A MINI PROJECT REPORT

ON

"Image Classification Using Convolutional Neural Networks"

Submitted to

SAVITRIBAI PHULE PUNE UNIVERSITY

in completion of
SKILL DEVELOPMENT LABORATORY
(T.E Computer Engineering)

BY

Yogiraj Gutte	Exam No: 71906165C
Aasit Vora	Exam No: 71905873C

Sinhgad Institutes

Department of Computer Engineering
Sinhgad College of Engineering, Pune-41
Accredited by NAAC with grade 'A'

YEAR 2020-21

CERTIFICATE

Sinhgad Technical Education Society,
Department of Computer Engineering
Sinhgad College of Engineering, Pune-41
Accredited by NAAC with grade 'A'

"Image Classification Using Convolutional Neural Networks"

Submitted to

SAVITRIBAI PHULE PUNE UNIVERSITY

in completion of

SKILL DEVELOPMENT LABORATORY
(T.E Computer Engineering)

BY

Yogiraj Gutte Exam No: 71906165C
Aasit Vora Exam No: 71905873C

Prof. N. A. Mhetre
Internal Guide,
Department of Computer Engineering

Prof. M. P. Wankhade
Head of Department,
Department of Computer Engineering

Dr. S.D. Lokhande
Principal
SCOE, Pune

CONTENTS

Acknowledgement

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend our sincere thanks to all of them. We are highly indebted to **Prof Nalini Mhetre** for her guidance and constant supervision as well as for providing necessary information regarding the project & also for her support in completing the project. We would like to express our gratitude towards **Prof M. P. Wankhede, H.O.D. of Computer Engineering, SCOE** for their kind co-operation and encouragement which helped us in completion of this project.

We would like to express our special gratitude and thanks to all our colleagues for giving us such attention and time. Our thanks and appreciations also go to our friends in developing the project and people who have willingly helped us out with their abilities.

Abstract

Deep Learning has emerged as a new area in Machine Learning and is applied to a number of signal and image applications. The main purpose of the work presented in this is to apply the concept of a Deep Learning algorithm namely, Convolutional Neural Networks (CNN) in image classification. This algorithm is one of the most used algorithms for the purpose of image classification, object detection, climate prediction, etc. It has been trained and tested on a self-curated dataset which was procured from multiple sources. The performance of this algorithm is evaluated based on quality metric known as Mean Squared Error (MSE) and classification accuracy. The graphical representation of performance of this algorithm is given on the basis of validation accuracy against the number of training epochs. To make the algorithm usable by common people, a GUI based standalone application is also designed which implements the algorithm in different ways. This project has been carried out with a view to ease people's lives and help them in various tasks that require image processing and classification.

List of Figures

List of Tables

1. INTRODUCTION

1.1 Background and basics

1.1.1 Computer Vision – Introduction

As humans, we are capable of understanding and describing a scene encapsulated in an image. This involves much more than just detecting the objects in front of us [1]. We can make our own reasonable inferences after seen objects in the image such as their color, size, material, etc.

These are also the skills a computer vision system needs. In essence, given a two-dimensional image, a computer vision system must recognize the present objects and their characteristics such as shapes, textures, colors, sizes, spatial arrangements, patterns, among other things, to provide a description as complete as possible of the image.

Computer vision has applications in various fields. It is used in field of biometrics for facial recognition; in industries to detect manufacturing defects in the products; in the field of healthcare for detection of visible anomalies like tumors, fractures, etc.; in autonomous vehicles to detect objects and people; in defence systems for detecting suspicious activities using satellites; and in many other fields in this era.

1.1.2 Image Processing

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core

research area within engineering and computer science disciplines too. Many image processing algorithms are used in computer vision for solving its core tasks [2].

Image processing basically includes the following three steps:

- Importing the image via image acquisition tools;
- Analyzing and manipulating the image;
- Output in which result can be altered image or report that is based on image analysis

1.1.3 Deep Learning

Deep Learning is a subset of Machine Learning which utilizes concept of Neural Networks to solve some special class of problems. It uses multiple number of neural layers to process large amount of data and provide expected results, in most cases predictions. Deep Learning has applications in following domains:

- Computer Vision
- Speech Recognition
- Language Translation
- Automatic Game Playing
- Text Processing

Among all these uses, we have used Deep Learning for the purpose of image recognition in this project. Deep Learning consists of many algorithms. Following are the most widely used Deep Learning algorithms:

- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)
- Long Short-Term Memory Networks (LSTMs)

1.1.4 Convolutional Neural Networks (CNN)

Convolutional Neural Networks or CNNs, are a special kind of neural network for processing data that has a known, grid-like topology. Examples include time-series data, which can be thought of as a 1D grid taking samples at regular time intervals, and image data, which can be thought of as a 2D grid of pixels [3]. The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. It is an operation on two functions which produces a third function that expresses how the shape of one is modified by the other.

The CNN is a combination of two basic building blocks:

1. The Convolution Block – Consists of the Convolution Layer and the Pooling Layer. This layer forms the essential component of Feature-Extraction.
2. The Fully Connected Block – Consists of a fully connected simple neural network architecture. This layer performs the task of Classification based on the input from the convolutional block.

Further sections contain detailed functioning of CNN and all the layers associated with it.

1.2 Problem Statement

Image Classification is one of the most important technologies used today. It solves many problems and gives insight about any image that is given as input. Even after so much progress in the field of image classification, there are very few options available when it comes to the usage of this technology by common people. Common people with no knowledge of programming are not able to access image classification algorithms. Also, there are very few tools available for it but they come at heavy cost.

The purpose of this project is to make a commonly used image classification algorithm (CNN) accessible to masses by means of a simple GUI application, and at the same time to analyze and study the CNN algorithm by implementing it from scratch and through transfer learning. This project aims to study all aspects of CNN and some basics of neural networks.

1.2.1 Scope Statement

The objective of this project is to study about convolutional neural networks and its important aspects by training and testing a CNN model to identify images belonging to 10 different categories. Along with that, this project aims to deploy that model through means of an easy to use GUI application. For this purpose, a self-curated dataset of 74,204 images (55745 for training and 18459 for validation) belonging to 10 different categories has been used. The GUI application has been named as “What’s On My Screen”.

2. PROJECT PLANNING AND MANAGEMENT

2.1 Software and Hardware requirements

2.1.1 Software requirements

- Python 3.8 64 bit – For programming and training of the model
- TensorFlow 2.5 – Library for deep learning algorithms
- Tesseract OCR Engine – For extraction of text from image
- Nvidia Cuda Deep Neural Network library – For GPU based processing
- Chrome Driver – To interact with brower
- Pyperclip – To access clipboard of OS.
- Python Image Library – For operations of images
- Streamlit – For browser-based GUI
- PyQt5 – For window-based GUI
- Browser – GUI Platform

2.1.2 Hardware requirements

- Intel Core i3 (2^{nd} gen) or better
- Cuda compatible GPU
- 4 GB RAM
- 10 GB free space

2.2 Process Model

2.2.1 Building model from scratch

1. Read image dataset form corresponding location.
2. Apply desired transformations to images.
3. Build image data generator with suitable attributes.
4. Build model by adding suitable layers.
5. Compile the model.
6. Fit the image data to the model.
7. Save the model.

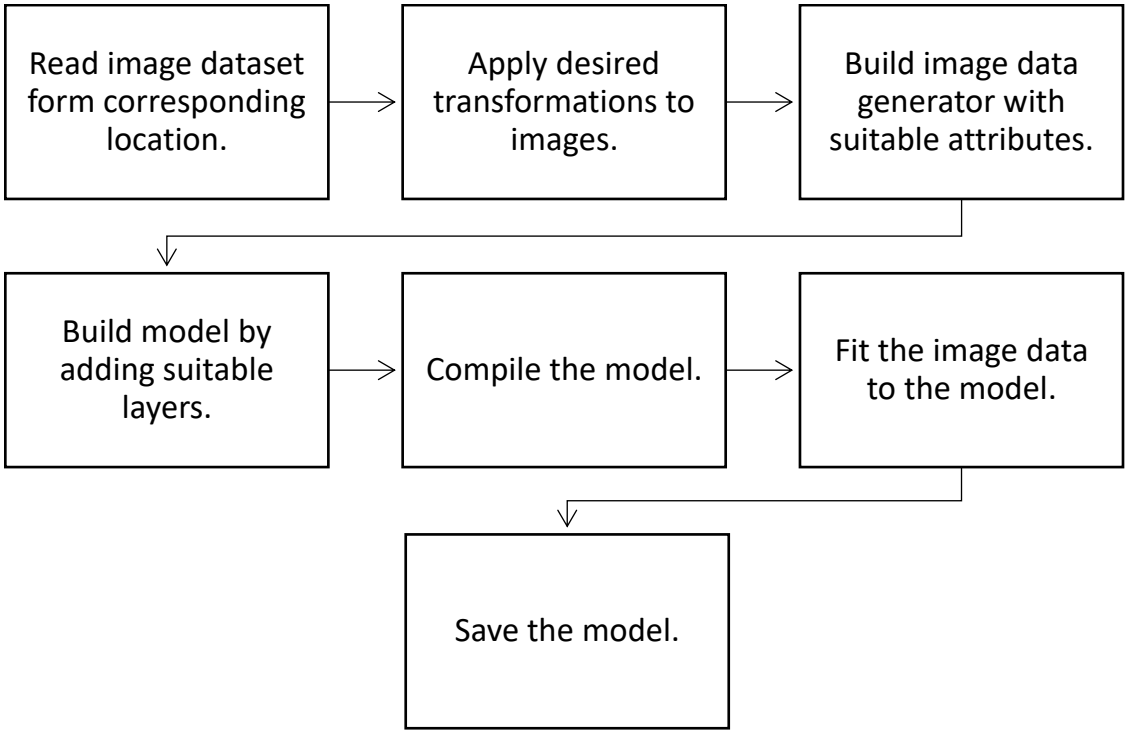

Fig. 2.1 Process of building model from scratch

2.2.2 Building model using Transfer Learning

1. Read image dataset form corresponding location.
2. Apply desired transformations to images.
3. Build image data generator with suitable attributes.
4. Import existing model.
5. Drop or freeze layers according to need.
6. Add additional layers if needed.
7. Compile the model.
8. Fit the image data to the model.
9. Save the model.

