JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

2020-2021

MINOR II

FINAL EVALUATION REPORT

AUTONOMOUS CAR USING DEEP LEARNING

GROUP MEMBERS -

AKASH SONI	**17103029**	**B 1**
SUYASH VERMA	**17103052**	**B 10**
VAIBHAV MISHRA	**17103346**	**B 9**

OUTLINE

PROBLEM STATEMENT

ABSTRACT

OBJECTIVES

TECHNOLOGY USED

PROJECT STRUCTURE

BACKGROUND STUDY AND FINDINGS

TRAFFIC SIGN RECOGNITION

LANE DETCTION SYSTEM

REFERENCES

PROBLEM STATEMENT

Recently the number of vehicles on the road has been increased enormously thanks to the technological achievement of the motor industry and very precisely the availability of motor vehicles at very low rates. With this remarkable growth of vehicles on the road there is an enormous increase in the road accidents as well and the ignorance of traffic signs is considered as the major cause. Nearly 1.25 million people dies of road accident yearly and about 20-50 million are injured or disabled.
DoT researchers estimates that fully autonomous vehicles could reduce the traffic fatalities by up to 94 percent by eliminating those accidents that are caused by Human errors. So millions of lives can be saved every year using Autonomous cars.
Cars with automated technology have sensors that never lose vigilance. "They are always looking for pedestrians. They are always looking for edge of the road. They are always watching the car in front. They don't become distracted or drunk, and I think that's really the main reason why most experts would say that there is definite possibility that automation can significantly reduce those Human error caused fatal crashes".
The biggest problem automated vehicles solve is that human beings are not well suited to travelling at high speeds. As speed increases, our time and distance perception degrades. We do not have sufficient range of lateral vision and no rear vision. We are limited in on ability to focus and process data and we tend to focus on one thing at a time (which leads to distraction from the task of driving). In short, we make mistakes behind the wheel and these mistakes are primary cause in more than 90% of accidents.

Abstract

For vehicles to be able to drive by themselves, they need to understands their surrounding world like human drivers, so that they can navigate their way in the streets, pause at traffic signs and traffic lights and avoid hitting obstacles such as others cars and pedestrians.
Autonomous Driving Car is one of the most disruptive innovations in AI. Fuelled by Deep Learning algorithms, they are continuously driving our society forward and creating new opportunities in the mobility sector. An autonomous car can go anywhere a traditional car can go and does everything that an experienced human driver does. But it’s very essential to train it properly. One of the many steps involved during the training of an autonomous driving car is lane detection.

Objectives

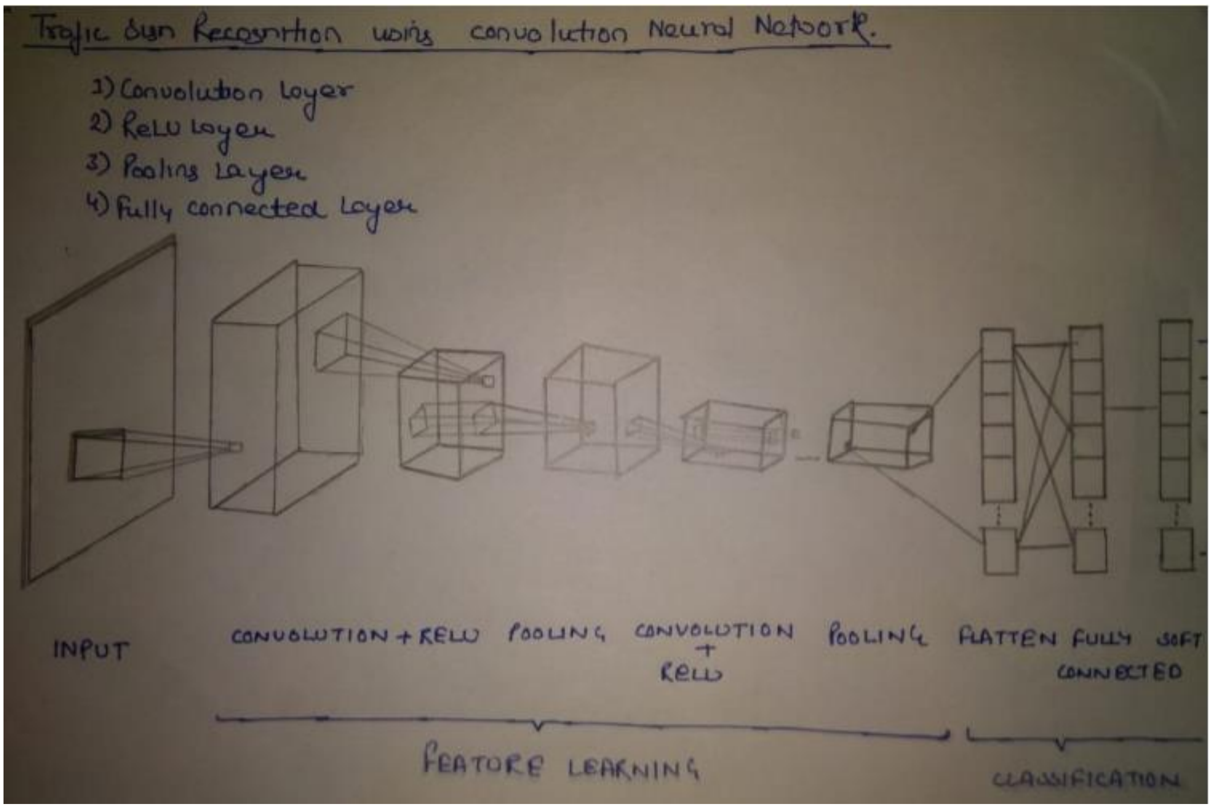
In our Minor Project II we are trying to solve above two problems of an Autonomous car

1. Traffic Sign Recognition and
2. Lane detection

In this Project we have implemented Traffic sign recognition system using Convolutional neural network and
Automatic Lane detection system using Computer Vision techniques via opencv

Background Study and Findings

CNN image classifications take an input image, process it and classify it under certain categories. Each input image will pass it through a series of convolution layers with filters. Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.

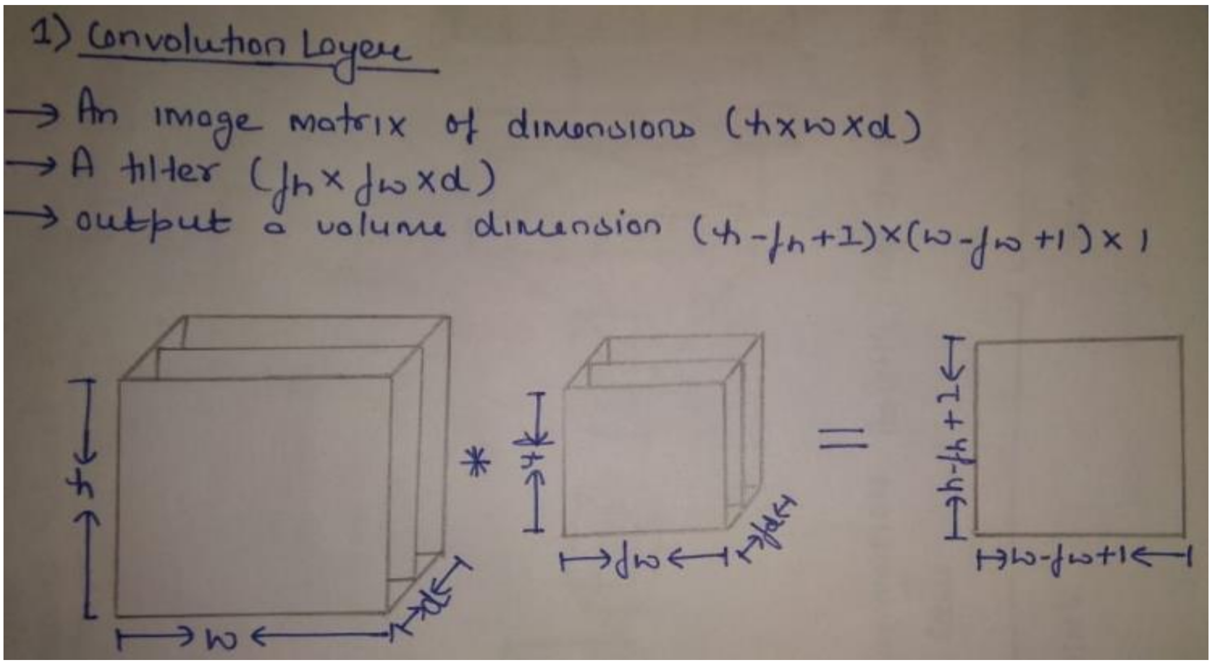


CNN have following layers –

1) Convolution Layer
2) ReLU Layer
3) Pooling Layer
4) Fully Connected Layer

Convolution Layer

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.



