TRAFFIC SIGN CLASSIFIER

A PROJECT REPORT

Submitted

in the partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

HARSHITH MAKKAPATI (19B81A0523)
SAI VAMSHI CHALLAMALLA (19B81A0538)

Under the guidance of
Dr.K. VENKATESH SHARMA
Senior Assistant Professor, CSE Department

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CVR COLLEGE OF ENGINEERING

(An Autonomous institution, NBA, NAAC Accredited and Affiliated toJNTUH, Hyderabad)

Vastunagar, Mangalpalli (V), Ibrahimpatnam(M), Rangareddy (D), Telangana- 501 510

October 2022

Executive Summary

We will use the German data set to import the images into this project. The starting step of this project is to load the data set, We have used the numpy library to calculate summary statistics of the traffic signs data set. The size of training set is 34799, The size of the validation set is 4410, The size of test set is 12630, The shape of a traffic sign image is (32, 32, 3), The number of unique classes/labels in the data set is 43. After that we will explore, summarize and visualize the data set. Design, training and testing of a model architecture occurs after we have visualized the whole data set. Then, we will use the model to make predictions on new images and analyse the softmax probabilities of the new image.
Finally, we test the model with a data set.

1 INTRODUCTION:

1.1 Overview:

The number of road accidents are increasing every year.There were around 500k accidents in India last year which caused 148,546 deaths. India contains 1% of the world's vehicles but accounts for 6% of the world's road traffic accidents. Due to this reason Traffic Sign Classifier can prove to be a great asset for the transportation industry as it will navigate and give information about all the traffic signs encountered by the vehicle.

1.2 Objective:

Recognizing traffic signs is one of the essential parts of the race to the ultimate Self Driving Car System and Deep Learning plays a big role in the development of this system. We will be building a Convolutional Neural Network model in order to detect traffic signs. This whole classifier is built in Python with the TensorFlow framework for numerical computation and a couple of other dependencies like numpy and scikit-image.
Also, we will be using the keras package to build CNN models.

1.3 Motivation:

With the development of automotive intelligent technology, many different famous car companies like Mercedes-Benz, BMW, etc., have actively invested in ADAS (Advanced Driver Assistance System) research. Commercialized ADAS also include TSR (Traffic Sign Recognition) systems to remind the drivers to pay attention to the speed and help in preventing road accidents. With the increase in demand of TSR (Traffic Sign Recognition) it is impossible for others to understand the advantages and needs of this system.\

2 PROJECT DESCRIPTION AND GOALS:

Abstract:

The traffic sign recognition system inside the vehicle plays an important role and can guarantee the safety of human life on the road since it feedback road information to the driver in time. Benefited from learning features of the traffic sign, the convolutional neural network (CNN) has been widely used in traffic sign recognition with high accuracy. However, the different kinds of traffic signs appear to have distinctive features. A deep and high complexity neural network with a larger number of parameters is usually required to adapt the distinctive features, while it tends to be time-consuming and can not meet real-time requirements. Our main goal is to overcome this above disadvantage and improve accuracy and detection of traffic signs and along with this solve the problem of providing intensive care as soon as possible happened due to traffic mishaps by integrating this system with our new age help providing website.

KEYWORDS:

Traffic Sign Detection and Tracking (TSDR), Advanced Driver Assistance System (ADAS), Computer Vision.

3 TECHNICAL SPECIFICATION:

SOFTWARE AND HARDWARE REQUIREMENTS:

- CNN MODEL 2.3.2
- BASIC K CLUSTERING SQL
- IMAGE PREPROCESSING PRO 2.1.1
- NUMPY LIBRARY

4 DESIGN APPROACH AND DETAILS:

4.1 Design Approach / Materials & Methods:

INNOVATION THAT WILL BE CARRIED OUT:

A large number of traffic recognition systems have been developed. Firstly, solutions were focusing on optical based micro-programmed hardware in order to avoid computational complexity. Even with the presence of existing systems, the innovation that we bring to our project is, integrating this technology with an end-to-end service which provides help to people who have been affected by accidents in real quick time, by providing a platform to social helpers such as police, hospitals and ambulance drivers to provide their expertise. For example providing ambulance drivers with shortest route to their respective hospitals along with sending alerts to nearby hospitals about the condition of the patient, so that they can be ready with the pre-requisites along with many other facilities.

PROPOSED METHODOLOGY:

CONVOLUTION NEURAL NETWORK, OBJECT CLASSIFICATION, K MEANS CLUSTERING, MACHINE LEARNING, IMAGE DETECTION

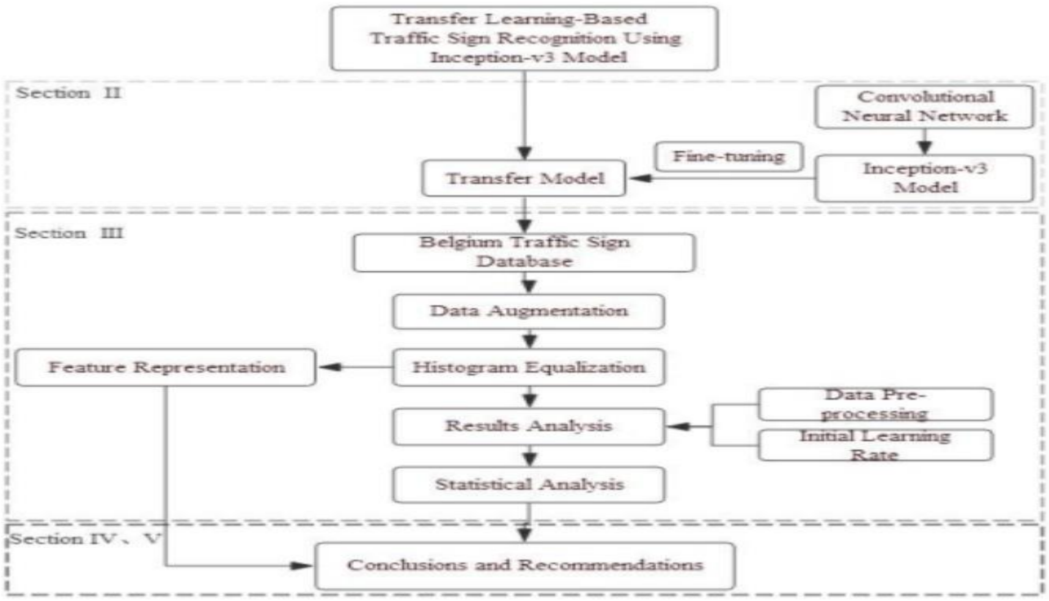
A convolutional neural network is a type of neural network that is most often applied to image processing problems The fact that they are useful for these fast-growing areas is one of the main reasons they're so important in deep learning and artificial intelligence today. The word convolutional refers to the filtering process that happens in this type of network. Think of it this way, an image is complex. A convolutional neural network simplifies it, so it can be better processed and understood. We will be designing two CNNs to classify the subclasses for the two super classes of traffic signs independently. One CNN will be used to classify the subclass of circular traffic signs, and the other for the triangular traffic signs. The CNN for circular traffic signs should classify the super class into 20 subclasses, and the CNN for triangular traffic signs classify the super class into 15 subclasses. We believe that further partitioning the subset of the database and classifying it independently, reduces the interference between different classes and improves classification accuracy. To verify the effectiveness of this approach, we designed the third CNN that classified all the traffic signs.

DATASET:

The German traffic signs detection dataset consists of 39209 images with 43 different classes. The images are distributed unevenly between those classes and hence the model may predict some classes more accurately than other classes. We can populate the dataset with various image modifying techniques such as rotation, colour distortion or blurring the image. We will be training the model on the original dataset and will see the accuracy of the model. Then we will be adding more data and making each class even and check the model's accuracy.

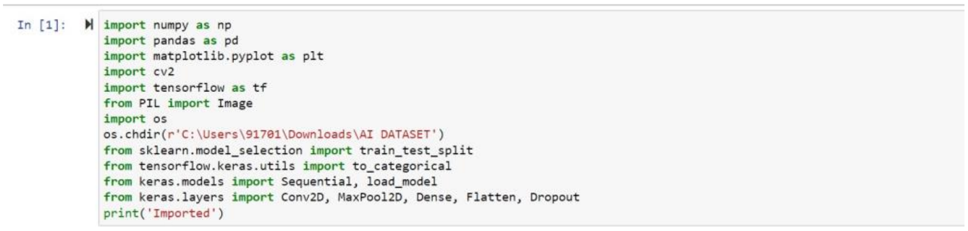
Our project is not based on any other reference projects from Stanford or MIT.

Flowchart:

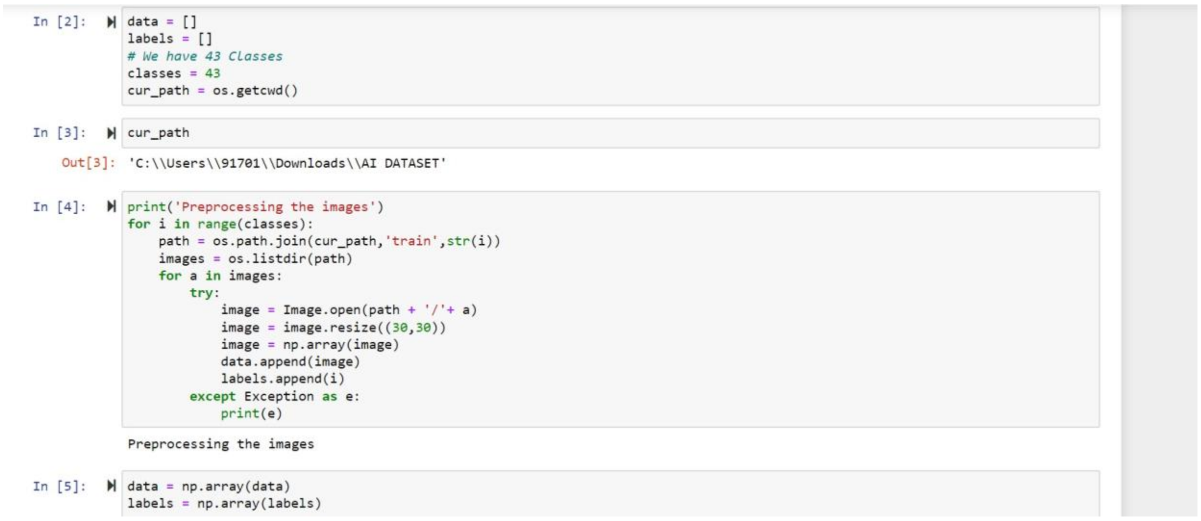


Transfer Learning-Based Traffic Sign Recognition Using Inception-v3 Model
Section II
Convolutional Neural Network
Fine-tuning
Transfer Model
Inception-v3 Model
Section III
Belgium Traffic Sign Database
Data Augmentation
Feature Representation
Histogram Equalization
Data Pre-processing
Results Analysis
Initial Learning Rate
Statistical Analysis
Section IV、V
Conclusions and Recommendations