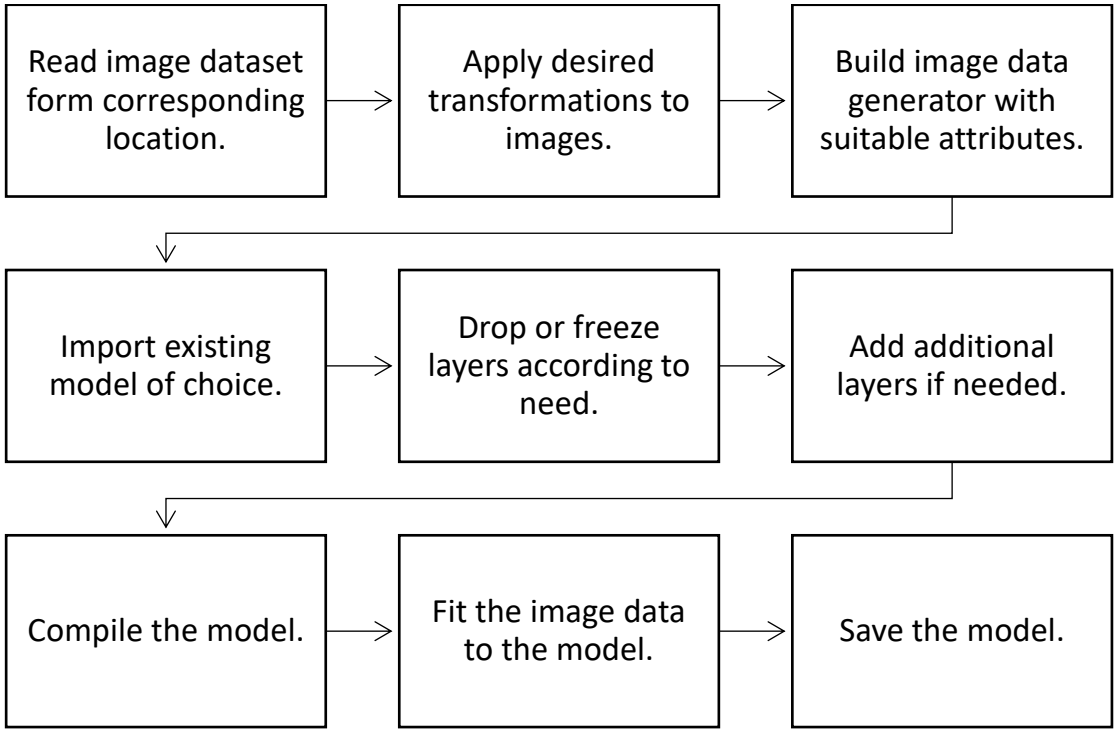

Fig. 2.2 Process of building model using Transfer Learning

3. ANALYSIS AND DESIGN

3.1 Use-Case Diagrams

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

3.1.1 What's on My Screen App

The user for this system is an ordinary person. The app provides three main functions as follows:

1. Identify Image: This function can be used by clicking on the corresponding button and selecting out the area of interest by drawing a rectangle around it. It outputs the category of image present in selected area.
2. Search About Image: This function can be used by clicking on the corresponding button and selecting out the area of interest by drawing a rectangle around it. It opens browser with search results about the selected area.
3. Copy Text to Clipboard: This function can be used by clicking on the corresponding button and selecting out the area of interest by drawing a rectangle around it. It processes text present in selected area and copies it to inbuilt clipboard of the operating system, which can be pasted in any text field of interest.

After any of the function is initiated, the user has to crop his/her area of interest. 'Identify Image' and 'Search About Image' functions are processed by the CNN Model and 'Copy Text to Clipboard' function is processed by Tesseract Engine.

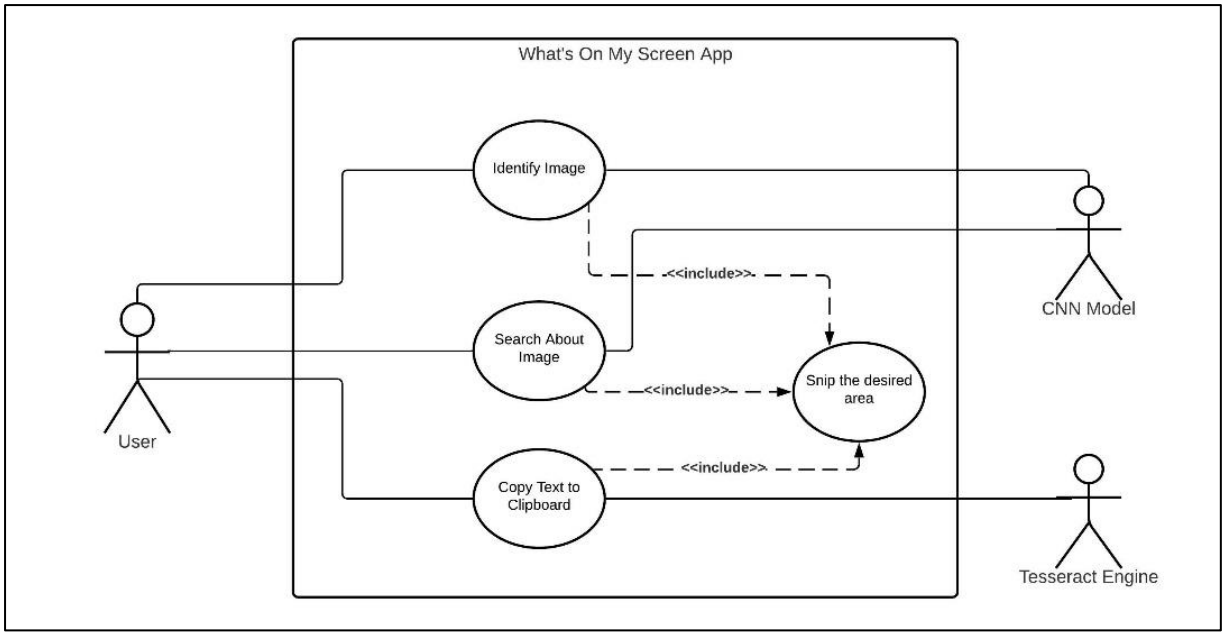

Fig. 3.1 App Use Case Diagram

3.1.2 Web-Based GUI

The User can perform two main operations using this system. They are as follows:

1. Data Visualization: It includes visualizations of various graphs. The user must select the graph he/she wishes to visualize. Then, the Streamlit Backend handles this request and displays corresponding graphs.
2. Model Testing: It takes an image as input and predicts its category. The user must upload an image from computer. Then it gets processed by the CNN Model and prediction gets displayed.

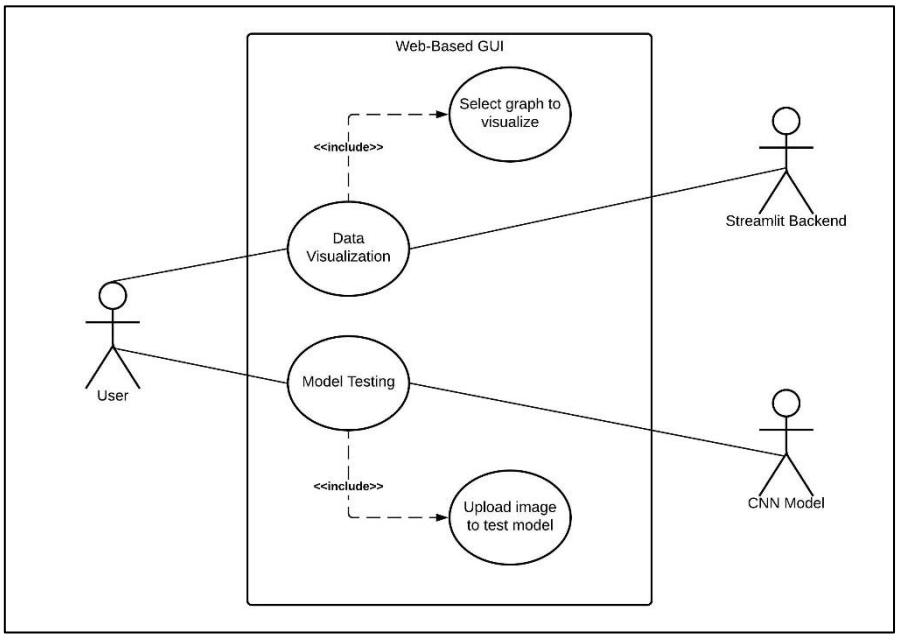

Fig. 3.2 Web-Based GUI Use Case Diagram

3.2 Class Diagrams

3.2.1 Menu Class

The Menu describes the data members and functions used for working of the main menu of the app. The Menu Class inherits form QMainWindow Class of PyQt5 package. The data members are: 'IdentifyImage', 'SearchAboutImage', 'CopyTextToClipboard', all of which have corresponding data type to the classes responsible for those functions. Similarly, for the basic functions of the app, some more functions have been designed in the class as shown in the figure. There are not inherited classes related to the Menu Class.

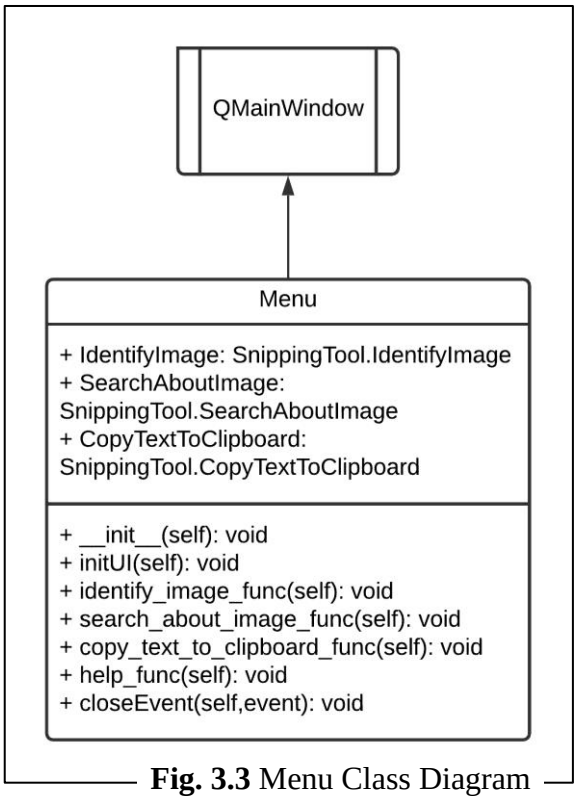

Fig. 3.3 Menu Class Diagram

3.2.2 SnippingWidget Class

The SnippingWidget class describes the data members and the functions used for the working of selection of desired area when the user initiates any of the three main functions. It is inherited from QtWidgets.QWidget class of PyQt5 package. The data members and member functions are as shown in the figure.

From SnippingWidget class, three other classes inherit namely, "IdentifyImage", "SearchAboutImage" and "CopyTextToClipboard". These classes don't have their own data members but have their own functions. These three classes are responsible for the three main functions of the application.

