# Analytics and Systems of Big Data Project

Traffic Sign Classification System for Self-Driving Cars

Group - 3
Group members

R Shreja COE18B043
Rithic Kumar COE18B044
Shresta M COE18B048
Sreedhar Arumugam COE18B051
Viknesh Rajaramon COE18B060

May 7, 2021

# About our project

Traffic sign classification is the process of automatically recognizing traffic signs along the road, including yield signs , pedestrians , children crossing , priority signs etc. Automatic recognition of traffic signs enables us to build "smarter cars".
In order to understand and properly parse the roadway, Self-driving cars need traffic sign recognition.Similarly, “driver alert” systems inside cars need to understand the roadway around them to help aid and protect drivers. Traffic sign recognition is just one of the problems that computer vision and deep learning can solve.

While building a self-driving car, it is necessary to make sure it identifies the traffic signs with a high degree of accuracy, otherwise the results might be catastrophic. To solve this problem we use CNN and Keras and built a deep neural network model that can classify traffic signs present in the image into different categories.

## 0.1 Libraries used :

There are a few requirements we need to install . The requirements can be installed by running **pip install -r requirements.txt** We make use of Keras, Matplotlib, Scikit-learn, Pandas, PIL for the image classification task.

## 0.2 Dataset used:

In this project , we have used a public dataset on Kaggle , which is Traffic signs dataset(GTSRB - German Traffic Sign Recognition Benchmark).
Link : https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign

The dataset as a whole contains more than 50,000 images of different traffic signs , where it can further be classified into 43 different classes.Some of the classes have few images , while some have many images , so the dataset is quite varying. The dataset has train and test folder , where train folder contains images inside each class and we will be using test folder for testing our model. The size of the dataset is around 300MB.

## 0.3 Approach :

Our approach to build this traffic sign classification model is basically in four steps:

1. Explore the dataset
2. Build a CNN model
3. Train and validate the model
4. Test the model with test dataset

## 0.4 Explore the dataset

The first task of this classifier is to explore the dataset. The ’train’ folder contains about 43 folders , which means there are 43 classes. The range of the folder is 0 to 42. With the help of OS module ,we have appended the images and their respective class labels into the dataset and class labels list by itertating over all the classes.

The PIL library is used to open image content into an array.

We have stored all the images and their respective labels into a lists, in the code it is data_set and class_labels. As we need to feed the data into the model , we need to convert the list into numpy arrays.

The Shape of the data is (39209,30,30,3), which means that there are 39209 images of size $30 \times 30$ pixels and the last 3 means the data contains colored images i.e RGB value.

```
import numpy as np
import pandas as pd
import os
import cv2
import matplotlib.pyplot as plt
import tensorflow as tf
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout

data_set = []
class_labels = []
total_classes = 43
curr_path = os.getcwd()

for i in range(total_classes):
        path = os.path.join(curr_path,'train',str(i))
        images = os.listdir(path)

        for j in images:
                try:
                        image = Image.open(path + '/' + j)
                        image = image.resize((30,30))
                        image = np.array(image)
                        data_set.append(image)
                        class_labels.append(i)
                except:
                        print("Error loading the image")

data_set = np.array(data_set)
class_labels = np.array(class_labels)
```

For splitting the data into test and train data , we used train_test_split() method from sklearn package. We used to_categorical method from keras.utils package for converting the labels present in y_test and y_train into one-hot encoding.

```
x_train,x_test,y_train,y_test = train_test_split(
data_set,class_labels,test_size = 0.3, random_state = 42)
print(x_train.shape,x_test.shape)

y_train = to_categorical(y_train,43)
y_test = to_categorical(y_test,43)
```

## 0.5 Build a CNN Model

In order to classify the images into their respective classes , we will build a Convolutional neural network. For image classification purposes , CNN is the best.
The architecture of the CNN model is

1. 2 Conv2D layer (filter=32, kernel_size=(5,5), activation="relu")
2. MaxPool2D layer ( pool_size=(2,2))
3. Dropout layer (rate=0.25)
4. 2 Conv2D layer (filter=64, kernel_size=(3,3), activation="relu")
5. MaxPool2D layer ( pool_size=(2,2))
6. Dropout layer (rate=0.25)
7. Flatten layer to squeeze the layers into 1 dimension
8. Dense Fully connected layer (256 nodes, activation="relu")
9. Dropout layer (rate=0.5)
10. Dense layer (43 nodes, activation="softmax")

As we have multiple classes to categorise , we compiled the model with Adam optimizer which performs well and loss is "categorical_crossentropy"

```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu',
input_shape=x_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))

#Compilation of the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

About the following terms used above:

1. **Conv2D** is the layer to convolve the image into multiple images
2. **Activation** is the activation function.
3. **MaxPooling2D** is used to max pool the value from the given size matrix and same is used for the next 2 layers.
4. **Flatten** is used to flatten the dimensions of the image obtained after convolving it.
5. **Dense** is used to make this a fully connected model and is the hidden layer.It is the output layer contains only one neuron which decide to which category image belongs.
6. **Dropout** is used to avoid overfitting on the dataset.

## 0.6 Train and Validate the model

We will train the model using model.fit() after building the Architecture of the model. We tried with the batch size 32 and 64 , where our model performed better with batch size 64 . After 15 epochs, the accuracy was stable.

The accuracy of the model on the training dataset was 95%. We plotted the graph for accuracy and the loss , using matplotlib.

