

# Traffic Board Sign Detector

TEAM 3

Naga Surendra Bethapudi

Kesava Sai Krishna Puligadda

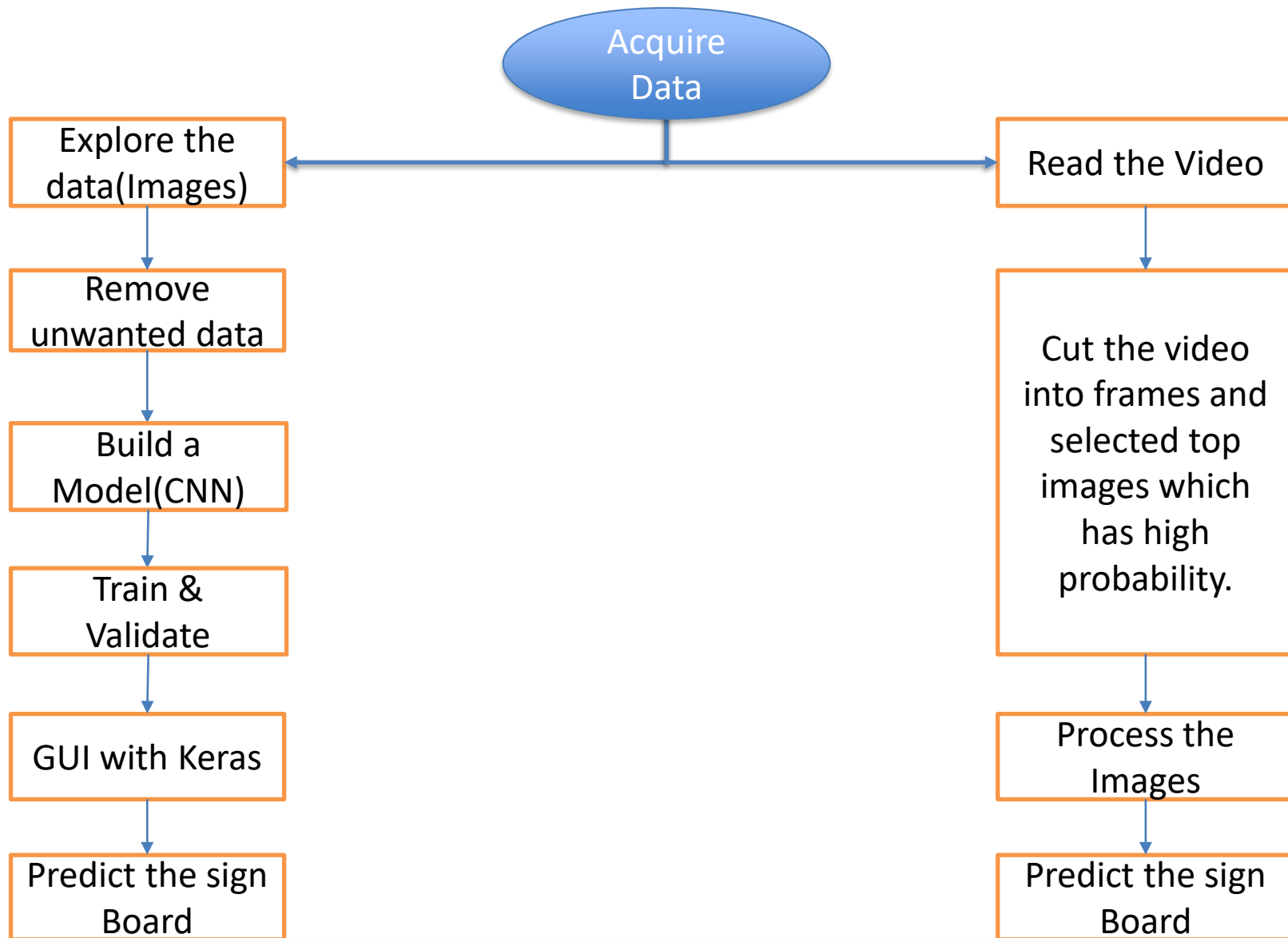
Sai Krishna Yarraguntla

# Project Purpose

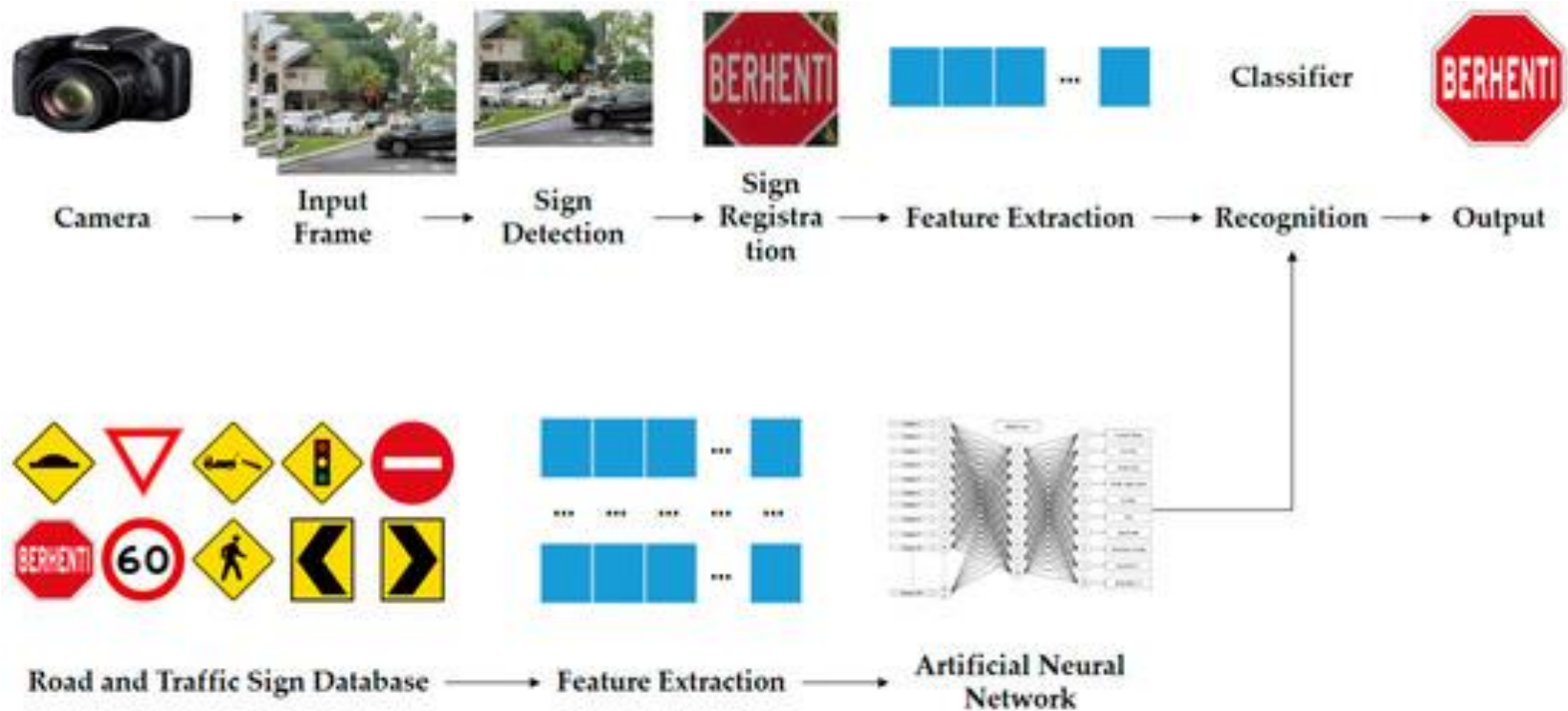
- To develop a responsive model that allows to vehicles to detect the traffic boards accurately.
- This model can be linked to vehicle software.

