KULLIYYAH OF INFORMATION & COMMUNICATION TECHNOLOGY

CSCI 4343: Data Science

SEMESTER 1, 2021/2022

Project Title: Traffic Signs Detection for Self-driving Vehicle

PREPARED BY:

NAME	MATRIC NO.
Md Borhan Uddin	1822665
Saif al faied	1828615
Mahin Islam	1723486
Imran	1828271
Mohamed Moubarak Mohamed Misbahou Mkouboi	1820705

LECTURER
Dr. Raini Binti Hassan

DUE
21 January 2022

Table of Contents

Project Title: Traffic Signs Detection for self-driving vehicle

Abstract

In this paper, the effectiveness of detecting and classifying various traffic indicators are being investigated. Traffic laws, restricted entry, traffic signals, turning left or right, children crossing, no passing of big trucks, and so on are all examples of traffic signs. The process of determining whether a category of a road sign belongs to is known as road signs identification. The whole system was designed more efficiently by adding the detection system that will read and classify traffic signs into different categories and notify vehicles for further movement.

Keywords- Traffic laws, Artificial Neural Network (ANN), k-Nearest Neighbour (KNN) , Support Vector Machine (SVM).

INTRODUCTION

Throughout the centuries', automated vehicles have been a hot topic of research. The use of a front-looking camera for vehicle localization and navigation, environmental mapping, and collision avoidance was the focus of much research. Every traffic sign detection algorithm strives to achieve four basic objectives. The algorithm must be accurate in terms of the initial goal. The algorithm's most basic requirement and assessment metric is accuracy. Prediction precision under standard test conditions may be sufficient for the deployment of driver assistance. However, in other situations, like completely driverless vehicles, reliability in the worst-case situation must be considered and adequately maintained. This ability to attain high accuracy in a range of natural situations instantly leads to the recognition algorithm's second evaluation metric: robustness. Because it is difficult to forecast what kinds of situations the vehicle will face, the algorithm must be able to accomplish the desired outcome in all conditions, including unfavourable ones, in order to achieve high accuracy. In addition to efficiency within standard test conditions, the algorithm's integrity and consistency must also be assessed. The detection algorithm must be fast as the third metric for evaluation. The detective's ultimate objective. The algorithm will be put into action instantaneously. Again, computing complexity is a limitation in the implementation of autonomous vehicles. Second, in an automated network system, traffic sign detection requires little effort. Finally, autonomous vehicles travel at a fast pace, necessitating a quick response to traffic signs. The expense seems to be the ultimate performance measure for the detection algorithm. The vehicle already requires a variety of sensors, including GPS, inertial sensors, radar, and even Lidar, in order to perform autonomous driving. Despite the fact that the camera is not the costliest sensor on board, being able to accomplish pretty high detection accuracy with a limited camera is still in the best interests of the industry. However, we will create a deep neural network model that can classify traffic signals in an image into many categories. We may read and understand traffic signs using our model, which is a critical duty for all autonomous vehicles.

Background

The improvement and development of generation has enabled the use of an increasing number of vehicles in recent time. Digital literacy has given the opportunity to make smarter, faster, low energy, and smaller computing structures that may be included in vehicle controllers. Vehicles can now provide several active safety features like real-time traffic sign detection using powerful cameras, obstacle detection using radar systems, and more often, highly automated vehicles by including controllers with multiple cores (Sirbu et al, 2018). Self-driving cars are becoming increasingly popular. Self-driving may reduce staff cost for big companies as they don't need extra numbers of driver employees. With the idea of self-driving automation driver vehicles, the giant firms like Tesla's efforts to automate their electric vehicles. The self-driving vehicles would have to correctly read traffic signs and observe traffic rules to achieve automation driver features. After identifying these traffic indicators, it should be able to make appropriate decisions. Therefore, vehicles should be able to comprehend traffic signs and make appropriate decisions to achieve accuracy in this technology. Moreover, autonomous mobility companies may use self-driving vehicle systems to produce smart vehicles with safety features which will increase the sales and decrease accident rate (Pymnts, 2019). Identifying the features of automated vehicles can be the fastest technique to enable automated vehicle implementation. By 2026, the worldwide autonomous vehicle market is expected to be worth $556.67 billion (Akshay Jadhav, 2018). Self-driving vehicles will have to use established highways, roads, and city streets to get around. Hence, our project will be about analysing traffic signs detection for self-driving vehicles.

Problem statement

Currently, by the growth of population, the public and personal vehicles are also increasing respectively. For this, people often break the traffic rules which leads to a heavy traffic jam and rises accident rate. On average 1578 people died in a year for traffic rules violation, especially 51% of them who died by violating traffic signals (Marks & Harrison (n.d). Many drivers misunderstand the traffic signal, particularly new drivers who are new on the road and often make mistakes and violate them. Moreover, big transportation provider companies need a large number of drivers as their employees. Public transportation companies such as government have needed many drivers to handle their transportation department. The challenge of traffic Signal Detection is to automatically detect traffic signs. It is only because of advances in computer vision and deep learning that it is now feasible to discern road tracks from camera frames and traffic signals during the self-driving process. However, there are other issues involved, such as vehicle form diversity, vehicle safety, over illumination, sharp-turned roads, collision warning, and variable road settings. The detection and identification of objects in real time is a serious problem. The 2016 Tesla auto-pilot accident is a noteworthy example of a safety failure, in which the vehicle's sensors were mixed by the sun and the system failed to notice the truck approaching from the traffic signal, resulting in the crash.

Research Question & hypothesis

How does a traffic signs detection system detect traffic signs for self-driving vehicles and its accuracy by using artificial intelligence tools?

Research Objectives:

Our objective is to apply our learning and research on this topic that

1. To study on existing traffic signal system and its limitations

2. To understand and visualise the data from processed data

3. To determine the model and its greatest working feature

4. To train and validate the model.

Research significances

1. A prediction algorithm that forecasts road traffic signals can help automobiles be more equipped to identify and read them on a self-driving car.
2. Identifying the high-risk characteristics that contribute to traffic sign recognition can help the automobile drive safely.
3. Identifying potential elements that have an impact on better recognition of traffic signal patterns for self-driving automobiles.

Literature review

Despite the latest innovations, the case for autonomous car navigation in everyday life has grown in prominence. Government-funded autonomous driving research programs include the Eureka Prometheus Project 1 and the V-Charge Project 2. The first initiative to use neural network models for autonomous vehicle navigation is called ALVIN (Autonomous Land Vehicle in a Neural Network). It is made up of shallow and fully-connected layers as opposed to network models with hundreds of layers. It excelled on basic tracks with few impediments and was the first to calculate steering angles directly from picture pixels. To extract characteristics from automotive sensor frames, NVIDIA deployed contemporary convolutional networks [1]. Simple real-world scenarios are achieved, such as lane keeping and driving on flat, unobstructed routes. Zhang and Cho have used research on query-efficient training [2] for autonomous cars [3]. Karslet al. [4] focused on using front-facing camera data synced with steering angles to train deep network algorithms. They created three distinct end-to-end deep learning models and tested their effectiveness on a racecar-sized self-driving vehicle. More advanced algorithms for aspects of the system of dynamic nature are also available in the literature [5], in addition to just using deep neural networks. Fully connected networks (FCN) and long-term memory recursive networks (LSTM) topologies are used in the model. Koutnik et al. [6] used the reinforcement learning method to build repeated neural networks (with over 1 million weights). The data was compiled using the TORCS racing vehicle simulator, and tests were conducted on the same platform. Chi and Mu [7] proposed a deep learning model that blends temporal and spatial information well. GANs, on the other hand, are networks that compete against one another to learn representations and then produce correct instances of those representations [8]. GAN was used by Kaufler et al. [9] to anticipate and mimic human driving behaviour. On the Caltech-101 and Caltech Pedestrian datasets, Uçar et al. [10] investigated a hybrid model of CNN and SVM for object recognition and pedestrian detection. Several methods for traffic sign identification have been developed over the last two decades [11]. It consists of three basic steps: (1) region segmentation to identify potential sign-containing regions. Because signs come in a variety of hues, we normally use colour features in this phase. (2) shape analysis to classify signs as circular, triangular, or rectangular. The third step, (3) recognition, involves identifying indicators discovered during the preceding feature extraction, i.e., determining their class and significance. For this phase, we have a variety of classification approaches to choose from, including Artificial Neural Network (ANN) [12], k-Nearest Neighbour (KNN) [13], Support Vector Machine (SVM) [14], and Random Forest (RF) [15]. Jo proved that KNN classifiers are effective in this situation. He does, however, come at a great cost in terms of processing time. Linear SVMs, on the other hand, have a quicker processing time while still producing good results.

As a result, this system will be implemented using Artificial Neural Network (ANN) of deep learning algorithm to detect traffic signs.