```
epochs = 15
history = model.fit(x_train, y_train, batch_size=32, epochs=epochs,..
..validation_data=(x_test, y_test))
model.save("trained_model.h5")

plt.figure(0)
plt.plot(history.history['accuracy'],label = 'Training Accuracy')
plt.plot(history.history['val_accuracy'],label = 'val accuracy')
plt.title('ACCURACY')
plt.xlabel('epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.figure(1)
plt.plot(history.history['loss'],label = 'Training loss')
plt.plot(history.history['val_loss'],label = 'val loss')
plt.title('LOSS')
plt.xlabel('epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

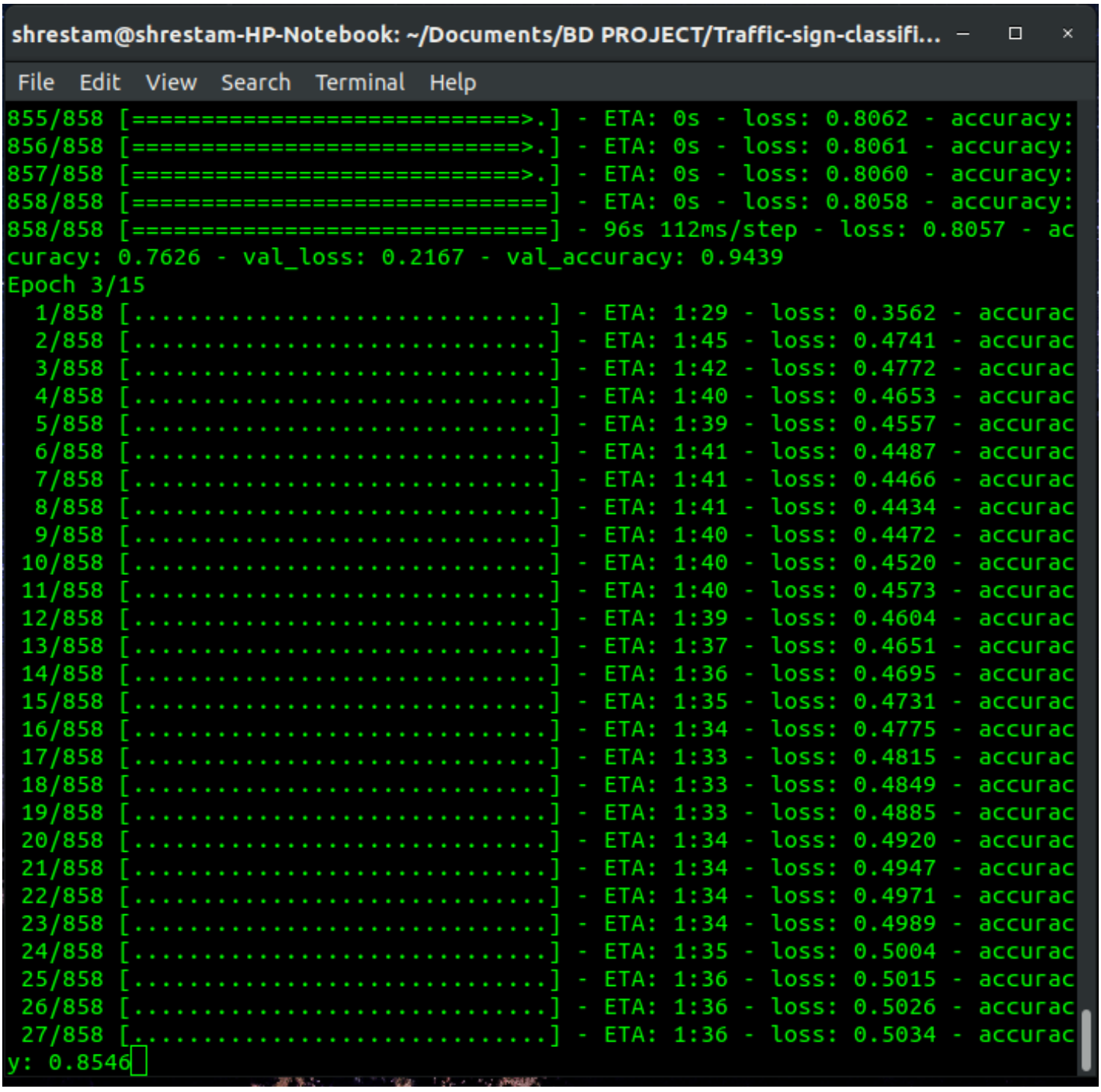


Figure 1: Screenshot of epoch 3

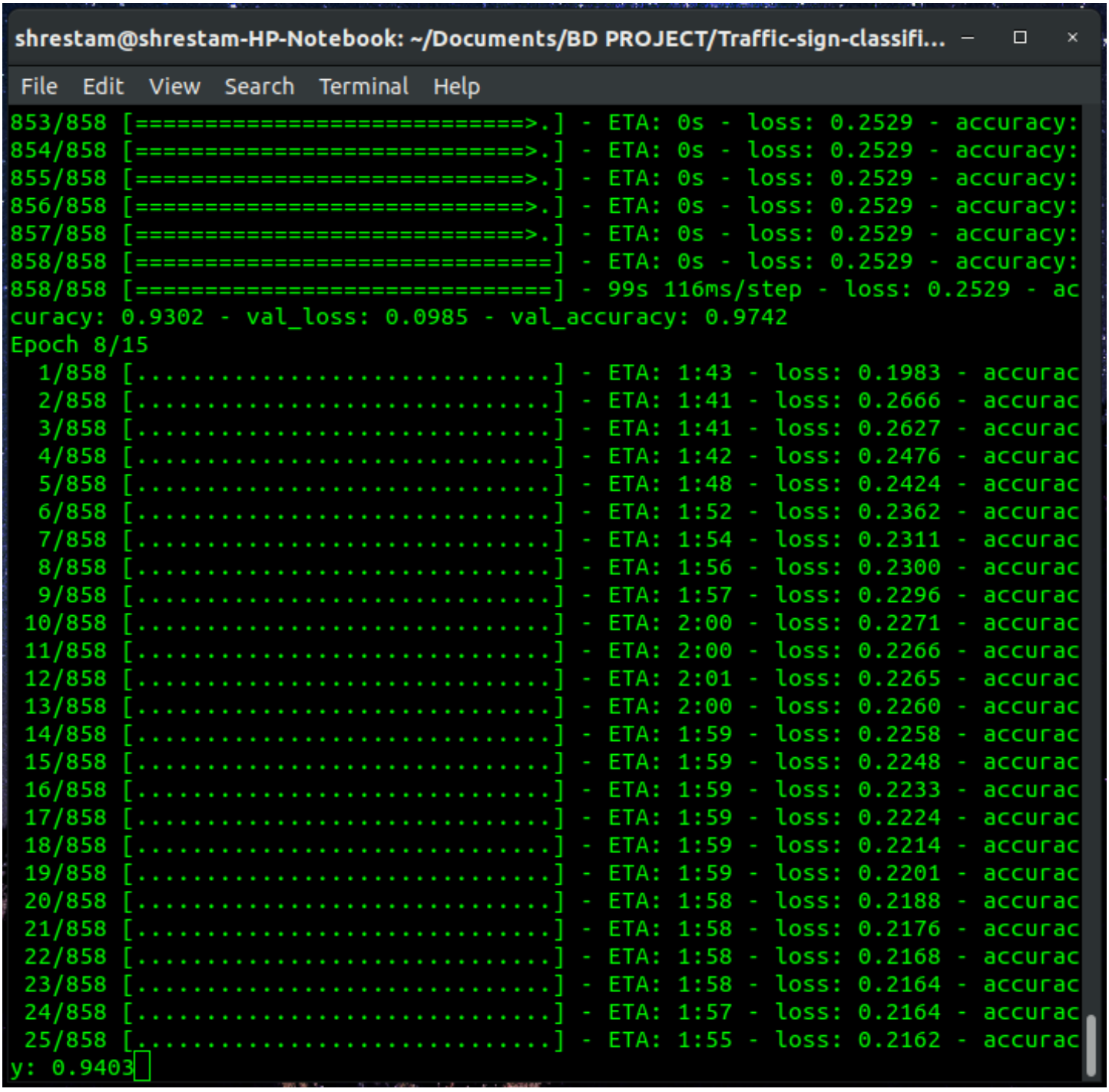


Figure 2: Screenshot of epoch 8

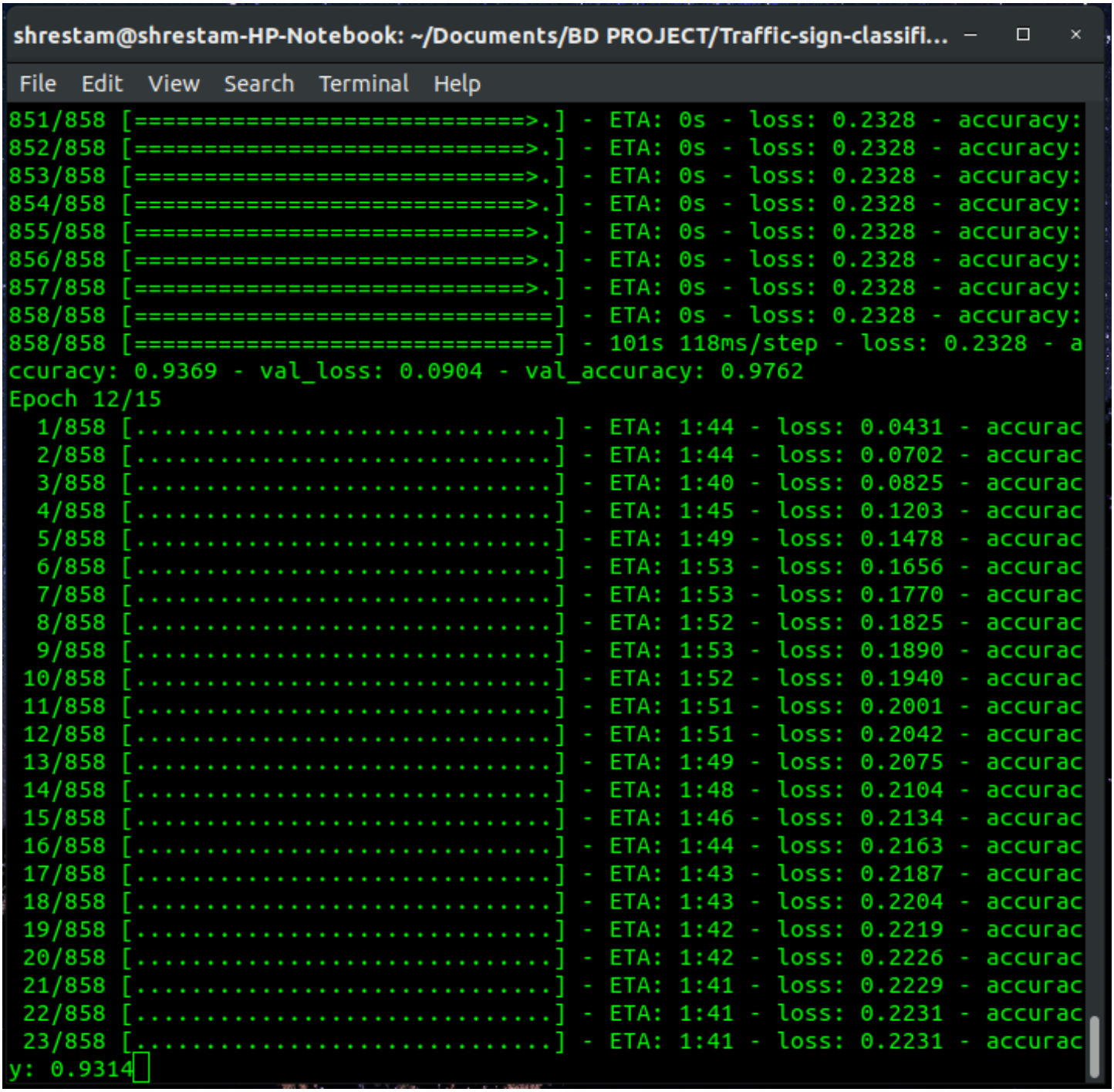


Figure 3: Screenshot of epoch 12