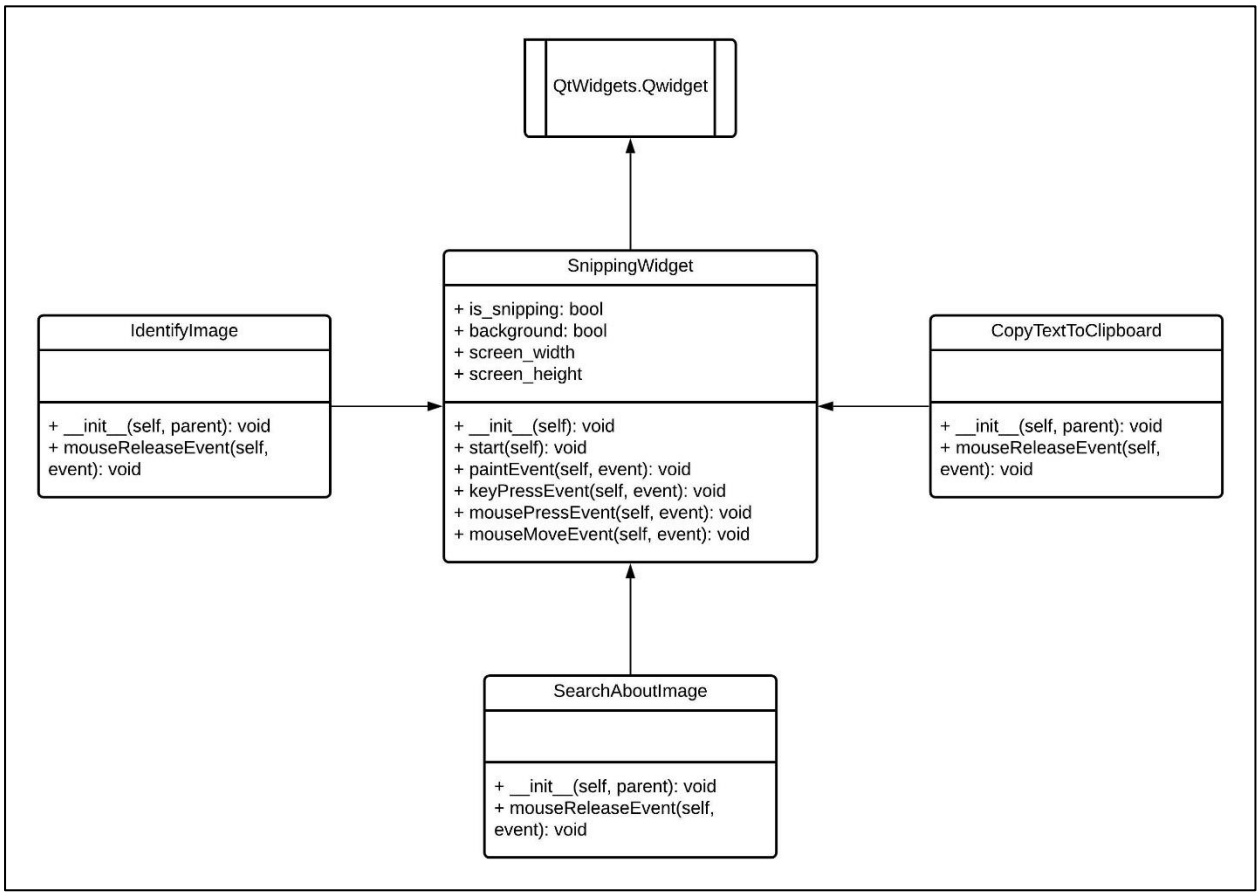

Fig. 3.4 SnippingWidget Class Diagram

4. IMPLEMENTATION AND CODING

4.1 Methodology

The idea behind this project is to study the CNN Model thoroughly and apply it in a way that ordinary people can make use of it. For the purpose of studying and analysis, the CNN Model has been implemented using web-based UI. And for the purpose of usability, it has been implemented as a standalone application called “What’s On My Screen?”. In both implementations, the model predicts the category of image that is given as input, but the use cases for both implementations are very different.

4.1.1 Web Based Implementation

The web-based GUI contains two main sections. Following is a brief description about both of them:

- **Data Visualization:** In this section, the use can find various illustrations about the model. It contains following graphs:
    - Dataset Breakdown
    - Model Training Stats
    - Model Testing Stats

  The user can use these graphs to analyze the patterns of training and testing. The illustrations are visually appealing and easy to read. They prove very insightful to even ordinary people.

- **Model Testing:** In this section, the user can upload any image of his/her choice. After uploading, the image gets processed by the CNN Model and then the prediction of category of the image is displayed.

4.1.2 Standalone Application

The standalone application "What's On My Screen?" provides three main features. Following is a brief description of all of them:

1. **Identify Image:** When the user clicks on the "Identify Image" button, a transparent mask covers whole screen and the application awaits the selection of area by the user. The user has to draw a rectangle over the area of interest by holding left mouse button. Then, after the mouse button is released, the selected area will be saved in memory as an image and then will be processed by the CNN Model. A small window pops up to show result, that is the prediction of category of image from selected area.

   The purpose of this operation is to give the user an idea about an object that he/she is not aware of. If the model is trained using images of unique objects, foods, etc. this feature can serve as an educational tool for the users by helping them gain insight about the things that they may have not seen in their life.

2. **Search About Image:** When the user clicks on the "Search About Image" button, a transparent mask covers whole screen and the application awaits the selection of area by the user. The user has to draw a rectangle over the area of interest by holding left mouse button. Then, after the mouse button is released, the selected area will be saved in memory as an image and then will be processed by the CNN Model. After the processing, the prediction for the image of selected area will be used by the application to return Google Search results about it. A browser window will pop up with the results.

   The purpose of this operation is to give the user more information about an object. As this feature displays Google Search results, this feature can be very helpful if the object is not known by the user. The model can be further trained with lot more categories of images to increase its scope of operation.

3. **Copy Text To Clipboard:** When the user clicks on the “Identify Image” button, a transparent mask covers whole screen and the application awaits the selection of area by the user. The user has to draw a rectangle over the area of interest by holding left mouse button. Then, after the mouse button is released, the selected area will be saved in memory as an image and then will be processed by the Tesseract Engine. The text appearing in the selected area will be converted into a string and then will get copied added to the clipboard of operating system. Then the user might paste it in desired text field by performing appropriate paste operation according to the operating system.

The purpose of this operation is to let the user copy some text from an image. There can be situations where the user might want to copy whatever is written in an image. But without a specialized tool the he/she may be forced to type all that text manually. In saving time and effort, this feature plays crucial role. It takes just a few seconds to copy all the text from an image using this feature.

4.2 Dataset

The dataset for this project was self-curated by procuring images from multiple sources. It contains 74,204 images in total among which 55745 were used for training and 18459 were used for validation purpose. All the images belonged to 10 different categories. Following is a breakdown of category wise number of images:

1. **Training Set**

Table 4.1 Training Dataset Breakdown

Category	**No. of Samples**
Building	2191
Car	8144
Dog	8749
Flower	8149
Forest	2271
People	9315
Pizza	4200
Sea	2274
Ship	6282
Traffic Sign	4170

2. **Validation Set**

Table 4.2 Validation Dataset Breakdown

Category	**No. of Samples**
Building	437
Car	3400
Dog	3751
Flower	1360
Forest	474
People	3083
Pizza	800
Sea	510
Ship	2650
Traffic Sign	1994

4.3 Algorithms and Flowchart

4.3.1 Algorithm for Training Model from scratch

1. Import required libraries and modules.

2. Define variables if needed.

   ```
   img_width, img_height= 150, 150
   train_data_dir="Dataset/train"
   validation_data_dir="Dataset/validate"
   train_samples = 55745
   validation_samples = 18459
   batch_size = 64
   ```

3. Initiate ImageDataGenerator class for transformation of training and validation images.

   ```
   train_datagen = ImageDataGenerator(
      rescale=1./255,
      shear_range = 0.2,
      zoom_range=0.2,
      horizontal_flip =True)

   val_datagen = ImageDataGenerator(rescale=1./255)
   ```

4. Build image data generator using flow_from_directory() function.

   ```
   train_generator = train_datagen.flow_from_directory(
      train_data_dir,
      target_size=(img_width,img_height),
      batch_size = batch_size,
      class_mode='categorical')

   val_generator = val_datagen.flow_from_directory(
      validation_data_dir,
   ```

```
    target_size=(img_width, img_height),
    batch_size = batch_size,
    class_mode='categorical')
```

5. Build a sequential model by adding appropriate layers with appropriate attributes.

```
model = Sequential()
model.add(Layer 1)
model.add(Layer 2)
model.add(Layer 3)
```

6. Compile the model with compile() function and mention desired loss function and optimizer.

```
model.compile(loss='categorical_crossentropy',
        optimizer='rmsprop',
        metrics=['accuracy'])
```

7. Fit the model for appropriate number of epochs (16 in our case) using fit() function.

```
model.fit(train_generator,
        steps_per_epoch=train_samples//batch_size,
        epochs=15,
        validation_data=val_generator,
        validation_steps=validation_samples//batch_size)
```

8. Save the model.

```
model.save("./My_Model", save_format = 'tf')
```

4.3.2 Algorithm for Training Model with Transfer Learning

1. Import required libraries and modules.

2. Define variables if needed.

```
img_width, img_height= 150, 150
train_data_dir="Dataset/train"
validation_data_dir="Dataset/validate"
train_samples = 55745
validation_samples = 18459
batch_size = 64
```

3. Initiate ImageDataGenerator class for transformation of training and validation images.

```
train_datagen = ImageDataGenerator(
   rescale=1./255,
   shear_range = 0.2,
   zoom_range=0.2,
   horizontal_flip =True)

val_datagen = ImageDataGenerator(rescale=1./255)
```

4. Build image data generator using flow_from_directory() function.

```
train_generator = train_datagen.flow_from_directory(
   train_data_dir,
   target_size=(img_width,img_height),
   batch_size = batch_size,
   class_mode='categorical')

val_generator = val_datagen.flow_from_directory(
   validation_data_dir,
   target_size=(img_width, img_height),
```

```
    batch_size = batch_size,
    class_mode='categorical')
```

5. Initiate VGG19 Model without the top layers.

```
pre_trained_model = VGG19(input_shape = (img_width,img_height,3),
                include_top = False,
                weights='imagenet',
                classes = 10)
```

6. Freeze the convolutional layers of the model.

```
for layer in pre_trained_model.layers:
  layer.trainable = False
```

7. Build a Sequential model using the pre trained model and add new layers if needed.

```
model = Sequential()
model.add(pre_trained_model)
model.add(Layer 1)
model.add(Layer 2)
model.add(Layer 3)
```

8. Compile the model with compile() function and mention desired loss function and optimizer.

```
model.compile(optimizer=RMSprop(lr=0.001),
          loss='categorical_crossentropy',
          metrics=['accuracy'])
```

9. Fit the model for appropriate number of epochs (20 in our case) using fit() function.

```
model.fit(train_generator,
        steps_per_epoch=train_samples//batch_size,
        epochs=20,
        validation_data=val_generator,
        validation_steps=validation_samples//batch_size)
```

10. Save the model.

```
model.save("./VGG19", save_format = 'tf')
```

4.3.3 Algorithm for Implementation in Standalone Application

A) Snipping Action:

1. Import required libraries and modules.

2. Load the saved CNN model.

   model = models.load_model(*Location of model*)

3. Inherit SnippingWidget class from QtWidgets.QWidget.

   class SnippingWidget(QtWidgets.QWidget()

4. Add __init__() method as constructor with appropriate data members to SnippingWidget.

   def __init__(self, parent=None):

5. Add start() function to SnippingWidget class for initializing the snipping window when it is called.

```
def start(self):
    SnippingWidget.background = False
    SnippingWidget.is_snipping = True
    self.setWindowOpacity(0.3)
    self.show()
```

6. Add paintEvent() function to SnippingWidget class with appropriate steps for masking whole screen with a transparent layer.

```
def paintEvent(self, event):
    if SnippingWidget.is_snipping:
        brush_color = (128, 128, 255, 100)
        lw = 3
        opacity = 0.3
    else:
        # reset points, so the rectangle won't show up again.
```

```
        self.begin = QtCore.QPoint()
        self.end = QtCore.QPoint()
        brush_color = (0, 0, 0, 0)
        lw = 0
        opacity = 0
```

7. Add keyPressEvent() method to SnippingWidget to abort when Escape key is pressed.

```
    def keyPressEvent(self, event):
        if event.key() == QtCore.Qt.Key_Escape:
            print('Quit')
            self.close()
        event.accept()
```

8. Add mousePressEvent() method to SnippingWidget to trigger snipping action.

```
    def mousePressEvent(self, event):
        self.begin = event.pos()
        self.end = self.begin
        self.update()
```

9. Add mouseMoveEvent() method to SnippingWidget to capture coordinates of cursor whenever it is moved.

```
    def mouseMoveEvent(self, event):
        self.end = event.pos()
        self.update()
```