Methodology:

Data description:

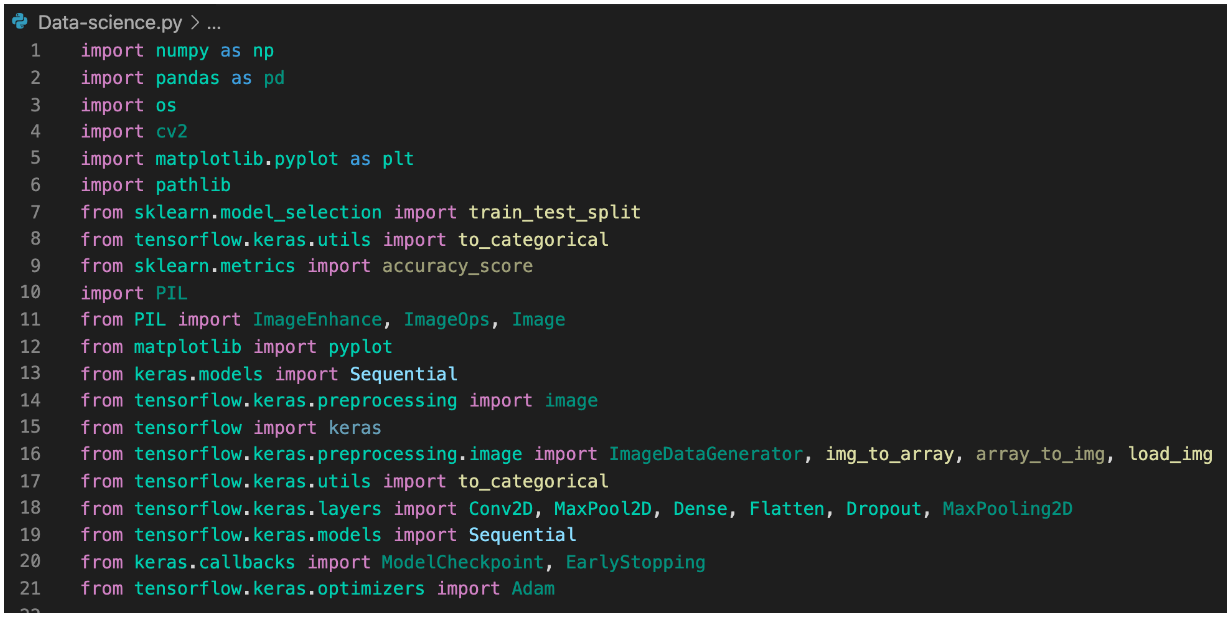
To build our system, we will use traffic signs dataset which will be found at Kaggle that is an open-source data providers. The dataset has included some csv file. Mainly, the dataset contains images of different traffic signs. The dataset classifies in different classes. More than 50,000 images of various traffic signs are included in the dataset. It's further divided into 43 separate categories. The dataset is highly diverse, some classes have a lot of photos, while others have very few. The dataset is approximately 300 MB in size. The dataset has two folders, a train folder that contains photos for each class and a test folder that will be used to evaluate our model. As we are predicting from traffic images, the dataset contains mostly of photos. So, the dataset does not have missing values and require data cleaning.

Algorithm:

To recognize traffic sign by self-driving, as we mentioned earlier, we are using Convolutional Neural Network (CNN) to our model. However, a Convolutional Neural Network (CNN) is a Deep Learning algorithm that can take an input image, assign relevance (learnable weights and biases) to various aspects/objects in the image, and distinguish between them. When compared to other classification methods, the amount of pre-processing required by a CNN is significantly less. While basic approaches require hand-engineering of filters, CNN can learn these filters with enough training. To Build the model using CNN the flows of algorithms are exploring the data including data cleaning, handling missing data, etc. Building the CNN model. Train and validate the model. Test the model with dataset. Finally, the model will predict a traffic sign for self-driving vehicle using CNN model.

Data preparation:

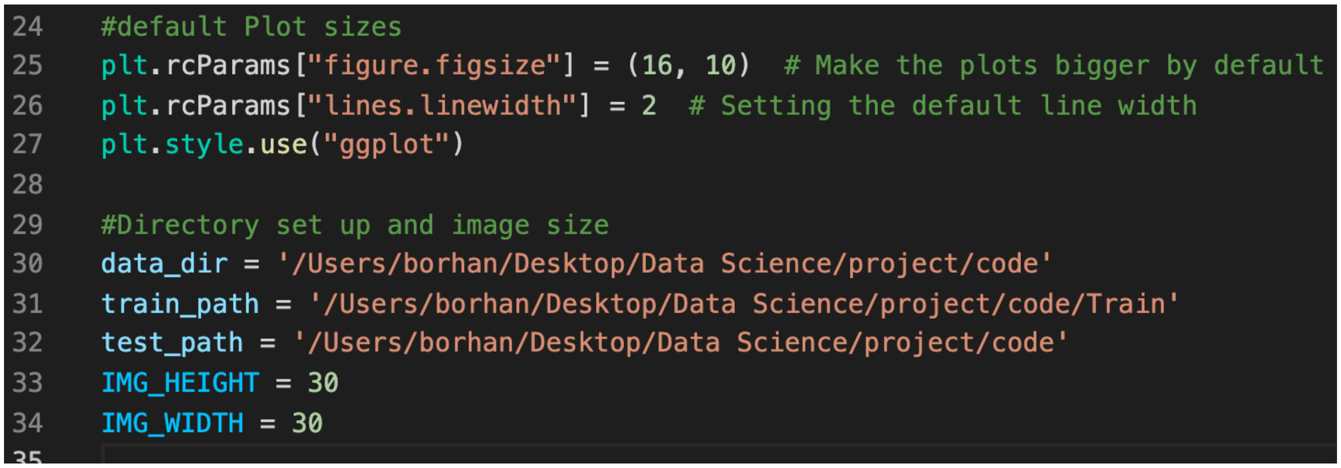
For this process, we need some python packages to install to visualize the data and create the CNN model. However, as we have mention earlier in the data description, here we are working with image datasets. We do not need to check missing value or data cleaning as it is found in a well manner. Based on image, the model will predict specific traffic sign. The required packages are below:



Data-science.py > ...

```python
import numpy as np
import pandas as pd
import os
import cv2
import matplotlib.pyplot as plt
import pathlib
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import accuracy_score
import PIL
from PIL import ImageEnhance, ImageOps, Image
from matplotlib import pyplot
from keras.models import Sequential
from tensorflow.keras.preprocessing import image
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, array_to_img, load_img
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.models import Sequential
from keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.optimizers import Adam
```

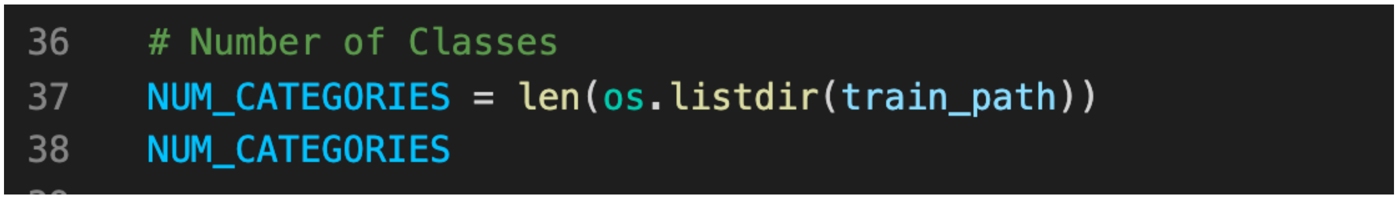
Before going to visualize the data, we have determined the plot sizes. As we have two data files: train data and test data, we have set the dataset directory. Also, to visualize the images, we have set the images sizes as 30 by 30.



```python
#default Plot sizes
plt.rcParams["figure.figsize"] = (16, 10)  # Make the plots bigger by default
plt.rcParams["lines.linewidth"] = 2  # Setting the default line width
plt.style.use("ggplot")

#Directory set up and image size
data_dir = '/Users/borhan/Desktop/Data Science/project/code'
train_path = '/Users/borhan/Desktop/Data Science/project/code/Train'
test_path = '/Users/borhan/Desktop/Data Science/project/code'
IMG_HEIGHT = 30
IMG_WIDTH = 30
```

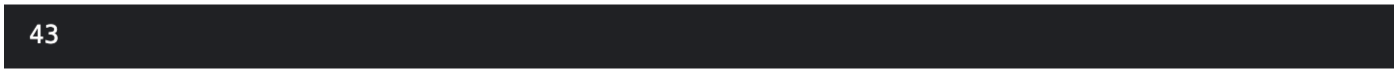
Visualization:

To begin this process, we will confirm the classes (43) that we have mention in the datasets.



