Traffic Sign Recognition using Convolutional Neural Network

Roll No	Name	Mentor
160118733130	Jyoshna Kandi	Smt.E.Kalpana
160118733131	Lahari Madishetty	Smt.E.Kalpana

INDEX

ABSTRACT

Traffic sign recognition system (TSRS) is a significant portion of intelligent transportation systems (ITS). Being able to identify traffic signs accurately and effectively can improve driving safety. Nowadays we can see that there is a huge increase in the number of accidents caused due to the increase in the number of vehicles and also due to the poor enforcement of laws,carelessness.One of the reasons is that people don't recognize or follow traffic sign boards. To deal with this problem we have designed a model which recognizes and classifies the traffic signs.This can reduce road accidents.

In this project, we will build a deep neural network model that can classify traffic signs present in the image into different classes. With this model, we will be able to read and understand traffic signs which are a very important task for all autonomous vehicles.

The traffic-sign detection and identification can be done through three parts: pre-processing, detection and classification. In the pre-processing phase, the static color image is enhanced, and then the color space is transformed. At this phase, we eliminate the unused information in the images and restore the useful information. Such operation will improve the accuracy of the detection and classification stages of signs.

In the detection stage, road signs are segmented on the basis of shape and color information of the image. At this phase, we will extract the areas of interest from the image and prepare for the classification stage. The color and shape information of traffic signs are two significant pieces of information, each traffic sign has a specific color and fixed shape.

In the recognition and classification stage, the extracted and segmented traffic sign area is used as input, and the convolutional neural network in deep learning is used to recognize and classify the detected information.

In this project we use CNN for classification of detected signs.Convolutional layer is the core part of convolutional neural network, and its main function is feature selection.The test result of the trained CNN shows the accuracy of the model in detecting and recognizing traffic signs.

INTRODUCTION

Nowadays, there is a lot of attention being given to the ability of the car to drive itself. One of the many important aspects for a self-driving car is the ability for it to detect traffic signs in order to provide safety and security for the people not only inside the car but outside of it.So, for achieving accuracy in this technology, the vehicles should be able to interpret traffic signs and make decisions accordingly.

The main objective of our project is to design and construct a computer based system which can automatically detect the road signs so as to provide assistance to the user or the machine so that they can take appropriate actions. The proposed approach consists of building a model using convolutional neural networks by extracting traffic signs from an image using color information. We have used convolutional neural networks (CNN) to classify the traffic signs and we used color based segmentation to extract/crop signs from images.

In this project, we are going to use the public dataset available at Kaggle which is German Traffic Sign Recognition Benchmark(GTSRB) dataset.This dataset contains more than 50,000 images of different traffic signs. It is further classified into 43 different classes. The dataset contains a train folder which is quite varying, some of the classes have many images while some classes have few images. The size of the dataset is around 300MB. The dataset has a train folder which contains images inside each class and a test folder which we will use for testing our model using CNN.

BACKGROUND INFORMATION

Many different techniques have been applied to detect traffic signs.Most of these techniques are based on using HOG (histogram of oriented gradients) and SIFT (scale-invariant feature transform features.

In our approach we use biologically inspired convolutional neural networks to build a model which can predict the type of traffic sign. One such related work based on convolutional neural networks is published in ‘Traffic Sign Recognition with Multi-Scale Convolutional Networks’ by Pierre Sermanet and Yann LeCun.

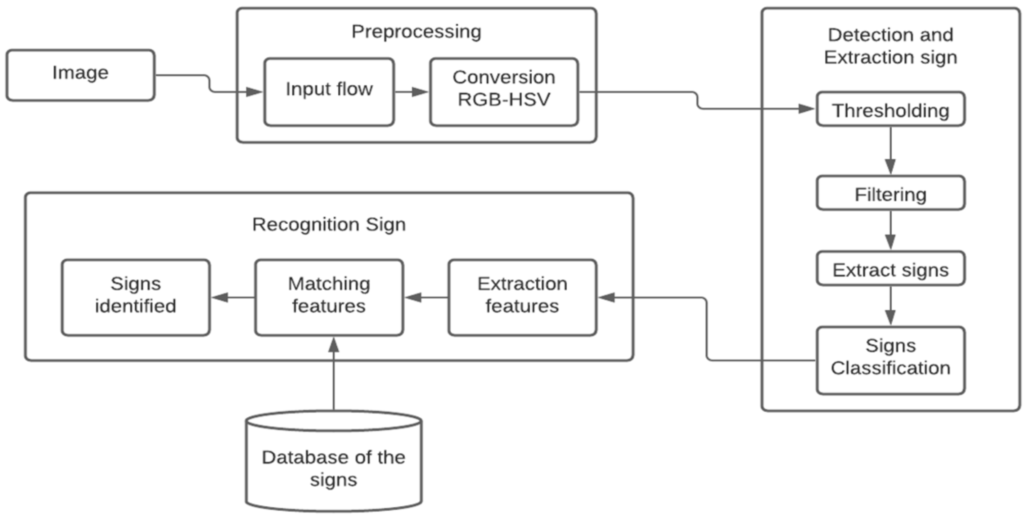
For a long time, there were no public and challenging datasets available in this area, but this situation changed in 2011. Larsson and Felsberg and Stallkamp et al. introduced challenging databases, including annotations for traffic sign detection classification. These databases included the Belgium Traffic Sign Classification (BTSC), the German traffic sign detection benchmark (GTSDB), and the German Traffic Sign Recognition Benchmark (GTSRB) databases. In particular, the GTSDB and GTRSB have attracted more scholars to find new methods to verify through this database, and some of them have achieved good results. The existence of open databases has made the research methods comparable. The above traffic signs databases belong to European traffic laws and regulations.

