UNDERGRADUATE FINAL YEAR PROJECT REPORT

Department of Computer & Information Systems Engineering

NED University of Engineering and Technology

SKETCH2CODE

Group Number: 24 **Batch: 2018 – 2022**

Group Member Names:

MUHAMMAD HUZAIFA ABID	CS-18127
MUHAMMAD EBADULLAH KHAN	CS-18042
FARAH YAQOOB	CS-18115
MUHAMMAD ZAIN	CS-18125

Approved by

…………………………………………………………………………………………

Dr. MARIA

WAQAS

Assistant Professor

Project Advisor

Author's Declaration

We declare that we are the sole authors of this project. It is the actual copy of the project that was accepted by our advisor(s) including any necessary revisions. We also grant NED University of Engineering and Technology permission to reproduce and distribute electronic or paper copies of this project.

Signature and Date	Signature and Date	Signature and Date
…………………….	………………….....	…………………..
M. Ebad Ullah Khan	Muhammad Huzaifa	Muhammad Zain
CS-18042	CS-18127	CS-18125
ebadullah371@gmail.com	abidhuzaifa975@gmail.com	mzain9231@gmail.com

Signature and Date

…………………….

Farah Yaqoob

CS-18115

farahyaqoob02@gmail.com

Statement of Contributions

- Almost 1500 samples were collected in total (400 for each member) using labelimg software.
- Ebadullah and Zain contributed to Tensorflow object detection model training. Furthermore, integrated it in fast API and deployed it on AWS.
- Huzaifa and Farah contributed to Yolo model training and integrated it in flask API.
- Ebadullah and Zain contributed in designing Flutter web application
- Huzaifa and Farah contributed in integrating FAST api into Flutter web application
- All authors contributed equally to writing the final year project report.

Executive Summary

An early step in creating an application is to sketch a wireframe on paper explaining the structure of the interface. Designers face a challenge when converting their wireframes into code, this often involves passing the design to a developer and having the developer implement the boilerplate graphical user interface (GUI) code. This work is time-consuming for the developer and therefore costly. In this project, the authors have solved this problem. They build an application that translates wireframe sketches directly into code. This technique is called sketch2code. This project uses computer vision techniques to sort out this problem and train the model in such a way that it generates a code of the UI interface. For training, the dataset was collected manually, and then the authors used data augmentation techniques on the dataset. This project uses both the Tensorflow object detection algorithm and Yolo algorithm for the detection of objects in a sketch such as buttons, text, etc, and converts it into a programming language. This project also uses other computer vision techniques to recognize colors and handwritten text on the sketch of the user interface. The user will just have to take a picture of the sketch and upload it to the application. The algorithms will work on the backend to produce a code for that sketch which will be downloadable by the user. The goal of this project is to save the time of developers and make it user-friendly that everybody can sketch their applications and convert them into the code.

Acknowledgments

The successful completion of the project would only be possible with the constant inspiration and uplifting of the people who helped us achieve this milestone. First, we would thank Almighty Allah for guiding us on the right path and helping us wherever we were stuck. The role our parents was also very important and cheering for us throughout this journey. We would like to express our appreciation towards our internal advisor Dr. Maria Waqas, Assistant Professor, Department of Computer and Information System Engineering for giving us the opportunity to work on this idea and guiding us regarding the project, and taking out time from her busy schedule to assist us in this long journey.

Table of Contents

List of Figures

United Nations Sustainable Development Goals

The Sustainable development Goals (SDGs) are the blueprint to achieve a better and more sustainable future for all. They address the global challenges we face, including poverty, inequality, climate change, environmental degradation, peace, and justice. There is a total of 17 SDGs as mentioned below. Check the appropriate SDGs related to the project.

- ◻ No poverty
- ◻ Zero Hunger
- ◻ Good Health and Wellbeing
- ✓ Quality Education
- ◻ Gender Equality
- ◻ Clean water and Economic Growth
- ◻ Affordable and Clean Energy
- ◻ Decent Work and Economic Growth
- ✓ Industry, Innovation and Infrastructure
- ◻ Reduced Inequalities
- ◻ Sustainable cities and Communities
- ◻ Responsible consumption and Production
- ◻ Climate Action
- ◻ Life Below Water
- ◻ Life on Land
- ◻ Peace and Justice and Strong Institutions
- ◻ Partnerships to Achieve the Goals

Chapter 1
Introduction

1.1 Background Information

Whenever a project is started. It goes through several stages among which one is the planning stage. The client along with the developers plan the whole project, but the main aspect that they focus on at this stage is the user interface. They sketch the whole user interface or even use third-party software, just to have an idea of the whole UI. Then this plan goes to the developers and they code it out. Doing this is the conventional way of developing an application and it takes a lot of time and effort of a developer.

Developing a project requires several people who are experts in their domain. As the project planning stage is the most emphasized, hence it is the stage where the client andthe developers work most closely on. If this stage succeeds then the next stages will notcause problems because those stages are just connecting the backend with the frontendin which the developers are experts. Although mimicking what the client actually wantsis true art and what most people fail to do so.

Our project focuses on the very first stage of developing a project. This project has completely automated the UI coding phase. The client himself uses our app and outputsa UI exactly matching his needs. This project decreases the time required for this stageand most importantly decreases the client-developer coordination because this is where the most issues are created, as the developers are good in coding and making applications, but exactly coordinating with the client and mimicking what they need is totally another thing.

This project also aims to reduce manual efforts in developing a project. Moreover, this project completely removes the planning stage from the whole lifecycle of project making. The client can simply draw and use our app to generate its code.

1.2 Significance and Motivation

An early stage of developing user-facing applications is creating a wireframe to lay outthe interface. The design process begins with collaboration on a whiteboard where designers share ideas. Once a design is drawn, it is then manually translated into a working wireframe. After the wireframe's creation, it is given to a developer to implement in code. Developing boilerplate user interface code is time-consuming work but still requires an experienced developer and hence the overall cost and time increase. Sketch2code can give faster iteration, which means that a wireframe can move to a website prototype with only the designers' involvement. This can give accessibility to non-developers to create applications. Now anyone with little or no knowledge of coding can create great-looking wireframes. The project has also removed the requirement on a developer for initial prototypes, allowing developers to focus on the application logic rather than boilerplate GUI code. With advanced regularizationtechniques, we can get better results in terms of output GUI. In this way, the designerscan themselves make the GUI code and hand it over to the developers for coding the backend. This project reduces one whole stage of the pipeline and the corresponding costs.

1.3 Aims and Objectives

- To collect data of multiple sketches along with their respective codes
- To train an application on gathered data by selecting an appropriate model to achieve maximum accuracy.
- Create an application that translates a pencil sketch of a UI into its corresponding code.

1.4 Methodology

1.4.1 Data and Tools Exploration

A dataset has been created which contains the sketches and then we have created the corresponding codes for those sketches. Jupyter Notebook and VS Code have been usedfor this. The sketches were created using simple pencil and the names of those widgets were then used as an input in lableimg software. Now what is the whole process? The process starts with getting all empty plain sheets in hand then we start sketching the widgets on each plain sheet. The widgets can be in any position and in any size and shape but for the sake of this prototype, we just selected some widgets and their names so this means that the widget were taken of a particular shape and with a single name allotted to them. This means, after the sketching phase we have some papers with only the sketch on them. It is not labeled and hence no one can identify these sketches as one whole. The team then needed to name these individual images and hence we used a third party software to do that task. We used labelimg, downloaded it, installed it and used it to generate xml files of each corresponding image. The process flow is really simple but time consuming, first we have to navigate to a folder then select an image, after we do that then we get the sketched image on the screen. We then use the skill of this software to highlight the sketched portion of the paper and then save it as an xml file. The whole process is time consuming and hence most of the starting time was used in this phase. We repeat this step for all images and separate them into training and testing data folders. Now, at the end we get two folders with number of files multiplied by two which means that every file has a corresponding xml file which shows the positions of the outlined widget in the sketch which we manually outlined. It gives us x and y coordinates of the image. It includes the position where the widget has started and where the widget ended. So this file includes all the positions of the widget which is very important for the other tasks like model training and prediction. It is used in the model in such a way that the model uses the x and y coordinates to find the position in the image and get the model trained on that. So with this the data exploration comes to an end. What we discussed in this step is how to make the training and testing data set and label it using a third party software. When a person does all this then they are ready for the next phase. The issues that can arise in this step is the incomplete or incorrect labeling of the images. Like for example, the portioned you highlighted does not completely contains the image you drew so with his error the following steps will suffer because when you will give the similar looking widget for testing then the model will not be able to detect the complete bounding positions and hence will not be able to show all the bounding boxes. In the next page you can see the labelimg software where there is a very simple User interface. The user interface includes directory selection and editing the images. Then we can simply name the widgets that are available in that image.

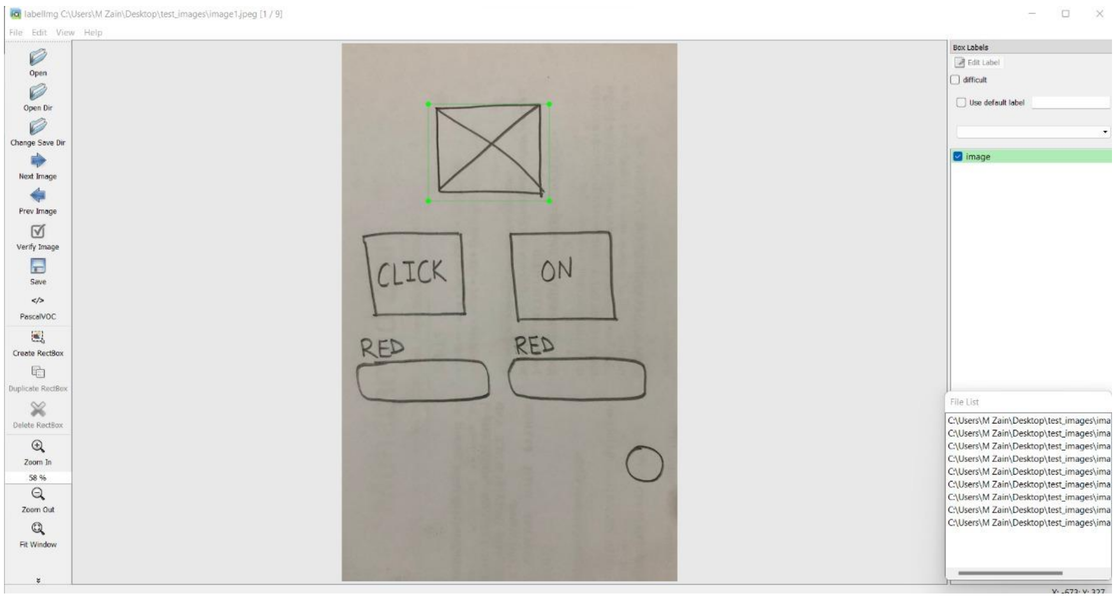
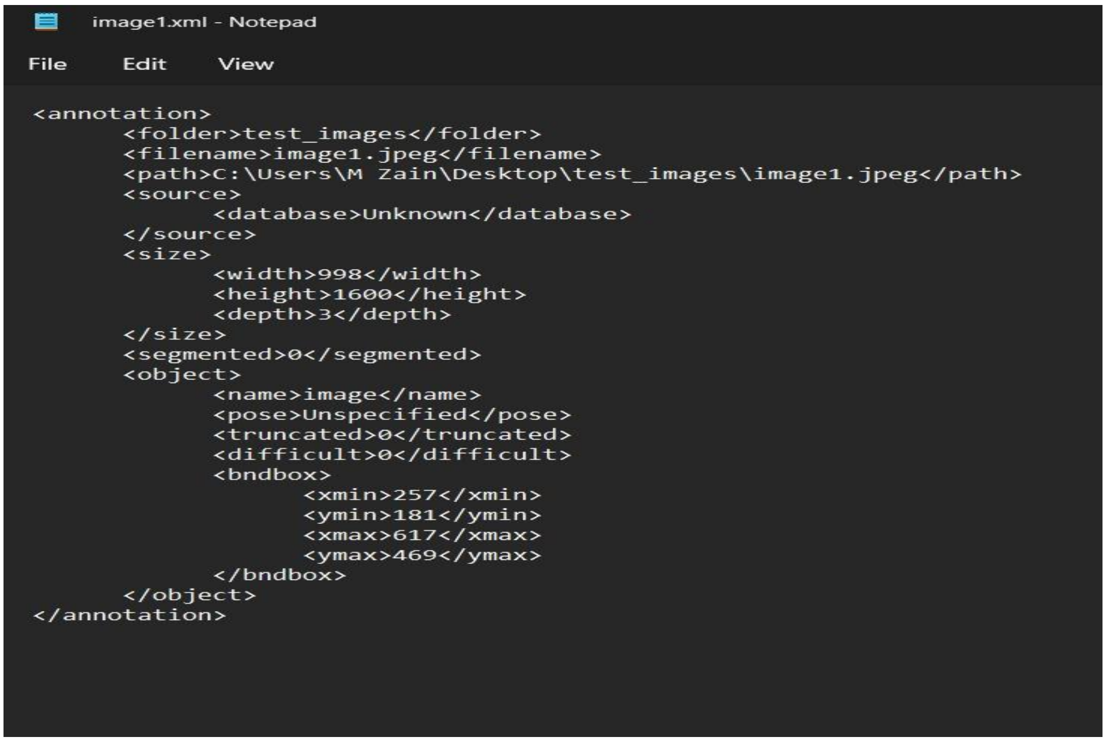


fig1. 1: labelimg

The next image is the xml file generated from the labelimg software. As we can see in this image that it includes the xml of the widgets. It includes the positions of the widgets that are present in that particular image. We can go through the files and see the x and y coordinates from the image that we selected. This file is very important and hence the naming conventions are also important. The name of the image fiile must be the same as the xml file or else the model will not detect this image with it’s corresponding xml file. If there is no xml file then there will be no image to be used for training



```
<annotation>
	<folder>test_images</folder>
	<filename>image1.jpeg</filename>
	<path>C:\Users\M Zain\Desktop\test_images\image1.jpeg</path>
	<source>
		<database>Unknown</database>
	</source>
	<size>
		<width>998</width>
		<height>1600</height>
		<depth>3</depth>
	</size>
	<segmented>0</segmented>
	<object>
		<name>image</name>
		<pose>Unspecified</pose>
		<truncated>0</truncated>
		<difficult>0</difficult>
		<bndbox>
			<xmin>257</xmin>
			<ymin>181</ymin>
			<xmax>617</xmax>
			<ymax>469</ymax>
		</bndbox>
	</object>
</annotation>
```

fig1. 2: xml file of image

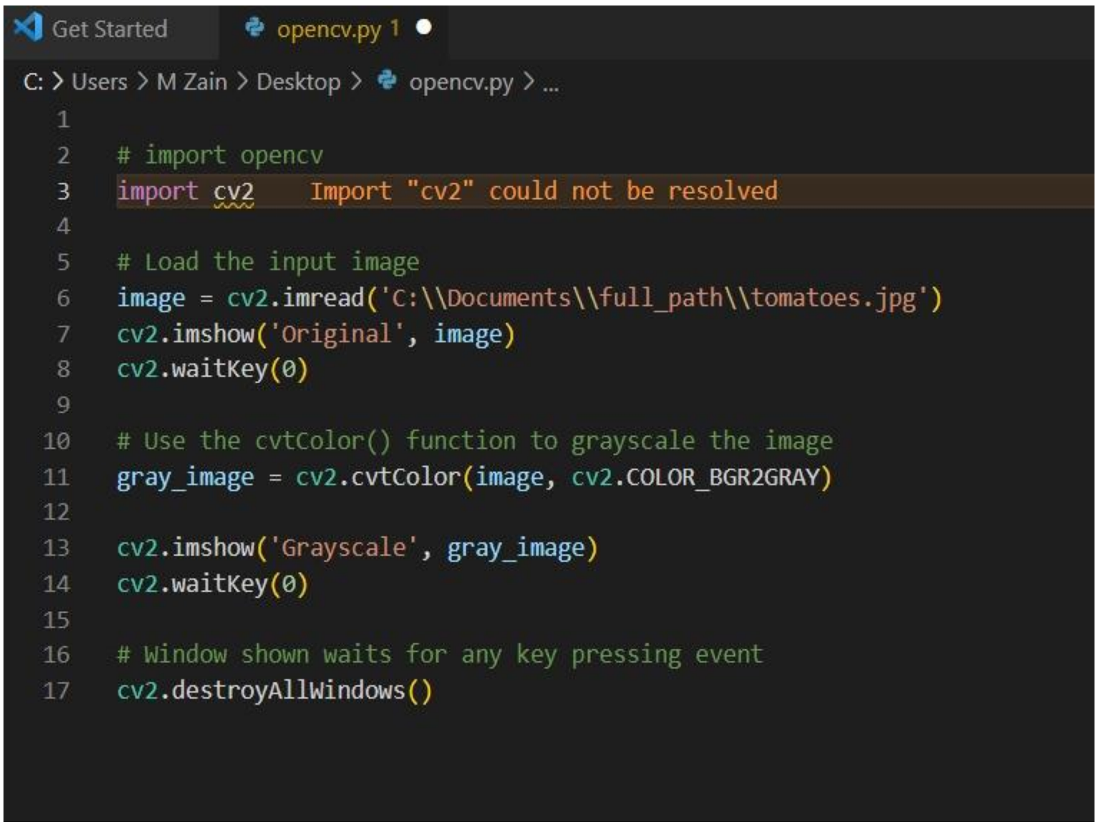
1.4.2 Data Cleaning and Feature Engineering

The image of the sketches has normalized and then we have augmented the images and optimized the code for the sketches so they can be referred to independently. The process starts with a very important phase of cleaning the data. So how do we start this phase? First let us see what we have till now, we have a training and testing folder with images and their corresponding positions. So, one thing to keep in mind is that the corresponding xml files is to not be edited and hence it will only be used by the model. We put the source of the training and testing folders and get the images to be printed in our code. When we get the images it is clear that the path given of the images is correct and we can start with some other steps. So the steps we have are as following:

- We need to clean the training and testing datasets
- We need to normalize the datasets
- We need to augment the dataset

For the cleaning of the training and testing datasets, we got the images in our code and hence we can start cleaning the images. We use open cv to read and manipulate the images as it includes several functions which easily can be used to clean and change the images. For example we need to read the image as black and white so, we can simple use open cv function to do that. Although in this project we do not read the images as black and white, we do a color read which is also very easily possible through the open cv library. When we read the image as required, then we can go on to the next step. The next step is to change the size of the image. The main reason we do this step is to cater all the client needs. As the clients can make the sketches on any kind of paper and hence the size of the papers can be different for every person using the app hence we have to find a solution to cater that. We do a simple thing of cropping the images into 320x320 size. We chose this size because the model we use was set to be trained on an image with size 320x320. When we do this step then we have an image which we can simple input into the model with its respective xml file. As creating the training and testing folders was a very tedious task so we just sketched very small amount of images. So we need to annotate those images that we sketched in order to give the model a significant amount of data. As we know that deep learning models need a vast amount of data to be trained on. When we have that then we can go onto the next step. Now how do we do this part? We use a sklearn library to do this step for us. Python has some machine learning libraries which we can use in order to use up our task. For augmenting the images we need to normalize them first again by using a machine learning python library. Wee do this to make every image look like one. As every person will take the image in a new environment with different shades on the paper, perhaps with different colored papers so we needed to think about that also. If we do not normalize the images then it will be an issue in the later stages. We first normalize the images so that any color in the background, any shade on the paper, any other object near the paper will be handled with this technique. At the end of the normalization step we get an image which is cropped and normalized to be given into the model. Although, one step is remaining of augmenting the dataset. When we have the normalized images we input them into a program which generates more images from one image. So it works in a way that it takes one image as input and generates five more similar looking images from it by cutting more edges, by reducing the height of the image, by changing the width of the image. We cannot do anything to the widgets as that will surely affect the training process. We defined some widgets and needed to be focused on those only. If the widgets change then their corresponding code will change and as this is only a prototype so were only focusing on five widgets. The widgets are a container, text box, an image, a button and a floating action button. When these are considered to be taken forward then we started the whole process from the very beginning. The widgets can be increased and changed, the only process which will be affected is the time required to do the labeling , the training and the detecting of those images as a whole.

The following screenshot is of an open cv code which includes some manipulation of the images. The code reads the images and show the image using opencv. When the image can be seen then we convert the image into grey scale. The function takes two things as input, one is the image path and the other is thee function to which we need to convert the image. Over here wee convert the image into grey scale as it is a requirement in the later steps. BGR2GRAY function converts the image into grey scale and hence it is done with the following code



```

# import opencv
import cv2    Import "cv2" could not be resolved

# Load the input image
image = cv2.imread('C:\\Documents\\full_path\\tomatoes.jpg')
cv2.imshow('Original', image)
cv2.waitKey(0)

# Use the cvtColor() function to grayscale the image
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

cv2.imshow('Grayscale', gray_image)
cv2.waitKey(0)

# Window shown waits for any key pressing event
cv2.destroyAllWindows()
```

fig1. 3: opencv code snippet

1.4.3 Model Training

An efficient model has been chosen after testing different models among Transfer learning. Here we are using two models. First we selected to work on tensorflow object detection and secondly we planned on working with YOLOv3. We took these two approaches to see what the results are. These two processes were very time consuming and each consisted of their own codes and model training. Let us start with tensorflow object detection; first we downloaded the object detection model from tensorflow hub and changed the parameters. Now this was a time-consuming task as we had to apply transfer learning and to do transfer learning many steps are required in order to attain accuracy. We just cannot just change the end layer and expect good results. We have to start the very beginning, we first have to create the images and label those images and create the corresponding xml files. Tensorflow takes xml files as input and inputs it into a model then all the magic happens. A deep learning model is kind of a black box. We input the image and get the output as detected bounding boxes. The following architecture shows the tensorflow object detection model.