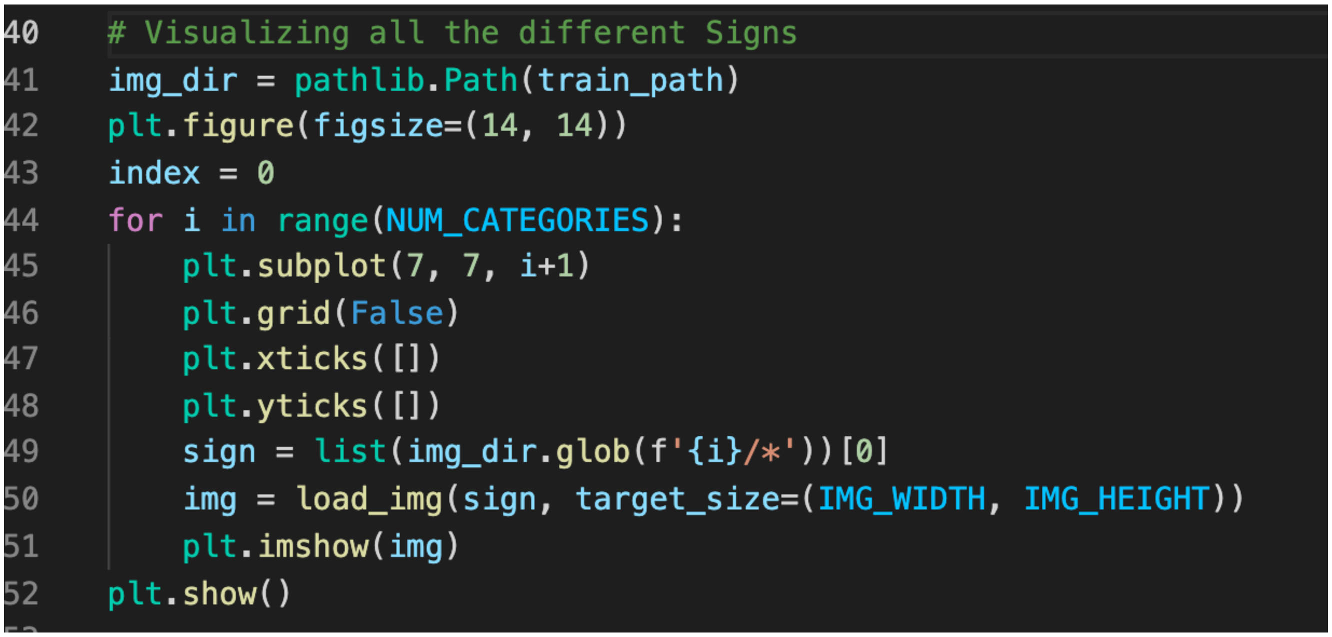
```python
# Number of Classes
NUM_CATEGORIES = len(os.listdir(train_path))
NUM_CATEGORIES
```

Output:



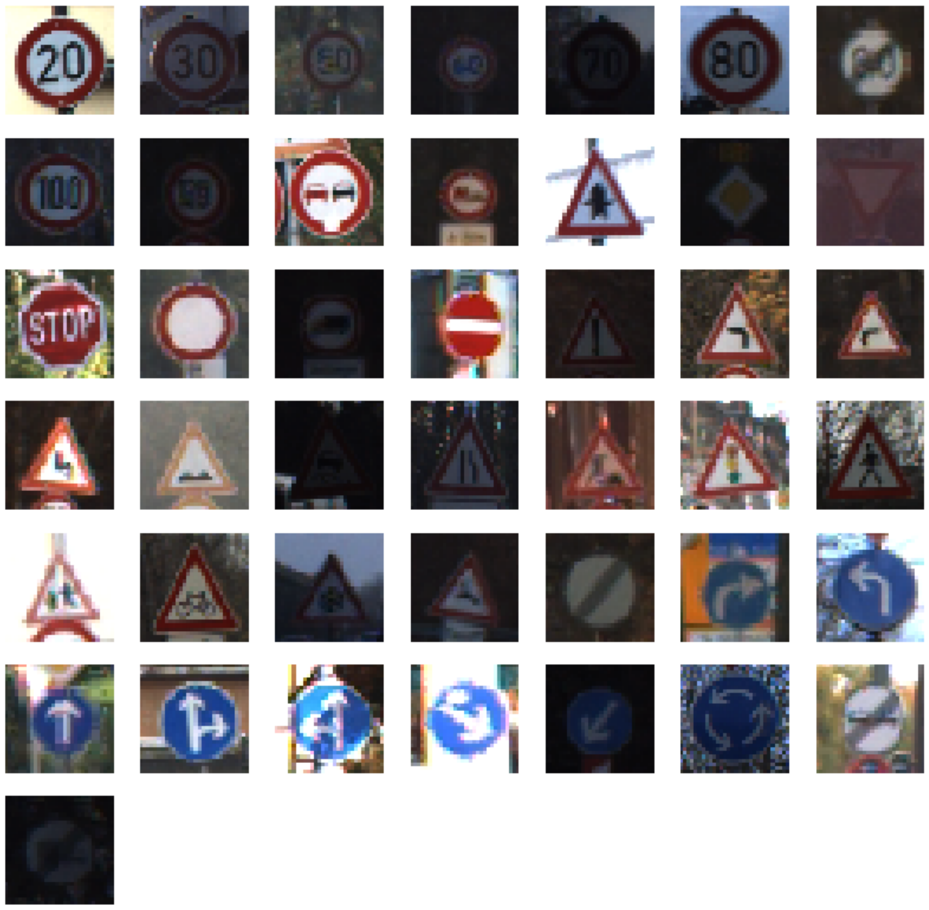
```
43
```

Now, we will visualize all the traffic signs that includes in our datasets to get better view of the datasets.

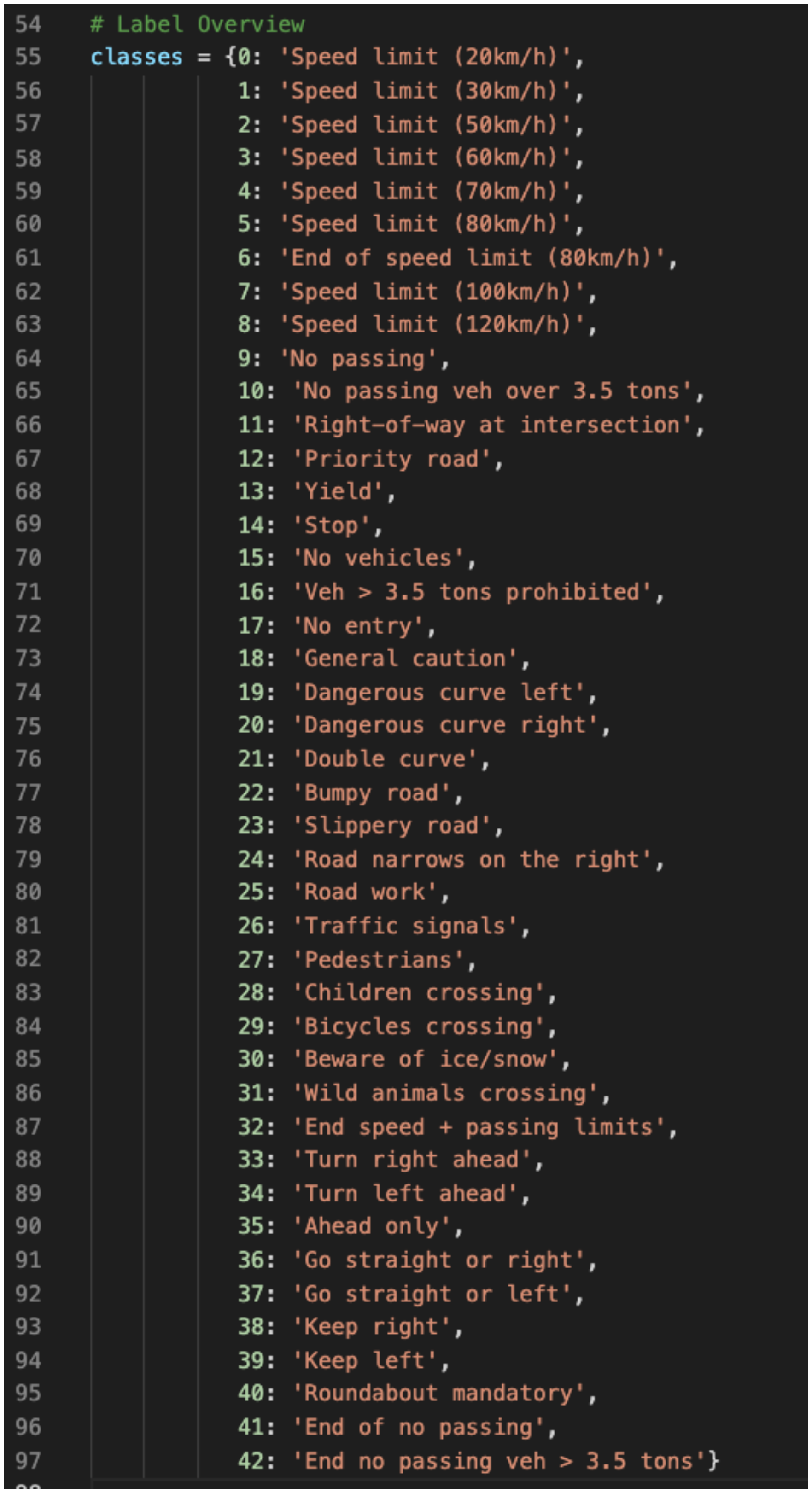


```python
# Visualizing all the different Signs
img_dir = pathlib.Path(train_path)
plt.figure(figsize=(14, 14))
index = 0
for i in range(NUM_CATEGORIES):
    plt.subplot(7, 7, i+1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    sign = list(img_dir.glob(f'{i}/*'))[0]
    img = load_img(sign, target_size=(IMG_WIDTH, IMG_HEIGHT))
    plt.imshow(img)
plt.show()
```

Output: showing all the traffic signs as a class.