10. Inherit IdentifyImage class from SnippingWidget.

```
    class IdentifyImage(SnippingWidget):
```

11. Add mouseReleaseEvent() method to IdentifyImage to trigger the capturing of image, processing and displaying result.

```
     def mouseReleaseEvent(self, event):
        self.close()
```

```
SnippingWidget.is_snipping = False
QtWidgets.QApplication.restoreOverrideCursor()
x1 = min(self.begin.x(), self.end.x())
y1 = min(self.begin.y(), self.end.y())
x2 = max(self.begin.x(), self.end.x())
y2 = max(self.begin.y(), self.end.y())

if x1==x2 or y1==y2:
    print("Error! Snipped image has zero pixels. Try Again.")
    msg = QtWidgets.QMessageBox()
    msg.setWindowTitle('ERROR!')
    msg.setText("Snipped image has zero pixels. Try Again.")
    msg.setIcon(QtWidgets.QMessageBox.Warning)
    x = msg.exec_()
else:
    img = ImageGrab.grab(bbox=(x1, y1, x2, y2))
    img = cv2.cvtColor(np.array(img), cv2.COLOR_BGR2RGB)

    cv2.imshow('Captured Image', img)
    cv2.waitKey(1)
    cv2.destroyAllWindows()

    img = image.resize(img,[150,150])
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)

    prediction = model.predict(img)
    print(class_dic[np.argmax(prediction)])

    msg = QtWidgets.QMessageBox()
    msg.setStyleSheet(dictToCSS(popup_CSS))
    msg.setWindowTitle('RESULT:')
    msg.setFont(QFont('Cambria', 14))
```

```
            msg.setText(class_dic[np.argmax(prediction)])
            msg.setIcon(QtWidgets.QMessageBox.Information)

            x = msg.exec_()

        self.begin = QtCore.QPoint()
        self.end = QtCore.QPoint()
        brush_color = (0, 0, 0, 0)
        lw = 0
        opacity = 0
```

12. Inherit SearchAboutImage class from SnippingWidget

```
class SearchAboutImage(SnippingWidget):
```

13. Add mouseReleaseEvent() method to SearchAboutImage to trigger the capturing of image, its processing and opening the browser window.

```
def mouseReleaseEvent(self, event):
    self.close()
    SnippingWidget.is_snipping = False
    QtWidgets.QApplication.restoreOverrideCursor()
    x1 = min(self.begin.x(), self.end.x())
    y1 = min(self.begin.y(), self.end.y())
    x2 = max(self.begin.x(), self.end.x())
    y2 = max(self.begin.y(), self.end.y())

    if x1==x2 or y1==y2:
        print("Error! Snipped image has zero pixels. Try Again.")
        msg = QtWidgets.QMessageBox()
        msg.setWindowTitle('ERROR!')
        msg.setText("Snipped image has zero pixels. Try Again.")
        msg.setIcon(QtWidgets.QMessageBox.Warning)
        x = msg.exec_()
    else:
```

```
            img = ImageGrab.grab(bbox=(x1, y1, x2, y2))
            img = cv2.cvtColor(np.array(img), cv2.COLOR_BGR2RGB)

            cv2.imshow('Captured Image', img)
            cv2.waitKey(1)
            cv2.destroyAllWindows()

            img = image.resize(img,[150,150])
            img = img_to_array(img)
            img = np.expand_dims(img, axis=0)

            prediction = model.predict(img)
            search_key = class_dic[np.argmax(prediction)]

            search_key = search_key.replace(' ', '+')
            browser = webdriver.Chrome('C:/Users/gutte/Desktop/College
Projects/SDL Projects/Mini Project/chromedriver.exe')
            browser.get("https://www.google.com/search?q=" + search_key +
"&start=1")

        self.begin = QtCore.QPoint()
        self.end = QtCore.QPoint()
        brush_color = (0, 0, 0, 0)
        lw = 0
        opacity = 0
```

14. Inherit CopyTextToClipboard class from SnippingWidget.

```
    class CopyTextToClipboard(SnippingWidget):
```

15. Add mouseReleaseEvent() method to CopyTextToClipboard to trigger capturing of image, extraction of text from it and adding text string to local clipboard.

```
    def mouseReleaseEvent(self, event):
        self.close()
```

```
            SnippingWidget.is_snipping = False
            QtWidgets.QApplication.restoreOverrideCursor()
            x1 = min(self.begin.x(), self.end.x())
            y1 = min(self.begin.y(), self.end.y())
            x2 = max(self.begin.x(), self.end.x())
            y2 = max(self.begin.y(), self.end.y())

            if x1==x2 or y1==y2:
                print("Error! Snipped image has zero pixels. Try Again.")
                msg = QtWidgets.QMessageBox()
                msg.setWindowTitle('ERROR!')
                msg.setText("Snipped image has zero pixels. Try Again.")
                msg.setIcon(QtWidgets.QMessageBox.Warning)
                x = msg.exec_()
            else:
                img = ImageGrab.grab(bbox=(x1, y1, x2, y2))
                img = cv2.cvtColor(np.array(img), cv2.COLOR_BGR2RGB)

                cv2.imshow('Captured Image', img)
                cv2.waitKey(5)
                cv2.destroyAllWindows()

                # gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
                # gray, img_bin = cv2.threshold(gray,128,255,cv2.THRESH_BINARY
| cv2.THRESH_OTSU)
                # gray = cv2.bitwise_not(img_bin)

                # kernel = np.ones((2, 1), np.uint8)
                # img = cv2.erode(gray, kernel, iterations=1)
                # img = cv2.dilate(img, kernel, iterations=1)
                text = pytesseract.image_to_string(img)

                pyperclip.copy(text)
```

```
msg = QtWidgets.QMessageBox()
msg.setStyleSheet(dictToCSS(popup_CSS))
msg.setWindowTitle('RESULT:')
msg.setText("Text Copied!")
msg.setIcon(QtWidgets.QMessageBox.Information)

x = msg.exec_()
```

```
self.begin = QtCore.QPoint()
self.end = QtCore.QPoint()
brush_color = (0, 0, 0, 0)
lw = 0
opacity = 0
```

B) Menu Window:

1. Import required libraries and modules along with SnippingTool.

2. Inherit Menu class from QMainWindow.

```
class Menu(QMainWindow):
```

3. Define __init__() method as a constructor to Menu class along with required data members.

```
def __init__(self):
    super(Menu, self).__init__()
    self.setStyleSheet(dictToCSS(CSS))
    self.IdentifyImage = SnippingTool.IdentifyImage()
    self.SearchAboutImage = SnippingTool.SearchAboutImage()
    self.CopyTextToClipboard = SnippingTool.CopyTextToClipboard()
    self.initUI()
```

4. Define initUI() function in Menu to initiate the menu window whenever it is called, along with the GUI elements.

```
def initUI(self):

    self.setObjectName("Image Analyser")
    self.setWindowTitle("What's On My Screen?")
    self.setWindowFlag(Qt.FramelessWindowHint)
    self.setEnabled(True)
    self.resize(220, 140)
    self.setGeometry(1680,880,220,140)
    self.setMouseTracking(True)
    self.setFocusPolicy(QtCore.Qt.NoFocus)

    self.centralwidget = QtWidgets.QWidget(self)
    self.centralwidget.setObjectName("centralwidget")

    self.label = QtWidgets.QLabel(self)
    self.label.setGeometry(QtCore.QRect(10,0,150,30))
    self.label.setText("What's On My Screen?")
    self.label.setFont(QFont('Arial',8))

    self.logo = QtWidgets.QLabel(self)
    self.logo.setPixmap(QPixmap(QIcon('C:/Users/gutte/Desktop/College
Projects/SDL Projects/Mini Project/logo.png').pixmap(QSize(60,60))))
    self.logo.setGeometry(QtCore.QRect(160,100,60,60))

    self.close_button = QtWidgets.QPushButton(self)
    self.close_button.setGeometry(QtCore.QRect(200,0,20,20))
    self.close_button.setObjectName("close_button")
    self.close_button.setIcon(QIcon('C:/Users/gutte/Desktop/College
Projects/SDL Projects/Mini Project/close_icon.png'))
    self.close_button.setIconSize(QSize(20,20))
    self.close_button.setToolTip("Close")
```

```
        self.close_button.clicked.connect(self.closeEvent)

        self.minimize_button = QtWidgets.QPushButton(self)
        self.minimize_button.setGeometry(QtCore.QRect(180,0,20,20))
        self.minimize_button.setObjectName("minimize_button")
        self.minimize_button.setIcon(QIcon('C:/Users/gutte/Desktop/College
Projects/SDL Projects/Mini Project/minimize_icon.png'))
        self.minimize_button.setIconSize(QSize(20,20))
        self.minimize_button.setToolTip("Minimize")
        self.minimize_button.clicked.connect(lambda: self.showMinimized())

        self.identify_image = QtWidgets.QPushButton(self)
        self.identify_image.setGeometry(QtCore.QRect(10, 40, 60, 60))
        self.identify_image.setObjectName("identify_image")
        self.identify_image.setIcon(QIcon('C:/Users/gutte/Desktop/College
Projects/SDL Projects/Mini Project/identify_image_icon.png'))
        self.identify_image.setIconSize(QSize(55,55))
        self.identify_image.setToolTip("Identify Image")
        self.identify_image.clicked.connect(self.identify_image_func)

        self.search_about_image = QtWidgets.QPushButton(self)
        self.search_about_image.setGeometry(QtCore.QRect(80, 40, 60, 60 ))
        self.search_about_image.setObjectName("search_about_image")
        self.search_about_image.setIcon(QIcon('C:/Users/gutte/Desktop/College
Projects/SDL Projects/Mini Project/search_image_icon.png'))
        self.search_about_image.setIconSize(QSize(55,55))
        self.search_about_image.setToolTip("Search About Image")
        self.search_about_image.clicked.connect(self.search_about_image_func)

        self.copy_text_to_clipboard = QtWidgets.QPushButton(self)
        self.copy_text_to_clipboard.setGeometry(QtCore.QRect(150, 40, 60, 60))
        self.copy_text_to_clipboard.setObjectName("copy_text_to_clipboard")
```

```
self.copy_text_to_clipboard.setIcon(QIcon('C:/Users/gutte/Desktop/College
Projects/SDL Projects/Mini Project/copy_text.png'))
        self.copy_text_to_clipboard.setIconSize(QSize(55,55))
        self.copy_text_to_clipboard.setToolTip("Copy Text To Clipboard")

self.copy_text_to_clipboard.clicked.connect(self.copy_text_to_clipboard_func
)



        self.help = QtWidgets.QPushButton(self)
        self.help.setGeometry(QtCore.QRect(0,120,20,20))
        self.help.setObjectName("help")
        self.help.setIcon(self.style().standardIcon(getattr(QStyle,
'SP_MessageBoxInformation')))
        self.help.setIconSize(QSize(20,20))
        self.help.clicked.connect(self.help_func)



        self.show()
```

5. Define identify_image_func() function in Menu, which will be called when user clicks ‘Identify Image’ button.

```
def identify_image_func(self):
    self.IdentifyImage.start()
```

6. Define search_about_iamge_func() function in Menu, which will be called when user clicks ‘Search About Image’ button.

```
def search_about_image_func(self):
    self.SearchAboutImage.start()
```

7. Define copy_text_to_clipboard_func() function in Menu, which will be called when user clicks ‘Copy Text To Clipboard’ button.

```
def copy_text_to_clipboard_func(self):
    self.CopyTextToClipboard.start()
```

8. Define closeEvent() method in Menu, which will be called when user clicks on ‘Close’ button.

```
def closeEvent(self,event):
    self.close()
    sys.exit()
```

