Vellore-632014, Tamil Nadu, India

## School of Computer Science and Engineering

# Traffic Sign Classifier

Submitted by

**Daksh Mall (18BCE2079)**

## Contents

**Title**
**Aim**
**Abstract**
**Introduction**
**Literature Review (Related papers and study)**
**Code and Results**
**Conclusion**
**References**

# Traffic Sign Classifier

## Aim:

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 couple of other dependencies like NumPy and scikit-image. Also, we will be using keras package to build CNN model.

## Abstract:

A self-driving car, also known as an autonomous vehicle, connected and autonomous vehicle, full self-driving car or driverless car is a vehicle that is capable of sensing its environment and moving safely with little or no human input. They have great potential to become future of transportation. Recognizing traffic signs is one of the essential parts of the race to ultimate Self-Driving Car System and Deep Learning plays a big role in the development this system. In these recent years every car, autonomous or not, comes with an operating system installed in them to support various smart features. Our aim is to develop a program that will use the vehicle's visual sensors to capture traffic and road signals in real-time and predict what the sign conveys to improve the vehicle's performance. This program makes use of convolution neural network to achieve this goal. This model has convolution neural network code to classify over 43 traffic signs. This model can classify over 43 different traffic signs with a validation accuracy of around 97%. With importing data, data visualization, data prepossessing the data is loaded, verified and trained. The following data is then fed to neural networks and thus results are predicted. Hence, we start with the German traffic signal dataset. The images are distributed unevenly between the 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, color 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'll be adding more data and making each class even and check the model's accuracy.

*Figure 1. A few images from the German traffic signal dataset*

The German Traffic Sign dataset consists of 43 different traffic sign with each image having 32×32 px size. This dataset has 39,209 images as training data and 12,630 images as a test data. Each image is a photo of one of the 43 class of traffic sign e.g.

**Introduction:**

This program is indented to work on any vehicle with smart OS integrated in them. It has several modules working simultaneously to produce desired results. For this model to work efficiently we decided to create our own neural network and use it to train and test our model. Figure 2 shows the architecture diagram in a flowchart manner to explain the working of our program. Explanation for each module is given further in the paper.

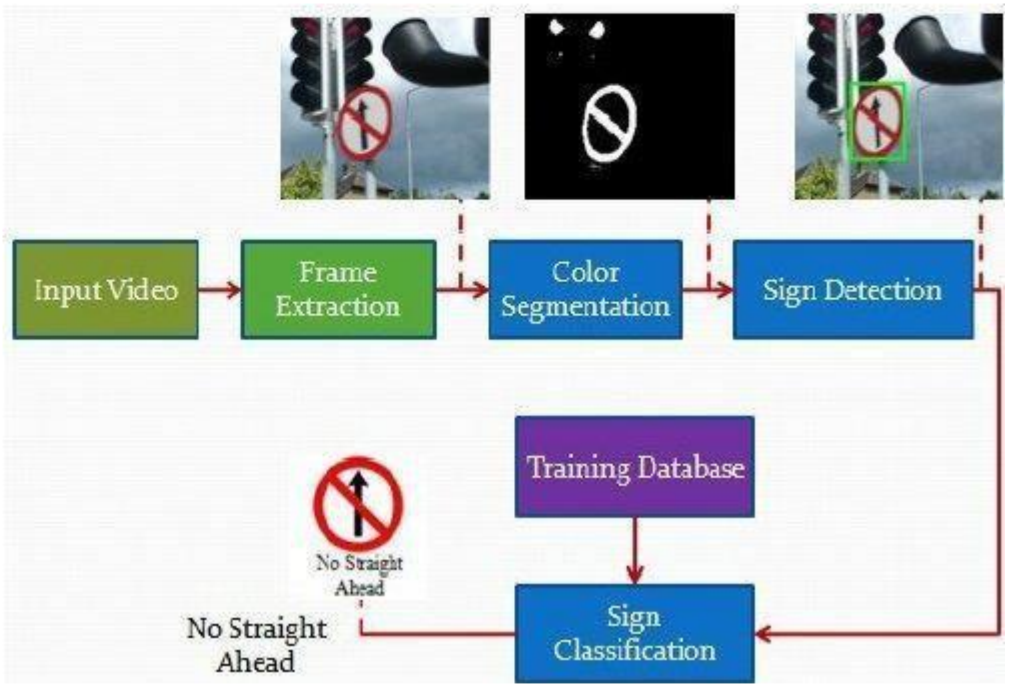


*Figure 2. Architecture Diagram*

Architecture Diagram:

For input, we will utilize the vehicle's visual sensors to capture an image of traffic sign in real-time. Frame extraction and color segmentation is the part where pre-processing of the input image is carried out. This is done to reduce the input's quality to increase prediction speed and accuracy of the model and to reduce load on the model. The model then identified the sign from the image and then uses the dataset to classify it and make a prediction.

**Proposed model (Explanation with diagram):**

We used LeNet as our base model and started playing with it. This is the point where we experimented a lot and tried to tune the parameter in the network. We made 4 convolutional layers and 2 max pooling layers which quite simple to understand. Playing with a number of convolutional filters made us learn more about CNN.

Figure 3 is a visual representation of the neural model we designed and decided to use for this program. The model has 4 convolution layers and 2 pooling layers. The input image is first preprocessed then passed to first 2 convolution layers and then to a max pooling layer. This process is repeated once again before dropping out the resultant images.