# Project Details

- Predicting the sign board from images.
- Predicting the sign board on frames (Video)



# Model



# Progress in the project

- We want to do the project in two parts:
  1. Predicted the sign boards from images.
  2. Predicted the sign boards from video.
- Predicted the signboards from video in two ways:
- Without OpenCV:
  - Recorded video in external device and converted the video to frames by using ffmpeg.
  - Passed each frame to the model and printed the model prediction on the image.
- With OpenCV:
  - Recorded video in external device and passed that video to the model and printed model prediction on the video

# Steps followed in the code

1. Importing the libraries and data.
2. Overlook into data.
3. Building a CNN model and finding the accuracy.
4. Accuracy and loss plots over train and test data
5. Saving the model.
6. Building a GUI by using above sequential model
7. Creating tkinter:
  1. Design
  2. Prediction function
  3. Display prediction
  4. Upload an Image

# Steps followed in the code

## Increment 2 :

1. Recorded a video in external device.
2. Converted the video to frames using ffmpeg.
3. Passed every frame to the model and labeled the model prediction on the images.
4. By using Open CV passed the video to the model and printed the predictions on the image.



# 1. Importing the libraries and data.

*#Importing the libraries*

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from PIL import Image
import os
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from keras.layers import Dense, Dropout, Flatten
```

*#Loading the data*

```
Images_data = [] #Loading Images to Images_data List
Images_labels = [] #Loading Labels to Images_labels List
classes = 43 #Classes

for i in range(classes): #Looping all the classes
    path = os.path.join('C:\\Users\\suren\\Downloads\\Project', 'Train', str(i))
    images = os.listdir(path)

    for a in images: #Looping through all the images
        image = Image.open(path + '\\ ' + a)
        image = image.resize((32,32)) #Resizing the images
        image = np.array(image)
        Images_data.append(image) #Appending all the images to Images_data List
        Images_labels.append(i) #Appending all the labels to Image_labels List

Images_data = np.array(Images_data) #List to arrays
Images_labels = np.array(Images_labels) #List to arrays
```

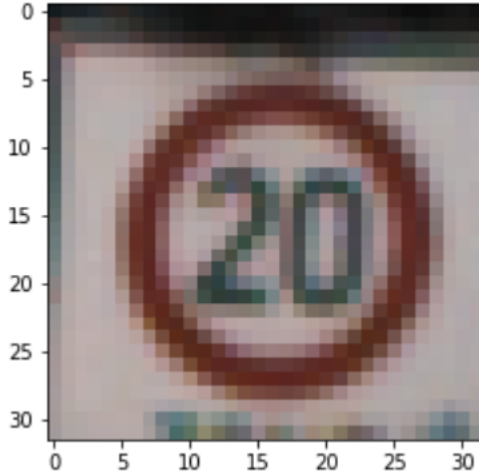
## 2. Overlook into data.

```
#printing the size of data and labels  
print('Size of Images : ', Images_data.shape)  
print('Size of Labels : ', Images_labels.shape)
```

```
Size of Images : (39209, 32, 32, 3)  
Size of Labels : (39209,)
```

(39209, 32, 32, 3) - tells us there are 39209 images and of size 32\*32 pixels and last 3 indicates colored images.

```
#display the first image in the training data  
plt.imshow(Images_data[105,:,:], cmap='gray')  
plt.show()
```



## 2. Overlook into data.

```
#Splitting the data into train and test
```

```
train_images, test_images, train_labels, test_labels = train_test_split(Images_data, Images_labels, test_size=0.2, random_state = 42)
```

```
#printing the size of train and test data
```

```
print('train_images size : ', train_images.shape)
```

```
print('train_labels size : ', train_labels.shape)
```

```
print('test_images size : ', test_images.shape)
```

```
print('test_labels size : ', test_labels.shape)
```

```
train_images size : (31367, 32, 32, 3)
```

```
train_labels size : (31367,)
```

```
test_images size : (7842, 32, 32, 3)
```

```
test_labels size : (7842,)
```

```
#change the labels from integer to one-hot encoding
```

```
train_labels = to_categorical(train_labels, 43)
```

```
test_labels = to_categorical(test_labels, 43)
```

### 3. Building a CNN model and finding the accuracy.

```
#Building the model
model = Sequential()
#hidden layer using activation relu
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=train_images.shape[1:]))
model.add(MaxPooling2D(pool_size=(2, 2))) #Adding extra hidden layers
model.add(Dropout(rate=0.25)) #Dropout frequency
#Flattening the model
model.add(Flatten())
model.add(Dense(256, activation='relu')) #more layers
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax')) #out layer
```

Built the sequential model using Conv2D, Maxpooling2D and used activation as relu, softmax.

Included Dropout rate.

### 3. Building a CNN model and finding the accuracy.

```
#Compilation
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
#Fitting or passing the data to the model
history = model.fit(train_images, train_labels, batch_size=256, epochs=5, verbose=1, validation_data=(test_images, test_labels))
```

```
Epoch 1/5
123/123 [=====] - 11s 90ms/step - loss: 7.6071 - accuracy: 0.3674 - val_loss: 0.9950 - val_accuracy:
0.7394
Epoch 2/5
123/123 [=====] - 11s 88ms/step - loss: 1.2157 - accuracy: 0.6929 - val_loss: 0.4634 - val_accuracy:
0.8790
Epoch 3/5
123/123 [=====] - 11s 88ms/step - loss: 0.8279 - accuracy: 0.7975 - val_loss: 0.6577 - val_accuracy:
0.8244
Epoch 4/5
123/123 [=====] - 11s 88ms/step - loss: 0.6847 - accuracy: 0.8438 - val_loss: 0.2356 - val_accuracy:
0.9429
Epoch 5/5
123/123 [=====] - 11s 88ms/step - loss: 0.5522 - accuracy: 0.8731 - val_loss: 0.1837 - val_accuracy:
0.9588
```

```
#Evaluating the model
[test_loss, test_acc] = model.evaluate(test_images, test_labels)
print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc)) #Printing the accuracy
```

```
246/246 [=====] - 1s 5ms/step - loss: 0.1837 - accuracy: 0.9588
Evaluation result on Test Data : Loss = 0.18372522294521332, accuracy = 0.9588115215301514
```

- Model got accuracy of 95% and loss of 18%

### 3. Building a CNN model and finding the accuracy.

- However, we got a good accuracy and loss, will try to add more dense layers and will see the model accuracy and loss again.