1) Convolution Layer

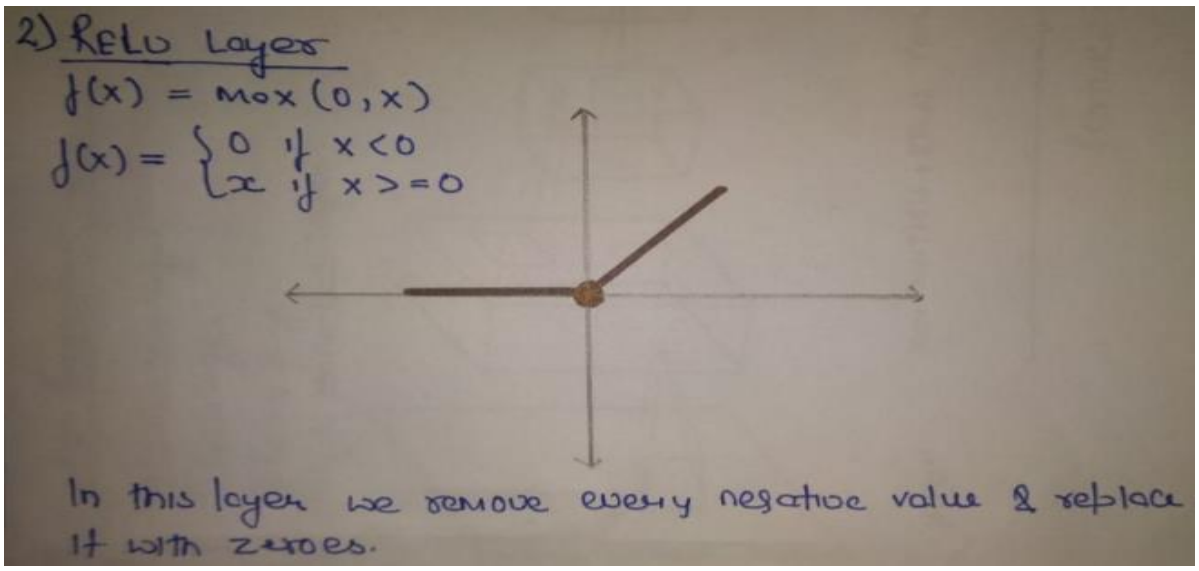
→ An image matrix of dimensions $(h \times w \times d)$

→ A filter $(f_h \times f_w \times d)$

→ output a volume dimension $(h - f_h + 1) \times (w - f_w + 1) \times 1$

ReLU Layer

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is (x) = max(0,x).



2) ReLu Layer

$f(x) = \max(0, x)$

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x >= 0 \end{cases}$$

In this layer we remove every negative value & replace it with zeroes.

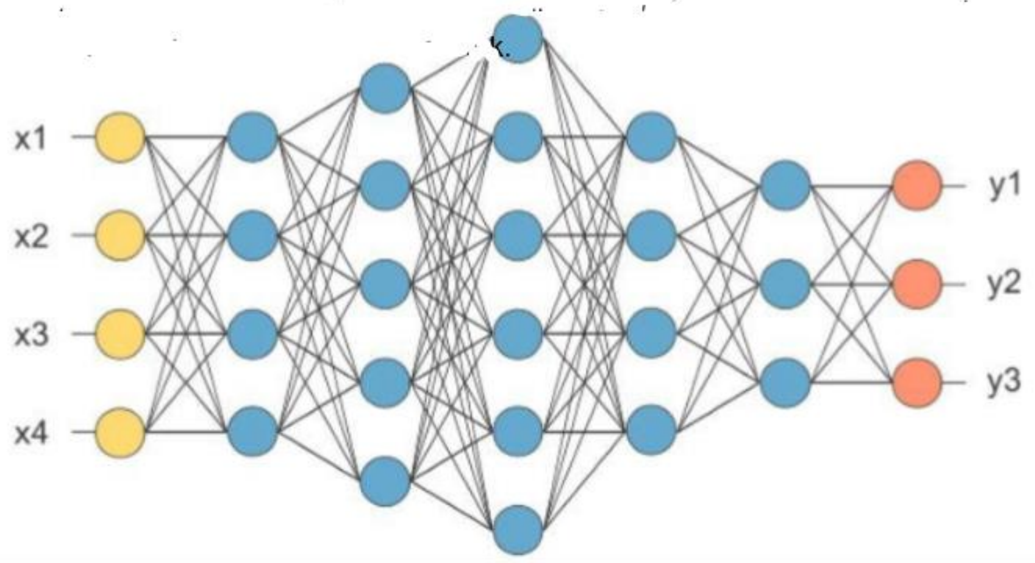
Pooling Layer

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling is also called sub sampling or down sampling which reduces the dimensionality of each map but retains important information.

Spatial pooling can be of different types:
Max pooling
Average pooling
Sum pooling

Fully Connected Layer

The layer we call the FC layer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network



Traffic Sign Recognition

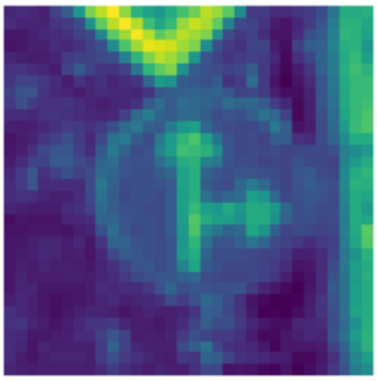
Preprocessing of Data

```
[ ]
    import cv2

    plt.imshow(X_train[1000])
    plt.axis("off")
    print(X_train[1000].shape)
    print(y_train[1000])
    def grayscale(img):
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        return img
```

```
(32, 32, 3)
36
```

```
[ ]
    img = grayscale(X_train[1000])
    plt.imshow(img)
    plt.axis("off")
    print(img.shape)
```

```
(32, 32)
```



```
    def equalize(img):
        img = cv2.equalizeHist(img)
        return img
    img = equalize(img)
    plt.imshow(img)
    plt.axis("off")
    print(img.shape)
```

```
[ ]
    def preprocess(img):
        img = grayscale(img)
        img = equalize(img)
        img = img/255
        return img
```

Preprocessing train, test and validation datasets

```
[ ]

     X_train = np.array(list(map(preprocess, X_train)))
     X_test = np.array(list(map(preprocess, X_test)))
     X_val = np.array(list(map(preprocess, X_val)))
```