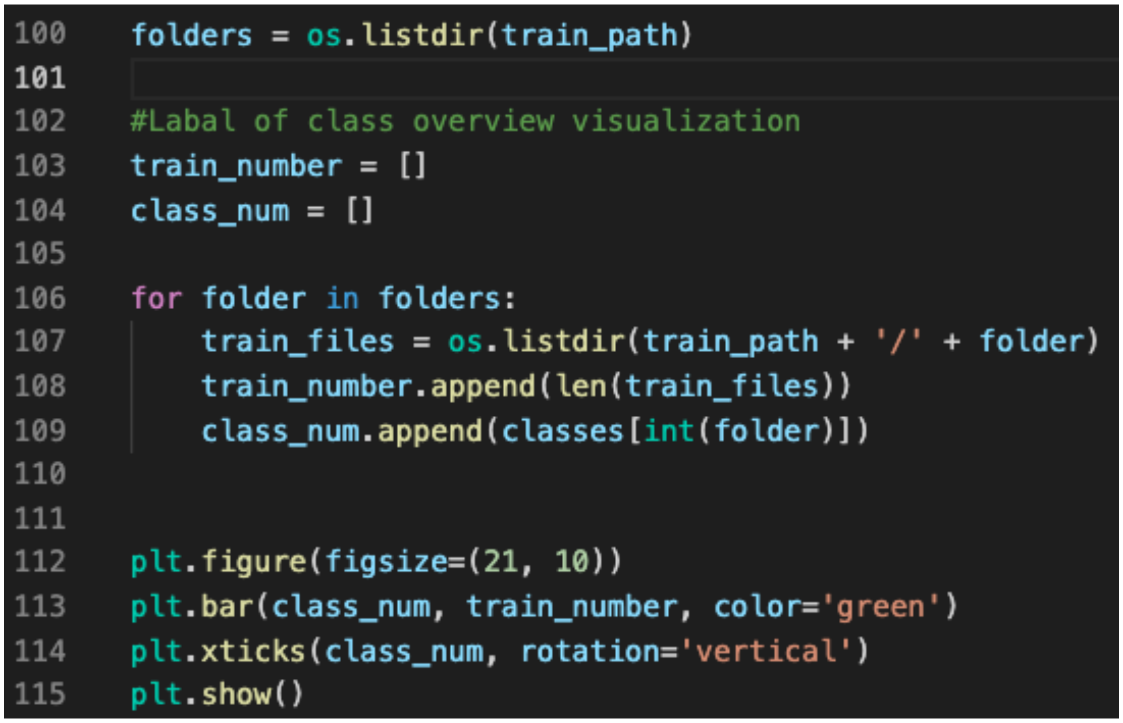


In this segment, the label of overview is defined, and all 43 classes are initializing with the respective name in a dictionary.



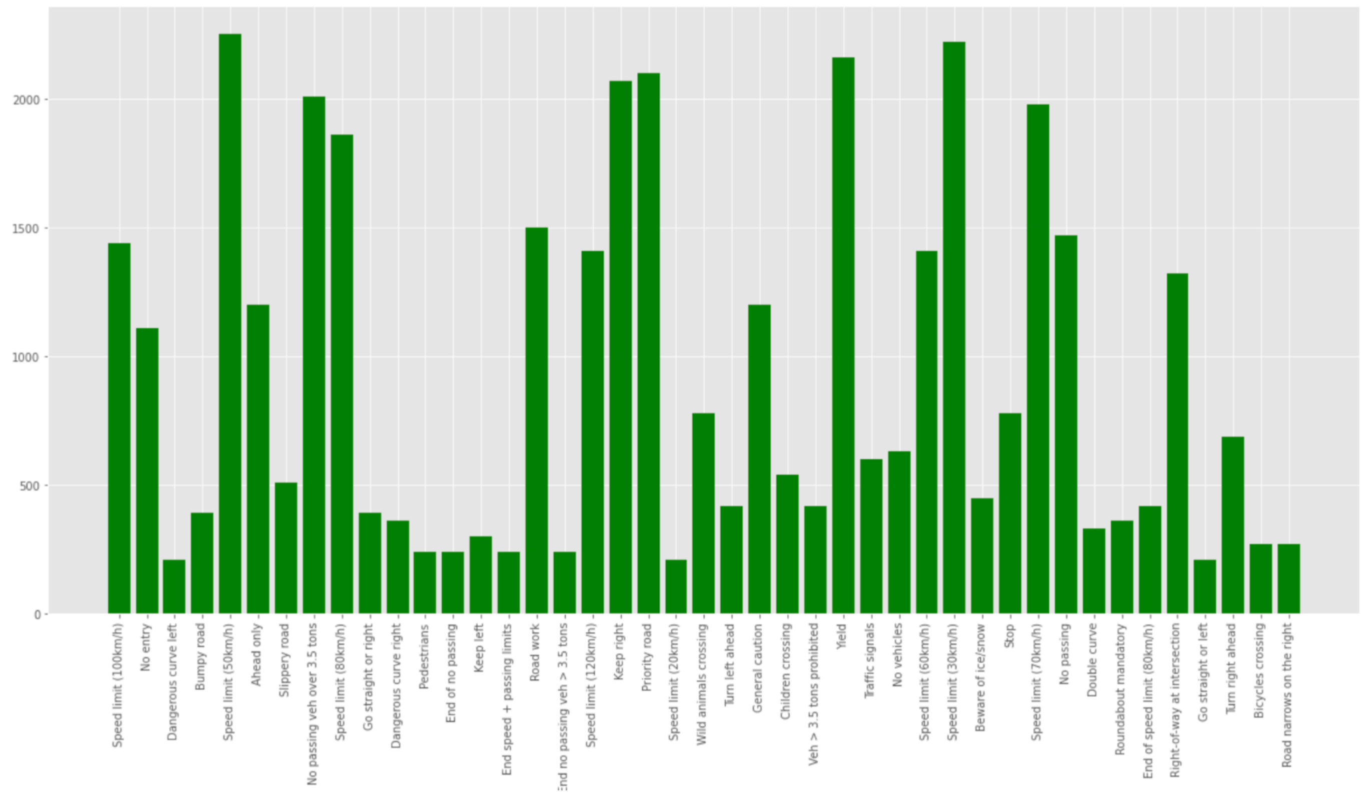
```
# Label Overview
classes = {0: 'Speed limit (20km/h)',
            1: 'Speed limit (30km/h)',
            2: 'Speed limit (50km/h)',
            3: 'Speed limit (60km/h)',
            4: 'Speed limit (70km/h)',
            5: 'Speed limit (80km/h)',
            6: 'End of speed limit (80km/h)',
            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'}
```

With the training data, label of all the classes visualize by class name (x axis) and number of training data (y axis)



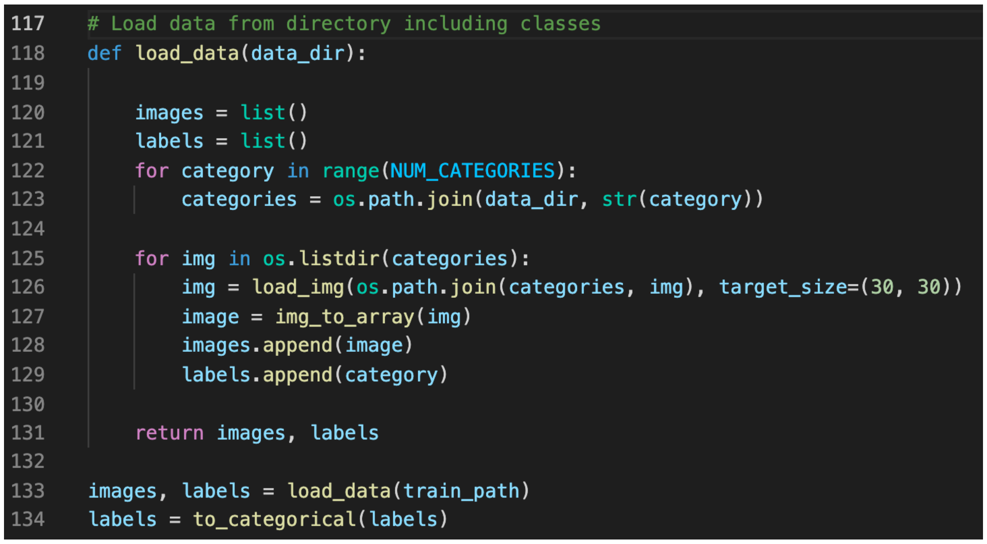
```
folders = os.listdir(train_path)

#Labal of class overview visualization
train_number = []
class_num = []

for folder in folders:
    train_files = os.listdir(train_path + '/' + folder)
    train_number.append(len(train_files))
    class_num.append(classes[int(folder)])


plt.figure(figsize=(21, 10))
plt.bar(class_num, train_number, color='green')
plt.xticks(class_num, rotation='vertical')
plt.show()
```

Output:



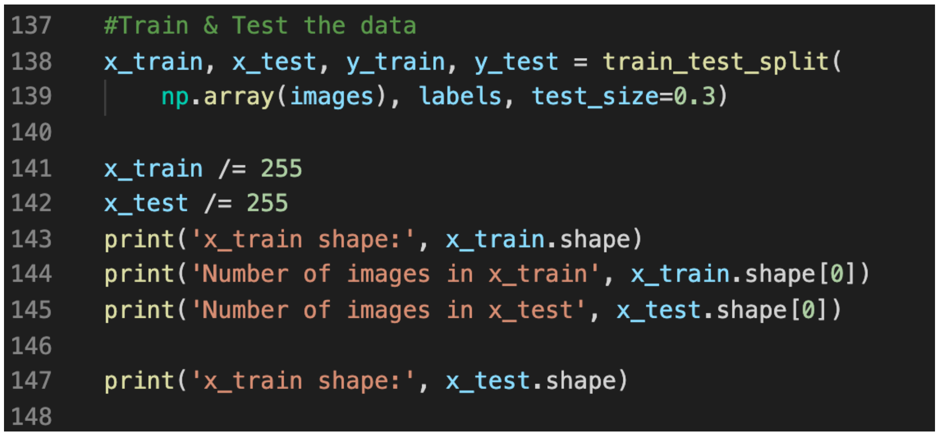
Modelling:

To build the model, we must load the data from directory, here is a function called load data that will hold all categories of data from directory including classes:



```python
# Load data from directory including classes
def load_data(data_dir):

    images = list()
    labels = list()
    for category in range(NUM_CATEGORIES):
        categories = os.path.join(data_dir, str(category))

    for img in os.listdir(categories):
        img = load_img(os.path.join(categories, img), target_size=(30, 30))
        image = img_to_array(img)
        images.append(image)
        labels.append(category)

    return images, labels

images, labels = load_data(train_path)
labels = to_categorical(labels)
```

To feed the model, we need to turn the list into numpy arrays. The data has the shape (27446, 30, 30, 3), indicating that there are 39,209 images of (30,30) pixels and that the last three indicate that the data comprises colored images (RGB value). The train test split () method in the sklearn package is used to split training and testing data. We utilize the two categorical method from the keras.utils package to transform the labels in y train and t test into one-hot encoding.



```python
#Train & Test the data
x_train, x_test, y_train, y_test = train_test_split(
    np.array(images), labels, test_size=0.3)

x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print('Number of images in x_train', x_train.shape[0])
print('Number of images in x_test', x_test.shape[0])

print('x_train shape:', x_test.shape)
```

Output:

```
x_train shape: (27446, 30, 30, 3)
Number of images in x_train 27446
Number of images in x_test 11763
```

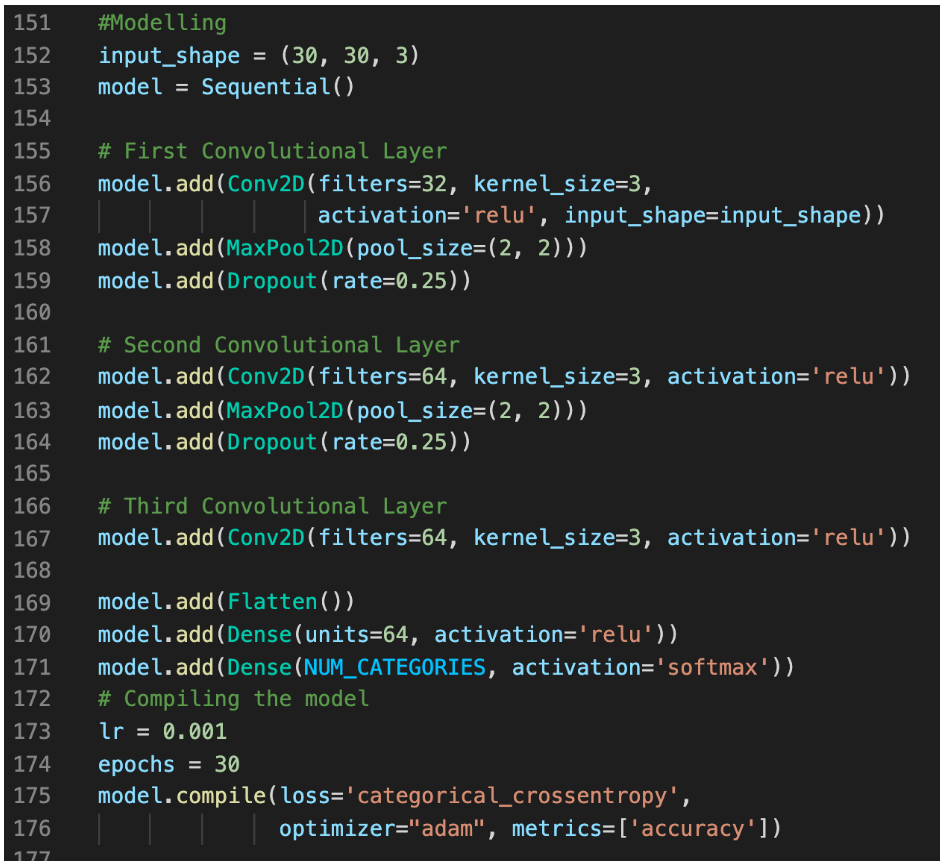
```
x_train shape: (11763, 30, 30, 3)
```

Creating model:

To classify the images into their respective categories, we will build a Convolutional Neural Network (CNN) model Now, the model will be created using a Convolutional Neural Network (CNN), there will be three layer and modelling shape 30 by 30, height and weight respectively. the architecture of our model is

The architecture of our model is:

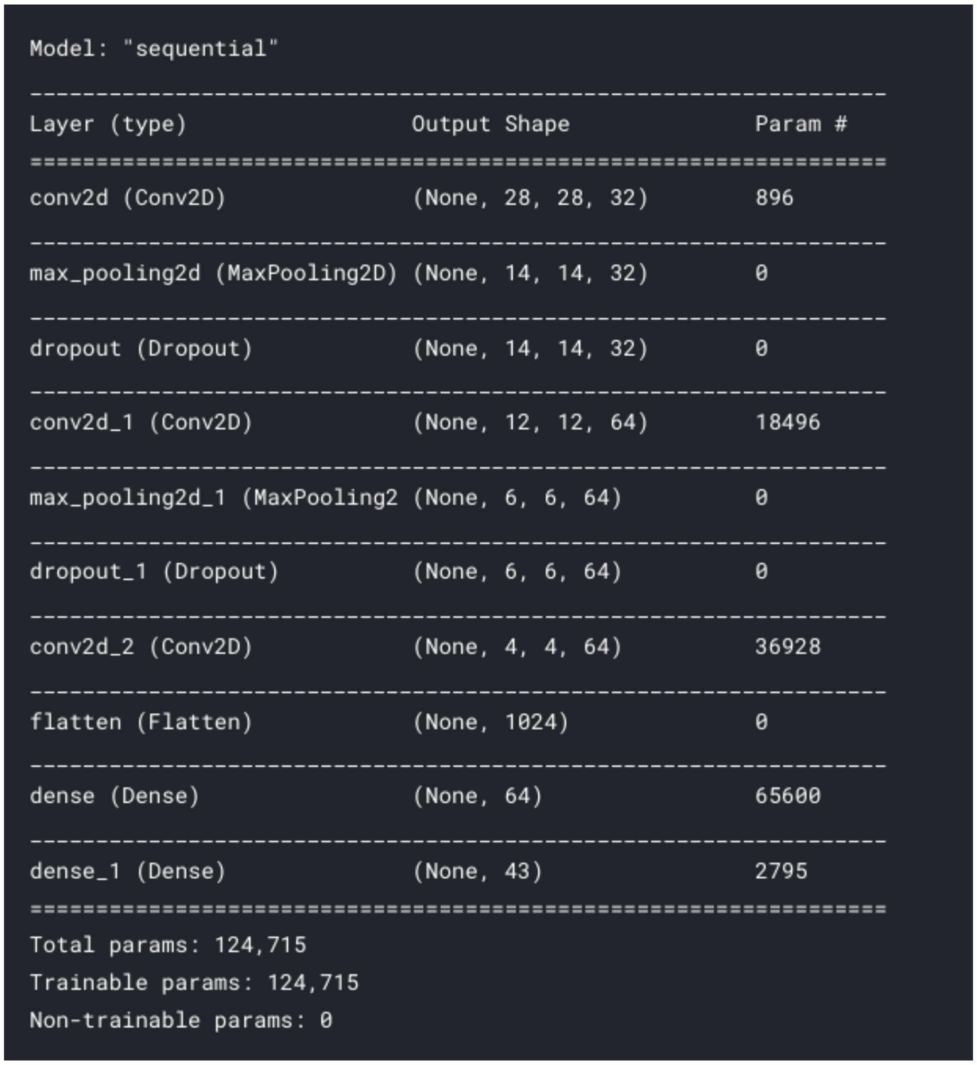
- 2 Conv2D layer (filter=32, kernel_size=(3), activation=”relu”)
- MaxPool2D layer (pool_size=(2,2))
- Dropout layer (rate=0.25)
- 2 Conv2D layer (filter=64, kernel_size=(3), activation=”relu”)
- MaxPool2D layer (pool_size=(2,2))
- Dropout layer (rate=0.25)
- Flatten layer to squeeze the layers into 1 dimension
- Dropout layer (rate=0.5)
- Dense layer (43 nodes, activation=”softmax”)



```python
#Modelling
input_shape = (30, 30, 3)
model = Sequential()

# First Convolutional Layer
model.add(Conv2D(filters=32, kernel_size=3,
                 activation='relu', input_shape=input_shape))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))

# Second Convolutional Layer
model.add(Conv2D(filters=64, kernel_size=3, activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))

# Third Convolutional Layer
model.add(Conv2D(filters=64, kernel_size=3, activation='relu'))

model.add(Flatten())
model.add(Dense(units=64, activation='relu'))
model.add(Dense(NUM_CATEGORIES, activation='softmax'))
# Compiling the model
lr = 0.001
epochs = 30
model.compile(loss='categorical_crossentropy',
              optimizer="adam", metrics=['accuracy'])
```

A CNN Model has been created with three convolution layers. The traffic signs images are test by layer to layer as the images are nothing but a matrix of pixel values so it will flatten images. However, now we are going to see details of the model.