```
#Adding more dense layers
model = Sequential()
#hidden layer using activation relu
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=train_images.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu')) #adding more layers
model.add(MaxPooling2D(pool_size=(2, 2))) #adding more layers
model.add(Dropout(rate=0.25))
#Adding more Conv2D, Maxpooling, Dense layers
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu')) #adding more layers
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu')) #adding more layers
model.add(MaxPooling2D(pool_size=(2, 2))) #adding more layers
model.add(Dropout(rate=0.25))
#Flattening the model
model.add(Flatten())
model.add(Dense(256, activation='relu')) #adding more layers
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax')) #out layer
```

### 3. Building a CNN model and finding the accuracy.

```
#Compilation
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
#Fitting or passing the data to the model
history = model.fit(train_images, train_labels, batch_size=256, epochs=5, verbose=1, validation_data=(test_images, test_labels))
```

```
Epoch 1/5
123/123 [=====] - 42s 338ms/step - loss: 3.4409 - accuracy: 0.3198 - val_loss: 0.8909 - val_accuracy: 0.8017
Epoch 2/5
123/123 [=====] - 42s 338ms/step - loss: 0.8440 - accuracy: 0.7730 - val_loss: 0.1911 - val_accuracy: 0.9570
Epoch 3/5
123/123 [=====] - 42s 345ms/step - loss: 0.3785 - accuracy: 0.8992 - val_loss: 0.0989 - val_accuracy: 0.9774
Epoch 4/5
123/123 [=====] - 43s 347ms/step - loss: 0.2468 - accuracy: 0.9387 - val_loss: 0.0941 - val_accuracy: 0.9787
Epoch 5/5
123/123 [=====] - 44s 359ms/step - loss: 0.1968 - accuracy: 0.9502 - val_loss: 0.0485 - val_accuracy: 0.9892
```

```
#Evaluating the model
[test_loss, test_acc] = model.evaluate(test_images, test_labels)
print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc)) #Printing the accuracy
```

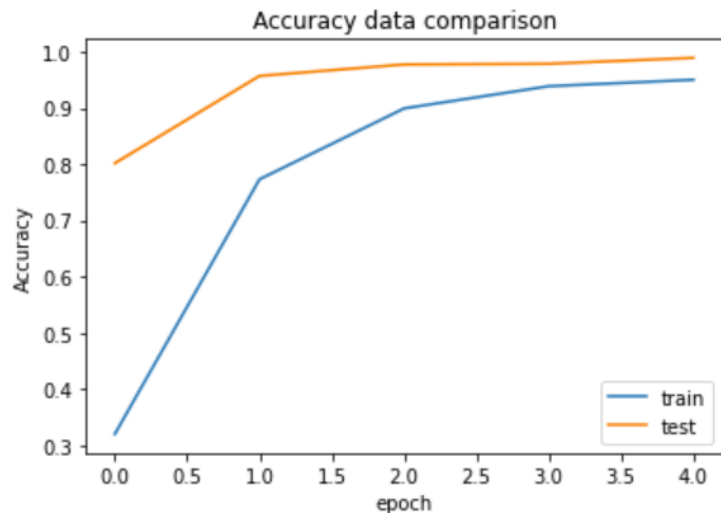
```
246/246 [=====] - 3s 10ms/step - loss: 0.0485 - accuracy: 0.9892
Evaluation result on Test Data : Loss = 0.04851296916604042, accuracy = 0.98916095495224
```

- Accuracy increased for the model from 95 to 98.9% and loss looks like same

# 4. Accuracy and loss plots over train and test data

- Accuracy :

```
# Plotting the Accuracy for both training data and validation data using the history object.  
plt.plot(history.history['accuracy'])  
plt.plot(history.history['val_accuracy'])  
plt.legend(['train', 'test'], loc='lower right')  
plt.title('Accuracy data comparison')  
plt.ylabel('Accuracy')  
plt.xlabel('epoch')  
plt.show()
```

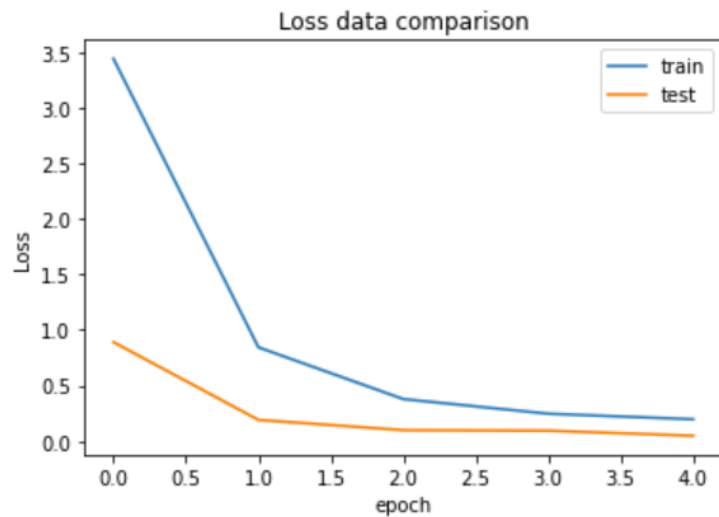




# 4. Accuracy and loss plots over train and test data

- Loss :

```
# Plotting the Loss for both training data and validation data using the history object.  
plt.plot(history.history['loss'])  
plt.plot(history.history['val_loss'])  
plt.title('Loss data comparison')  
plt.legend(['train', 'test'], loc='upper right')  
plt.xlabel('epoch')  
plt.ylabel('Loss')  
plt.show()
```



## 5. Saving the model

- `model.save('traffic_sign_board_detector.h5')`

# Building a GUI by using above sequential model

- Imported libraries for building GUI
- Loaded the classes using CSV file
- Created tkinter by following functions
  - 1.Design
  - 2.Prediction function
  - 3.Display prediction
  - 4.Upload an Image

# 1.Design

```
#https://realpython.com/python-gui-tkinter/  
  
window=tk.Tk()  
window.geometry('800x600')  
window.title('Traffic sign board detector')  
window.configure(background='#466df0')  
label=Label(window,background='#466df0', font=('arial',15,'bold'))  
sign_image = Label(window)
```

- Created a tkinter using below reference  
<https://realpython.com/python-gui-tkinter/>
- Geometrical dimension of 800\*600
- Done basic configuration

## 2. Prediction

*#prediction\_function : By using this function we can pass the uploaded image to model and model will predict the image.*