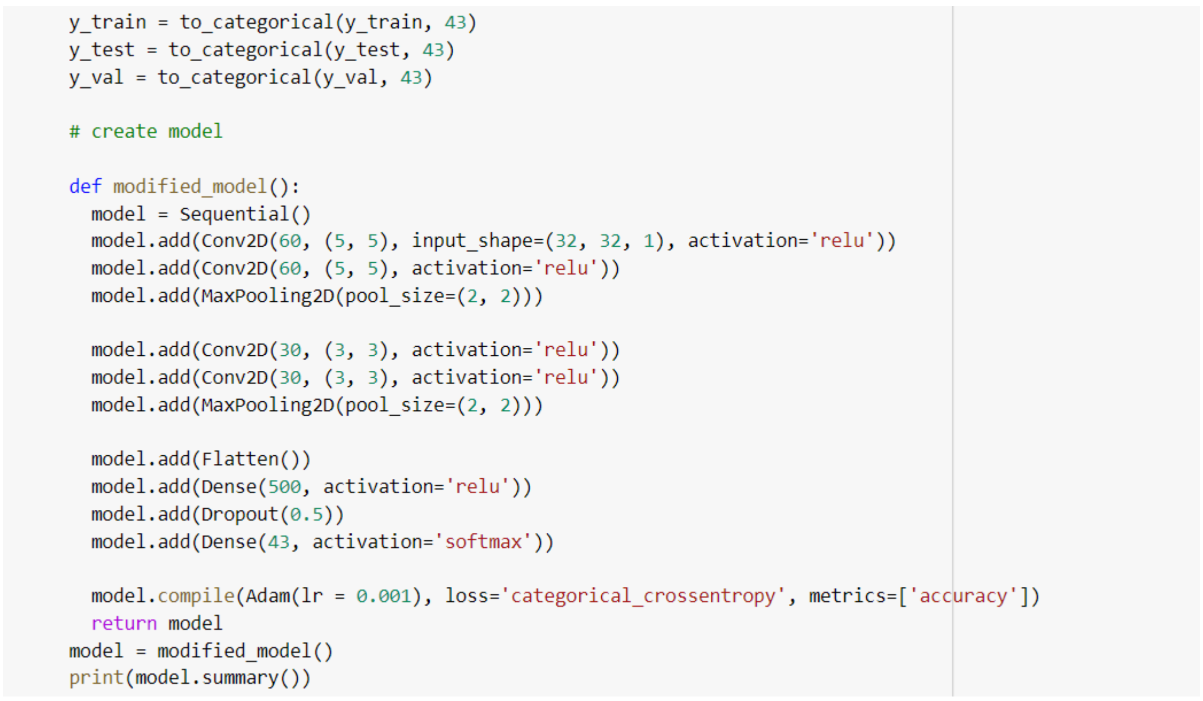
Reshaping images to 32*32*1

```
X_train = X_train.reshape(34799, 32, 32, 1)
X_test = X_test.reshape(12630, 32, 32, 1)
X_val = X_val.reshape(4410, 32, 32, 1)
```

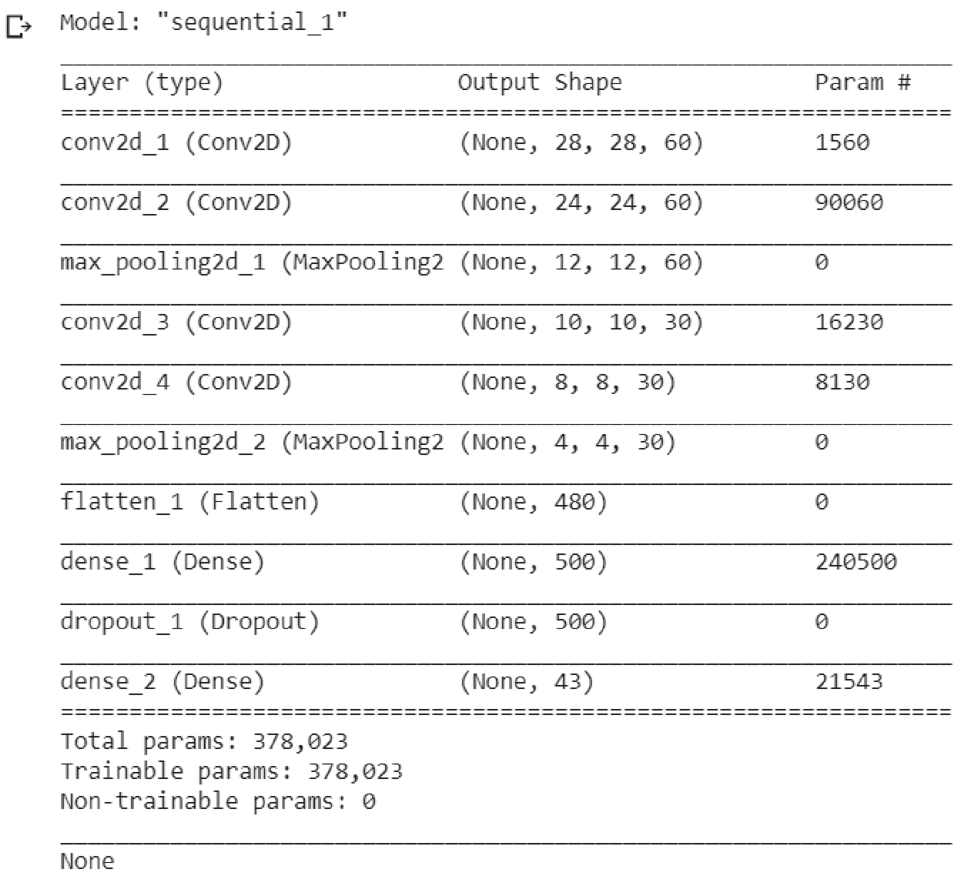
Image Data Generator

```
from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(width_shift_range=0.1,
                             height_shift_range=0.1,
                             zoom_range=0.2,
                             shear_range=0.1,
                             rotation_range=10.)

datagen.fit(X_train)
# for X_batch, y_batch in


batches = datagen.flow(X_train, y_train, batch_size = 15)
X_batch, y_batch = next(batches)

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

for i in range(15):
    axs[i].imshow(X_batch[i].reshape(32, 32))
    axs[i].axis("off")

print(X_batch.shape)
```