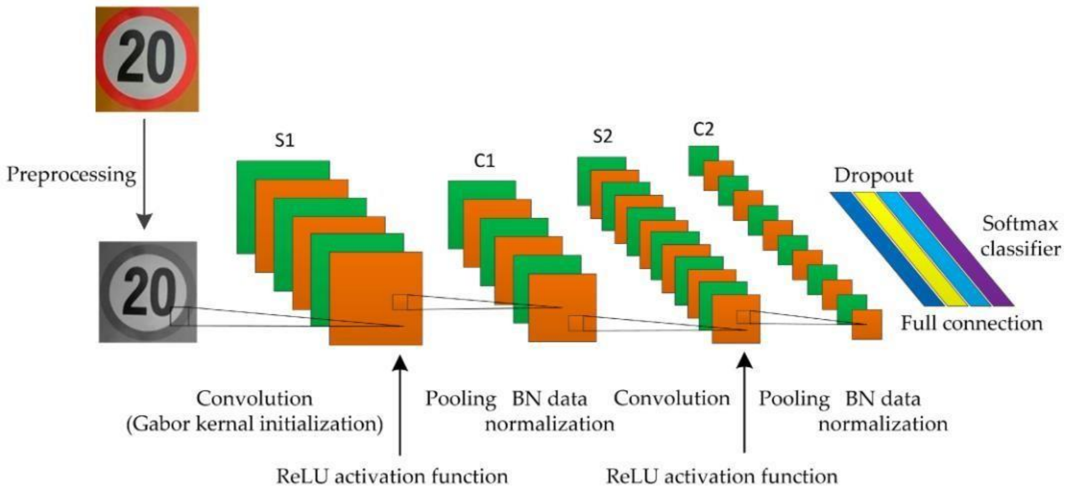


*Figure 3. Proposed model*

**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, color 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.

cv2 is a package of OpenCV. resize method transforms the image into the given dimension. Here, were transforming an image into the 64*64 dimension. 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. The techniques are `INTER_AREA, INTER_NEAREST, INTER_LINEAR, INTER_CUBIC, INTER_LANCZOS4` We’ll be using `INTER_NEAREST` interpolation technique it’s more preferred for image decimation but for extrapolation technique it’s similar as `INTER_NEAREST`. We could have used `INTER_CUBIC` but it requires high computation power so will be not using it.

**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 number of images are present in the dataset.

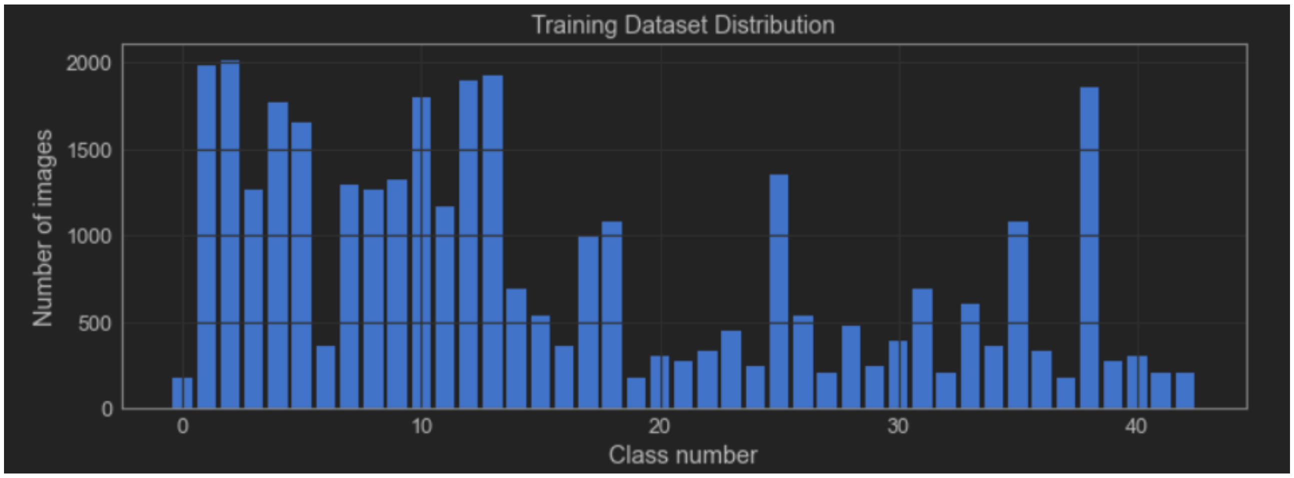


*Figure 4. Number of images present for different traffic signs*

The graph above (Figure 4) shows the number of images present in the dataset with respect to their ClassID. As mentioned earlier, this dataset consists of 43 different types of traffic signs. The horizontal axis of graph depicts these 43 different ClassIDs (0 to 42) and the vertical axis shows the number of images for each ClassID.

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

We 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.

**Training the model:**

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

**Neural network:**

To create our own neural network we've used the Dropout method. It drops some neurons from training and hence helps in preventing overfitting. Dropout(0.2) represents 20% of neurons to be dropped randomly in every cycle. It is also a regularization method.

**Evaluating the model:**

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

The model gets evaluated and you can find accuracy of 97%.

**Predicting the result:**

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

we can predict the class for each image and can verify how the model works.