9. Initialize the Menu class.

```
ui = Menu()
```

4.3.4 Algorithm for Implementation through Web-Based GUI

A) Data Visualization Section:

1. Import all required modules and libraries.

2. Load required data from appropriate documents into Streamlit cache.

```
@st.cache()
def load_train_val_data():
   df = pd.read_csv("Training_Validation_Dataset.csv")
   return df

@st.cache()
def load_model_stats():
   df = pd.read_csv("Model_stats.csv", usecols=['Epoch', 'Val_Accuracy'])
   return df

@st.cache()
def load_vgg19_stats():
   df = pd.read_csv("VGG19_Stats.csv", usecols=['Epoch',
'Val_Accuracy'])
```

```
    return df


@st.cache()
def load_accuracy():
    df = pd.read_csv("Accuracy.csv")
    return df
```

3. Write functions for plotting of data for corresponding dataframes.

```
def visualize_Training_and_validation_data():
    st.header("Dataset Breakdown")
    df = load_train_val_data()
    layout = go.Layout(
                title= "<b>Dataset Breakdown</b>",
                xaxis= dict(title='Category',linecolor='white',),
                yaxis= dict(title='No. of Samples',linecolor='white'),
                template="plotly_dark")
    fig = go.Figure(data=[go.Bar(
        name = 'Training Set',
        x = df['Category'],
        y = df['No. of Training Samples'],),
        go.Bar(
            name='Validation Set',
            x = df['Category'],
            y = df['No. of Validation Samples'])],
            layout = layout)
    st.plotly_chart(fig)


def visualize_Model_stats():
    st.header("Model Training Stats")
    df = load_model_stats()
    layout = go.Layout(
                title= "<b>Original Model</b>",
                xaxis= dict(title='Epoch',linecolor='white'),
```

```
                yaxis= dict(title='% Validation Accuracy',linecolor='white'),
                template="plotly_dark",)
    fig = go.Figure(data=[go.Scatter(
        name = 'Training Set',
        x = df['Epoch'],
        y = df['Val_Accuracy'],
        mode='lines+markers',
        line=dict(color='#fc9003'))],
            layout = layout,
            )
    st.plotly_chart(fig)

def visualize_VGG_stats():
    st.header("Transfer Learning Stats")
    df = load_vgg19_stats()
    layout = go.Layout(
                title= "<b>Transfer Learning Model</b>",
                xaxis= dict(title='Epoch', linecolor='white'),
                yaxis= dict(title='% Validation Accuracy', linecolor='white'),
                template="plotly_dark")
    fig = go.Figure(data=[go.Scatter(
        name = 'Training Set',
        x = df['Epoch'],
        y = df['Val_Accuracy'],
        mode='lines+markers',
        line=dict(color='#fcba03'))],
            layout = layout)
    st.plotly_chart(fig)

def visualize_accuracy():
    st.header("Model Testing Stats")
    df = load_accuracy()
    layout = go.Layout(
```

```
                title= "<b>Model Testing Accuracy</b>",
                xaxis= dict(title='Model', linecolor='white'),
                yaxis=                                          dict(title='%
Accuracy',range=[1,100],linecolor='white'),
                template="plotly_dark")
    fig = go.Figure(data=[go.Bar(
        name = 'Testing Accuracy',
        x = ['Original Model', 'Transfer Learning (VGG19)'],
        y = df['Accuracy'],
        width= .3)],
            layout = layout)
    st.plotly_chart(fig)
```

4. Add a selectbox to main screen with names of all graphs.

```
    selectgraph = st.selectbox(
        " ",
        ("<NONE>","Dataset Breakdown", "Model Training Stats", "Model
    Testing Stats")
      )
```

5. Call corresponding functions whenever a graph is selected from selectbox.

```
    if selectgraph=="<NONE>":
        st.write("No Graph Selected")
    if selectgraph=="Dataset Breakdown":
        visualize_Training_and_validation_data()
  if selectgraph=="Model Training Stats":
        visualize_Model_stats()
        visualize_VGG_stats()
  if selectgraph == "Model Testing Stats":
        visualize_accuracy()
```

B) Model Testing Section:

1. Import all the required modules and libraries.

2. Load the trained model.

```
model = load_model("./VGG19_tf")
```

3. Write function to import an image and return a prediction.

```
def import_and_predict(image_data, model):
    size = (150,150)
    image = ImageOps.fit(image_data, size, Image.ANTIALIAS)
    image = np.asarray(image)
    img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    img_resize = (cv2.resize(img, dsize=(150, 150), interpolation=cv2.INTER_CUBIC))/255
    img_reshape = img_resize[np.newaxis,...]
    prediction = model.predict(img_reshape)
    return prediction
```

4. Add a facility for user to upload an image.

```
file = st.file_uploader("Please Upload the Image", type=["jpg", "png"])
```

5. Display prediction about an image whenever the user uploads one.

```
image = Image.open(file)
prediction = import_and_predict(image, model)
st.subheader("Prediction: "+ class_dic[np.argmax(prediction)])
```