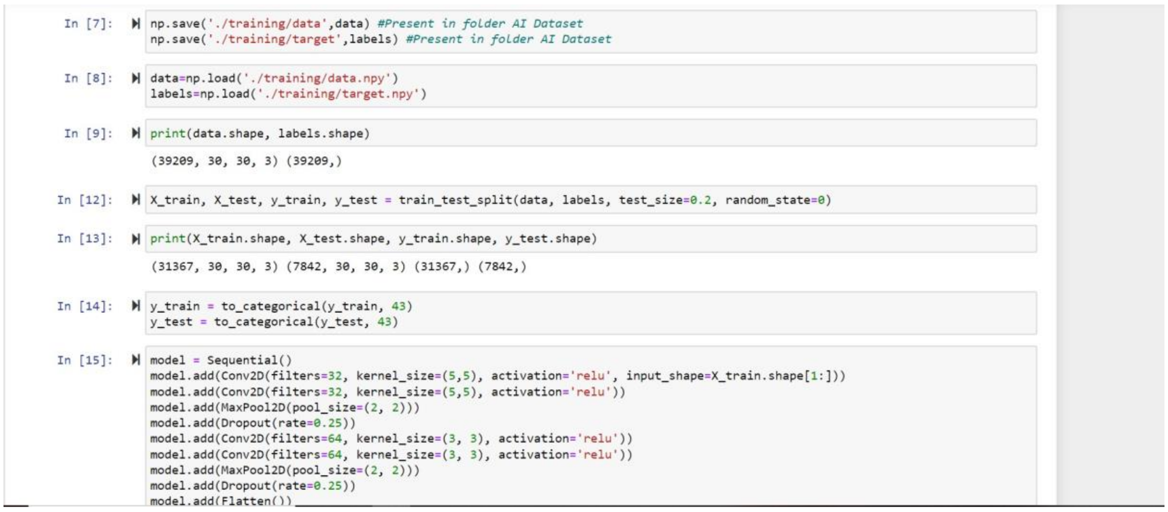
4.2 Codes and Standards:



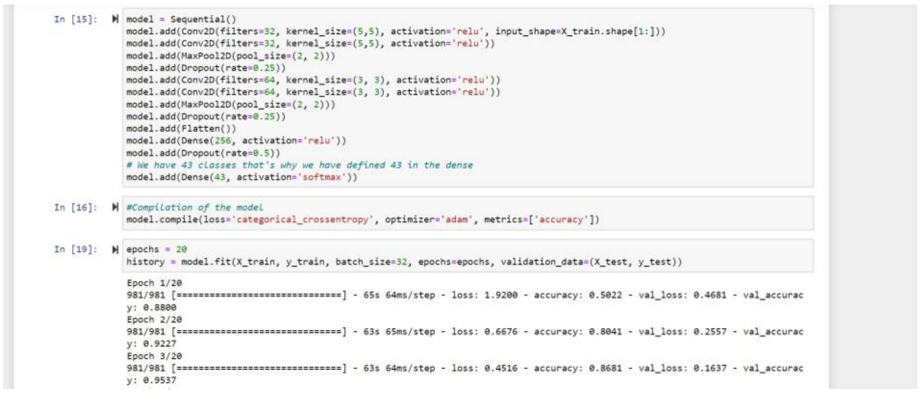
```
In [1]:  import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import cv2
         import tensorflow as tf
         from PIL import Image
         import os
         os.chdir(r'C:\Users\91701\Downloads\AI DATASET')
         from sklearn.model_selection 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
         print('Imported')
```



```
In [2]:  data = []
         labels = []
         # We have 43 Classes
         classes = 43
         cur_path = os.getcwd()

In [3]:  cur_path

Out[3]:  'C:\\Users\\91701\\Downloads\\AI DATASET'

In [4]:  print('Preprocessing the images')
         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)
                     data.append(image)
                     labels.append(i)
                 except Exception as e:
                     print(e)

         Preprocessing the images

In [5]:  data = np.array(data)
         labels = np.array(labels)
```

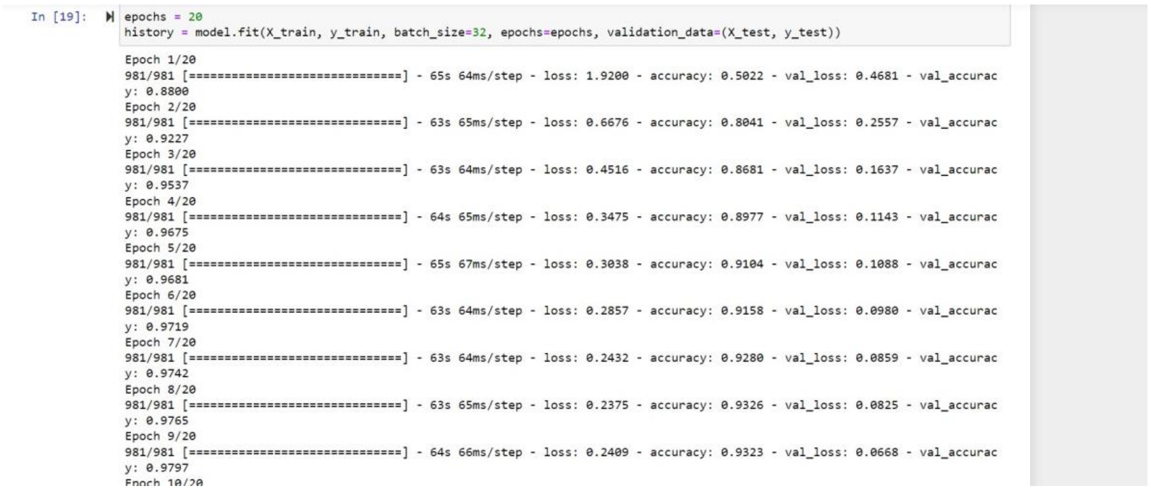


```
In [7]:  np.save('./training/data',data) #Present in folder AI Dataset
         np.save('./training/target',labels) #Present in folder AI Dataset

In [8]:  data=np.load('./training/data.npy')
         labels=np.load('./training/target.npy')

In [9]:  print(data.shape, labels.shape)

         (39209, 30, 30, 3) (39209,)

In [12]: X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=0)

In [13]: print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

         (31367, 30, 30, 3) (7842, 30, 30, 3) (31367,) (7842,)

In [14]: y_train = to_categorical(y_train, 43)
         y_test = to_categorical(y_test, 43)

In [15]: 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())
```

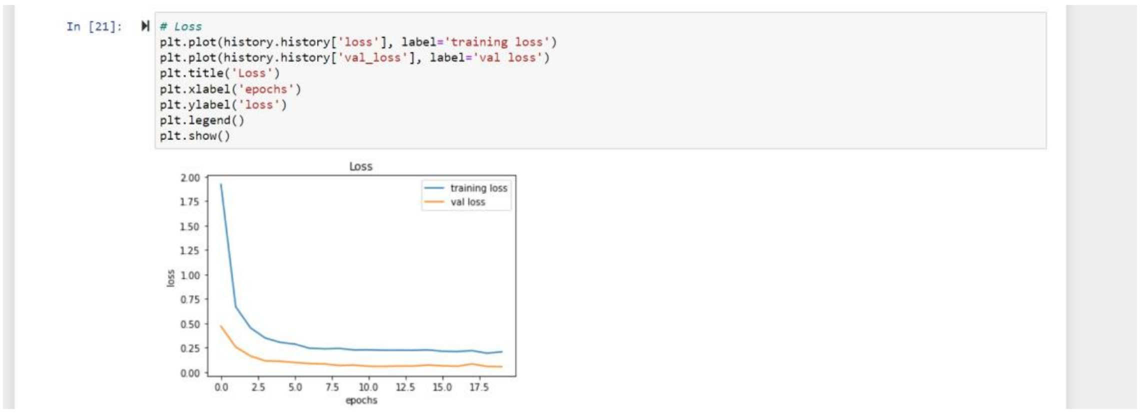


```
In [15]: model = Sequential()
         model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
         model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
         model.add(MaxPool2D(pool_size=(2, 2)))
         model.add(Dropout(rate=0.25))
         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))
         # We have 43 classes that's why we have defined 43 in the dense
         model.add(Dense(43, activation='softmax'))

In [16]: #Compilation of the model
         model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

In [19]: epochs = 20
         history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))

         Epoch 1/20
         981/981 [==============================] - 65s 64ms/step - loss: 1.9200 - accuracy: 0.5022 - val_loss: 0.4681 - val_accurac
         y: 0.8800
         Epoch 2/20
         981/981 [==============================] - 63s 65ms/step - loss: 0.6676 - accuracy: 0.8041 - val_loss: 0.2557 - val_accurac
         y: 0.9227
         Epoch 3/20
         981/981 [==============================] - 63s 64ms/step - loss: 0.4516 - accuracy: 0.8681 - val_loss: 0.1637 - val_accurac
         y: 0.9537
```

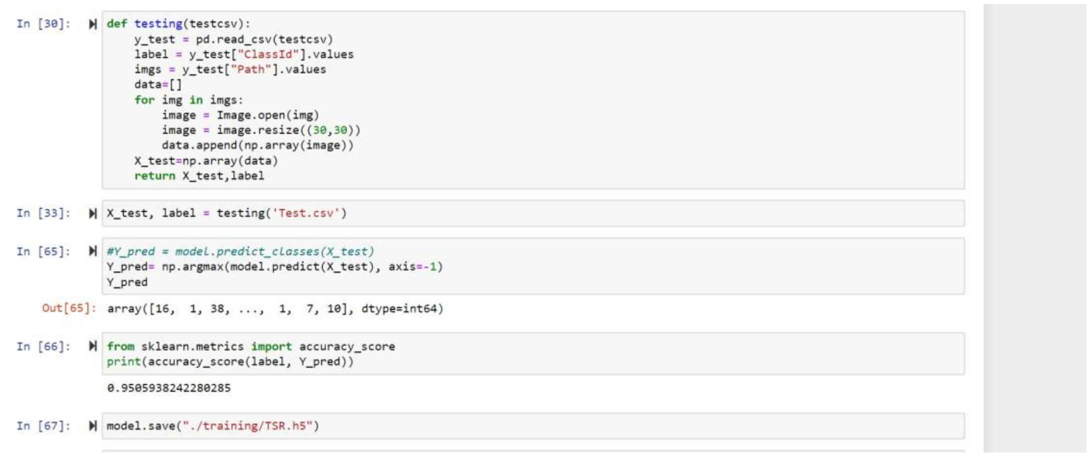
4.2 Codes and Standards:



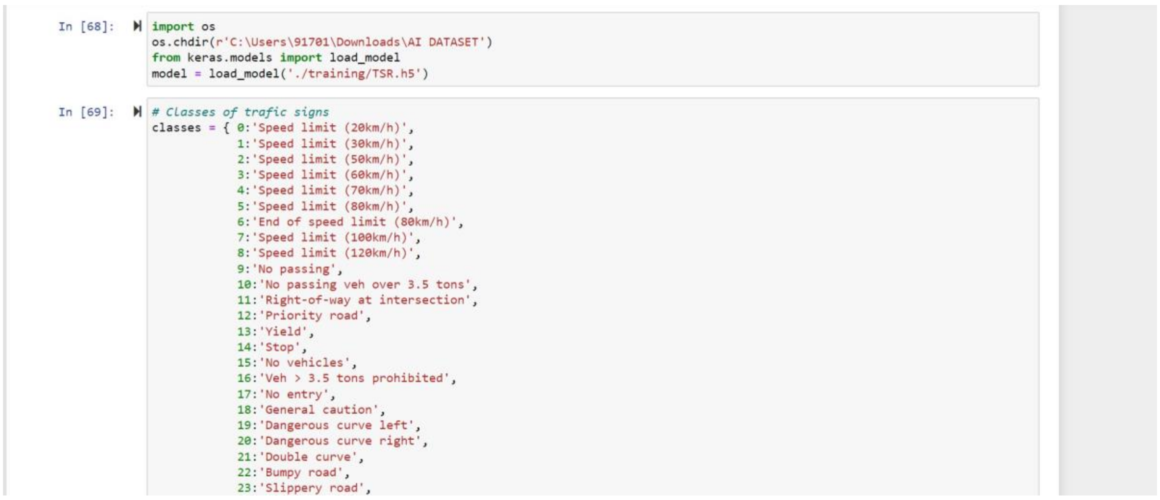
```
In [19]:  epochs = 20
          history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))

          Epoch 1/20
          981/981 [==============================] - 65s 64ms/step - loss: 1.9200 - accuracy: 0.5022 - val_loss: 0.4681 - val_accurac
          y: 0.8800
          Epoch 2/20
          981/981 [==============================] - 63s 65ms/step - loss: 0.6676 - accuracy: 0.8041 - val_loss: 0.2557 - val_accurac
          y: 0.9227
          Epoch 3/20
          981/981 [==============================] - 63s 64ms/step - loss: 0.4516 - accuracy: 0.8681 - val_loss: 0.1637 - val_accurac
          y: 0.9537
          Epoch 4/20
          981/981 [==============================] - 64s 65ms/step - loss: 0.3475 - accuracy: 0.8977 - val_loss: 0.1143 - val_accurac
          y: 0.9675
          Epoch 5/20
          981/981 [==============================] - 65s 67ms/step - loss: 0.3038 - accuracy: 0.9104 - val_loss: 0.1088 - val_accurac
          y: 0.9681
          Epoch 6/20
          981/981 [==============================] - 63s 64ms/step - loss: 0.2857 - accuracy: 0.9158 - val_loss: 0.0980 - val_accurac
          y: 0.9719
          Epoch 7/20
          981/981 [==============================] - 63s 64ms/step - loss: 0.2432 - accuracy: 0.9280 - val_loss: 0.0859 - val_accurac
          y: 0.9742
          Epoch 8/20
          981/981 [==============================] - 63s 65ms/step - loss: 0.2375 - accuracy: 0.9326 - val_loss: 0.0825 - val_accurac
          y: 0.9765
          Epoch 9/20
          981/981 [==============================] - 64s 66ms/step - loss: 0.2409 - accuracy: 0.9323 - val_loss: 0.0668 - val_accurac
          y: 0.9797
          Epoch 10/20
```



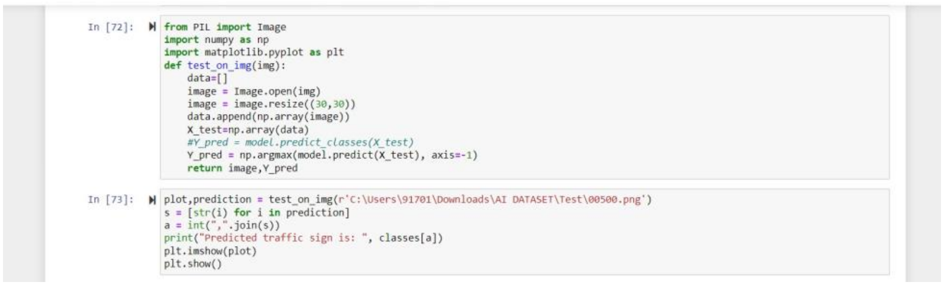
```
In [21]:  # Loss
          plt.plot(history.history['loss'], label='training loss')
          plt.plot(history.history['val_loss'], label='val loss')
          plt.title('Loss')
          plt.xlabel('epochs')
          plt.ylabel('loss')
          plt.legend()
          plt.show()
```



```
In [30]:  def testing(testcsv):
              y_test = pd.read_csv(testcsv)
              label = 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)
              return X_test,label

In [33]:  X_test, label = testing('Test.csv')

In [65]:  #Y_pred = model.predict_classes(X_test)
          Y_pred= np.argmax(model.predict(X_test), axis=-1)
          Y_pred

Out[65]:  array([16,  1, 38, ...,  1,  7, 10], dtype=int64)

In [66]:  from sklearn.metrics import accuracy_score
          print(accuracy_score(label, Y_pred))

          0.9505938242280285

In [67]:  model.save("./training/TSR.h5")
```

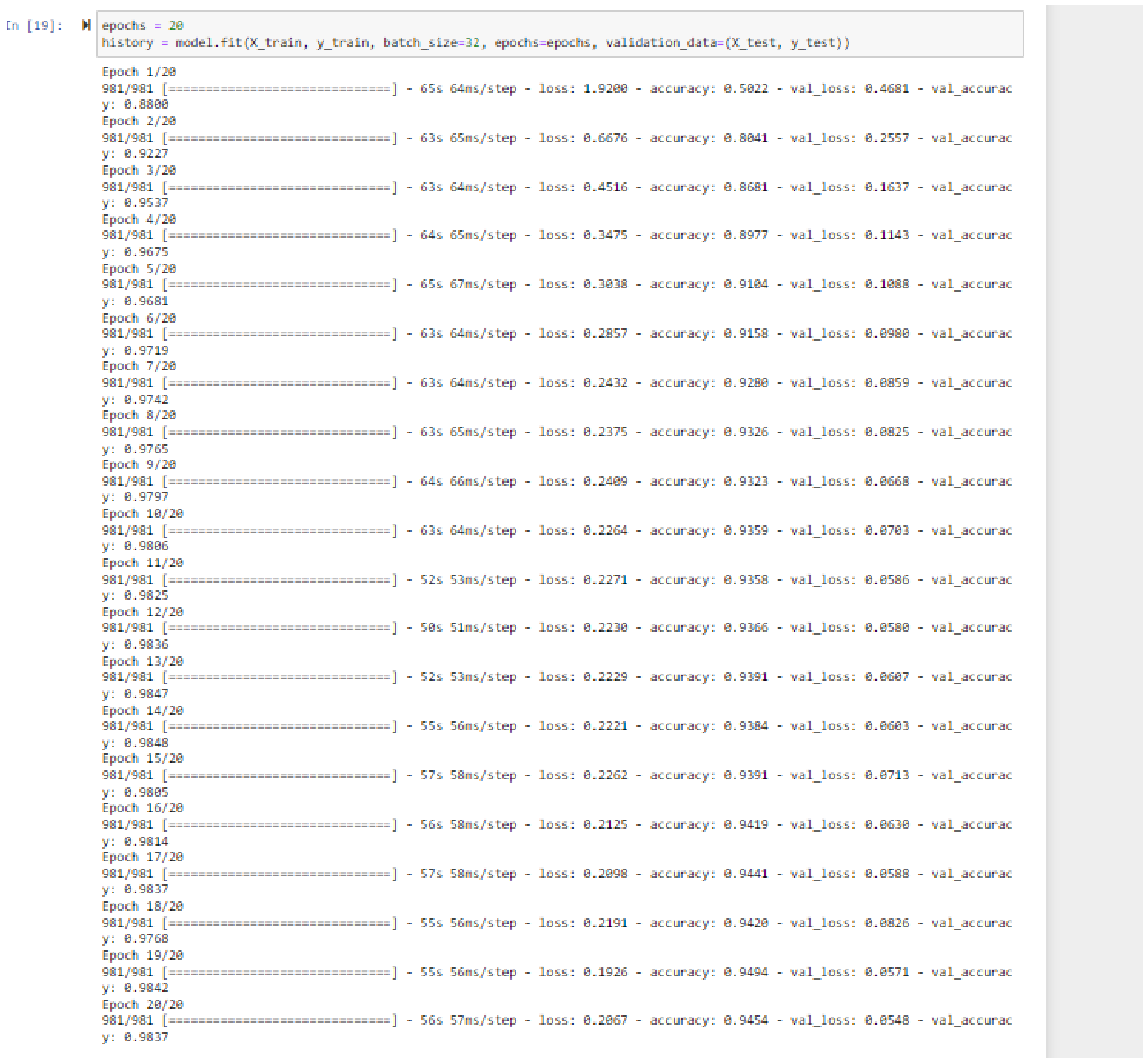


```
In [68]:  import os
          os.chdir(r'C:\Users\91701\Downloads\AI DATASET')
          from keras.models import load_model
          model = load_model('./training/TSR.h5')

In [69]:  # Classes of trafic signs
          classes = { 0:'Speed limit (20km/h)',
                      1:'Speed limit (30km/h)',
                      2:'Speed limit (50km/h)',
                      3:'Speed limit (60km/h)',
                      4:'Speed limit (70km/h)',
                      5:'Speed limit (80km/h)',
                      6:'End of speed limit (80km/h)',
                      7:'Speed limit (100km/h)',
                      8:'Speed limit (120km/h)',
                      9:'No passing',
                      10:'No passing veh over 3.5 tons',
                      11:'Right-of-way at intersection',
                      12:'Priority road',
                      13:'Yield',
                      14:'Stop',
                      15:'No vehicles',
                      16:'Veh > 3.5 tons prohibited',
                      17:'No entry',
                      18:'General caution',
                      19:'Dangerous curve left',
                      20:'Dangerous curve right',
                      21:'Double curve',
                      22:'Bumpy road',
                      23:'Slippery road',
```

