Deep Learning & Neural Network

Traffic Sign Recognizer using CNN

Introduction:

You must have heard about the self-driving cars in which the passenger can fully depend on the car for traveling. to accomplish level 5 self-sufficient, it is important for vehicles to comprehend and adhere to all traffic rules. In the realm of Artificial Intelligence and headway in advances, numerous analysts and large organizations like Tesla, Uber, Google, Mercedes-Benz, Toyota, Ford, Audi, and so forth are chipping away at self-sufficient vehicles and self-driving vehicles. Along these lines, for accomplishing exactness in this innovation, the vehicles should be able to interpret traffic signs and make decisions accordingly.

There are a few unique kinds of traffic signs like speed restricts, no access, traffic lights, turn left or right, kids crossing, no passing of substantial vehicles, and so on Traffic signs arrangement is the way toward recognizing which class a traffic sign has a place with.

In this Python project model, we will build a deep neural network model that can order traffic signs present in the picture into various classes. With this model, we are able to read and understand traffic signs which are a very important task for every single self-ruling vehicle.

Our approach to building this traffic sign classification model is discussed in four steps:

- *Explore the dataset*
- *Build a CNN model*
- *Train and validate the model*
- *Test the model with test dataset*

In this work the objective is threefold:

1. *to gain knowledge and understanding deep learning, in particular, convolutional Neural Networks (CNNs);*
2. *Use deep learning to automatically classifying traffic signs. In particular, in this work we use the standard German Traffic Sign Benchmark Dataset (GTSRB) for training and testing.*
3. *Compare classification results obtained with CNNs on raw GTSRB images and those obtained with CNNs on enhanced images.*

Literature Review:

Solution 1: *This paper presents a Deep Learning approach for traffic sign recognition systems. Several classification experiments are conducted over publicly available traffic sign datasets from Germany and Belgium using a Deep Neural Network which comprises Convolutional layers and Spatial Transformer Networks. Such trials are built to measure the impact of diverse factors with the end goal of designing a Convolutional Neural Network that can improve the state-of-the-art* *of traffic sign classification task. First, different adaptive and non-adaptive stochastic gradient descent optimization algorithms such as SGD, SGD-Nesterov, RMSprop and Adam are evaluated. Subsequently, multiple combinations of Spatial Transformer Networks placed at distinct positions within the main neural network are analyzed.*

Link:

https://github.com/aarcosg/tsr-torch

Result: *The recognition rate of the proposed Convolutional Neural Network reports an accuracy of 99.71% in the German Traffic Sign Recognition Benchmark, outperforming previous state-of-the-art methods and also being more efficient in terms of memory requirements.*

Solution 2:

Automatic detection and recognition of traffic signs plays a crucial role in management of the traffic-sign inventory. It provides accurate and timely way to manage traffic-sign inventory with a minimal human effort. In the computer vision community, the recognition and detection of traffic signs is a well-researched problem. A vast majority of existing approaches perform well on traffic signs needed for advanced drivers-assistance and autonomous systems. However, this represents a relatively small number of all traffic signs (around 50 categories out of several hundred) and performance on the remaining set of traffic signs, which are required to eliminate the manual labor in traffic-sign inventory management, remains an open question. In this paper, we address the issue of detecting and recognizing a large number of traffic-sign categories suitable for automating traffic-sign inventory management. We adopt a convolutional neural network (CNN) approach, the Mask R-CNN, to address the full pipeline of detection and recognition with automatic end-to-end learning. We propose several improvements that are evaluated on the detection of traffic signs and result in an improved overall performance. This approach is applied to detection of 200 traffic-sign categories represented in our novel dataset.

Link:

https://github.com/skokec/detectron-traffic-signs

Result: *Results are reported on highly challenging traffic-sign categories that have not yet been considered in previous works. We provide comprehensive analysis of the deep learning method for the detection of traffic signs with large intra-category appearance variation and show below 3% error rates with the proposed approach, which is sufficient for deployment in practical applications of traffic-sign inventory management.*

Solution 3: I*n this work, we propose a novel deep network for traffic sign classification that achieves outstanding performance on GTSRB surpassing all previous methods. Our deep network* *consists of spatial transformer layers and a modified version of inception module specifically designed for capturing local and global features together. This features adoption allows our network to classify precisely intraclass samples even under deformations. Use of spatial transformer layer makes this network more robust to deformations such as translation, rotation, scaling of input images. Unlike existing approaches that are developed with hand-crafted features, multiple deep networks with huge parameters and data augmentations, our method addresses the concern of exploding parameters and augmentations.*

Link:

https://github.com/vxy10/p2-TrafficSigns

Result:

In this project they achieved the state-of-the-art performance of 99.81\% on GTSRB dataset which is very appreciating.