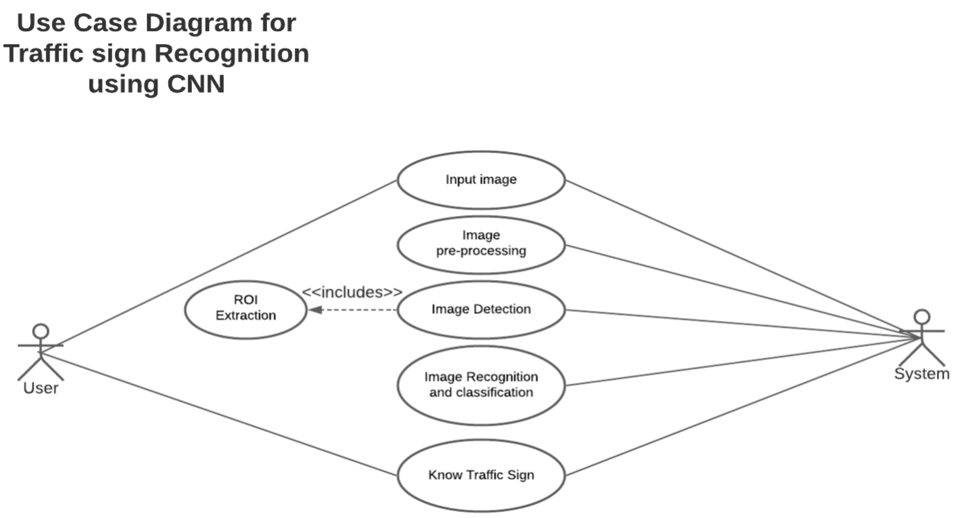
SCOPE

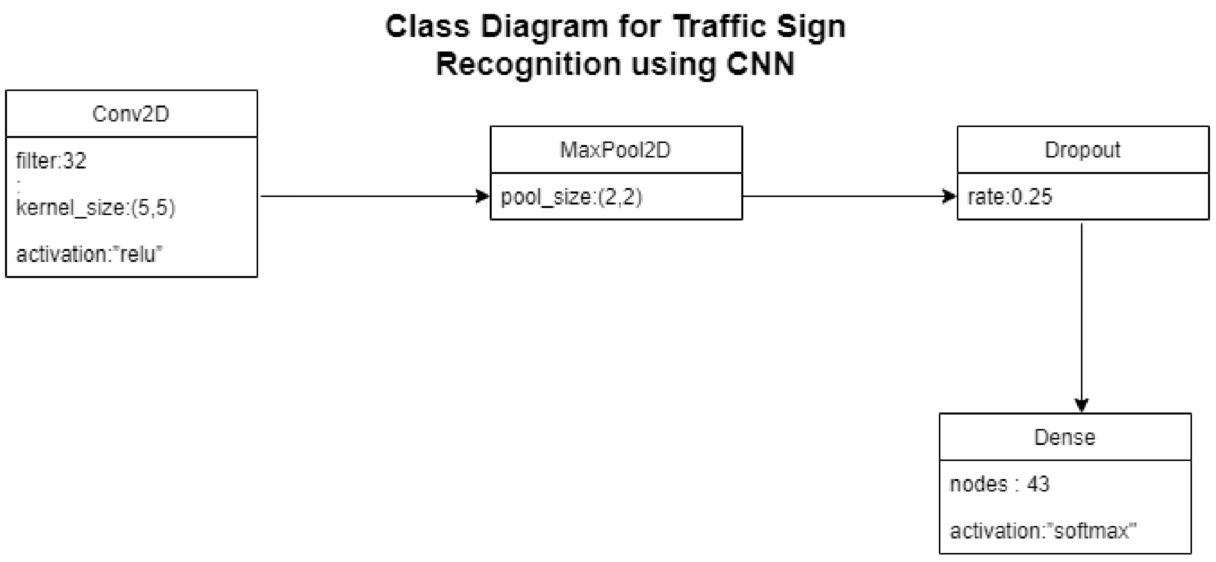
Traffic sign detection and recognition plays an important role in expert systems, such as traffic assistance driving systems and automatic driving systems. It instantly assists drivers or automatic driving systems in detecting and recognizing traffic signs effectively.Our objective is to apply Convolutional Neural Network(CNN) for Traffic Sign Recognition and correctly classify the Traffic Sign into one of 43 different categories.Future improvements can be made for extracting signs from test images by using advanced segmentation methods.It is likely that we would obtain better results by reinforcing the convolution stage of the network with more layers in order to extract more features.