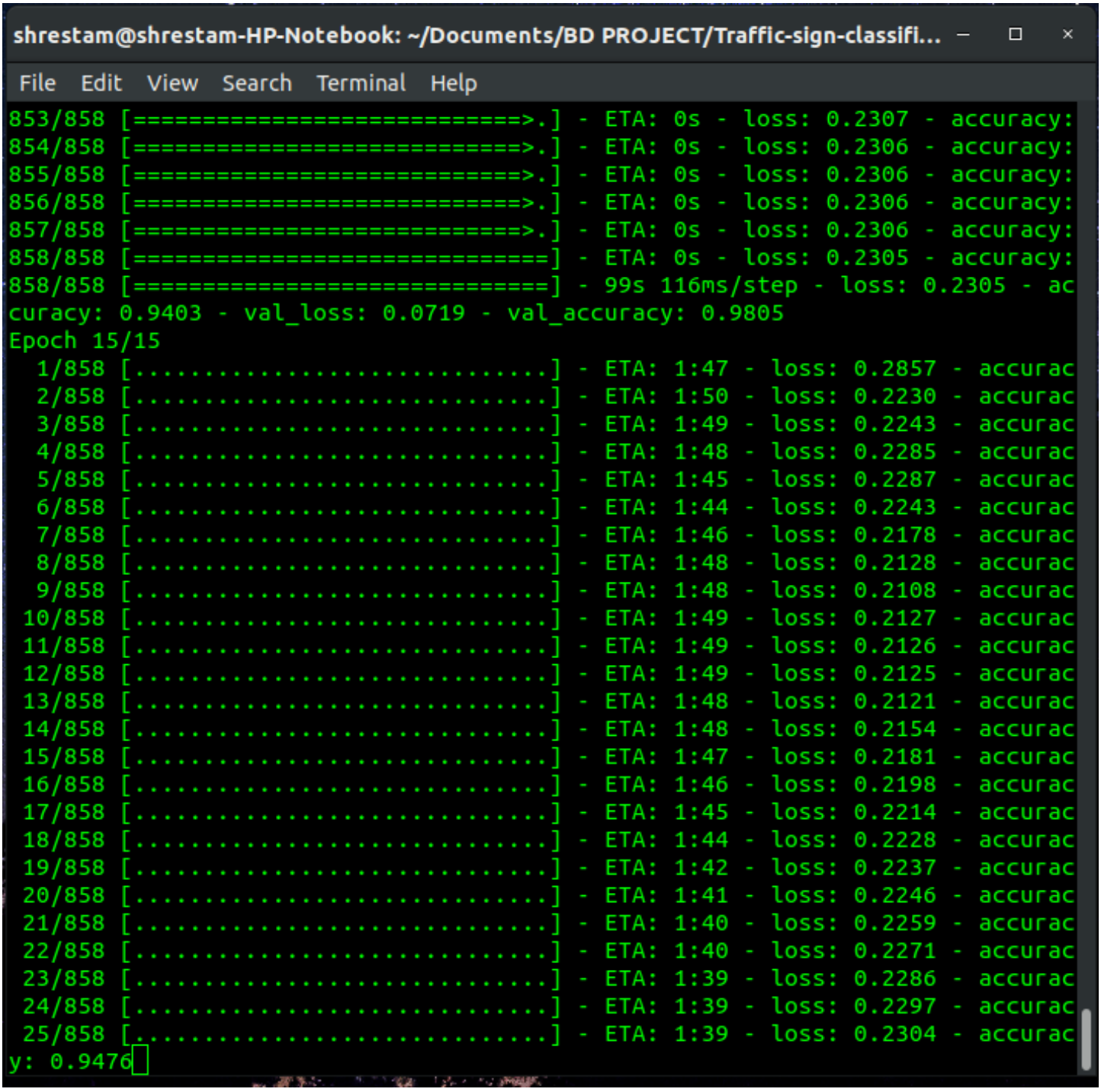


Figure 4: Screenshot of epoch 15

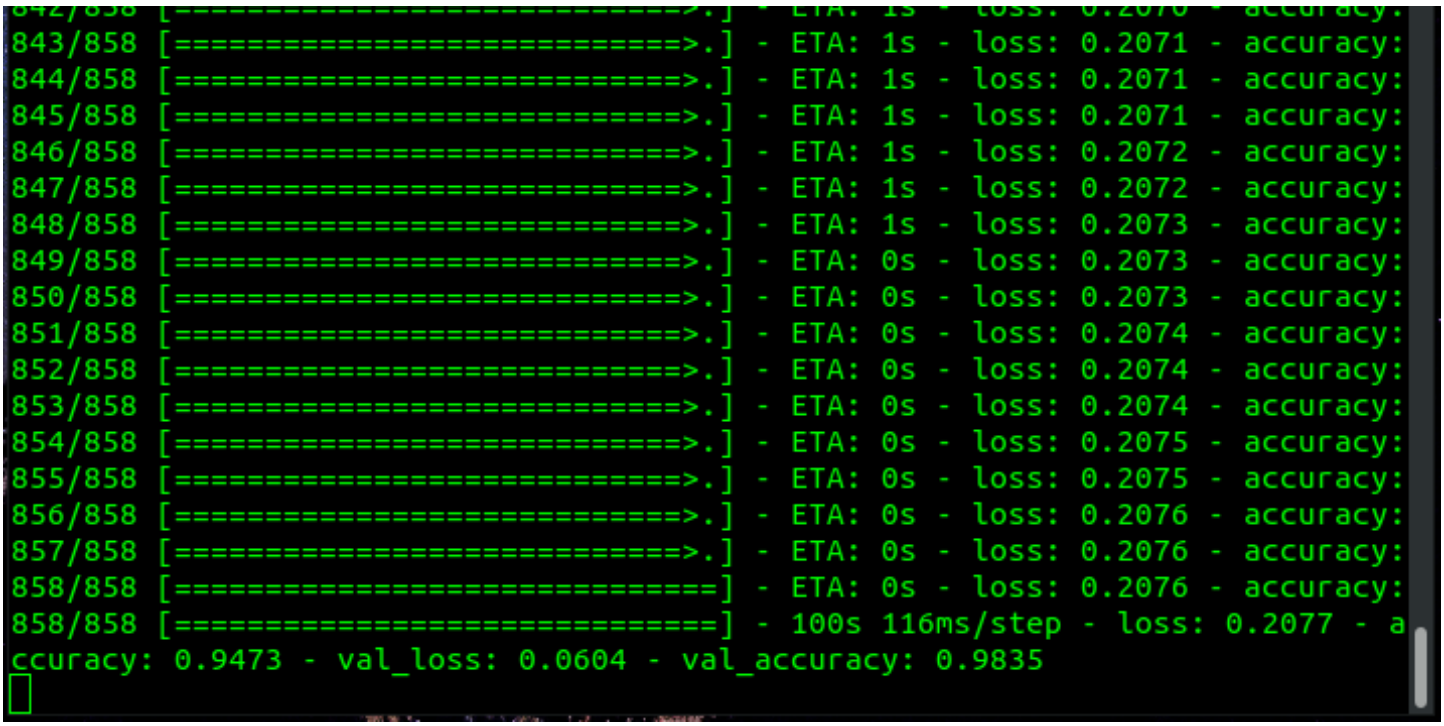


Figure 5: Screenshot of epoch 15

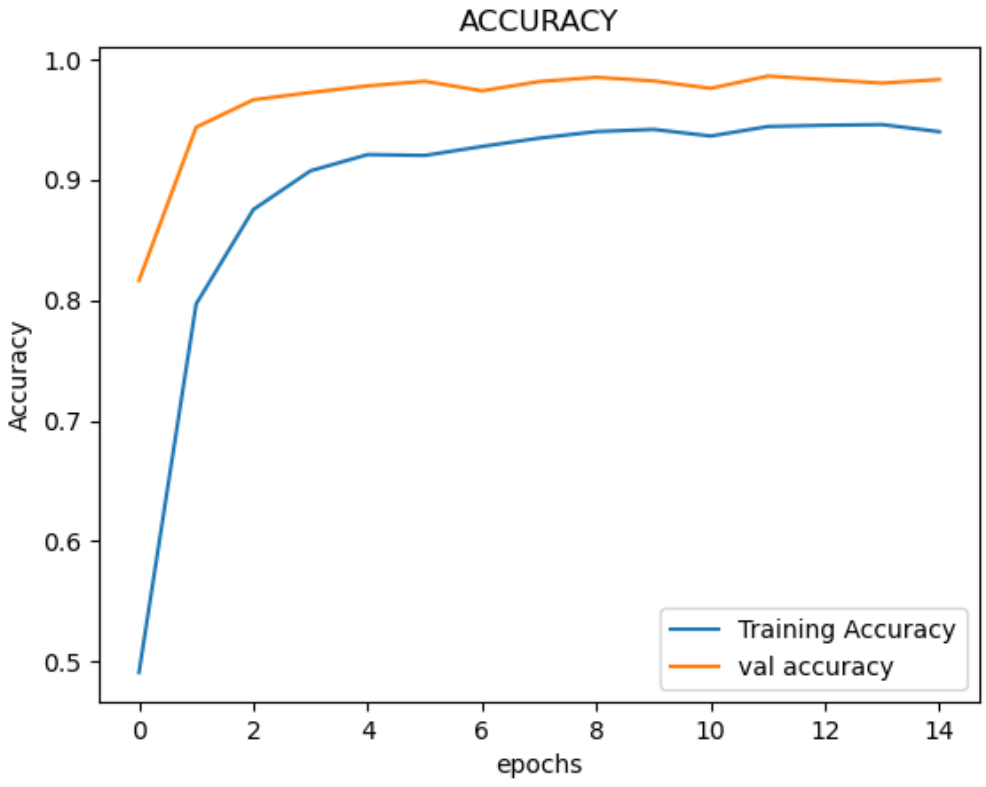


Figure 6: Graph of Accuracy v/s epochs

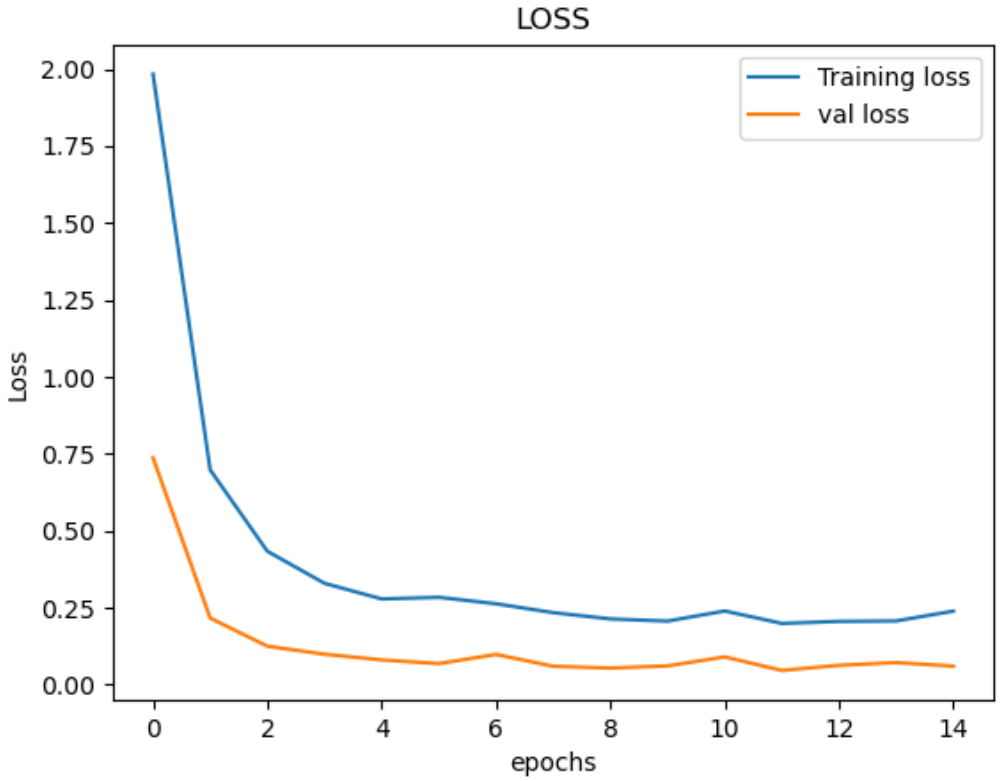


Figure 7: Graph of Loss v/s epochs

## 0.7 Testing the model with test dataset

The dataset contains the test folder and we have the details related to the image path and their respective class labels in the test.csv file.We extracted the image path and labels from csv file using pandas. Then in order to predict the model , we have to resize the images to $30 \times 30$ and make a numpy array containing all the image data.

We used accuracy_score method from sklearn.metrics,and observed how the model predicted the actual labels and we got the accuarcy as 95%.