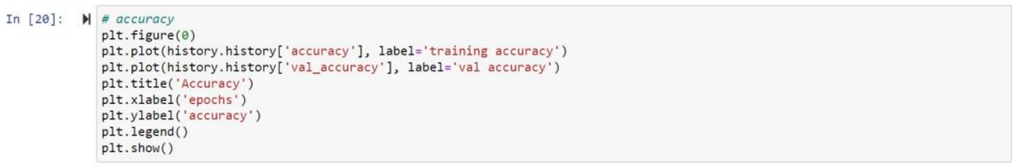


```
In [72]:  from PIL import Image
          import numpy as np
          import matplotlib.pyplot as plt
          def test_on_img(img):
              data=[]
              image = Image.open(img)
              image = image.resize((30,30))
              data.append(np.array(image))
              X_test=np.array(data)
              #Y_pred = model.predict_classes(X_test)
              Y_pred = np.argmax(model.predict(X_test), axis=-1)
              return image,Y_pred

In [73]:  plot,prediction = test_on_img(r'C:\Users\91701\Downloads\AI DATASET\Test\00500.png')
          s = [str(i) for i in prediction]
          a = int(",".join(s))
          print("Predicted traffic sign is: ", classes[a])
          plt.imshow(plot)
          plt.show()
```

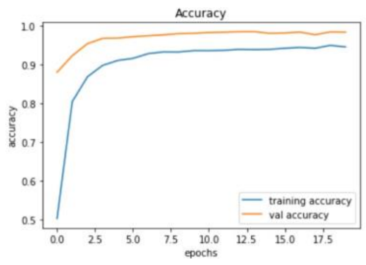
4.3 Constraints, Alternatives and Tradeoffs:



```
In [19]:  epochs = 20
          history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))

          Epoch 1/20
          981/981 [==============================] - 65s 64ms/step - loss: 1.9200 - accuracy: 0.5022 - val_loss: 0.4681 - val_accurac
          y: 0.8800
          Epoch 2/20
          981/981 [==============================] - 63s 65ms/step - loss: 0.6676 - accuracy: 0.8041 - val_loss: 0.2557 - val_accurac
          y: 0.9227
          Epoch 3/20
          981/981 [==============================] - 63s 64ms/step - loss: 0.4516 - accuracy: 0.8681 - val_loss: 0.1637 - val_accurac
          y: 0.9537
          Epoch 4/20
          981/981 [==============================] - 64s 65ms/step - loss: 0.3475 - accuracy: 0.8977 - val_loss: 0.1143 - val_accurac
          y: 0.9675
          Epoch 5/20
          981/981 [==============================] - 65s 67ms/step - loss: 0.3038 - accuracy: 0.9104 - val_loss: 0.1088 - val_accurac
          y: 0.9681
          Epoch 6/20
          981/981 [==============================] - 63s 64ms/step - loss: 0.2857 - accuracy: 0.9158 - val_loss: 0.0980 - val_accurac
          y: 0.9719
          Epoch 7/20
          981/981 [==============================] - 63s 64ms/step - loss: 0.2432 - accuracy: 0.9280 - val_loss: 0.0859 - val_accurac
          y: 0.9742
          Epoch 8/20
          981/981 [==============================] - 63s 65ms/step - loss: 0.2375 - accuracy: 0.9326 - val_loss: 0.0825 - val_accurac
          y: 0.9765
          Epoch 9/20
          981/981 [==============================] - 64s 66ms/step - loss: 0.2409 - accuracy: 0.9323 - val_loss: 0.0668 - val_accurac
          y: 0.9797
          Epoch 10/20
          981/981 [==============================] - 63s 64ms/step - loss: 0.2264 - accuracy: 0.9359 - val_loss: 0.0703 - val_accurac
          y: 0.9806
          Epoch 11/20
          981/981 [==============================] - 52s 53ms/step - loss: 0.2271 - accuracy: 0.9358 - val_loss: 0.0586 - val_accurac
          y: 0.9825
          Epoch 12/20
          981/981 [==============================] - 50s 51ms/step - loss: 0.2230 - accuracy: 0.9366 - val_loss: 0.0580 - val_accurac
          y: 0.9836
          Epoch 13/20
          981/981 [==============================] - 52s 53ms/step - loss: 0.2229 - accuracy: 0.9391 - val_loss: 0.0607 - val_accurac
          y: 0.9847
          Epoch 14/20
          981/981 [==============================] - 55s 56ms/step - loss: 0.2221 - accuracy: 0.9384 - val_loss: 0.0603 - val_accurac
          y: 0.9848
          Epoch 15/20
          981/981 [==============================] - 57s 58ms/step - loss: 0.2262 - accuracy: 0.9391 - val_loss: 0.0713 - val_accurac
          y: 0.9805
          Epoch 16/20
          981/981 [==============================] - 56s 58ms/step - loss: 0.2125 - accuracy: 0.9419 - val_loss: 0.0630 - val_accurac
          y: 0.9814
          Epoch 17/20
          981/981 [==============================] - 57s 58ms/step - loss: 0.2098 - accuracy: 0.9441 - val_loss: 0.0588 - val_accurac
          y: 0.9837
          Epoch 18/20
          981/981 [==============================] - 55s 56ms/step - loss: 0.2191 - accuracy: 0.9420 - val_loss: 0.0826 - val_accurac
          y: 0.9768
          Epoch 19/20
          981/981 [==============================] - 55s 56ms/step - loss: 0.1926 - accuracy: 0.9494 - val_loss: 0.0571 - val_accurac
          y: 0.9842
          Epoch 20/20
          981/981 [==============================] - 56s 57ms/step - loss: 0.2067 - accuracy: 0.9454 - val_loss: 0.0548 - val_accurac
          y: 0.9837
```

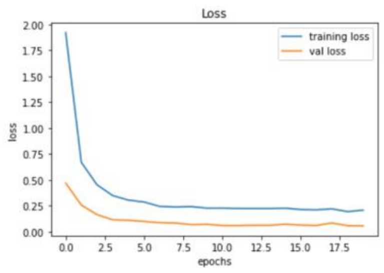


```
In [20]:  # accuracy
          plt.figure(0)
          plt.plot(history.history['accuracy'], label='training accuracy')
          plt.plot(history.history['val_accuracy'], label='val accuracy')
          plt.title('Accuracy')
          plt.xlabel('epochs')
          plt.ylabel('accuracy')
          plt.legend()
          plt.show()
```



In [21]:

```python
# Loss
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()
```



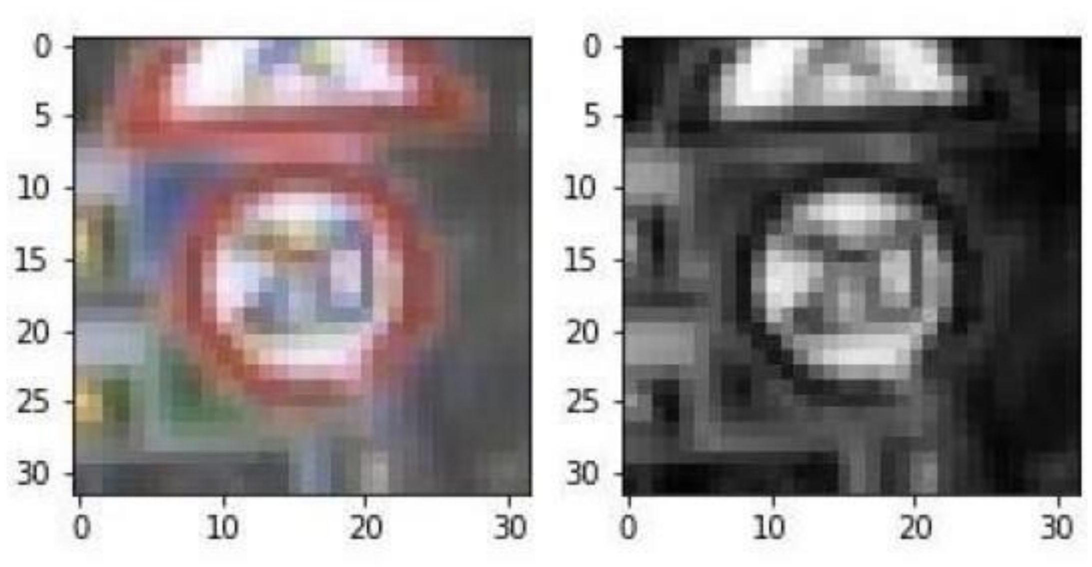
In [69]:

```python
# Classes of trafic signs
classes = { 0:'Speed limit (20km/h)',
            1:'Speed limit (30km/h)',
            2:'Speed limit (50km/h)',
            3:'Speed limit (60km/h)',
            4:'Speed limit (70km/h)',
            5:'Speed limit (80km/h)',
            6:'End of speed limit (80km/h)',
            7:'Speed limit (100km/h)',
            8:'Speed limit (120km/h)',
            9:'No passing',
            10:'No passing veh over 3.5 tons',
            11:'Right-of-way at intersection',
            12:'Priority road',
            13:'Yield',
            14:'Stop',
            15:'No vehicles',
            16:'Veh > 3.5 tons prohibited',
            17:'No entry',
            18:'General caution',
            19:'Dangerous curve left',
            20:'Dangerous curve right',
            21:'Double curve',
            22:'Bumpy road',
            23:'Slippery road',
            24:'Road narrows on the right',
            25:'Road work',
            26:'Traffic signals',
            27:'Pedestrians',
            28:'Children crossing',
            29:'Bicycles crossing',
            30:'Beware of ice/snow',
            31:'Wild animals crossing',
            32:'End speed + passing limits',
            33:'Turn right ahead',
            34:'Turn left ahead',
            35:'Ahead only',
            36:'Go straight or right',
            37:'Go straight or left',
            38:'Keep right',
            39:'Keep left',
            40:'Roundabout mandatory',
            41:'End of no passing',
            42:'End no passing veh > 3.5 tons' }
```