4.3.5 Flowchart for Standalone App

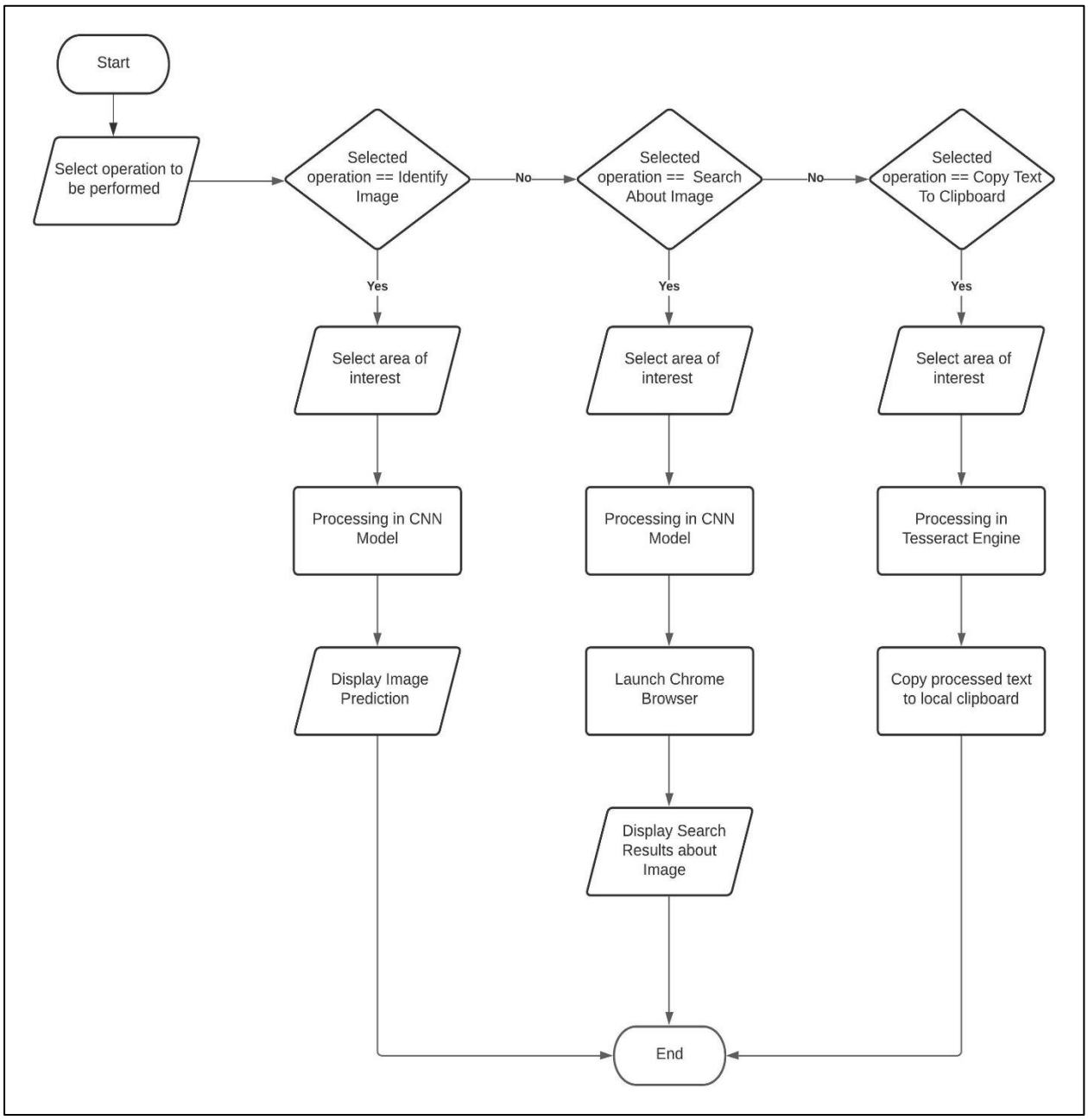

Fig. 4.1 Standalone App Flowchart

4.3.6 Flowchart for Web-Based GUI

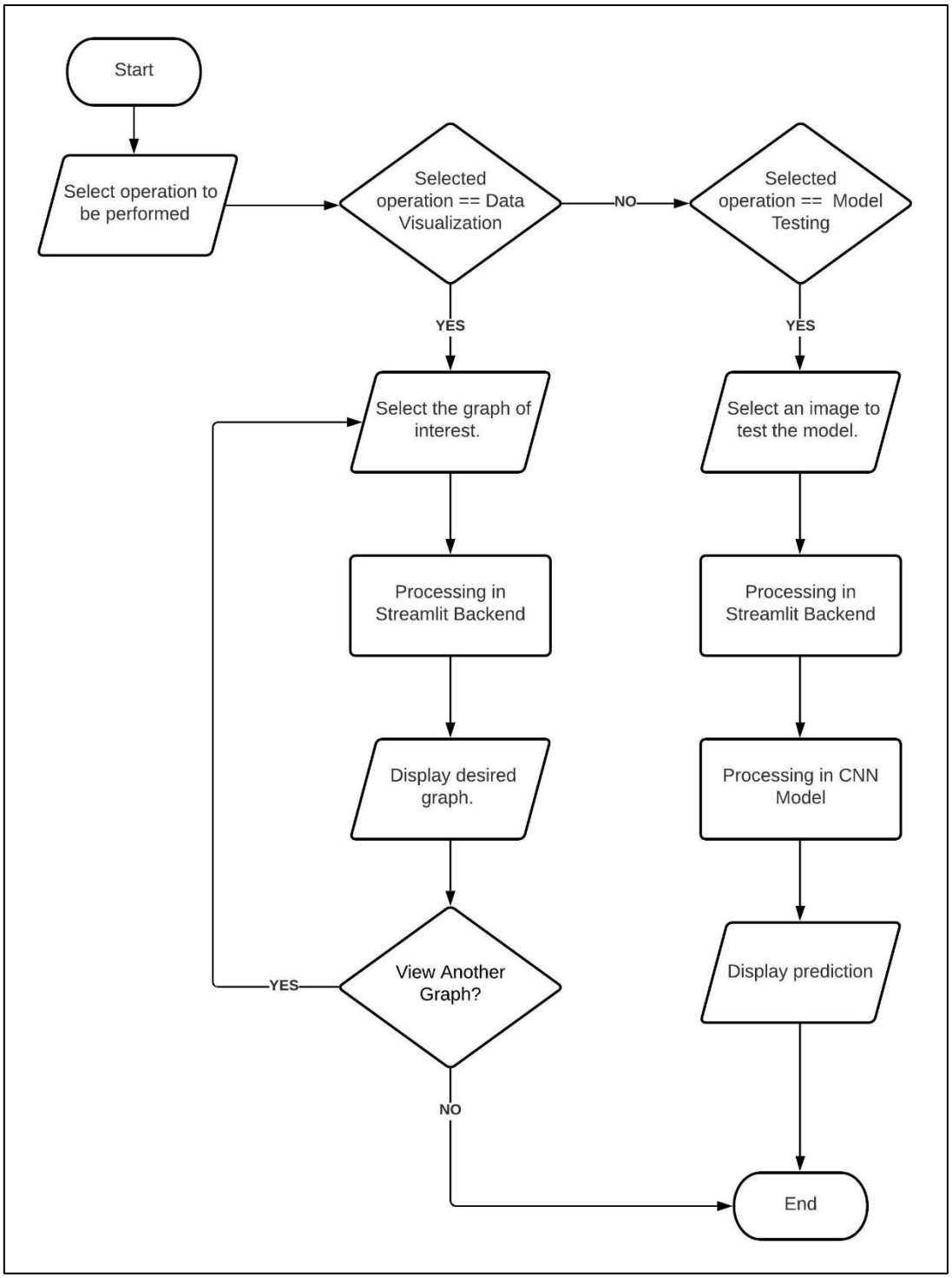

Fig. 4.2 Web-Based GUI Flowchart

4.4 GUI Design and Screenshots

4.4.1 Standalone App

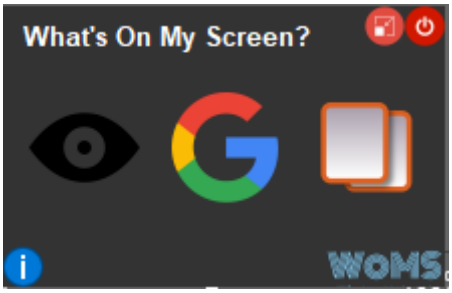

Fig. 4.3 Main Menu

Fig. 4.4 Identify Image Button

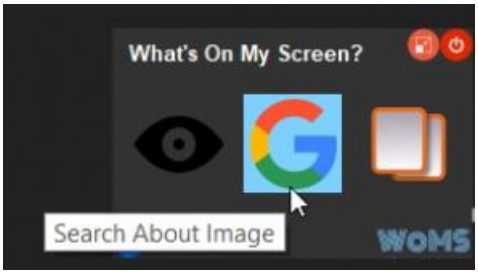

Fig. 4.5 Search About Image Button

Fig. 4.6 Copy Text To Clipboard Button

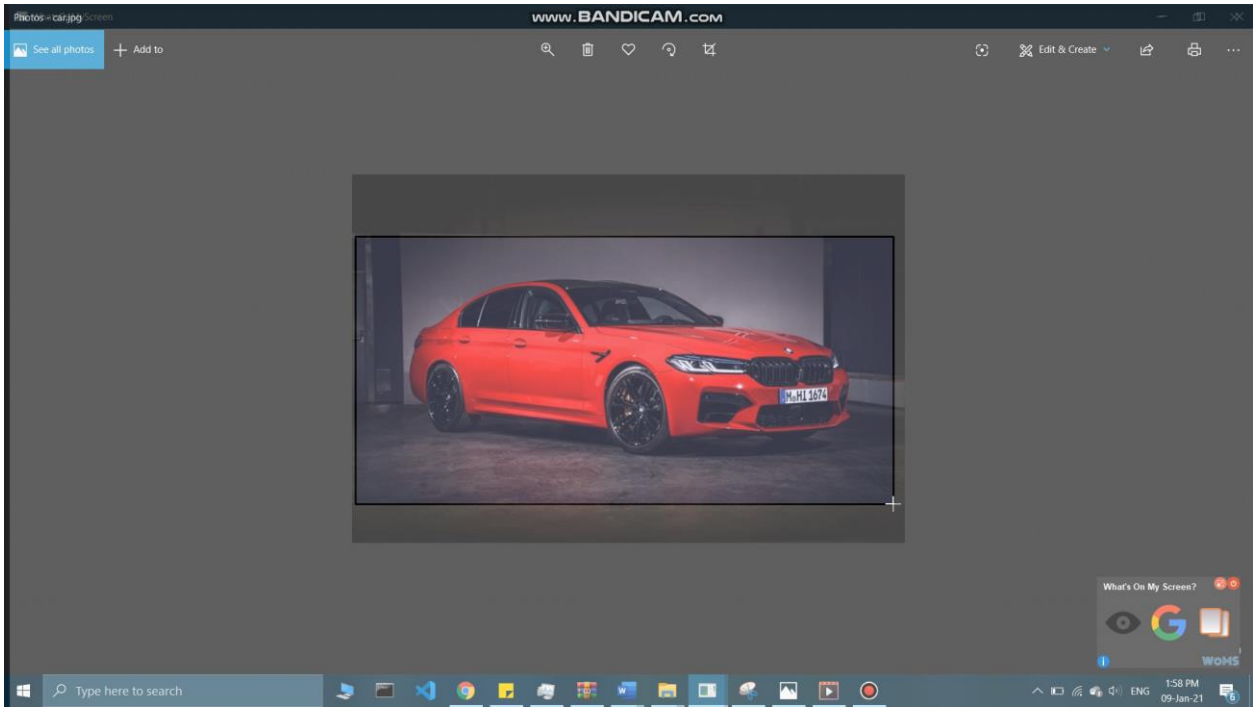

Fig. 4.7 Selection of Desired Area

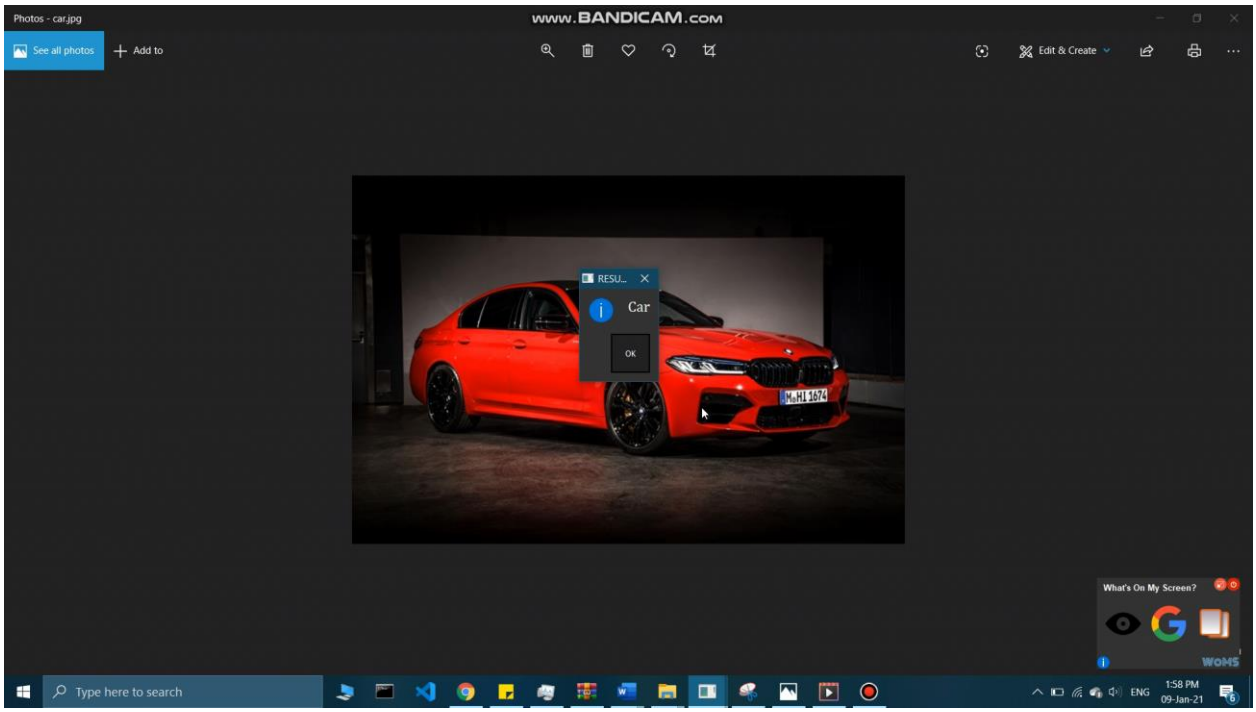