```
y_test = pd.read_csv('Test.csv')
class_labels = y_test['ClassId'].values
images_data = y_test['Path'].values

data = []
for x in images_data:
        image = Image.open(x)
        image = image.resize((30,30))
        data.append(np.array(image))

x_test = np.array(data)
predicted_labels = model.predict_classes(x_test)
print(accuracy_score(class_labels,predicted_labels))
```

**Accuracy = 0.9437054631828978**

Using the Keras model.save() function , we are going to save the model that we have trained.
**model.save('traffic_classifier.h5')**

## 0.8 Traffic sign classifier GUI

Next , we built a GUI for traffic signs classifier with Tkinter.
Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit. In this below code , we have first loaded the trained model 'trained_model.h5' using Keras.

And then we build the GUI for uploading the image and a button is used to classify which calls the classify() function. The classify() function is converting the image into the dimension of shape (1, 30, 30, 3). This is because to predict the traffic sign we have to provide the same dimension we have used when building the model.
Then we predict the class, the model.predict_classes(image) returns us a number between (0-42) which represents the class it belongs to. We use the dictionary to get the information about the class
Firstly,we import the required libraries

```
import numpy
from keras.models import load_model
from PIL import Image, ImageTk
import tkinter as tk
from tkinter import filedialog
from tkinter import *


#load the trained model to classify sign
model = load_model('trained_model.h5')
