. CNN's are difficult to train and extremely taxing even on high-end machines. While sources like Colab exist, that would still mean looking at several hours a day to train (provided the model worked properly and no errors occurred). The SVM on the other hand would not slow us down as much, being much lighter and faster to train as well as allowing us to quickly add revisions if necessary. With lesser data required and more speed in training, the SVM was a better choice.

4. The normal counterpoint to the SVM is the lack of multiclass detection. However this was circumvented through the use of the famous "kernel trick", wherein the data is projected onto a higher dimension to allow selection of better hyperplanes for multiple classes. This would at the very least equally match the CNN in terms of multiclass quality detection.

———————

10.4.   The model

- Being an SVM, the structure is straightforward. The addition of the kernel trick was simple due to past experience of using the SVM and a plentiful of resources present to find a good implementation of the SVM.

- This project uses GridSearchCV which is an exhaustive search over all the parameter values for a given estimator. This leads to an optimized result produced by the cross validation grid search on a parameter grid.

- For linear searches, this project uses the linear kernel (Obviously, as that's pretty much the default kernel for 2D)

- Considering SVMs perform best with RBF, this application uses RBF only. Polynomial kernels are usually better suited to NLP operations which were not relevant.

- The dimensions used were standard 64x64 and while considering 256x256 for the images, it was eventually settled on 200x200 as per recommendations from the literature survey and other sources.

- GridSearchCV also allowed us to loosely pass an array of C values (1-1000) in exponents of 10 and gamma values of 0.001 and 0.0001 respectively.

- Test size was chosen personally by us, despite the recommended amount being around 15% as per the literature survey, this project considered up to 30% just for a better and more thorough testing.

10.5.   Usefulness

With such an application at hand it is extremely convenient to quickly build workable front-ends and prototype web pages without prior knowledge of HTML/CSS. With a set of

basic rules, and basic intuition it is simple to draw an image/wireframe on any suitable platform, and have the result generated in a matter of minutes.

The SVM is the crux of the project. The whole project relies on the ability of the SVM to accurately detect and predict the class of the image.

The SVM is accurate and requires very less time to retrain for new classes. This gives us the confidence to work towards the rest of the project as well as include future improvements to it.

Considering the data used in this project is custom, the algorithm cannot be replaced, making it a unique application.

10.6.   Why This Solution is Better

- There are not a lot of solutions, to begin with, there were limited sources to compare to. However, from what was looked at, it was seen that :
- This solution could easily be extended with just a small folder of extra data
- CSS can be improved at the front-end if not satisfied
- Pages are responsive
- Software is free and open-source
- Pages can be downloaded once ready and used immediately.

10.7.   Project Learnings

10.7.1.   Techniques

- Inheritance and OOPs,
- Clustering ideas and algorithms,
- Processing of images and reasons
- Sharing data between different HTML windows
- Cache-control
- Element positioning, responsiveness, and 2D geometry

- GridSearch
- Canny edge detection
- Binary conversion of images
- Skeletonization
- Contours detection
- Manipulating channels in images
- Various filters used in image processing

10.7.2. Software

- Selenium
- OpenCV
- Pillow
- Jinja / Flask
- jQuery animations
- CSS Keyframes
- Flexbox
- Py-lint

10.7.3. Issues Faced

- Detection of boundaries and tweaking parameters to find ideal values for the boundaries
- Selection of parameters for SVM and testing the best possible values. Testing out a CNN to realize the severe lack of data
- Having a lack of data and the crowdsourcing option removed due to unforeseen circumstances.
- Shape for the SVM and figuring out how to manipulate or remove channels in images.
- Creating data manually as well as finding ways to improve data quality and quantity

- Mathematics involved with positioning and calculating width of tags and shapes
- Manipulation of JSON data multiple times in multiple ways at multiple interfaces
- Handling iframes with Selenium where the click event occurs on the parent page instead of the child.
- Adding click functionality into the iframe and gathering data.
- Making sure pages don't cache and reuse old CSS, involved a lot of cache control.
- Lack of useful papers to study from, while individual components could be found, the task of integrating them together was beyond complex.
- Maintaining the speed of conversion of drawings to images