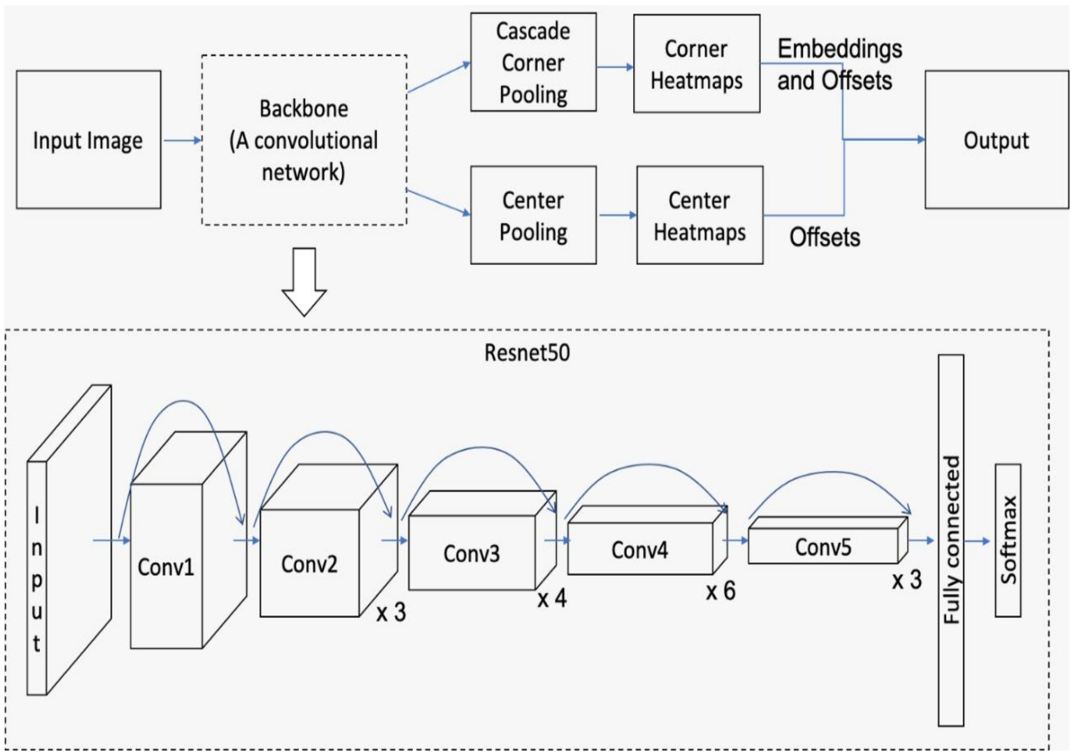


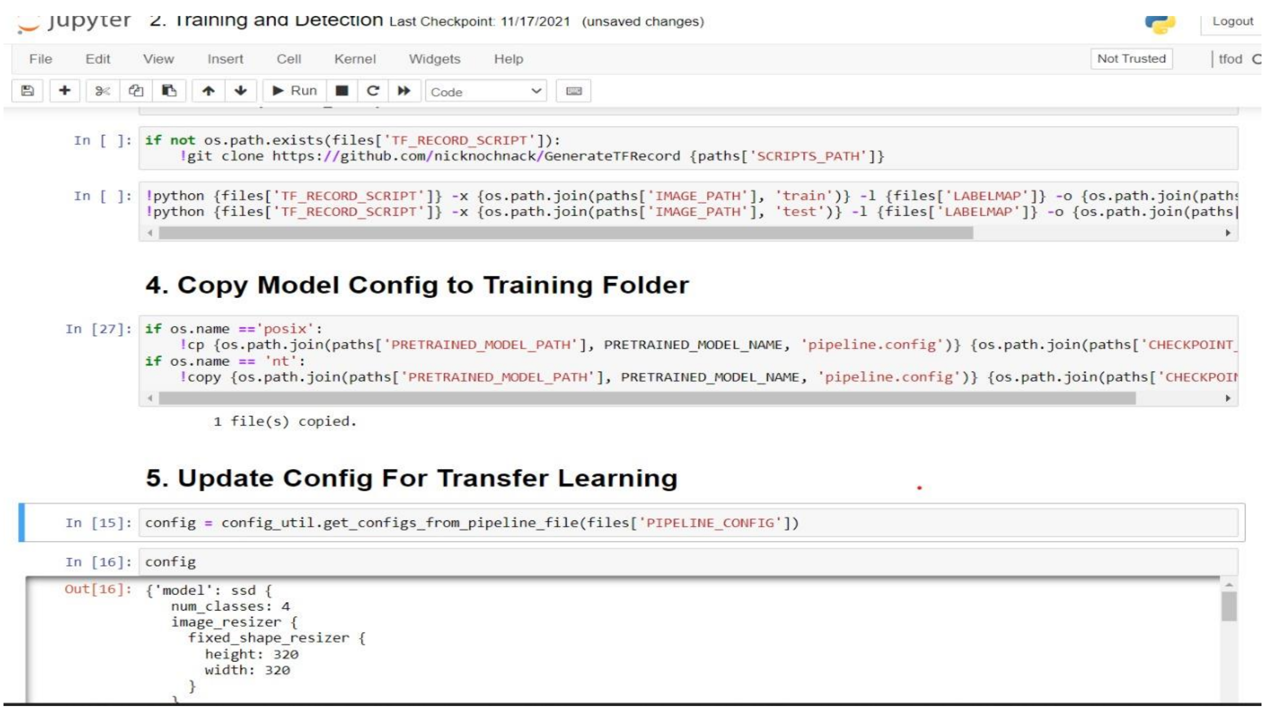
fig1. 4: Tfod architecture

The tensorflow model was first imported into the code and its hyperparameters were changed in order to cater the new images. The images are ready as we discussed in the previous steps. Now we just have to input those images into the model. We just have to change the outcomes. The model was trained on another dataset and hence the output was completely different. We first went into the configuration file of the model and edited some parameters, like the widget names and everything. We have to keep one thing in mind and that is to keep the same image name as we did in labelimg. Even if the capital letters are changed then it will consider it as a new widget and hence it will not work. So, this tells us that we need to keep the same spellings of the widgets and make no other changes in them. When we are sure that the names in labelimg are same and the y labels in the model are same then we can start the training phase of the model. The training requires the train images as input and the number of epochs. When we set these at the model then we can run the code and train it. After running the code, the training will start and it will take a lot of time. After the cell has run and no error comes then this means that the training is done. Hence, we can now start testing the images. The test image folder is separate from the train folder hence the model hasn't seen the test images till now so when we test the model on the test images then we can know the accurate results of the model in use. We can input the test image to the model and make it predict the bounding boxes. We can see the results in two ways. One way is for novices and the other way is for people who know how this all works. Let's start with the first approach for novices. The people can see the bounding boxes in the image. For this we will use open cv function for creating rectangles on the detected images. The images will produce some bounding boxes but the training model will not directly give those boxes so we have to do some pre computations in order to show those boxes. The model will give the x and y coordinates of the image detected and open cv will then we be used in order to show those boxes. The open cv function takes positions as input and returns a rectangle as an output over the image. The images on paper can be shown as a code output but with those rectangle boxes over the detected widgets. Now this technique is for the novices. Our project focuses on people who do not code or who do not have experience working in an environment like this. So for them we did this, they can see an image detected by giving an image as input. They can simply sketch the widgets on the paper with a pencil and upload it in our application and then they will see magic. With this they can see the images with bounding boxes drawn over them.

For summary we can see what we did here;

- Prepared model for training
- Adjusted the hyperparameters
- Edited the widget names
- Set the accurate epochs
- Train the model
- Test the model
- Calculate the accuracy

The above steps conclude the training off tensorflow model and hence if we do all these steps then we will be able to detect undetected images. The api is developed to ensure this process and hence if all this is done then we can deploy the project into amazon ec2 instance. The deployment will only work if all the preceding steps are working fine. So we will discuss on the deployment afterwards. First let us discuss on the models that we can work with and hence we can be able to train them with more and more epochs. A time will come that the models work fine and they predict good. The following image is the training phase of tfod



```
In [ ]: if not os.path.exists(files['TF_RECORD_SCRIPT']):
            !git clone https://github.com/nicknochnack/GenerateTFRecord {paths['SCRIPTS_PATH']}

In [ ]: !python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'train')} -l {files['LABELMAP']} -o {os.path.join(paths
        !python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'test')} -l {files['LABELMAP']} -o {os.path.join(paths
```

4. Copy Model Config to Training Folder

```
In [27]: if os.name =='posix':
             !cp {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'pipeline.config')} {os.path.join(paths['CHECKPOINT_
         if os.name == 'nt':
             !copy {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'pipeline.config')} {os.path.join(paths['CHECKPOI

             1 file(s) copied.
```

5. Update Config For Transfer Learning

```
In [15]: config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])

In [16]: config

Out[16]: {'model': ssd {
           num_classes: 4
           image_resizer {
             fixed_shape_resizer {
               height: 320
               width: 320
             }
```

fig1. 5: Tfod model training

The other model that we worked with was Yolov3. Yolo is quite different from tensorflow object detection and hence all the before steps were done before need to be repeated. Now, first we start with collecting the images. The sketches will remain same but the labelimg software producing the xml files will now change, instead of generating xml files the labelimg software will generate json files. The images will then have corresponding json files with each one. We can see the following architecture of YOLOv3 model. It takes different size images and the whole process is completely different than tensorflow object detection

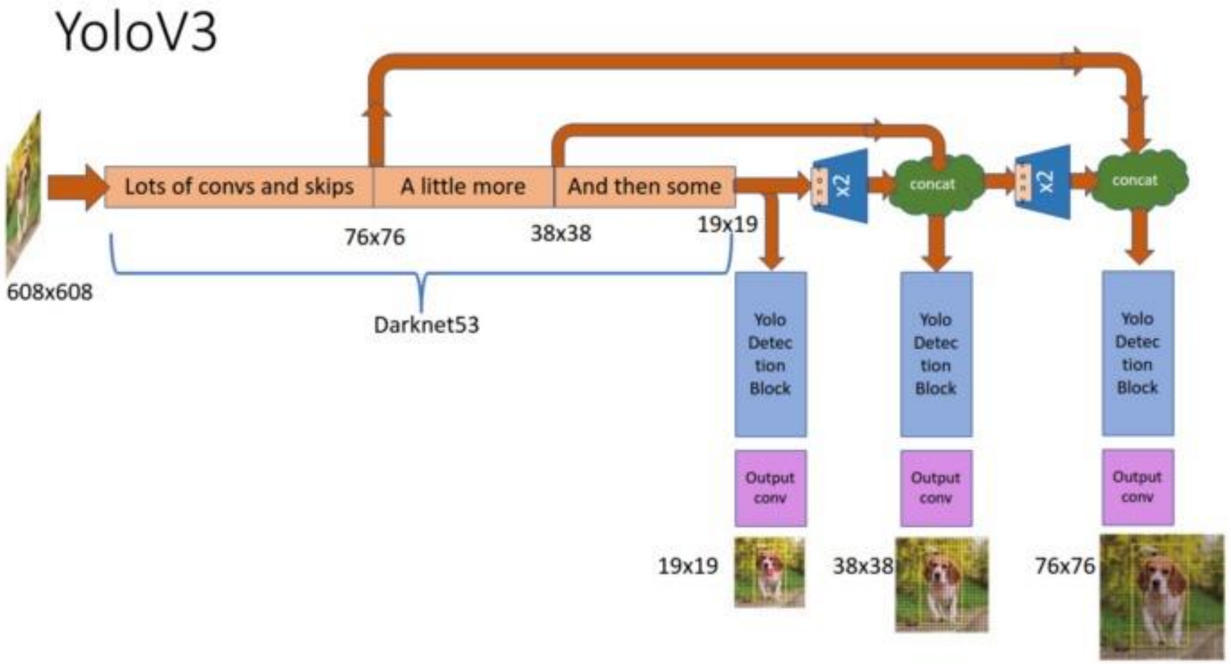


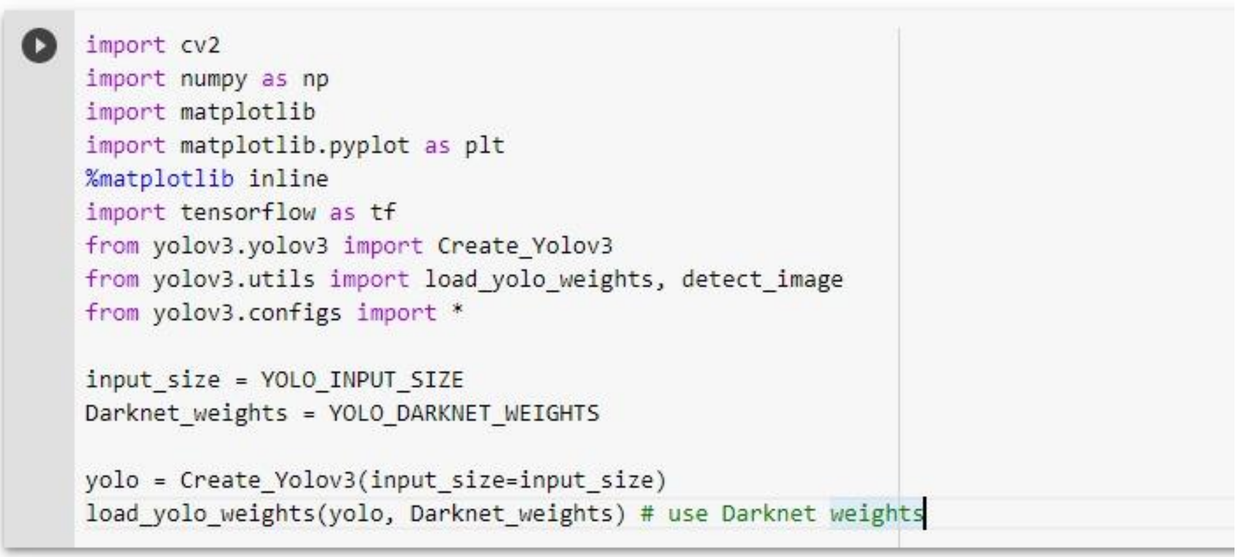
fig1. 6: Yolov3 architecture

The steps are ass follows:

- Prepare images
- Label and generate .json files
- Prepare the training model
- Change hyperparameters
- Train the model
- Test the model
- Note down the accuracy
- Compare with Tfod

We already have the sketches from before and now we just have to use the labelimg software to make the .json files. Before in the tfod model we generated xml files hence those files are not included and are not important over here so we cannot use the xml files over here. Yolov3 takes input as differently because the model architecture is completely different. It takes json file which has positions of the images in some other manner. So when we have the json files then we can prepare the model for training. The model requires images, their json files and the widget name correctly written. Before we discussed in tensorflow object detection that the widget names written in the labelimg software must be exactly the same over here too. So this means that the names must be same and if that is done from our end then we change the hyperparameters of the model. If this is done then we run the code which contains the training part hence we can run it and the model will be trained.

In the following code we are importing various python libraries and also yolov3. We will perform training on our dataset using yolov3 algorithm.



```python
import cv2
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
from yolov3.yolov3 import Create_Yolov3
from yolov3.utils import load_yolo_weights, detect_image
from yolov3.configs import *

input_size = YOLO_INPUT_SIZE
Darknet_weights = YOLO_DARKNET_WEIGHTS

yolo = Create_Yolov3(input_size=input_size)
load_yolo_weights(yolo, Darknet_weights) # use Darknet weights
```

fig1. 7: Yolov3 code snippet

1.4.4 Comparing Results and Performance Analysis

A comparison has been done on the basis of output code. The UI has been tested againstthe sketched UI manually. We hit the API on postman and get the results on the bounding boxes of the image so we compare the image at input with the image at output and see if the widgets are detected correctly and the bounding boxes are at proper locations and hence we also test the API which returns the JSON to us and see if the all values are correct including text, so it should detect everything. The following results can be seen that tfod detection is much more accurate and the bounding boxes are much closer to the image. The results really helped us in recognizing the models and seeing the results helped us to move on with tensorflow object detection. The container and buttons are accurately detected and yolo also detects properly but the bounding boxes are really far apart

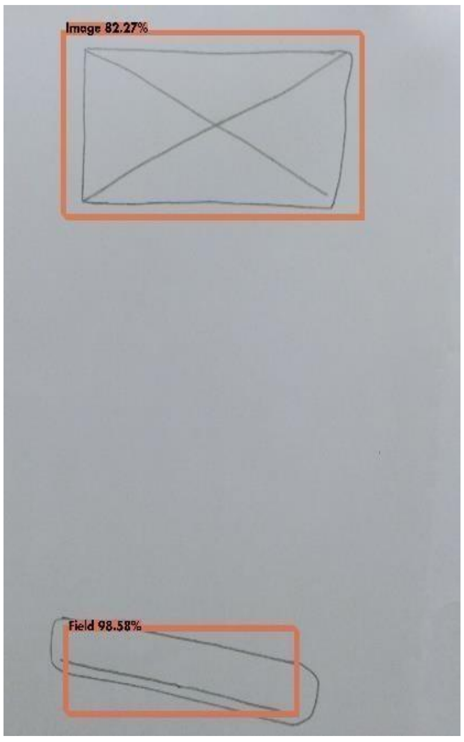


fig1. 9: image detection in sketch 1

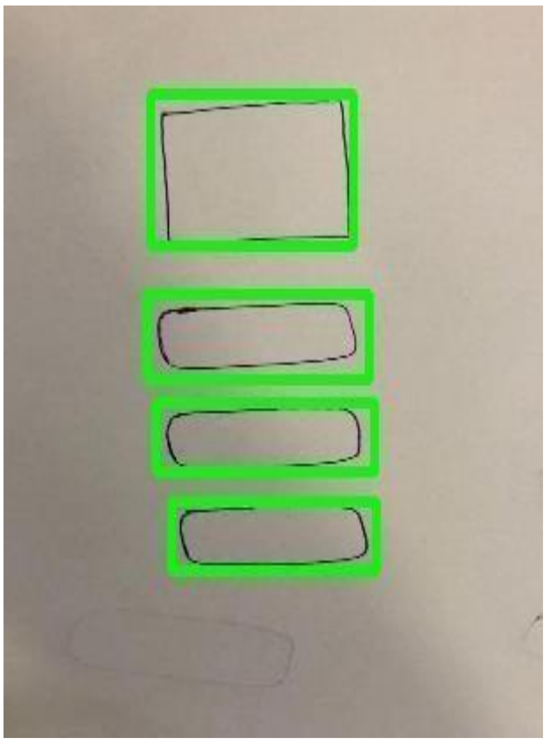
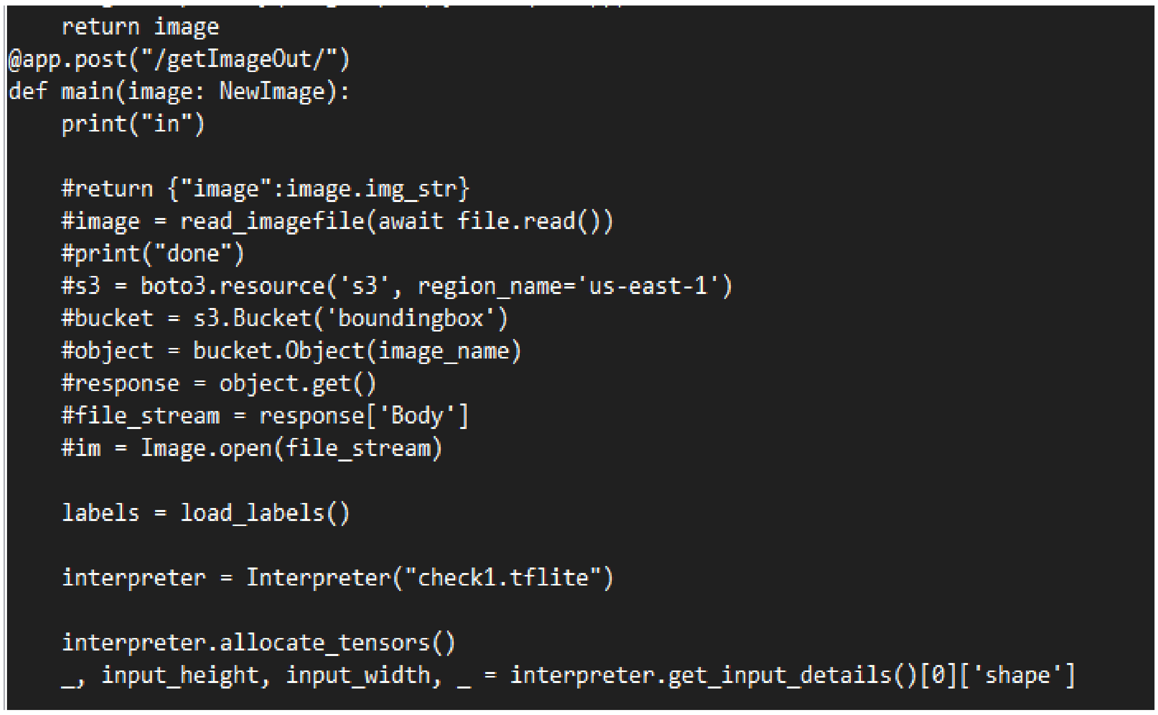


fig1. 8: image detection in sketch 2

1.4.5 Deployment

The proposed model has been deployed as an API in which the client can uploada sketch and the code will be generated with the UI. There are two models Tfod and text recognition model so we have deployed that on separate endpoints on AWS EC2 instance, we reserved that instance and got all the code on that instance. First we were working on windows but then we switched to Ubuntu as EC2 instance works better on it then we deployed it using fast API then we deployed nginx server through which we access endpoints of API.