```

Create a dictionary hashing the output of the trained_model.h5 to what the value actually depicts

```
classes = { 1:'Speed limit (20km/h)',
            2:'Speed limit (30km/h)',
            3:'Speed limit (50km/h)',
            4:'Speed limit (60km/h)',
            5:'Speed limit (70km/h)',
            6:'Speed limit (80km/h)',
            7:'End of speed limit (80km/h)',
            8:'Speed limit (100km/h)',
            9:'Speed limit (120km/h)',
           10:'No passing',
           11:'No passing veh over 3.5 tons',
           12:'Right-of-way at intersection',
           13:'Priority road',
           14:'Yield',
           15:'Stop',
           16:'No vehicles',
           17:'Veh > 3.5 tons prohibited',
           18:'No entry',
           19:'General caution',
           20:'Dangerous curve left',
           21:'Dangerous curve right',
           22:'Double curve',
           23:'Bumpy road',
           24:'Slippery road',
           25:'Road narrows on the right',
           26:'Road work',
           27:'Traffic signals',
           28:'Pedestrians',
```

```
            29:'Children crossing',
            30:'Bicycles crossing',
            31:'Beware of ice/snow',
            32:'Wild animals crossing',
            33:'End speed + passing limits',
            34:'Turn right ahead',
            35:'Turn left ahead',
            36:'Ahead only',
            37:'Go straight or right',
            38:'Go straight or left',
            39:'Keep right',
            40:'Keep left',
            41:'Roundabout mandatory',
            42:'End of no passing',
            43:'End no passing veh > 3.5 tons' }
```