11. SNAPSHOTS

```
Classification report for -
GridSearchCV(cv=None, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
                         {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001],
                          'kernel': ['rbf']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0):
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        22
           1       0.92      0.91      0.91       112
           2       0.93      0.93      0.93       123

    accuracy                           0.93       257
   macro avg       0.92      0.95      0.93       257
weighted avg       0.93      0.93      0.93       257
```
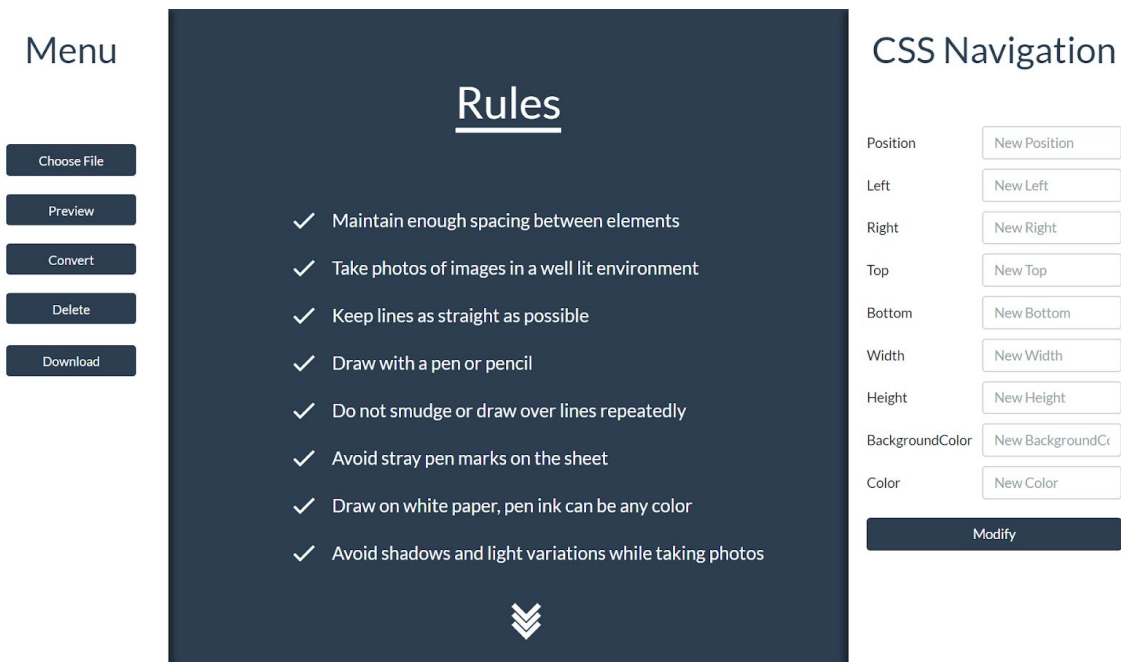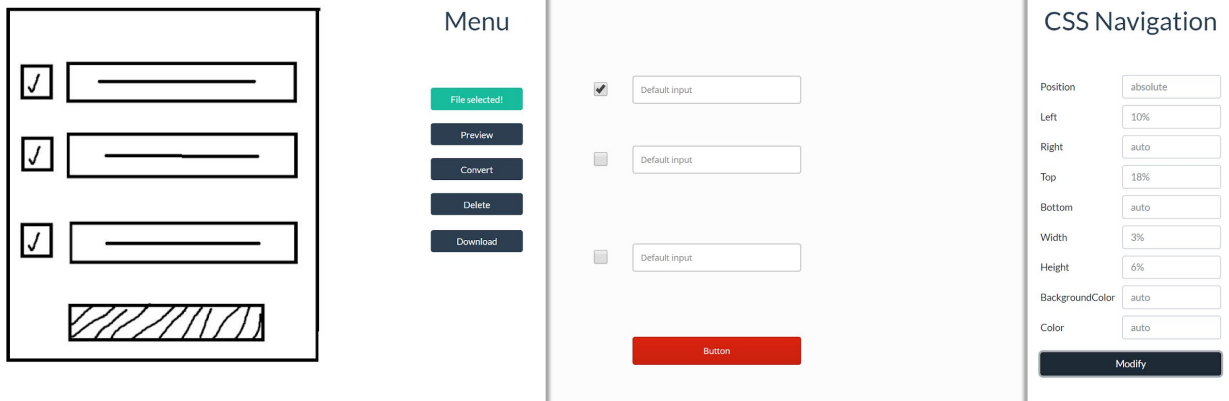
Image 11.1: SVM Training Accuracies