The api is shown below. We used Fast API as a deployment type of our model, The api has several endpoints, some are for testing and some are integrated into the web app. The two testing endpoints are getPos/image1.png and getImage/image1.png. The other endpoints integrated are getPosOut/image1.png and getImageOut/image1.png.



```
    return image
@app.post("/getImageOut/")
def main(image: NewImage):
    print("in")

    #return {"image":image.img_str}
    #image = read_imagefile(await file.read())
    #print("done")
    #s3 = boto3.resource('s3', region_name='us-east-1')
    #bucket = s3.Bucket('boundingbox')
    #object = bucket.Object(image_name)
    #response = object.get()
    #file_stream = response['Body']
    #im = Image.open(file_stream)

    labels = load_labels()

    interpreter = Interpreter("check1.tflite")

    interpreter.allocate_tensors()
    _, input_height, input_width, _ = interpreter.get_input_details()[0]['shape']
```

fig1. 10: API

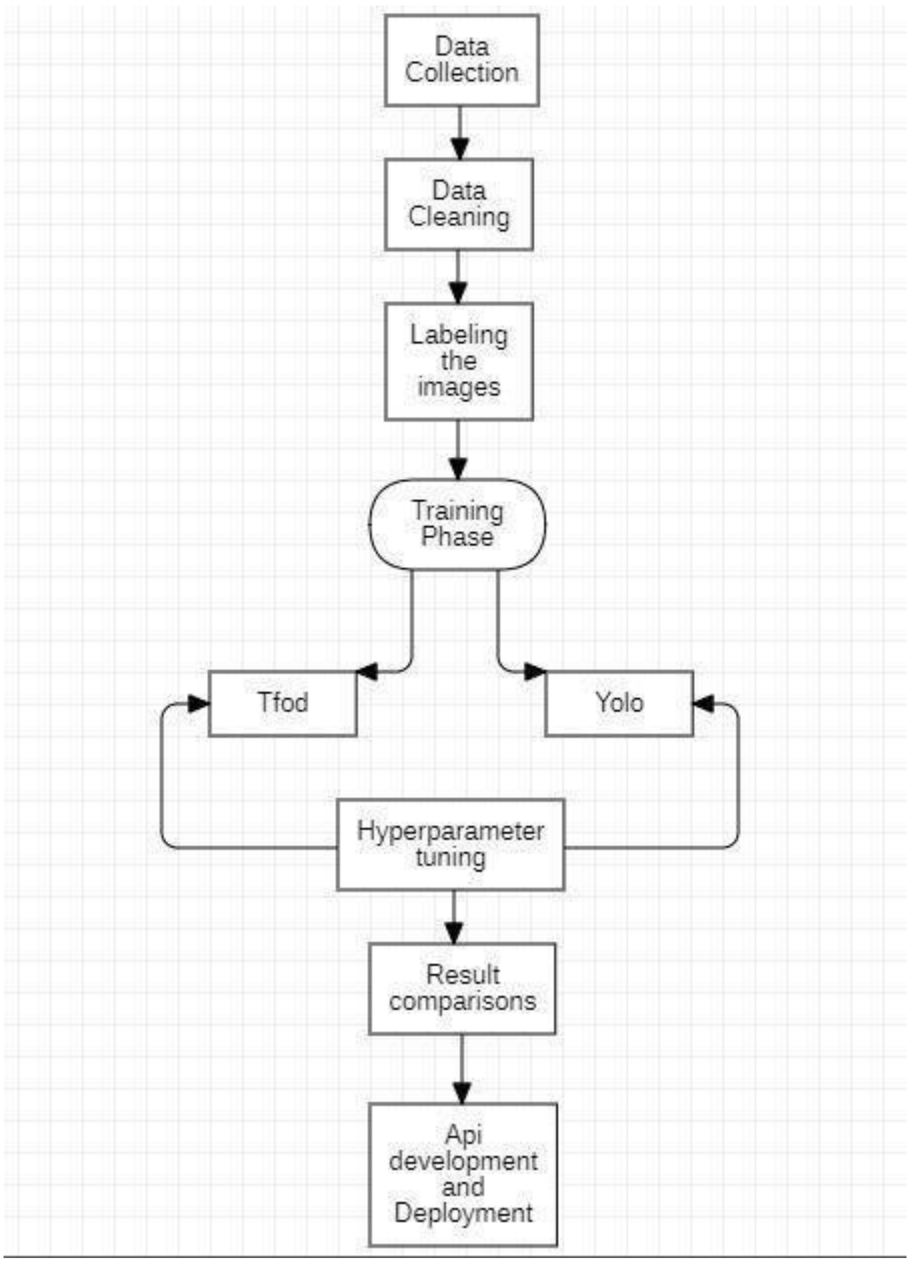


fig1. 11: block diagram of the process

1.5 Goals

- To help fast track the design phase of any development project
- Help non-coders to develop a fully coded user-interface
- Provide an easy-to-use interface with minimum manual work
- Fast and accurate
- Reduce time loss
- Easy to integrate into other applications

1.6 Applications

- Software houses can use this to reduce time in the development of UI
- Non-coders can use it to generate front end with its code
- Removes the planning and developing stage of SDLC

1.7 Report Outline

The report is structured in such a way that first the authors wrote the literature review, describing all the main projects which correlate with the project they built. The next chapter focuses on the data collection phase. As the data was not available online so the authors sketched all the sketches manually and labelled them. The targeted widgets arealso mentioned. In the 4th chapter the backend of our project is explained i.e. the model training. In the 5th chapter frontend of the web app is explained we the author mentioned the screens and also their screen shots. The 6th chapter focuses on API integration and deployment we have integrated two APIs in our project and deployed it on AWS. The 7th and last chapter is of conclusion where we have described the additional features i.e. text and color detection, for text recognition we have used OCR.

1.8 Summary

Providing a fast planning stage increases productivity in the later stages. The focus nowadays is to use Artificial Intelligence and reduce manual labor. This project will remove the whole user-interface designing and coding phase and hence the developerswill be focused on the main functionality of the project. Slight or harsh changes in the interface will be just one click away from our project.

Chapter 2
Literature Review

2.1 Introduction

An early step in creating an application is to sketch a wireframe on paper blocking out the structure of the interface. Designers face a challenge when converting their wireframe into code, this often involves passing the design to a developer and having the developer implement the boilerplate graphical user interface (GUI) code. This work is time-consuming for the developer and therefore costly. So, in our project, we have solved this problem. We have built an application that translates wireframe sketches directly into code. This technique is called sketch2code. We are using computer vision techniques to sort out this problem and train our model in such a way that it generates a code of the UI interface. For training we have collected our datasets manually after that we will use the data augmentation technique on our dataset. We have used the Yolo algorithm for the detection of objects in sketches such as buttons, text, etc, and converted it into a programming language. So basically, the goal of our project is to save the time of developers and make it user-friendly that everybody can sketch their applications and convert them into code.

2.2 Review of Literature Surveys

Researchers at University of California, Berkeley and Carnegie Mellon University (CMU) have implemented SILK (Sketching Interfaces Like Krazy), a sketching tool that combines many of the benefits of paper-based sketching with the advantages of current electronic tools. With SILK, designers can quickly sketch an interface using an electronic pad and pointer, and SILK recognizes widgets and other interface elements as the person draws them. Unlike paper-based sketching, however, designers can use these elements in their sketchy state. For example, a sketched scoller is likely to contain an elevator, the small container the user drags with a mouse. In a paper sketch, the elevator would just sit there, but in a SILK sketch, users can drag it up and down, which lets them test the components or widgets' behavior. [J.A. Landay and B. A. Mayers]

Through a study of web design practices, we observed that web site makers design sites at different levels of refinement; storyboard, and individual page— and those designers sketch at all levels during the early stages of the design. However, current web design tools do not support these tasks very well. Informed by these previous observations, DENIM was created, a system which helps web site designers in the early stages of a design. DENIM supports sketching input, allows design at different levels, and unifies the levels through increasing the size. Informed by the study, we designed and implemented a prototyping tool to assist the designers in the early stages of web site design. DENIM is more informal than SILK, the authors named the system DENIM, which also conveniently stands for Design Environment for Navigation and Information Models. DENIM is used for prototyping in the starting stages of design, but not for the creation of complete web sites. For example, it does not output finished HTML and CSS pages. The authors built DENIM in Java 2, on top of SATIN, a toolkit which supports informal pen-based interaction. In DENIM, web pages are helped by a label that represents the name or description of the page. The labels remain in a consistent size throughout all the zoom levels, so that they can always be read. There are two ways to create a page in DENIM. One way is to simply write some words on the canvas while being in the site map view. A blank page is created with the words as its label. The other way is to draw a rectangle, which is converted to a page without a label.[James Lin]

Single image super-resolution (SR), which recovers a high-resolution image from a low-resolution image, is a classical problem in computer vision. The sparse-coding-based method is one of the representative SR methods. This method involves several steps in its pipeline. First, overlapping patches are densely cropped from the input image (e.g., subtracting mean and normalization). These patches are then encoded by a resolution dictionary. The sparse coefficients are passed into another resolution dictionary for reconstructing high-resolution patches. The overlapping reconstructed patches are combined (e.g., by weighted averaging) to produce the final output. This pipeline is shared by most external methods, which pay attention to learning and optimizing the dictionaries or building efficient functions. However, the rest of the steps in the pipeline have not been optimized or considered in a unified optimization framework.[Chao Dong]

Video classification is the task of producing a label that is a lot relevant to the video given with its frames. A good video classifier is a one that not only provides accurate frame labels, but also best describes the video given the features of the various frames in the video. For example, a video might contain a tree, but the label that is central to that particular video might be something else (e.g., “hiking”). The accuracy of the labels that are needed to describe the video depends on the task. Typical tasks include assigning one or more labels to the video, and assigning one or more labels for each frame of the video.[Balakrishnan Varadarajan]

Being able to describe the content of an image using properly formed English sentences is a challenging task, but it could have great impact, for instance by helping visually impaired people so can better understand the content of images on the web or mobile app. This task is significantly harder, for example, than the accurate image classification or object recognition tasks, which have been and still are a main focus in the computer vision community. Indeed, a description must capture not only the objects in an image, but it also must express how the objects are related to each other as well as their attributes they are involved in. Moreover, the above semantic knowledge graph has to be expressed in a language like English, which means that a language converting model is needed in addition to a visual understanding model. The main inspiration of this work comes from recent advances in machine and single barrio translation, where the task is to transform a sentence written in a language, into its translation in the other target language, by maximizing probability target given source. For many years, machine translation was achieved by a series of separate tasks like translating words individually etc, but recent work has shown that language translation can be done in a much simpler way using RNNs and still reach state-of-the-art performance. An RNN reads the source sentence and transforms it into a rich length vector representation, which in turn in used as the initial state of a RNN that generates the target sentence.[Oriol Vinyals]

The user inputs a GUI image of an application to our program. Looking at this image, the model must detect all the components present in the image. The model is trained on synthetically generated data. This dataset consists of several images of the components and their corresponding labels i.e., text-input, text, image, or button, which are required for supervised learning. In order to generate this dataset, we have automated the process of writing code to generate individual components such as text inputs, buttons, images, and text with varying attributes such as dimensions and color. The process of taking screenshots of the generated components is also automated and it puts cropped images of all these components in the training dataset. The images are labeled according to the name given while sketching the component. The dataset generated by this method was used for the training of our model. To simplify the process of converting the GUI design to the respective React Native code, we built a model using computer vision and machine learning. The GUI structure consists of basic components such as buttons, textbox, etc. After compiling the generated React Native code, we get a prototype app screen layout for the given GUI design.[Alex Robinson]

Chapter 3
Data Collection

3.1 Introduction

The data group is the process of assemblage and weighing facts from innumerable various sources. We are utilizing the dossier we accumulate to expand useful machine intelligence answers. A data group is a research component that fully studies fields, containing tangible, social sciences, trade, and arts. Data must be calm and stocked to comprehend misrepresentation questions within reach.

In our project, we secondhand the plan to draw the sketches of the UI in theory and we take the pictures of those sketches and secondhand two approaches for the detection of the UI that will be considered further. In our first point, we point in a direction of five gadgets for UI discovery, we draw each gadget in theory accompanying various blends, then accepted an exact likeness of each page and secondhand algorithms for the discovery of each figure.

Our treasure gives the forecast and produces a rule of the matching UI in the scoot expression. We have collected countenances of help-illustration gadgets in theory and used the dossier improving to the ruling class. The purpose of this project search to draw the images of the UI in theory and take an exact likeness it, that is the recommendation of our model, and search out create the rule of the UI in a scamper vocabulary that is secondhand for mobile, netting, and personal computer requests, the law is the product of our model.

The amount of our model holds 2 concepts, the first representation holds the law of the UI and the second output is the representation holding the restricting box about each gadget that gives an allotment of the incident of each gadget. The purpose of our project searches out to resolve the question of hiring additional UI creators. Our model is bestowing as correct a result as likely.

3.2 Targeted Widgets

1. Button
2. Container
3. Floating action button
4. Image
5. Text field

In this phase, we target 5 basic widgets for UI but for later on, we have target text detection using OCR and also include the color feature in our project which is the advanced feature of our model.

3.3 Dataset

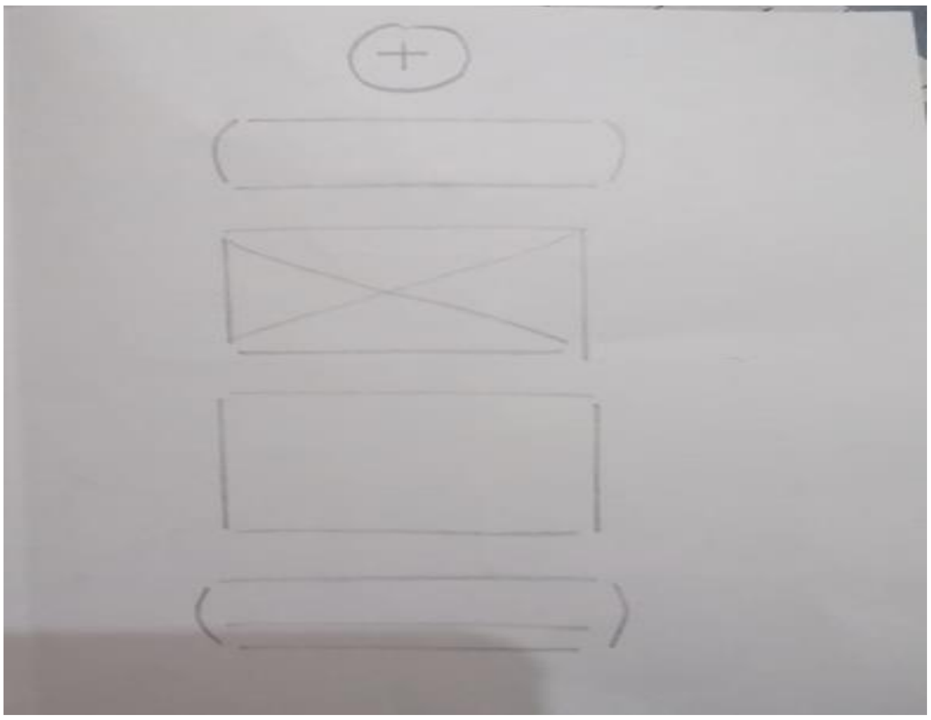


fig 3. 1: dataset sample 1

The following images are drawn by hand and they consist of all the five individual widgets. The widgets are a container, buttons and an image. There are more widgets in this training dataset but for now only these are shown in order to just see the predictions

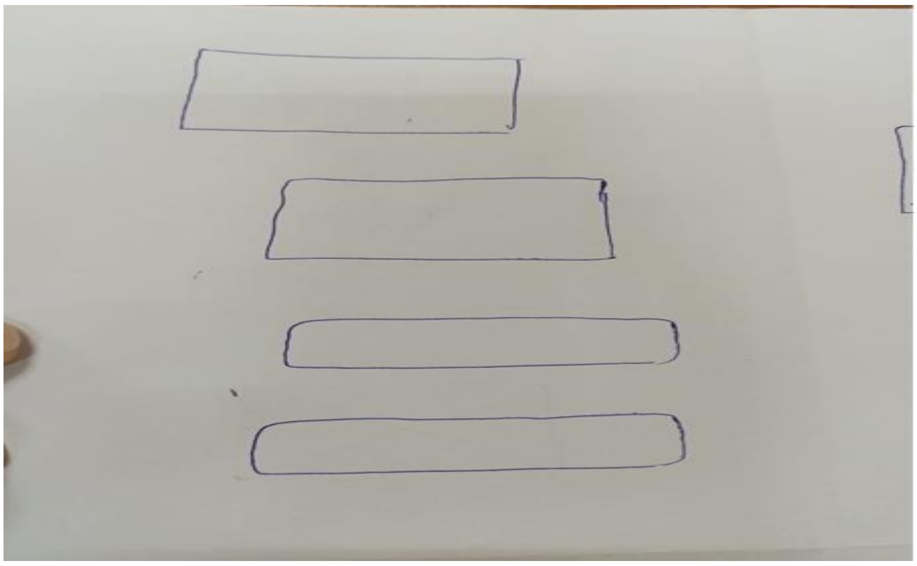


fig 3. 2: dataset sample 2

The following image consists of an image, a container and a button. They are aligned in a single column and hence the output will also be aligned in a single column. Slight changes in the UI can also be detected by our model and hence shown in code

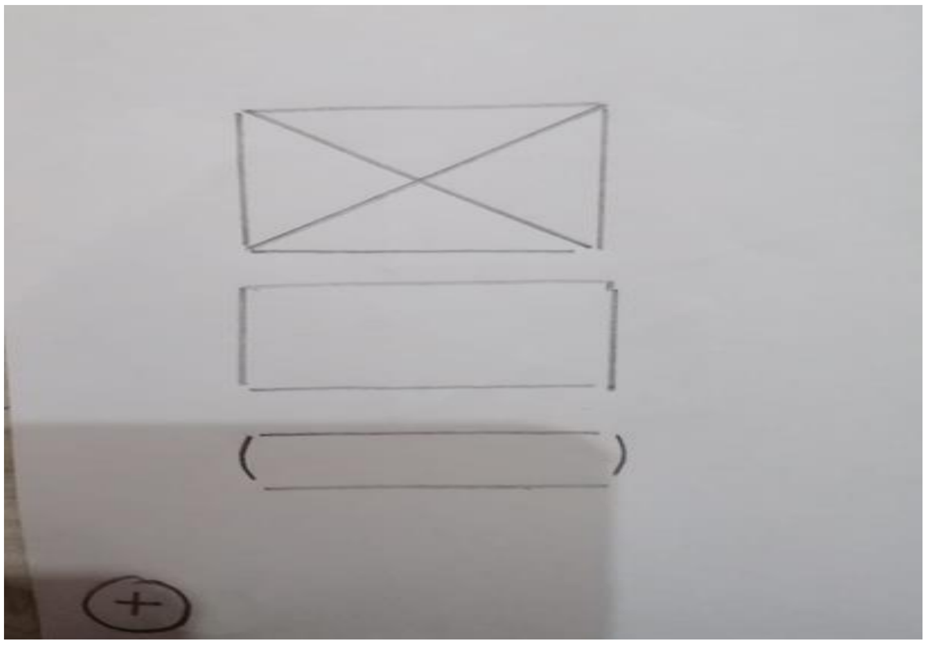


fig 3. 3:dataset sample 3

3.4 Summary

Now we have collected 1500 images of different widgets with different combinations which give accurate results but later on, we will collect more data to get better results as possible. We will also collect the data for text and color detection which is the advanced feature we will add to our model.

Chapter 4

Backend Development

4.1 Data Collection

We have collected data manually by drawing sketches on paper. We have used labelimg to label the data. The widgets can be in any position and in any size and shape but for the sake of this prototype, we just selected some widgets and their names so this means that the widget were taken of a particular shape and with a single name allotted to them. This means, after the sketching phase we have some papers with only the sketch on them. It is not labeled and hence no one can identify these sketches as one whole. The team then needed to name these individual images and hence we used a third party software to do that task. We used labelimg, downloaded it, installed it and used it to generate xml files of each corresponding image. The process flow is really simple but time consuming, first we have to navigate to a folder then select an image, after we do that then we get the sketched image on the screen. We then use the skill of this software to highlight the sketched portion of the paper and then save it as an xml file. The whole process is time consuming and hence most of the starting time was used in this phase. We repeat this step for all images and separate them into training and testing data folders. Now, at the end we get two folders with number of files multiplied by two which means that every file has a corresponding xml file which shows the positions of the outlined widget in the sketch which we manually outlined. It gives us x and y coordinates of the image. It includes the position where the widget has started and where the widget ended. So this file includes all the positions of the widget which is very important for the other tasks like model training and prediction. It is used in the model in such a way that the model uses the x and y coordinates to find the position in the image and get the model trained on that. So with this the data exploration comes to an end. What we discussed in this step is how to make the training and testing data set and label it using a third party software. When a person does all this then they are ready for the next phase.

