A PROJECT REPORT
ON

TRAFFIC SIGN RECOGNITION USING NEURAL NETWORKS

Submitted to
KIIT Deemed to be University
In Partial Fulfillment of the Requirement for the Award
BACHELOR'S DEGREE IN COMPUTER SCIENCE & ENGINEERING

By,
ANIRBAN BOSE (1728191)
DEEPANSHU JAYSWAL (1728263)
DIPENDRA UPADHYAY (1728258)

UNDER THE GUIDANCE OF
PROF. MANOJ KUMAR MISRA

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
BHUBANESWAR, ODISHA – 751024 APRIL 2020
KIIT - Deemed to be University
School of Computer Science Engineering
Bhubaneswar, ODISHA- 751024

CERTIFICATE

This is to certify that the project entitled
"**TRAFFIC SIGN RECOGNITION USING NEURAL NETWORKS**"

Submitted by:
ANIRBAN BOSE - 1728191
DIPENDRA UPADHYA -1728258
DEEPANSHU JAYSWAL -1728263

It is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & System Engineering) at KIIT Deemed to be University, Bhubaneswar. This work is done during the year 2018-2019, under my guidance. Date: 07/06/2020

Prof. Manoj Kumar Misra,

Project Guide

ACKNOWLEDGEMENTS

We are profoundly grateful to **Prof.MANOJ MISRA** for his expert guidance and encouragement throughout to see that this project rights its target since its commencement to its completion.

I again thank him for providing his precious time to me in the completion of my project. I shall remain ever grateful to him, who unselfishly inculcated in me the spirit of hard work and helped me in each and every aspect of this project. He had always provided me with new tricks and techniques to me for solving the project's problems.

Anirban Bose 1728191
Deepanshu Jayswal 1728263
Dipendra Upadhya 1728268

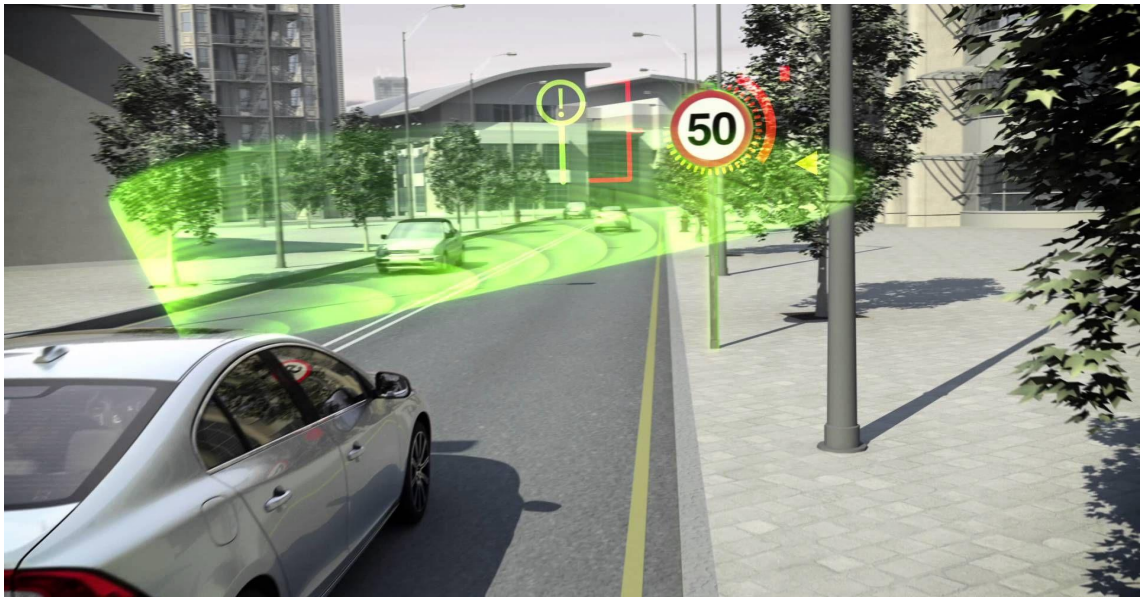
ABSTRACT

The objective of this project is to work on the development of an algorithm for the automatic recognition of traffic signs. Two major problems exist in the process of detection and recognition of traffic signals. Road signs are frequently occluded partially by other vehicles and many objects are present in traffic scenes which make the sign detection hard and pedestrians, other vehicles, buildings and billboards may confuse the detection system by patterns similar to that of road signs. Also, information from traffic scene images is affected by varying illumination caused by weather conditions, time (day-night) and shadowing. This method detects the location of the sign in the image, based on its geometrical characteristics and recognizes it using colour information. For the road sign recognition subsystem, a convolutional neural network (CNN) is adopted to classify traffic signs for candidate regions. Experimental results show that our approach can obtain the desired results effectively.

Keywords -- Traffic sign recognition, shape detection and Region of interest.

INTRODUCTION

In recent years many algorithms for the traffic sign detection and classification have been introduced. Extensive research is being made by major car manufacturing companies in collaboration with Universities and other institutes on real-time and automatic recognition of traffic signs so that it can be a part of the so-called “Driver Support Systems”. Traffic sign recognition (TSR) can be considered part of the bigger problem of autonomous vehicles. An autonomous vehicle system relies on vision-based recognition of the surrounding area in order to make driving decisions. This vision-based recognition system may function as the feedback provider for control of steering wheel, accelerator, brake, etc. It may recognize road and lane to allow control system follow the course of own vehicle, detect obstacles on the road till control system avoids them, detect the passing vehicles (e.g. by side or back cameras) to notify the control system about probable hazards and detect and interpret the traffic signs to provide feedback for safe driving. It detects signs in pics using a camera or webcam and the algorithm made using neural networks will recognise the shown sign and stores the evidence in the device’s hard drive. We have used python as our programming language and have used various packages of python.