Image 11.2: Main page of the UI

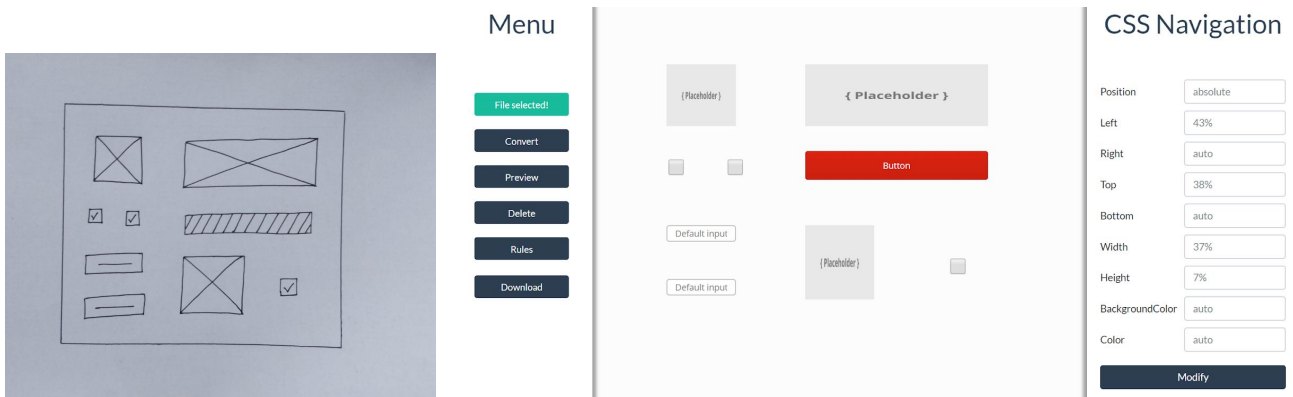**Image 11.3: Sample input and corresponding output (1)**
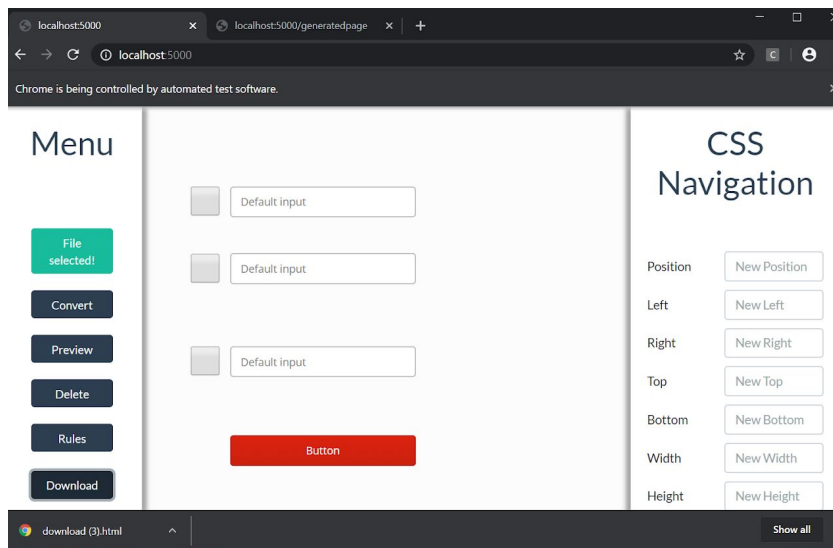

Image 11.4: Sample input and corresponding output (2)

Image 11.5: Automated Selenium running

_____

12. CONCLUSION

To conclude, this project consists of :

1. Getting a well-drawn or computer drawn image of a web page using the object structure provided.

2. Detecting all the objects from the sheet, applying OTSU filters and reducing the image to only binary colors. From here, all coordinates have to be obtained and cropped out for individual scanning.

3. The parts of the image are fed to the multiclass SVM which classifies the objects into relevant HTML elements.

4. The HTML elements are placed proportional to their location in the drawing and some basic CSS is added.

5. The HTML elements are further processed and have their alignment improved to have the elements line up cleanly. The page is also now fully responsive

6. The user's view is updated with the rendered page. From here the user can add final touches and clean up the page.

7. The user can then download a compiled HTML page with the desired CSS.

_____

13.   FURTHER ENHANCEMENT

Due to the vast scope of the project, the future improvements are nearly endless.

1. Optical character recognition (OCR) to allow users to enter text and have it show up in the HTML page.

2. More HTML elements can be added as required.

3. JavaScript code snippets can be integrated to allow basic functionality.

4. Optional stylesheets for users to pick the styles they want. (Fonts / Colors etc.)

5. Nested element detection can be added to that elements can be grouped into components.

6. Elements can be grouped into forms for submission.

7. Bigger datasets would allow more accurate detection.

8. More advanced ML models like Denoising Autoencoders would allow photos to be taken even in shabby lighting. Alternatively, GANs could entirely figure out how to clean all defects in an image and redraw the same image with sharper lines and improve detection. Better networks like YOLO CNNs could also give better detection results, however, this is contingent on the dataset.

9. Skeletonization of the images could make even extremely thick lines work normally and normalize the dataset.

—————————

BIBLIOGRAPHY/REFERENCES

[1] Adrian Rosebrock, "Computer Vision Blog"

[2] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You Only Look Once: Unified, Real-Time Object Detection"

[3] Zhong-Qiu Zhao (Member, IEEE), Peng Zheng, Shou-tao Xu, and Xindong Wu (Fellow, IEEEY). "Object Detection with Deep Learning: A Review". Ieee Transactions On Neural Networks And Learning Systems.

[4] Alexander Robinson, University of Bristol. "sketch2code: Generating a website from a paper mockup"

[5] Yichong Xu (Tsinghua University), Tianjun Xiao (Peking University), Jiaxing Zhang (Microsoft Research Asia), Kuiyuan Yang (Microsoft Research Asia), and Zheng Zhang (NYU Shanghai). "Scale-Invariant Convolutional Neural Network".

—————————

APPENDIX A DEFINITIONS, ACRONYMS AND ABBREVIATIONS

1. HTML = HyperText Markup Language
2. IDE = Integrated Development Interface
3. CSS = Cascading Style Sheets
4. GUI = Graphical User Interface
5. ML = Machine Learning
6. CV = Computer Vision
7. SVM = Support Vector Machine
8. SVC = Support Vector Classifier
9. RNN = Recurrent Neural Network
10. RGB = Red Green Blue
11. SWT = Stroke Width Transform
12. MLP = Multilayer Perceptron
13. ANN = Artificial Neural Network
14. ASPP = Atrous Spatial Pyramid Pooling