4.2 Model Training

For the preparation of the dataset, we have secondhand two approaches YOLO and Tensorflow object discovery. YOLO invention engages convolutional affecting animate nerve organs networks (CNN) to discover objects in palpable opportunity and take recommendation labels. Object Detection utilizing Tensorflow is a calculating view method. As the name implies, it helps us in detecting, establishing, and the path an object from a countenance or a broadcast.

The below image shows the YOLOv3 output. As we can see the model correctly predicts the container with a probability of 86 percent and also correctly predicts the text field with an accuracy of 92 percent. The model works great but as the number of images increase the accuracy tends to decrease

Also, the json files made from labelimg software were predicting good but still the model epochs or other hyperparameter needed to be changed in order to predict much more accurately

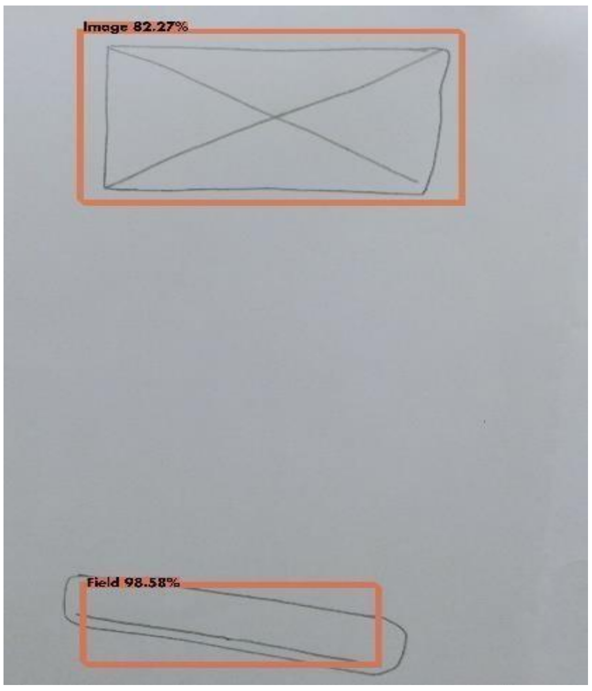


fig 4. 1: Yolo prediction 1

The following image is a YOLO output and it predicts the floating action button with a really good accuracy and also predicts the container with a good accuracy. The container as you can see is predicted but the rectangular bounding boxes are very far apart, some people would consider this as normal but this can cause a lot of issue afterwards

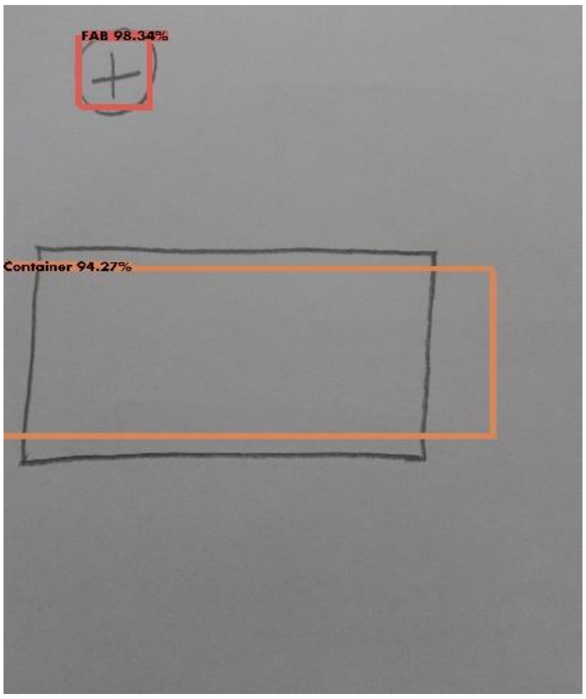


fig 4. 2:Yolo prediction 2

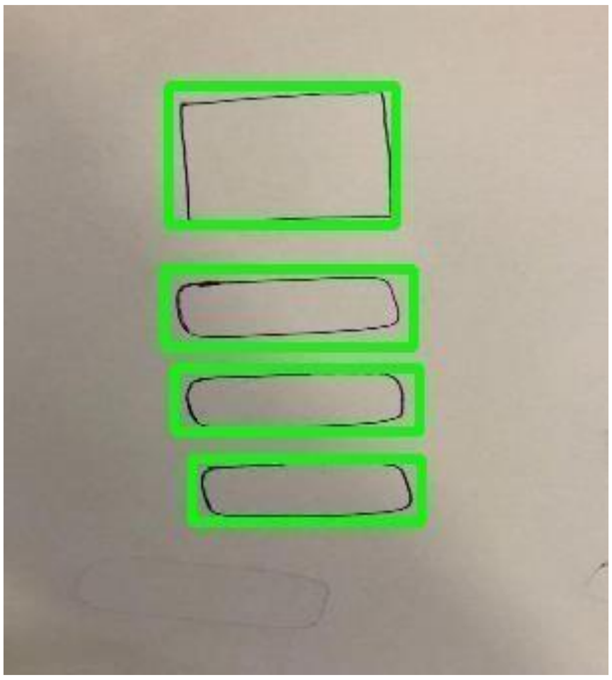


fig 4. 3:Tfod prediction 1

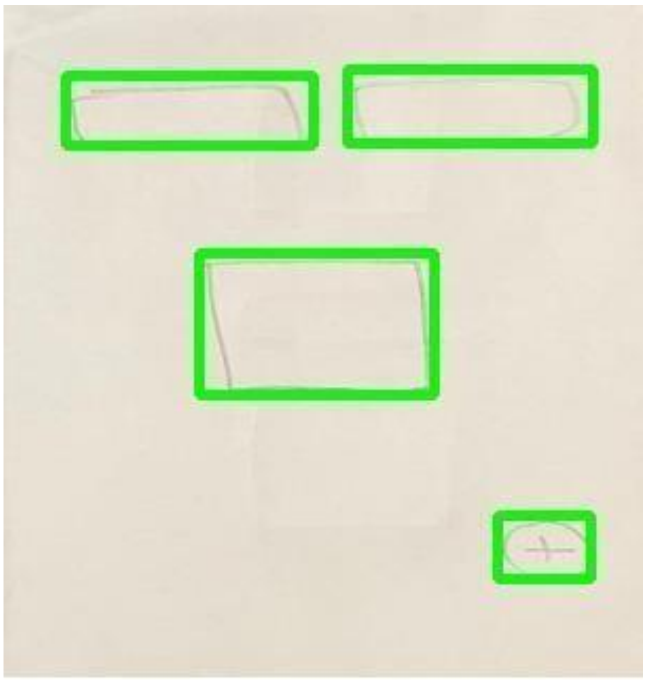


fig 4. 4:Tfod prediction 2

Chapter 5

Feature Addition

Previously we targeted five features which are container, raised button, floating action button, image, and a text field. Our model detects these features and places them in the right position with their respective height and width. Our targeted widgets do not contain any text in them and also there is no color in them, it’s a simple UI having no colors in it. Our widgets seem like an HTML in which there is no color. To make these widgets’ code more automated we also add colors in it so that users will be able to automate the code of color along with the code of each widget. Not only did we automate the code of color but also added the text detection feature in it. Now, are targeted widgets also contain some text in it which our model is going to detect and generate the corresponding code of it. In short, our widgets contain text in it and also colors. So we are adding two more features to our project i.e text detection and color detection. For text detection, we used easy OCR and for color detection, we also used the same technique which we used in easy OCR.

5.1 Additional Features

For the advancement of our project, we added some new features in our project to make it a complete prototype. We added features which are given below

1. Colors of widgets
2. Text in widgets

5.1.1 Colors

We have also brought an extra characteristic to our mission which is color detection. To observe the color of each widget like image, container, floating action button, raised button, and text field. We skilled our model with all the five widgets that we used in our project and labeled every widget with a coloration name. We train our model which contains text with every widget to increase the accuracy of our model. For now, in an early step of our project, as a prototype, we used solely three hues which are red, blue, and green. Initially, we train, our model with three colors as a prototype later we will add more colors to it for further advancement. In the backend, we used two APIS both the APIS takes an image as an input but the output of them are different from one another. One API that takes an image as input also returns an image as an output but at that time the image contains a bounded box around it which detects the position of each widget and the other API contains the JSON response as an output. That JSON contains an independent list of each widget. In each widget list, the list contains the prediction of the widget, height, and width of the widget, and also contains an encoded value of the color of the widget containing the color in it otherwise it will be zero. That encoded value shows that a widget contains the color in it. In the front end which is the Flutter web app the encoded value of color is converted into flutter code which at last produces the UI as output and that output contains a widget along with its color. For the above whole implementation, we draw handwritten sketches of these widgets and also write the name of the color with the widget. We used easy OCR for text detection of that color and converted it into an encoded value.

This image shows the widgets drawn and on top their respective colors are written. When we take the image and upload it into the web app then the code will be returned

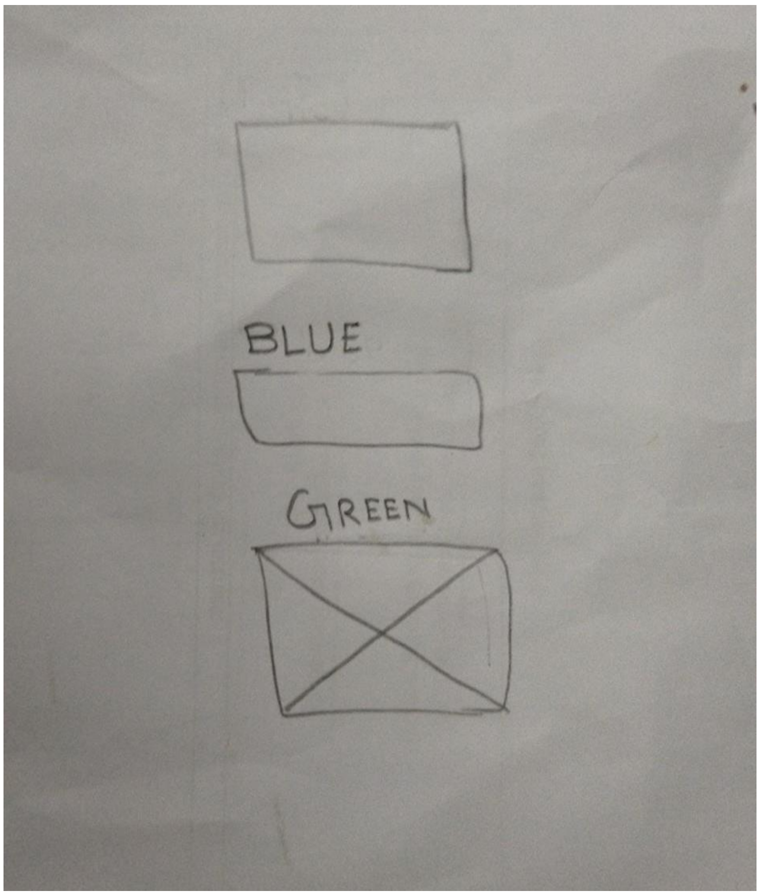


fig 5. 1: input image

The following image Shows the code of dart language. The container had no color hence a default gray color is shown but the button had a blue color and image has a green color

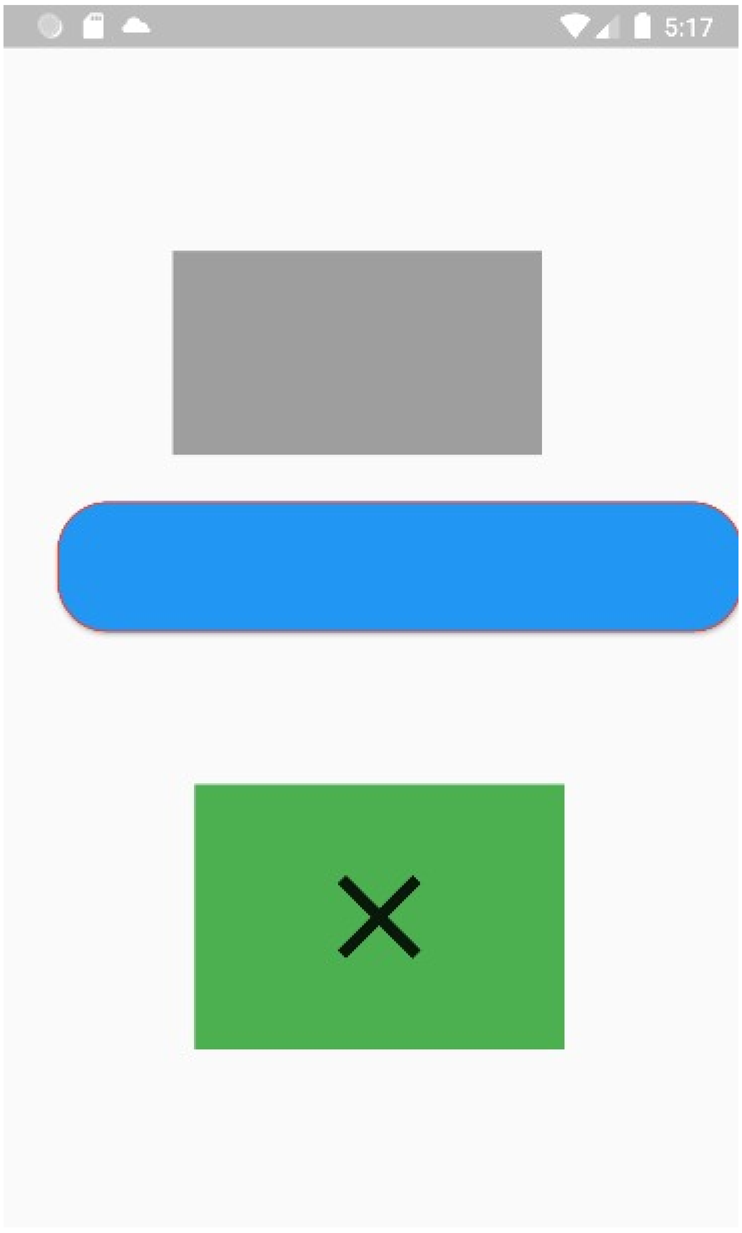


fig 5. 2:Output image

5.1.2 Text

We used the same strategy for text detection which we used in color detection. All 26 alphabets are encoded into numbers in the backend. That JSON contains an independent list of each widget. In each widget list, the list contains the prediction of the widget, height, and width of the widget, and also contains an encoded value of the text, if a widget is a text then that list contains an encoded value of that text otherwise it will contain zero in it. Now this encoded text would be converted into Text at the front end. At the front end, we build a web app that is Flutter based so that encoded value is converted into text in the front end. But here comes the question that how we convert our text into an encoded value. For this, we used easy OCR. Easy OCR is used to detect the text.

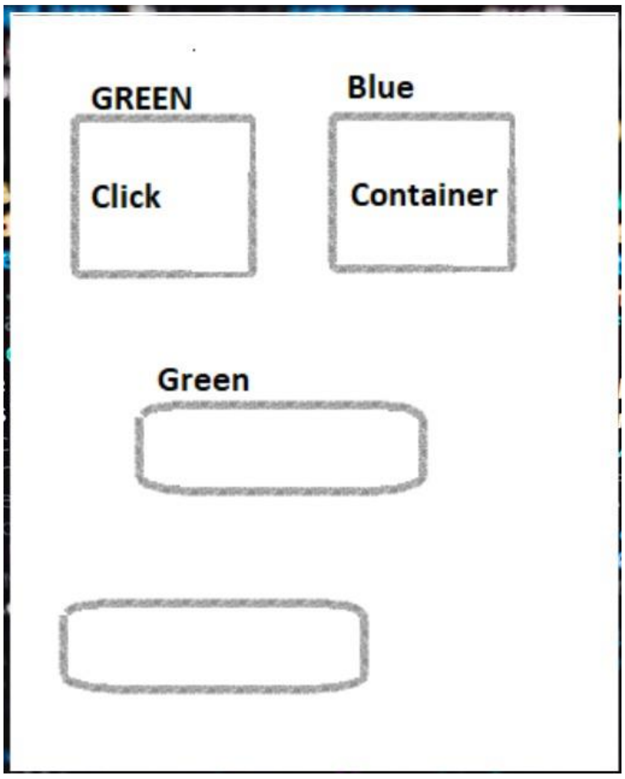


fig 5. 3:Input image

Now as you can see the sketch we gave at input contained text in a widget as well as colors above the widget as labels and as a result we got accurate output of all widgets the colors of the first two container were green and blue which was produces as output also and the color of the button was labeled as green and actually got green. Those widgets whose colors are not defined are given grey color by default

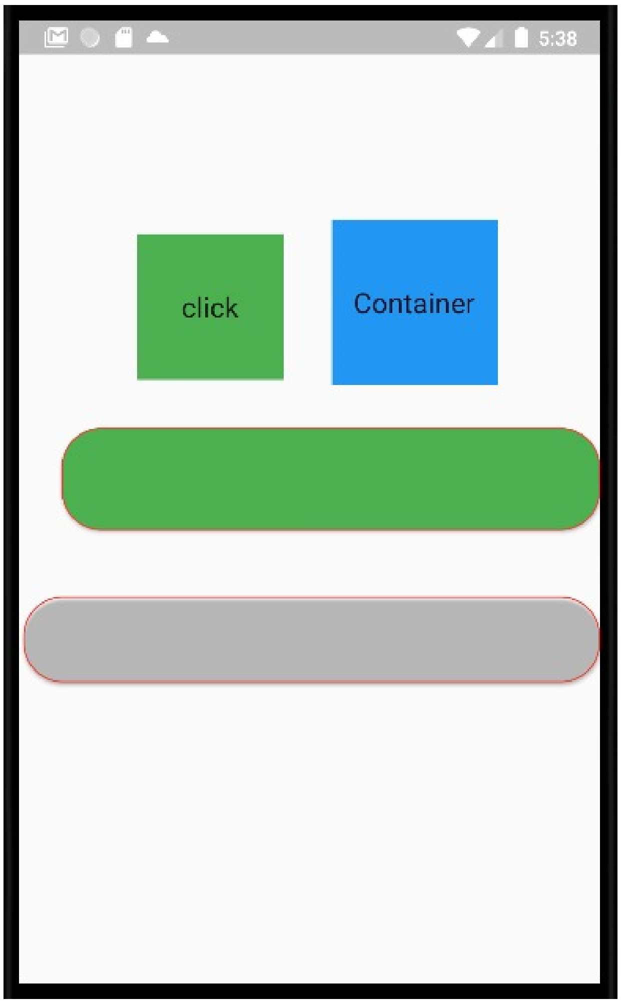


fig 5. 4:Output image

5.1.2.1 OCR

OCR (scanning of documents) is the use of technology to identify printed or manuscript text figures inside digital images of tangible documents, such as a leaf through the paper document. The basic process of OCR includes examining the manual of a document and translating the characters into rules that can be secondhand for data processing. OCR is consistently also referred to as text recognition. OCR orders are made up of an alliance of hardware and program that is used to convert tangible documents into machine-legible text. Hardware, to a degree an optical scanner or specific circuit board is used to copy or read text while a spreadsheet typically handles the leading processing. The software can again take advantage of machine intelligence (AI) to implement more advanced means of intelligent figure recognition (ICR), like labeling languages or styles of calligraphy. The process of OCR is most commonly used to turn hard-copy legal or historic documents into PDFs. Once established in this soft copy, users can rewrite, format, and search the document as if it was generated with a discussion processor. The beginning of OCR is using scanning of documents to process the physical form of a document. Once all pages are reproduced, OCR software converts the document into a two-color, or black and white, rendition. The scanned-in countenance or bitmap is analyzed for light and dark extents, where the dark extents are identified as figures that need to be acknowledged, and light areas are recognized as background. The dark extents are then processed further to find alphabetic memorandums or numeric digits. OCR programs can change in their techniques, but usually involve targeting one personality, word, or block of the theme at a time.

