TRAFFIC SIGN NEURAL NETWORK REPORT

By Vahin Reddy, B.Tech Mechatronics – VIth Sem

(1). Introduction

In this report, I will cover how I modelled an image recognition model using a given data set and how it was trained and tested using sample images of traffic signs. By using the means of Keras from TensorFlow, we can construct a Convolution Neural Network model that can be trained over multiple iterations to understand and classify data into predetermined classes. Using this data, we can input our own sample data and receive what traffic sign is on the provided image.

Using this report, I will go through the step-by-step procedures, results found after creating the model, limitations of the model, and potential improvements that could improve the model.

(2). Procedure

To start, we will divide the procedures into 2 parts when creating the Convolution Neural Network model from scratch, verifying the prediction, and applying local pictures. This will include:

- Part 1 – Model Creation
    - Importing all necessary libraries
    - Assign the dataset path to variables
    - Assigning all possible signs into a dictionary
    - Visualizing the dataset
    - Collecting the Training Data
    - Shuffling the training data
    - Splitting the data into train and validation set
    - Encoding the labels
    - Making the model
    - Evaluating Model performance

- Part 2 – Testing Model
  - Loading the Test data and running predictions
  - Testing using field data

With reference to YouTube video, "Traffic Signs Recognition with 95% Accuracy using CNN & Keras" [1] [2], the method to create this Convolution Neural Network was created.

Language: Python

Libraries: NumPy, Pandas, OpenCV, Matplotlib, TensorFlow

GitHub Link: https://github.com/CabinOnTheIsle/TSignRecognition_VReddy.git

Part 1 – Model Creation

(3). Importing Libraries

Using the five libraries – NumPy, Pandas, Cv2, Matplotlib.pyplot, TensorFlow - we can extract data, process data, and output data in a desirable manner. Keras from TensorFlow is what we will use to construct our deep neural network that can recognize different traffic signs that are visible on many roads. To represent our data, we will use

the Matplot style ‘fivethirtyeight’. This style helps differentiate the accuracy and losses from each other using vivid colors.

(4). Dataset path to variables

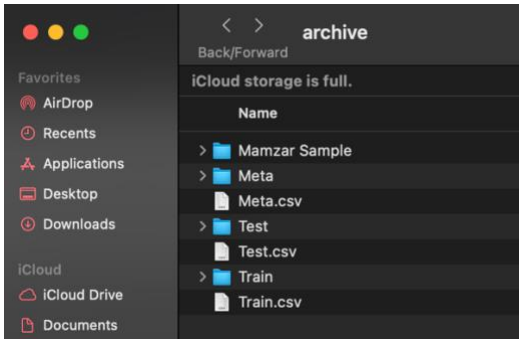

Fig. 1: Image of workspace directory

Using the German Traffic Sign Recognition Benchmark on Kaggle [3], we can train the Convolution Neural Network with multiple traffic signs. We save the files in the workspace folder. The train and test path are within the folder.

The images collected will be resized to 30 by 30 pixels long and will consist of 3 channels of color – RGB.

(5). Classification into dictionary

```
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 vehicles over 3.5 tons',
            11:'Right-of-way at intersection',
            12:'Priority road',
            13:'Yield',
            14:'Stop',
            15:'No vehicle',
            16:'Vehicles > 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 vehicles > 3.5 tons'}
```

Fig. 2: Categories of Traffic signs

The traffic signs in the data set can be classified into 43 categories. Each of the 43 signs correspond to individual traffic signs that will help categorize the training and testing data.