DESIGN APPROACH AND DETAILS

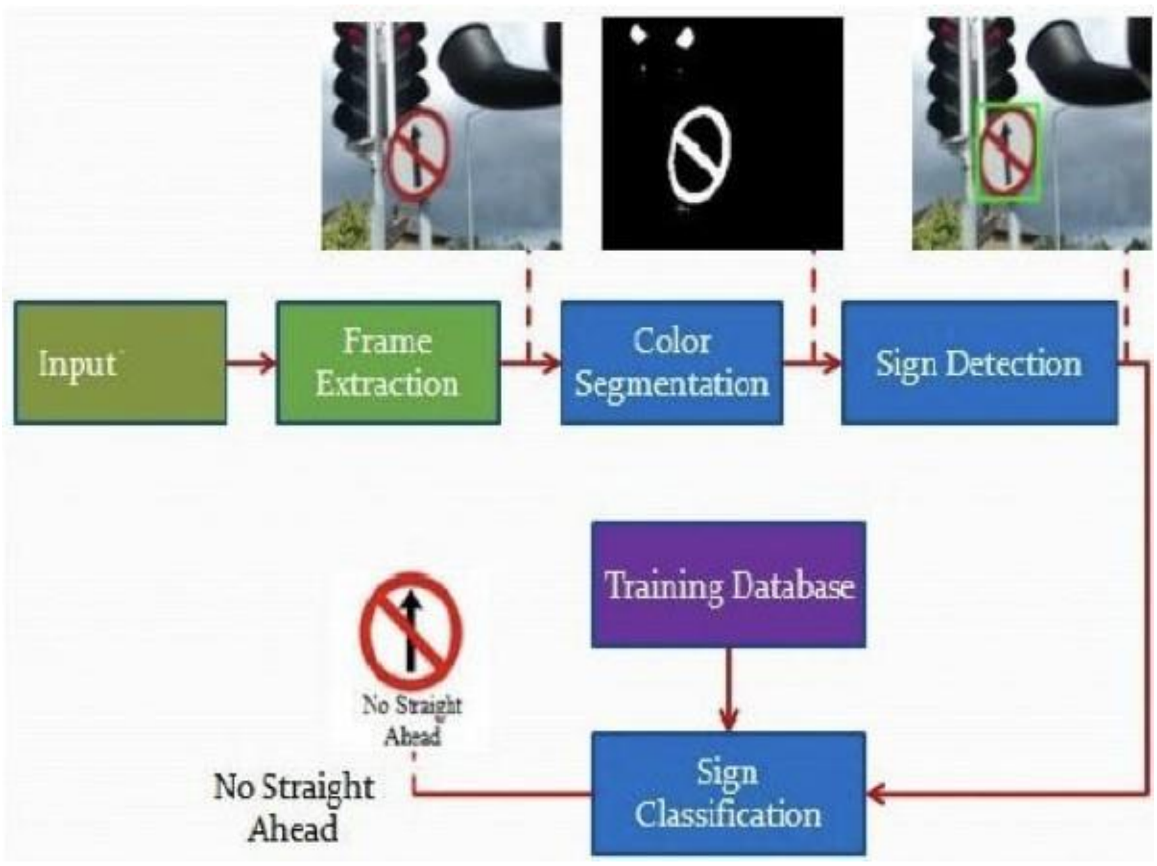
As a first step, We have decided to convert the images to grayscale because several images in the training were pretty dark and contained only little color and the gray scaling reduces the amount of features and thus reduces execution time.

Additionally, several research papers have shown good results with rescaling of the images.
Here is an example of a traffic sign image before and after gray scaling.

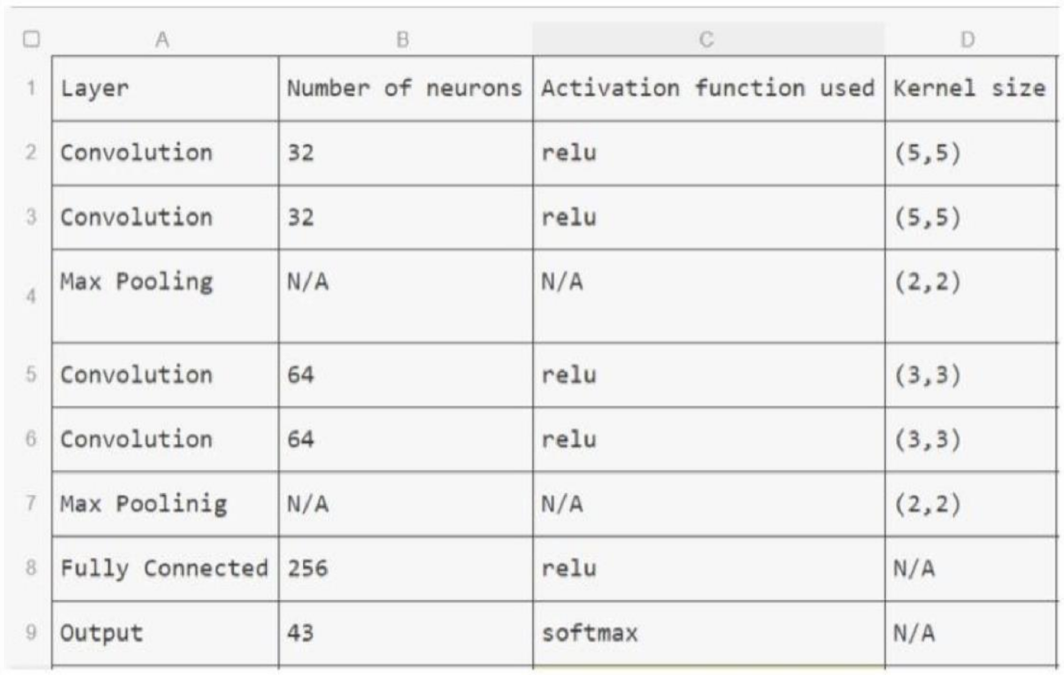


Then, we normalize the image using the formula (pixel - 128)/ 128 which converts the int values of each pixel [0,255] to float values with range [-1,1].

ARCHITECTURE:



ARCHITECTURE TABLE



Layer	Number of neurons	Activation function used	Kernel size
Convolution	32	relu	(5,5)
Convolution	32	relu	(5,5)
Max Pooling	N/A	N/A	(2,2)
Convolution	64	relu	(3,3)
Convolution	64	relu	(3,3)
Max Poolinig	N/A	N/A	(2,2)
Fully Connected	256	relu	N/A
Output	43	softmax	N/A

5 SCHEDULE,TASKS AND MILESTONES:

Get Dataset:

The German traffic signs detection dataset consists of 39209 images with 43 different classes. The images are distributed unevenly between those classes and

hence the model may predict some classes more accurately than other classes. We can populate the dataset with various image modifying techniques such as rotation, colour distortion or blurring the image. We will be training the model on the original dataset and will see the accuracy of the model. Then we will be adding more data and making each class even and check the model's accuracy.

Data Pre-Processing:

One of the limitations of the CNN model is that they cannot be trained on a different dimension of images. So, it is mandatory to have same dimension images in the dataset. We will check the dimension of all the images of the dataset so that we can process the images into having similar dimensions. In this dataset, the images have a very dynamic range of dimensions from 16*16*3 to 128*128*3 hence cannot be passed directly to the ConvNet model. We need to compress or interpolate the images to a single dimension. Not, to compress much of the data and not to stretch the image too much we need to decide the dimension which is in between and keep the image data mostly accurate. We had decided to use dimension 64*64*3. We will transform the image into the given dimension using opencv package.

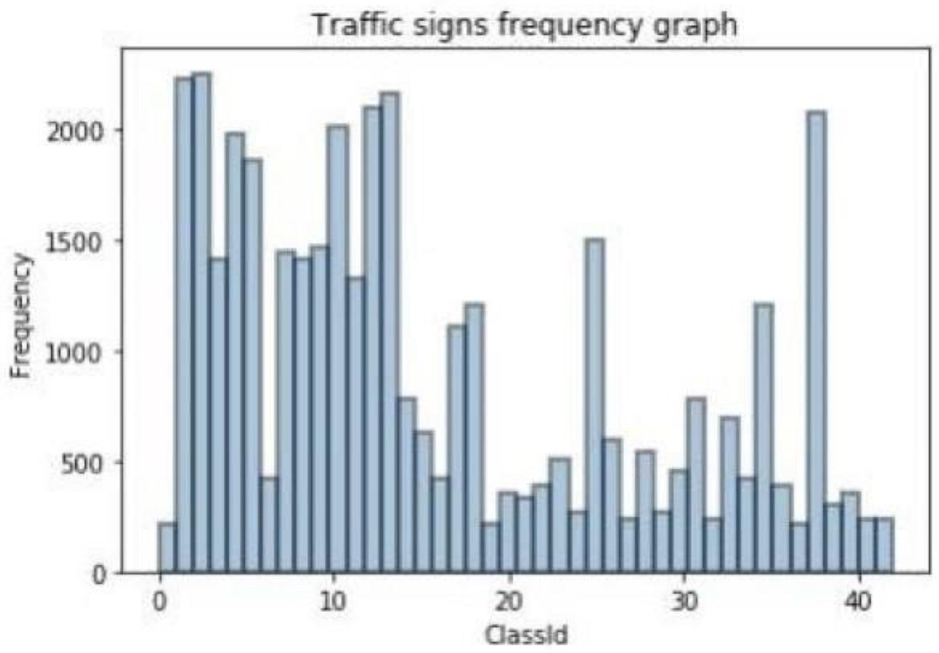
```
import cv2def resize_cv(img):
return cv2.resize(img, (64, 64), interpolation
= cv2.INTER_AREA)
```

cv2 is a package of opencv. the resize method transforms the image into the given dimension. Here, were transforming an image into the 64*64 dimensions. Interpolation will define what type of technique you want to use for stretching or for compressing the images. Opencv provides 5 types of interpolation techniques based on the method they use to evaluate the pixel values of the resulting image.