The below flow diagram shows the whole architecture of optical character recognition. The user uploads the picture which includes the text in it. The picture is preprocessed and then to enhance it we have to input the same image to the feature extraction model of optical character recognition. After that phase the features go to the feature store where they can be used afterwards. So that model works fast and accurately. After that we make use of the classifier and hence we need it. The classifier detects the text in the image and then it outputs the text it detected. The text is returned with the positions of each text. So with the positions in our hand we can show the bounding boxes over the text images as shown above

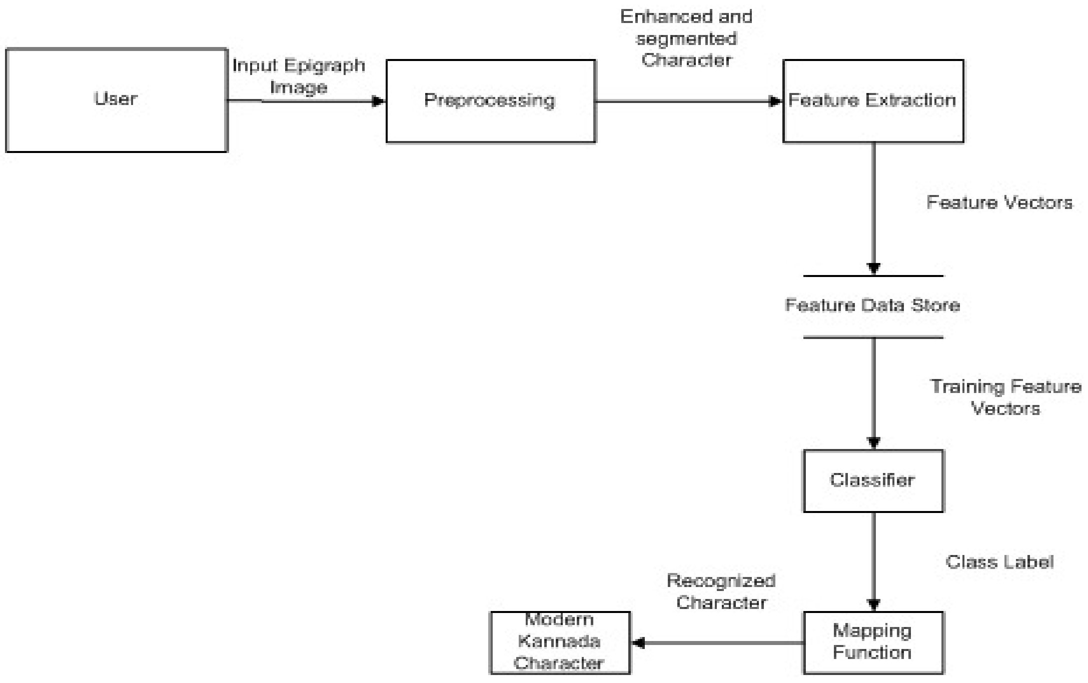


fig 5. 5:OCR architecture

Chapter 6

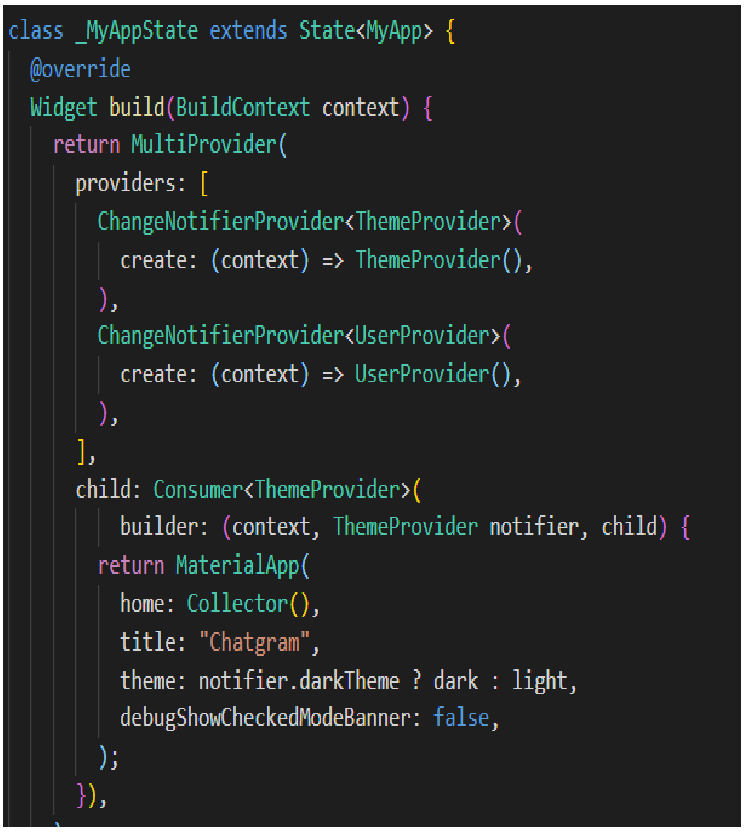
Frontend Development

6.1 Tool

We used Flutter in our project to serve Frontend. Flutter is a Cross-platform hybrid framework that is used to develop mobile applications, web applications, and desktop applications. It is launched by Google. Flutter uses Dart programming language. In our project, the Web interface is developed using Flutter.

6.2 Flutter

Below mention is the screenshot of the code that is written in dart language in Flutter. Dart is a programming language that is used in flutter for coding purposes. It's a single language that is used to write the entire code of the mobile applications built for both platforms that are Android and Ios. In our project, we used Flutter to build the web-based Front end. The Screenshot contains the code written in the main class of Flutter.



```
class _MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ChangeNotifierProvider<ThemeProvider>(
          create: (context) => ThemeProvider(),
        ),
        ChangeNotifierProvider<UserProvider>(
          create: (context) => UserProvider(),
        ),
      ],
      child: Consumer<ThemeProvider>(
          builder: (context, ThemeProvider notifier, child) {
        return MaterialApp(
          home: Collector(),
          title: "Chatgram",
          theme: notifier.darkTheme ? dark : light,
          debugShowCheckedModeBanner: false,
        );
      }),
```

fig 6. 1:flutter code snippet

6.3 Android Emulator

The image below looks like a mobile device. This image is an Android emulator used internally to run and debug apps. An Android emulator is a software application that allows your mobile phone to mimic the functionality of the Android operating system on your PC. This allows you to install Android apps on your computer or laptop and use them natively. Used primarily for debugging purposes. We used the android emulator by downloading it from the AVD manager. In AVD manager different versions of mobiles are present for all sizes like iPhones, iPad, and other android devices.

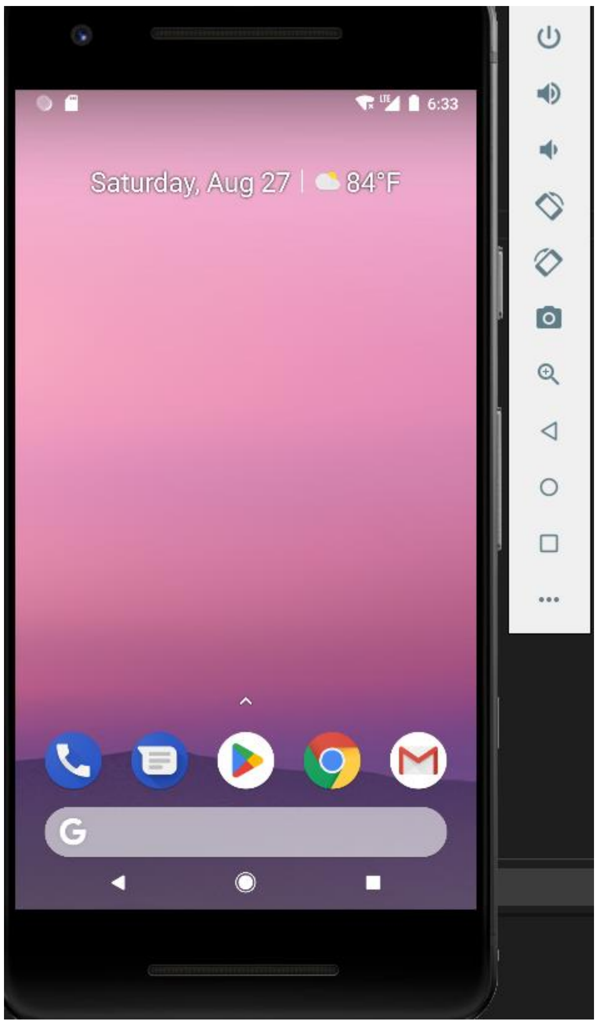


fig 6. 2: android emulator

6.4 First Screen

In Figure 5.1, you can see that this is the first screen of our web application which is built in Flutter using dart language. This screen contains a background image and two buttons that are Upload image and Get prediction. The upload image button will load the image from the computer which we are selecting from our personal computer and local storage and then display it in the window while the getting prediction button will call 2 APIS. One API will take an image and also return an image as an output with a bounded box around each widget that we used in our project for detection while the other API will return JSON with three properties (class, x, y) and this will be shown on the second screen.

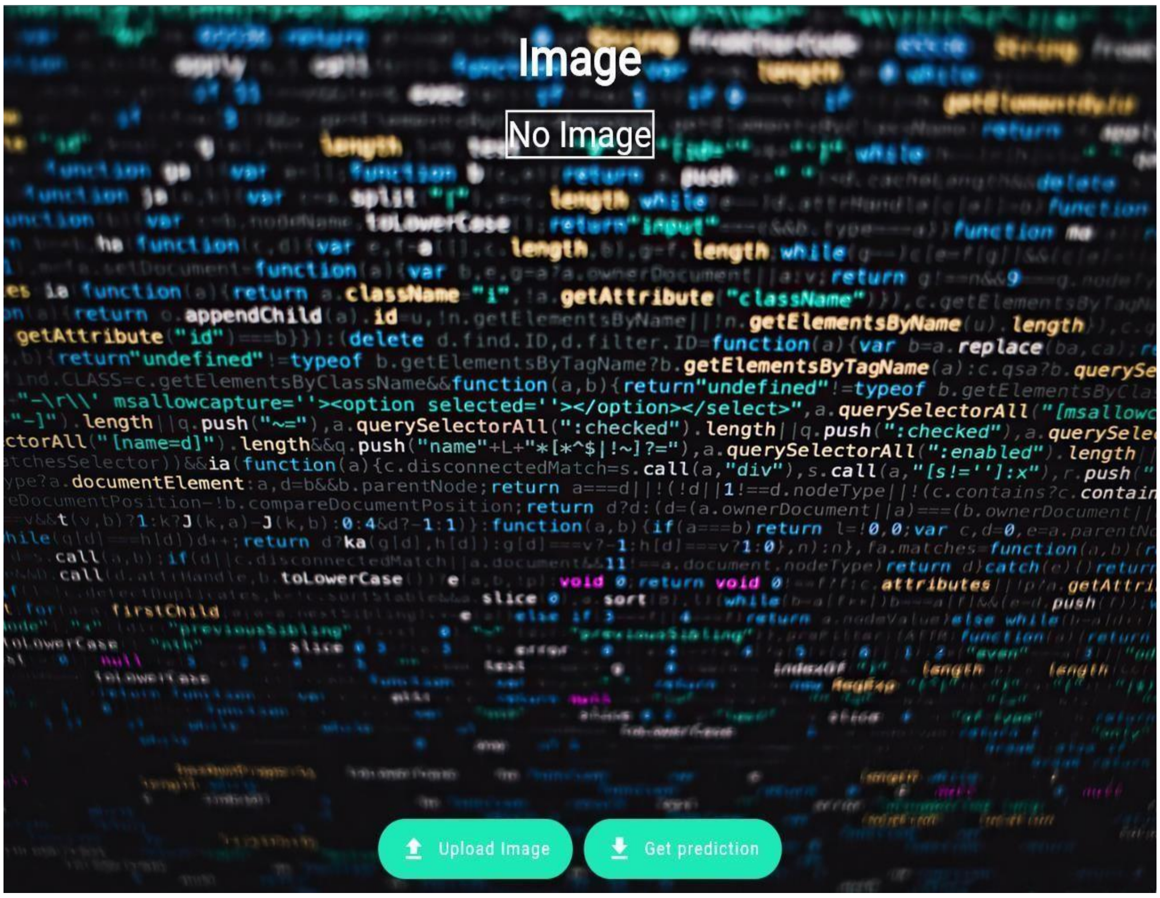


fig 6. 3: UI of webapp

6.5 Slider

In this screen, you can see that we are using the slider feature of flutter. A slider in Flutter is a material design widget used for selecting a range of values. It is an input widget where we can set a range of values by dragging or pressing on the desired position. We used a slider in our web app to provide our users samples that they can use for testing purposes. We added more than one testing image on this screen so that the users select the image of interest. Below mentioned, the image shows that it contains 4 widgets which are container, an image, another container, and a floating action button which can be used to test our web app for demo purposes.

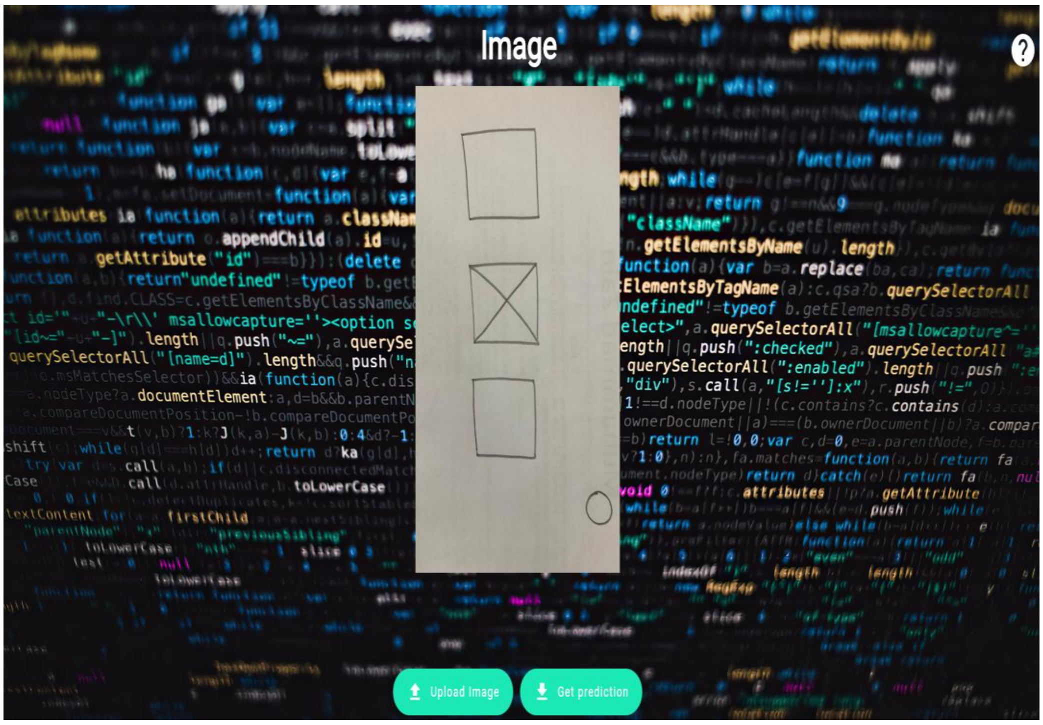


fig 6. 4:slider 1

On this screen, you can see that we are showing the second demo of our dataset which users can use and try our web app first as a demo, and then they can use our app for their own choice of handwriting sketches. This sample dataset represents a total of 5 widgets you can see that it contains 3 text fields, 1 floating action button, and 1 container with a color feature in it. This sample dataset is different from the first dataset because it also contains the color detection feature in it for a demo purpose. If users want to see a demo of color detection they can use this sample data as a demo.

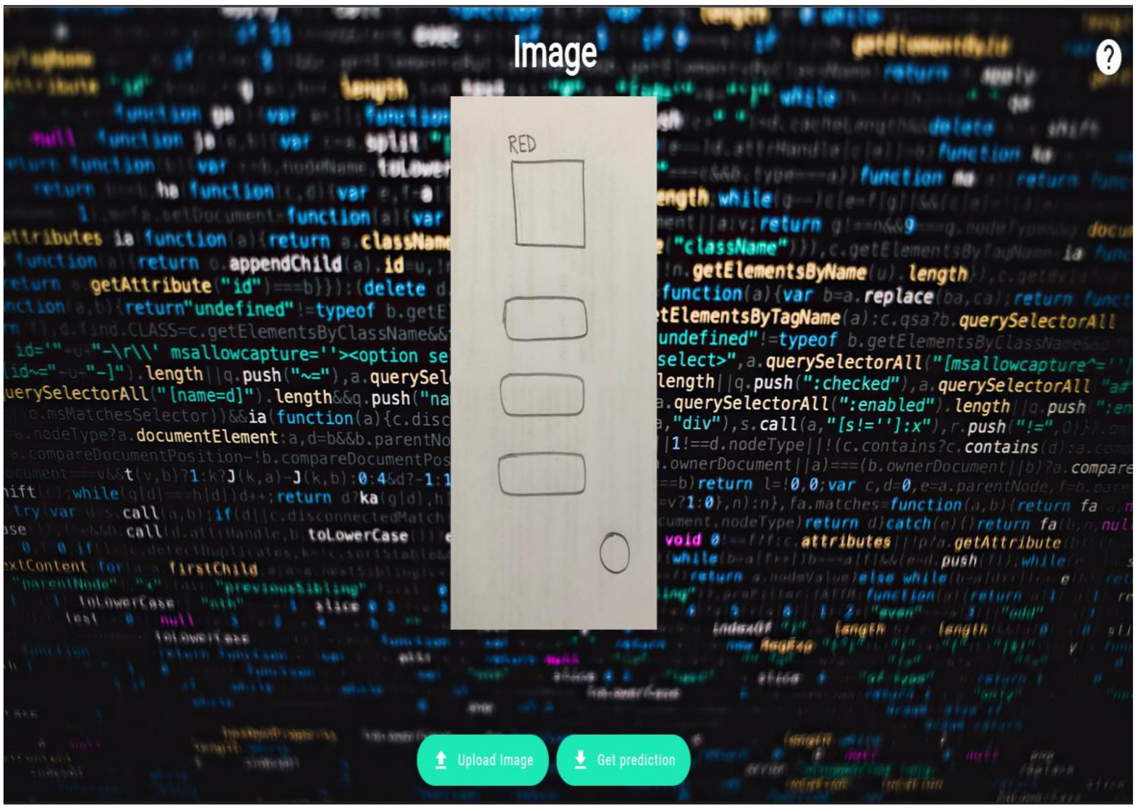


fig 6. 5:slider 2

6.6 Second Screen

The second screen also has 2 buttons. One button (see prediction) takes a bounded box image and navigates to the third screen to show the corresponding UI similar to the sketch, the other button (Download Dart code) will generate the corresponding dart code of the sketch. By clicking on the download dart code button our app generates a text file of the corresponding code of the sketch which the user draws on handwritten paper or they can also use some tool like paint to draw it. Our API detects it and we can download that code from the download code button and use this code in VS code or in android studio to run it in an emulator or mobile device.

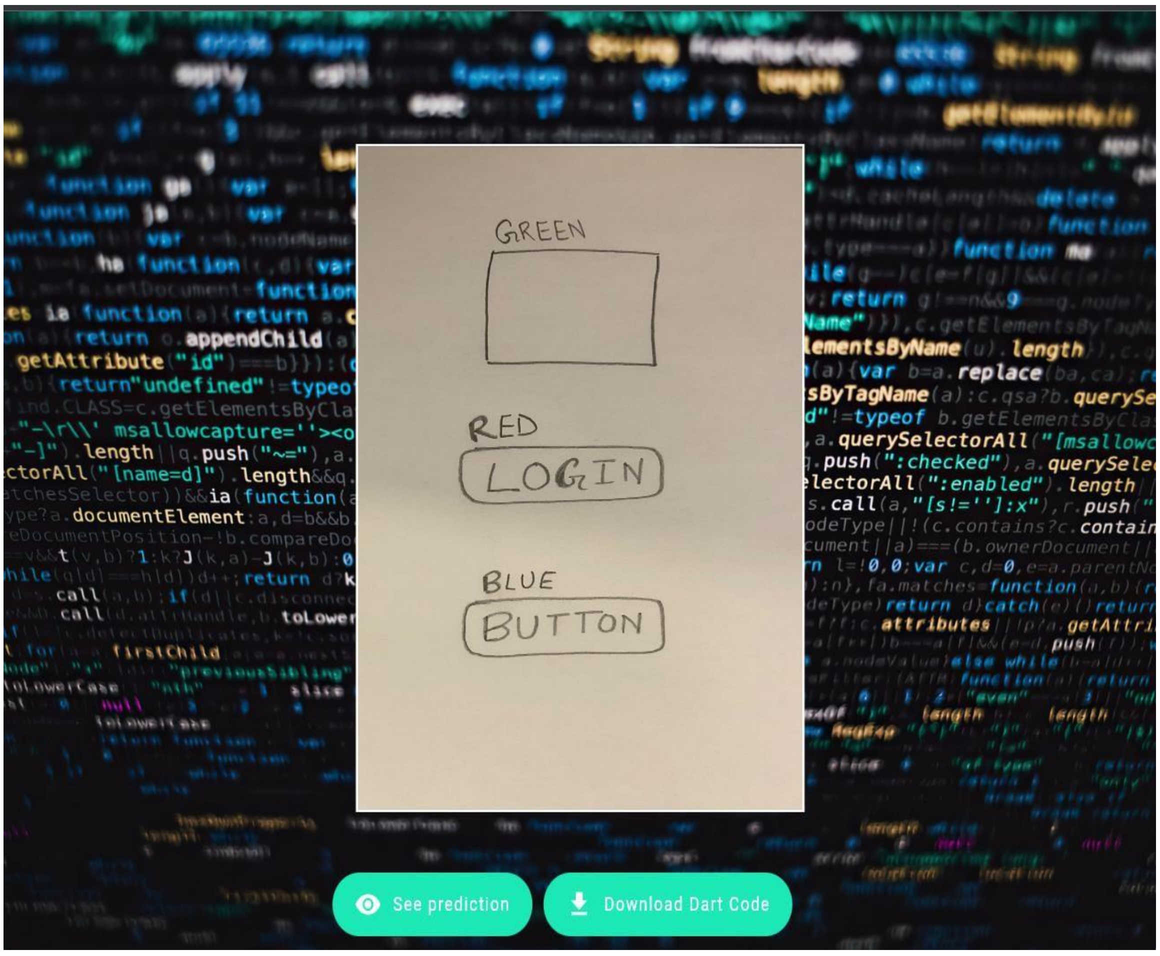


fig 6. 6: second screen of app

6.7 Dialog

This will appear when we will click on the see prediction button. It will contain images with a prediction. We used flutter Dialog in our project to aware the user can see their predictions by clicking on the set prediction button. This picture shows that our model accurately detects the container and 2 raised buttons along with the text. The bounded box around each widget shows that it predicts all the widgets accurately if it is designed or drawn according to the standard which we set to draw each widget.

