Traffic Sign Recognition Classifier

Build a Traffic Sign Recognition Classifier

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

Rubric Points - Justification

Files Submitted:

- Submission Files :

  Ipython notebook with code - Traffic_Sign_Classifier.ipynb
  HTML output of the code - Traffic_Sign_Classifier.html
  Writeup report

Dataset Exploration:

- Dataset Summary

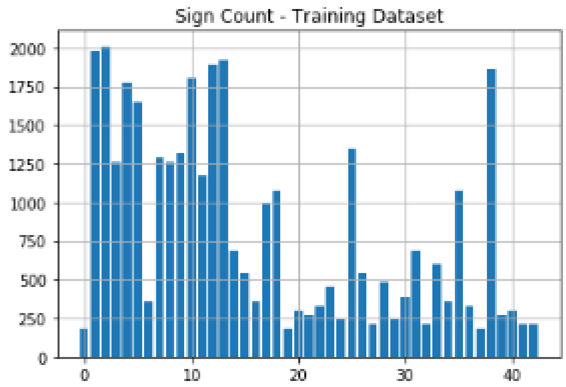
  Loaded the provided training, test & validation dataset. Based on the dataset, identified the unique set of classes.

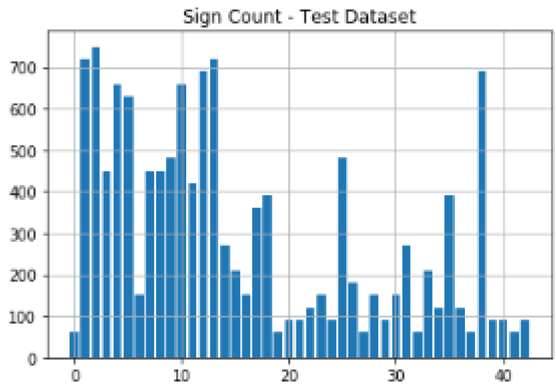
  ```
  Number of training examples = 34799
  Number of testing examples = 12630
  Image data shape = (34799, 32, 32, 3)
  Number of classes = 43
  ```

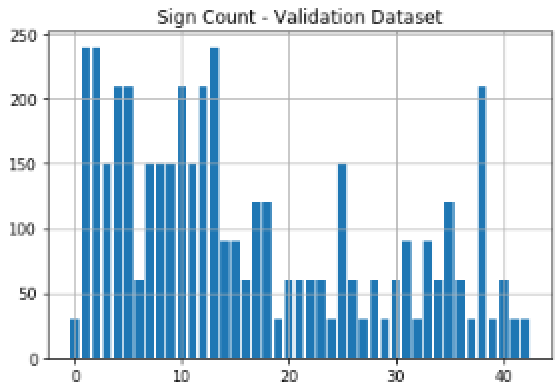
- Exploratory Visualization

  Based on the class identifier, identified corresponding sign names from signnames.csv and plotted few images for reference.

For visualization, grouped the dataset classification class wise and charted using **matplotlib.pyplot.bar** function.