Data Loading:

Above we learned how we will pre-process the images. Now, we will load the dataset along with converting them in the decided dimension. The dataset consists of 43 classes total. In other words, 43 different types of traffic signs are present in that dataset and each sign has its own folder consisting of images in different sizes and clarity. Total 39209 numbers of images are present in the dataset. We can plot the histogram for the number of images present for different traffic signs.

```
import seaborn as sns
fig = sns.distplot(output, kde=False, bins = 43, hist = True,
hist_kws=dict(edgecolor="black",linewidth=2)) fig.set(title
= "Traffic signs frequency graph", xlabel = "ClassId", ylabel = "Frequency")
```



The dataset is loaded and now we need to divide it into a training and testing set. And, in the validation set. But if we divide directly then the model will not be trained on all the traffic signs as the dataset is not randomized. So, first we will randomize the dataset.

```
input_array = np.stack(list_images)import keras train_y
=keras.utils.np_utils.to_categorical(output)randomize
= np.arange(len(input_array)) np.random.shuffle(randomize) x
=  input_array[randomize]  y  =
train_y[randomize]
```

We can see that we have converted the output array to categorical output as the model will return in such a way. Now, splitting the dataset. We will split the dataset in 60:20:20 ratio as training, validation, test dataset respectively.

```
split_size = int(x.shape[0]*0.6) train_x, val_x = x[:split_size], x[split_size:] train1_y,
val_y = y[:split_size], y[split_size:]split_size = int(val_x.shape[0]*0.5) val_x, test_x =
val_x[:split_size], val_x[split_size:] val_y, test_y = val_y[:split_size], val_y[split_size:]
```

Training the model:

We have used the keras package.
Now we'll create a model with 4 Convolutional layers, 2 max-pooling layers.

Evaluating the model:

```
model.evaluate(test_x, test_y)
```

The model gets evaluated and you can find accuracy of 99%

Predicting the result:

```
pred = model.predict_classes(test_x)
```

6 PROJECT DEMONSTRATION:

SAMPLE CODE:

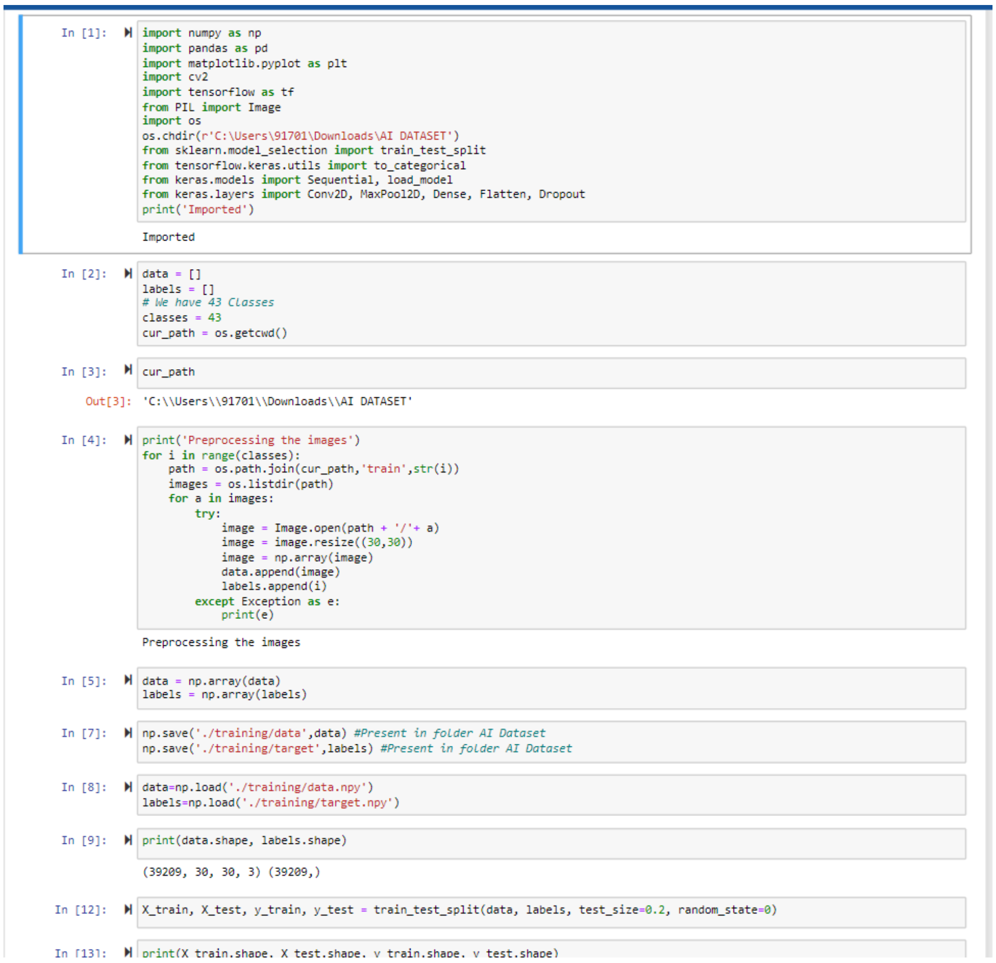
```
import numpy as np import
p a n d a s a s p d i m p o r t
matplotlib.pyplot as plt import
cv2 import tensorflow as tf
from PIL import Image import
os
os.chdir('/Users/aishaandatt/Downloads/Study/College/AI/PROJECT/AI Dataset')
from sklearn.model_selection import train_test_split from keras.utils import
to_categorical from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
print('Imported')


d a t a = [ ]
labels = []
# We have 43 Classes
c l a s s e s = 4 3
c u r _ p a t h =
os.getcwd()


print('Preprocessing the images')
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))
            i m a g e = n p . a r r a y ( i m a g e )
            d a t a . a p p e n d ( i m a g e )
            labels.append(i)
        except Exception as e:
            print(e)
```

```
data  = np.array(data)
labels = np.array(labels)


#os.mkdir('training')

np.save('./training/data',data) #Present in folder AI Dataset np.save('./training/
target',labels) #Present in folder AI Dataset


data=np.load('./training/data.npy') labels=np.load('./
training/target.npy')


print(data.shape, labels.shape)

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=0)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

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))
# We have 43 classes that's why we have defined 43 in the dense model.add(Dense(43,
activation='softmax'))

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

epochs = 20
```

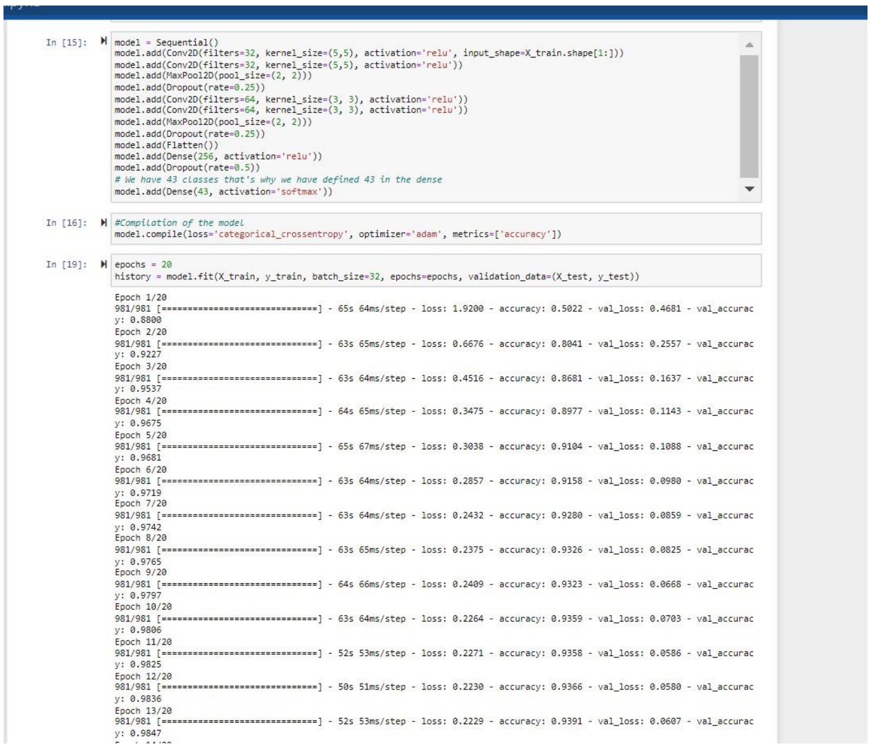
```
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs,
validation_data=(X_test, y_test))

#         a c c u r a c y
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()

# Loss
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()

def testing(testcsv):
    y_test   =   pd.read_csv(testcsv)
    label  =  y_test["ClassId"].values
    imgs    =    y_test["Path"].values
    data=[] for img in imgs:
        image    =    Image.open(img)
        i m a g e                    =
        im ag e.r e siz e((30,30))
        data.append(np.array(image
        ))
    X_test=np.array(data)
    return  X_test,label  X_test,
    label = testing('Test.csv')


Y_pred = model.predict_classes(X_test) Y_pred


from  sklearn.metrics  import  accuracy_score  print(accuracy_score(label,
Y_pred))



model.save("./training/TSR.h5")
```

```
import os
os.chdir(r'/Users/aishaandatt/Downloads/Study/College/AI/PROJECT/Backend/AI        Dataset')
from keras.models import load_model model = load_model('./training/TSR.h5')


# Classes of trafic signs classes =
{ 0:'Speed limit (20km/h)', 1:'Speed
limit (30km/h)', 2:'Speed limit (50km/
h)', 3:'Speed limit (60km/h)', 4:'Speed
limit (70km/h)', 5:'Speed limit (80km/
h)',
            6:'End of speed limit (80km/h)',
            7:'Speed limit (100km/h)',
            8:'Speed limit (120km/h)',
            9:'No passing',
            10:'No passing veh over 3.5 tons',
            11:'Right-of-way at intersection',
            12:'Priority road',
            13:'Yield', 14:'Stop',
            15:'No vehicles', 16:'Veh > 3.5
            tons prohibited', 17:'No entry',
            18:'General caution',
            19:'Dangerous curve left',
            20:'Dangerous curve right',
            21:'Double curve',
            22:'Bumpy road',
            23:'Slippery road',
            24:'Road narrows on the right',
            25:'Road work',
            26:'Traffic signals', 27:'Pedestrians',
            28:'Children crossing', 29:'Bicycles
            crossing',
            30:'Beware of ice/snow',
            31:'Wild animals crossing',
            32:'End speed + passing limits',
            33:'Turn right ahead',
            34:'Turn left ahead', 35:'Ahead
            only',
            36:'Go straight or right',
            37:'Go straight or left',
            38:'Keep right',
            39:'Keep left',
                40:'Roundabout mandatory',
                41:'End of no passing',
                42:'End no passing veh > 3.5 tons' }


from PIL import Image import
n u m p y a s n p i m p o r t
matplotlib.pyplot as plt def
test_on_img(img): data=[]
     image = Image.open(img)
     i m a g e                =
     im ag e.r e siz e((30,30))
     data.append(np.array(image
     )) X_test=np.array(data)
     Y_pred = model.predict_classes(X_test) return
     image,Y_pred


plot,prediction =
test_on_img(r'/Users/aishaandatt/Downloads/Study/College/AI/PROJECT/Backend/AI
Dataset/Test/00500.png') s = [str(i) for i in prediction] a = int("".join(s))
print("Predicted traffic sign is: ", classes[a])
plt.imshow(plot) plt.show()
```