Fig. 4.8 Result

4.4.2 Web-Based UI

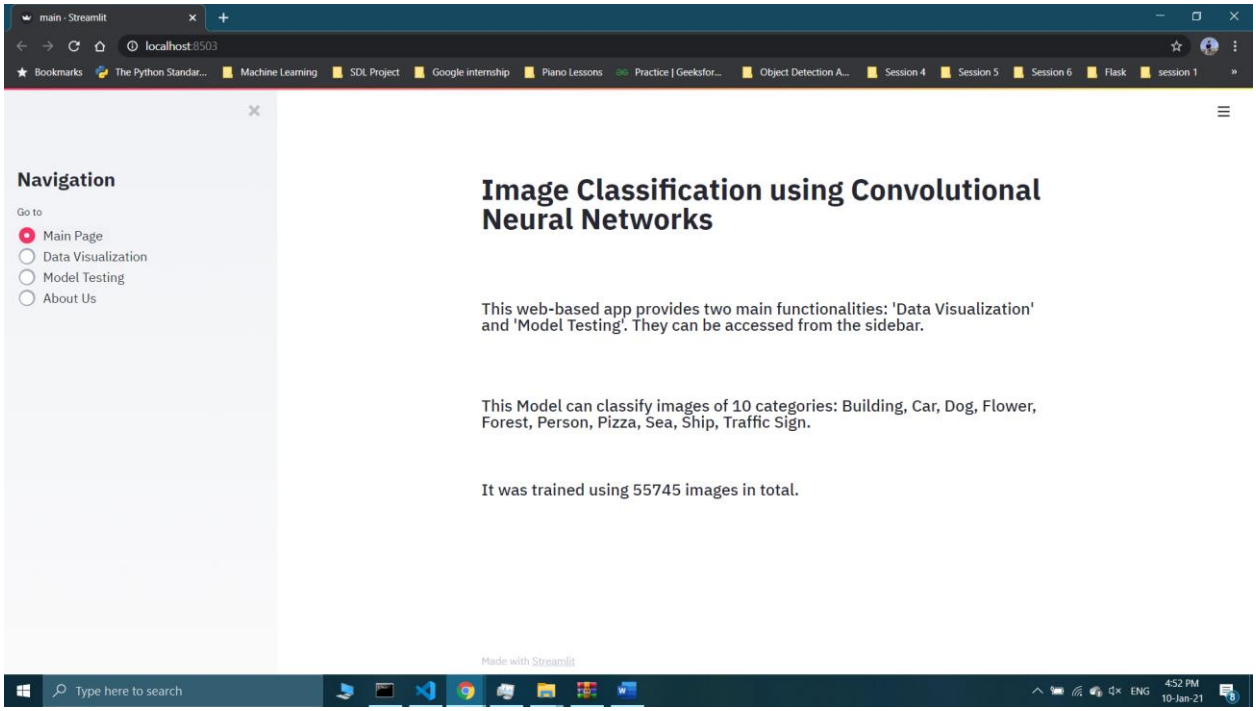

Fig. 4.9 Main Page

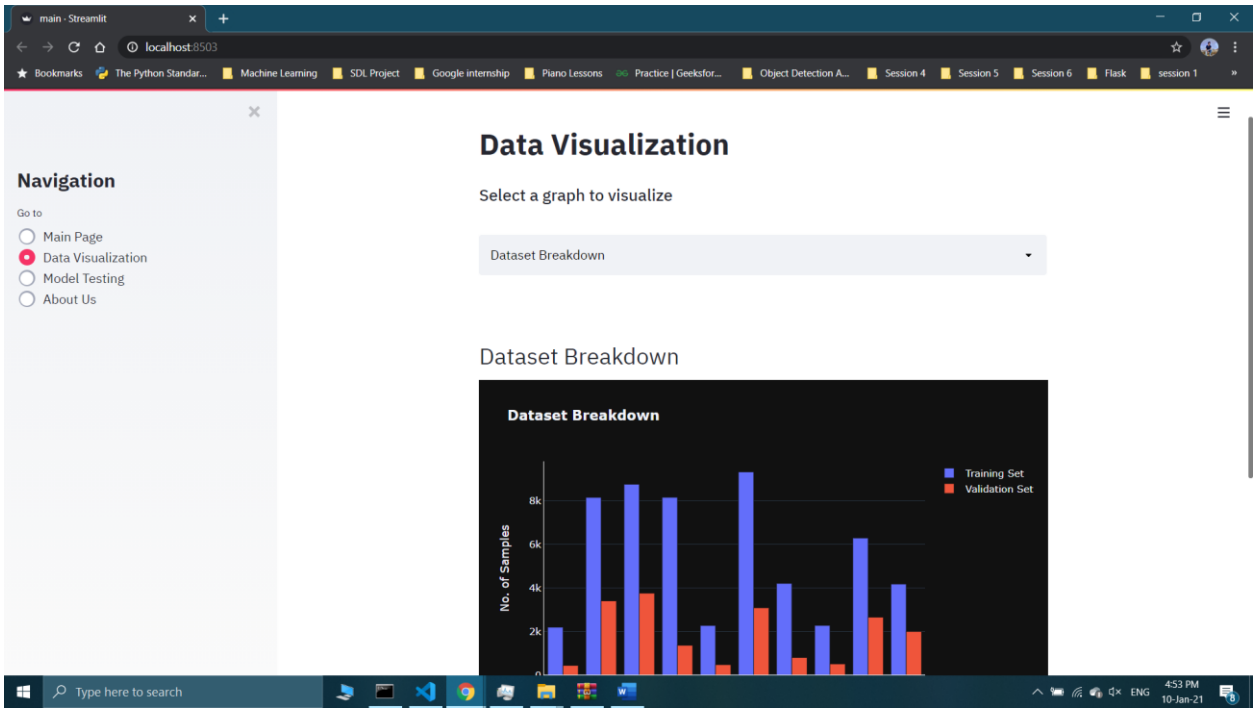

Fig. 4.10 Data Visualization Section

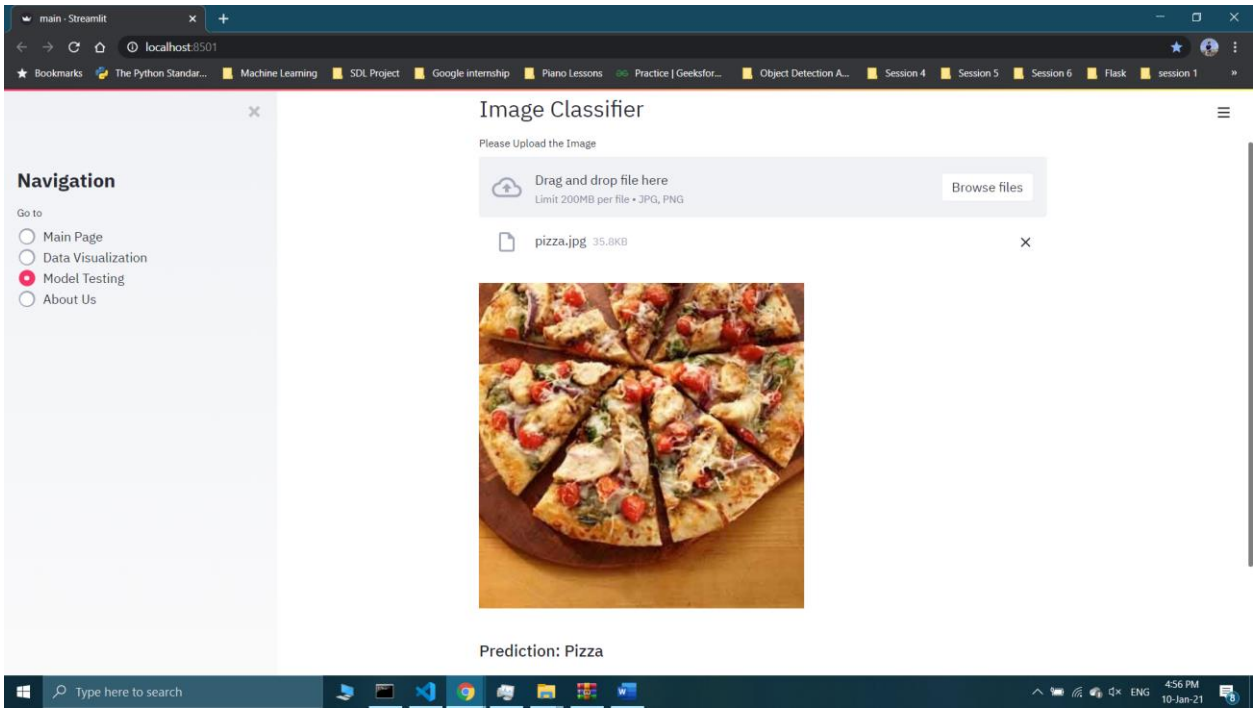

Fig. 4.11 Model Testing Section

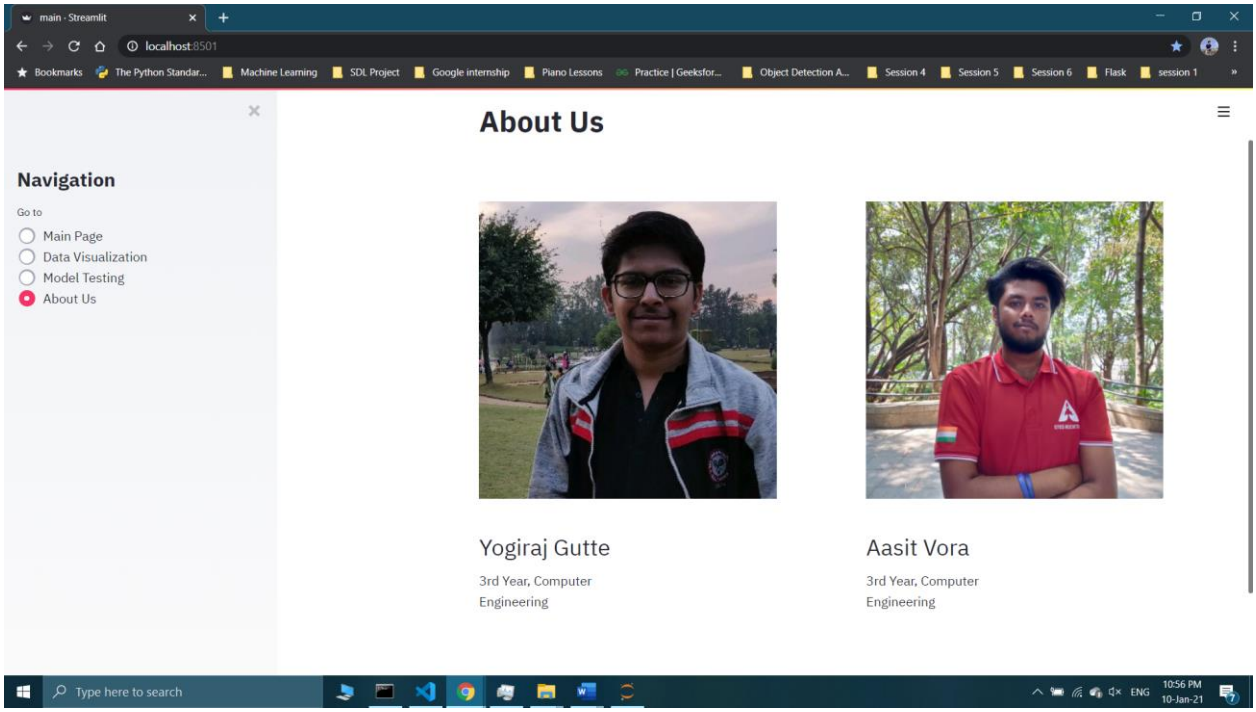

Fig. 4.12 About Us Section

5. RESULTS AND DISCUSSION

5.1 Visualization of Results

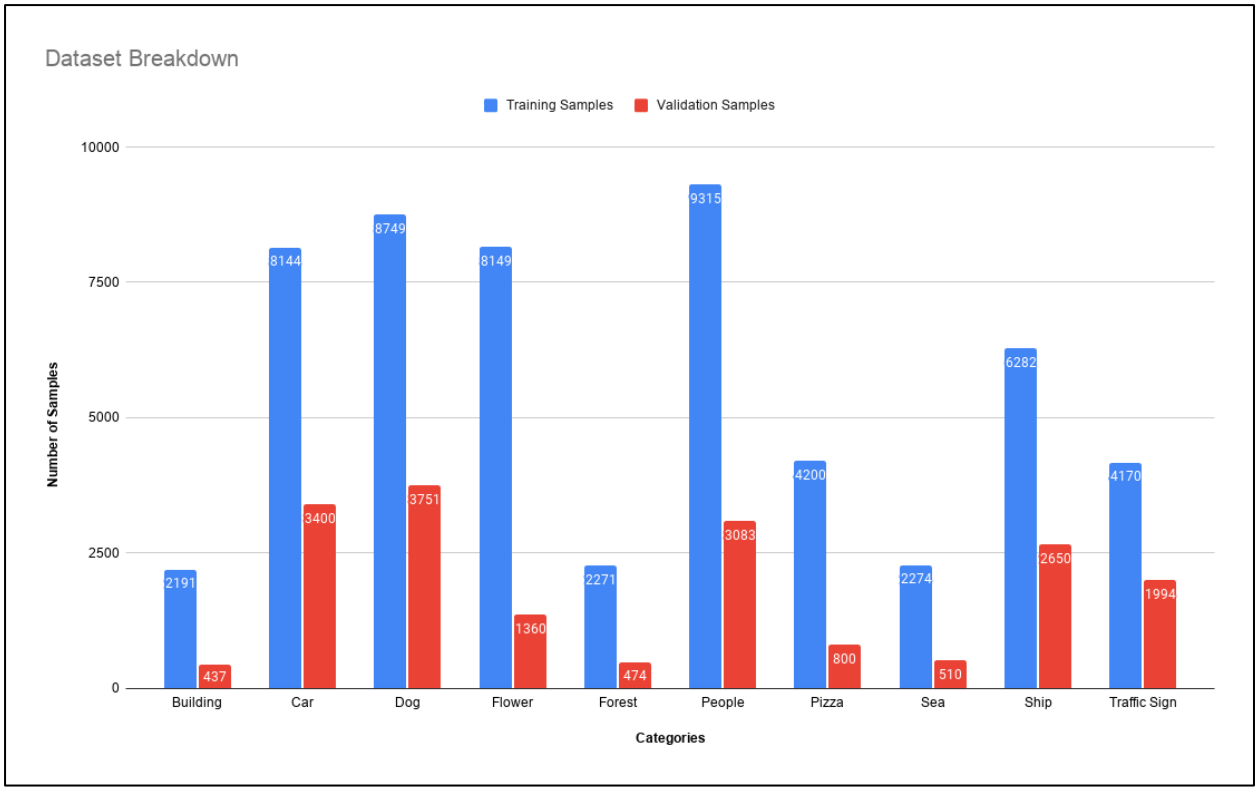

Fig. 5.1 Dataset Breakdown

Above illustration depicts the information about number of samples of each category used for training and validation of the model.