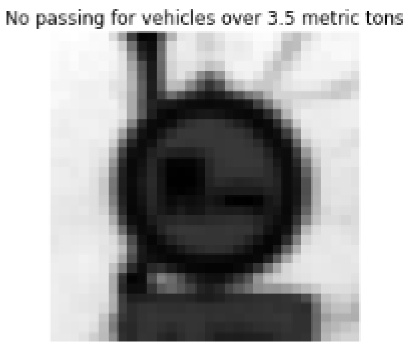


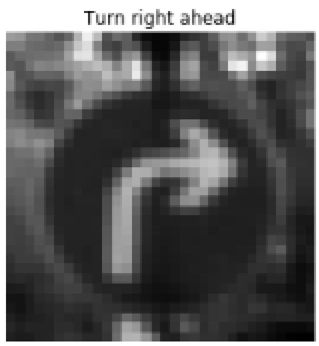




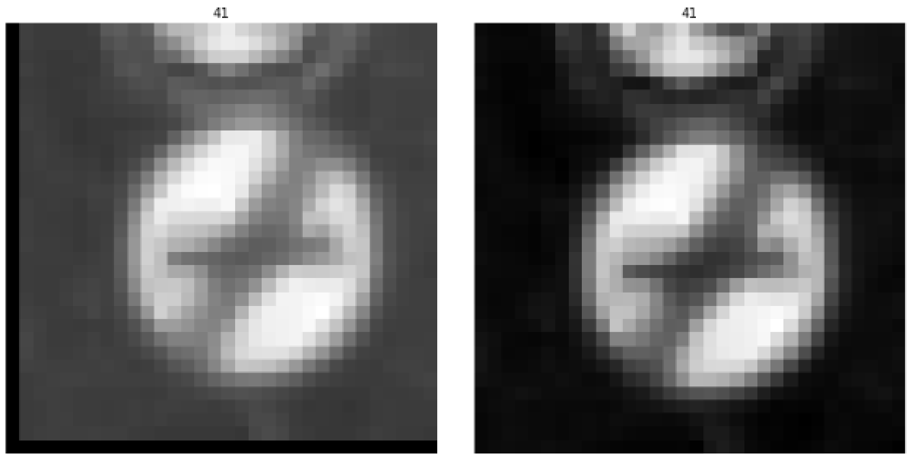
Design and Test a Model Architecture:

- Preprocessing
  Under preprocessing phase, below actions are performed
  - ➢ Grayscale conversion
  - ➢ Visualize the images for reference

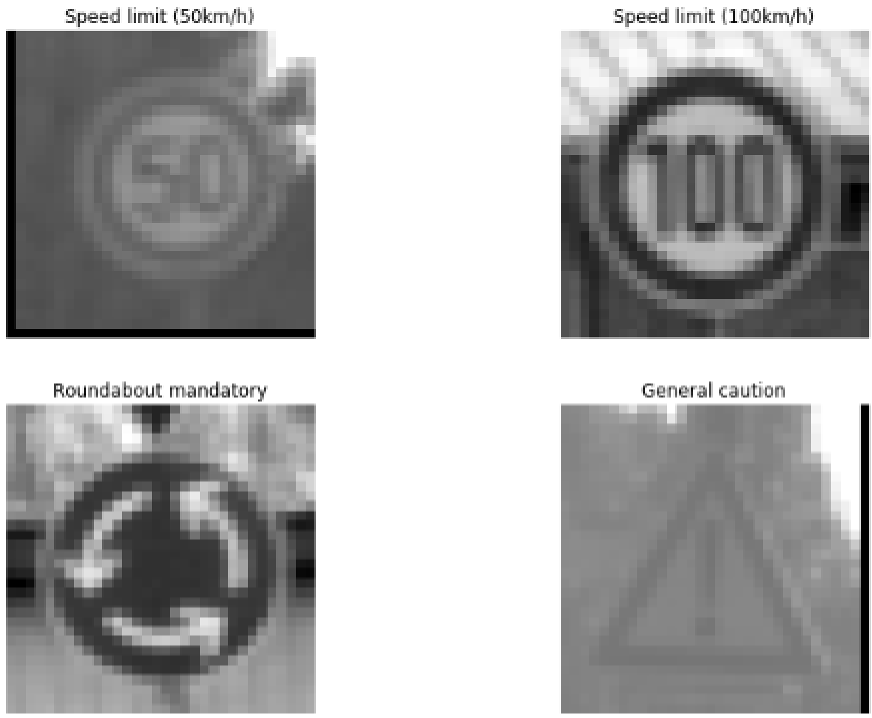




  - ➢ Apply transformations like affine & perspective transformation for retaining lines/planes and convert image from 3D space to 2D space



- Normalize the dataset using log function



- Model Architecture

  Modelled a five layer architecture which consists of two convolutional layers with max pool, two ReLU activation function layers and finally added bias.