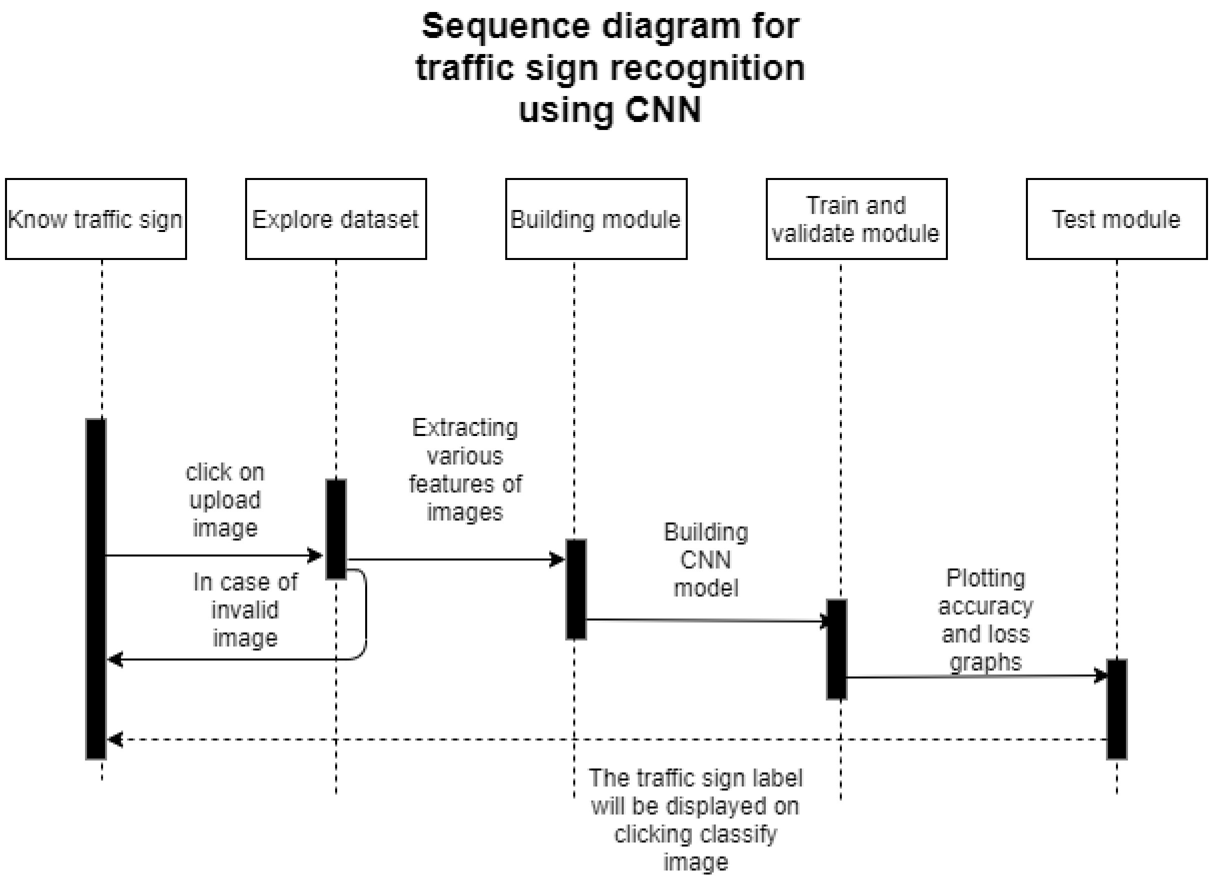
DESIGN

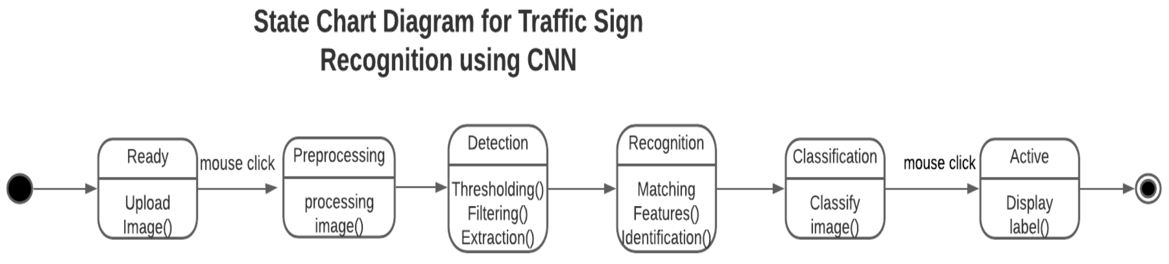
Block Diagram for Traffic sign Recognition