OBJECTIVE

Our aim is to design an image processing algorithm that can determine the type of traffic sign that is displayed as an image and is robust to different real-life conditions such as poor lighting, obstructions or the sign being far away. Advanced control systems interpret sensory information to identify appropriate navigation paths as well as obstacles and relevant signage. This project will help autonomous driving vehicles (AI enabled) to drive safely on the road, with fewer hindrances and very few chances of getting into an accident by identifying any traffic signs and obstructions in its way.

Our motive is to identify “traffic signs” for the self-driving automated cars and further enhancing the detection techniques that currently prevail. Traffic signs provide important information for drivers about road condition and hazards. Their discriminating shape and colours make them easily recognizable by humans. Besides the application of Traffic sign recognition (TSR) in autonomous vehicles, it can also serve as an assistant driver (e.g. when combined with speedometer output) to notify the driver about approaching a traffic sign (e.g. even before the driver sees it) or his risky behaviour (like driving above the speed limit).

DEFINITIONS & OVERVIEW

PROBLEM STATEMENT

It is very critical to teach a machine to identify a traffic sign and to interpret it correctly. In this project, research work has been done to gain an insight about how to train the data set to teach an automated automobile and get accurate results using and produce a reliable software on our own in this field.

DATA SET

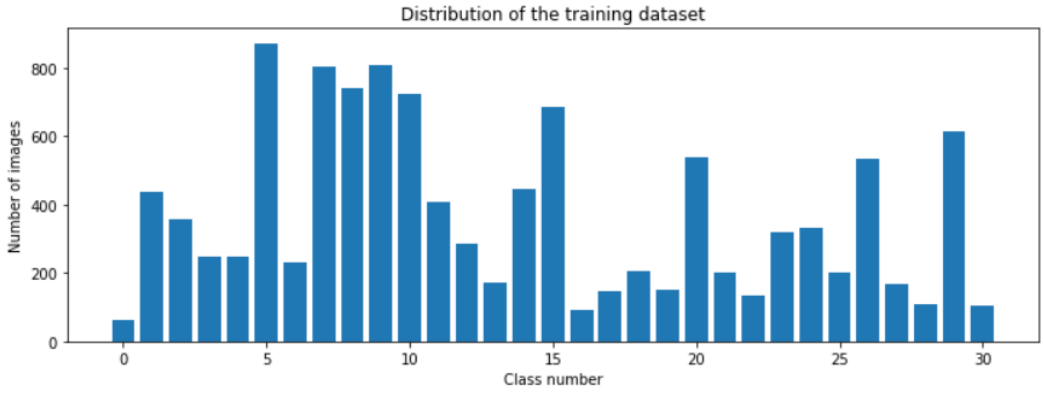
Data set is no of images

Train(11388, 32, 32, 3) (11388)

Here we take 11388 images to train our machine learning algorithm

Validation(2848, 32, 32, 3) (2848)

Here we take 2848 images to train and validate our machine learning algorithm
Data Shape (31, 2)
There are 31 types of data/Traffic signs to train



This is the distribution graph of the training set for Traffic Sign Recognition

SOFTWARE AND HARDWARE SPECIFICATION

Software Used:

Operating System: Windows 10 Home

Language: Python

IDE: Jupyter Notebook

Database: Image dataset

Database location: Local drive(HDD)

Packages:

I. Numpy

II. Open Cv

III. Keras

IV. Pandas

V. matplotlib

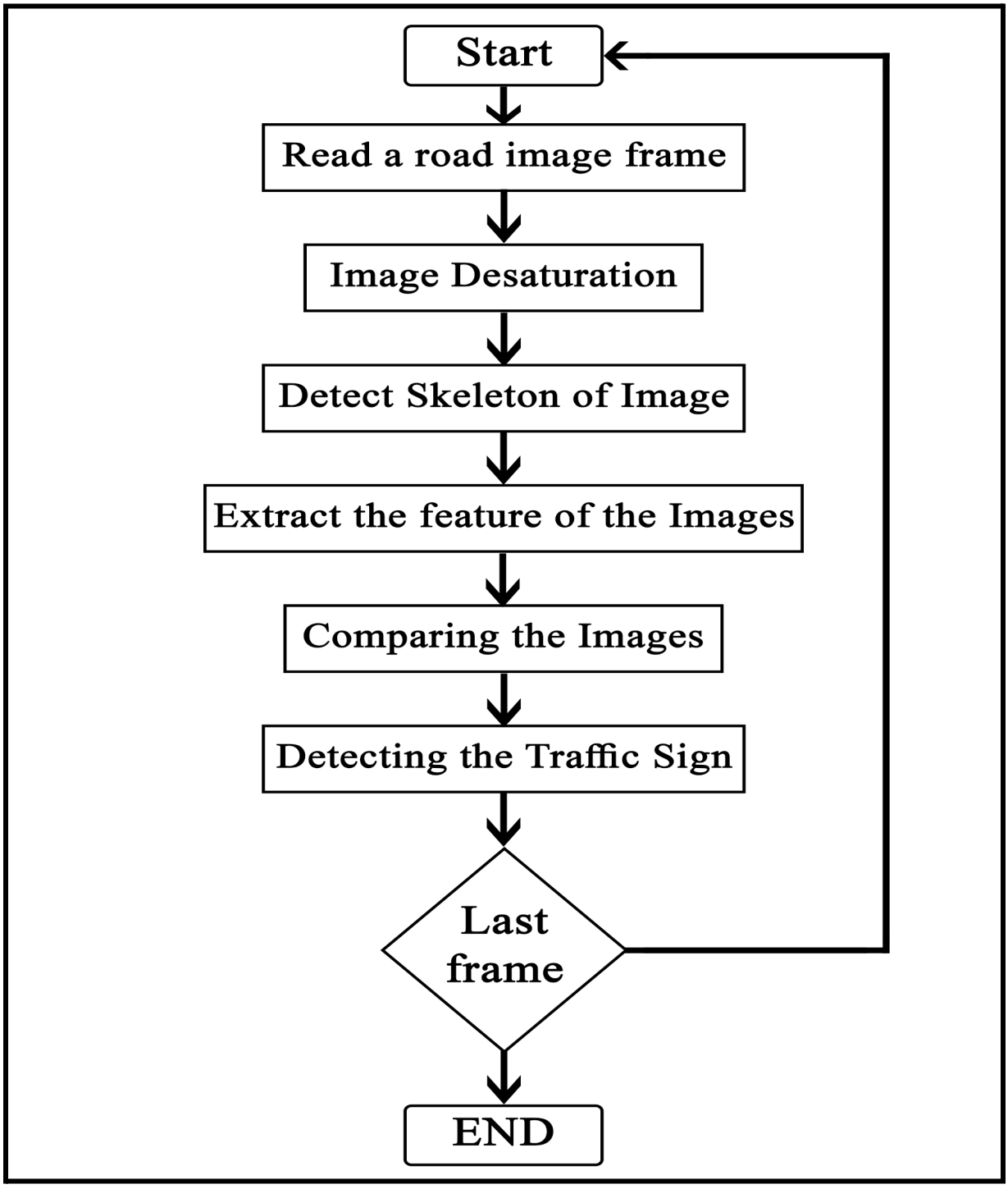
Hardware Used:

Processor: Intel i5

Primary Memory(RAM): 8Gb

Secondary Memory(HDD): 1TB

METHODOLOGY



Traffic Sign Flow Chart

Read a road image frame: Read the image(frame) from the camera.

Image Desaturation: It converts the coloured image(frame) to black and white frames for fast calculation of the algorithm

Skeleton of Image: Using shape-based detection, OpenCV has the ability to detect specific circular objects in a given gravy scale image. It gives the total number of detected circles with three important parameters. i.e. centre coordinates (x, y) and radius.
At this stage, we have detected the information on red circles.
• Cropped and Extract features: Using Rect and mask extract features from the circle.
• Separation: Contour method is used to separate the digits. i.e. “50” -> '5' , '0'

Extract the Feature of the images: Using Keras package we have extracted the feature of the image i.e the various distinguishing characteristics of the sign image is captured whether the shown sign is of the speed limit, stiff end, school ahead etc.

Comparing the images:After extracting the features of the sign image the algorithm now compares the two images i.e the image detected in the camera with the images that the machine is being trained.

Detecting the traffic sign: If after comparing the two signs matches then the traffic sign’s name is displayed on the screen.

Last Frame: If it does not detect any traffic sign then it again checks the new frame.

IMPLEMENTATION

The implementation of the project has been done in python programming language for predicting the Traffic Sign. Also, we have used various python packages that are explained below-

Pandas: pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
Here we used Pandas to import different types of traffic sign for labelled data and then creating a numpy array of that.

Numpy: NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object and tools for working with these arrays.
Here we used Numpy for creating an array of images and class number(Traffic Sign Numbers).

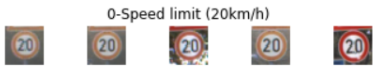
Sklearn: It is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms.
We have used sklearn for creating training, validation, test data and in creating batches

Keras: Keras is an open source neural network library written in Python. It is capable of running on top of Tensorflow, Microsoft cognitive toolkit, R etc. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.
Using Keras(tensorflow) we trained our model. We used 11 hidden layers for the better optimization of the software.

Pickle: The pickle module implements binary protocols for serializing and de-serializing a Python object structure.
We are saving the data model(trained data) in a pickle file so we can retrieve the data from here to test.

Matplotlib: Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy.
We have used this to generate graphs as graphs are more easily understandable.

```
Using TensorFlow backend.

D:\S T U D Y\P R O J E C T S\Traffic Sign Recognition\Traffic Sign Rec 1\TrafficSignLabels.csv
Total Classes Detected: 31
Importing Classes.....
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Data Shapes
Train(11388, 32, 32, 3) (11388,)
Validation(2848, 32, 32, 3) (2848,)
Test(3560, 32, 32, 3) (3560,)
data shape  (31, 2) <class 'pandas.core.frame.DataFrame'>
[64, 436, 356, 246, 248, 873, 231, 804, 741, 807, 725, 409, 286, 173, 448, 685, 91, 148, 207, 153, 539, 200, 132, 321, 334,
201, 536, 166, 108, 616, 104]
```



Dataset (image class)

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 60)	1560
conv2d_2 (Conv2D)	(None, 24, 24, 60)	90060
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 60)	0
conv2d_3 (Conv2D)	(None, 10, 10, 30)	16230
conv2d_4 (Conv2D)	(None, 8, 8, 30)	8130
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 30)	0
dropout_1 (Dropout)	(None, 4, 4, 30)	0
flatten_1 (Flatten)	(None, 480)	0
dense_1 (Dense)	(None, 500)	240500
dropout_2 (Dropout)	(None, 500)	0
dense_2 (Dense)	(None, 31)	15531