## Literature Review:

EFFICIENTNET: RETHINKING MODEL SCALING FOR CONVOLUTIONAL NEURAL NETWORKS, BY MINGXING TAN AND QUOC V. LE –
The authors show that if just one of these parameters is scaled up, or if the parameters are all scaled up arbitrarily, this leads to rapidly diminishing returns relative to the extra computational power needed. Instead, they demonstrate that there is an optimal ratio of depth, width, and resolution in order to maximize efficiency and accuracy. This is called **compound scaling**. The result is that EfficientNet's performance surpasses the accuracy of other CNNs on ImageNet by up to 6% while being up to ten times more efficient in terms of speed and size.
https://www.researchgate.net/publication/333444574_EfficientNet_Rethinking_Model_Scaling_for_Convolutional_Neural_Networks

REASONING-RCNN: UNIFYING ADAPTIVE GLOBAL REASONING INTO LARGE-SCALE OBJECT DETECTION, BY HANG XU, CHENHAN JIANG, XIAODAN LIANG, LIANG LIN, ZHENGUO LI –
The researchers introduce a simple global reasoning framework, **Reasoning-RCNN**, which explicitly incorporates multiple kinds of commonsense knowledge and also propagates visual information globally from all the categories. The experiments demonstrate that the proposed method significantly outperforms current state-of-the-art object detection methods on the VisualGenome, ADE, and COCO benchmarks.
https://ieeexplore.ieee.org/abstract/document/8953842

FIXING THE TRAIN-TEST RESOLUTION DISCREPANCY, BY HUGO TOUVRON, ANDREA VEDALDI, MATTHIJS DOUZE, HERVÉ JÉGOU –
This paper first shows that existing augmentations induce a significant discrepancy between the typical size of the objects seen by the classifier at train and test time. We experimentally validate that, for a target test resolution, using a lower train resolution offers better classification at test time.
https://www.researchgate.net/publication/340021973_Fixing_the_train-test_resolution_discrepancy_FixEfficientNet

SINGAN: LEARNING A GENERATIVE MODEL FROM A SINGLE NATURAL IMAGE, BY TAMAR ROTT SHAHAM, TALI DEKEL, TOMER MICHAELI –
The researchers from Technion and Google Research introduce **SinGAN**, a new model for the unconditional generation of high-quality images given a **single natural image**. Their approach is based on the notion that the internal statistics of patches within a single image are usually sufficient for learning a powerful generative model.
https://ieeexplore.ieee.org/abstract/document/9008787

LOCAL AGGREGATION FOR UNSUPERVISED LEARNING OF VISUAL EMBEDDINGS, BY CHENGXU ZHUANG, ALEX LIN ZHAI, DANIEL YAMINS –
The research team from Stanford University addresses the problem of object detection and recognition with unsupervised learning. To tackle this problem, they introduce the **Local Aggregation (LA)** procedure, which causes dissimilar inputs to move apart in the embedding space while allowing similar inputs to converge into clusters.
https://www.researchgate.net/publication/332110441_Local_Aggregation_for_Unsupervised_Learning_of_Visual_Embeddings

**Code and Result:**

**Screenshots (all modules):**

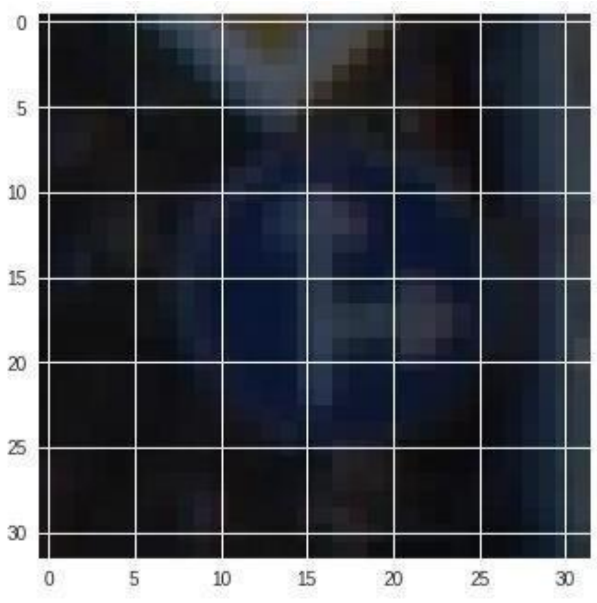


*Figure 5. Data Pre processing*

Data preprocessing is the step where the input image is altered to reduce its size and quality without harming the traffic sign it contains.

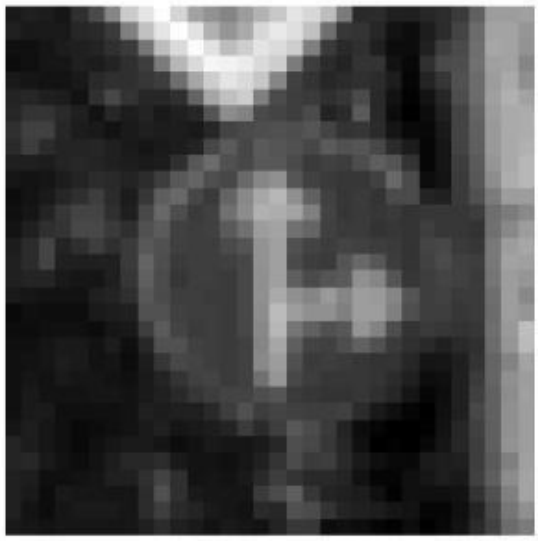


*Figure 6. Grey Scaling*

To recognize and predict traffic signs colors hold least significance, hence the input image is greyscale to reduce the stress on our model.