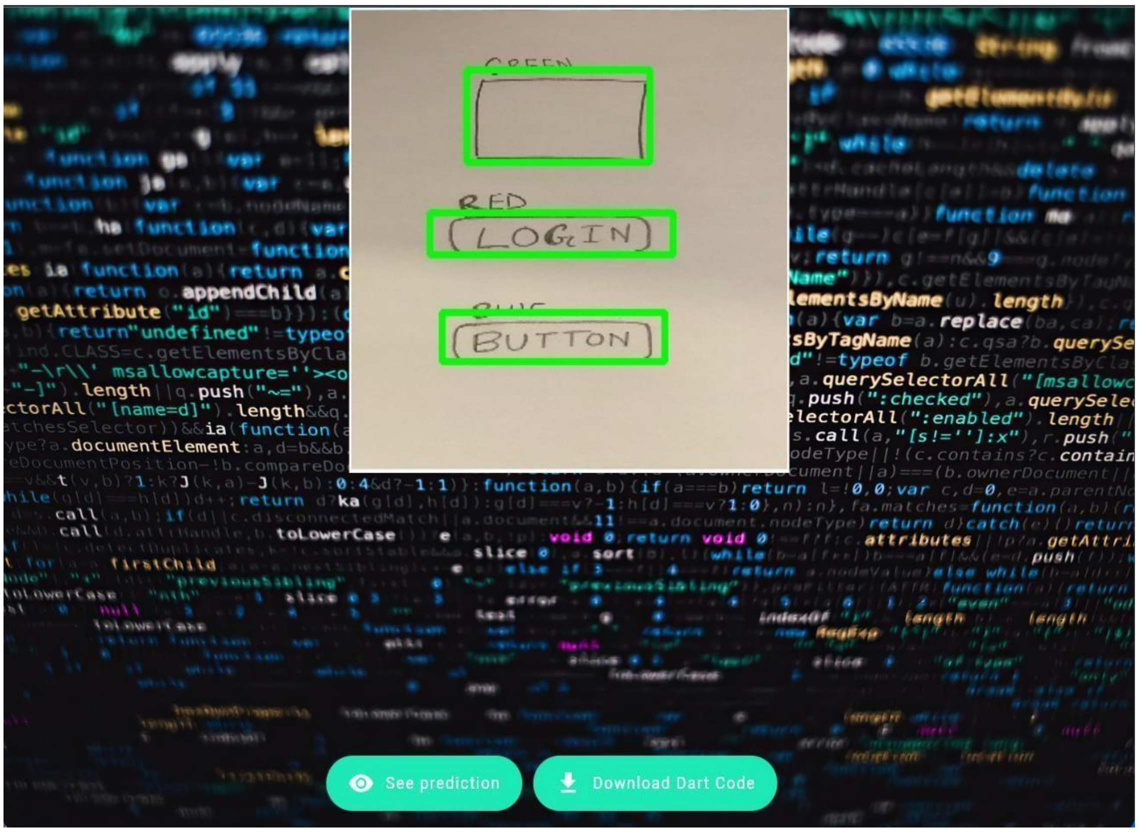


fig 6. 7: prediction in dialog box

6.8 Help Screen

This screen will contain the description about the project and also the guidelines which helps the user to understand how to draw widgets on paper with certain rules.

This screen will contain the description of the project and also the guidelines whichhelp the user to understand how to draw widgets on paper with certain rules. These guidelines clearly show the user that they can draw each image using this guideline and it helps them how to draw each widget which we set as a standard for our project. Container draws in the shape of a square, button drawn as a rectangular shape, Text field draw as a rectangular shape but also contain a line in it, image drawn as a square shape with a cross in it and last floating button draw as a circle shape with a plus sign in it.

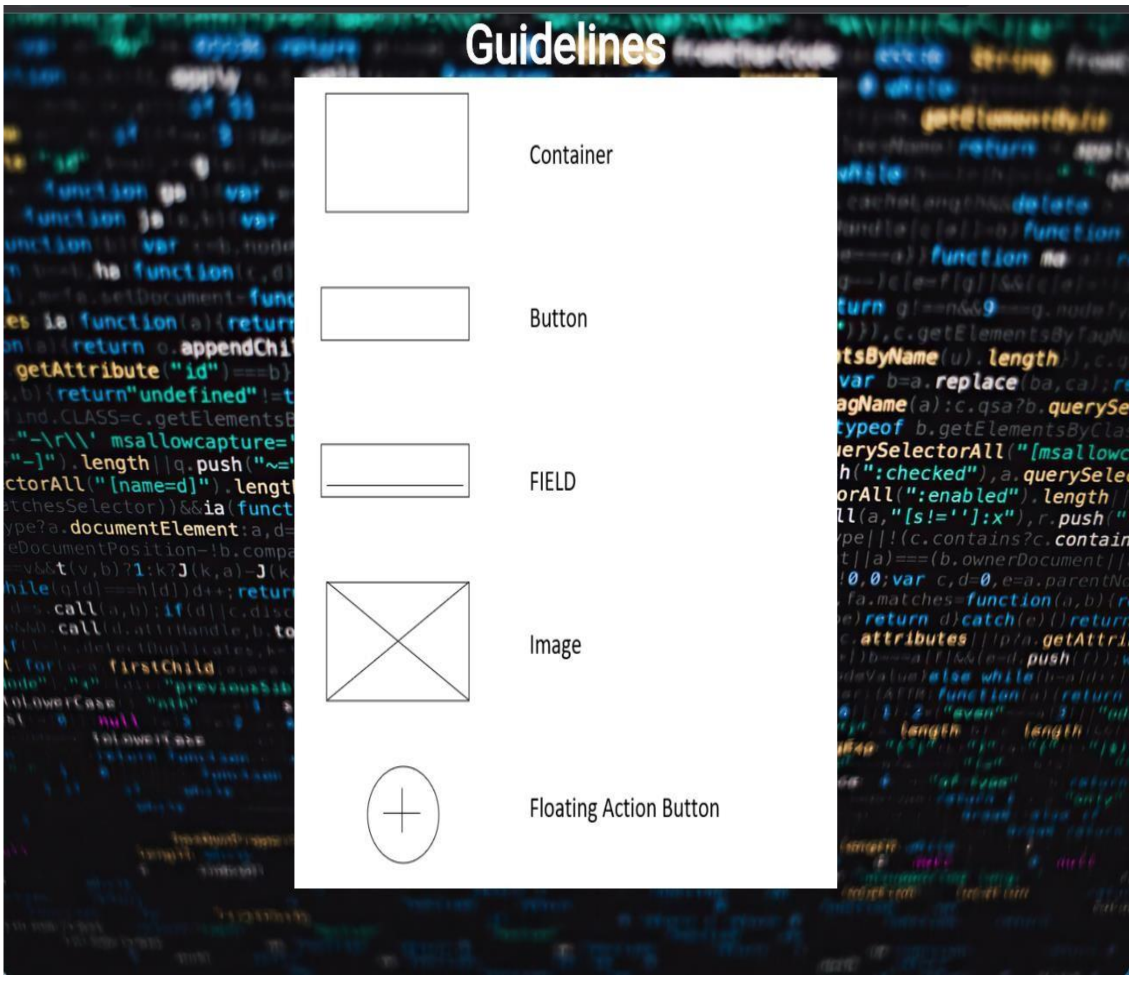
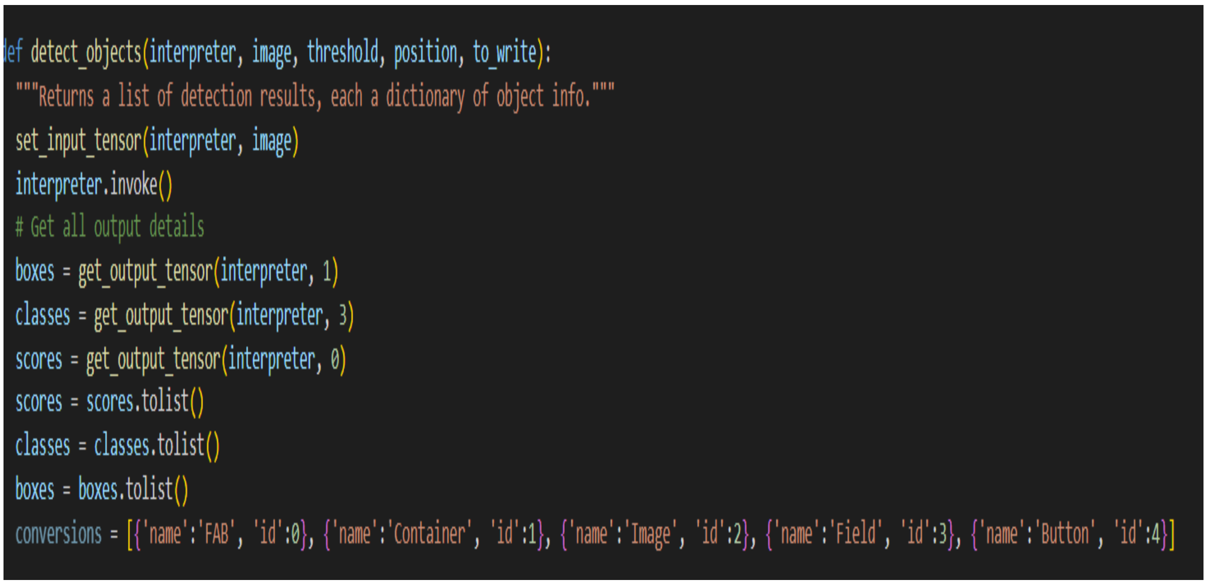


fig 6. 8: help screen

Chapter 7

API Integration and deployment

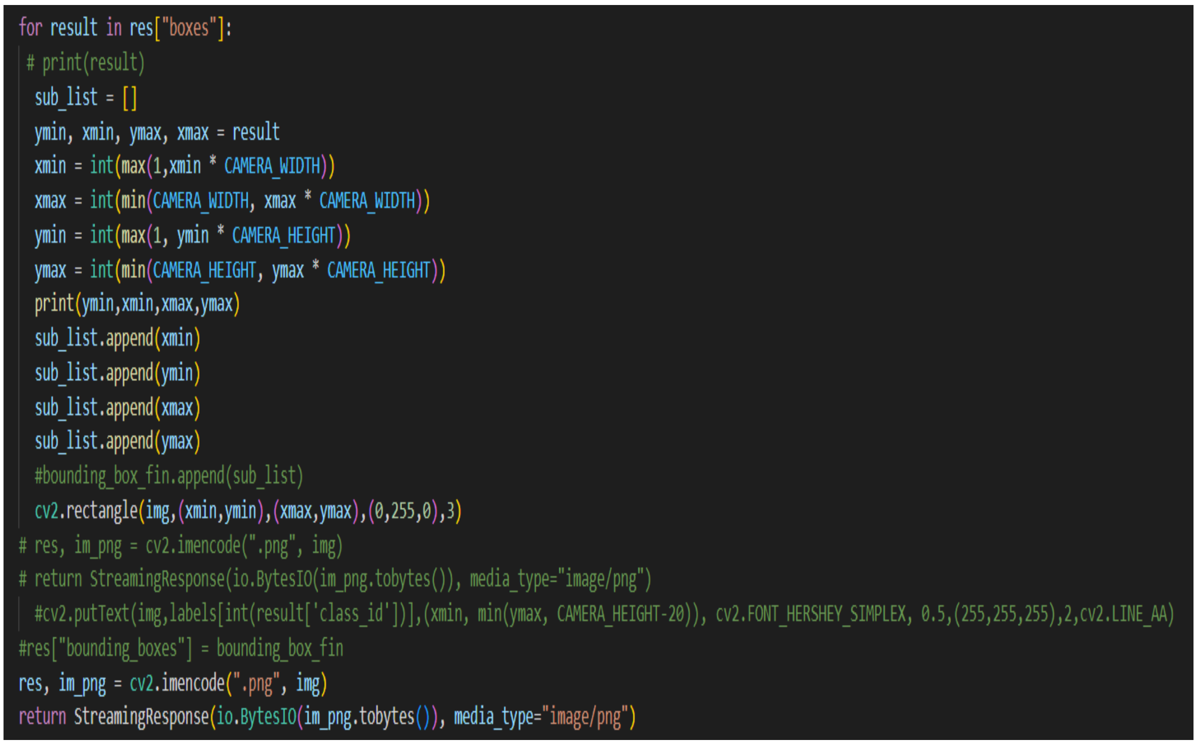
In this chapter there are 2 parts, the first one is API Integration and the second part is API deployment. In the API integration part, we have 2 APIS the first API takes an image as an input similarly the second API is also taking an image as an input. Our web application takes an image as an input, the image contains five widgets which are container, raised button, floating action button, image and text.



```
ef detect_objects(interpreter, image, threshold, position, to_write):
  """Returns a list of detection results, each a dictionary of object info."""
  set_input_tensor(interpreter, image)
  interpreter.invoke()
  # Get all output details
  boxes = get_output_tensor(interpreter, 1)
  classes = get_output_tensor(interpreter, 3)
  scores = get_output_tensor(interpreter, 0)
  scores = scores.tolist()
  classes = classes.tolist()
  boxes = boxes.tolist()
  conversions = [{'name':'FAB', 'id':0}, {'name':'Container', 'id':1}, {'name':'Image', 'id':2}, {'name':'Field', 'id':3}, {'name':'Button', 'id':4}]
```

fig 7. 1: function for detecting classes

There all are widgets which are used in flutter as an frontend. The image is in jpg format. Our web app takes that image as an input and in the backend, we used the Fast framework to write an API in python programming language. Fast is a popular framework for building web applications. While an open-source framework, Fast API is fully production-ready, with excellent documentation, support, and an easy-to-use interface. It can be used to build and run applications that are as fast as those written in other scripting languages. In fast API we integrated the TensorFlow object detection model to train our model. The TensorFlow object detection API is the framework for creating a deep learning network that solves object detection problems. The TensorFlow Object Detection API is an open-source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models. The TensorFlow object detection model is a pretrained model which is trained on different objects. In the backend we used the Tfod model which is integrated with our flask API, that model applies the algorithm to our image which we give as input to our web app and gives the prediction of the image as an output.



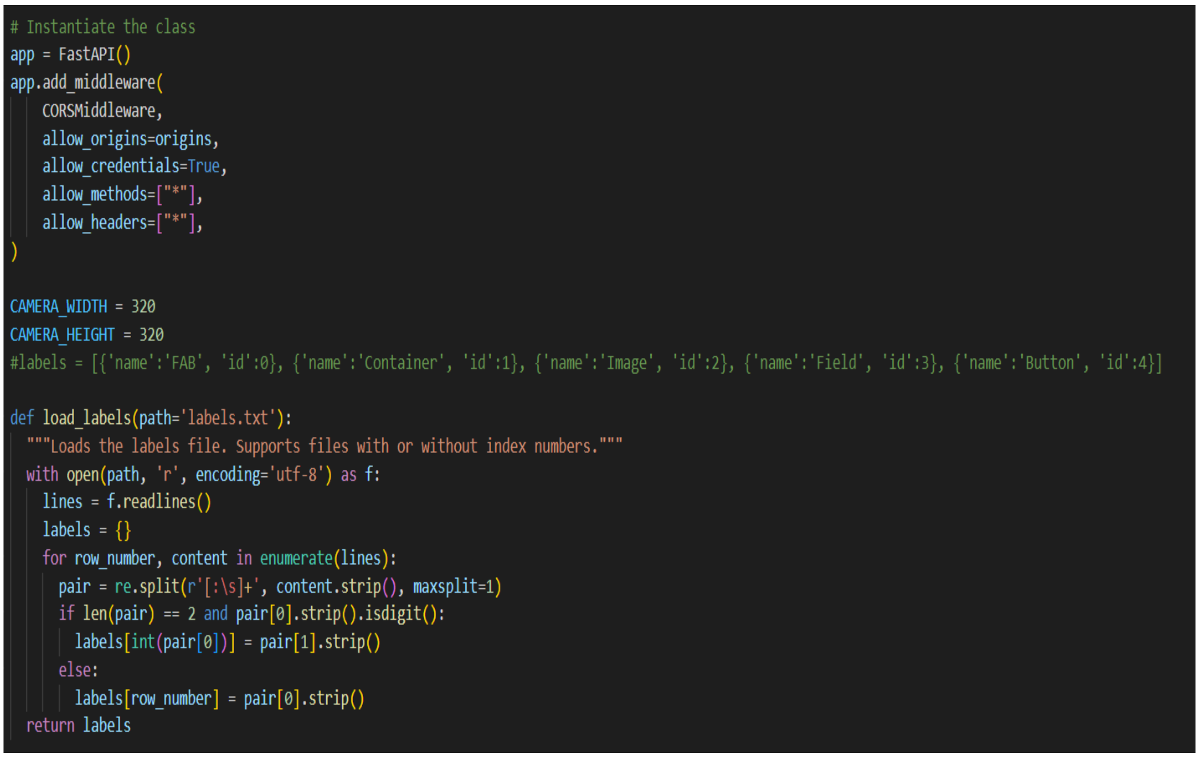
```
for result in res["boxes"]:
  # print(result)
    sub_list = []
    ymin, xmin, ymax, xmax = result
    xmin = int(max(1,xmin * CAMERA_WIDTH))
    xmax = int(min(CAMERA_WIDTH, xmax * CAMERA_WIDTH))
    ymin = int(max(1, ymin * CAMERA_HEIGHT))
    ymax = int(min(CAMERA_HEIGHT, ymax * CAMERA_HEIGHT))
    print(ymin,xmin,xmax,ymax)
    sub_list.append(xmin)
    sub_list.append(ymin)
    sub_list.append(xmax)
    sub_list.append(ymax)
    #bounding_box_fin.append(sub_list)
    cv2.rectangle(img,(xmin,ymin),(xmax,ymax),(0,255,0),3)
# res, im_png = cv2.imencode(".png", img)
# return StreamingResponse(io.BytesIO(im_png.tobytes()), media_type="image/png")
    #cv2.putText(img,labels[int(result['class_id'])],(xmin, min(ymax, CAMERA_HEIGHT-20)), cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255,255,255),2,cv2.LINE_AA)
#res["bounding_boxes"] = bounding_box_fin
res, im_png = cv2.imencode(".png", img)
return StreamingResponse(io.BytesIO(im_png.tobytes()), media_type="image/png")
```

fig 7. 2:function to give position to widgets

Our APIS and our model both are deployed on amazon web services. And we are accessing it using Linux commands. Amazon Web Services (AWS) is the world’s most comprehensive and broadly adopted cloud platform, offering over 200 fully featured services from data centers globally. Millions of customers—including the fastest-growing startups, largest enterprises, and leading government agencies—are using AWS to lower costs, become more agile, and innovate faster. The image which we give to our web app in the format of 320x320 pixels.