Modified Model of CNN



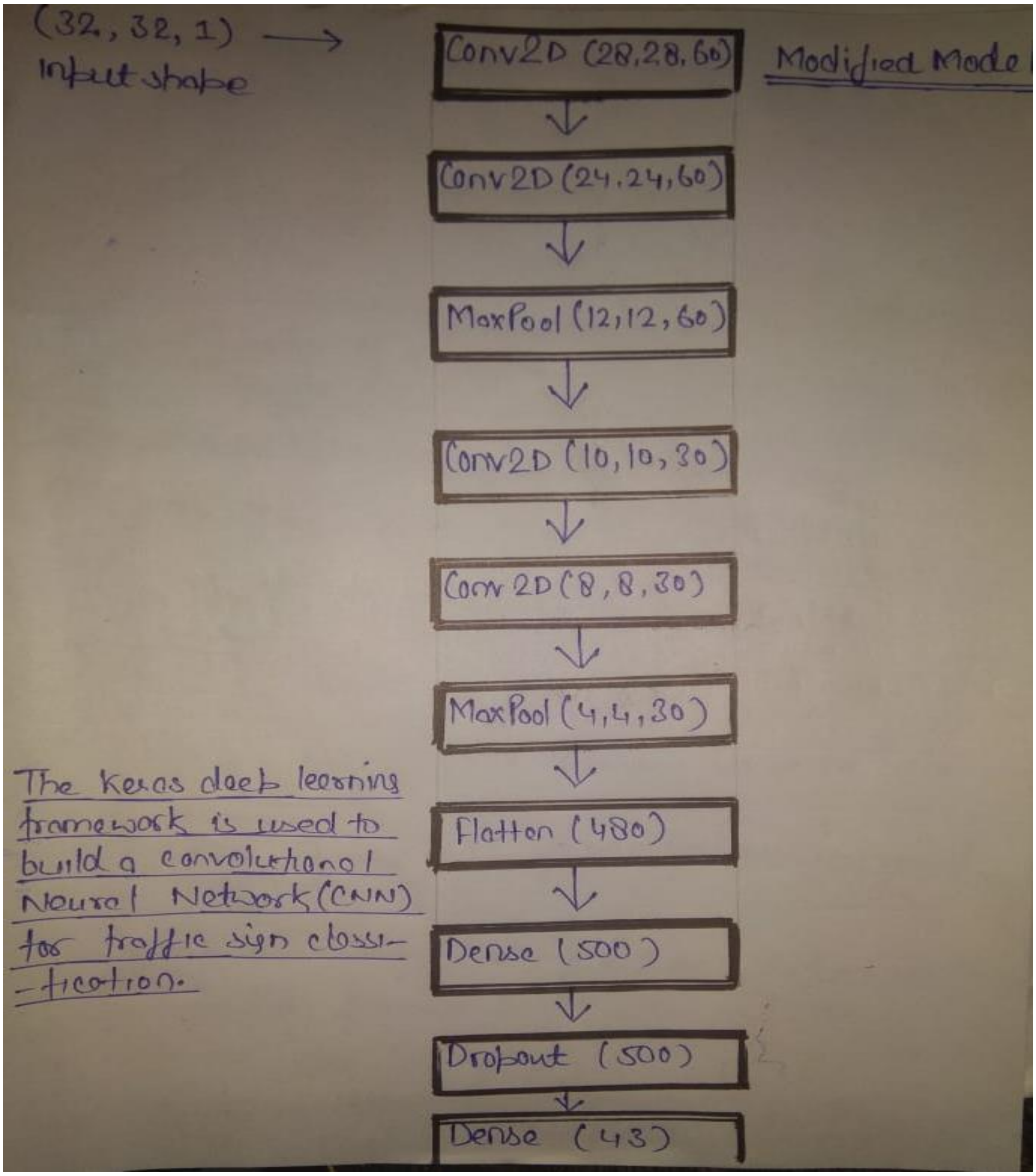
```
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)
y_val = to_categorical(y_val, 43)

# create model

def modified_model():
  model = Sequential()
  model.add(Conv2D(60, (5, 5), input_shape=(32, 32, 1), activation='relu'))
  model.add(Conv2D(60, (5, 5), activation='relu'))
  model.add(MaxPooling2D(pool_size=(2, 2)))

  model.add(Conv2D(30, (3, 3), activation='relu'))
  model.add(Conv2D(30, (3, 3), activation='relu'))
  model.add(MaxPooling2D(pool_size=(2, 2)))

  model.add(Flatten())
  model.add(Dense(500, activation='relu'))
  model.add(Dropout(0.5))
  model.add(Dense(43, activation='softmax'))

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



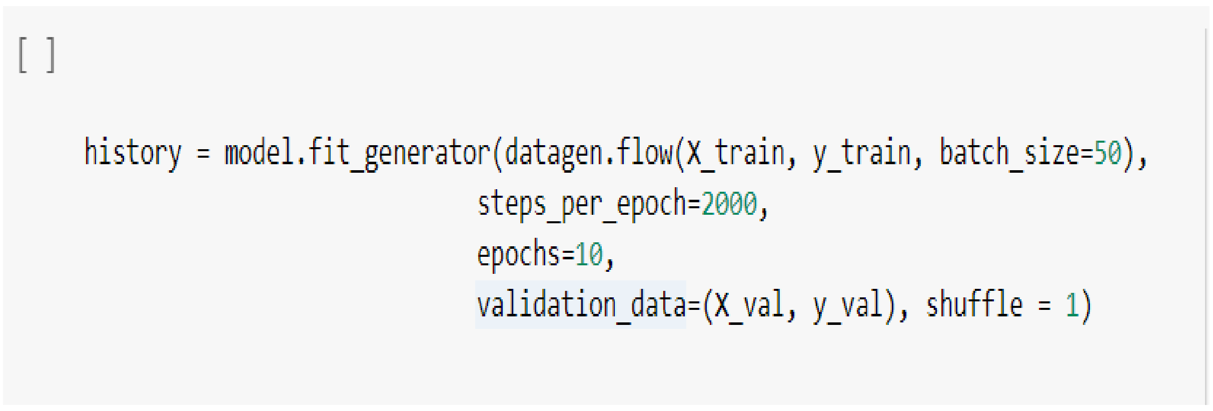
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 60)        1560
_________________________________________________________________
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
_________________________________________________________________
flatten_1 (Flatten)          (None, 480)               0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               240500
_________________________________________________________________
dropout_1 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 43)                21543
=================================================================
Total params: 378,023
Trainable params: 378,023
Non-trainable params: 0
_________________________________________________________________
None
```

Architecture of our Model

This set of layers uses a *5×5* kernel to learn larger features — it will help to distinguish between different traffic sign shapes and color blobs on the traffic signs themselves. The head of our network consists of sets of fully connected layers and a softmax classifier.



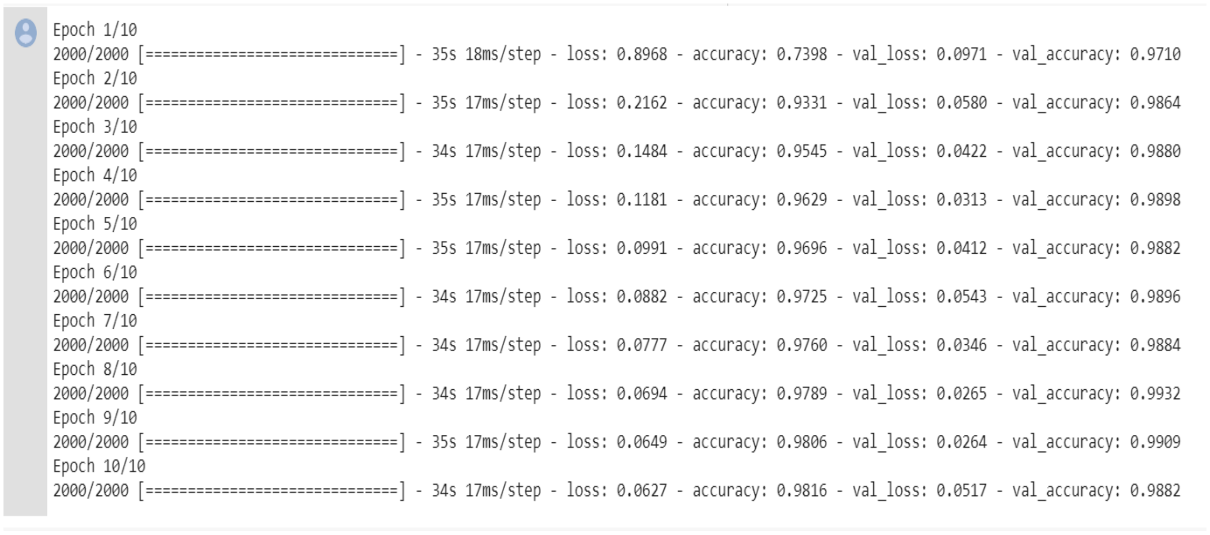
Training the Model



```
[ ]

    history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=50),
                                  steps_per_epoch=2000,
                                  epochs=10,
                                  validation_data=(X_val, y_val), shuffle = 1)
```

No of Epochs = 10

Steps per Epochs = 2000



```
Epoch 1/10
2000/2000 [==============================] - 35s 18ms/step - loss: 0.8968 - accuracy: 0.7398 - val_loss: 0.0971 - val_accuracy: 0.9710
Epoch 2/10
2000/2000 [==============================] - 35s 17ms/step - loss: 0.2162 - accuracy: 0.9331 - val_loss: 0.0580 - val_accuracy: 0.9864
Epoch 3/10
2000/2000 [==============================] - 34s 17ms/step - loss: 0.1484 - accuracy: 0.9545 - val_loss: 0.0422 - val_accuracy: 0.9880
Epoch 4/10
2000/2000 [==============================] - 35s 17ms/step - loss: 0.1181 - accuracy: 0.9629 - val_loss: 0.0313 - val_accuracy: 0.9898
Epoch 5/10
2000/2000 [==============================] - 35s 17ms/step - loss: 0.0991 - accuracy: 0.9696 - val_loss: 0.0412 - val_accuracy: 0.9882
Epoch 6/10
2000/2000 [==============================] - 34s 17ms/step - loss: 0.0882 - accuracy: 0.9725 - val_loss: 0.0543 - val_accuracy: 0.9896
Epoch 7/10
2000/2000 [==============================] - 34s 17ms/step - loss: 0.0777 - accuracy: 0.9760 - val_loss: 0.0346 - val_accuracy: 0.9884
Epoch 8/10
2000/2000 [==============================] - 34s 17ms/step - loss: 0.0694 - accuracy: 0.9789 - val_loss: 0.0265 - val_accuracy: 0.9932
Epoch 9/10
2000/2000 [==============================] - 35s 17ms/step - loss: 0.0649 - accuracy: 0.9806 - val_loss: 0.0264 - val_accuracy: 0.9909
Epoch 10/10
2000/2000 [==============================] - 34s 17ms/step - loss: 0.0627 - accuracy: 0.9816 - val_loss: 0.0517 - val_accuracy: 0.9882
```