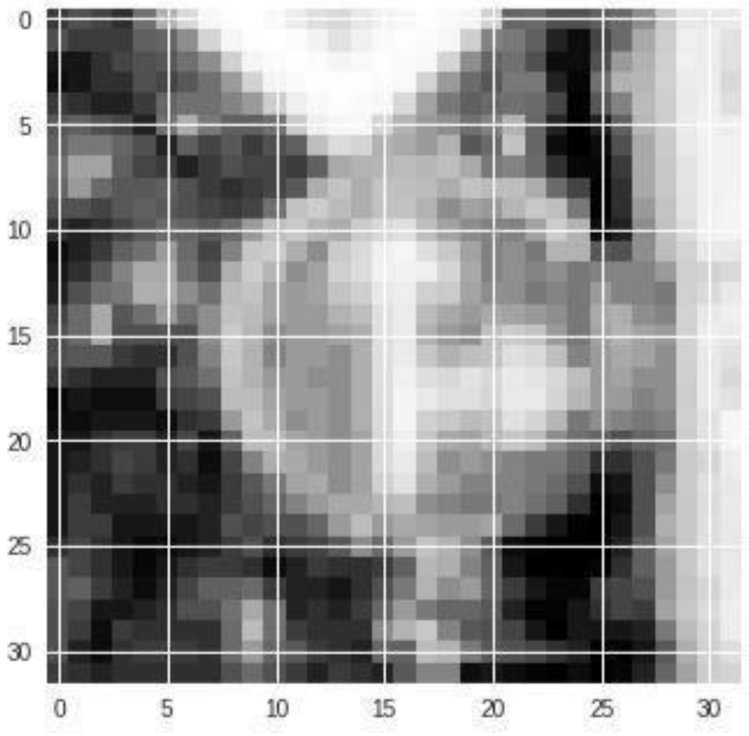
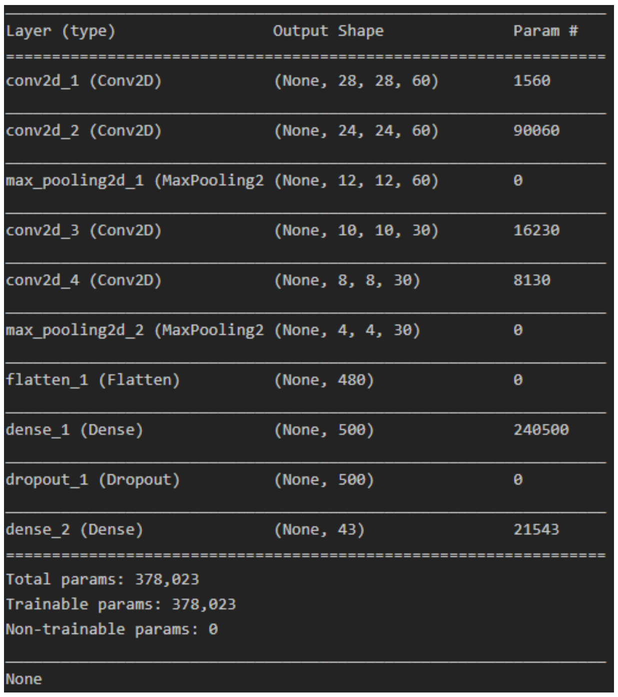


*Figure 7. With Equalization*

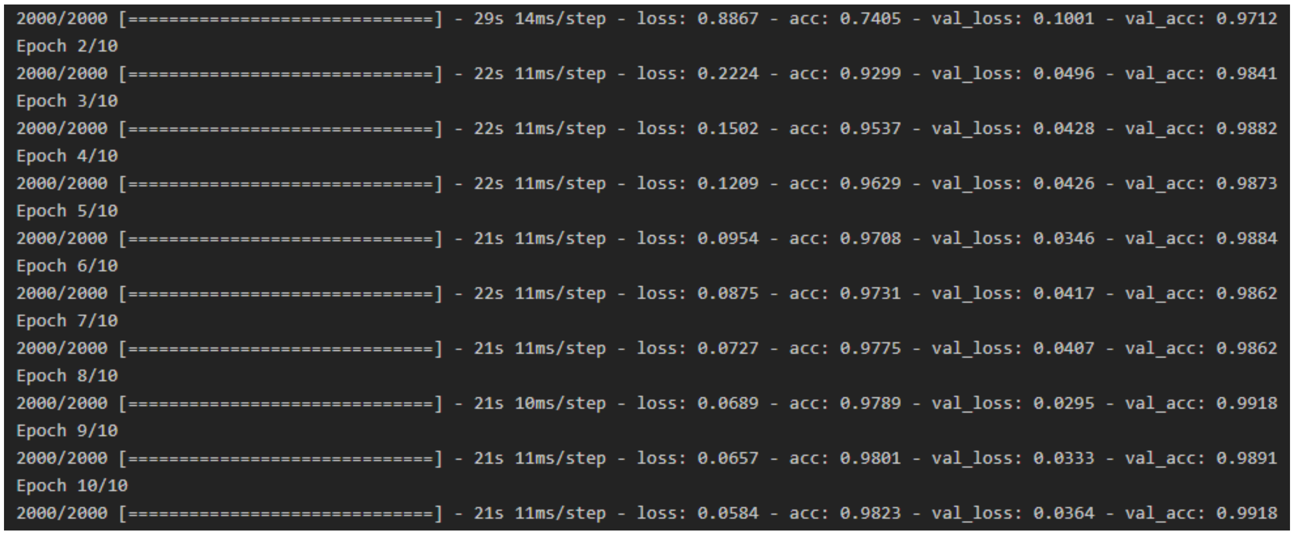
Equalization is a method in image processing of contrast adjustment using the image's histogram. This method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values.



```
_________________________________________________________________
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
```