Total params: 372,011
Trainable params: 372,011
Non-trainable params: 0

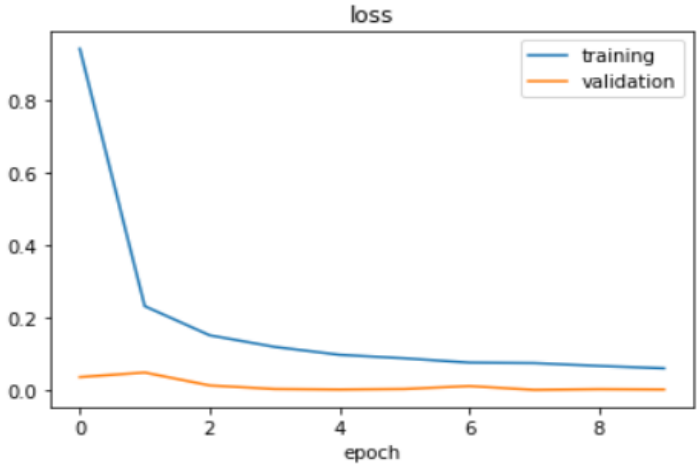
There are a total of 372011 trainable parameters

```
2000/2000 [==============================] - 723s 362ms/step - loss: 0.9438 - acc: 0.7112 - val_loss: 0.0378 - val_acc: 0.98
56
Epoch 2/10
2000/2000 [==============================] - 738s 369ms/step - loss: 0.2337 - acc: 0.9246 - val_loss: 0.0504 - val_acc: 0.98
21
Epoch 3/10
2000/2000 [==============================] - 773s 386ms/step - loss: 0.1532 - acc: 0.9515 - val_loss: 0.0148 - val_acc: 0.99
30
Epoch 4/10
2000/2000 [==============================] - 717s 359ms/step - loss: 0.1212 - acc: 0.9631 - val_loss: 0.0049 - val_acc: 0.99
93
Epoch 5/10
2000/2000 [==============================] - 656s 328ms/step - loss: 0.0993 - acc: 0.9694 - val_loss: 0.0035 - val_acc: 0.99
86
Epoch 6/10
2000/2000 [==============================] - 653s 326ms/step - loss: 0.0897 - acc: 0.9721 - val_loss: 0.0050 - val_acc: 0.99
89
Epoch 7/10
2000/2000 [==============================] - 656s 328ms/step - loss: 0.0778 - acc: 0.9763 - val_loss: 0.0131 - val_acc: 0.99
54
Epoch 8/10
2000/2000 [==============================] - 663s 332ms/step - loss: 0.0765 - acc: 0.9766 - val_loss: 0.0026 - val_acc: 0.99
96
Epoch 9/10
2000/2000 [==============================] - 661s 330ms/step - loss: 0.0686 - acc: 0.9794 - val_loss: 0.0044 - val_acc: 0.99
86
Epoch 10/10
2000/2000 [==============================] - 665s 333ms/step - loss: 0.0621 - acc: 0.9814 - val_loss: 0.0036 - val_acc: 0.99
86
```

Trained the software by splitting the training data into 10 epochs, the number of passes of the entire training dataset the algorithm has completed and by grouping the dataset into batches.

```
Test Score: 0.0030610846584126746
Test Accuracy: 0.9991573033707866
```

We got an accuracy of 99.91%




```
Model saved
Successfully trained
```

This is the loss graph showing how the software is learning and improving its accuracy in the next iteration.