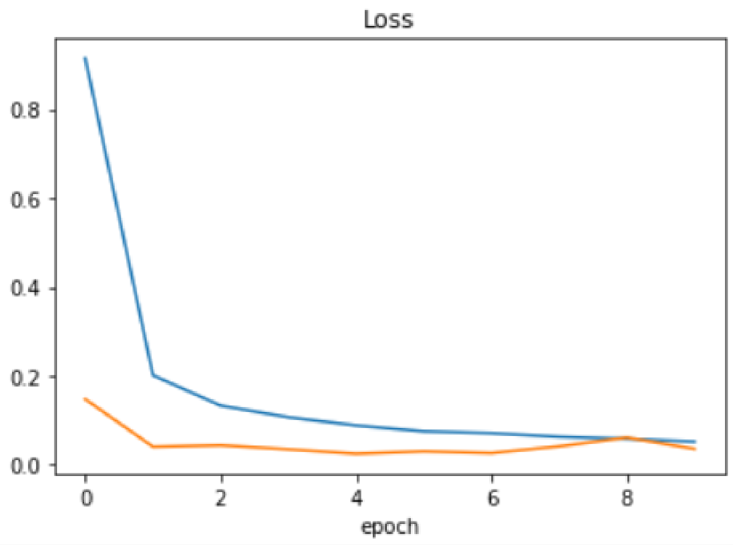
Val_accuracy = 0.9882

Val_loss = 0.0517

Graph of Loss and Accuracy

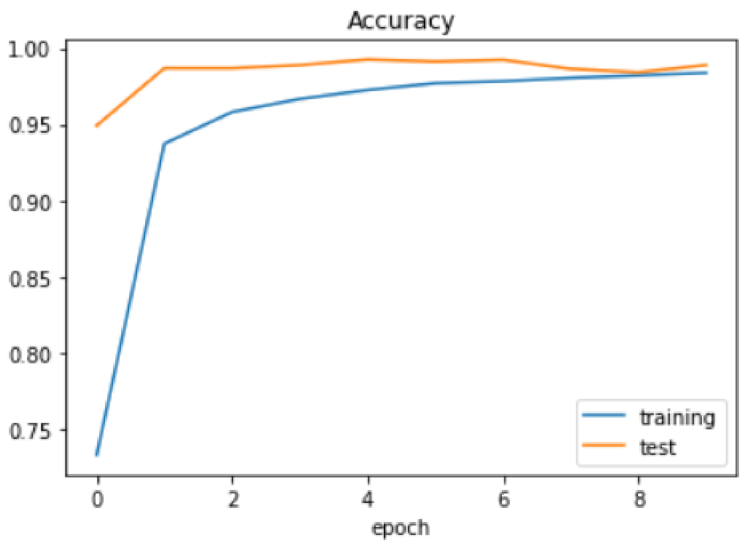
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss')
plt.xlabel('epoch')
```

```
Text(0.5, 0, 'epoch')
```

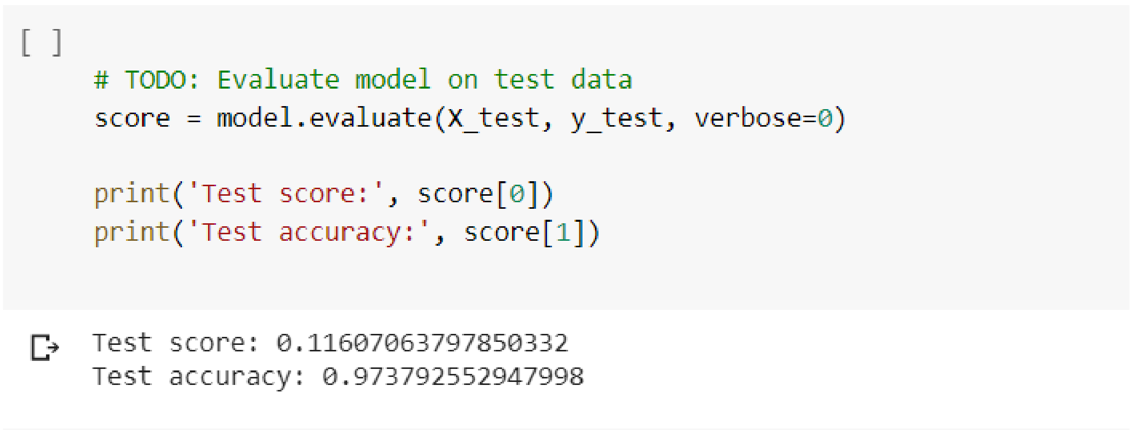


```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training','test'])
plt.title('Accuracy')
plt.xlabel('epoch')
```

```
Text(0.5, 0, 'epoch')
```



Accuracy of Test dataset

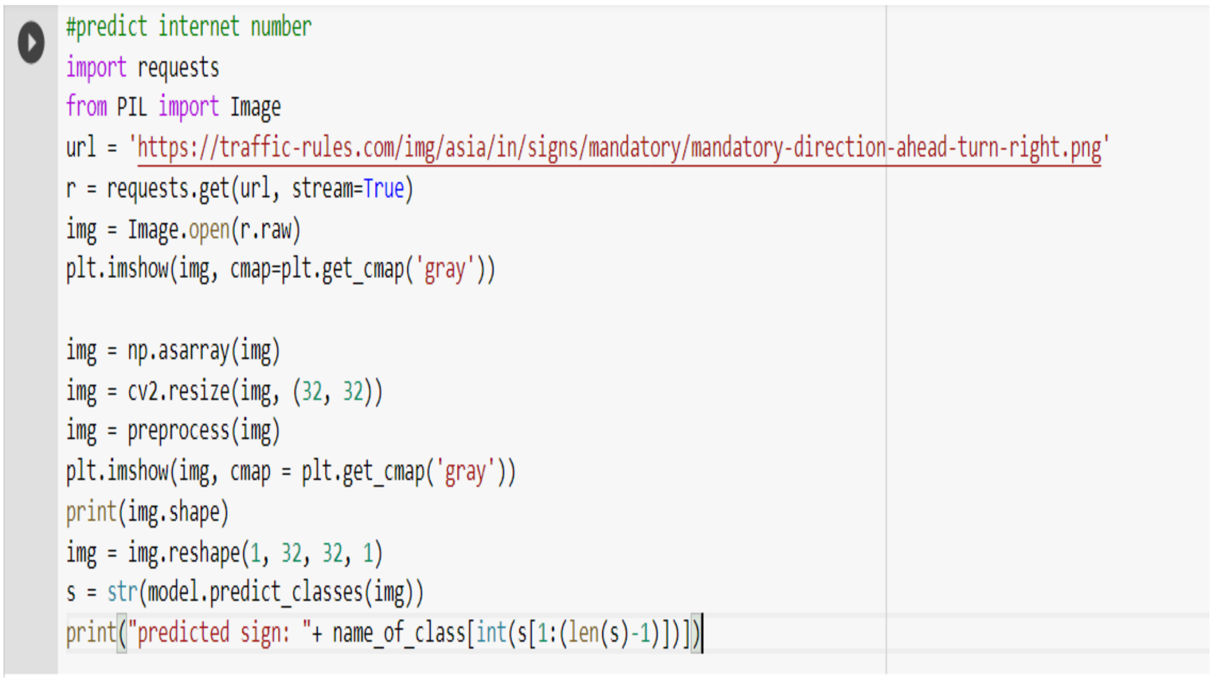