Solution 4:

The German Traffic Sign Benchmark is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011. We cordially invite researchers from relevant fields to participate: The competition is designed to allow for participation without special domain knowledge. Our benchmark has the following properties:

1. Single-image,
2. Multi-class classification problem
3. More than 40 classes
4. More than 50,000 images in total
5. Large, lifelike database

Link:

https://www.kaggle.com/pritamaich/traffic-sign-classification-using-cnn

Result: *In this project they achieved an overall accuracy of 97% on our model. This is pretty good and we can use this model for predicting some other Traffic signs as well in future.*

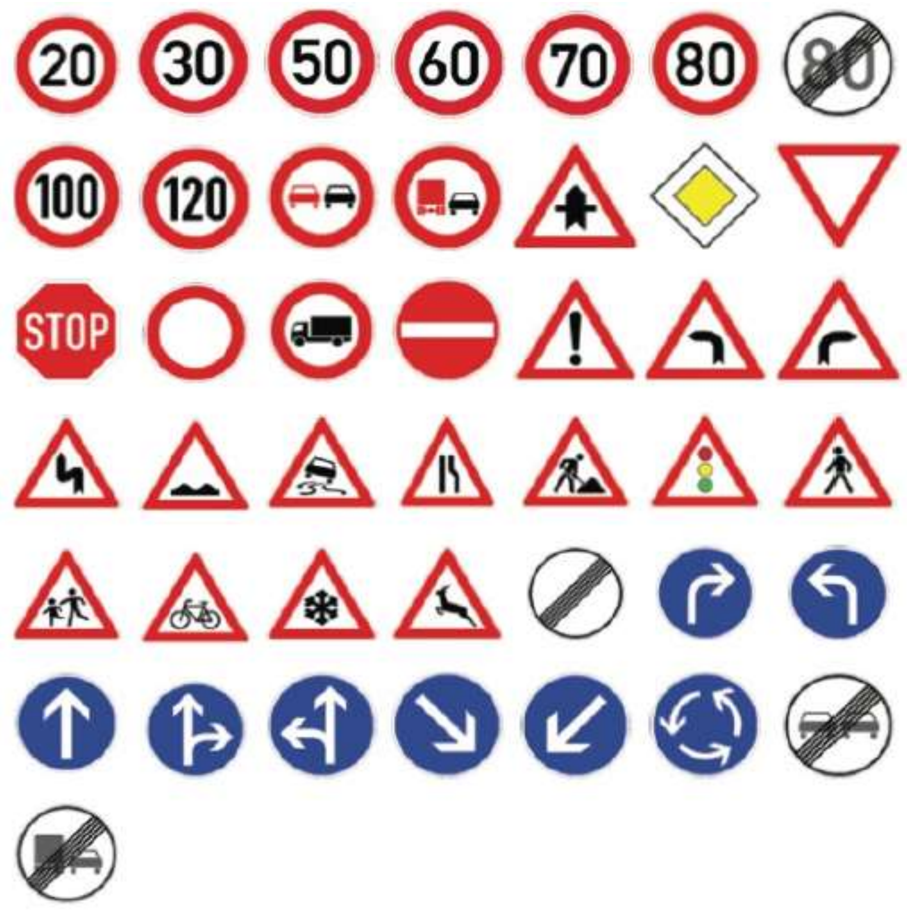
Solution 5: *In this project we use the GTSRB - German Traffic Sign Recognition Benchmark (https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign) dataset to train a Convolutional Neural Network perform single-image, multi-class classification. The 'Train' folder contains 43 folders each representing a different class of image. We will create an array* *with respective data and labels for training and validation. Then, model building and training after that, we visualize the model performance and next, we will see how well the model does on unseen test data. Our dataset contains a Test.csv file, we will use it to evaluate our model. The data is in the form of a .csv file so we can use pandas to read in the data and then extract the images and the corresponding labels. We will then perform necessary data preparation like resizing and converting into NumPy array.*

Link:

https://www.kaggle.com/ankandash/99-accuracy-for-traffic-sign-recognition

Result: *In this project, the accuracy of the model is 99%.*

Dataset Details:

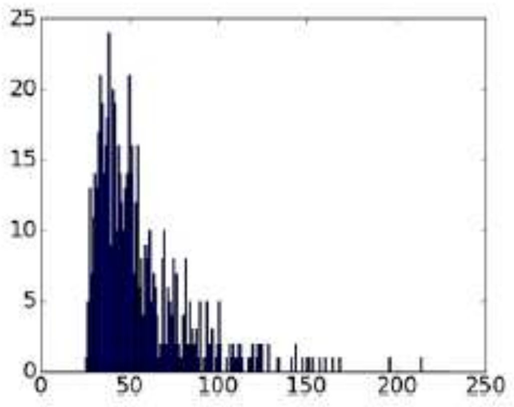


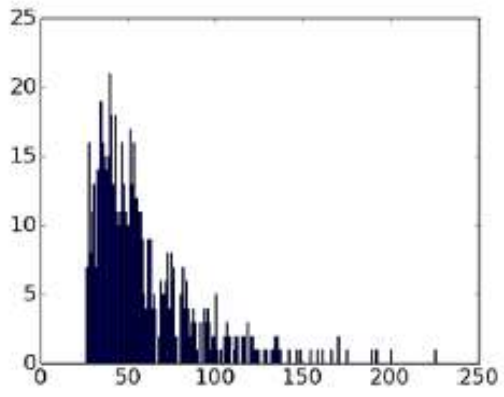
The German Traffic Sign Recognition Benchmark (GTSRB) , was the object of analysis at the classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011. This dataset satisfies two important requirements.

1. it is sufficiently big for training a CNN,
2. it is a good representation of its own classes.