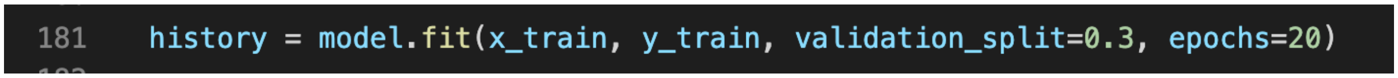
>> model.summary()



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 12, 12, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dense_1 (Dense)              (None, 43)                2795
=================================================================
Total params: 124,715
Trainable params: 124,715
Non-trainable params: 0
```

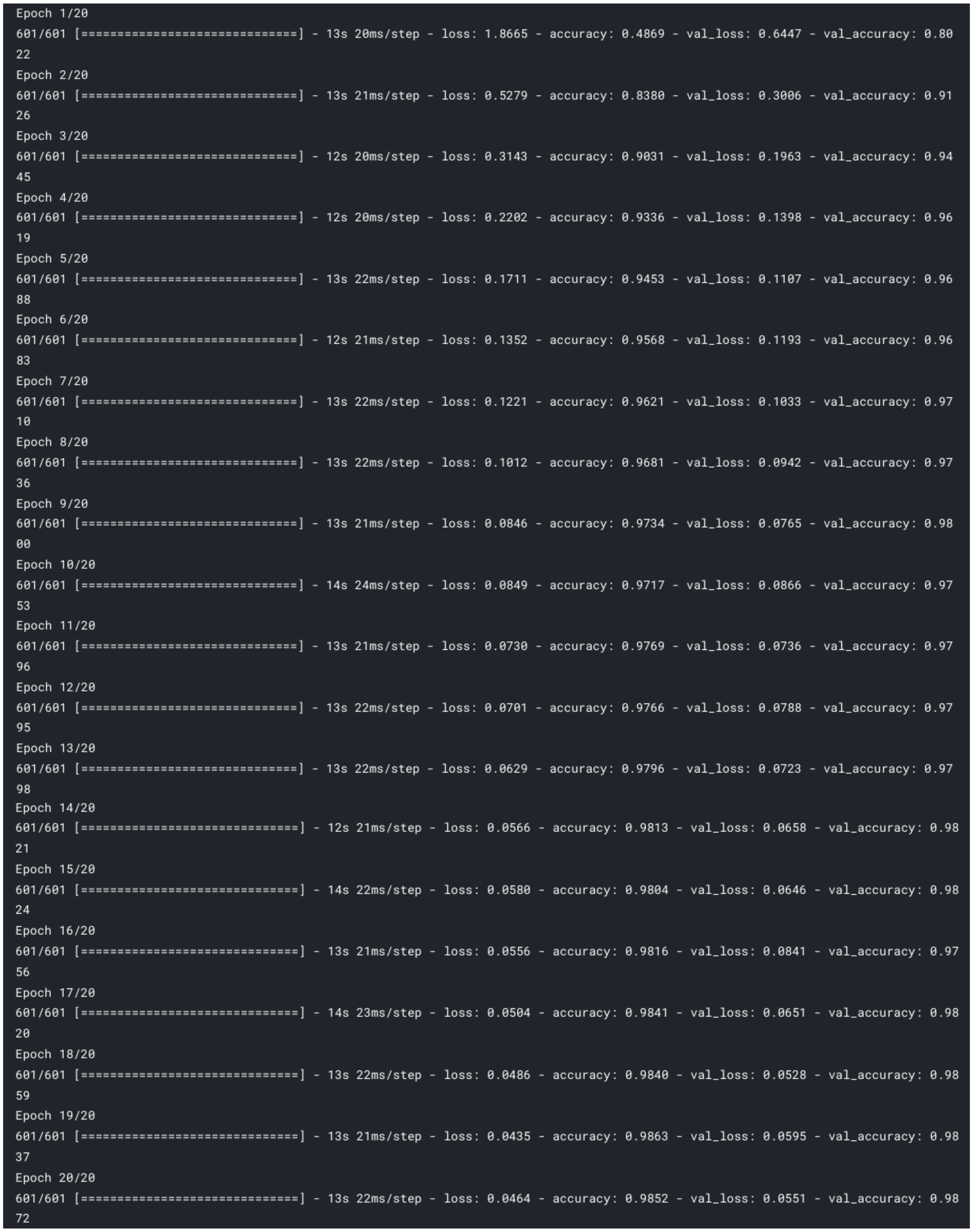
Train and model validation:

After constructing the model architecture, we use model.fit to train the model (). We teste by splitting the model by 0.3, our model fared better. The accuracy was stable after 20 epochs.



```
history = model.fit(x_train, y_train, validation_split=0.3, epochs=20)
```

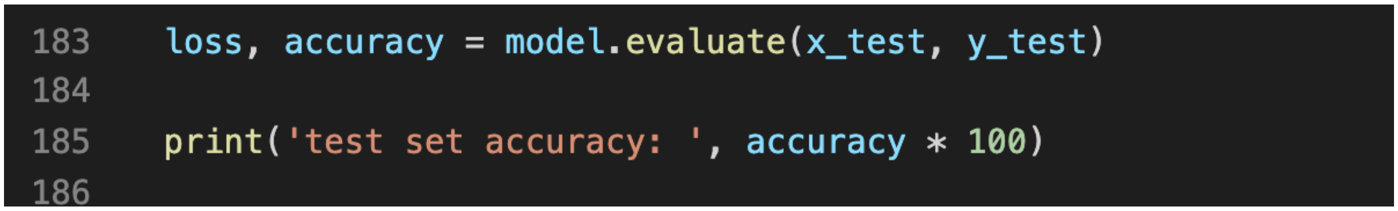
Output:



```
Epoch 1/20
601/601 [==============================] - 13s 20ms/step - loss: 1.8665 - accuracy: 0.4869 - val_loss: 0.6447 - val_accuracy: 0.80
22
Epoch 2/20
601/601 [==============================] - 13s 21ms/step - loss: 0.5279 - accuracy: 0.8380 - val_loss: 0.3006 - val_accuracy: 0.91
26
Epoch 3/20
601/601 [==============================] - 12s 20ms/step - loss: 0.3143 - accuracy: 0.9031 - val_loss: 0.1963 - val_accuracy: 0.94
45
Epoch 4/20
601/601 [==============================] - 12s 20ms/step - loss: 0.2202 - accuracy: 0.9336 - val_loss: 0.1398 - val_accuracy: 0.96
19
Epoch 5/20
601/601 [==============================] - 13s 22ms/step - loss: 0.1711 - accuracy: 0.9453 - val_loss: 0.1107 - val_accuracy: 0.96
88
Epoch 6/20
601/601 [==============================] - 12s 21ms/step - loss: 0.1352 - accuracy: 0.9568 - val_loss: 0.1193 - val_accuracy: 0.96
83
Epoch 7/20
601/601 [==============================] - 13s 22ms/step - loss: 0.1221 - accuracy: 0.9621 - val_loss: 0.1033 - val_accuracy: 0.97
10
Epoch 8/20
601/601 [==============================] - 13s 22ms/step - loss: 0.1012 - accuracy: 0.9681 - val_loss: 0.0942 - val_accuracy: 0.97
36
Epoch 9/20
601/601 [==============================] - 13s 21ms/step - loss: 0.0846 - accuracy: 0.9734 - val_loss: 0.0765 - val_accuracy: 0.98
00
Epoch 10/20
601/601 [==============================] - 14s 24ms/step - loss: 0.0849 - accuracy: 0.9717 - val_loss: 0.0866 - val_accuracy: 0.97
53
Epoch 11/20
601/601 [==============================] - 13s 21ms/step - loss: 0.0730 - accuracy: 0.9769 - val_loss: 0.0736 - val_accuracy: 0.97
96
Epoch 12/20
601/601 [==============================] - 13s 22ms/step - loss: 0.0701 - accuracy: 0.9766 - val_loss: 0.0788 - val_accuracy: 0.97
95
Epoch 13/20
601/601 [==============================] - 13s 22ms/step - loss: 0.0629 - accuracy: 0.9796 - val_loss: 0.0723 - val_accuracy: 0.97
98
Epoch 14/20
601/601 [==============================] - 12s 21ms/step - loss: 0.0566 - accuracy: 0.9813 - val_loss: 0.0658 - val_accuracy: 0.98
21
Epoch 15/20
601/601 [==============================] - 14s 22ms/step - loss: 0.0580 - accuracy: 0.9804 - val_loss: 0.0646 - val_accuracy: 0.98
24
Epoch 16/20
601/601 [==============================] - 13s 21ms/step - loss: 0.0556 - accuracy: 0.9816 - val_loss: 0.0841 - val_accuracy: 0.97
56
Epoch 17/20
601/601 [==============================] - 14s 23ms/step - loss: 0.0504 - accuracy: 0.9841 - val_loss: 0.0651 - val_accuracy: 0.98
20
Epoch 18/20
601/601 [==============================] - 13s 22ms/step - loss: 0.0486 - accuracy: 0.9840 - val_loss: 0.0528 - val_accuracy: 0.98
59
Epoch 19/20
601/601 [==============================] - 13s 21ms/step - loss: 0.0435 - accuracy: 0.9863 - val_loss: 0.0595 - val_accuracy: 0.98
37
Epoch 20/20
601/601 [==============================] - 13s 22ms/step - loss: 0.0464 - accuracy: 0.9852 - val_loss: 0.0551 - val_accuracy: 0.98
72
```

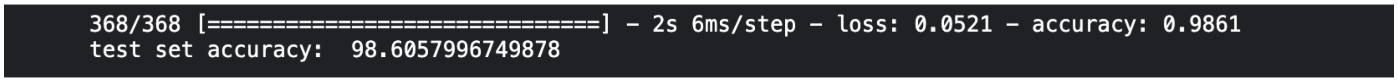
Result analysis:

Our model got a 98% accuracy on the training dataset. With matplotlib, we plot the graph for accuracy by per epoch .



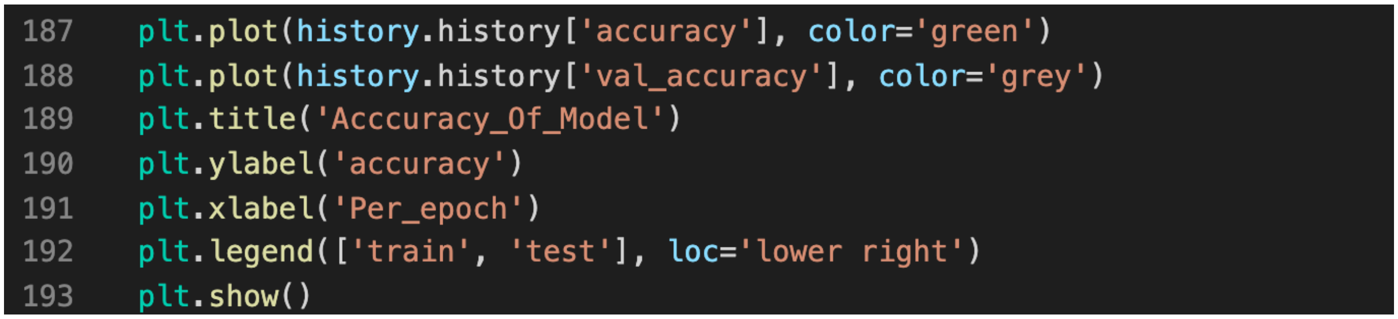
```
loss, accuracy = model.evaluate(x_test, y_test)