- Model Training

  Prepared Logits with the above LeNet architecture.

  With the help of Softmax cross entropy & AdamOptimizer functions, I was training the dataset by fine tuning the hyper parameters like Epochs, Batch_Size & Learning_Rate.

  Finally sticked with the below values for hyper parameters,

  Epochs – 20 (tried values like 10, 15, 18, 20, 24)

  Batch_Size – 200 (tried values like 150, 180, 200, 210, 250)

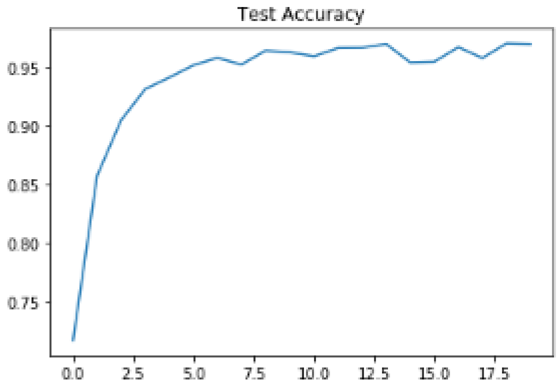
  Learning_Rate – 0.001 (tried values like 0.0003, 0.0008, 0.0009, 0.01)

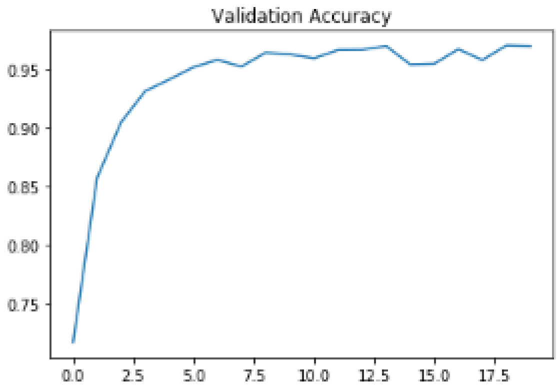
  Dropout – 1.0

  I was able to observe deviations between test & validation accuracy with different parameter values,

- Solution Approach

  I was using the similar LeNet architecture which was explained during the classroom. I have added one sub convolution layer in Layer 2 convolution to improve the accuracy of the LeNet pipeline. Hyper parameters are tuned based on the accuracy resulted below.






```
INFO:tensorflow:Restoring parameters from ./lenet_arch
Train Dataset Accuracy = 0.999
Validation Dataset Accuracy = 0.970
Test Dataset Accuracy = 0.943
```

Test a Model on New Images:

- Acquiring New Images

  Downloaded few images of traffic signs from internet and loaded with corresponding labels for evaluation.