(6). Visualizing the dataset

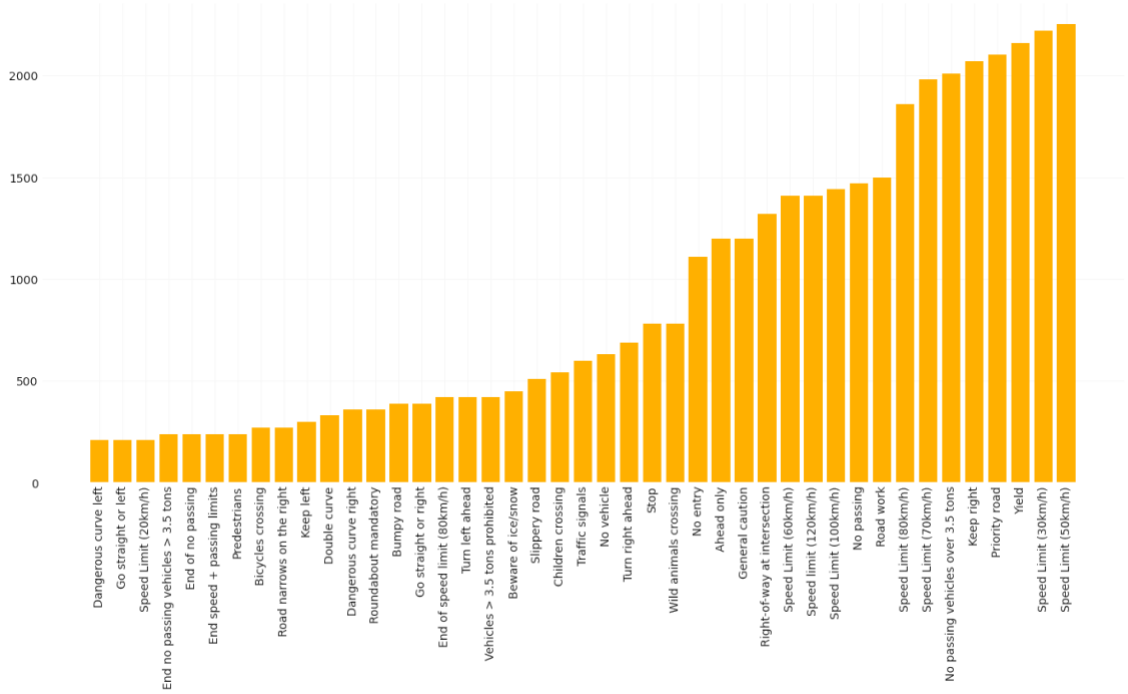

Fig. 3: Histogram of Image density per category

By visualizing the dataset using Matplotlib's histogram, we can observe the quantity of data for each of the 43 data categories. The largest number of images in a category in the dataset is the Speed Limit (50km/h), while the least number of images in a category in the dataset is the Dangerous curve left.

(7). Collecting, Shuffling and Splitting data

We start of by collecting data and sorting all the train sub dataset into 2 variables: image_data and image_label. In total, there are 39209 images that were used to train the data set.

We then procced to shuffling the training data within the two variables stated above. This is crucial as it will minimize variance and ensure that the model will generalize well to new, undiscovered data points.

Lastly, we split the data into 2 sets: Train set and Validation set. This will help the model train the neural network and validate which of the images fall into which category.

Before the model is created, the data is labeled.

