Traffic Sign Recognition

This project involves building a convolution neural network architecture using TensorFlow to classify traffic sign images.

The goals of this project are as following:

- Load the 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

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Data Set Summary and Exploration:

1. **Summary of data set:**

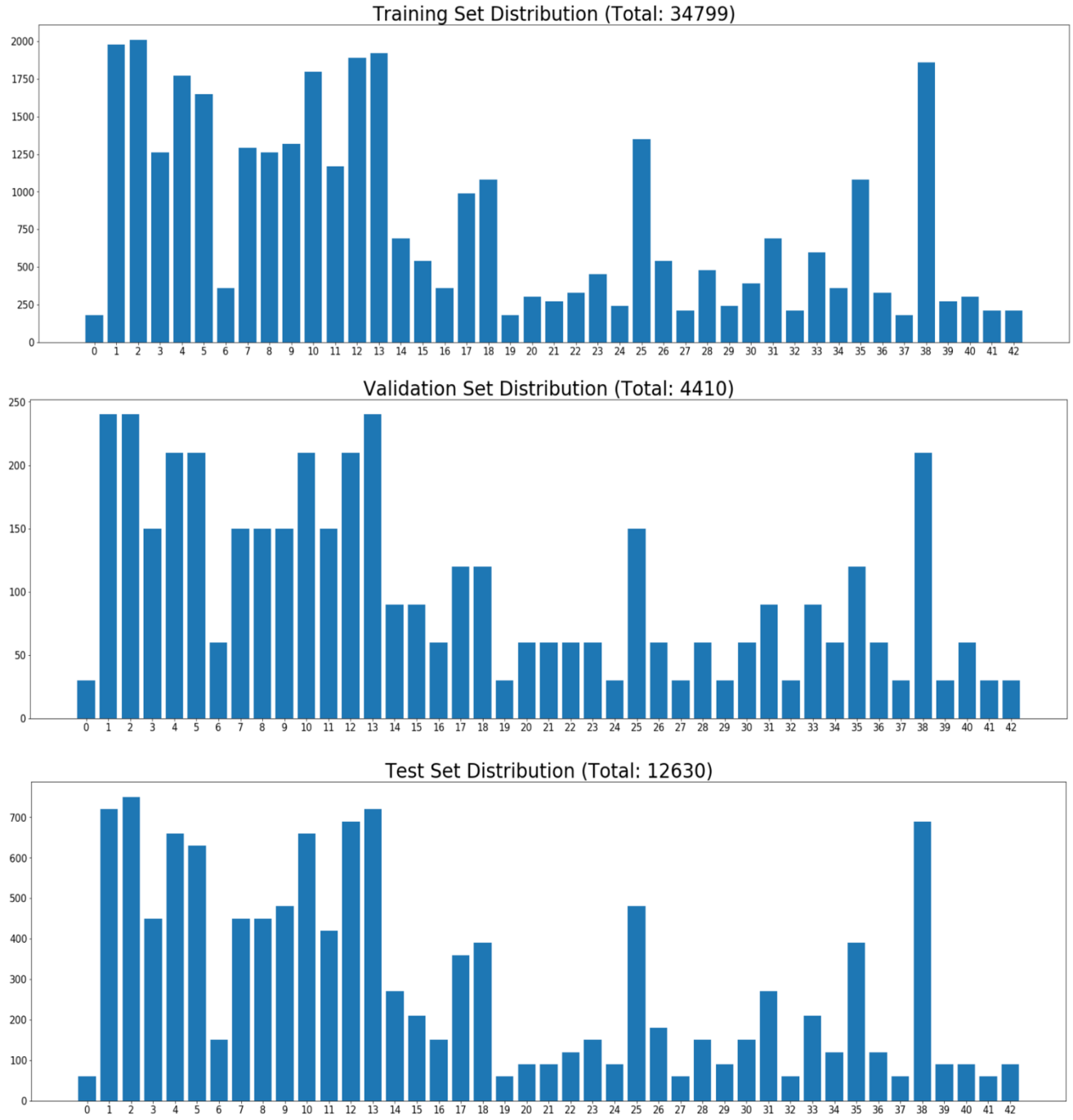
   I used Python and Numpy operations to compute the summary statistics of the traffic sign data.

   - Number of training examples: 34799
   - Number of testing examples: 12630
   - Image data shape: 32 x 32 x 3
   - Number of classes: 43

2. **Explanatory Visualization:**

   Firstly, I visualized few images belonging to each class in training set. Some of the images were very hard to classify manually.

Secondly, I noticed that some traffic signs are coming more frequently than others, so I decided to plot a histogram showing number of images in each class for training, validation and test data set. From the plots we can see that distribution of images across classes is not uniform, but the distribution is similar in each data set.



Design and Test a Model Architecture:

1. **Preprocessing:**

In this step, I scaled all the image pixels from 0-255 scale to 0-1 scale. I did not do more preprocessing as I wanted to test my architecture on the base images first. Also, I did not convert the images into gray scale as I wanted to use all of the available information.

2. Model Architecture:

Firstly, I chose the Lenet-5 architecture for my model. After training the model, I found that the model is not big enough to efficiently fit the images as I used color images. So, I decided to increase the size of the layers until I see signs of over-fitting. The final model I used for classification is as follows:

Layer	Description
Input Layer	32 x 32 x 3 RGB Image
Convolution 5x5	1x1 stride, same padding, output: 28 x 28 x 20
RELU	
Max Pooling	2x2 stride, 2x2 size, output: 14 x 14 x 20
Convolution 5x5	1x1 stride, same padding, output: 10 x 10 x 60
RELU	
Max Pooling	2x2 stride, 2x2 size, output: 5 x 5 x 60
Flatten	output: 1 x 1500
Fully Connected	output: 1 x 400
RELU	
Fully Connected	output: 1 x 129
RELU	
Output Layer	output: 1 x 43

3. Model Training:

To train the model, I used Adam Optimizer to minimize the cross entropy with logits. Hyperparameters used for the final training are as follows:

- Batch Size: 128
- Epochs: 60
- Learning Rate: 0.0015

4. Solution Approach:

Firstly, I chose the LENET-5 architecture to train the model and tuned the hyperparameters to get best results. After seeing the results, I found that it was not able to give good accuracy even on the training set, therefore, I inferred that the model is not big enough for the data. Due to this limitation, I decided to increase the depth of convolution layers. I iteratively increased the size of the model until I noticed that the model is giving 100% accuracy on training set but low accuracy on test set, i.e. model is overfitting the data. For tuning hyperparameters after finalizing the architecture, my approach was to keep the learning rate low with large number of epochs, so that the accuracy would increase slowly but to a higher value, as discussed in the lesson.

The results I got with the final model are as follows:

Data Set	Accuracy
Training Set	100 %
Validation Set	97.17 %
Test Set	95.73 %

Test the Model on New Images:

1. **Acquiring New Images:**

   Firstly, I acquired images of German traffic signs from web, after this, I cropped the images and then resized them to a size of 32 x 32. After this, I created a numpy set of these images similar to the format of test dataset.

2. **Performance New Images:**

   The model performed very well on these images and classified all the images perfectly. Therefore, the accuracy is 100%.

3. **Model Certainty – SoftMax Probabilities:**

   I used tf.nn.softmax() function on the output of the model to calculate the SoftMax probabilities. Surprisingly, model was very certain for each of the image. I got close to 1.0 probability for correct class in each case. After this, when I observed other classes that had higher probabilities for the image under consideration, I found that images pertaining to those classes also had similar features such as blue background, triangular border or circular border.