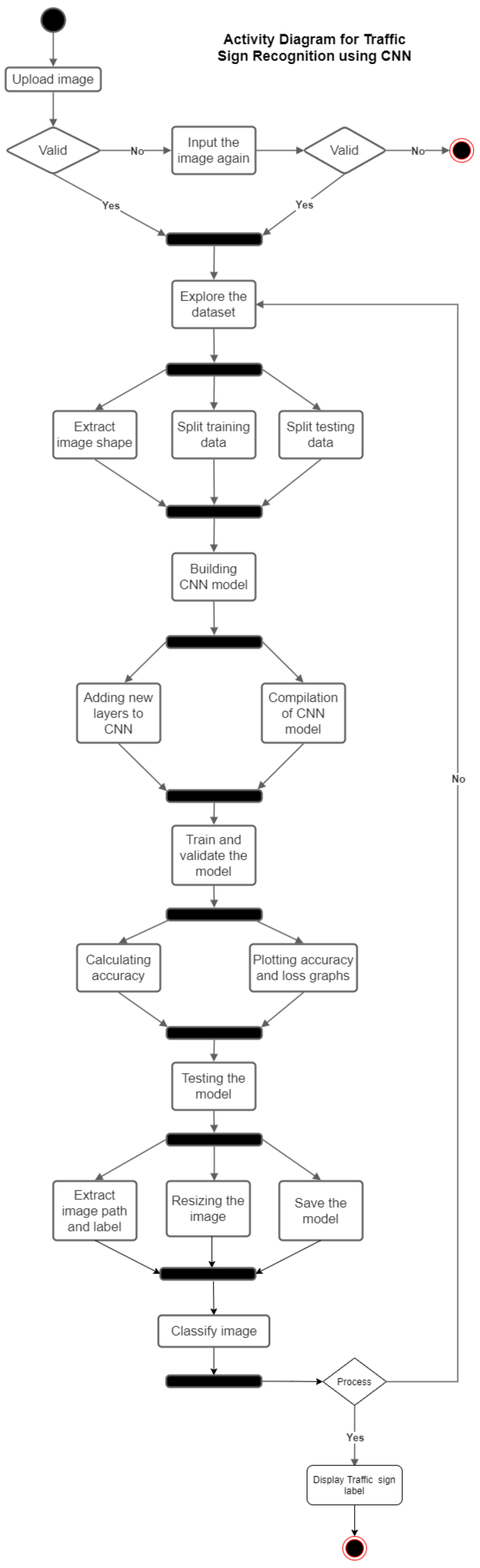





Class Diagram for Traffic Sign Recognition using CNN
Conv2D
filter:32
kernel_size:(5,5)
activation:"relu"
MaxPool2D
pool_size:(2,2)
Dropout
rate:0.25
Dense
nodes : 43
activation:"softmax"


Sequence diagram for traffic sign recognition using CNN
Know traffic sign
Explore dataset
Building module
Train and validate module
Test module
click on upload image
In case of invalid image
Extracting various features of images
Building CNN model
Plotting accuracy and loss graphs
The traffic sign label will be displayed on clicking classify image


State Chart Diagram for Traffic Sign Recognition using CNN
Ready
Upload Image()
mouse click
Preprocessing
processing image()
Detection
Thresholding()
Filtering()
Extraction()
Recognition
Matching Features()
Identification()
Classification
Classify image()
mouse click
Active
Display label()


Activity Diagram for Traffic Sign Recognition using CNN
Upload image
Valid
No
Input the image again
Valid
No
Yes
Yes
Explore the dataset
Extract image shape
Split training data
Split testing data
Building CNN model
Adding new layers to CNN
Compilation of CNN model
Train and validate the model
Calculating accuracy
Plotting accuracy and loss graphs
Testing the model
Extract image path and label
Resizing the image
Save the model
Classify image
Process
No
Yes
Display Traffic sign label

IMPLEMENTATION

Our approach to building this traffic sign classification model is discussed in four steps:

- Explore the dataset
- Build a CNN model
- Train and validate the model
- Test the model with test dataset

Traffic_signs.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from PIL import Image
import os
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
data = []
labels = []
classes = 43
cur_path = os.getcwd()
#Retrieving the images and their labels
for i in range(classes):
    path = os.path.join(cur_path,'train',str(i))
    images = os.listdir(path)
    for a in images:
        try:
            image = Image.open(path + '\\'+ a)
            image = image.resize((30,30))
             image = np.array(image)
            #sim = Image.fromarray(image)
            data.append(image)
            labels.append(i)
        except:
            print("Error loading image")
#Converting lists into numpy arrays
data = np.array(data)
labels = np.array(labels)
print(data.shape, labels.shape)
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2,random_state=42)
```

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
#Converting the labels into one hot encoding
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)
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'])
epochs = 15
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test,
y_test))
model.save("my_model.h5")
#plotting graphs for accuracy
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()
#testing accuracy on test dataset
from sklearn.metrics import accuracy_score
y_test = pd.read_csv('Test1.csv')
labels = y_test["ClassId"].values
imgs = y_test["Path"].values
```

```
data=[]
for img in imgs:
   image = Image.open(img)
   image = image.resize((30,30))
   data.append(np.array(image))
X_test=np.array(data)
pred = model.predict_classes(X_test)
#Accuracy with the test data
from sklearn.metrics import accuracy_score
print(accuracy_score(labels, pred))
model.save('traffic_classifier.h5')
```

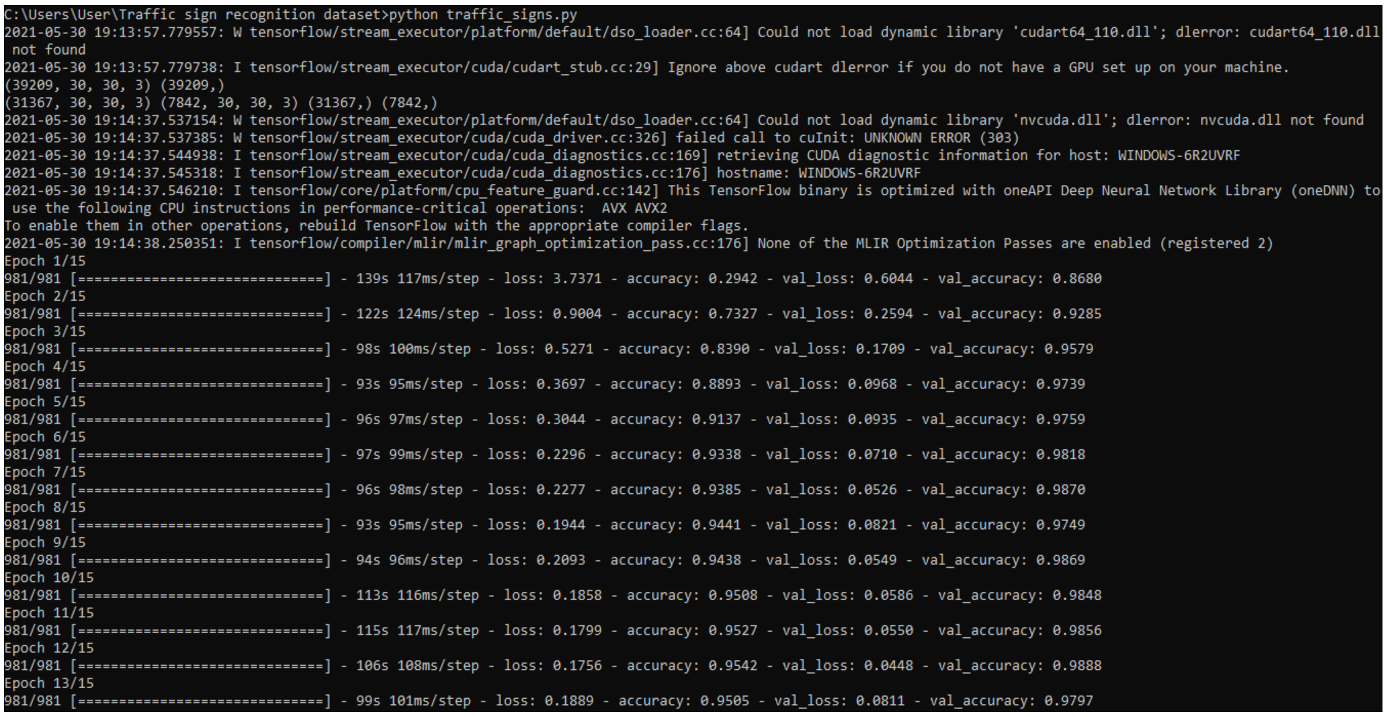
Now we are going to build a graphical user interface for our traffic signs classifier with Tkinter.