```
def prediction(file_path):  
    global label_packed  
    sign_board = Image.open(file_path) #Opening the random image from test data  
    sign_board = sign_board.resize((30,30)) #Reshaping the size of image  
    sign_board = numpy.expand_dims(sign_board, axis=0) #Expanding the dimensions  
    sign_board = numpy.array(sign_board)  
    pred = model.predict_classes([sign_board])[0] #Predicting the traffic sign using the model built above  
    sign = classes[pred+1]  
    print(sign) #Printing the traffic sign what model has predicted  
    label.configure(foreground='yellow', text=sign) #adding some color configuration
```

- This function will request the image from test data and will pass the image to the model and model will predict the image.
- Done basic configuration for the predicted answer.

# 3. Display prediction

```
#display_prediction : By using this function we will display the prediction button after uploading the image  
  
def display_prediction(file_path):  
    classification=Button(window,text="Predict", command=lambda: prediction(file_path),padx=10,pady=5) #predicting the imag  
    classification.configure(background='#466df0',foreground='yellow',font=('arial',10,'bold'))  
    classification.place(relx=0.79,rely=0.46)
```

- Display prediction function will display the prediction button after uploading the image and will prediction function is called.
- Done basic configuration to the button

# 4. Upload an Image

```
#Upload_Button : By using this function will request for uploading an image

def upload_sign_board():
    try:
        file_path=filedialog.askopenfilename()    #Opening the file Location where images are stored
        uploaded=Image.open(file_path)
        uploaded.thumbnail(((window.winfo_width()/2.25),(window.winfo_height()/2.25)))
        uploaded=uploaded.resize((180,180))      #Resizing the images which are uploaded
        ima=ImageTk.PhotoImage(uploaded)
        sign_image.configure(image=ima)
        sign_image.image=ima
        label.configure(text='')
        display_prediction(file_path)            #Displaying the prediction
    except:
        pass
```

- By using upload\_sign\_board function , images are uploaded to the tkinter.
- Done resizing of 180\*180 for better display.