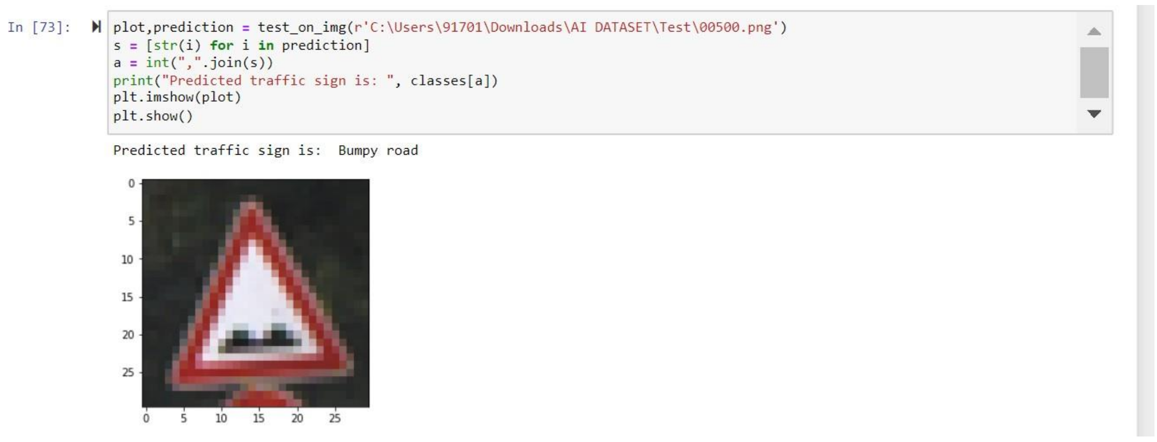
SCREENSHOTS:



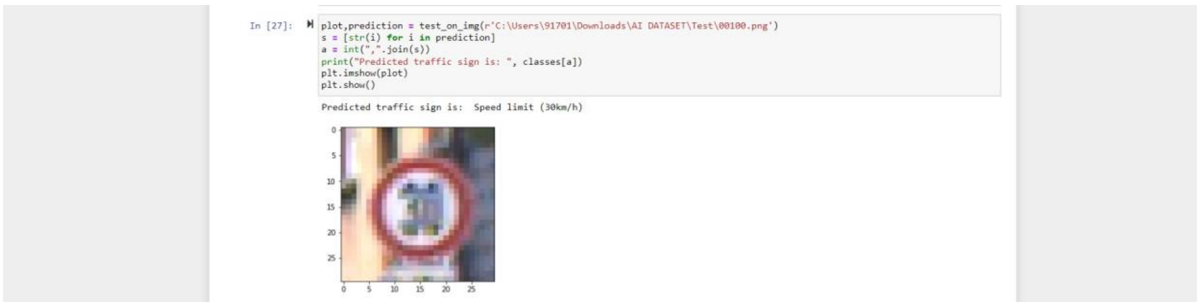
```
In [1]:  import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import cv2
         import tensorflow as tf
         from PIL import Image
         import os
         os.chdir(r'C:\Users\91701\Downloads\AI DATASET')
         from sklearn.model_selection 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
         print('Imported')

         Imported

In [2]:  data = []
         labels = []
         # We have 43 Classes
         classes = 43
         cur_path = os.getcwd()

In [3]:  cur_path

Out[3]:  'C:\\Users\\91701\\Downloads\\AI DATASET'

In [4]:  print('Preprocessing the images')
         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)
                     data.append(image)
                     labels.append(i)
                 except Exception as e:
                     print(e)

         Preprocessing the images

In [5]:  data = np.array(data)
         labels = np.array(labels)

In [7]:  np.save('./training/data',data) #Present in folder AI Dataset
         np.save('./training/target',labels) #Present in folder AI Dataset

In [8]:  data=np.load('./training/data.npy')
         labels=np.load('./training/target.npy')

In [9]:  print(data.shape, labels.shape)

         (39209, 30, 30, 3) (39209,)

In [12]: X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=0)

In [13]: print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```



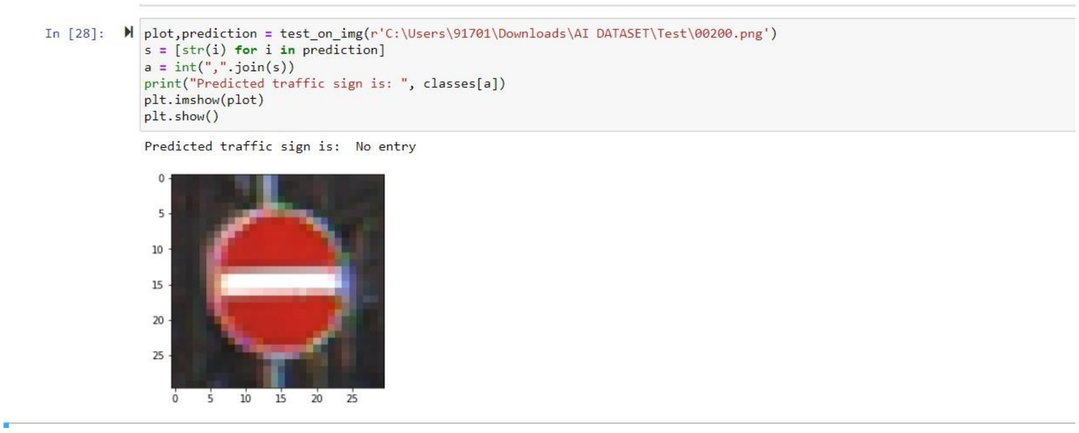
```
In [15]: model = Sequential()
         model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
         model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
         model.add(MaxPool2D(pool_size=(2, 2)))
         model.add(Dropout(rate=0.25))
         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))
         # We have 43 classes that's why we have defined 43 in the dense
         model.add(Dense(43, activation='softmax'))

In [16]: #Compilation of the model
         model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

In [19]: epochs = 20
         history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))

         Epoch 1/20
         981/981 [==============================] - 65s 64ms/step - loss: 1.9200 - accuracy: 0.5022 - val_loss: 0.4681 - val_accurac
         y: 0.8800
         Epoch 2/20
         981/981 [==============================] - 63s 65ms/step - loss: 0.6676 - accuracy: 0.8041 - val_loss: 0.2557 - val_accurac
         y: 0.9227
         Epoch 3/20
         981/981 [==============================] - 63s 64ms/step - loss: 0.4516 - accuracy: 0.8681 - val_loss: 0.1637 - val_accurac
         y: 0.9537
         Epoch 4/20
         981/981 [==============================] - 64s 65ms/step - loss: 0.3475 - accuracy: 0.8977 - val_loss: 0.1143 - val_accurac
         y: 0.9675
         Epoch 5/20
         981/981 [==============================] - 65s 67ms/step - loss: 0.3038 - accuracy: 0.9104 - val_loss: 0.1088 - val_accurac
         y: 0.9681
         Epoch 6/20
         981/981 [==============================] - 63s 64ms/step - loss: 0.2857 - accuracy: 0.9158 - val_loss: 0.0980 - val_accurac
         y: 0.9719
         Epoch 7/20
         981/981 [==============================] - 63s 64ms/step - loss: 0.2432 - accuracy: 0.9280 - val_loss: 0.0859 - val_accurac
         y: 0.9742
         Epoch 8/20
         981/981 [==============================] - 63s 65ms/step - loss: 0.2375 - accuracy: 0.9326 - val_loss: 0.0825 - val_accurac
         y: 0.9765
         Epoch 9/20
         981/981 [==============================] - 64s 66ms/step - loss: 0.2409 - accuracy: 0.9323 - val_loss: 0.0668 - val_accurac
         y: 0.9797
         Epoch 10/20
         981/981 [==============================] - 63s 64ms/step - loss: 0.2264 - accuracy: 0.9359 - val_loss: 0.0703 - val_accurac
         y: 0.9806
         Epoch 11/20
         981/981 [==============================] - 52s 53ms/step - loss: 0.2271 - accuracy: 0.9358 - val_loss: 0.0586 - val_accurac
         y: 0.9825
         Epoch 12/20
         981/981 [==============================] - 50s 51ms/step - loss: 0.2230 - accuracy: 0.9366 - val_loss: 0.0580 - val_accurac
         y: 0.9836
         Epoch 13/20
         981/981 [==============================] - 52s 53ms/step - loss: 0.2229 - accuracy: 0.9391 - val_loss: 0.0607 - val_accurac
         y: 0.9847
```



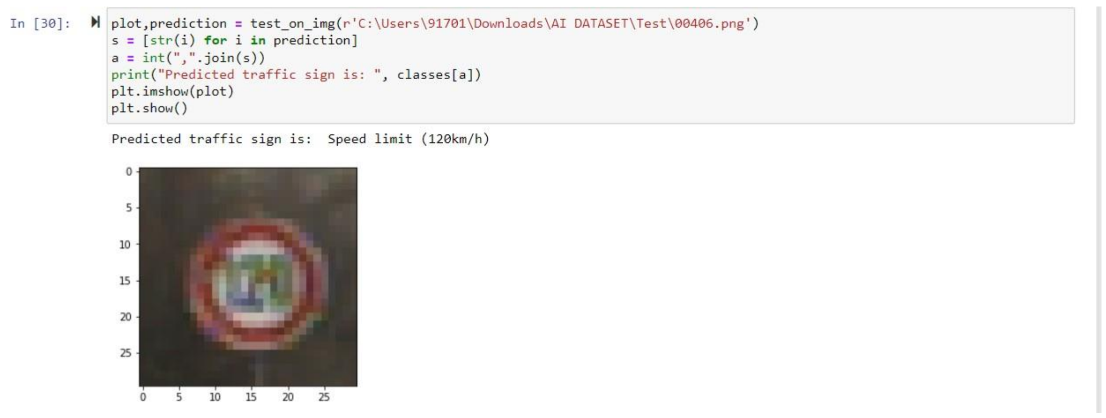
```
In [73]:  plot,prediction = test_on_img(r'C:\Users\91701\Downloads\AI DATASET\Test\00500.png')
          s = [str(i) for i in prediction]
          a = int(",".join(s))
          print("Predicted traffic sign is: ", classes[a])
          plt.imshow(plot)
          plt.show()