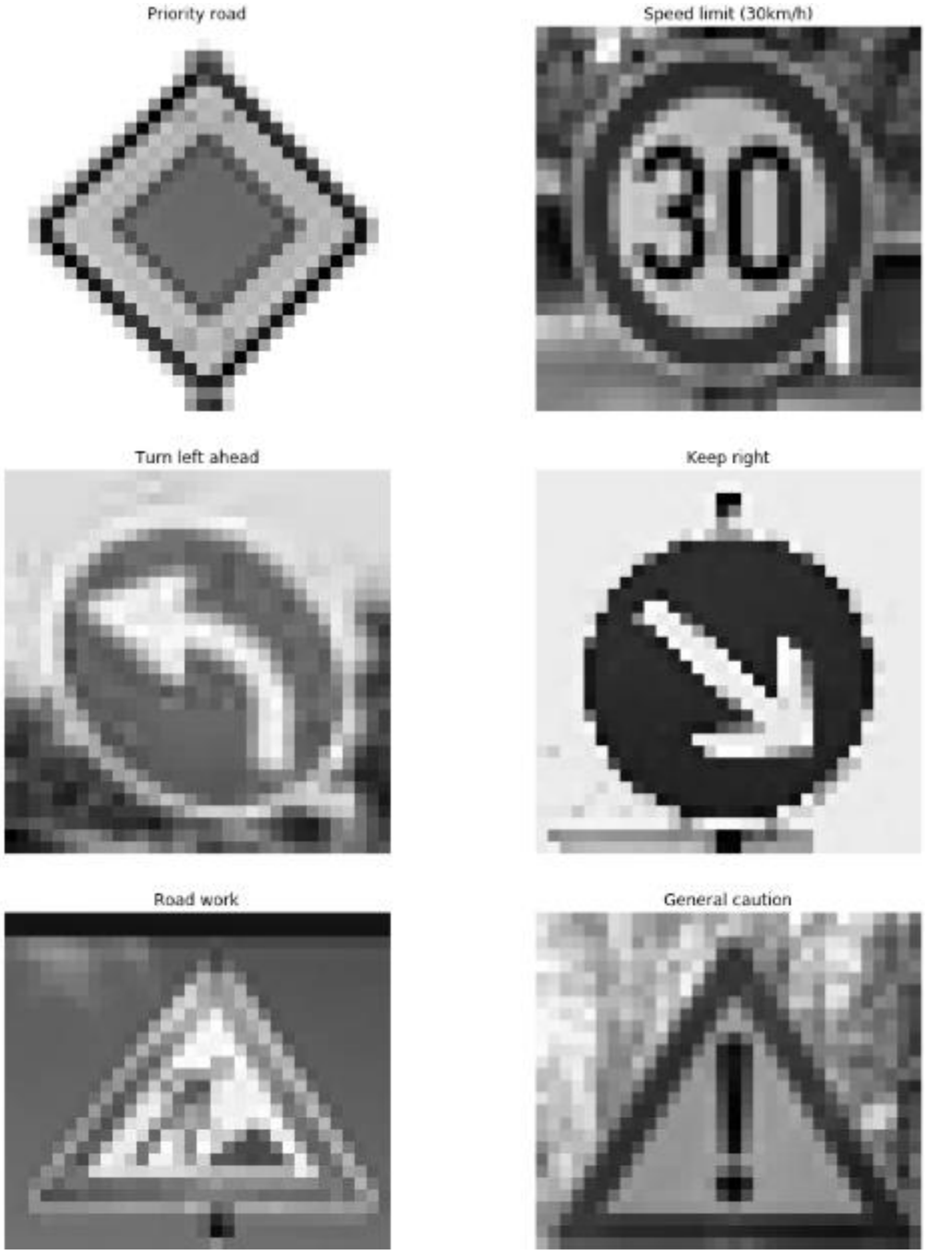
Characteristics of the images (3.png & 5.png) that would make the model difficult to classify are,

- Image capture angle & rotation
- Lighting conditions

Pre-processing step would have prepared the test images for better prediction that resulted in 100% accuracy.

- Performance on New Images

With the arrived LeNet Architecture, I've evaluated the test images and resulted with appropriate prediction as below.



```
INFO:tensorflow:Restoring parameters from ./lenet_arch
Image 1
Image Accuracy = 1.000

INFO:tensorflow:Restoring parameters from ./lenet_arch
Image 2
Image Accuracy = 1.000

INFO:tensorflow:Restoring parameters from ./lenet_arch
Image 3
Image Accuracy = 1.000

INFO:tensorflow:Restoring parameters from ./lenet_arch
Image 4
Image Accuracy = 1.000

INFO:tensorflow:Restoring parameters from ./lenet_arch
Image 5
Image Accuracy = 1.000

INFO:tensorflow:Restoring parameters from ./lenet_arch
Image 6
Image Accuracy = 1.000
```

- Model Certainty - Softmax Probabilities
  The model was able to correctly guess 5 of the 5 traffic signals, which resulted in 100% accuracy.

```
INFO:tensorflow:Restoring parameters from ./lenet_arch
```