- display\_function is called after uploading an image.

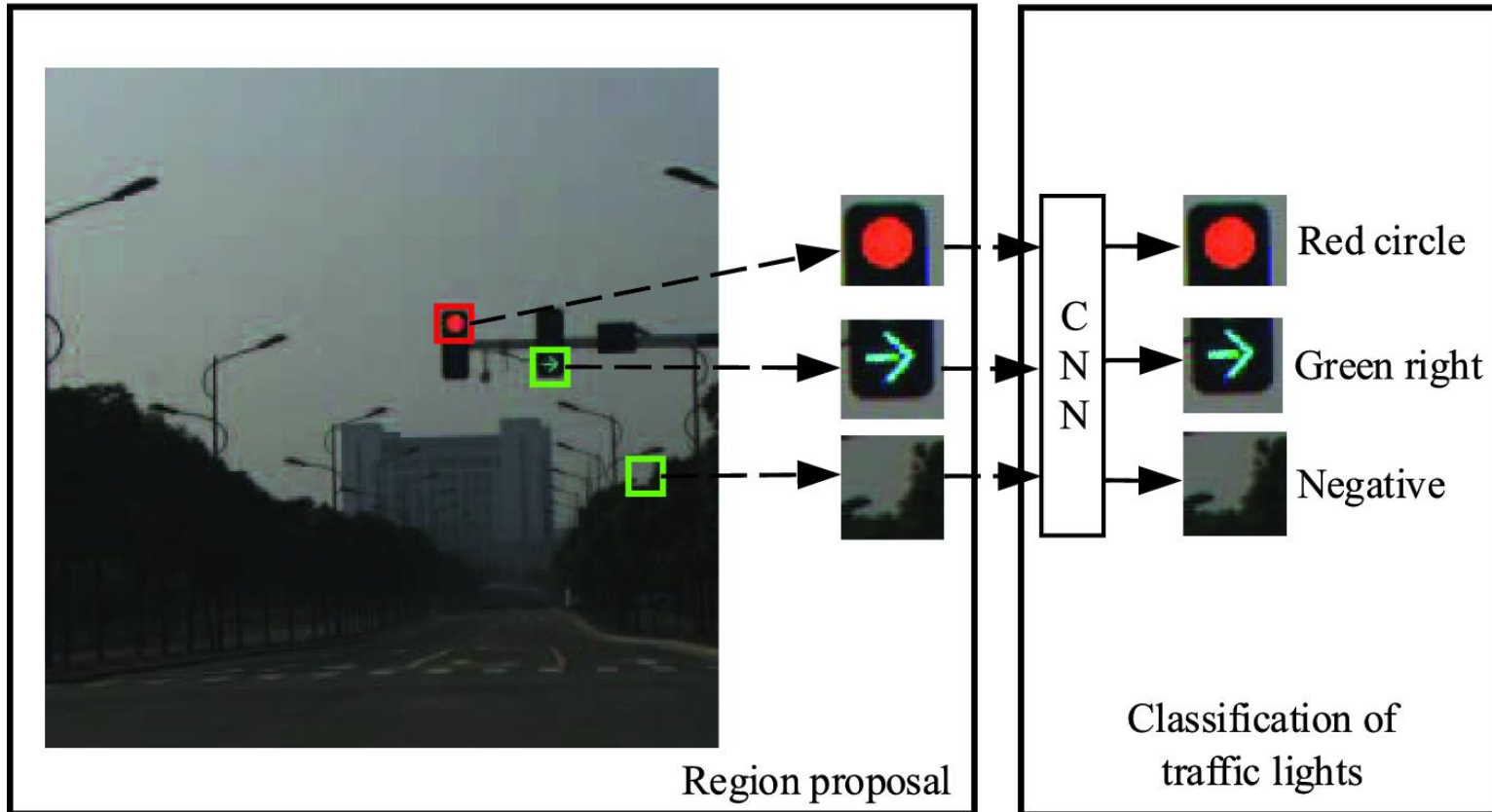
```
#Uploading the image and classifying the type of image
upload=Button(window,text="Upload an image",command=upload_sign_board,padx=10,pady=5) #Button configuration
upload.configure(background='#466df0', foreground='yellow',font=('calibri',10,'bold'))
upload.pack(side=BOTTOM,pady=50)
sign_image.pack(side=BOTTOM,expand=True) #Button Location
label.pack(side=BOTTOM,expand=True)
heading = Label(window, text="Predict the traffic sign",pady=20, font=('calibri',20,'bold'))
heading.configure(background='#466df0', foreground='white')
heading.pack()
window.mainloop()
```

- Upload button configuration was done.
- Flow of the functions:

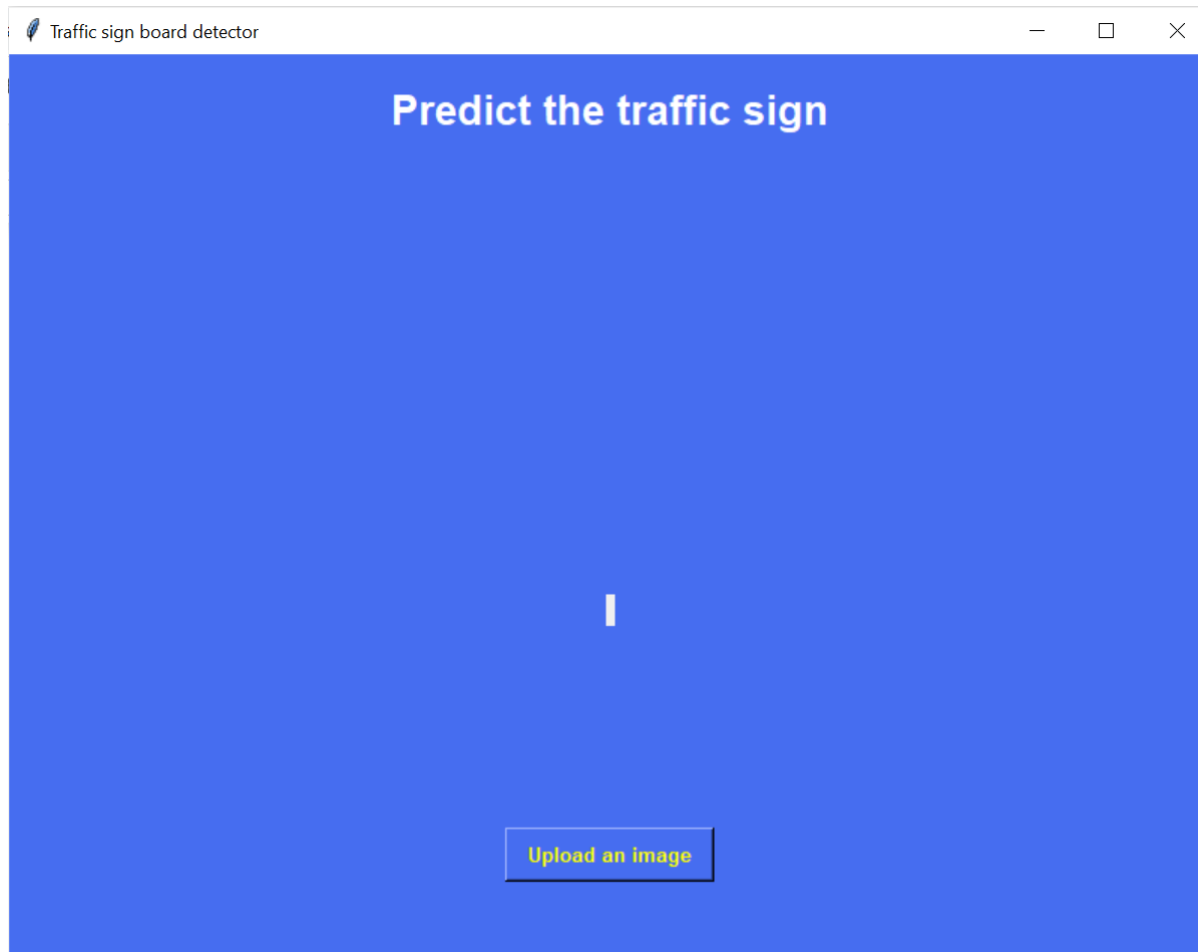
Upload an Image  display prediction  prediction



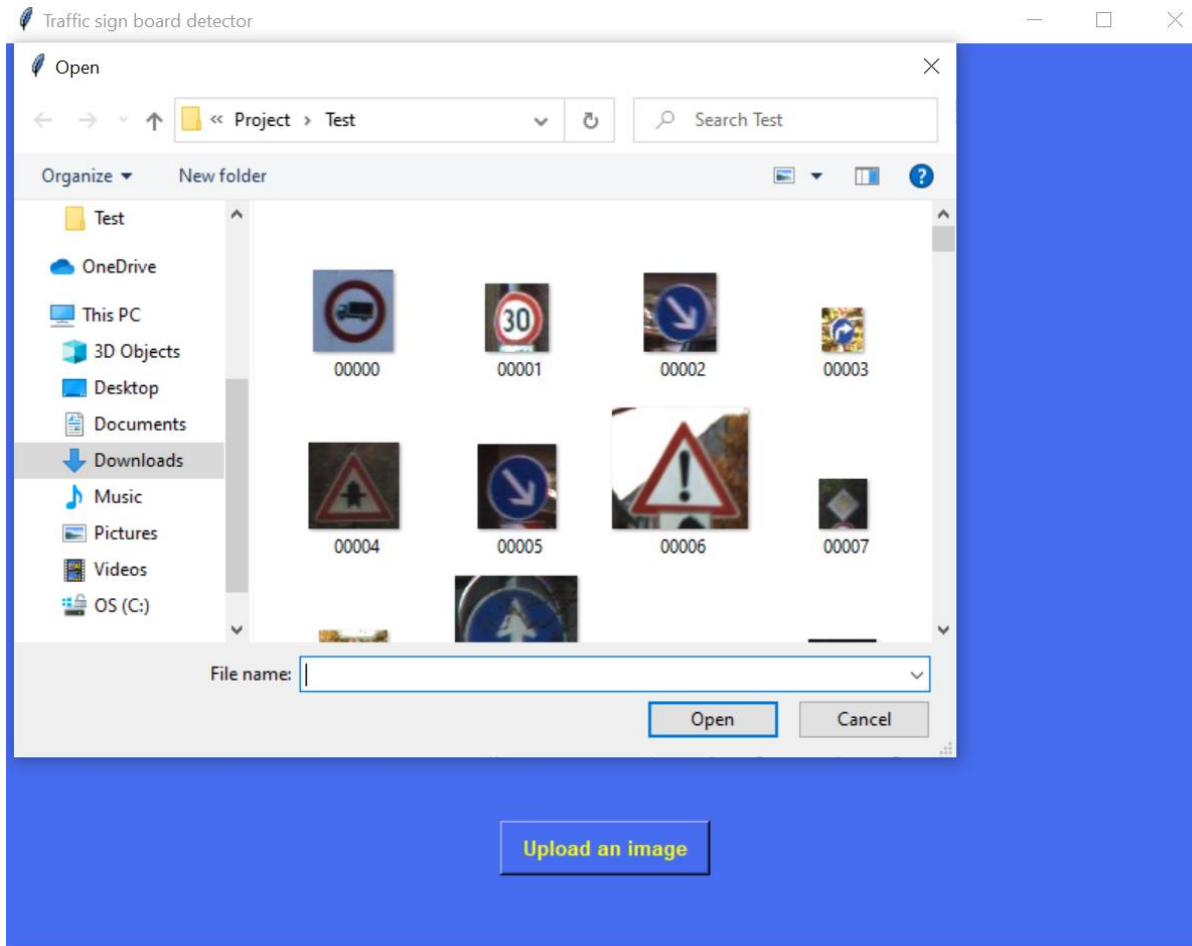
# Results



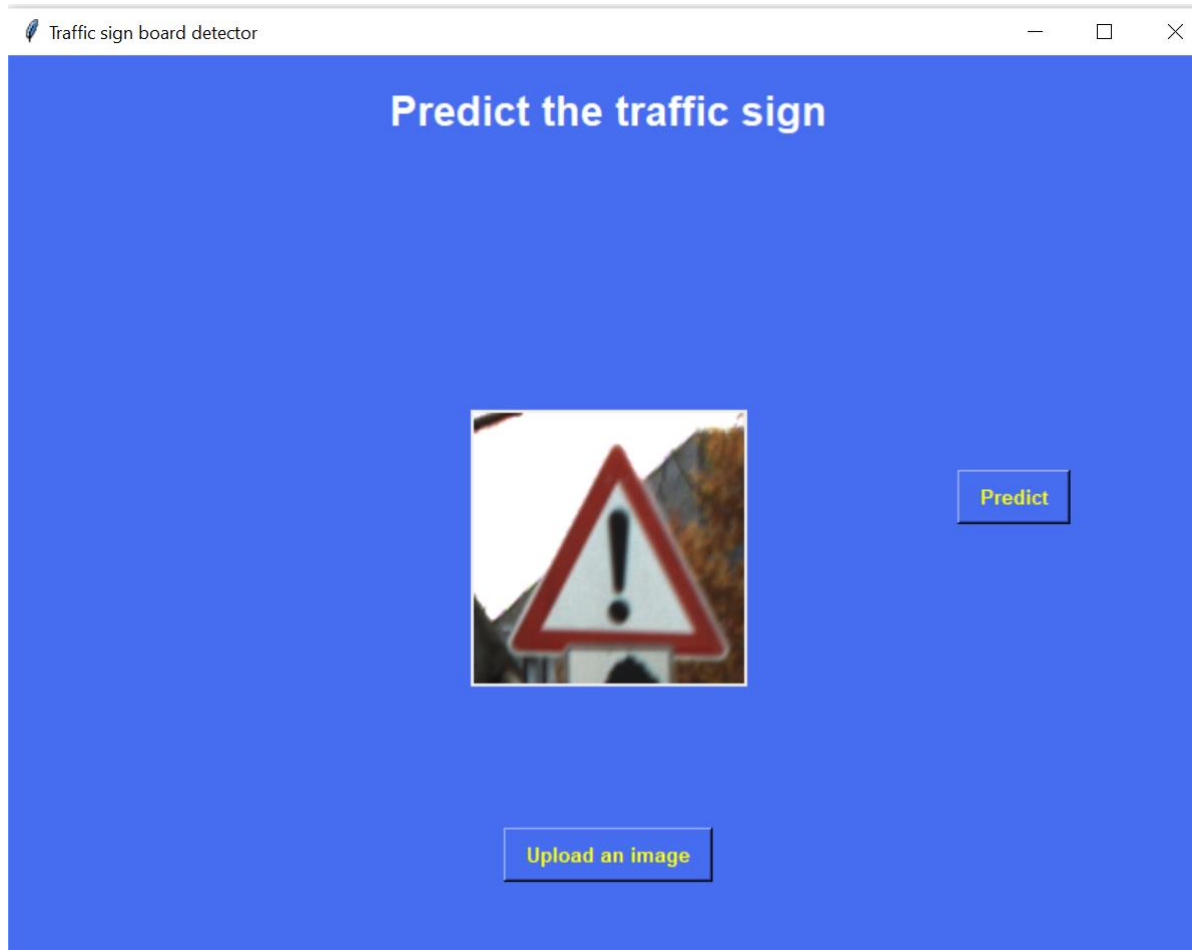
# Results



# Results



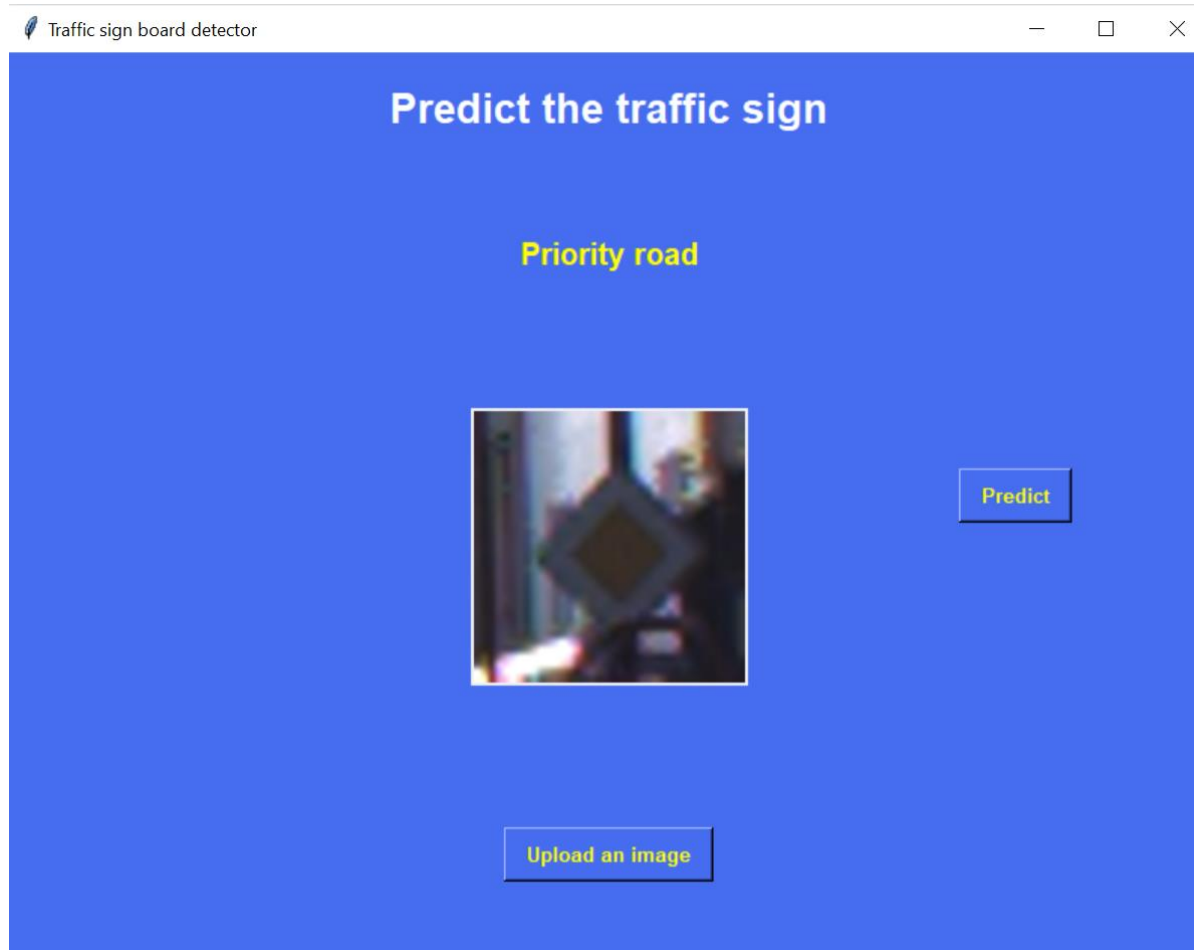
# Results



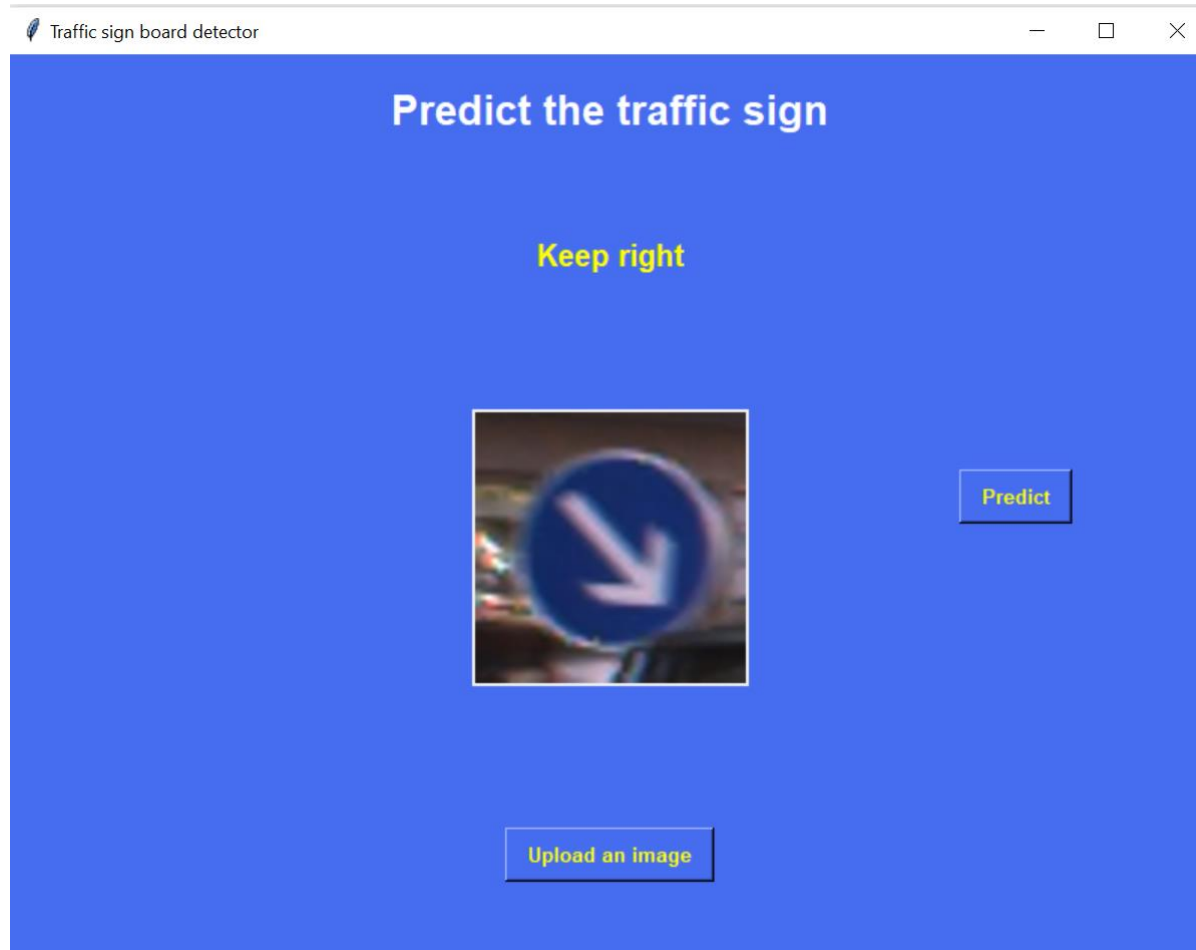
# Results



# Results



# Results



# Image prediction

- Imported the Libraries