```
[ ]
    # TODO: Evaluate model on test data
    score = model.evaluate(X_test, y_test, verbose=0)

    print('Test score:', score[0])
    print('Test accuracy:', score[1])
```

```
Test score: 0.11607063797850332
Test accuracy: 0.973792552947998
```

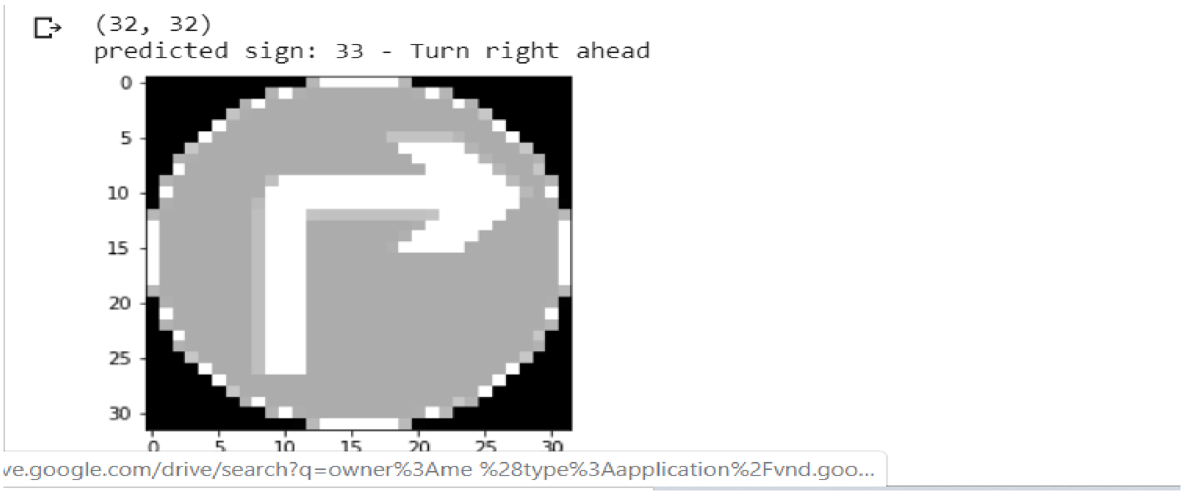
Accuracy = 97.37%

Input of an image -



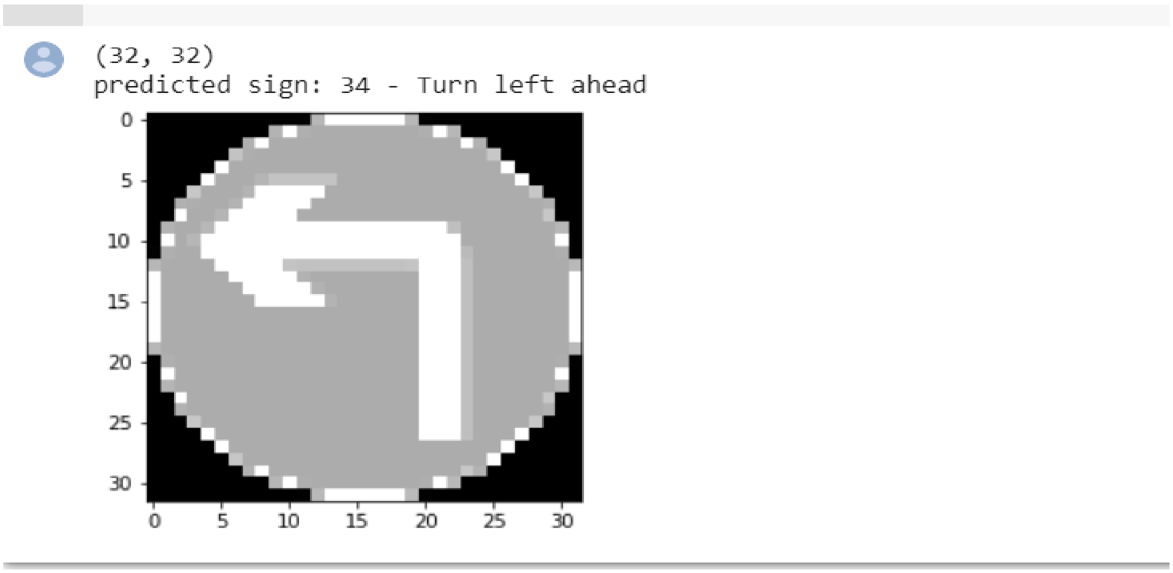
```
#predict internet number
import requests
from PIL import Image
url = 'https://traffic-rules.com/img/asia/in/signs/mandatory/mandatory-direction-ahead-turn-right.png'
r = requests.get(url, stream=True)
img = Image.open(r.raw)
plt.imshow(img, cmap=plt.get_cmap('gray'))

img = np.asarray(img)
img = cv2.resize(img, (32, 32))
img = preprocess(img)
plt.imshow(img, cmap = plt.get_cmap('gray'))
print(img.shape)
img = img.reshape(1, 32, 32, 1)
s = str(model.predict_classes(img))
print("predicted sign: "+ name_of_class[int(s[1:(len(s)-1)])])
```

Output



```
(32, 32)
predicted sign: 33 - Turn right ahead
```

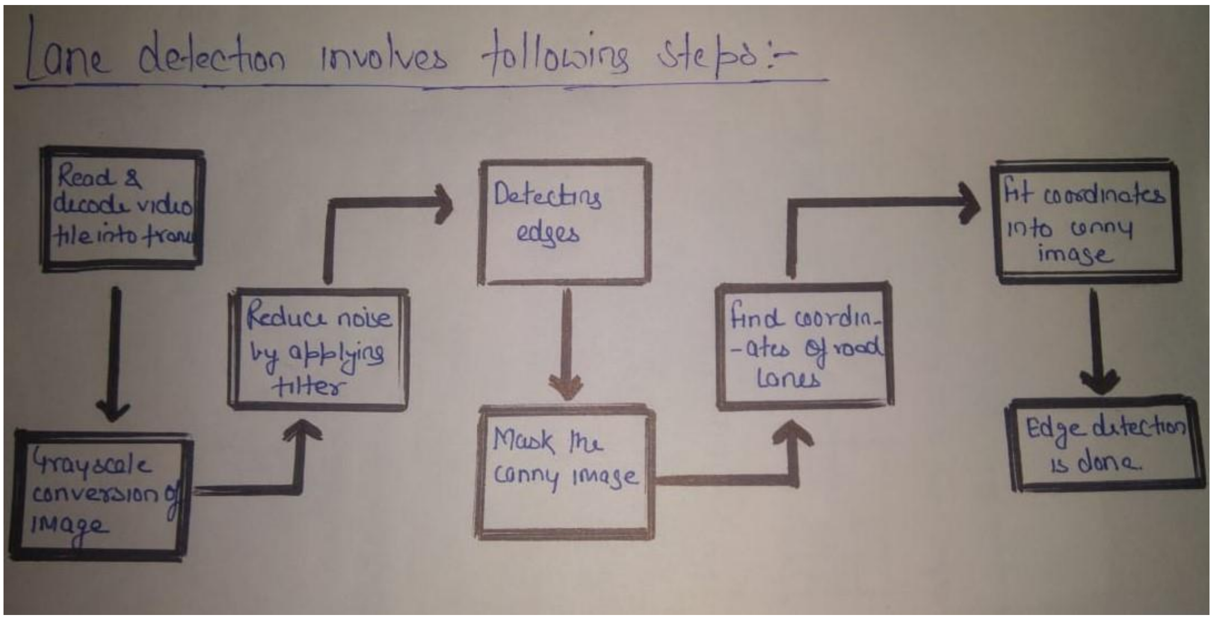
ve.google.com/drive/search?q=owner%3Ame %28type%3Aapplication%2Fvnd.goo...