Gui.py

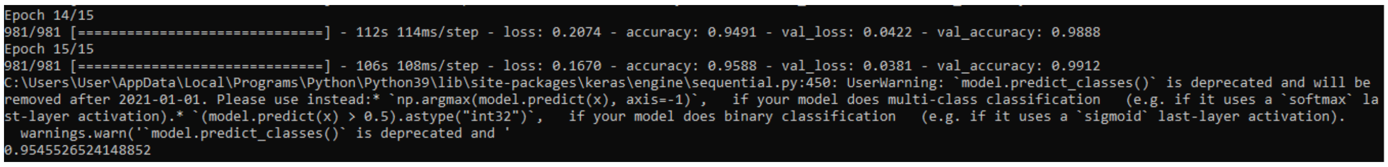
```
import tkinter as tk
from tkinter import filedialog
from tkinter import *
from PIL import ImageTk, Image
import numpy
#load the trained model to classify sign
from keras.models import load_model
model = load_model('traffic_classifier.h5')
#dictionary to label all traffic signs class.
classes = { 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' }
#initialise GUI
top=tk.Tk()
top.geometry('800x600')
top.title('Traffic sign classification')
top.configure(background='#9fede7')
label=Label(top,background='#9fede7', font=('arial',15,'bold'))
sign_image = Label(top)
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)
   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="ClassifyImage",command=lambda:classify(file_path),padx
=10,pady=5)
            classify_b.configure(background='#FF69B4',foreground='white',font=('arial',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='#FF69B4', foreground='white',font=('arial',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 Your Traffic Sign",pady=20, font=('arial',20,'bold'))
heading.configure(background='#9fede7',foreground='#364156')
heading.pack()
top.mainloop()
```

RESULT ANALYSIS

The test set was obtained by splitting the whole dataset into 80% train data and 20% test data. In this project, we have successfully classified the traffic signs with 95% accuracy and also visualized how our accuracy and loss changes with time, which is pretty good from a simple CNN model.