Table 1

Table 1 is the summary of the model after every image is processed and passed through out model. It consists information like shape and parameters after each step. It can be observed that after each convolution layer the output shape size of image is reducing i.e. our model has to work with smaller sized images which results in drastic increase in processing speed. It can also be observed that all of the parameters from our dataset are trainable.



```
2000/2000 [==============================] - 29s 14ms/step - loss: 0.8867 - acc: 0.7405 - val_loss: 0.1001 - val_acc: 0.9712
Epoch 2/10
2000/2000 [==============================] - 22s 11ms/step - loss: 0.2224 - acc: 0.9299 - val_loss: 0.0496 - val_acc: 0.9841
Epoch 3/10
2000/2000 [==============================] - 22s 11ms/step - loss: 0.1502 - acc: 0.9537 - val_loss: 0.0428 - val_acc: 0.9882
Epoch 4/10
2000/2000 [==============================] - 22s 11ms/step - loss: 0.1209 - acc: 0.9629 - val_loss: 0.0426 - val_acc: 0.9873
Epoch 5/10
2000/2000 [==============================] - 21s 11ms/step - loss: 0.0954 - acc: 0.9708 - val_loss: 0.0346 - val_acc: 0.9884
Epoch 6/10
2000/2000 [==============================] - 22s 11ms/step - loss: 0.0875 - acc: 0.9731 - val_loss: 0.0417 - val_acc: 0.9862
Epoch 7/10
2000/2000 [==============================] - 21s 11ms/step - loss: 0.0727 - acc: 0.9775 - val_loss: 0.0407 - val_acc: 0.9862
Epoch 8/10
2000/2000 [==============================] - 21s 10ms/step - loss: 0.0689 - acc: 0.9789 - val_loss: 0.0295 - val_acc: 0.9918
Epoch 9/10
2000/2000 [==============================] - 21s 11ms/step - loss: 0.0657 - acc: 0.9801 - val_loss: 0.0333 - val_acc: 0.9891
Epoch 10/10
2000/2000 [==============================] - 21s 11ms/step - loss: 0.0584 - acc: 0.9823 - val_loss: 0.0364 - val_acc: 0.9918
```

Figure 8. Loss and accuracy after each epoch

TO train the model we decided to run it for 10 epochs of size 2000 each. Figure 8 shows the time taken, loss, value loss, accuracy and value accuracy after each epoch.

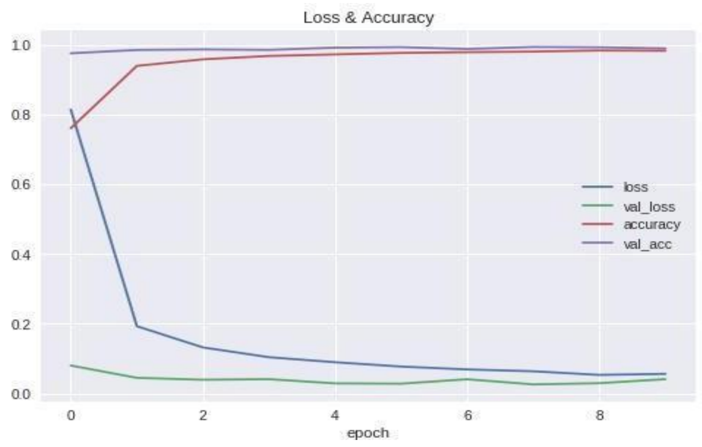


*Figure 9. Accuracy-loss graph*

Figure 9 is a graph generated after completing training of our model. It shows various parameters, mentioned earlier, after each epoch. The vertical axis depicts values of accuracy and loss ranging from 0 to1 and horizontal axis shows the epoch number ranging from 0 to 9. An increase in accuracy and huge decrease in loss can be observed after completing the 2nd epoch (epoch 1). At after completing 8th epoch we can observed that accuracy reaching its maximum value and after completing all 10 epochs our model obtains an accuracy of 0.9823 i.e. 98%.

Testing:

For testing we randomly chose a traffic sign image from the internet and used it as input for our program. The input was also preprocessed before passing it to the model. Our program was able to successfully identify the traffic sign. The output consists of the traffic sign classid (out of 43 classes mentioned earlier) and the sign name.