```
(32, 32)
predicted sign: 34 - Turn left ahead
```

Lane detection using opencv

Autonomous Driving Car is one of the most disruptive innovations in AI. Fuelled by Deep Learning algorithms, they are continuously driving our society forward and creating new opportunities in the mobility sector. An autonomous car can go anywhere a traditional car can go and does everything that an experienced human driver does. But it’s very essential to train it properly. One of the many steps involved during the training of an autonomous driving car is lane detection.


Lane detection involves following steps:-
Read & decode video file into frame
Grayscale conversion of image
Reduce noise by applying filter
Detecting edges
Mask the canny image
Find coordinates of road lanes
Fit coordinates into canny image
Edge detection is done

- Decoding video file**:** After the capturing has been initialized every video frame is decoded (i.e. converting into a sequence of images).

- Grayscale conversion of image**:** The video frames are in RGB format, RGB is converted to grayscale because processing a single channel image is faster than processing a three-channel colored image.

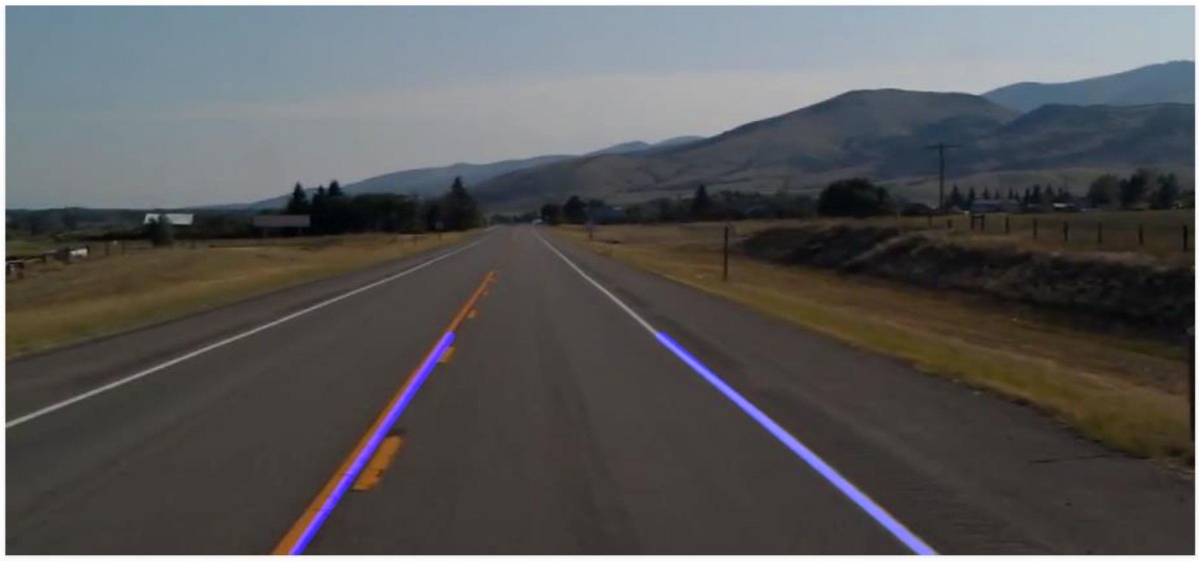
- Reduce noise**:** Noise can create false edges, therefore before going further, it’s imperative to perform image smoothening. Gaussian filter is used to perform this process.
- Canny Edge Detector: It computes gradient in all directions of our blurred image and traces the edges with large changes in intensity.

- Region of Interest**:** This step is to take into account only the region covered by the road lane. A mask is created here, which is of the same dimension as our road image. Furthermore, bitwise AND operation is performed between each pixel of our canny image and this mask. It ultimately masks the canny image and shows the region of interest traced by the polygonal contour of the mask.

- Hough Line Transform: The Hough Line Transform is a transform used to detect straight lines. The Probabilistic Hough Line Transform is used here, which gives output as the extremes of the detected lines.

Input :

Output :



Instead of letting Convolutional neural network choose its kernel, some predefined kernels used for image processing such as sharpening, edge-detecting, discrete cosine transformation and blurring were applied to the first layer of CNN. This concept was named as the General Filter Convolutional Neural Network (GFNN).