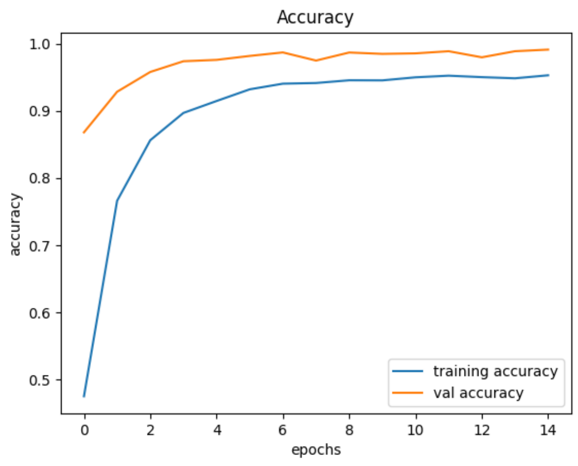


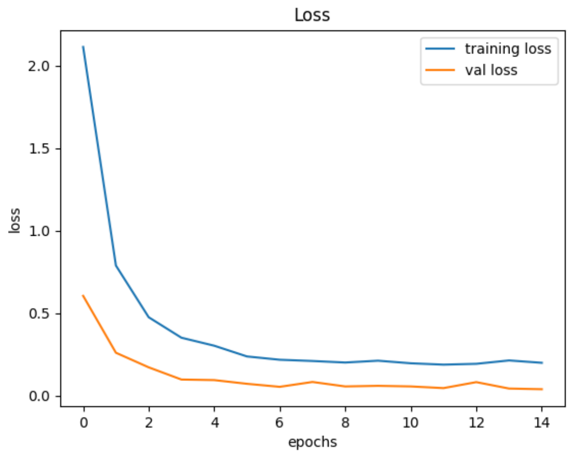
```
C:\Users\User\Traffic sign recognition dataset>python traffic_signs.py
2021-05-30 19:13:57.779557: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudart64_110.dll'; dlerror: cudart64_110.dll
 not found
2021-05-30 19:13:57.779738: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
(39209, 30, 30, 3) (39209,)
(31367, 30, 30, 3) (7842, 30, 30, 3) (31367,) (7842,)
2021-05-30 19:14:37.537154: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'nvcuda.dll'; dlerror: nvcuda.dll not found
2021-05-30 19:14:37.537385: W tensorflow/stream_executor/cuda/cuda_driver.cc:326] failed call to cuInit: UNKNOWN ERROR (303)
2021-05-30 19:14:37.544938: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic information for host: WINDOWS-6R2UVRF
2021-05-30 19:14:37.545318: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: WINDOWS-6R2UVRF
2021-05-30 19:14:37.546210: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to
 use the following CPU instructions in performance-critical operations:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-05-30 19:14:38.250351: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/15
981/981 [==============================] - 139s 117ms/step - loss: 3.7371 - accuracy: 0.2942 - val_loss: 0.6044 - val_accuracy: 0.8680
Epoch 2/15
981/981 [==============================] - 122s 124ms/step - loss: 0.9004 - accuracy: 0.7327 - val_loss: 0.2594 - val_accuracy: 0.9285
Epoch 3/15
981/981 [==============================] - 98s 100ms/step - loss: 0.5271 - accuracy: 0.8390 - val_loss: 0.1709 - val_accuracy: 0.9579
Epoch 4/15
981/981 [==============================] - 93s 95ms/step - loss: 0.3697 - accuracy: 0.8893 - val_loss: 0.0968 - val_accuracy: 0.9739
Epoch 5/15
981/981 [==============================] - 96s 97ms/step - loss: 0.3044 - accuracy: 0.9137 - val_loss: 0.0935 - val_accuracy: 0.9759
Epoch 6/15
981/981 [==============================] - 97s 99ms/step - loss: 0.2296 - accuracy: 0.9338 - val_loss: 0.0710 - val_accuracy: 0.9818
Epoch 7/15
981/981 [==============================] - 96s 98ms/step - loss: 0.2277 - accuracy: 0.9385 - val_loss: 0.0526 - val_accuracy: 0.9870
Epoch 8/15
981/981 [==============================] - 93s 95ms/step - loss: 0.1944 - accuracy: 0.9441 - val_loss: 0.0821 - val_accuracy: 0.9749
Epoch 9/15
981/981 [==============================] - 94s 96ms/step - loss: 0.2093 - accuracy: 0.9438 - val_loss: 0.0549 - val_accuracy: 0.9869
Epoch 10/15
981/981 [==============================] - 113s 116ms/step - loss: 0.1858 - accuracy: 0.9508 - val_loss: 0.0586 - val_accuracy: 0.9848
Epoch 11/15
981/981 [==============================] - 115s 117ms/step - loss: 0.1799 - accuracy: 0.9527 - val_loss: 0.0550 - val_accuracy: 0.9856
Epoch 12/15
981/981 [==============================] - 106s 108ms/step - loss: 0.1756 - accuracy: 0.9542 - val_loss: 0.0448 - val_accuracy: 0.9888
Epoch 13/15
981/981 [==============================] - 99s 101ms/step - loss: 0.1889 - accuracy: 0.9505 - val_loss: 0.0811 - val_accuracy: 0.9797
```



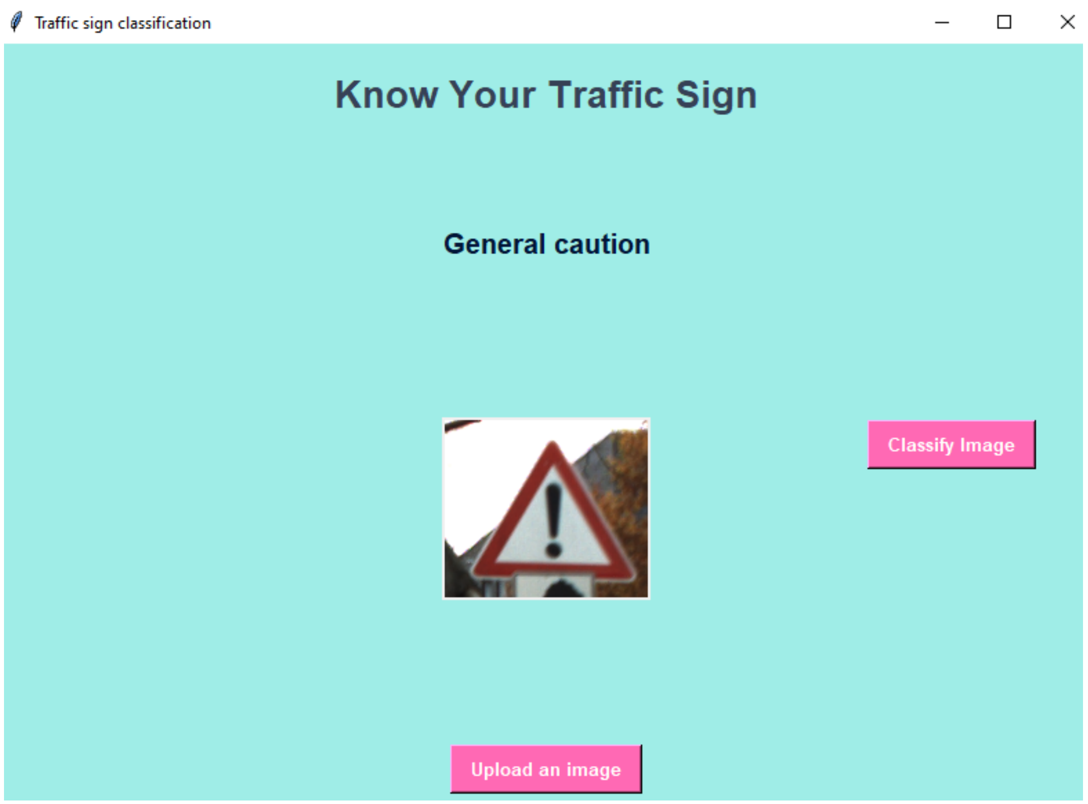
```
Epoch 14/15
981/981 [==============================] - 112s 114ms/step - loss: 0.2074 - accuracy: 0.9491 - val_loss: 0.0422 - val_accuracy: 0.9888
Epoch 15/15
981/981 [==============================] - 106s 108ms/step - loss: 0.1670 - accuracy: 0.9588 - val_loss: 0.0381 - val_accuracy: 0.9912
C:\Users\User\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\engine\sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will be
removed after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it uses a `softmax` la
st-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
0.9545526524148852
```