The dataset contains more than 50,000 images of different traffic signs. It is further classified into 43 different classes. Traffic sign recognition (TSR) is typically a multi-class classification challenge which requires to cope with unbalanced class frequencies. For instance, it is easy to realize that a 50 km/h speed limit is more frequent than a 120 km/h one. Also, a TSR dataset should account for situations of different illumination changes, partial occlusions, rotations, weather conditions etc. It consists of highly uneven classes in terms of number of samples for each class. The images reflect the strong variations in visual appearance of signs due to distance, illumination, weather, partial occlusions, and rotations. Images are stored in PPM format (Portable Pixmap, P6) in separate locations for each class. On Figure it is possible to see the histograms of the heights and widths of all GTSRB images. The dataset is quite varying, some of the classes have many images while some classes have few images. The size of the dataset is around 600 MB. The dataset has a train folder which contains images inside each class and a test folder which we will use for testing our model. In this paper, the choice of deep learning for a supervised learning approach is done by design because even though basic traffic signs are limited yet combined with road signs, street name signs, etc. the dataset becomes larger with endless possibilities. The ultimate goal is to have a system fitted into cars and that can detect and recognize any traffic sign to assist the driver or assist in the self-driving process. With deep learning algorithms, unlabeled data can be used and the system can extract features automatically without human intervention.

The German Traffic Sign Recognition Benchmark was used to train in a ***supervised*** *way the CNN model. German Traffic Sign Recognition Benchmark is using* ***multi-class*** *classification.*





On the left: histogram of heights of GTSRB. **On the right**: histogram of widths of GTSRB.

Their sizes vary between 15x15 to 250x250 pixels (not necessarily squared). The actual traffic sign is also not necessarily centered within the image. Averaging the size of the dataset (computed over shapes of the images) forces the resizing method to make use of different scaling factors. Given that training and testing images are 32x32 their size is roughly halved on average, although some of the smallest are *almost unchanged. Most images contain a border of about 10% around the actual traffic sign (5 pixels or more on average) that allow for edge-based segmentation approaches. The samples of each class have been randomly split into training and testing data, to maintain a proportion of roughly 70% of images as training data and 30% as testing data.* All 43 benchmark classes in GTSRB.

Class num.	**Class name**	**Traffic sign**	**IJCNN 2011 class group**	**Total available samples**
00	20 Speed limit		red-round, speed	210
01	30 Speed limit		red-round, speed	2220
02	50 Speed limit		red-round, speed	2250
03	60 Speed limit		red-round, speed	1410
04	70 Speed limit		red-round, speed	1980
05	80 Speed limit		red-round, speed	1860
06	End of 80 speed limit		end-of	420
07	100 Speed limit		red-round, speed	1440
08	120 Speed limit		red-round, speed	1410
09	No overtaking		red-round, red-other	1470
10	No overtaking by heavy vehicles		red-round, red-other	2010
11	Crossroads		danger	1320
12	Priority road		spezial	2500
13	Give way		spezial	2510
14	Stop and give way		spezial	780
15	Restricted vehicular access		red-round, red-other	630
16	No large heavy vehicles		red-round, red-other	420
17	No entry for vehicular traffic		spezial	1110
18	Other danger		danger	1200
19	Bend, to left		danger	210
20	Bend, to right		danger	360
21	Double bend, first to left		danger	330
22	Uneven road		danger	390
23	Slippery road		danger	510
24	Road narrows on right		danger	270
25	Road works		danger	1500
26	Traffic lights		danger	600
27	Pedestrians		danger	240
28	Children crossing		danger	540
29	Bicycles crossing		danger	270
30	Snow		danger	450
31	Animal crossing		danger	780
32	End of all speed limits		end-of	240
33	Turn right ahead		blue	659
34	Turn left ahead		blue	423
35	Ahead only		blue	1200
36	Go straight or right		blue	390
37	Go straight or left		blue	210
38	Keep right		blue	2070
39	Keep left		blue	300
40	Roundabout mandatory		blue	360
41	End of no overtaking		end-of	240
42	End of no overtaking by trucks		end-of	240

Challenges with the dataset: There are a number of challenges in the GTSRB dataset, the first being that images are low resolution**,** and worse, have poor contrast. These images are pixelated, and in some cases, it's extremely challenging, if not impossible, for the human eye and brain to recognize the sign. The second challenge with the dataset is handling class skew.

In order to successfully train an accurate traffic sign classifier we'll need to devise an experiment that can:

- Pre-processing our input images to improve contrast.
- Account for class label skew.