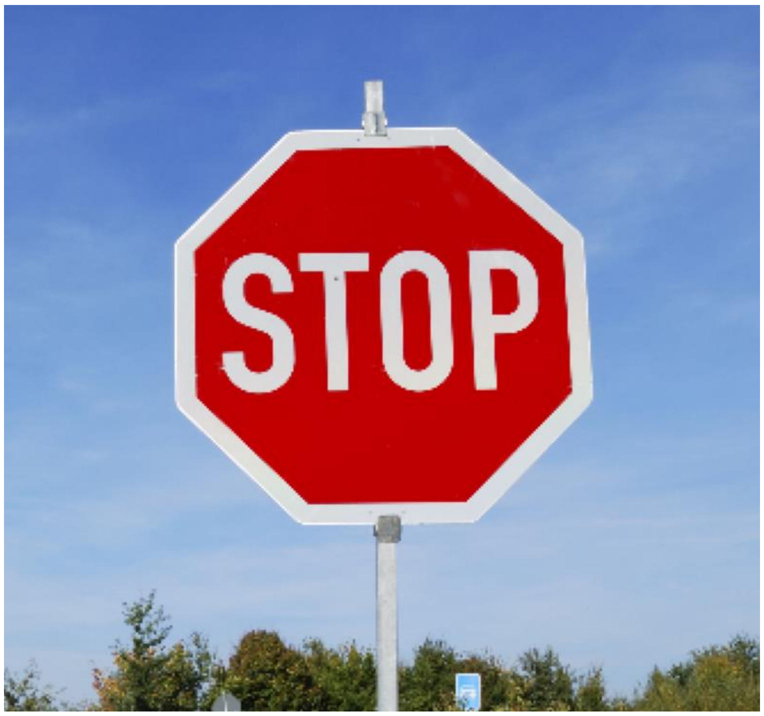


*Figure 10. Input Image*

The input image can be either a URL of the image or a locally saved image. Figure 10 is the input image used for the testing purpose. Here we used a URL of the shown image.

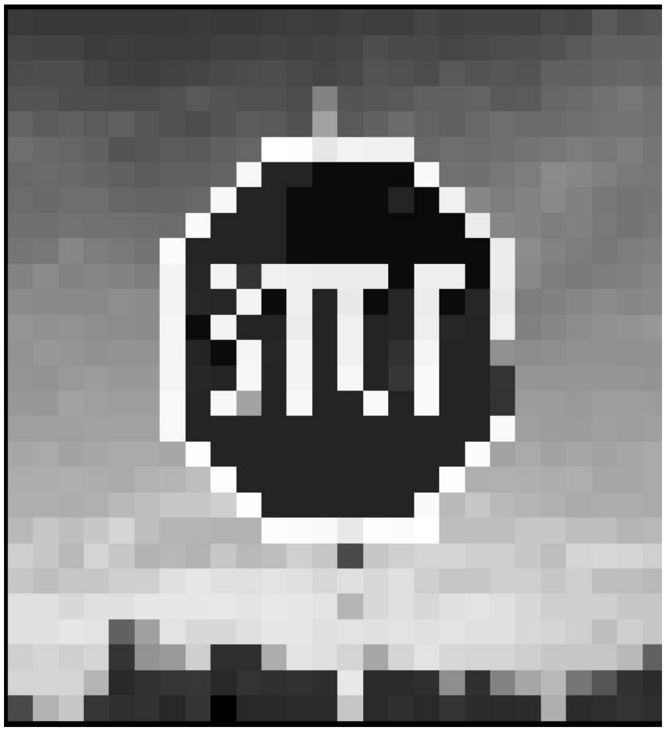
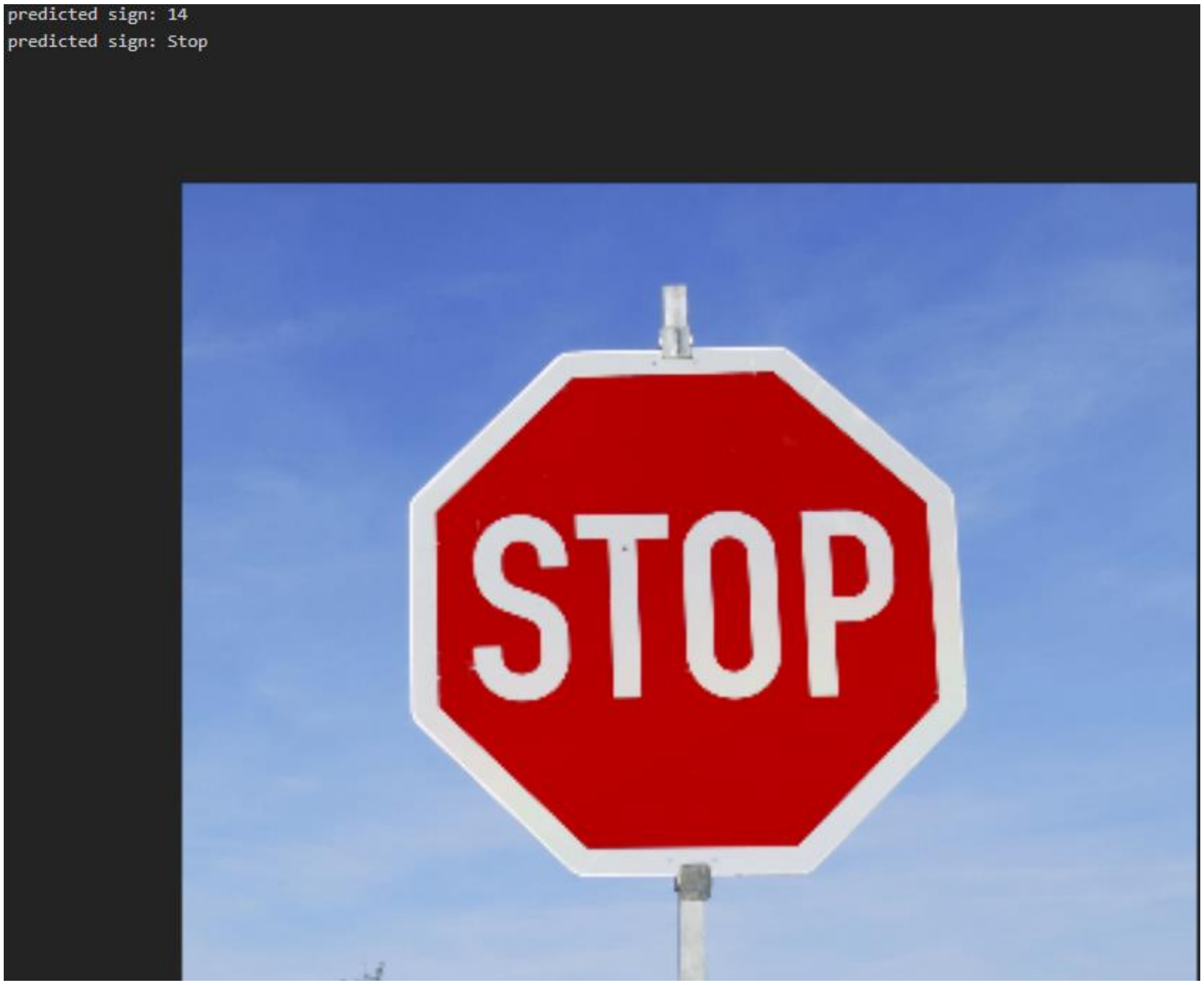