print('test set accuracy: ', accuracy * 100)

```

Output:



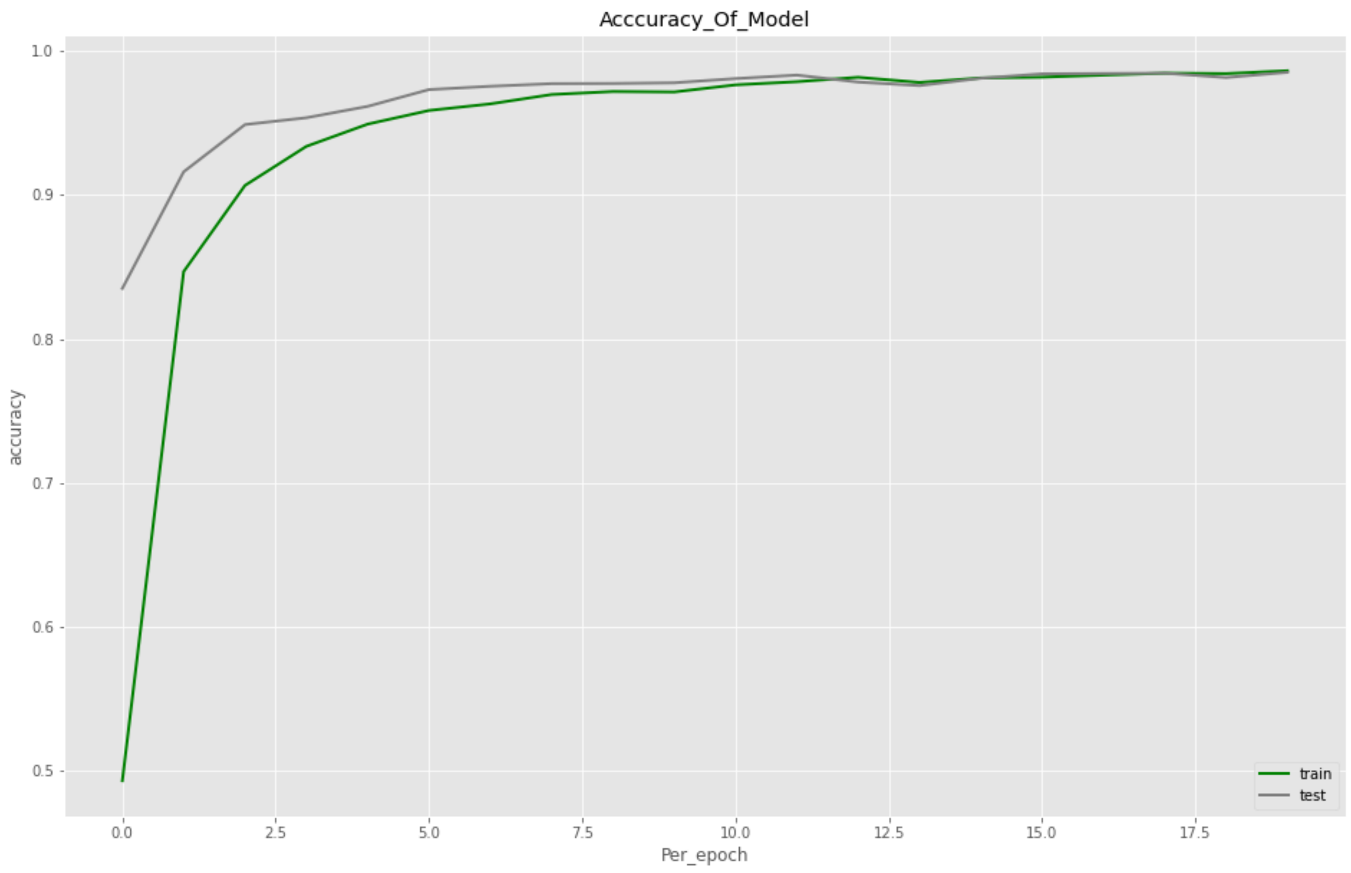
```
368/368 [==============================] - 2s 6ms/step - loss: 0.0521 - accuracy: 0.9861
test set accuracy:  98.6057996749878
```

Accuracy visualization by per epoch:

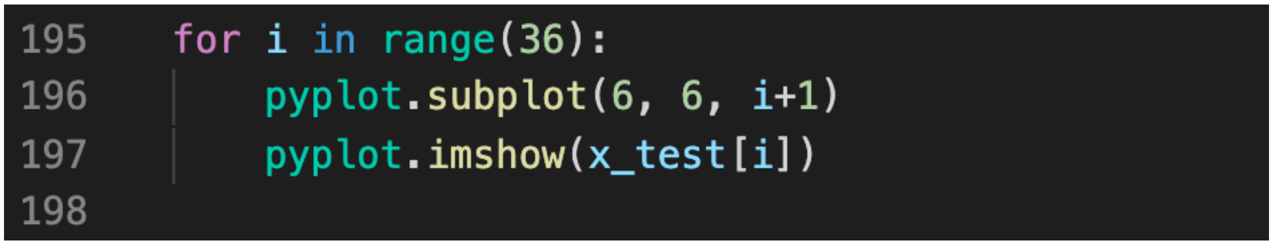