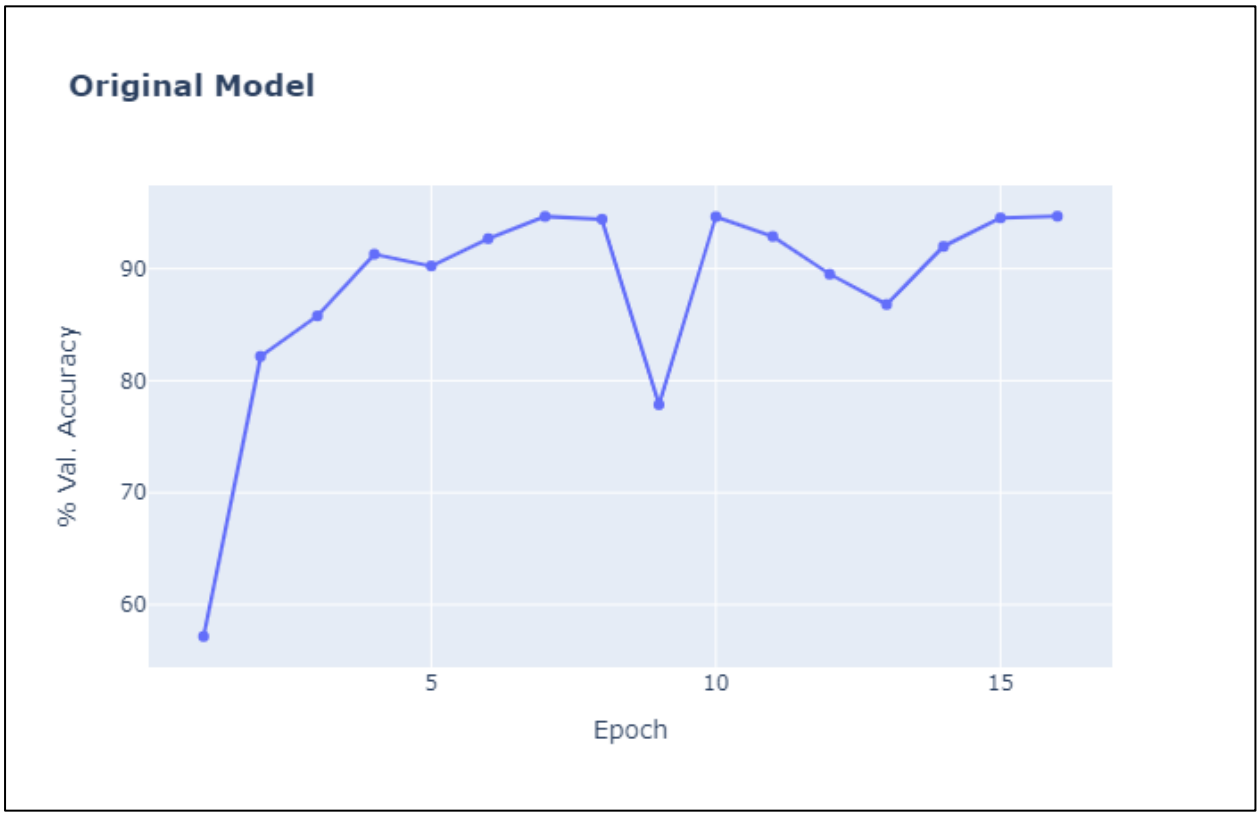

Fig. 5.2 Original Model Training Stats

Above illustration depicts epoch-wise validation accuracy of the original model. It can be observed that the validation accuracy increases with number of epochs. There can be some drops because of the nature of training. If a drop is observed for many consecutive epochs, the model is said to be overfitting and the training must stop there.

Fig. 5.3 Transfer Learning Stats

Above illustration depicts epoch-wise validation accuracy of the transfer learning model based on VGG19 model. It can be observed that the validation accuracy increases with number of epochs. There can be some drops because of the nature of training. If a drop is observed for many consecutive epochs, the model is said to be overfitting and the training must stop there. Initial accuracy of transfer learning model is better than that of the original model. This is because the feature extracting layers are pre-trained.

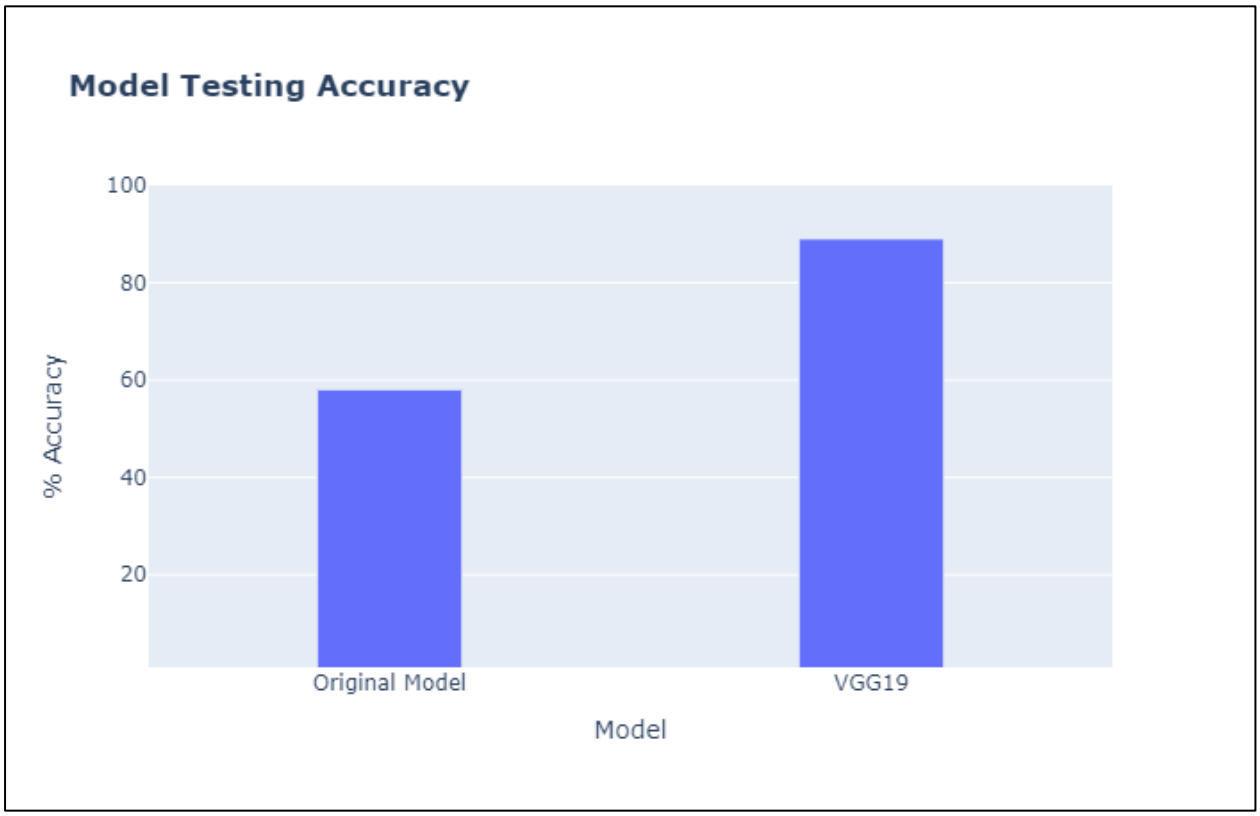

Fig. 5.4 Testing Accuracy

Both models were tested on a test set containing 20 images of each class. The original model gave 58% correct predictions while the transfer learning model gave 89% correct predictions. It implies that the transfer learning based models are better than those built from scratch. This is because the feature extractor layers of transfer learning models are generally pretrained on a very large dataset.

5.2 Discussion

The purpose of this project was to create an image classification model based on Convolutional Neural Network and to implement it using different methods. While building the model, we have used two different methods: building whole model from scratch and building using transfer learning. The testing accuracy graph (**Fig. 5.4)** shows that the model based on transfer learning is way more accurate than the one build from scratch. Because of this reason, today most of the CNN models are based on transfer learning. We have used VGG19 as our base model in transfer learning.

The model built from scratch could give better testing accuracy if it were trained on a larger dataset and for a greater number of epochs. But because of computational limitations and time limit, our team decided to build another model using transfer learning.

Even though the accuracy improved with the use of VGG19 model as feature extractor, the model faces difficulty in identifying some images. There are many drawbacks of Convolutional Neural Network models. Following are the major drawbacks of CNN:

1. Translational Invariance: A same object with slight change of orientation may not fire up the neuron that is supposed to recognize that object. So, for many objects, a CNN model would not give correct prediction if they are differently oriented. Data augmentation could solve this problem but can't get complete rid of it [4].

2. Pooling Layers: Pooling layers lose a lot of valuable information and it ignores the relation between the part and the whole. For example, if we talk about a face detector, we have to combine some features (mouth, 2 eyes, face oval and nose) at their appropriate positions to say that is a face. But a CNN model would say

that if those 5 features are present, then there is high probability that it is a face, without considering where those features are present [4].

Further research in the field of deep learning may open new paths for image classification. New algorithm called Capsule Networks is being researched and is said to have promising results [5]. It will overcome all flaws of Convolutional Neural Networks.

These all were our findings when we studied Convolutional Neural Networks and built a model based on it.

6. CONCLUSION

In this project, we have built a Deep Learning Model using Convolutional Neural Network from scratch and using transfer learning. The model built from scratch was not as accurate as the one built with transfer learning. For transfer learning we had used convolutional layers of VGG19 Model. That is why the final implementations run on the transfer learning model.

Even with the flaws of Convolutional Neural Networks, the “What’s On My Screen” application can prove very useful in day-to-day life. It can serve very well as an educational tool for children and adults. If trained well with images of unique objects, it can come very handy when users come across those objects. Its text processing capability allows users to convert any text image into editable string.

The web-based GUI serves very well in analysis of the dataset and to get insight about the model’s training stats. It is useful for testing one image at a time. It can prove helpful to future students in studying about Convolutional Neural Networks.

Further, we plan to modify this project to enable users to add new categories of their choice and train the model from the main window itself. This was a humble attempt by our team to create something that will be helpful to ordinary people.

REFERENCES

- **World Wide Web**

[1] https://tryolabs.com/resources/introductory-guide-computer-vision/

[2] https://sisu.ut.ee/imageprocessing/book/1

[3] Himadri Sankar Chatterjee, "A Basic Introduction to Convolutional Neural Network", https://medium.com/@himadrisankarchatterjee/a-basic-introduction-to-convolutional-neural-network-8e39019b27c4 , July 2019

[4] Mohmoud Tarrasse, "What is wrong with Convolutional Neural Networks?", https://towardsdatascience.com/what-is-wrong-with-convolutional-neural-networks-75c2ba8fbd6f , January 2018

- **Journal Article/ IEEE Paper**

[5] Geoffrey Hilton, Sara Sabour, Nicholas Frosst, "Dynamic Routing Between Capsules", https://arxiv.org/pdf/1710.09829.pdf , November 2017