CODE

TrafficSignMain.py

```
import matplotlib.pyplot as plt
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.utils.np_utils import to_categorical
from keras.layers import Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
import cv2
from sklearn.model_selection import train_test_split
import pickle
import os
import pandas as pd
import random
from keras.preprocessing.image import ImageDataGenerator



################## Parameters ####################


path = "D:\S T U D Y\P R O J E C T S\Traffic Sign Recognition\Traffic Sign Rec
    1\TrafficSignData\myData" # folder with all the class folders


labelFile = 'D:\S T U D Y\P R O J E C T S\Traffic Sign Recognition\Traffic Sign Rec
    1\TrafficSignLabels.csv' # file with all names of classes
print(labelFile)

batch_size_val=50  # how many to process together


steps_per_epoch_val=2000
epochs_val=10
imageDimesions = (32,32,3)
```

```
testRatio = 0.2    # if 1000 images split will 200 for testing

validationRatio = 0.2 # if 1000 images 20% of remaining 800 will be 160 for validation

################################ Importing of the Images

count = 0
images = []
classNo = []
myList = os.listdir(path)
print("Total Classes Detected:",len(myList))
noOfClasses=len(myList)
print("Importing Classes.....")
for x in range (0,len(myList)):
  myPicList = os.listdir(path+"/"+str(count))
  for y in myPicList:
     curImg = cv2.imread(path+"/"+str(count)+"/"+y)
     images.append(curImg)
     classNo.append(count)
  print(count, end =" ")
  count +=1
print(" ")
images = np.array(images)
classNo = np.array(classNo)



################################ Split Data

X_train, X_test, y_train, y_test = train_test_split(images, classNo, test_size=testRatio)
X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train,
   test_size=validationRatio)

# X_train = ARRAY OF IMAGES TO TRAIN
```

```
# y_train = CORRESPONDING CLASS ID


################################## TO CHECK IF NUMBER OF IMAGES
   MATCHES TO NUMBER OF LABELS FOR EACH DATA SET


print("Data Shapes")
print("Train",end = "");print(X_train.shape,y_train.shape)
print("Validation",end = "");print(X_validation.shape,y_validation.shape)
print("Test",end = "");print(X_test.shape,y_test.shape)
assert(X_train.shape[0]==y_train.shape[0]), "The number of images in not equal to the
   number of lables in training set"
assert(X_validation.shape[0]==y_validation.shape[0]), "The number of images in not
   equal to the number of lables in validation set"
assert(X_test.shape[0]==y_test.shape[0]), "The number of images in not equal to the
   number of lables in test set"
assert(X_train.shape[1:]==(imageDimesions)),"  The dimesions of the Training images
   are wrong "
assert(X_validation.shape[1:]==(imageDimesions)),"  The dimesionas of the Validation
   images are wrong "
assert(X_test.shape[1:]==(imageDimesions)),"  The dimesionas of the Test images are
   wrong"



################################## READ CSV FILE


data = pd.read_csv(labelFile)
print("data shape ",data.shape,type(data))


################################## DISPLAY SOME SAMPLES IMAGES  OF ALL
   THE CLASSES


num_of_samples = []
cols = 5
```

```
num_classes = noOfClasses
fig, axs = plt.subplots(nrows=num_classes, ncols=cols, figsize=(5, 300))
fig.tight_layout()
for i in range(cols):
  for j,row in data.iterrows():
     x_selected = X_train[y_train == j]
     axs[j][i].imshow(x_selected[random.randint(0, len(x_selected)- 1), :, :],
   cmap=plt.get_cmap("gray"))
     axs[j][i].axis("off")
     if i == 2:
        axs[j][i].set_title(str(j)+ "-"+row["Name"])
        num_of_samples.append(len(x_selected))



################################## DISPLAY A BAR CHART SHOWING NO OF
  SAMPLES FOR EACH CATEGORY


print(num_of_samples)
plt.figure(figsize=(12, 4))
plt.bar(range(0, num_classes), num_of_samples)
plt.title("Distribution of the training dataset")
plt.xlabel("Class number")
plt.ylabel("Number of images")
plt.show()


################################## PREPROCESSING THE IMAGES



def grayscale(img):
  img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  return img
def equalize(img):
  img =cv2.equalizeHist(img)
  return img
def preprocessing(img):
  img = grayscale(img)     # CONVERT TO GRAYSCALE
```

```
    img = equalize(img)      # STANDARDIZE THE LIGHTING IN AN IMAGE

    img = img/255            # TO NORMALIZE VALUES BETWEEN 0 AND 1 INSTEAD
     OF 0 TO 255

    return img



X_train=np.array(list(map(preprocessing,X_train)))  # TO IRETATE AND
    PREPROCESS ALL IMAGES


X_validation=np.array(list(map(preprocessing,X_validation)))
X_test=np.array(list(map(preprocessing,X_test)))
cv2.imshow("GrayScale Images",X_train[random.randint(0,len(X_train)-1)]) # TO
    CHECK IF THE TRAINING IS DONE PROPERLY


################################## ADD A DEPTH OF 1


X_train=X_train.reshape(X_train.shape[0],X_train.shape[1],X_train.shape[2],1)
X_validation=X_validation.reshape(X_validation.shape[0],X_validation.shape[1],X_vali
    dation.shape[2],1)
X_test=X_test.reshape(X_test.shape[0],X_test.shape[1],X_test.shape[2],1)



################################## AUGMENTATION OF IMAGES: TO MAKE IT
    MORE GENERIC


dataGen= ImageDataGenerator(width_shift_range=0.1,   # 0.1 = 10%     IF MORE
    THAN 1 E.G 10 THEN IT REFERS TO NO. OF  PIXELS EG 10 PIXELS
```

```
                    height_shift_range=0.1,
                    zoom_range=0.2,  # 0.2 MEANS CAN GO FROM 0.8 TO 1.2

                    shear_range=0.1,  # MAGNITUDE OF SHEAR ANGLE

                    rotation_range=10)  # DEGREES


dataGen.fit(X_train)
batches= dataGen.flow(X_train,y_train,batch_size=20)  # REQUESTING DATA
   GENERATOR TO GENERATE IMAGES  BATCH SIZE = NO. OF IMAGES
   CREATED EACH TIME ITS CALLED


X_batch,y_batch = next(batches)



# TO SHOW AUGMENTED IMAGE SAMPLES


fig,axs=plt.subplots(1,15,figsize=(20,5))
fig.tight_layout()



for i in range(15):
  axs[i].imshow(X_batch[i].reshape(imageDimesions[0],imageDimesions[1]))
  axs[i].axis('off')
plt.show()



y_train = to_categorical(y_train,noOfClasses)
y_validation = to_categorical(y_validation,noOfClasses)
y_test = to_categorical(y_test,noOfClasses)
```

```
############################### CONVOLUTION NEURAL NETWORK
    MODEL


def myModel():
    no_Of_Filters=60
    size_of_Filter=(5,5) # THIS IS THE KERNEL THAT MOVE AROUND THE
     IMAGE TO GET THE FEATURES.


                    # THIS WOULD REMOVE 2 PIXELS FROM EACH BORDER WHEN
     USING 32 32 IMAGE


    size_of_Filter2=(3,3)
    size_of_pool=(2,2)  # SCALE DOWN ALL FEATURE MAP TO GERNALIZE
     MORE, TO REDUCE OVERFITTING


    no_Of_Nodes = 500   # NO. OF NODES IN HIDDEN LAYERS


    model= Sequential()

     model.add((Conv2D(no_Of_Filters,size_of_Filter,input_shape=(imageDimesions[0],i
     mageDimesions[1],1),activation='relu')))  # ADDING MORE CONVOLUTION
     LAYERS = LESS FEATURES BUT CAN CAUSE ACCURACY TO INCREASE


    model.add((Conv2D(no_Of_Filters, size_of_Filter, activation='relu')))
    model.add(MaxPooling2D(pool_size=size_of_pool)) # DOES NOT EFFECT THE
     DEPTH/NO OF FILTERS



    model.add((Conv2D(no_Of_Filters//2, size_of_Filter2,activation='relu')))
    model.add((Conv2D(no_Of_Filters // 2, size_of_Filter2, activation='relu')))
    model.add(MaxPooling2D(pool_size=size_of_pool))
    model.add(Dropout(0.5))
```

```
  model.add(Flatten())
  model.add(Dense(no_Of_Nodes,activation='relu'))
  model.add(Dropout(0.5)) # INPUTS NODES TO DROP WITH EACH UPDATE 1
   ALL 0 NONE


  model.add(Dense(noOfClasses,activation='softmax')) # OUTPUT LAYER


  # COMPILE MODEL


  model.compile(Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accuracy'])
  return model


################################### TRAIN


model = myModel()
print(model.summary())
history=model.fit_generator(dataGen.flow(X_train,y_train,batch_size=batch_size_val),st
   eps_per_epoch=steps_per_epoch_val,epochs=epochs_val,validation_data=(X_validati
   on,y_validation),shuffle=1)




############################### Test Score
score =model.evaluate(X_test,y_test,verbose=0)
print('Test Score:',score[0])
print('Test Accuracy:',score[1])


################################### PLOT
```

```
#plt.subplot(211)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['training','validation'])
plt.title('loss')
plt.xlabel('epoch')


#plt.subplot(212)
#plt.plot(history.history['accuracy'])
#plt.plot(history.history['val_accuracy'])
#plt.legend(['training','validation'])
#plt.title('Acurracy')
#plt.xlabel('epoch')
plt.show()


# STORE THE MODEL AS A PICKLE OBJECT
################ Store as a Pickle object

pickle_out= open("model_trained2.p","wb")  # wb = WRITE BYTE
print("Model saved")

print("Successfully trained")
pickle.dump(model,pickle_out)
pickle_out.close()

cv2.waitKey(0)
```

