Traffic Sign Recognition

Build a Traffic Sign Recognition Project

The goals / steps of this project are the following:

- Load the data set (see below for links to the project data set)
- Explore, summarize and visualize the data set
- Design, train and test a model architecture
- Use the model to make predictions on new images
- Analyze the softmax probabilities of the new images
- Summarize the results with a written report

Steps to achieve these goals are as follows:

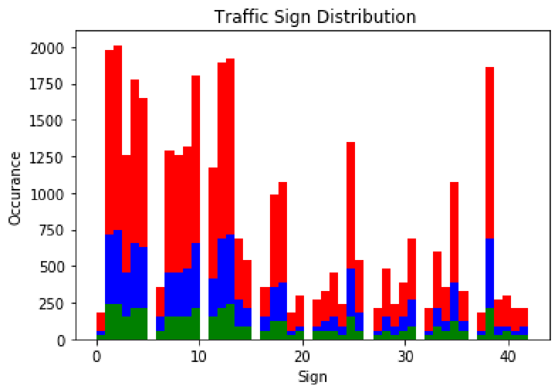
1. Data sets chosen:

Pickle files were provided for train, validation and test data sets. Each pickle file had a set of features and its labels. Using these pickle files, below are the numbers chosen as our dataset:

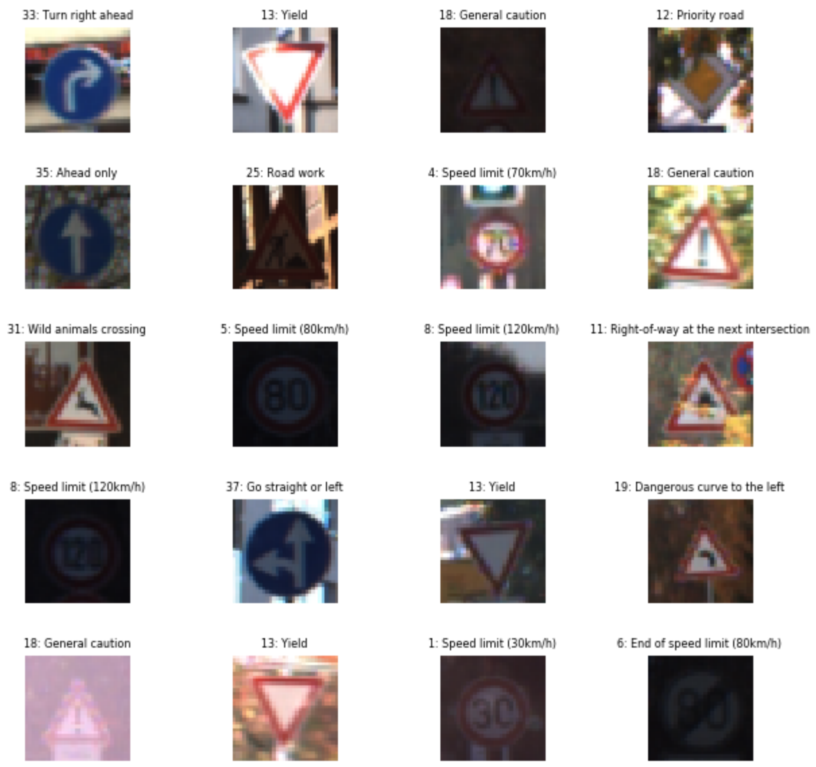
a. The number of training set chosen was 34799.
b. The number of validation set chosen was 4410.
c. The number of test set chosen was 12630.
d. Each of the images from these sets were of the shape 32x32x3.
e. Finally, the number of unique classes/labels chosen from these datasets were 43.

2. Visualize the datasets:

With the help of plot and hist methods in python, the datasets were visualized as below.
Legends:
Red – Training dataset
Blue – Test dataset
Green – Validation dataset



Also with the help of pandas read_csv API, the signnames.csv file is read and then stored the SignName column from it to a list called sign_name_list. As the class Ids start from 0 to 42 in sequential order, the corresponding sign name can be accessed with sign_name_list[class_id]. Finally, random images from the training set are then considered and plotted with its sign name as its title.