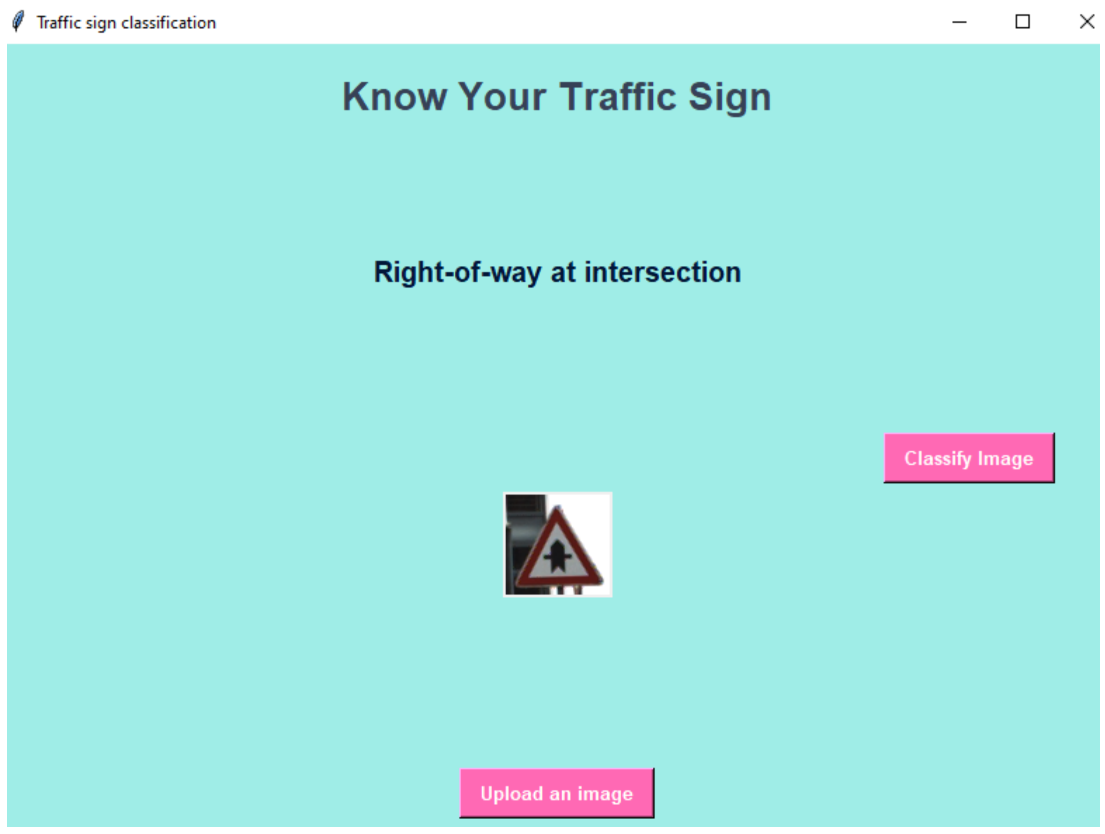
The following graphs were obtained by plotting the Training and validation accuracy and loss where the dependent variable is accuracy in first graph and loss in second graph and independent variable is epochs in both the graphs.