Test.py

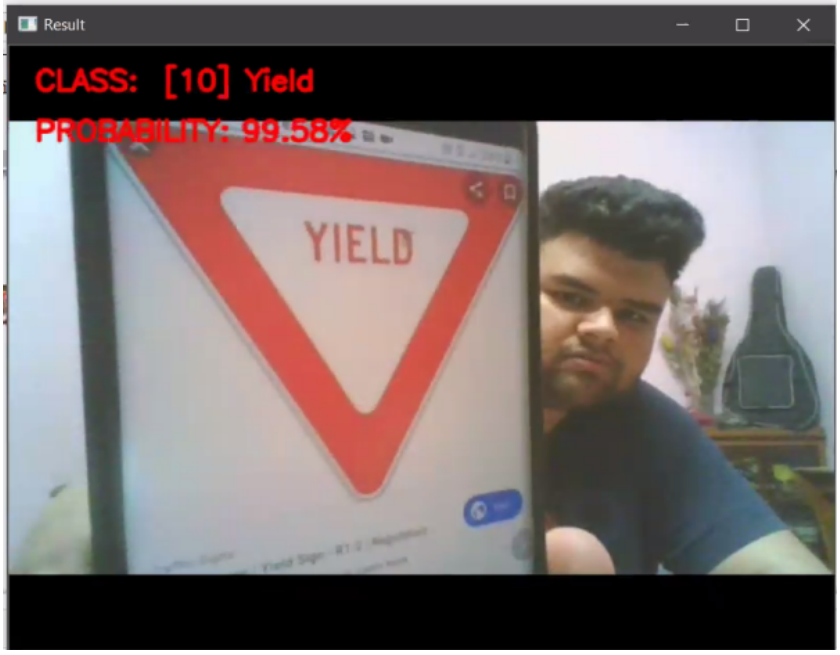
```
import cv2
import numpy as np
import pickle


#############################################

frameWidth= 640         # CAMERA RESOLUTION

frameHeight = 480
brightness = 180
threshold = 0.75         # PROBABLITY THRESHOLD

font = cv2.FONT_HERSHEY_SIMPLEX
#############################################


# SETUP THE VIDEO CAMERA

cap = cv2.VideoCapture(0)
cap.set(3, frameWidth)
cap.set(4, frameHeight)
cap.set(10, brightness)

print("Camera Setuped")
# IMPORT THE TRANNIED MODEL

pickle_in=open("model_trained2.p","rb")  ## rb = READ BYTE

model=pickle.load(pickle_in)
print("pickle loaded")

def grayscale(img):
   img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
   return img
def equalize(img):
   img =cv2.equalizeHist(img)
   return img
```

```
def preprocessing(img):
  img = grayscale(img)
  img = equalize(img)
  img = img/255
  return img
print("processing done")
def getCalssName(classNo):
  if   classNo == 0: return 'Speed Limit 20 km/h'
  elif classNo == 1: return 'Speed Limit 30 km/h'
  elif classNo == 2: return 'Speed Limit 50 km/h'
  elif classNo == 3: return 'Speed Limit 60 km/h'
  elif classNo == 4: return 'Speed Limit 70 km/h'
  elif classNo == 5: return 'Speed Limit 80 km/h'
  elif classNo == 6: return 'Speed Limit 100 km/h'
  elif classNo == 7: return 'Speed Limit 120 km/h'
  elif classNo == 8: return 'Right-of-way at the next intersection'
  elif classNo == 9: return 'Priority road'
  elif classNo == 10: return 'Yield'
  elif classNo == 11: return 'Stop'
  elif classNo == 12: return 'No vechiles'
  elif classNo == 13: return 'Vechiles over 3.5 metric tons prohibited'
  elif classNo == 14: return 'No entry'
  elif classNo == 15: return 'General caution'
  elif classNo == 16: return 'Dangerous curve to the right'
  elif classNo == 17: return 'Double curve'
  elif classNo == 18: return 'Bumpy road'
  elif classNo == 19: return 'Road narrows on the right'
  elif classNo == 20: return 'Road work'
  elif classNo == 21: return 'Traffic signals'
  elif classNo == 22: return 'Pedestrians'
  elif classNo == 23: return 'Wild animals crossing'
  elif classNo == 24: return 'Turn right ahead'
  elif classNo == 25: return 'Turn left ahead'
  elif classNo == 26: return 'Ahead only'
  elif classNo == 27: return 'Go straight or right'
  elif classNo == 28: return 'Go straight or left'
  elif classNo == 29: return 'Keep right'
  elif classNo == 30: return 'Roundabout mandatory'
  else: print("Obstacle")
```

```
print("got Class no")
while True:

  # READ IMAGE

  success, imgOrignal = cap.read()

  # PROCESS IMAGE

  img = np.asarray(imgOrignal)
  img = cv2.resize(img, (32, 32))
  img = preprocessing(img)
  cv2.imshow("Processed Image", img)
  img = img.reshape(1, 32, 32, 1)
  cv2.putText(imgOrignal, "CLASS: " , (20, 35), font, 0.75, (0, 0, 255), 2,
   cv2.LINE_AA)
  cv2.putText(imgOrignal, "PROBABILITY: ", (20, 75), font, 0.75, (0, 0, 255), 2,
   cv2.LINE_AA)

  # PREDICT IMAGE

  predictions = model.predict(img)
  classIndex = model.predict_classes(img)
  probabilityValue =np.amax(predictions)
  if probabilityValue > threshold:
     print(getCalssName(classIndex))

     cv2.putText(imgOrignal,str(classIndex)+" "+str(getCalssName(classIndex)), (120,
   35), font, 0.75, (0, 0, 255), 2, cv2.LINE_AA)
     cv2.putText(imgOrignal, str(round(probabilityValue*100,2) )+"%", (180, 75), font,
   0.75, (0, 0, 255), 2, cv2.LINE_AA)
  cv2.imshow("Result", imgOrignal)


  if cv2.waitKey(1) & 0xFF == ord('q'):
     break
```