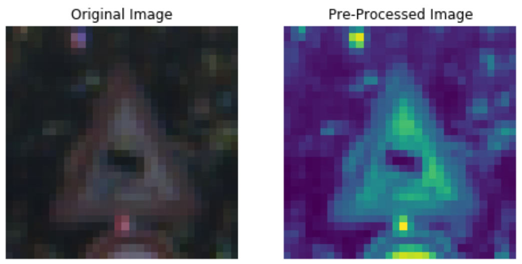


3. Design and Test a Model Architecture:

a. Preprocess the dataset:

Preprocessing are usually optional, but it effectively improves the performance. In our case, the color doesn't matter, so converting the images to grayscale reduces the computation time and thus improves performance. This is done with the help of numpy's sum method which converts the entire array of images to grayscale. Also, normalizing the obtained grayscale image helps in achieving the dataset with mean zero and equal variance and as mentioned in the file, it is done through the formula -> (grayscaled_images – 128)/128. Here is an example of the original image and the preprocessed image.

```
Original images size: (32, 32, 3)
Pre-processed images size: (32, 32)
```



b. Model Architecture:

My final architecture looks like a modified version of LeNet architecture and thus the method is called LeNet_modified. It contains 5 layers as mentioned below in the table and the final output is 43.

Layer	**Input Dimension**	**Description**	**Output Dimension**
Layer 1 - Convolutional	32x32x1	strides=[1, 1, 1, 1], padding='VALID'	28x28x6
	28x28x6	Relu	28x28x6
	28x28x6	Max pooling with ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID'	14x14x6
Layer 2 – Convolutional	14x14x6	strides=[1, 1, 1, 1], padding='VALID'	10x10x16
Activation	10x10x16	Relu	10x10x16
Pooling	10x10x16	Max pooling with ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID'	5x5x16
Flatten	5x5x16	Flattens the input with same batch_size	5x5x16 = 400
Dropout	400	Keep_prob=0.5 for train and 1.0 for test/valid	400
Layer 3 – Fully Connected	400	xW+b	120
Activation	120	Relu	120
Dropout	120	Keep_prob=0.5 for train and 1.0 for test/valid	120
Layer 4 – Fully Connected	120	xW+b	84
Activation	84	Relu	84
Dropout	84	Keep_prob=0.5 for train and 1.0 for test/valid	84

Layer 5 – Fully Connected	84	xW+b	43

c. **Training the Model:**

The code for training the model is located with the comment (# Train the Model on preprocessed images). For training, I used the modified LeNet architecture with dropouts added after the input is flattened in layer 2, after activation in layer 3 and 4. Also, preprocessed images are provided as the input rather than the actual images obtained from the pickle file. I used the below hyper parameters to finally obtain a test accuracy of **94.2%** and validation accuracy of **95.3%.**

EPOCHS = 65
BATCH_SIZE = 128
DROPOUT_PROB = 0.5
EVAL_DROPOUT_PROB = 1.0
LEARNING_RATE = 0.001

d. **Iterative approach to find the solution:**

Following iterative approach was taken to improve the accuracy:

i. With the initial LeNet model architecture, the validation set accuracy came around 89%, however to meet the specification the hyper parameters had to be tuned to improve the accuracy.

ii. Also, with the help of grayscaling and normalizing the accuracy increased by a lot and it also became comparatively faster to compute.

iii. I initially had the EPOCHS set to 32 during which I got a test accuracy of 93.4% and a validation accuracy of 94.8%. However, as I increased the EPOCHS I got a better accuracy until it reached till 65. So, I have it set to 65.

iv. To avoid overfitting, dropout was added into the layer 2, 3 and 4. Dropout probability is set to 0.5 while training but for validating or testing the model, the dropout probability is changed to 1.0. This finally helped to get a good accuracy.