The results for this model GFCNN showed 30% less training time than that of CNN and the accuracy was higher compared to CNN despite the model being quicker.

Under the architecture, as the processing happens on each layer of convolutional neural network, the target area in the image becomes smaller.

The Canny edge detection technique

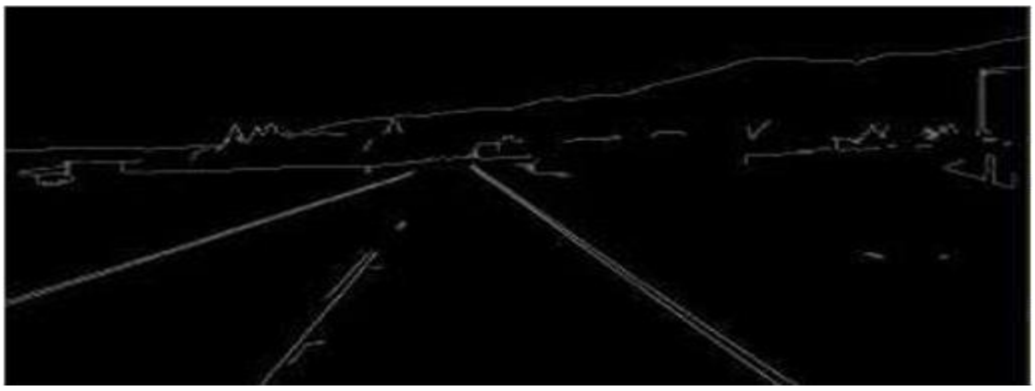
The Canny Detector is a multi-stage algorithm optimized for fast real-time edge detection. The fundamental goal of the algorithm is to detect sharp changes in luminosity (large gradients), such as a shift from white to black, and defines them as edges, given a set of thresholds.
An image is an array of pixel and pixel store data about intensity ranging from 0 to 255. Gradient denotes the change in brightness in a series of pixels.

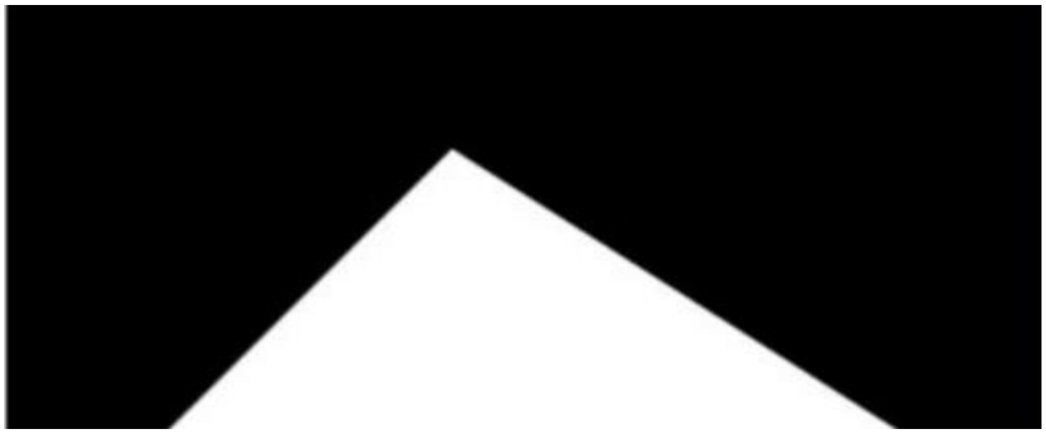
Gaussian Blur

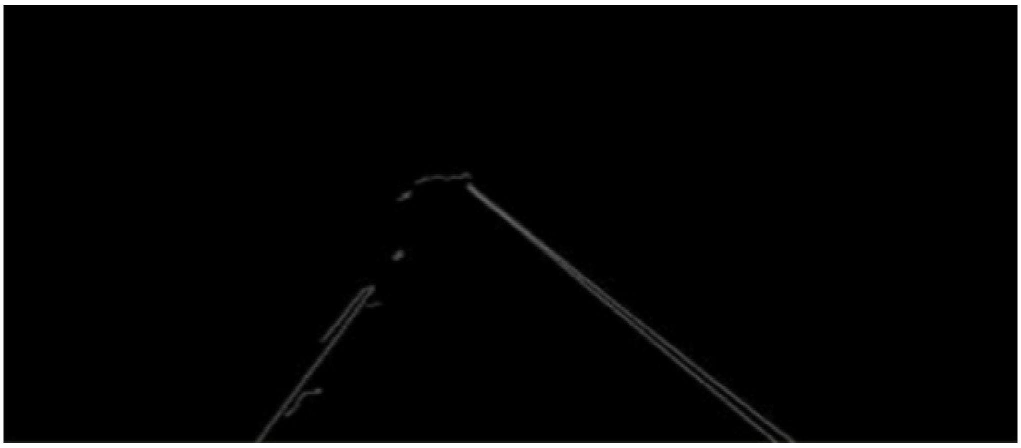
We have to smooth the image before processing it. It is done by modifying the value of the pixel by the average value of the pixel around it.

Edge Detection

An edge corresponds to the region in an array where there is sharp change in the intensity or color between adjacent pixels.

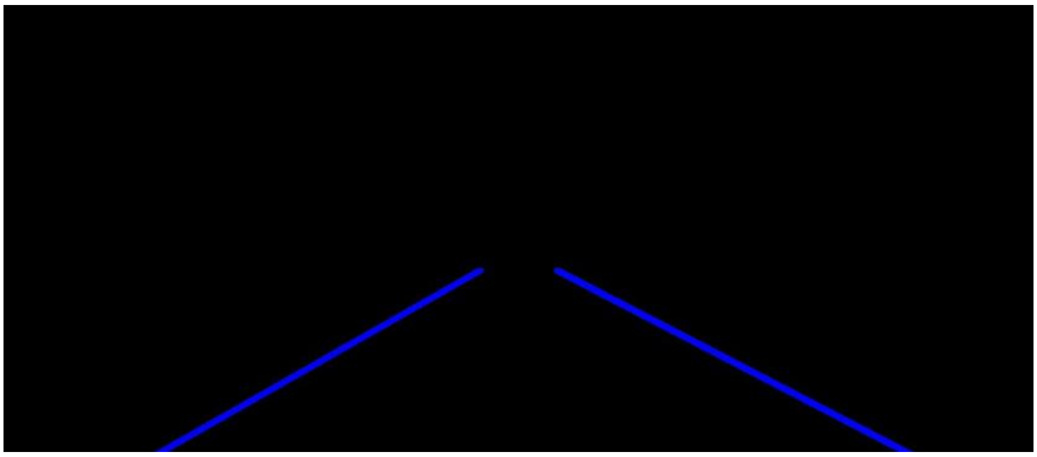




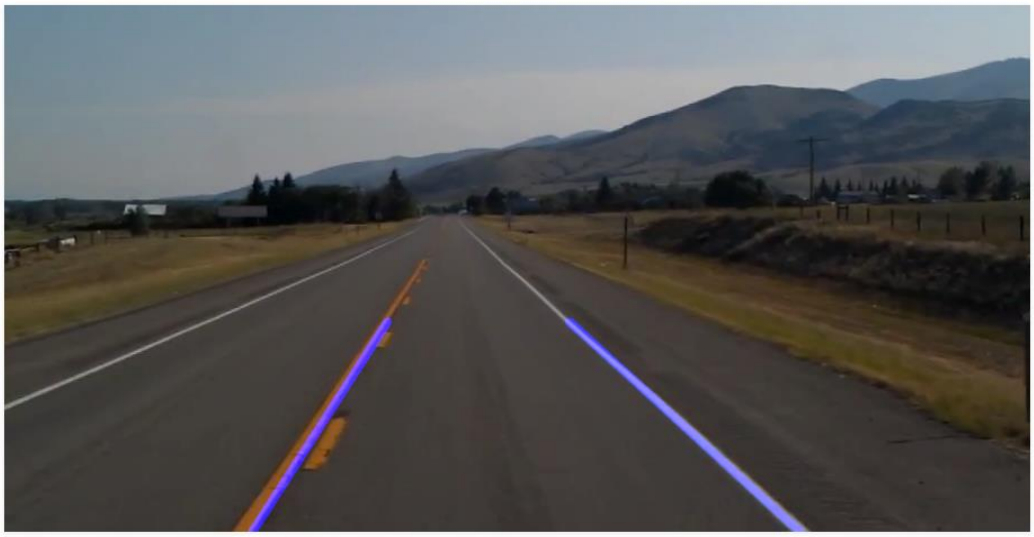


Hough Transform

Now we make use of Hough transform technique that will detect straight lines in the image and thus identify the lane lines.

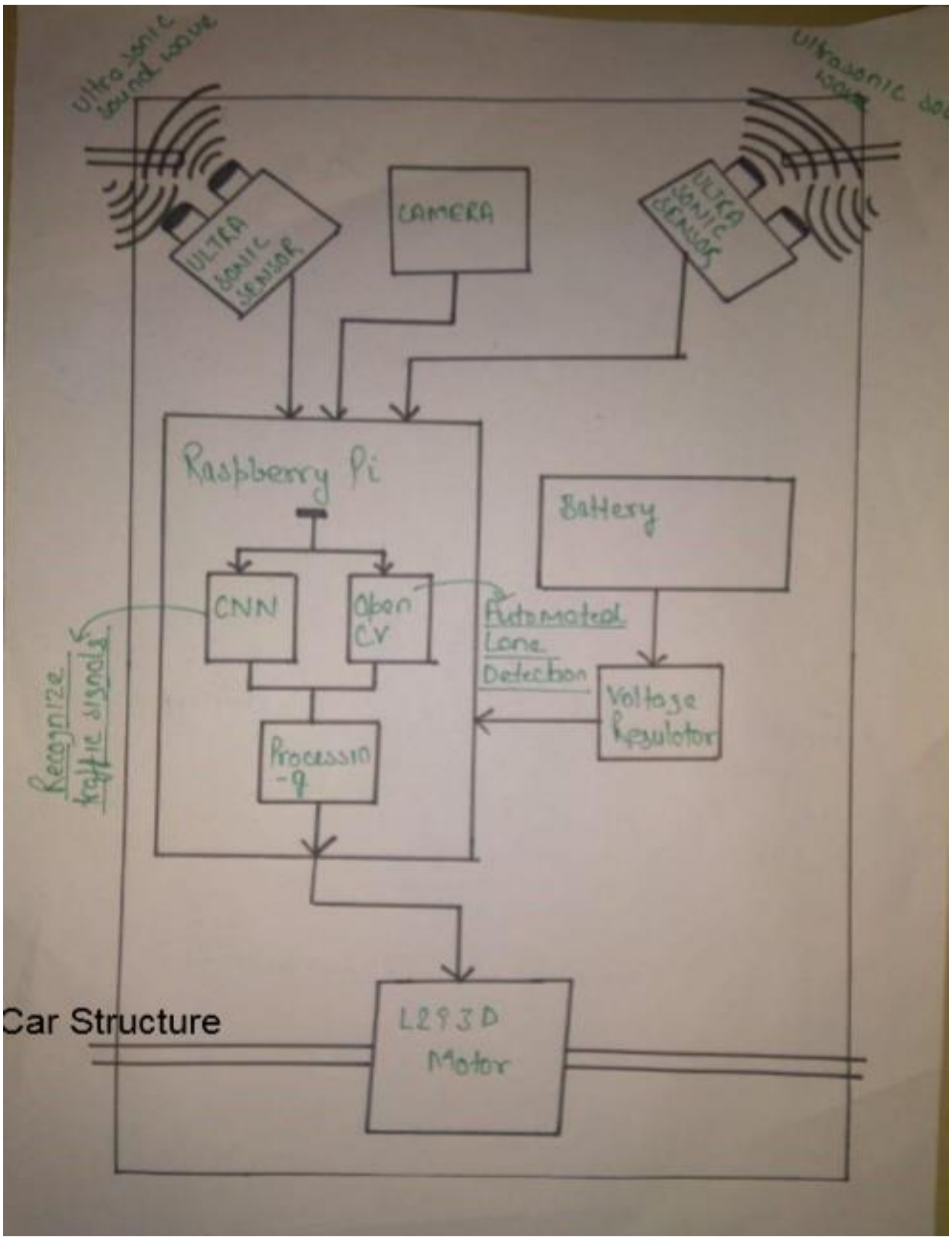


Output :



Conclusion

An autonomous car can be built with some extra hardware an, Traffic recognition system and Automatic Lane detection system integrated within Raspberry pi. Below is the Structure of a simple autonomous car.



References

1.https://www.researchgate.net/publication/276027654_Design_and_Implementation _of_Autonomous_Car_using_Raspberry_Pi

2.https://www.irjet.net/archives/V6/i3/IRJET-V6I3178.pdf

3.https://www.technoarete.org/common_abstract/pdf/IJERECE/v5/i4/Ext_56874.pdf

4.https://www.ijraset.com/fileserve.php?FID=13254

5.https://ijcsmc.com/docs/papers/February2014/V3I2201491.pdf

6.https://www.researchgate.net/publication/326831252_Traffic_signal_detection_and _classification_in_street_views_using_an_attention_model
dataset:https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german