As already mentioned we have used Tkinter for the gui.The below code initializes the tkinter with appropriate UI elements.

```
#initialise GUI
top=tk.Tk()
top.geometry('800x600')
top.title('Traffic sign classification')
top.configure(background='#cae7fc')

label=Label(top,background='#cae7fc', font=('Helvetica',15,'bold'))
sign_image = Label(top)
```

The classify function takes the

```
def classify(file_path):
    global label_packed
    image = Image.open(file_path)
    image = image.resize((30,30))
    image = numpy.expand_dims(image, axis=0)
    image = numpy.array(image)
    print(image.shape)
    pred = model.predict_classes([image])[0]
    sign = classes[pred+1]
    print(sign)
    label.configure(foreground='#011638', text=sign)


def show_classify_button(file_path):
    classify_b=Button(top,text="Classify Image",
    command=lambda: classify(file_path),padx=10,pady=5)
    classify_b.configure(background='#364156', foreground='white',
    font=('Helvetica',10,'bold'))
    classify_b.place(relx=0.79,rely=0.46)

def upload_image():
    try:
        file_path=filedialog.askopenfilename()
        uploaded=Image.open(file_path)
        uploaded.thumbnail(((top.winfo_width()/2.25),(top.winfo_height()/2.25)))
        im=ImageTk.PhotoImage(uploaded)
```

```
        sign_image.configure(image=im)
        sign_image.image=im
        label.configure(text='')
        show_classify_button(file_path)
    except:
        pass

upload=Button(top,text="Upload an image",command=upload_image,padx=10,pady=5)
upload.configure(background='#364156', foreground='white',font=('Helvetica',10,'bold'))

upload.pack(side=BOTTOM,pady=50)
sign_image.pack(side=BOTTOM,expand=True)
label.pack(side=BOTTOM,expand=True)
heading = Label(top, text="KNOW THE INPUT TRAFFIC SIGN",pady=20, font=('arial',20,'bold'))
heading.configure(background='#cae7fc',foreground='#364156')
heading.pack()
top.mainloop()
```

## 0.9 Demo video and Output screenshots:

Here we have included the link for the demo video of working gui.
Link: https://drive.google.com/file/d/1XnZfPhl1xlYQ3keaN3CHmAxWdfXL4UQn/view?usp=sharing


Figure 8: GUI of traffic sign classifier


Figure 9: Output for case 1


Figure 10: Output for case 2

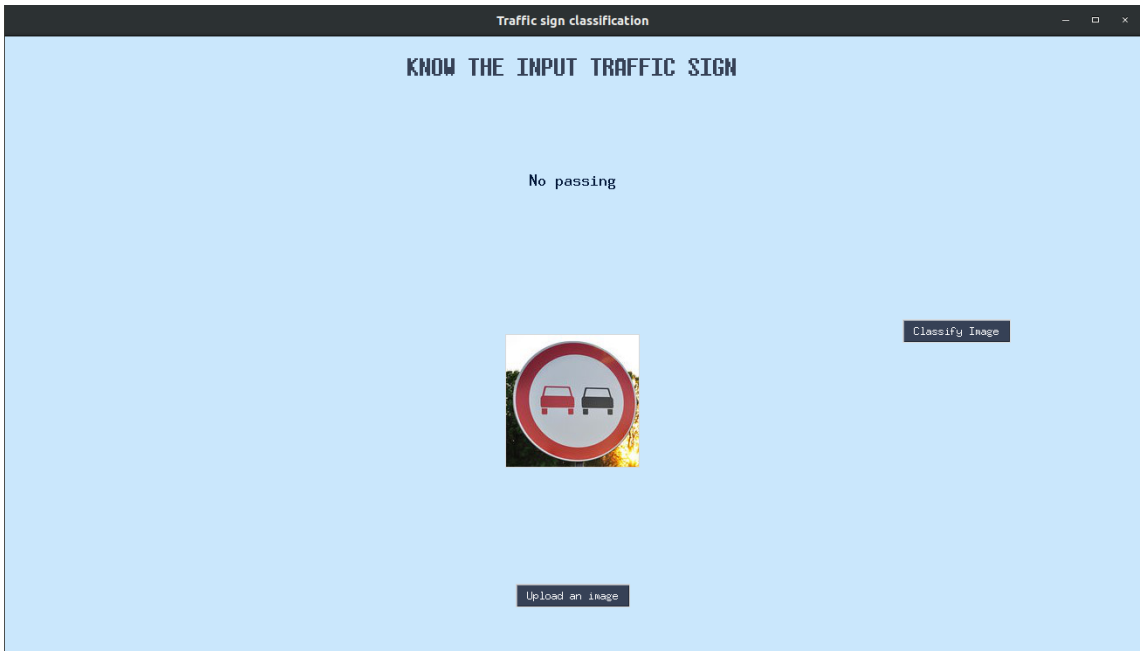


Figure 11: Output for case 3


Figure 12: Output for case 4