*Figure 11. Input After Preprocessing*

Figure 11 shows the input image after preprocessing is completed. The preprocessing removes colors, reduces quality and size and removes any noise that might hinder our model. This image is then passed to the model for prediction and identification.



```
predicted sign: 14
predicted sign: Stop
```

*Figure 12. Output*

Figure 12 is the output obtained after the model completes identifying the input sign image. Ad it can be observed the model was able to successfully identify the input sign. The output contains the sign class number, the sign name and the input image.

**Sample source code:**

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

np.random.seed(0)



"""## Impoting Data"""



!git clone https://bitbucket.org/jadslim/german-traffic-signs



with open('german-traffic-signs/train.p','rb') as f:
    train_data = pickle.load(f)
with open('german-traffic-signs/valid.p','rb') as f:
    val_data = pickle.load(f)
with open('german-traffic-signs/test.p','rb') as f:
    test_data = pickle.load(f)

X_train, y_train = train_data['features'], train_data['labels']
X_val, y_val = val_data['features'], val_data['labels']
X_test, y_test = test_data['features'], test_data['labels']



print(X_train.shape)
print(X_val.shape)
print(X_test.shape)

assert(X_train.shape[0] == y_train.shape[0]), "The number
of images is not equal to the number of labels"
```

```
assert(X_val.shape[0] == y_val.shape[0]), "The number of images
is not equal to the number of labels"
assert(X_test.shape[0] == y_test.shape[0]), "The number of
images is not equal to the number of labels"
assert(X_train.shape[1:] == (32, 32, 3)), "The dimensions of the
image is not 32*32*3"
assert(X_val.shape[1:] == (32, 32, 3)), "The dimensions of
the image is not 32*32*3"
assert(X_test.shape[1:] == (32, 32, 3)), "The dimensions of the
image is not 32*32*3"



"""## Data Visualisation"""



data = pd.read_csv('german-traffic-signs/signnames.csv')



num_of_samples = []



cols = 5
num_classes = 43



fig, axs = plt.subplots(nrows = num_classes, ncols = cols,
figsize = (5, 50))
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["SignName"])
            num_of_samples.append(len(x_selected))

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


"""## Data Preprocessing"""


plt.imshow(X_train[1000])
plt.axis('off')
print(X_train[1000].shape)
print(y_train[1000])

def grayscale(img):
    image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    plt.axis('off')
    return image


img = grayscale(X_train[1000])
plt.imshow(img, cmap = 'gray')
print(img.shape)
def equalize(img):
    img = cv2.equalizeHist(img)
    return img

img = equalize(img)
plt.imshow(img, cmap = 'gray')
plt.axis('off')
print(img.shape)

def preprocessing(img):
    img = grayscale(img)
    img = equalize(img)
    img = img/255
    return img


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

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



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)



batches = datagen.flow(X_train, y_train, batch_size = 20)
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')

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

"""## Neural Network"""

def neural_model():
    model = Sequential()
    model.add(Conv2D(60, (5, 5), input_shape = (32, 32, 1),
activation = 'relu'))
    model.add(Conv2D(60, (5, 5), input_shape = (32, 32, 1),
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(Dropout(0.5))


    model.add(Flatten())
    model.add(Dense(500, activation = 'relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation = 'softmax'))
    model.compile(Adam(lr = 0.001), loss =
'categorical_crossentropy', metrics = ['accuracy'])
    return model


model = neural_model()
print(model.summary())

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)


plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.legend(['loss', 'val_loss', 'accuracy', 'val_acc'])
plt.title('Loss & Accuracy')
plt.xlabel('epoch')