7.1 Fast API

Fast API is a up-to-date, extreme-acting netting foundation for construction APIs accompanying Python established standard type hints. It has the following key physiognomy: Fast to run: It offers very souped up. Fewer bugs: Reduce 40% chances of human errors. Easy: Minimize rule reproduction. Multiple facial characteristics each limit proclamation One of allure most important looks is that it is fast (for entity is contained in the name). The speed of the foundation is distinguished accompanying additional competitions like Node JS or Go. Also is fast to rule, easy to use, intuitive, and healthy.



```
# Instantiate the class
app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

CAMERA_WIDTH = 320
CAMERA_HEIGHT = 320
#labels = [{'name':'FAB', 'id':0}, {'name':'Container', 'id':1}, {'name':'Image', 'id':2}, {'name':'Field', 'id':3}, {'name':'Button', 'id':4}]

def load_labels(path='labels.txt'):
  """Loads the labels file. Supports files with or without index numbers."""
  with open(path, 'r', encoding='utf-8') as f:
    lines = f.readlines()
    labels = {}
    for row_number, content in enumerate(lines):
      pair = re.split(r'[:\s]+', content.strip(), maxsplit=1)
      if len(pair) == 2 and pair[0].strip().isdigit():
        labels[int(pair[0])] = pair[1].strip()
      else:
        labels[row_number] = pair[0].strip()
  return labels
```

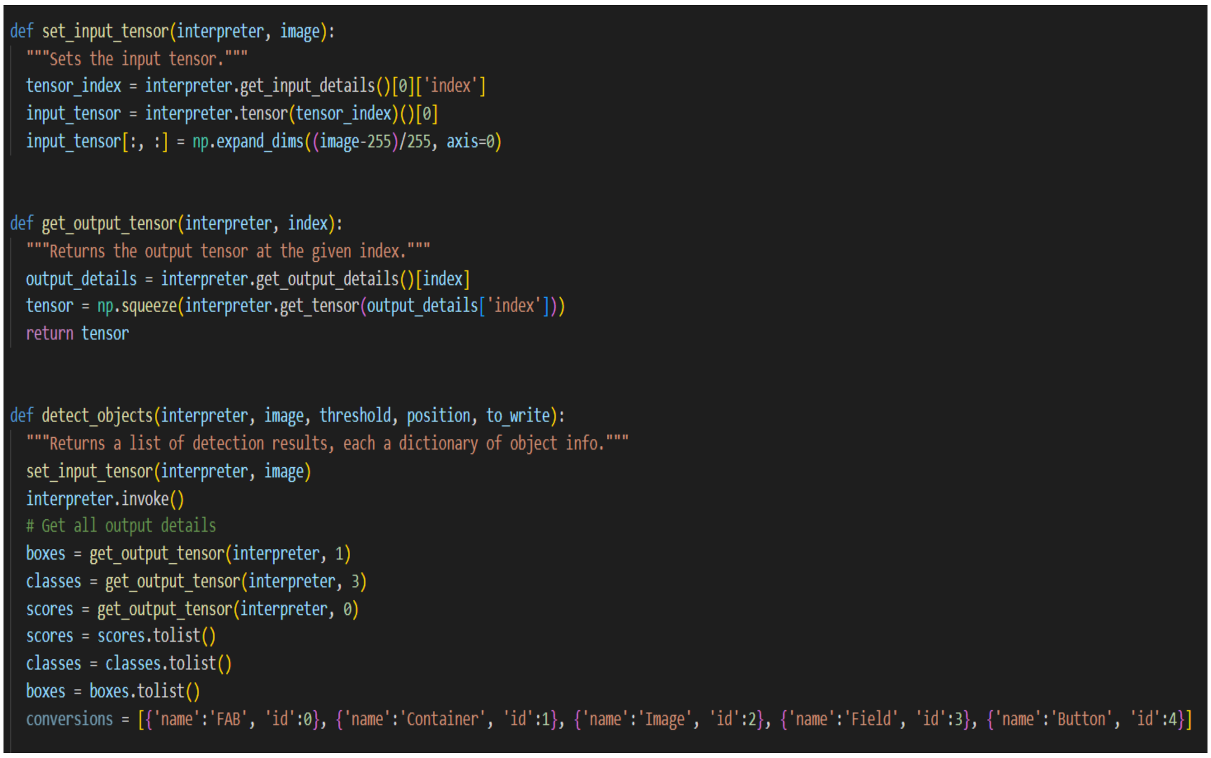
fig 7. 3: fast api code snippet

Fast API is a quiet API. REST API is an use set up connect that maybe secondhand by diversified customers to write accompanying a attendant. Rest API is a somewhat netting aid that stores and retrieves essential dossier. It supplies great adaptability to planners because it does not need some reliant rule athenaeums to approach computer network aids. REST APIs are stateless, cacheable, and logical. They're excellent for construction approximate-purpose and adaptable netting uses. Fast API is excellent for production-level work. It gives you a honestly fast habit to redistribute your ML to result accompanying unusually wonderful accomplishment. Fast API specifies a nearby form to makeup your request while consistency all the adaptability. In our project, we are utilizing the FAST

API for the unification of our model to the front end. Fast APIS gives united states of america response very fastly and it's too smooth to law it. We build our endpoints utilizing Fast APIS, skilled are 4 endpoints. In which 2 endpoints are for result level while the surplus 2 are for experiment purposes. We secondhand 2 endpoints for the experiment purpose cause we need few experiment point. We hit these endpoints on postal worker and catch the result of above 2 APIs. Here in the code below we have we can see the functions input_tensor, output_tensor and detect objects. The function detect_objects takes 4 parameters and gives the classes as outputs. These classes are numbered for example:

- 0 is for floating action button
- 1 is for container
- 2 is for field
- 3 is for button

So when we get response from JSON on index 1 we will get numbers through which we can identify the widgets so in the first index of JSON if we get 0 we know it floating action button, if we get 1 we know its container, if we get 2 we know it is field and if we get 3 we know its button. These five widgets we labeled with numbers before hand



```
def set_input_tensor(interpreter, image):
  """Sets the input tensor."""
  tensor_index = interpreter.get_input_details()[0]['index']
  input_tensor = interpreter.tensor(tensor_index)()[0]
  input_tensor[:, :] = np.expand_dims((image-255)/255, axis=0)


def get_output_tensor(interpreter, index):
  """Returns the output tensor at the given index."""
  output_details = interpreter.get_output_details()[index]
  tensor = np.squeeze(interpreter.get_tensor(output_details['index']))
  return tensor


def detect_objects(interpreter, image, threshold, position, to_write):
  """Returns a list of detection results, each a dictionary of object info."""
  set_input_tensor(interpreter, image)
  interpreter.invoke()
  # Get all output details
  boxes = get_output_tensor(interpreter, 1)
  classes = get_output_tensor(interpreter, 3)
  scores = get_output_tensor(interpreter, 0)
  scores = scores.tolist()
  classes = classes.tolist()
  boxes = boxes.tolist()
  conversions = [{'name':'FAB', 'id':0}, {'name':'Container', 'id':1}, {'name':'Image', 'id':2}, {'name':'Field', 'id':3}, {'name':'Button', 'id':4}]
```

fig 7. 4: fast api code snippte 2

7.2 First Endpoint

In the first API, we give an image as input through our web app, the image contains five widgets that are used in a flutter as a frontend. These widgets are image, container, text, raised button, and floating action button. The output of our API is also an image, an image that contains a bounded box in it. The output of the image contains a bounding box around each widget with a fixed length. We compare our output image with the input image, to see that our model predicts the right output and that the bounding box is located at the correct position. Given below is an output of our image which contains the bounding box around it. We can see that our model predict correctly all the widgets and located each bounding box at a correct position which shows that our first API predicts the correct result.

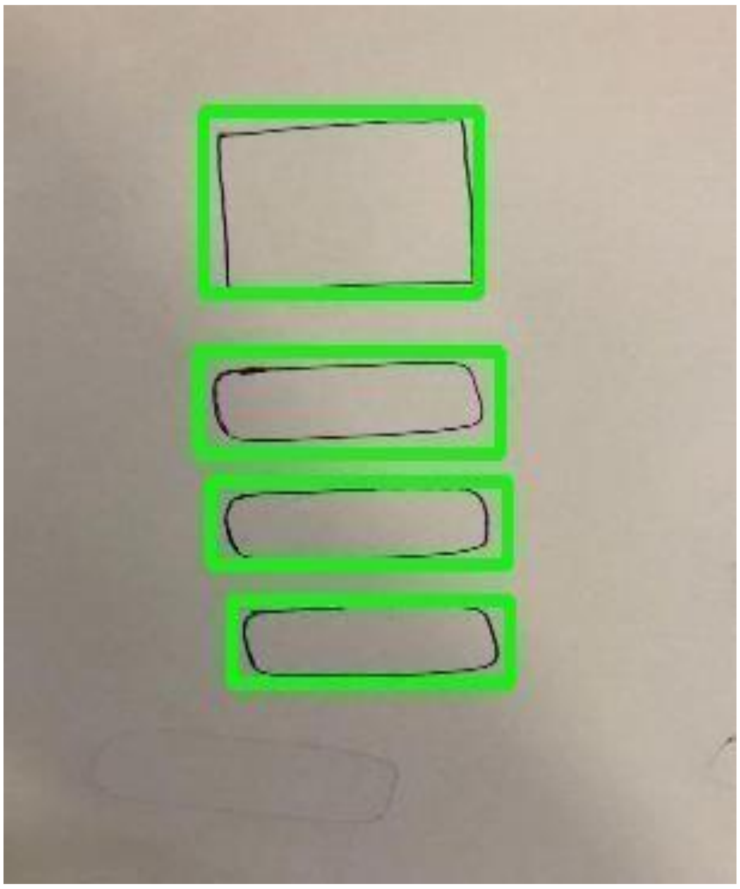


fig 7. 5: output from first enpoint of api

Now as you can see the API will receive image as input and will give detected widgets in image in its first endpoint.

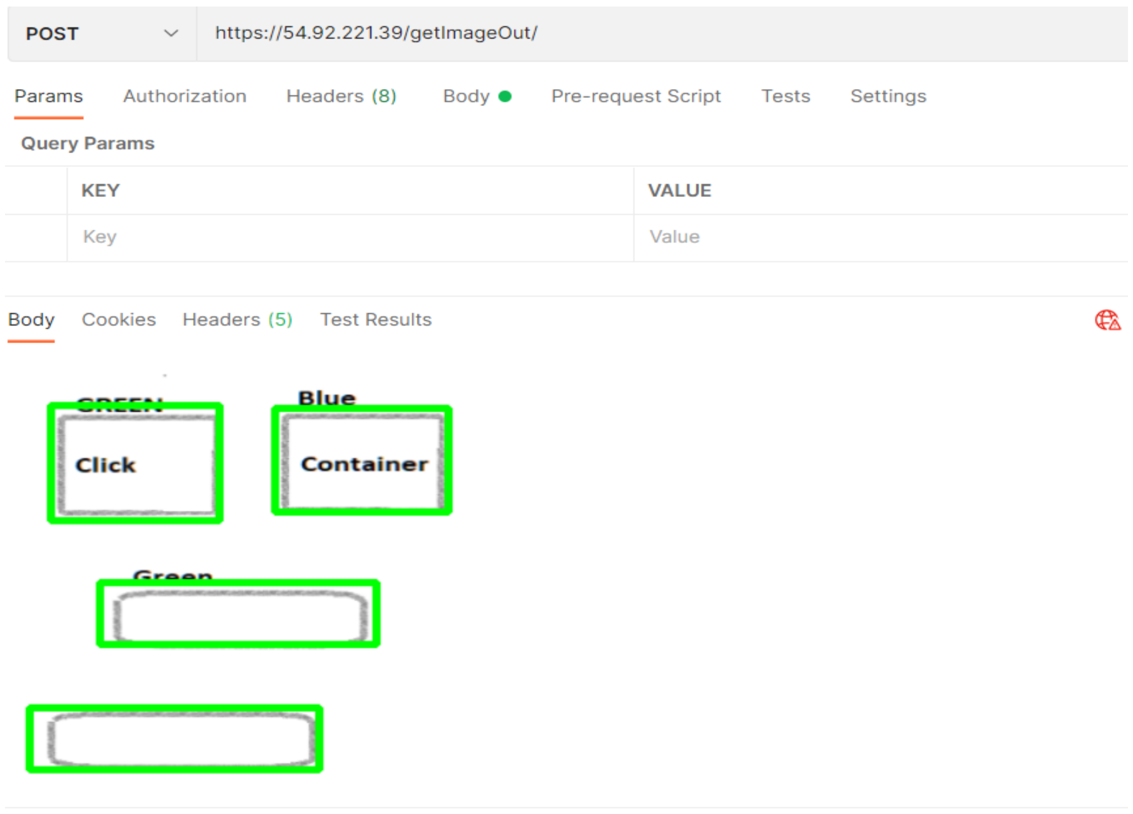
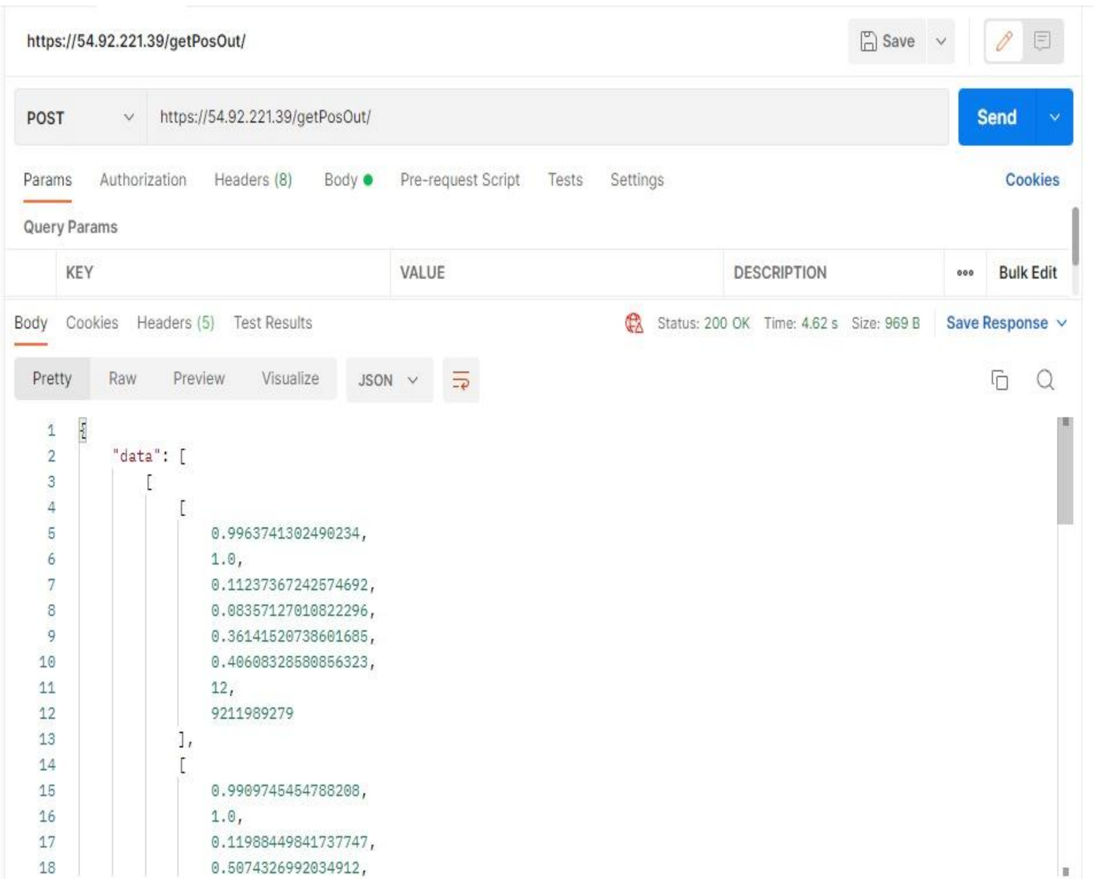


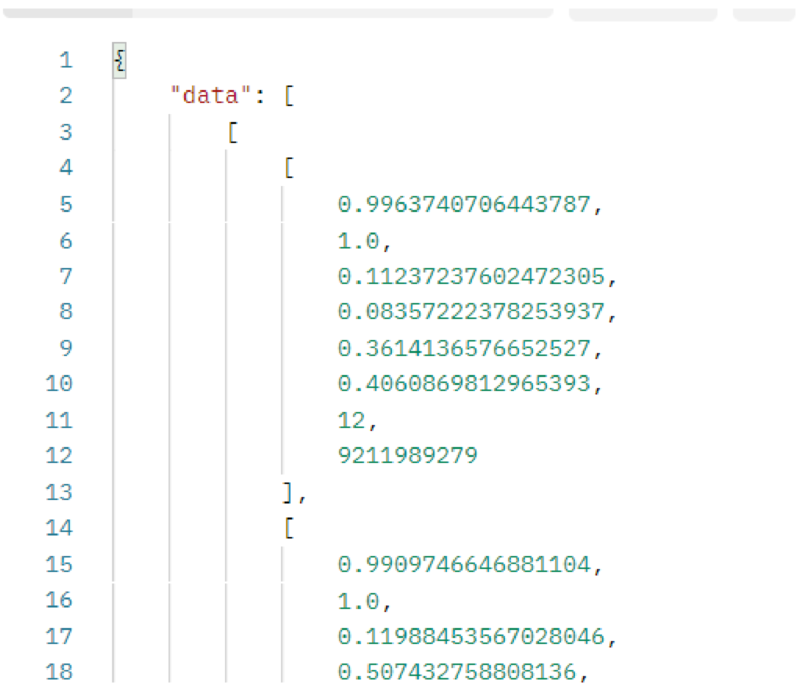
fig 7. 6: output shown in postman

7.3 Second Endpoint

In the second API, we gives also an image as input through our web app. But this time the output is JSON. Our API returns JSON as an output. The JSON contains an independent list of all the widgets which we are using in our web app. In each list, JSON contains the x and y axis i.e position of the widget along with its height and width. An independent list also contains the encoded value of the text, if the image contains the text widgets in it otherwise it is 0, and also contains the prediction of the widget in the percentage value. If the image contains the color in it. It will also give the value of that color otherwise gives 0. At last, we converted our prediction into dart code. This dart code is the output of our image and we match it with our input image. Below is the output of our image in JSON format.



```
{
    "data": [
        [
            [
                0.9963741302490234,
                1.0,
                0.11237367242574692,
                0.08357127010822296,
                0.36141520738601685,
                0.40608328580856323,
                12,
                9211989279
            ],
            [
                0.9909745454788208,
                1.0,
                0.11988449841737747,
                0.5074326992034912,
```



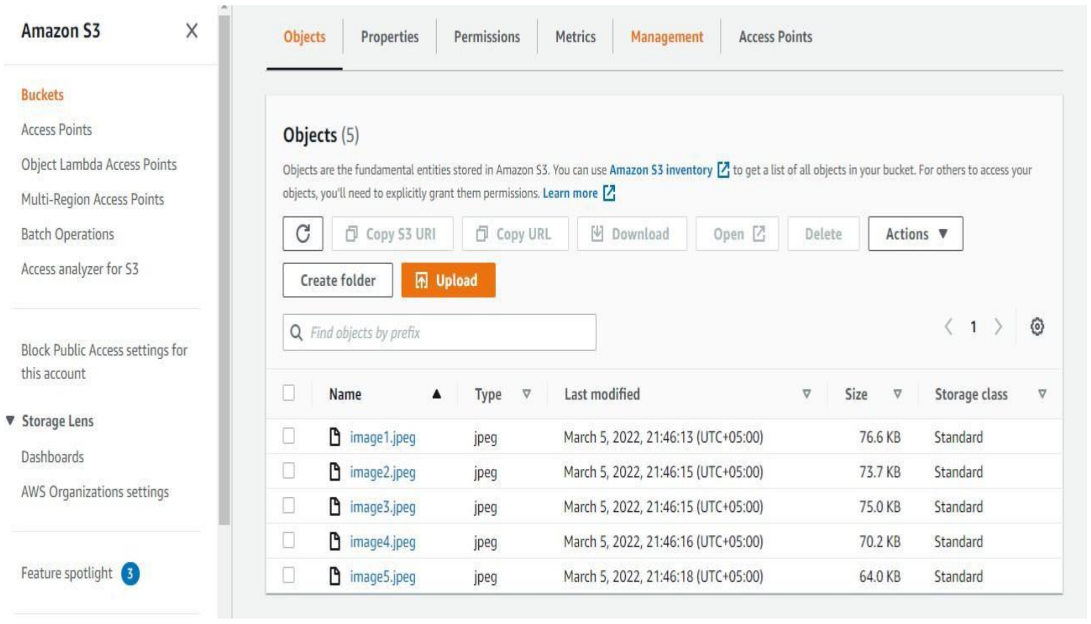
```
{
    "data": [
        [
            [
                0.9963740706443787,
                1.0,
                0.11237237602472305,
                0.08357222378253937,
                0.3614136576652527,
                0.4060869812965393,
                12,
                9211989279
            ],
            [
                0.9909746646881104,
                1.0,
                0.11988453567028046,
                0.507432758808136,
```

fig 7. 7: output of second endpoint of api

7.4 AWS

REST APIs and models both are deployed on AWS. We can access it through Linux commands. AWS is a cloud provider and we used the free tier of AWS. It was really handy for us and the compute power helped us a lot. We first started by using the ec2 instance. We chose ubuntu as we have worked on it before then we chose all the configurations and at the end we selected the security groups. The security groups are really important as we have to deliberately allow HTTP and HTTPS, ssh is default allowed because we need to ssh into the ubuntu machine in order to deploy the code

When we chose the particular instance then we focused on working with the networking settings of the instance. We had to first configure everything, obviously the part was not difficult at all as everything was pre done by AWS. We just needed to do some clicks and at the end we got the instance. When the instance was created then we need to wait for some minutes in order for AWS to reserve some space in their servers. We chose US EAST 2 as it has most of the services within it and we needed fast response time



Objects (5)

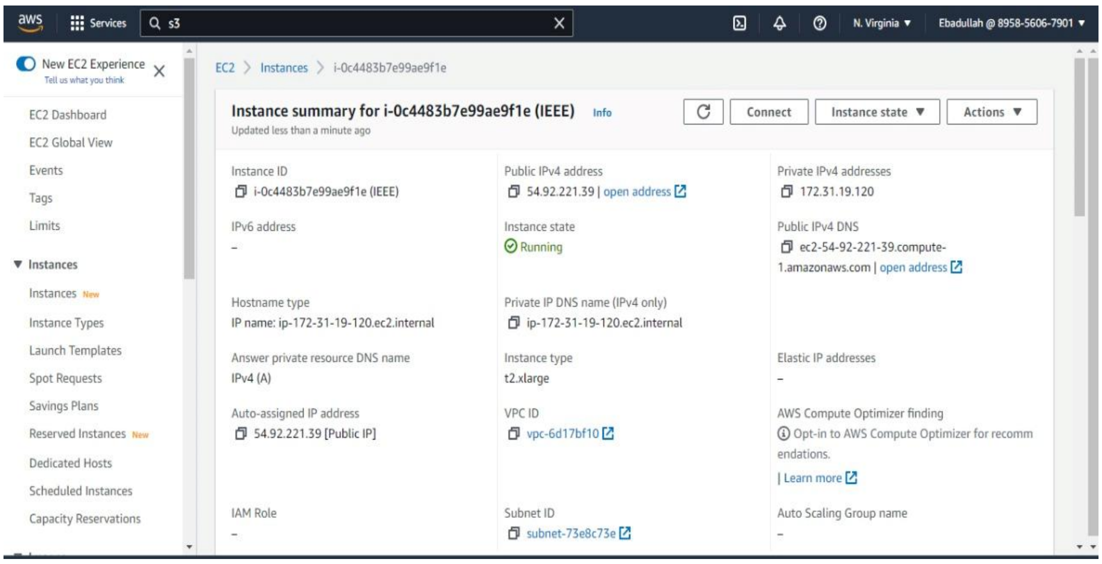
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
image1.jpeg	jpeg	March 5, 2022, 21:46:13 (UTC+05:00)	76.6 KB	Standard
image2.jpeg	jpeg	March 5, 2022, 21:46:15 (UTC+05:00)	73.7 KB	Standard
image3.jpeg	jpeg	March 5, 2022, 21:46:15 (UTC+05:00)	75.0 KB	Standard
image4.jpeg	jpeg	March 5, 2022, 21:46:16 (UTC+05:00)	70.2 KB	Standard
image5.jpeg	jpeg	March 5, 2022, 21:46:18 (UTC+05:00)	64.0 KB	Standard

fig 7. 8: AWS

The following screenshot is of AWS S3 bucket. This played a really important role in this project. While we were planning on making the project we thought that the data is very important and this is a prototype then several clients and stake holders will use this app hence we needed to save the data somehow. The easiest and quickest solution was AWS