(8). Making the model

```
Epoch 1/30
858/858 [==============================] - 47s 54ms/step - loss: 1.1508 - accuracy: 0.6925 - val_loss: 0.0853 - val_a
ccuracy: 0.9723
Epoch 2/30
858/858 [==============================] - 49s 57ms/step - loss: 0.1865 - accuracy: 0.9434 - val_loss: 0.0271 - val_a
ccuracy: 0.9925
Epoch 3/30
858/858 [==============================] - 51s 59ms/step - loss: 0.1019 - accuracy: 0.9697 - val_loss: 0.0196 - val_a
ccuracy: 0.9949
Epoch 4/30
858/858 [==============================] - 51s 60ms/step - loss: 0.0729 - accuracy: 0.9780 - val_loss: 0.0154 - val_a
ccuracy: 0.9957
Epoch 5/30
858/858 [==============================] - 49s 57ms/step - loss: 0.0661 - accuracy: 0.9792 - val_loss: 0.0969 - val_a
ccuracy: 0.9795
Epoch 6/30
858/858 [==============================] - 53s 62ms/step - loss: 0.0571 - accuracy: 0.9822 - val_loss: 0.0176 - val_a
ccuracy: 0.9952
Epoch 7/30
858/858 [==============================] - 46s 53ms/step - loss: 0.0517 - accuracy: 0.9842 - val_loss: 0.0189 - val_a
ccuracy: 0.9942
Epoch 8/30
858/858 [==============================] - 46s 53ms/step - loss: 0.0436 - accuracy: 0.9858 - val_loss: 0.0076 - val_a
ccuracy: 0.9976
Epoch 9/30
858/858 [==============================] - 46s 53ms/step - loss: 0.0363 - accuracy: 0.9888 - val_loss: 0.0176 - val_a
ccuracy: 0.9946
Epoch 10/30
858/858 [==============================] - 46s 54ms/step - loss: 0.0386 - accuracy: 0.9876 - val_loss: 0.0056 - val_a
ccuracy: 0.9987
Epoch 11/30
858/858 [==============================] - 47s 55ms/step - loss: 0.0346 - accuracy: 0.9899 - val_loss: 0.0113 - val_a
ccuracy: 0.9969
Epoch 12/30
858/858 [==============================] - 48s 56ms/step - loss: 0.0271 - accuracy: 0.9918 - val_loss: 0.0087 - val_a
ccuracy: 0.9975
Epoch 13/30
858/858 [==============================] - 51s 60ms/step - loss: 0.0243 - accuracy: 0.9926 - val_loss: 0.0067 - val_a
ccuracy: 0.9982
```

Fig. 4: Iterations of learning performed by model

We define the model to a variable and use the keras.models.Sequential() function to construct our model. The model will iterate over 30 times each by approximately 60 seconds each. Through each iteration, the accuracy of the model is increased, and the losses are minimized. If we increase the iterations, the model will improve further in accuracy.

(9). Evaluating Model performance

```
> classification_report
```

	precision	recall	f1-score	support
0	0.92	1.00	0.96	60
1	0.99	1.00	1.00	720
2	1.00	1.00	1.00	750
3	0.99	0.98	0.99	450
4	1.00	0.99	0.99	660
5	0.98	1.00	0.99	630
6	0.99	0.95	0.97	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	1.00	0.98	0.99	420
12	1.00	0.99	1.00	690
13	1.00	1.00	1.00	720
14	0.98	1.00	0.99	270
15	1.00	1.00	1.00	210
16	0.99	1.00	1.00	150
17	1.00	0.98	0.99	360
18	0.99	0.93	0.96	390
19	1.00	1.00	1.00	60
20	0.96	1.00	0.98	90
21	0.92	1.00	0.96	90
22	1.00	0.86	0.92	120
23	0.97	1.00	0.99	150
24	0.99	0.98	0.98	90
25	0.97	0.99	0.98	480
26	0.98	0.99	0.98	180
27	0.86	0.53	0.66	60
28	0.99	0.99	0.99	150
29	1.00	1.00	1.00	90
30	0.84	0.97	0.90	150
31	0.94	0.99	0.97	270
32	0.98	1.00	0.99	60
33	0.98	1.00	0.99	210
34	1.00	1.00	1.00	120
35	0.99	0.99	0.99	390
36	0.98	1.00	0.99	120
37	0.97	1.00	0.98	60
38	1.00	1.00	1.00	690
39	0.99	0.99	0.99	90
40	0.99	0.96	0.97	90
41	1.00	1.00	1.00	60
42	0.98	1.00	0.99	90
accuracy			0.99	12630
macro avg	0.98	0.98	0.98	12630
weighted avg	0.99	0.99	0.99	12630