```
#Importing the Libraries  
#Used PIL,glob,imageio instead of OpenCV  
from PIL import Image  
import glob  
import imageio  
import matplotlib.pyplot as plt  
from matplotlib.pyplot import imshow  
from keras.models import load_model  
import numpy as np  
from PIL import Image, ImageOps
```

Used PIL for image open, display and rotating the images.

Used glob for assigning specific path for reading the images.

Used load\_model for reloading the model again.



# Image prediction

- Created a function  
def getClassName

By using this function, we will pass the predicted label from the model to the function and will get back the respective class name from that.

Some ClassNo be like :

```
classNo == 2:  
    return 'Speed Limit 50 km/h'  
classNo == 3:  
    return 'Speed Limit 60 km/h'
```

# Image prediction

Passed the video to the ffmpeg to get frames

ffmpeg:

This is software, where libraries and logics will be there for processing the videos to frames or converting frames to videos.

Recorded a video of traffic signs and passed that video to ffmpeg and followed below commands:

1. Opened Anaconda Command Prompt and activated tensorflow environment.

```
(base) C:\Users\suren>conda activate tf
```

2. Changed the default directory to the video path.

```
(tf) C:\Users\suren>cd C:\Users\suren\Pictures\Camera Roll\Traffi_Signs
```

3. Passed video to the ffmpeg command, which will convert the video into frames , in this we passed frame rate as 1 sec which will capture one frame per second.

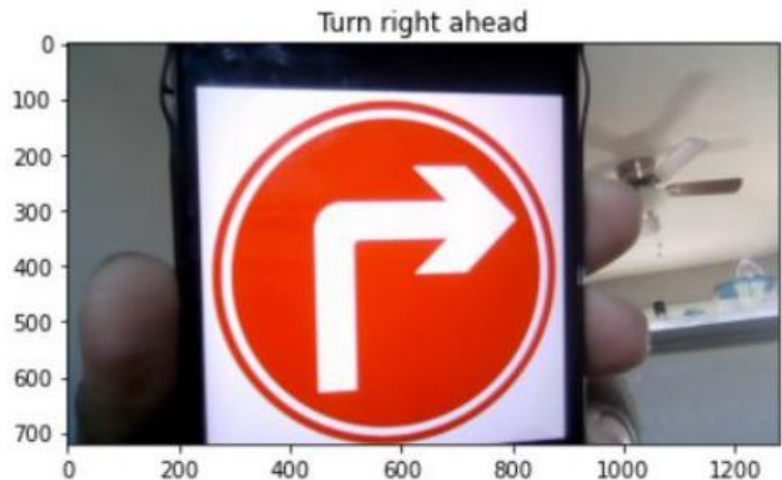
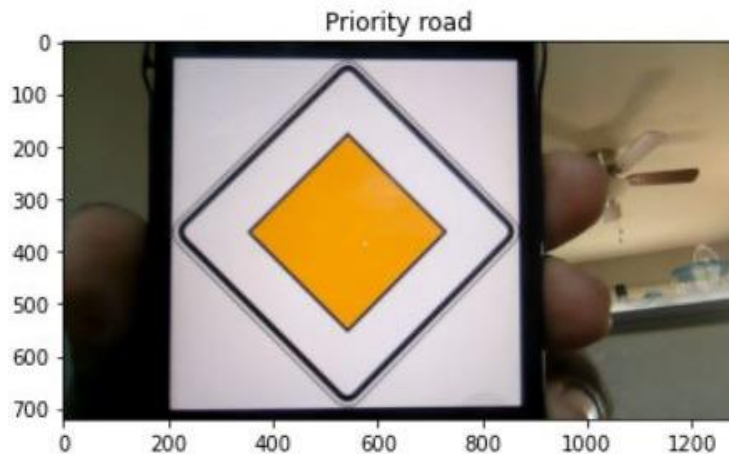
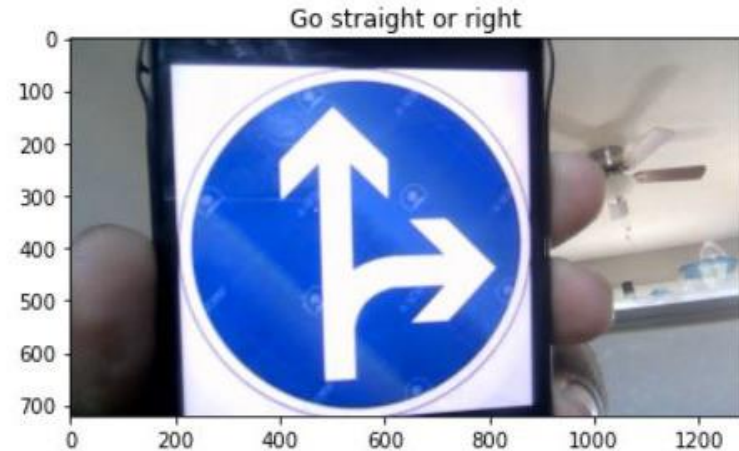
```
(tf) C:\Users\suren\Pictures\Camera Roll\Traffi_Signs>ffmpeg -i video.mp4 -vf fps=1 out%d.png
```

# Image prediction

Reloaded the model again which we built last time for image prediction.