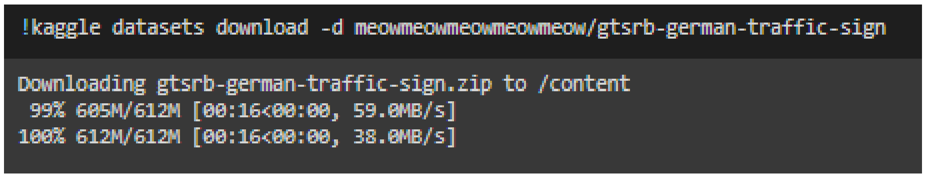
Algorithm:

A. ***Pre-Processing:***

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves below steps:

- *Getting the dataset*



```
!kaggle datasets download -d meowmeowmeowmeowmeow/gtsrb-german-traffic-sign
```

```
Downloading gtsrb-german-traffic-sign.zip to /content
 99% 605M/612M [00:16<00:00, 59.0MB/s]
100% 612M/612M [00:16<00:00, 38.0MB/s]
```

- *Importing libraries*

```
import numpy as np
import pandas as pd
import os
import cv2
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from PIL import Image
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score
np.random.seed(42)

from matplotlib import style
style.use('fivethirtyeight')
```

- *Importing datasets*

```
data_dir = '/content'
train_path = '/content/Train'
test_path = '/content/Test'

# Resizing the images to 30x30x3
IMG_HEIGHT = 30
IMG_WIDTH = 30
channels = 3
```

- *Finding Missing Data*

German Traffic Sign Recognition Benchmark data contain images that's why it has no missing value.

- *Splitting dataset into training and test set*

```
image_data = []
image_labels = []

for i in range(NUM_CATEGORIES):
    path = data_dir + '/Train/' + str(i)
    images = os.listdir(path)

    for img in images:
        try:
            image = cv2.imread(path + '/' + img)
            image_fromarray = Image.fromarray(image, 'RGB')
            resize_image = image_fromarray.resize((IMG_HEIGHT, IMG_WIDH
))
            image_data.append(np.array(resize_image))
            image_labels.append(i)
        except:
            print("Error in " + img)

# Changing the list to numpy array
image_data = np.array(image_data)
image_labels = np.array(image_labels)
```

```python
print(image_data.shape, image_labels.shape)
```

```python
shuffle_indexes = np.arange(image_data.shape[0])
np.random.shuffle(shuffle_indexes)
image_data = image_data[shuffle_indexes]
image_labels = image_labels[shuffle_indexes]
```

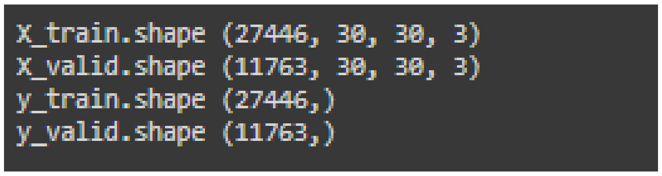
Splitting the data into train and validation set

```python
X_train, X_val, y_train, y_val = train_test_split(image_data, image_labels
, test_size=0.3, random_state=42, shuffle=True)

X_train = X_train/255
X_val = X_val/255

print("X_train.shape", X_train.shape)
print("X_valid.shape", X_val.shape)
print("y_train.shape", y_train.shape)
print("y_valid.shape", y_val.shape)
```

OUTPUT:



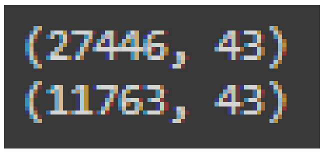
```
X_train.shape (27446, 30, 30, 3)
X_valid.shape (11763, 30, 30, 3)
y_train.shape (27446,)
y_valid.shape (11763,)
```

- *Encoding Categorical Data*

```python
# One Hot encoding The Labels
y_train = keras.utils.to_categorical(y_train, NUM_CATEGORIES)
y_val = keras.utils.to_categorical(y_val, NUM_CATEGORIES)

print(y_train.shape)
print(y_val.shape)
```

OUTPUT:

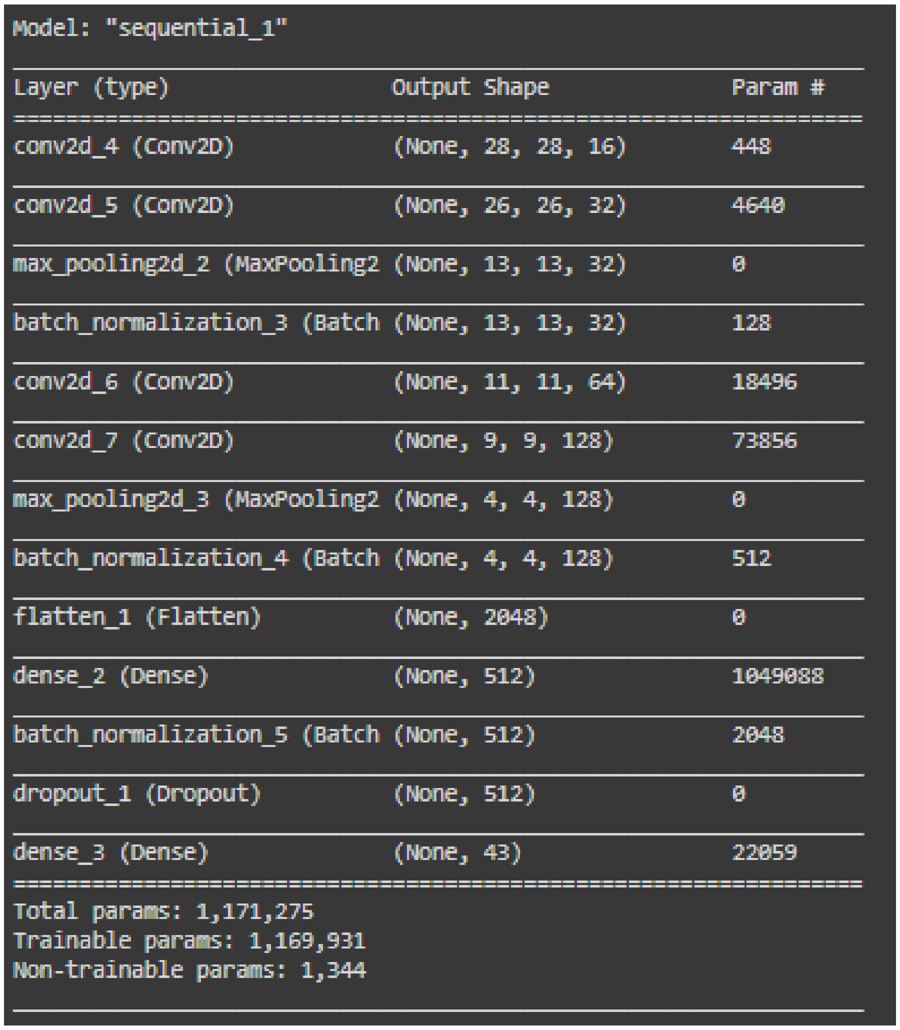


```
(27446, 43)
(11763, 43)
```

B. <u>*Model Architecture and Details*</u>

Here, we are going to import 'keras' and using convolution neural network (CNN) layer to make and train our model in good way, in our first step we have used filter size 16, kernel size was (3,3) and passed input shape of height, width of images and also include the channels. After that we used filter size 32 with same kernel size and activation we have used "relu" for both steps. After that, we have used Max_Pooling with pooling size of (2,2), and apply normalization using column(axis=-1). Next, we apply same steps with different filter size (like: 64,128). In last of making model, we use dense layer with softmax activation. The output shape of the Dense layer is affected by the number of neuron / units specified in the Dense layer.

The summary of our model is given below:



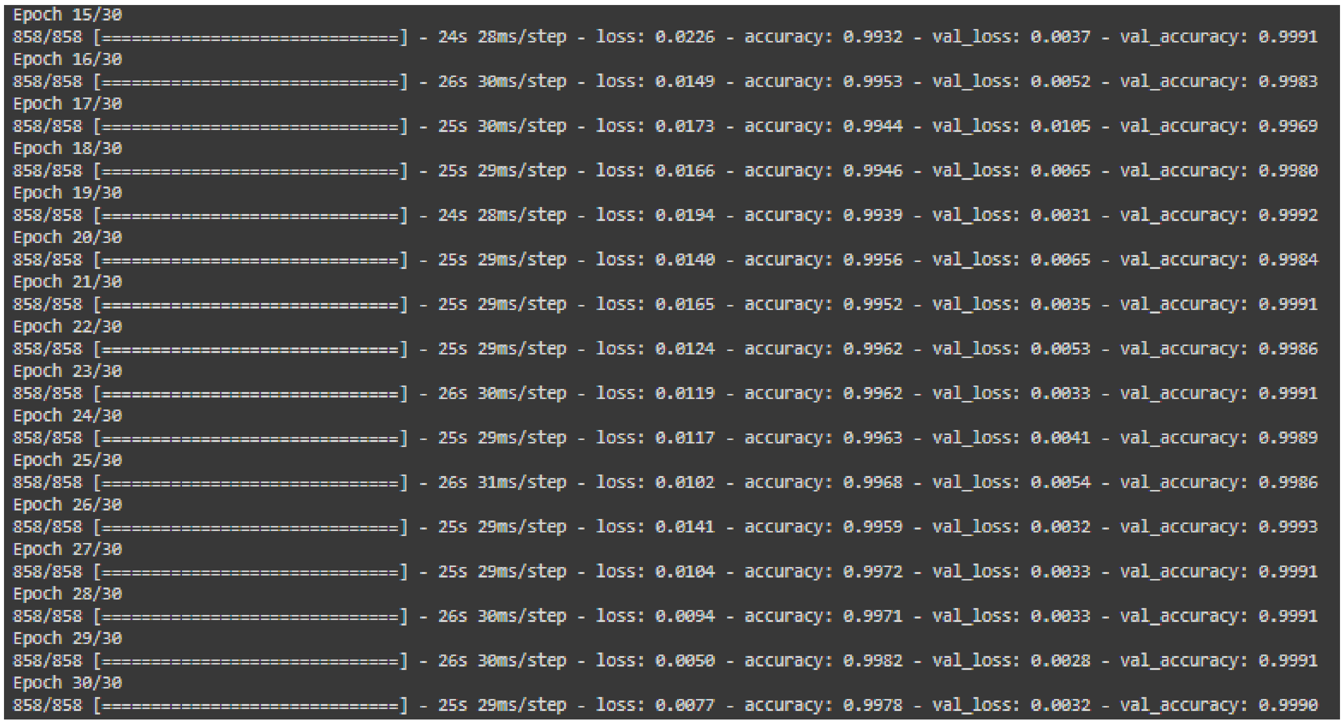
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 28, 28, 16)        448
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 26, 26, 32)        4640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 13, 13, 32)        128
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 9, 9, 128)         73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 4, 4, 128)         512
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               1049088
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 43)                22059
=================================================================
Total params: 1,171,275
Trainable params: 1,169,931
Non-trainable params: 1,344
_________________________________________________________________
```

And then, we use "categorical crossentropy" loss function.

For augmenting the data and train the model using below code:

```
aug = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.15,
    horizontal_flip=False,
    vertical_flip=False,
    fill_mode="nearest")
# loss
history = model.fit(aug.flow(X_train, y_train, batch_size=32), epochs=epoc
hs, validation_data=(X_val, y_val))
```

after that we are execute these command and output is:

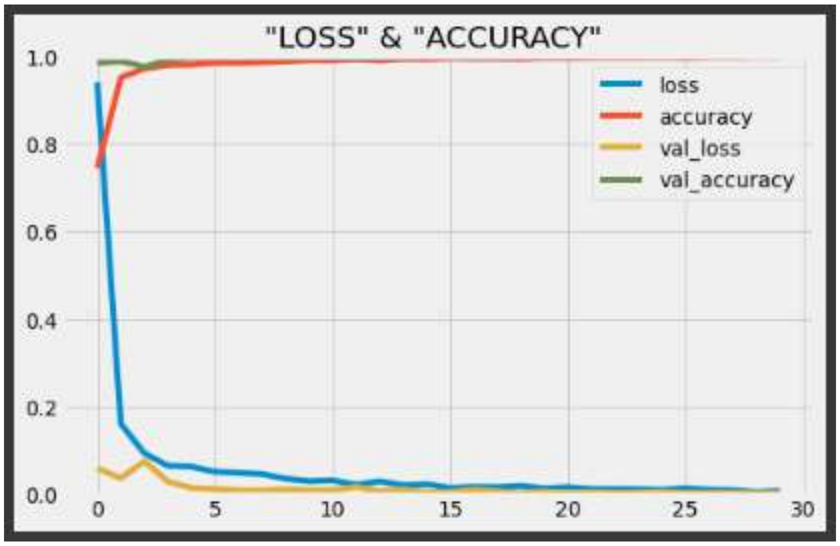