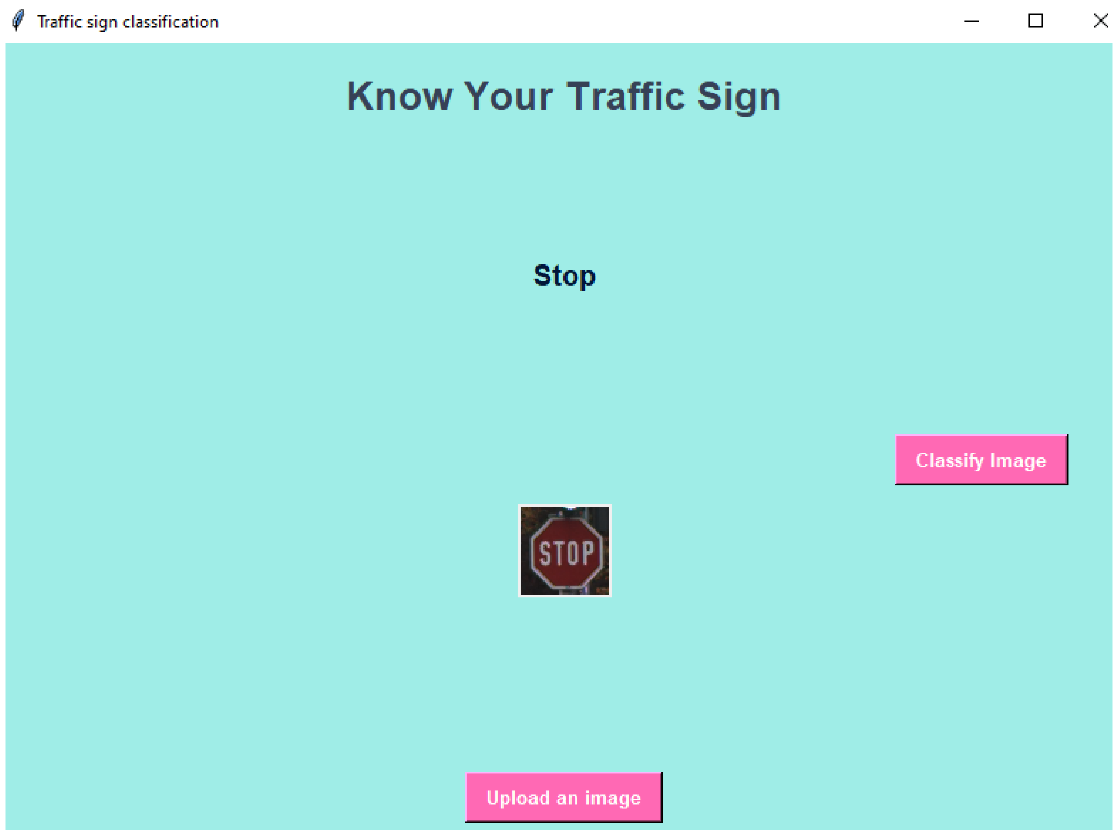


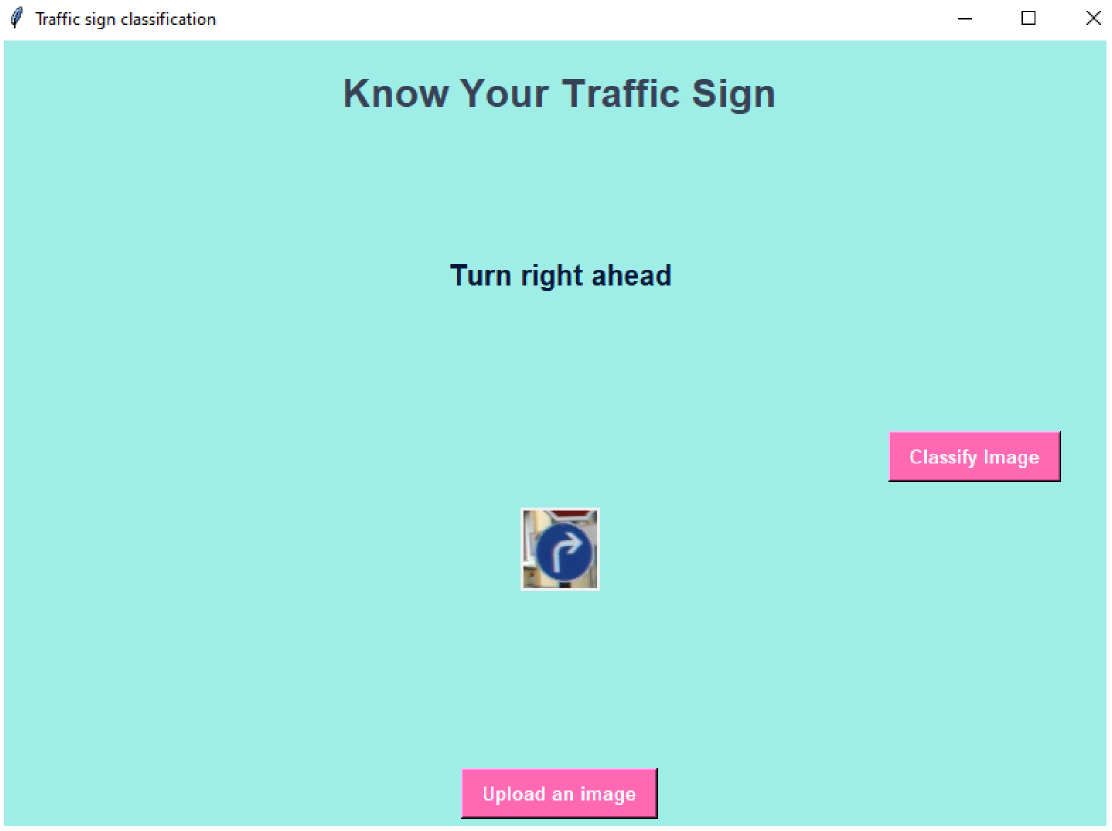


The following are the results that we have obtained after uploading an input image using the upload an image button.The title of the image will be displayed after clicking the classify image button.




Traffic sign classification
Know Your Traffic Sign
Right-of-way at intersection
Classify Image
Upload an image


Traffic sign classification
Know Your Traffic Sign
Stop
Classify Image
Upload an image



CONCLUSION

We have observed that, by using image preprocessing, traffic sign detection, recognition and classification, this method can effectively detect and identify traffic signs.We have trained our model with different batch sizes(for eg.32,64,128 etc.),but our model performed well with a batch size of 32.Test result displays that the accuracy of this method is 9% and the CNN is doing a good job in classifying different types of traffic signs.

REFERENCES

[1] **Traffic signs dataset**
[2] **Paper related to Traffic sign recognition**
[3] **Deep insight on CNN**
[4] **Tutorial on Keras**
[5] **Tutorial on how to build a CNN model**