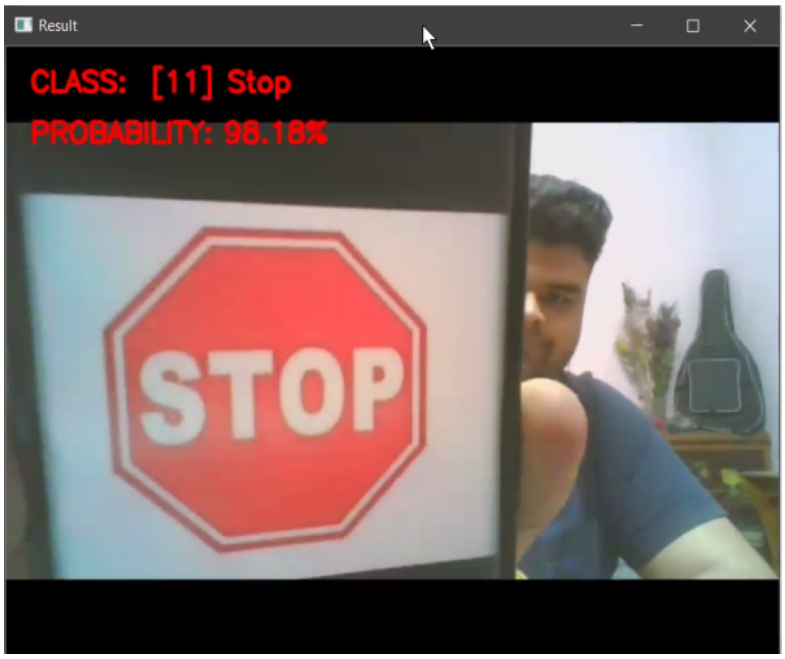
RESULTS & DISCUSSIONS

The purpose of this project starts from the assumption that the traffic recognition field can be proven very helpful and supportive in order to control road accidents in modern automobiles. There are a lot many ways already discovered and devised to recognize Traffic signs and classify them. This project is totally based on the study of various methods already available in order to classify and recognize traffic signs with much accuracy and implementation one such method for observing how it works.