```
plt.plot(history.history['accuracy'], color='green')
plt.plot(history.history['val_accuracy'], color='grey')
plt.title('Acccuracy_Of_Model')
plt.ylabel('accuracy')
plt.xlabel('Per_epoch')
plt.legend(['train', 'test'], loc='lower right')
plt.show()
```

Output: green line denotes the train data accuracy per epoch and grey is test.

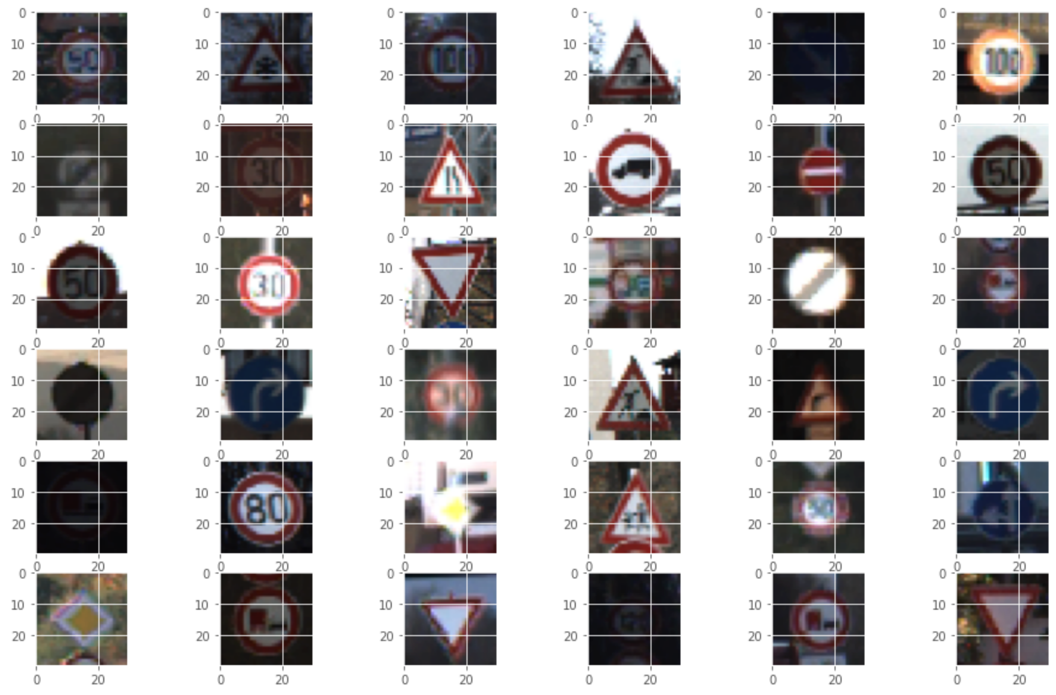


Now, we are going to plot by 36 images by making subplot that will look into test images to further prrdict



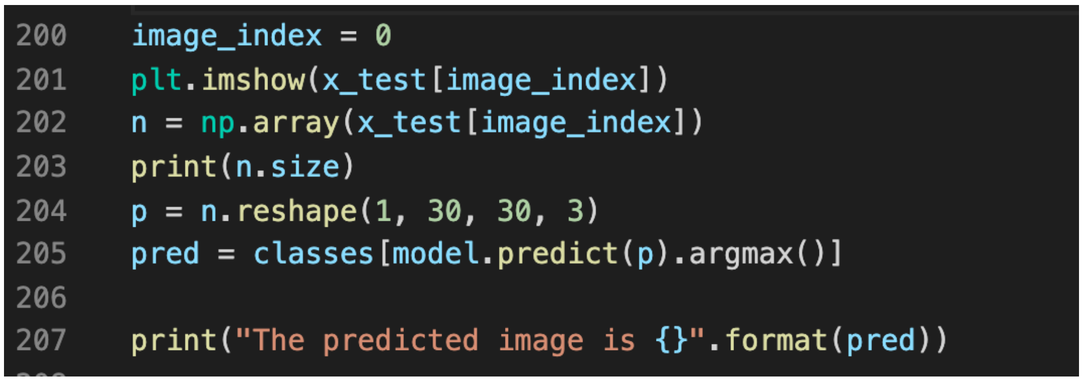
```
for i in range(36):
    pyplot.subplot(6, 6, i+1)
    pyplot.imshow(x_test[i])
```

Output:



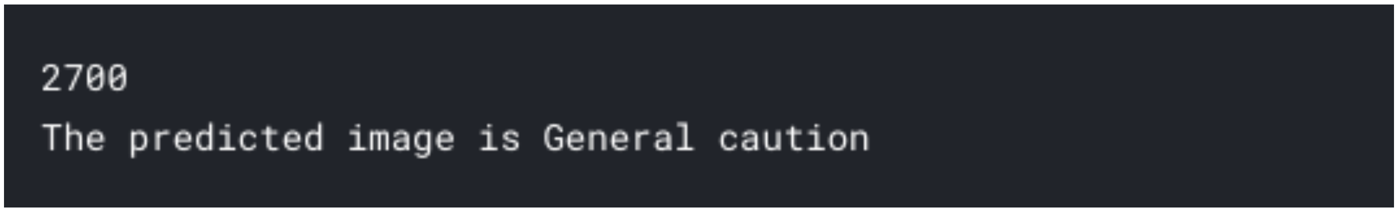
Finally,

Here is the final prediction of our result by evaluating three coevolution layers looking into index sizes of the images.



```
image_index = 0
plt.imshow(x_test[image_index])
n = np.array(x_test[image_index])
print(n.size)
p = n.reshape(1, 30, 30, 3)
pred = classes[model.predict(p).argmax()]

print("The predicted image is {}".format(pred))
```

Output: As output is showing test image index is 2700. The predicted image is general caution



```
2700
The predicted image is General caution
```

Discussions:

At we know Traffic signs classification is the process of identifying which class a traffic sign belongs to. There are various kinds of traffic signals like speed limit, no entry, turn left or light but as a trial the above experiment we have had a prediction on traffic sign detection. However, to build the model, we have use Convolutional Neural Network (CNN) algorithm and data sets which contained images of traffic signals including two files test data and train data which have been visualize the visualization segment. The model had three convolution layers which analyze the images. Based on result analysis, we have concluded a result that gives us 98% accuracy of our model. In the phase of evaluation, the model finally predicts the image index number that was match with a traffic sign which is indicated a general caution.

Future works

For future work, we would like to use datasets which is more realistic and easy to detect because existing dataset's images are not very clear to look as sometimes it is very hard to detect. The experimental setup of this project, we build the model to detects traffic signs using CNN but as progression, to minimize the accident, we will expand the model to detects not only traffic sign but also store traffic rules database in the system to obey all the traffic rules by getting signals from the database through the system. Moreover, it is also planned that features will be added to make the model more realistic that have a Graphical user interface (GUI) where users may upload the images and will have a button which will classify the classes of traffic sign images using classify function. The classify function transforms an image into a shape dimension that will allow us to evaluate the model easily.

References

1. Bojarski, M., et al.: End-to-end learning for self-driving cars (2016). arXiv preprint: arXiv:1604.07316

2. Ross, S., Gordon, G.J., Bagnell, D.: A reduction of imitation learning and structured prediction to no-regret online learning. In: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS), pp. 627–635 (2011)

3. Zhang, J., Cho, K.: Query efficient imitation learning for end-to-end simulated driving. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, California, USA, pp. 2891–2897 (2017)

4. Karslı, M., et al.: End-to-End Learning Model Design for Steering Autonomous Vehicle, 26. Sinyal İşleme ve İletişim UygulamalarıKurultayı(2018)

5. Xu, H., Gao, Y., Yu, F., Darrell, T.: End-to-end learning of driving models from large-scale video datasets. arXiv preprint: arXiv:1612.01079

6. Koutník, J., et al.: Evolving large-scale neural networks for vision-based TORCS, pp. 206–212 (2013)

7. Chi, L., Mu, Y.: Deep steering: learning end-to-end driving model from spatial and temporal visual cues. In: Proceedings of the Workshop on Visual Analysis in Smart and Connected Communities, Mountain View, California, USA, pp. 9–16 (2017)

8. Mnih, V.B., et al.: Asynchronous methods for deep reinforcement learning. In: Proceedings of the International Conference on Machine Learning, pp. 1928–1937 (2016)

9. Kuefler, A., Morton, J., Wheeler, T., Kochenderfer, M.: Imitating driver behavior with generative adversarial networks. In: IEEE Intelligent Vehicles Symposium (IV), pp. 204–211 (2017)

10. Uçar, A., Demir, Y., Güzeliş, C.: Object recognition and detection with deep learning for autonomous driving applications. Simulation 93(9), 759–769 (2017)

11. Vavilin, A., Jo, K.-H.: Graph-based approach for robust road guidance sign recognition from differently exposed images. J. Univ. Comput. Sci. 15(4), 786–804 (2009)

12. Islam, Kh.T., Raj, R.G.: Real-time (vision-based) road sign recognition using an artificial neural network. Sensors 17(4), 853 (2017)

13. Han, Y., Virupakshappa K., Oruklu, E.: Robust traffic sign recognition with feature extraction and k-NN classification methods. In: IEEE International Conference on Electro/Information Technology (EIT), pp. 484–848 (2015)

14. Gudigar, A., Jagadale, B.N., Mahesh, P.K., Raghavendra, U.: Kernel-based automatic traffic sign detection and recognition using SVM. In: Mathew, J., Patra, P., Pradhan, D.K.,Kuttyamma, A.J. (eds.) Eco-Friendly Computing and Communication Systems. Communications in Computer and Information Science, vol. 305. Springer, Heidelberg (2012)

15. Ellahyani, A., ElAnsari, M., ElJaafari, I.: Traffic sign detection and recognition based on random forests. Appl. Soft Comput. 46, 805–815 (2016)

16. Jo, K.H.: A comparative study of classification methods for traffic signs recognition. In: 2014IEEE International Conference on Industrial Technology (ICIT), pp. 614–619. IEEE (2014)

17. Sirbu, Maria-Adelina & Baiasu, Andrei & Bogdan, Razvan & Crișan-Vida, Mihaela. (2018). Smart traffic sign detection on autonomous car. 1-4. 10.1109/ISETC.2018.8583948.

18. Akshay Jadhav (2018), Electric and Hybrid Vehicles.
https://www.alliedmarketresearch.com/autonomous-vehicle-market

19. Pymnts, (2019). Who Will Use Self-Driving Cars?.
https://www.pymnts.com/innovation/2019/who-will-use-self-driving-cars/

20. Marks & Harrison (n.d). Causes of Car Accidents: Disobeying Traffic Signals in Virginia. Marks and Harrison. https://www.marksandharrison.com/accident-attorney/car-accidents/causes-of-car-accidents/disobeying-traffic-signals/