```
Epoch 15/30
858/858 [==============================] - 24s 28ms/step - loss: 0.0226 - accuracy: 0.9932 - val_loss: 0.0037 - val_accuracy: 0.9991
Epoch 16/30
858/858 [==============================] - 26s 30ms/step - loss: 0.0149 - accuracy: 0.9953 - val_loss: 0.0052 - val_accuracy: 0.9983
Epoch 17/30
858/858 [==============================] - 25s 30ms/step - loss: 0.0173 - accuracy: 0.9944 - val_loss: 0.0105 - val_accuracy: 0.9969
Epoch 18/30
858/858 [==============================] - 25s 29ms/step - loss: 0.0166 - accuracy: 0.9946 - val_loss: 0.0065 - val_accuracy: 0.9980
Epoch 19/30
858/858 [==============================] - 24s 28ms/step - loss: 0.0194 - accuracy: 0.9939 - val_loss: 0.0031 - val_accuracy: 0.9992
Epoch 20/30
858/858 [==============================] - 25s 29ms/step - loss: 0.0140 - accuracy: 0.9956 - val_loss: 0.0065 - val_accuracy: 0.9984
Epoch 21/30
858/858 [==============================] - 25s 29ms/step - loss: 0.0165 - accuracy: 0.9952 - val_loss: 0.0035 - val_accuracy: 0.9991
Epoch 22/30
858/858 [==============================] - 25s 29ms/step - loss: 0.0124 - accuracy: 0.9962 - val_loss: 0.0053 - val_accuracy: 0.9986
Epoch 23/30
858/858 [==============================] - 26s 30ms/step - loss: 0.0119 - accuracy: 0.9962 - val_loss: 0.0033 - val_accuracy: 0.9991
Epoch 24/30
858/858 [==============================] - 25s 29ms/step - loss: 0.0117 - accuracy: 0.9963 - val_loss: 0.0041 - val_accuracy: 0.9989
Epoch 25/30
858/858 [==============================] - 26s 31ms/step - loss: 0.0102 - accuracy: 0.9968 - val_loss: 0.0054 - val_accuracy: 0.9986
Epoch 26/30
858/858 [==============================] - 25s 29ms/step - loss: 0.0141 - accuracy: 0.9959 - val_loss: 0.0032 - val_accuracy: 0.9993
Epoch 27/30
858/858 [==============================] - 25s 29ms/step - loss: 0.0104 - accuracy: 0.9972 - val_loss: 0.0033 - val_accuracy: 0.9991
Epoch 28/30
858/858 [==============================] - 26s 30ms/step - loss: 0.0094 - accuracy: 0.9971 - val_loss: 0.0033 - val_accuracy: 0.9991
Epoch 29/30
858/858 [==============================] - 26s 30ms/step - loss: 0.0050 - accuracy: 0.9982 - val_loss: 0.0028 - val_accuracy: 0.9991
Epoch 30/30
858/858 [==============================] - 25s 29ms/step - loss: 0.0077 - accuracy: 0.9978 - val_loss: 0.0032 - val_accuracy: 0.9990
```

We are showing half of output for sample purpose, and it shows us "loss" and "accuracy" ratio.

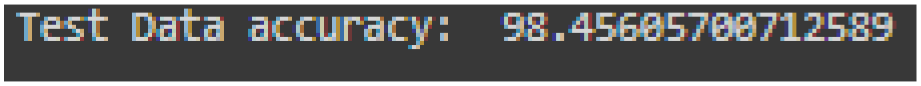
Result:

If we evaluate the model in graph its show like that



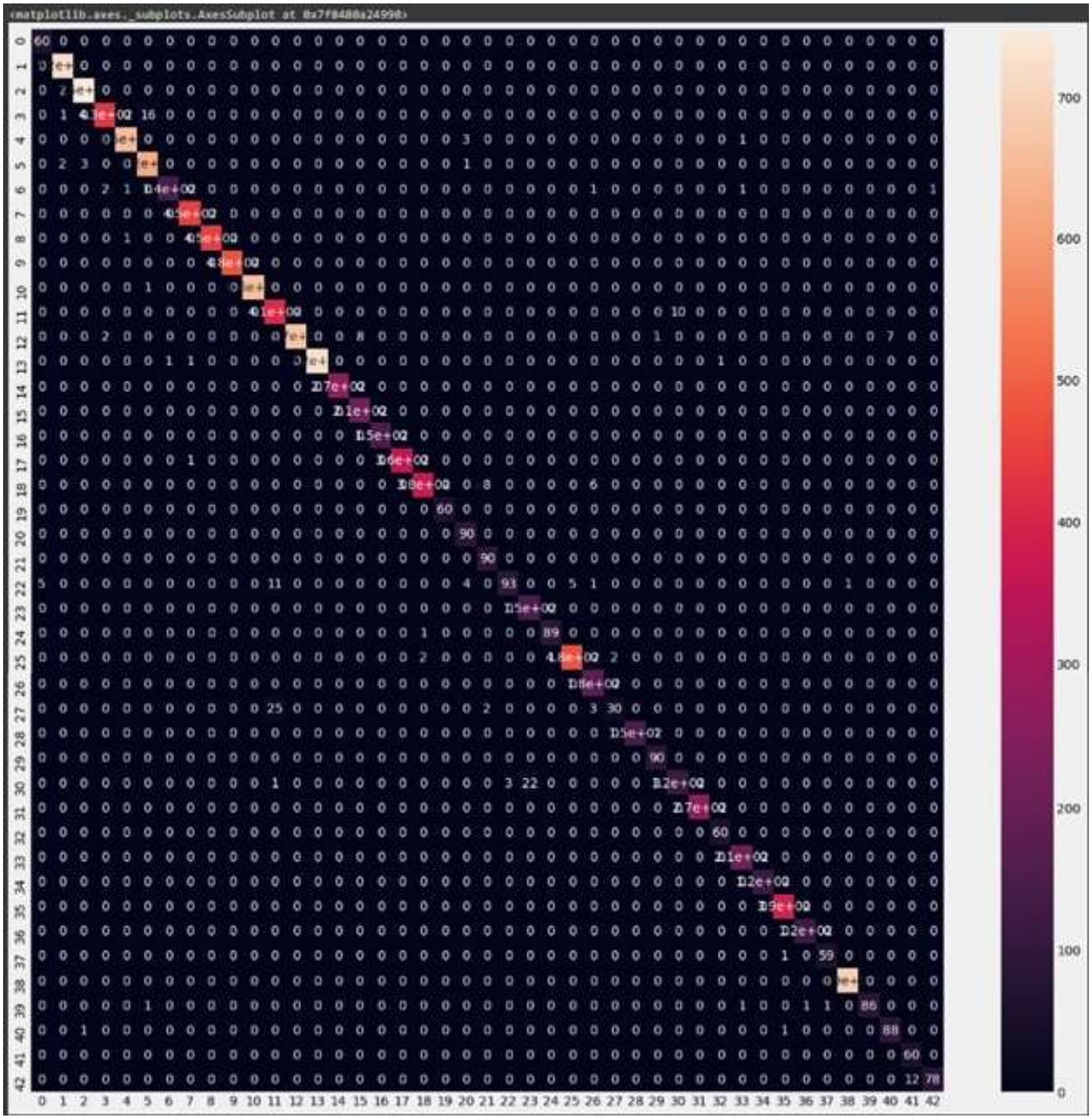
<u>Accuracy:</u>

*we achieve **98.456%** accuracy and it is considered among best accuracies.*