S3 bucket, what it does is save files. The files can be anything, it can be an image, a file, a folder. Hence, we needed to save the images over here. We wrote a simple python program in order to take the image as input and change whatever we wanted to change and upload the bucket in our tenant. The image then comes into a particular folder specified by us before hand. The folder name and the file name must be given in order to save the image in S3 bucket. This code needed AWS authentication (AWS cli) to be installed in our computer and if we used AWS EC2 instance to deploy the code then we would need aws cli over there too. Aws cli requires an id and a password to authenticate if the person trying to use the service is someone. When the person is authenticated then we can use aws cli to use its resources. The aws cli is much more harder than aws portal, because over there we can just deploy everything with simple clicks but not everything can be done through aws portal right? We cannot automate image uploading to S3 bucket via aws portal hence we needed to use aws cli for this purpose

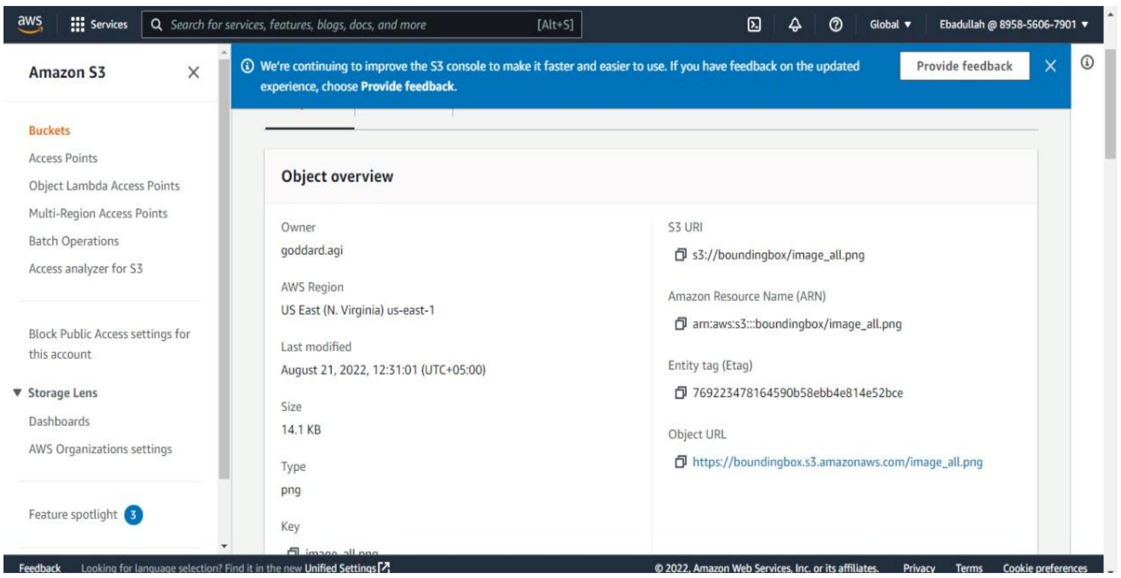


Instance summary for i-0c4483b7e99ae9f1e (IEEE)
Updated less than a minute ago

Instance ID: i-0c4483b7e99ae9f1e (IEEE)
Public IPv4 address: 54.92.221.39 | open address
Private IPv4 addresses: 172.31.19.120
IPv6 address: –
Instance state: Running
Public IPv4 DNS: ec2-54-92-221-39.compute-1.amazonaws.com | open address
Hostname type: IP name: ip-172-31-19-120.ec2.internal
Private IP DNS name (IPv4 only): ip-172-31-19-120.ec2.internal
Answer private resource DNS name: IPv4 (A)
Instance type: t2.xlarge
Elastic IP addresses: –
Auto-assigned IP address: 54.92.221.39 [Public IP]
VPC ID: vpc-6d17bf10
AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for recommendations. | Learn more
IAM Role: –
Subnet ID: subnet-73e8c73e
Auto Scaling Group name: –

fig 7. 9: AWS EC2 instance

This is the next page of aws s3 bucket. Now what it shows is really simple. We uploaded the image from our code in the ec2 instance or even our local computer. Now if the code ran successfully then we will be able to see the image when we refresh the aws ss3 portal page. If we see the image name then we can click on it and we will be directed to another page for that image. All the information regarding that image will be shown in this page like for example there is an image url which we can just copy and paste it in the browerr and wee can see it easily. So in a matter of seconds we are able to upload the image and see it worldwide. This is the power of the cloud. With just some clicks you are available worldwide. There are more things regarding the image like its unique identifier. The image can be uniquely identified within aws and outside aws too. This is a very important feature. No two images can overlap. The one thing we need to take care of is the name of any two images should not repeat within a same bucket (folder) If this happens then the code will return error


aws
Services
Search for services, features, blogs, docs, and more [Alt+S]
Global
Ebadullah @ 8958-5606-7901
Amazon S3
Buckets
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
Access analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
AWS Organizations settings
Feature spotlight 3
Provide feedback
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

We're continuing to improve the S3 console to make it faster and easier to use. If you have feedback on the updated experience, choose **Provide feedback**.

Object overview

Owner
goddard.agi

AWS Region
US East (N. Virginia) us-east-1

Last modified
August 21, 2022, 12:31:01 (UTC+05:00)

Size
14.1 KB

Type
png

Key
image_all.png

S3 URI
s3://boundingbox/image_all.png

Amazon Resource Name (ARN)
arn:aws:s3:::boundingbox/image_all.png

Entity tag (Etag)
769223478164590b58ebb4e814e52bce

Object URL
https://boundingbox.s3.amazonaws.com/image_all.png

fig 7. 10: AWS s3 bucket

Following is a very simple AWS flow. The user interacts with the web application and then the user also has an option of either choosing the tfod api or the yolo api. For the sake of this prototype we just went forward with tfod model. So in this app there will be no option to select but as this is a high level flow so we mentioned it in here. After the api is hit it will take the image as input and produce the outputs. The output will contain bounding box positions and the prediction of the widget with a certain confidence. The response will also return the text (if any) and color (if any). So if there is a text written inside a widget then the response json will contain that particular text in it. And if there is a color name written over the widget then the json will include the color name in it. Then there are some very complex mathematical equations which are needed in order to convert those positions into code. We need to create the widgets inn the eaxt same position the returned by the model endpoint

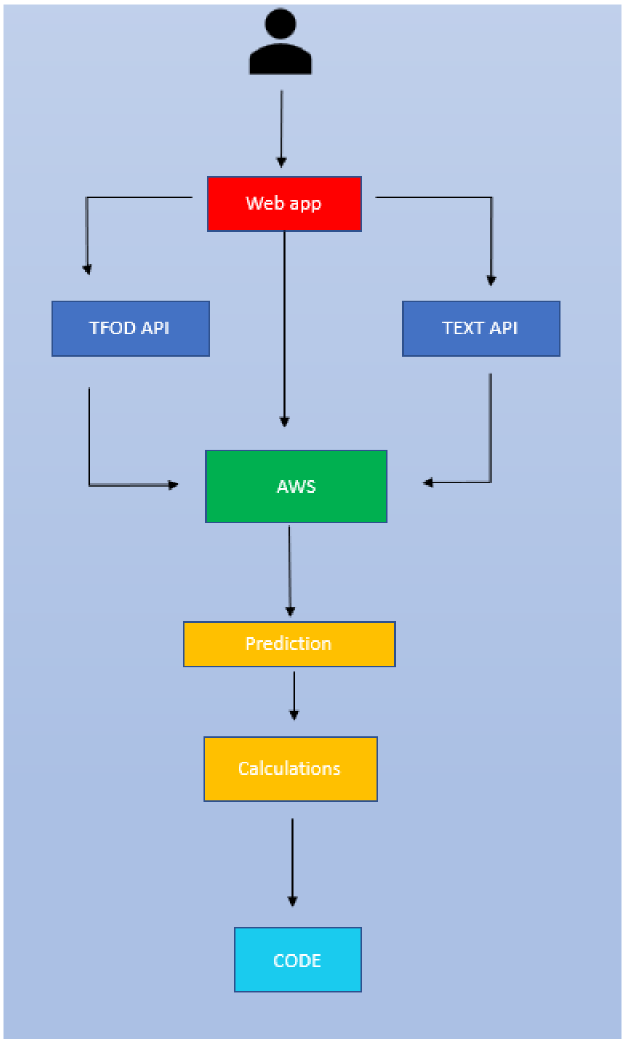


fig 7. 11: AWS flow

This is a more in-depth architecture of the project. Now let us start from the beginning. The user first interacts with our application. The application resides on a laptop and then the application is connected to other apis deployed on the cloud. So when the user draws a sketch of the UI, then the sketch is to be uploaded in the web application, we can also select any test data from the application itself. when we do that then we click on the predictions button, the api is called and two endpoints of that api are called at the backend. The first endpoint retrieves the image with the bounding boxes and the second api retrieves the json with positions of those detected widgets with their names and if there is any text or color then that too, so the main work starts now. We take those positions and convert them into code. We got the places where the widgets must be so now we have to create those widgets at those positions, now obviously this task must be dynamic and hence we could not pre make the code for it to be downloadable.

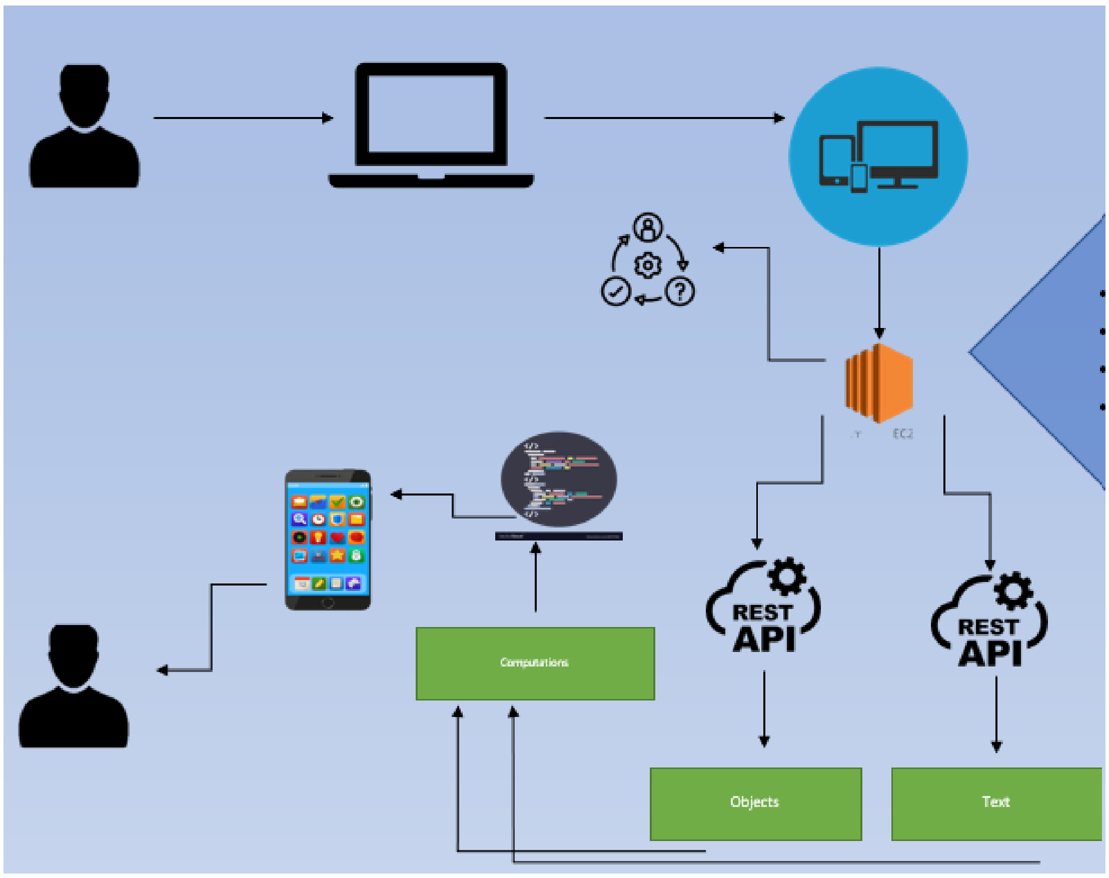


fig 7. 12: data travelling through AWS

This architecture was displayed by us at the mid-year presentation. We were planning on using several aws services but due to some cost cuts and already built code we didn't use these. Still we will discuss this over here. We used aws sagemaker to train the tensorflow model and hence the model was trained very quickly. If we used colab then yes it has a gpu but a very slow one and google also reduces the time to restart kernel if we use it again and again. We were planning on to deploying the web app onto aws amplify but due to time lacking we could not do it. There are kind of three apis in this project. One is the object detection one which gives us the bounding box image and also the json response of those bounding boxes. Then we have a text recognizer api, although it is integrated in one now because we were having cost issues so to reduce the cost of another virtual machine we did the work in the same one. Then there is an api which takes the response from the object detection api,, does several computations and somehow makes the code to be generated. The text file is produced in the output and the user can download the text file and use it in their code.

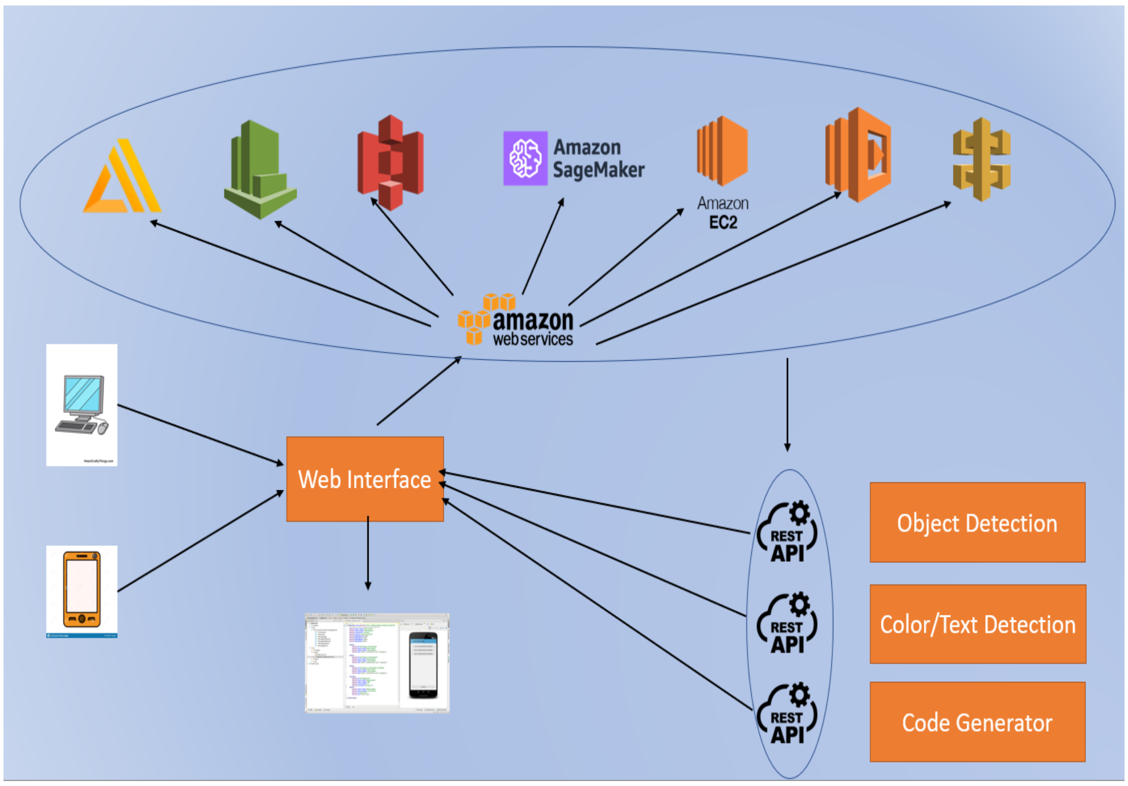


fig 7. 13: AWS architechture

Chapter 8
Conclusion

Designers face a challenge when turning their wireframes into law, this frequently involves passing the design to a builder and bearing the builder implement the typical program that controls display (GUI) code. This project decreases moment of truth necessary for this stage and most basically decreases the customer-developer arrangement cause this is place ultimate issues are formed, as the developers are adept systematize and making uses, but particularly relating with the customer and imitating what they need is completely another act. The project has more removed the necessity of a planner for primary originals, allowing planners to devote effort to something the use rationale alternatively boilerplate GUI rule.

This project uses computer vision techniques to sort out this problem and train the model in such a way that it generates a code of the UI interface. This project also uses other computer vision techniques to recognize colors and handwritten text on the sketch of the user interface.

Through a study of site design practice, we noticed that site designers design sites at various levels of refinement—site pictures, illustration, and individual pages— what designers sketch by any means levels all the while at the beginning of design. Informed by our study, we planned and executed a prototyping form to assist designers in the inception of site design. Typical tasks contain designating individual or more worldwide labels to the program and designating an individual or more labels for each frame inside the broadcast. [Balakrishnan Varadarajan] Being capable of thinking to characterize the content of a concept utilizing correctly made English sentences is a very questioning task, but it takes care of have an excellent impact, instance by a portion of food optically injured society better comprehend the content of figures on computer network.

Inour first state, we point in a direction five gadgets for UI discovery, we draw each gadget in theory accompanying various alliances, then accepted a exact likeness each page and secondhand algorithms for the discovery of each concept. The purpose concerning this project search out draw the representations of the UI in theory and take a exact likeness it, that is the recommendation of our model, and search out produce the rule of the UI in a scamper language that is secondhand for movable, netting, and personal computer requests, the law is the productivity of our model. The harvest of our model holds 2 representations, the first countenance holding the law of the UI and the second profit is the image containing the bounding box around each widget which gives a percentage of the occurrence of each widget.

A dataset has been created which contains the sketches and then we have created the corresponding codes for those sketches. The image of the sketches has normalized and then we have augmented the images and optimized the code for the sketches so they can be referred to independently. The proposed model has been deployed as a web service in which the client can upload a sketch and the code will be generated with the UI.

8.1 Future Addition

Now let us talk about some future additions which we can do in this project. There are several limitations right now. The biggest limitation is the number of widgets. The client do not much vast options to choose the widgets right now. So thee widgets we will increase to are:

- Paragraphs
- Carousal
- Navigation bar
- Search bar
- Logos

The paragraphs can help those who want to add text in their UI. There are several user interfaces which are developed without a paragraph in them but there is no UI without short sentences in them. So this feature is very important in the later stages. Then the carousal is towards a more fancy UI. The people who want to show images or any other stuff in a very small space they use carousals. The navigation bar is also very important. It will show all the buttons to be clickable and all the other important links. The search bar is also vastly used in every UI. People need to search in order to go somewhere hence thy will need a search bar

The other addition is training the text model on handwritten text. For now, we took a pre-trained model and just integrated it in our application. The model works great but for someone who has a bad handwriting or who writes very closely so then that will not be detected or even if it gets detected then it won't be that accurate. So for this purpose we need to train the model with some handwritten text. And for this s3 bucket will help a lot.

As we discussed earlier that we will be storing the images inside the s3 bucket hence that work will help us a lot over here.

When we store those images then we can simply connect that s3 bucket with our text recognition model and input the images from there. Also, this is a very long task as we will need to first label the images and then train it. Although, this was planned to be done in this project but we could not do it due to the lack of online resources.

The other feature to be added is the deployment of the web application. We need to do that as we need the clients to be able to open the application in their mobile browser. When we do that then that can simply open the application in their mobile, take an image of the sketch and upload it directly. No other work needed

References

1) Alex Robinson. (2019). Sketch2code: Generating a website from a paper mockup.
2) Balakrishnan Varadarajan. (2015). Efficient Large Scale Video Classification.
3) Chao Dong. (2015). Image Super- Resolution Using Deep Convolutional Networks.
4) J.A. Landay and B.A Mayers. (2001). Sketching interfaces: toward more human interface design.
5) james Lin. (2000). DENIM: Finding a Tighter Fit Between Tools and Practice for Web Site Design.
6) Oriol Vinyals. (2014). Show and Tell: A Neural Image Caption Generator.