          Predicted traffic sign is:  Bumpy road
```



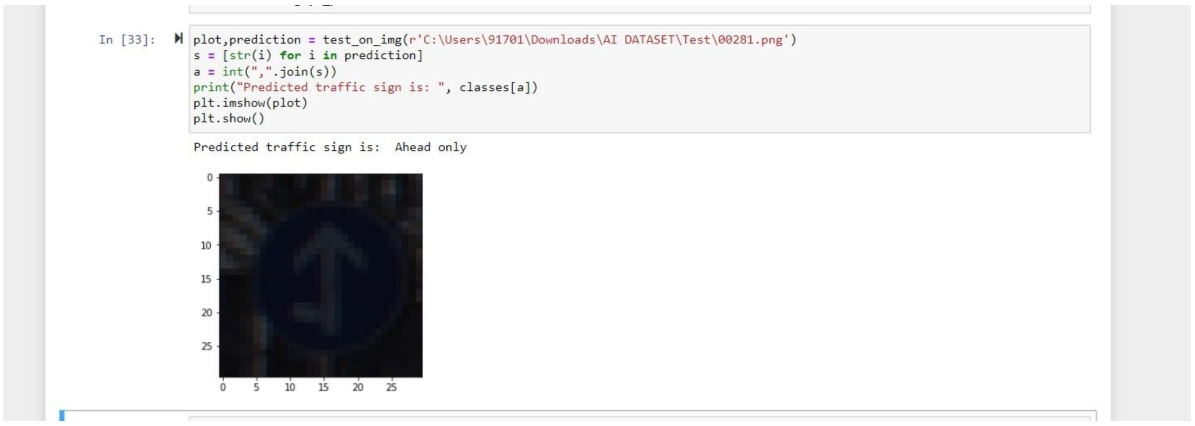
```
In [27]:  plot,prediction = test_on_img(r'C:\Users\91701\Downloads\AI DATASET\Test\00100.png')
          s = [str(i) for i in prediction]
          a = int(",".join(s))
          print("Predicted traffic sign is: ", classes[a])
          plt.imshow(plot)
          plt.show()

          Predicted traffic sign is:  Speed limit (30km/h)
```



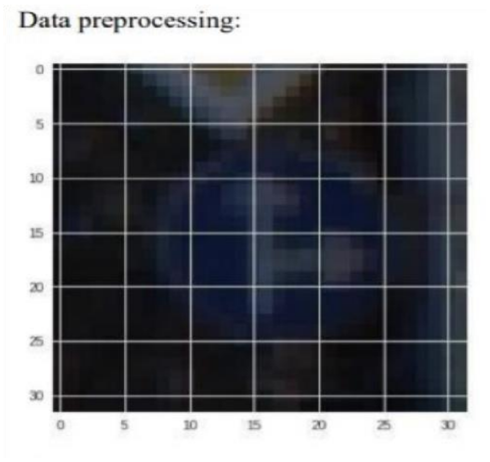
```
In [28]:  plot,prediction = test_on_img(r'C:\Users\91701\Downloads\AI DATASET\Test\00200.png')
          s = [str(i) for i in prediction]
          a = int(",".join(s))
          print("Predicted traffic sign is: ", classes[a])
          plt.imshow(plot)
          plt.show()

          Predicted traffic sign is:  No entry
```

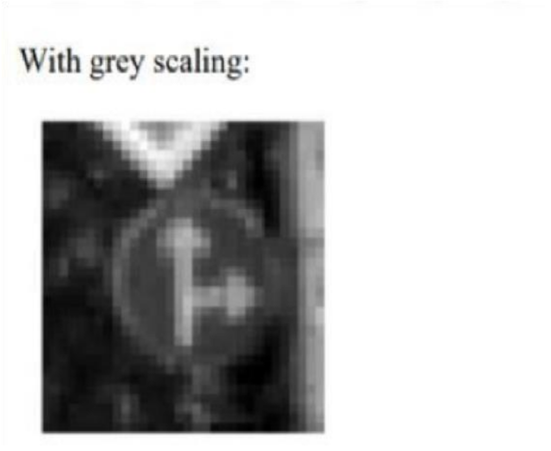


```
In [30]:   plot,prediction = test_on_img(r'C:\Users\91701\Downloads\AI DATASET\Test\00406.png')
           s = [str(i) for i in prediction]
           a = int(",".join(s))
           print("Predicted traffic sign is: ", classes[a])
           plt.imshow(plot)
           plt.show()

           Predicted traffic sign is:  Speed limit (120km/h)
```

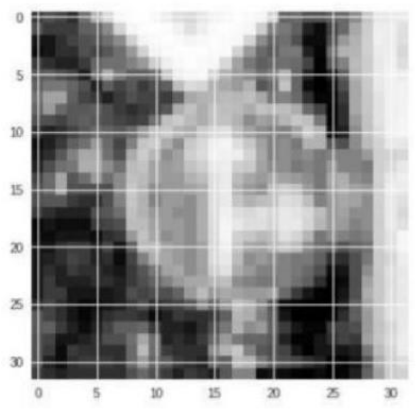


```
In [33]:   plot,prediction = test_on_img(r'C:\Users\91701\Downloads\AI DATASET\Test\00281.png')
           s = [str(i) for i in prediction]
           a = int(",".join(s))
           print("Predicted traffic sign is: ", classes[a])
           plt.imshow(plot)
           plt.show()

           Predicted traffic sign is:  Ahead only
```





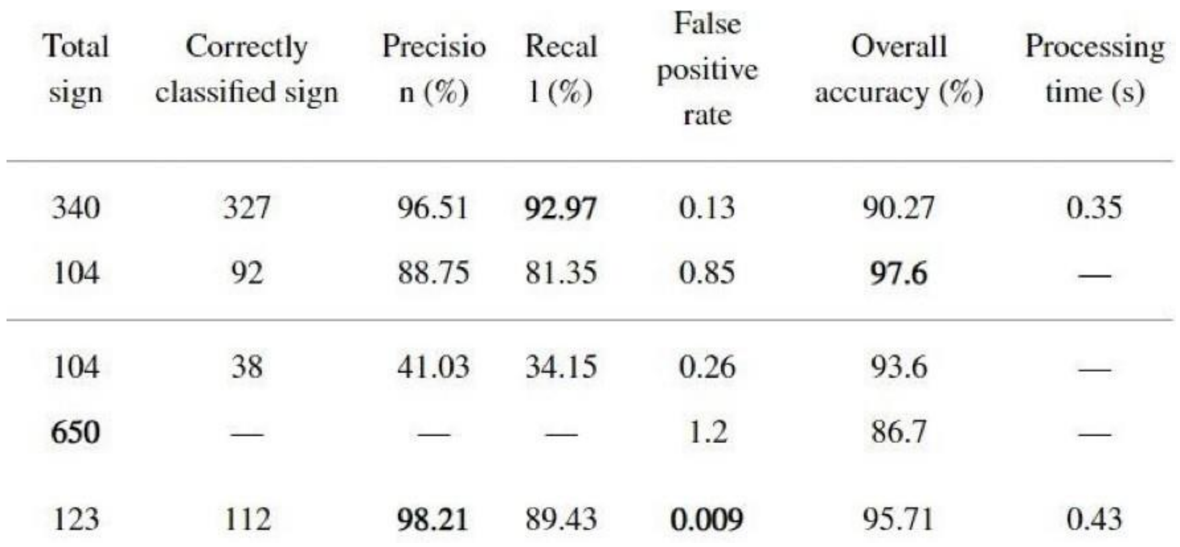
With equalizing:


0
5
10
15
20
25
30
0
5
10
15
20
25
30

7 COST ANALYSIS / RESULT & DISCUSSION:

This project gave a steep learning curve and for the first time, we experimented with different architecture of the Convolutional Neural Network. This motivated us to experiment more with the neural networks and think of new architecture also.

RESULT TABULATION



Total sign	Correctly classified sign	Precisio n (%)	Recal l (%)	False positive rate	Overall accuracy (%)	Processing time (s)
340	327	96.51	**92.97**	0.13	90.27	0.35
104	92	88.75	81.35	0.85	**97.6**	—
104	38	41.03	34.15	0.26	93.6	—
650	—	—	—	1.2	86.7	—
123	112	**98.21**	89.43	**0.009**	95.71	0.43

CONCLUSION

In detail, during the feature extraction, the starting step of this project is to load the data set, We have used the numpy library to calculate summary statistics of the traffic signs data set:The size of training set is 34799, The size of the validation set is 4410, The size of test set is 12630, The shape of a traffic sign image is (32, 32,
3), The number of unique classes/labels in the data set is 43.

After that we will explore, summarize and visualize the data set .Design, training and testing of a model architecture occurs after we have visualized the whole data set. Then, we will use the model to make predictions on new images and analyse the softmax probabilities of the new image.

8 SUMMARY:

We will use the German data set to import the images into this project. The starting step of this project is to load the data set, We have used the numpy library to calculate summary statistics of the traffic signs data set. The size of training set is 34799, The size of the validation set is 4410, The size of test set is 12630, The shape of a traffic sign image is (32, 32, 3), The number of unique classes/labels in the data set is 43. After that we will explore, summarize and visualize the data set. Design, training and testing of a model architecture occurs after we have visualized the whole data set. Then, we will use the model to make predictions on new images and analyse the softmax probabilities of the new image.
Finally, we test the model with a data set.

8 REFERENCES:

1. Soendoro W.D., Supriana I. Traffic sign recognition with Color-based Method, shape-arc estimation and SVM; Proceedings of the 2011 International Conference on Electrical Engineering and Informatics; Bandung, Indonesia. 17–19 July 2011.
2. Li H., Sun F., Liu L., Wang L. A novel traffic sign detection method via color segmentation and robust shape matching. Neurocomputing. 2015;169:77–88. doi:
10.1016/j.neucom.2014.12.111.
3. Bahlmann C., Zhu Y., Ramesh V., Pellkofer M., Koehler T. A system for traffic sign detection, tracking, and recognition using color, shape, and motion information; Proceedings of the 2005 IEEE Intelligent Vehicles Symposium; Las Vegas, NV, USA. 6–8 June 2005.
4. Ardianto S., Chen C., Hang H. Real-time traffic sign recognition using color segmentation and SVM; Proceedings of the 2017 International Conference on Systems, Signals and Image Processing (IWSSIP); Poznan, Poland. 22–24 May 2017.
5. Shadeed W.G., Abu-Al-Nadi D.I., Mismar M.J. Road traffic sign detection in color images; Proceedings of the 10th IEEE International Conference on Electronics, Circuits and Systems; Sharjah, UAE. 14–17 December 2003
6. Malik R., Khurshid J., Ahmad S.N. Road sign detection and recognition using colour segmentation, shape analysis and template matching; Proceedings of the 2007 International Conference on Machine Learning and Cybernetics; Hong Kong, China. 19–22 August 2007