```
for image_path in glob.glob(r'C:\Users\suren\Pictures\Camera Roll\Traffi_Signs\*.png'):
    image1 = Image.open(image_path) #opening the every frame in the given path
    #Image Enhancement
    img = image1.resize((32, 32)) #resizing the images inorder to fit for model
    img = ImageOps.grayscale(img) #done grayscale
    img = ImageOps.equalize(img, mask = None) #done equalize before passing to model
    im = np.array(img) #converted arrays to image
    im = im/255 #Normalising the values between 0 and 1 instead of 0 and 255
    im = im.reshape(1, 32, 32, 1) #Reshaping the image of 1024 pixel to 32*32 1 dim
    #Image prediction
    predictions = model.predict(im) #Passing the image to the model
    classIndex = model.predict_classes(im) #predicting the Class Index
    #Recalling the pre-defined function
    ClassName = getClassNames(classIndex)
    #Image plotting
    plt.figure()
    plt.imshow(image1)
    plt.title(ClassName) #Title of the image
```

# Image prediction - Output



# Image prediction – with OpenCV

Imported Open-source computer vision Library

- By using this we can do resize, reshape, converting images to grayscale, image equalization.
- Defined some resolutions and threshold values.

```
import numpy as np
import cv2
```

```
#Camera window details
font = cv2.FONT_HERSHEY_SIMPLEX #Loading a font
cap = cv2.VideoCapture(r'C:\Users\suren\Pictures\Camera Roll\Traffi_Signs\videoo.mp4')
cap.set(3, 640) #resolution
cap.set(4, 480) #resolution
cap.set(10, 0.75) #threshold value
frame_threshold = 0.75
```

# Image prediction – with OpenCV

- Defined three functions
  - Grayscale
  - Equalize
  - Pre-Processing

```
def grayscale(frame): #Function for converting the image to grayscale  
    frame = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)  
    return frame  
def equalize(frame): #Function for image equalization  
    frame = cv2.equalizeHist(frame)  
    return frame  
def preprocessing(frame): #Function for preprocessing  
    frame = grayscale(frame)  
    frame = equalize(frame)  
    frame = frame/255  
    return frame
```



# Image prediction – with OpenCV

```
while(cap.isOpened()):  
    # Image reading from video  
    success, frameOriginal = cap.read() #Checking for frames  
    frame = np.asarray(frameOriginal) #Converting to numpy  
    frame = cv2.resize(frame, (32, 32)) #Resizing  
    frame = preprocessing(frame) #Processing all the predclared functions on the image  
    cv2.imshow("Processed Image", frame)  
    frame = frame.reshape(1, 32, 32, 1) #Image reshaping  
    #Printing text on the video  
    cv2.putText(frameOriginal, "CLASS: ", (20, 35), font, 0.75, (0, 0, 255), 2, cv2.LINE_AA) #printing CLASS text  
    cv2.putText(frameOriginal, "PROBABILITY: ", (20, 75), font, 0.75, (0, 0, 255), 2, cv2.LINE_AA) #Printing probability text  
    # Model Prediction  
    predictions = model.predict(frame) #Passing image to the model.  
    classIndex = model.predict_classes(frame) #predicting every frame in the model.  
    probabilityValue = np.amax(predictions) #by based on the prediction value  
    if probabilityValue > frame_threshold: #If the threshold is greater than the given threshold  
        #Printing Class and Class Name predictions  
        cv2.putText(frameOriginal, str(classIndex)+" "+str(getClassName(classIndex)), (120, 35), font, 0.75, (0, 0, 255), 2, cv2.LINE_AA)  
        cv2.putText(frameOriginal, str(round(probabilityValue*100,2))+"%", (180, 75), font, 0.75, (0, 0, 255), 2, cv2.LINE_AA)  
    cv2.imshow("Result", frameOriginal) #Re-displaying the image again.  
  
    if cv2.waitKey(1) and 0xFF == ord('q'): #Break  
        break
```

# Image prediction – with OpenCV

Predicted the frames in the video with probabilities and displayed them on video.

Input Video :

<https://drive.google.com/file/d/1nhNav9Jxb94P7FvTpcssORKYXSJ6ioXz/view>

Output Video :

<https://drive.google.com/file/d/18OC1aweRrJ7kha6eYTE4WErzCjAF1TkP/view>



# Image prediction – with OpenCV



# Learnings

- About Models
- GUI
- PIL
- Processing video without open CV

# Challenges

- Building GUI
- Processing video without open CV

# Technologies Used

- Python
- Deep Learning

# Future Scope

- By adding more essence to this code with Open cv like getting bounding boxes and making the predictions with real quick answers , we can deploy this some testing vehicles for getting feedback data.
- By using that feedback data, we can train the model again with negative answers. So, that we can increase the accuracy of model.
- Further it can be implemented in driverless cars or even in normal cars for better travel.

# References

- Training data : <https://www.kaggle.com/shivamsinghal1012/traffic-sign-data-set>
- <https://realpython.com/python-gui-tkinter/>

Thank you