Fig. 5: Classification report of model over 30 iterations

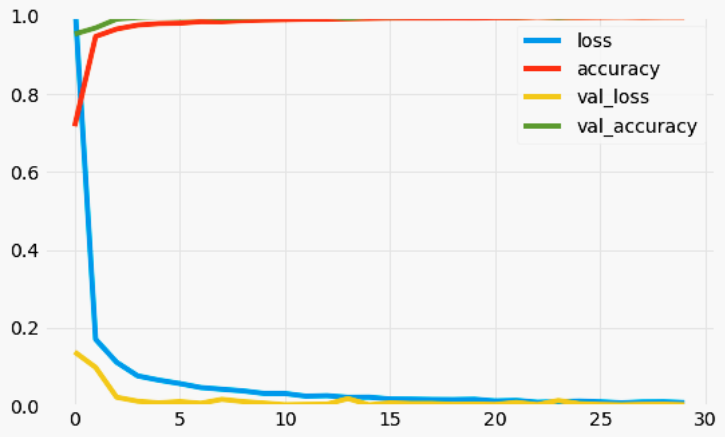

Fig. 6: Graph of accuracy and losses over 30 iterations

As seen in the graph, the model's accuracy and losses grew and shrunk exponentially, respectively, over time.

This graph uses the Matplot style 'fivethirtyeight' to differentiate each of the labelled data in the legend (3).

Part 2 – Testing the Model

(10). Prediction on Test Data

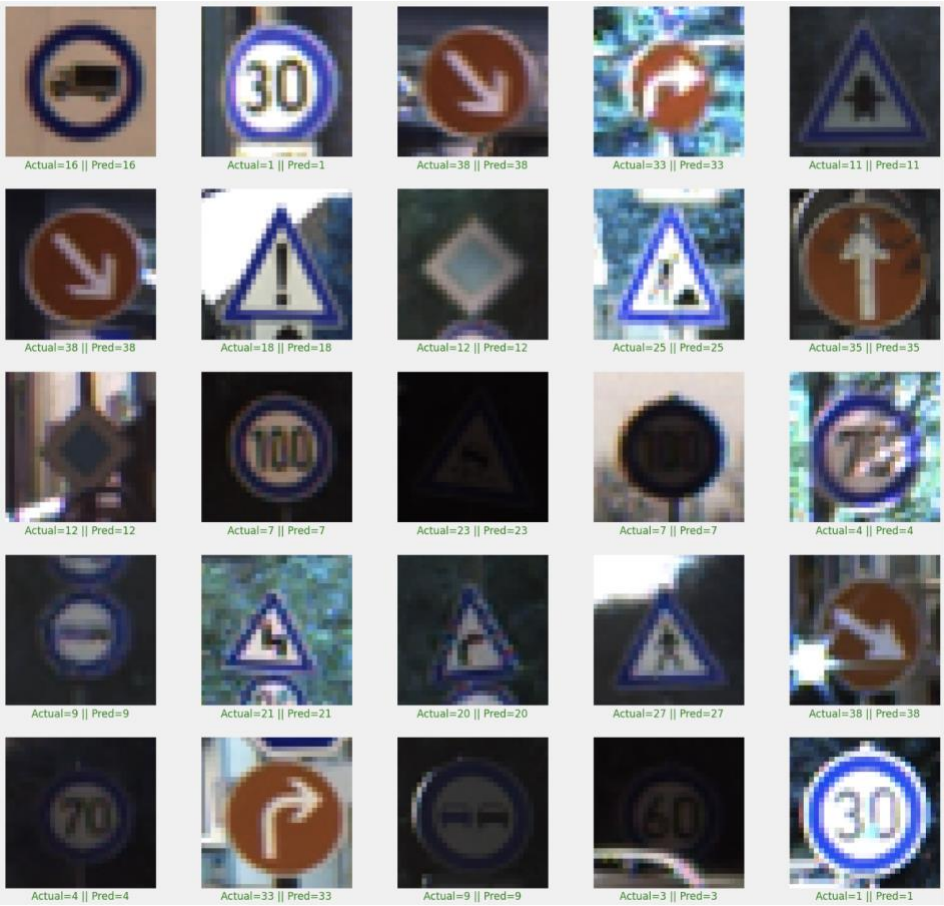

Fig. 7: Actual and predicted category for sample images

After using the model on 25 images provided by the test data, we can observe that the model is able to accurately predict what each of the signs provided are classified under each of the 43 categories. Below each of the pictures shown above, there is an Actual class that the image falls under and the predicted class based of the Neural Network model created.