Recognition of Yield sign

Accuracy: 99.58%

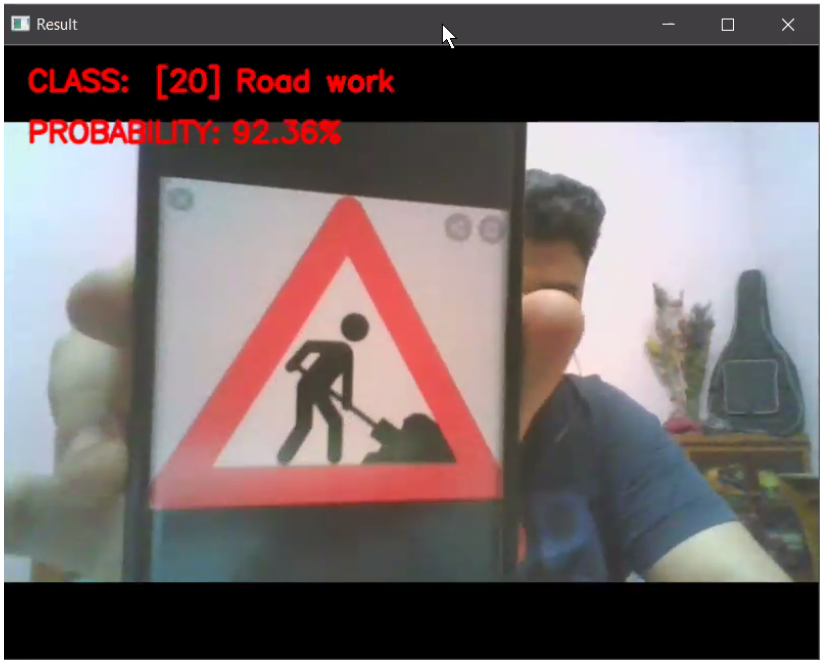


Recognition of STOP sign

Accuracy: 98.18%

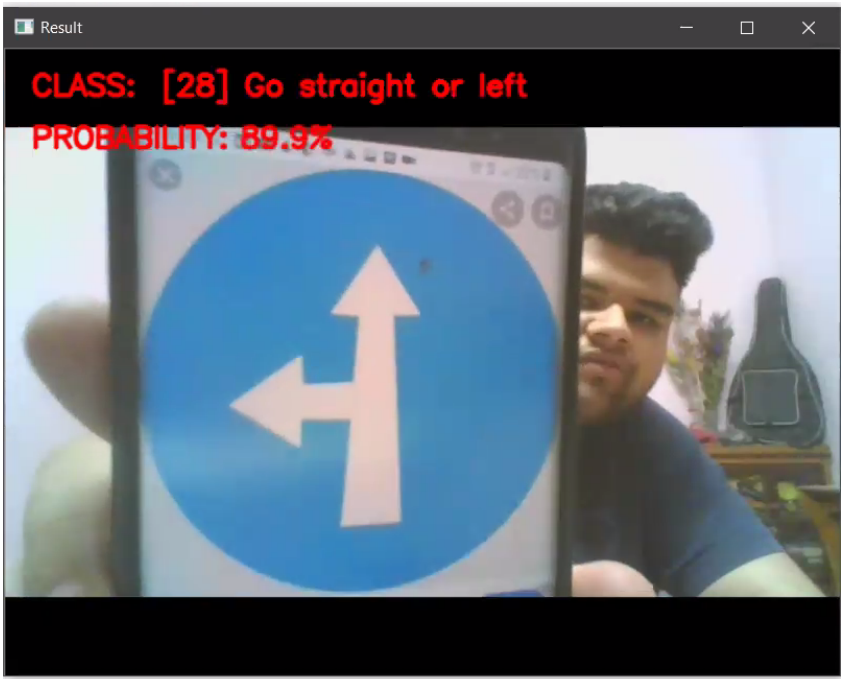
Recognition of 80 km/h max speed sign

Accuracy: 98.60%



Recognition of Road Work Ahead sign

Accuracy: 92.36%



Recognition of Go Straight or Left sign

Accuracy: 89.90%

PAST WORKS ON TSR

Various algorithms and classification techniques have been used earlier to enhance the traffic sign detection field and a few are illustrated here:

PROMETHEUS (Program for European Traffic with Highest Efficiency and Unprecedented Safety) was an initiative started in 1986 in Europe to stimulate research in development of a transport system to exploit full potential of information and telecommunication technologies to improve traffic efficiency and safety. Safe driving was one of the three identified main work areas and meant to employ autonomous vehicle control for safer driving with a less mental load on the driver. A TSR system was developed in Daimler-Benz as part of the collision avoidance project. It employed a detection process to scan the image for possible traffic sign candidates and a tracking process to identify the sign and track it in the following images. Three sub-units (called specialists) were used: color segmentation specialist to find a sign candidate based on color of regions in the image, shape recognition specialist to classify the candidate based on its contour and pictogram recognition specialist to identify the pictogram inside the traffic sign by comparing it against a library of possible pictograms. UC Berkeley's PATH is another example. It is a collaboration between the California Department of Transportation (Caltrans) and the University of California with the mission of applying advanced technology to increase highway capacity and safety, and to reduce traffic congestion, air pollution, and energy consumption. It consists of several projects. For example, its Stereo Drive project explored the feasibility of the use of stereo vision (for providing range information) in conjunction with a scanning laser radar sensor to detect obstacles and also maintain a fixed distance from a lead vehicle using the feedback provided by range sensors. In a particular area of TSR, there have been much research in academia and also industry (e.g. as part of PROMETHEUS) especially in the mid-'90s. Different approaches have been used in different stage of the problem from colour segmentation, control theory, feature extraction, learning-based neural networks, morphological based recognition, etc.

CONCLUSIONS

This project surely helped us a lot to gain knowledge in the fields which we had only heard about and never researched it. From the viewpoint of traffic sign recognition accuracy and algorithm time-consuming, the proposed traffic sign detection and recognition algorithm has remarkable advantages. Considerably enhancing the driving safety of intelligent vehicles in the actual driving environments and effectively meeting the real-time target requirements of smart cars are conducive. Furthermore, a strong technical guarantee is provided for the steady development of intelligent vehicle driving assistance. In the future, the inclusiveness and anti-error recognition of the traffic sign recognition algorithm can be further optimized and improved to exploit the overall performance of the algorithm.

Appendix A

CONTRIBUTION

The team comprised of three members and each one of them had contributed well on their Part. The efficiency of the team is surely reflected in the work showcased above.

Anirban Bose (1728191, CSSE): Had done research work based on the topics relevant to the project and found the papers holding much importance for the project. Equal contribution in writing the report. Helped with the documentation part basis open cv and their implementation.

Dipendra Upadhya (1728258, CSSE): Had some role in each and every step of the project as it progressed with time. The start included study and construction of the renowned research papers and get the idea as to how to implement the project in terms of documentation, writing the report. Project planning and methodology lying under the work exhibited.

Deepanshu Jayswal (1728263, CSSE): The mind behind the idea of shifting the project towards traffic sign detection and getting a device out of it, had an equal contribution in the study of the research papers. Took the initiative to design the presentation for the same with the help of all the members. Worked out with the code to implement the project and did the performance evaluation of different types of algorithms and hence finding the best-suited algorithm to implement.

11. REFERENCES

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6767627/

https://en.wikipedia.org/wiki/Traffic-sign_recognition

https://www.design-reuse.com/articles/41154/traffic-sign-recognition-tsr-system.html