```
Test Data accuracy:  98.45605700712589
```

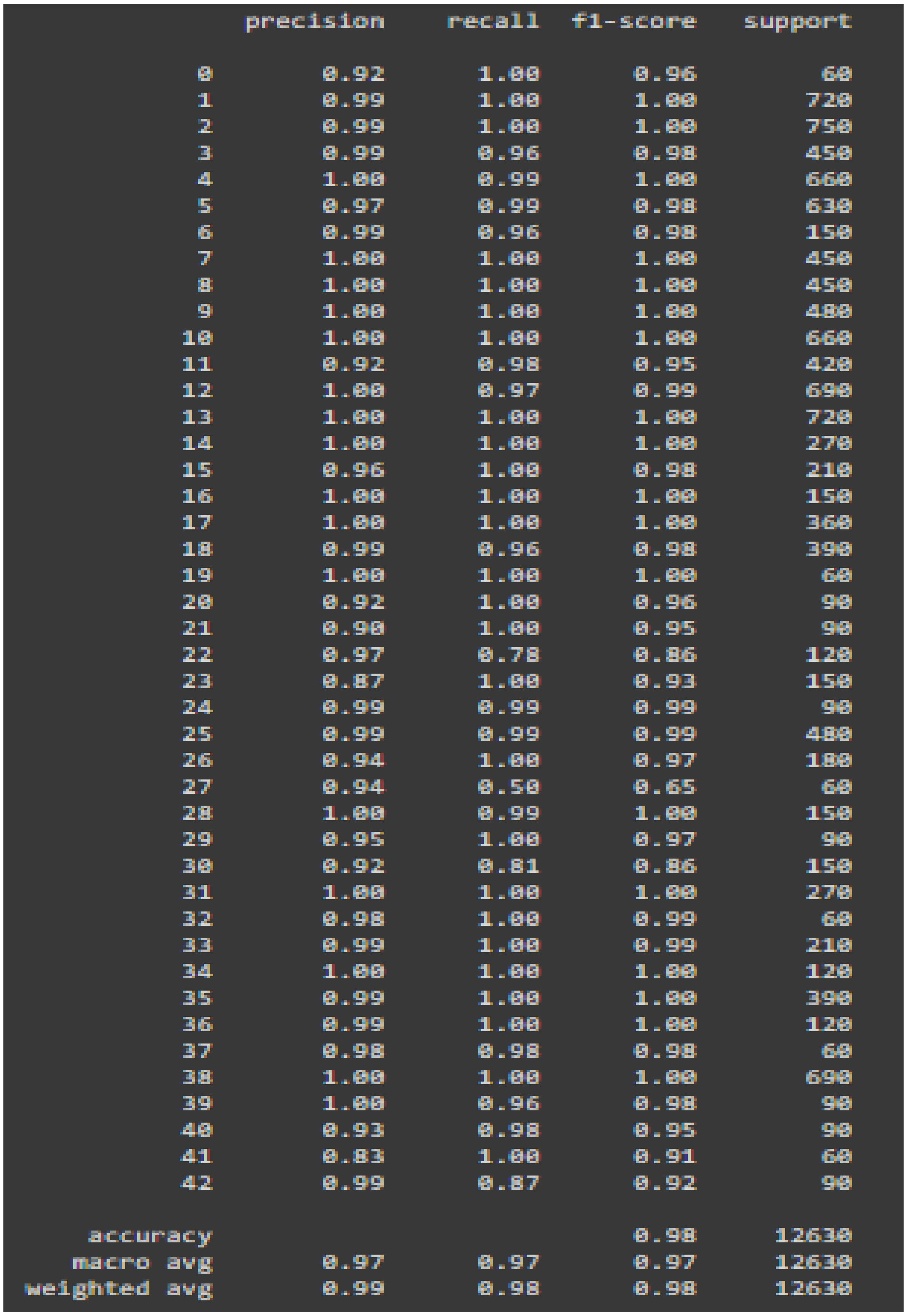
Confusion Metrix:



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8488a24990>
```

<u>Classification Report:</u>

It shows precision, recall, f1-score and support values.



```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        60
           1       0.99      1.00      1.00       720
           2       0.99      1.00      1.00       750
           3       0.99      0.96      0.98       450
           4       1.00      0.99      1.00       660
           5       0.97      0.99      0.98       630
           6       0.99      0.96      0.98       150
           7       1.00      1.00      1.00       450
           8       1.00      1.00      1.00       450
           9       1.00      1.00      1.00       480
          10       1.00      1.00      1.00       660
          11       0.92      0.98      0.95       420
          12       1.00      0.97      0.99       690
          13       1.00      1.00      1.00       720
          14       1.00      1.00      1.00       270
          15       0.96      1.00      0.98       210
          16       1.00      1.00      1.00       150
          17       1.00      1.00      1.00       360
          18       0.99      0.96      0.98       390
          19       1.00      1.00      1.00        60
          20       0.92      1.00      0.96        90
          21       0.90      1.00      0.95        90
          22       0.97      0.78      0.86       120
          23       0.87      1.00      0.93       150
          24       0.99      0.99      0.99        90
          25       0.99      0.99      0.99       480
          26       0.94      1.00      0.97       180
          27       0.94      0.50      0.65        60
          28       1.00      0.99      1.00       150
          29       0.95      1.00      0.97        90
          30       0.92      0.81      0.86       150
          31       1.00      1.00      1.00       270
          32       0.98      1.00      0.99        60
          33       0.99      1.00      0.99       210
          34       1.00      1.00      1.00       120
          35       0.99      1.00      1.00       390
          36       0.99      1.00      1.00       120
          37       0.98      0.98      0.98        60
          38       1.00      1.00      1.00       690
          39       1.00      0.96      0.98        90
          40       0.93      0.98      0.95        90
          41       0.83      1.00      0.91        60
          42       0.99      0.87      0.92        90

    accuracy                           0.98     12630
   macro avg       0.97      0.97      0.97     12630
weighted avg       0.99      0.98      0.98     12630
```

Prediction on Test Data:

Here, are some predicted results from test dataset.