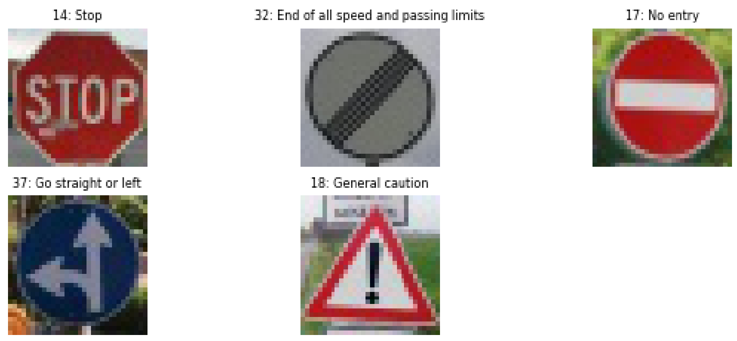
v. Also, I used the AdamOptimizer as it was being done in the lecture with a learning rate of 0.001.

vi. The LeNet architecture is suitable for this operation as the accuracy obtained was good enough to make some good prediction on the new images. However, there might be other architectures which I still need to explore.

e. **Test the model on New images:**

i. Load the New images:

New German traffic signs were downloaded and then added to a directory named german_traffic_signs. These files are made sure to be 32x32x3 so that the code would run without any failures. The file names are renamed with the label in the front and then the sign name (ex: 14_stop.jpg). As per the project, only 5 of those files are considered, these files are then loaded and titled with the signname obtained from the csv file as below:



ii. Predict the sign type from our model:

The images which are loaded are then preprocessed just like before and then passed as input datasets. The model which was trained before is stored in ./lenet and that is restored to make predictions here and the predicted labels are found to be same as the true label [14 32 17 37 18], thus giving a test accuracy of 100%.

iii. Find Top 5 Softmax probabilities:

These new images which are preprocessed are fed as the input and then the top 5 softmax probabilities are found using the tf.nn.softmax and tf.nn.top_k APIs. The tf.nn.top_k provides the top 5 softmax values along with its indices. This is used to determine the percentage of each of the prediction type and “predicted name” is printed for the one which has the maximum prediction out of these 5 softmax values.

```
FileName: 14_stop.jpg
Stop: 63.57%
----------------------> Predicted name
Speed limit (50km/h): 27.15%
Speed limit (60km/h): 2.23%
Speed limit (30km/h): 1.65%
Keep right: 1.37%

FileName: 32_end_speed.jpg
End of all speed and passing limits: 97.84%
----------------------> Predicted name
End of no passing: 1.83%
End of speed limit (80km/h): 0.29%
Go straight or right: 0.03%
End of no passing by vehicles over 3.5 metric tons: 0.01%

FileName: 17_no_entry.jpg
No entry: 100.00%
```

```
---------------------> Predicted name
Stop: 0.00%
Keep right: 0.00%
Speed limit (20km/h): 0.00%
Priority road: 0.00%

FileName: 37_str_or_left.jpg
Go straight or left: 100.00%
---------------------> Predicted name
Roundabout mandatory: 0.00%
Traffic signals: 0.00%
General caution: 0.00%
Right-of-way at the next intersection: 0.00%

FileName: 18_caution.jpg
General caution: 100.00%
---------------------> Predicted name
Traffic signals: 0.00%
Pedestrians: 0.00%
Right-of-way at the next intersection: 0.00%
Children crossing: 0.00%
```

4. Discussions:

1. outputFeatureMap method needs to be used to explore more about each network state
2. Needs to play around more with the hyper parameters to get even more better accuracy
3. Need to study more about other architecture/model to train the datasets.
4. Augmenting the training set might help improve model performance

5. References:

- https://stackoverflow.com/questions/46615554/how-to-display-multiple-images-in-one-figure-correctly
- https://github.com/Goddard/udacity-traffic-sign-classifier