(11). Application of Model

When using the model on captured images of traffic signs in the local region, Dubai, we can review the performance of the model in real life applications. In total, 15 images were provided to the model and the predicated definition of the sign was the output. Below is the table which consists of the image of the signs, the actual meaning of the sign, the predicated output using the neural network, and lastly weather the prediction was correct or not.

Image	Actual	Prediction	Results
	No Entry	No entry	✓
	Pass either side	Roundabout manda tory	✗
	Bumpy road	Bumpy road	✓
	Roundabout mandatory	Slippery road	✗
	Speed limit (60km/h)	Speed Limit (50k m/h)	✗
	Ahead only	Ahead only	✓
	Stop	Stop	✓
	Yield	Yield	✓
	Turn right ahead	Turn right ahead	✓
	U - Turn	General caution	✗
	Traffic Signal	Traffic signal	✓
	Pedestrians	Traffic signal	✗
	Keep Left	Keep left	✓
	Turn right ahead	Go straight or l eft	✗

(12). Result

In conclusion, we were able to create a convolution neural network to train and understand the traffic signs provided. We started off by importing all the necessary libraries that would help us collect, process, and display the data. The data from the archives from the provided data set was stored, shuffled, and split into training and validating paths. We moved on further by creating the Keras neural network model that would be used in the predicting the test data set provided. With the German Dataset, the Model was able to get a 99% accuracy after the 30 iterations. It was able to predict all 25 out of the 25 images provided by the test sample.

Lastly, we used road signs seen in the local region to analyze the results of the prediction done using the model. The model was able to accurately predict 8 out of the 15 images provided.

Overall, we were able to input a picture to the model and retrieve back the prediction of what the traffic sign within the images was.

(13). Limitation

When using the model on local traffic signs, it was visible that there are a few pitfalls in the model (12). Primarily, since the data set is comprised of German Traffic signs, the neural network model is unable to determine signs from other regions. This maybe evident when using the sign for speed limits where the English and Arabic digits are visible, or when camel crossing signs are provided.
Secondly, the angle or dimensions of the picture, and lighting of the picture may affect the reliability of the prediction. Few of the symbols provided were miss categorized and the output prediction was incorrect.

(14). Improvements

It is possible to improve the neural network model [4] further by:

- Gathering more data
- Adding more layers to the model
- Changing the image sizes
- Increase Epochs
- Decrease Color Channels

(15). Table of figures

Fig. 1: Image of workspace directory 3
Fig. 2: Categories of Traffic signs 3
Fig. 3: Histogram of Image density per category 4
Fig. 4: Iterations of learning performed by model.....5
Fig. 5: Classification report of model over 30 iterations 6
Fig. 6: Graph of accuracy and losses over 30 iterations 6
Fig. 7: Actual and predicted category for sample images.....7

(16). Bibliography

[1] D. K. Gupta, "YouTube," StudyGyaan, 14 June 2021. [Online]. Available: https://www.youtube.com/watch?v=Vtc64rPHZ6I&ab_channel=StudyGyaan.

[2] D. K. Gupta, "GitHub," 24 August 2021. [Online]. Available: https://github.com/DeepranjanG/Traffic_sign_classification/blob/main/Model_training.ipynb.

[3] L. X. T. Wei, Y. P. Chen, Y.-W. Tai and C.-K. Tang, "Fss-1000: A 1000-class dataset for few-shot segmentation," IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020.

[4] J. Dsouza, "How to Improve the Accuracy of Your Image Recognition Models," Free Code Camp, 29 November 2021. [Online]. Available: https://www.freecodecamp.org/news/improve-image-recognition-model-accuracy-with-these-hacks/.