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


"""## Testing"""
```

```
import requests
from PIL import Image
url = 'https://c8.alamy.com/comp/A0RX23/cars-and-
automobiles-must-turn-left-ahead-sign-A0RX23.jpg'
r = requests.get(url, stream=True)
image = Image.open(r.raw)
plt.axis('off')
plt.imshow(image, cmap=plt.get_cmap('gray'))



img = np.asarray(image)
img = cv2.resize(img, (32, 32))
img = preprocessing(img)
plt.imshow(img, cmap = plt.get_cmap('gray'))
print(img.shape)
img = img.reshape(1, 32, 32, 1)



prediction = str(model.predict_classes(img))



prediction = prediction[1:-1]
##print("predicted sign: "+ prediction )



pred = int(prediction)
plt.imshow(image)
plt.axis('off')

for num, name in data.iteritems():
  name = name.values
  print("predicted sign: "+ str(name[pred]))
```

## Conclusion:

Traffic signs hold significant importance and should not be ignored while travelling. The purpose of this program is to make self-driven vehicles more advanced and safer. By integrating this model in self-driven vehicles, we not only ensure the safety of passengers but also improve the vehicles performance and enhance their decision making. This program is in its very early stages and there is always room for improvement. Currently we require an input image which is already specified i.e. it is not capture in real-time scenario. A visual sensor can be integrated with this program to capture images in real-time. The model itself can be further improved. With newer technology and concepts, it is possible to further enhance the speed and accuracy of this model.

This program at its core is an image classification program. What makes this program special is the use of neural network that we designed and implemented. This model is able to achieve 99% accuracy score. Thus, this model has limitless applications as an image classification model. The model we designed is very easy to understand and thus it can be altered in countless ways to benefit several image classification requirements.

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

## References:

Sanket Doshi (March 2019). Traffic Sign Detection using Convolutional Neural Network. https://towardsdatascience.com/ .India

Sanket Doshi (September 2019). Convolutional Neural Network: Learn And Apply https://medium.com/. India

Stayasheel (Feb 2017). Convolutional Neural Network for Traffic Sign Classification — CarND https://medium.com/. India

Adit Deshpande. A Beginner's Guide To Understanding Convolutional Neural Networks https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural- Networks/

cs231n. Convolutional Neural Networks for Visual Recognition https://cs231n.github.io/convolutional-networks/

J. Zeng, J. Hu and Y. Zhang, "Training Reinforcement Learning Agent for Traffic Signal Control under Different Traffic Conditions," 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 2019, pp. 4248-4254, doi: 10.1109/ITSC.2019.8917342.

Méneroux, Y., Le Guilcher, A., Saint Pierre, G. *et al.* Traffic signal detection from in-vehicle GPS speed profiles using functional data analysis and machine learning. *Int J Data Sci Anal* **10,** 101–

119 (2020). https://doi.org/10.1007/s41060-019-00197-x

D. Patel and Y. Rohilla, "Infrared Sensor based Self–Adaptive Traffic Signal System using Arduino Board," *2020 12th International Conference on Computational Intelligence and Communication Networks (CICN)*, Bhimtal, India, 2020, pp. 175-181, doi: 10.1109/CICN49253.2020.9242560.

Norouzi, M., Abdoos, M. & Bazzan, A.L.C. Experience classification for transfer learning in traffic signal control. *J Supercomput* (2020). https://doi.org/10.1007/s11227-020-03287-x

S. M. Hegazy and M. N. Moustafa, "Classifying aggressive drivers for better traffic signal control," 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), Yokohama, 2017, pp. 702-707, doi: 10.1109/ITSC.2017.8317930.

A. Shekade, R. Mahale, R. Shetage, A. Singh and P. Gadakh, "Vehicle Classification in Traffic Surveillance System using YOLOv3 Model," 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), Coimbatore, India, 2020, pp. 1015-1019, doi: 10.1109/ICESC48915.2020.9155702.

M. Miletić, K. Kušić, M. Gregurić and E. Ivanjko, "State Complexity Reduction in Reinforcement Learning based Adaptive Traffic Signal Control," 2020 International Symposium ELMAR, Zadar, Croatia, 2020, pp. 61-66, doi: 10.1109/ELMAR49956.2020.9219024.

Leonardo Bruno, Giuseppe Parla, Clara Celauro, Improved Traffic Signal Detection and Classification via Image Processing Algorithms, Procedia - Social and Behavioral Sciences,Volume 53,2012,Pages 810-820,ISSN 1877-0428, https://doi.org/10.1016/j.sbspro.2012.09.930.

K. Yoneda, N. Suganuma and M. A. Aldibaja, "Simultaneous state recognition for multiple traffic signals on urban road," 2016 11th France-Japan & 9th Europe-Asia Congress on Mechatronics (MECATRONICS) /17th International Conference on Research and Education in Mechatronics (REM), Compiegne, 2016, pp. 135-140, doi: 10.1109/MECATRONICS.2016.7547129.

S. Liu and S. Lin, "An Optimization for Traffic Signal Control with a Stochastic Link Flow Model," 2020 Chinese Control And Decision Conference (CCDC), Hefei, China, 2020, pp. 60-65, doi: 10.1109/CCDC49329.2020.9164053.

T. S. Tsoi and C. Wheelus, "Traffic Signal Classification with Cost-Sensitive Deep Learning Models," 2020 IEEE International Conference on Knowledge Graph (ICKG), Nanjing, China, 2020, pp. 586-592, doi: 10.1109/ICBK50248.2020.00088.

J. Zeng, J. Hu and Y. Zhang, "Training Reinforcement Learning Agent for Traffic Signal Control under Different Traffic Conditions," 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 2019, pp. 4248-4254, doi: 10.1109/ITSC.2019.8917342.