Traffic Sign Classifier:

Writeup:

This project "Traffic Sign Classifier" uses Convolutional Deep Neural Networks to classify Traffic Signs. Below is the detailed explanation of the steps followed and the Deep Neural Network used.

The Jupyter Notebook "Traffic_Sign_Classifier.ipynb" has the implementation of the project along with the obtained results. Below are the steps performed in the implementation:

- Load Dataset
- Dataset Summary and Exploration
- Design and Test Model Architecture
- Test Model on New Images

Load Dataset:

The Training, Validation and Test datasets are provided in the "/data/" folder. The datasets are loaded using the "pickle" library.

- Training dataset is loaded from the file "data/train.p"
- Validation dataset is loaded from the file "data/valid.p"
- Test dataset is loaded from the file "data/test.p"

Dataset Summary and Exploration:

Dataset Summary:

Below is the basic summary of the datasets loaded in the above step.

- Number of Training Examples: 34799
- Number of Validation Examples: 4410
- Number of Testing Examples: 12630
- Shape of each Image in the Datasets: (32, 32, 3)
- Number of Classes in the data (used numpy library to calculate): 43

Exploratory Visualization:

The implementation in the above-mentioned notebook also plots an image for each class by randomly picking one image of the class.

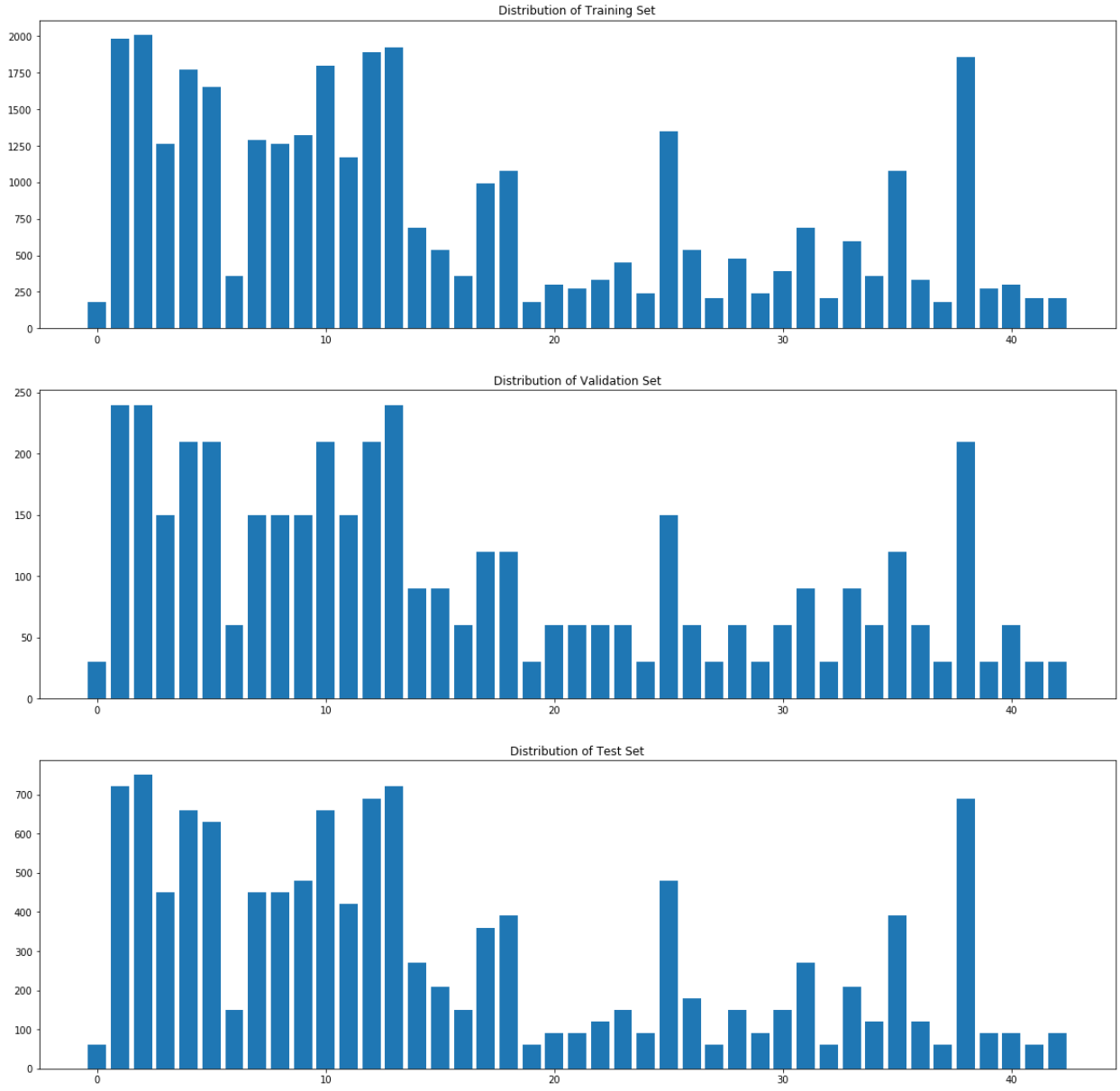
Each image has its class mentioned over it.

Images are displayed as shown below.

Also included in the implementation are the plots that show the distribution of the images with respect to their classes. Below are the plots that were charted using library “matplotlib”.

It can be clearly seen from the below plots that the data is not uniformly distributed, that is, there are a greater number of examples for some classes and very few examples for others.

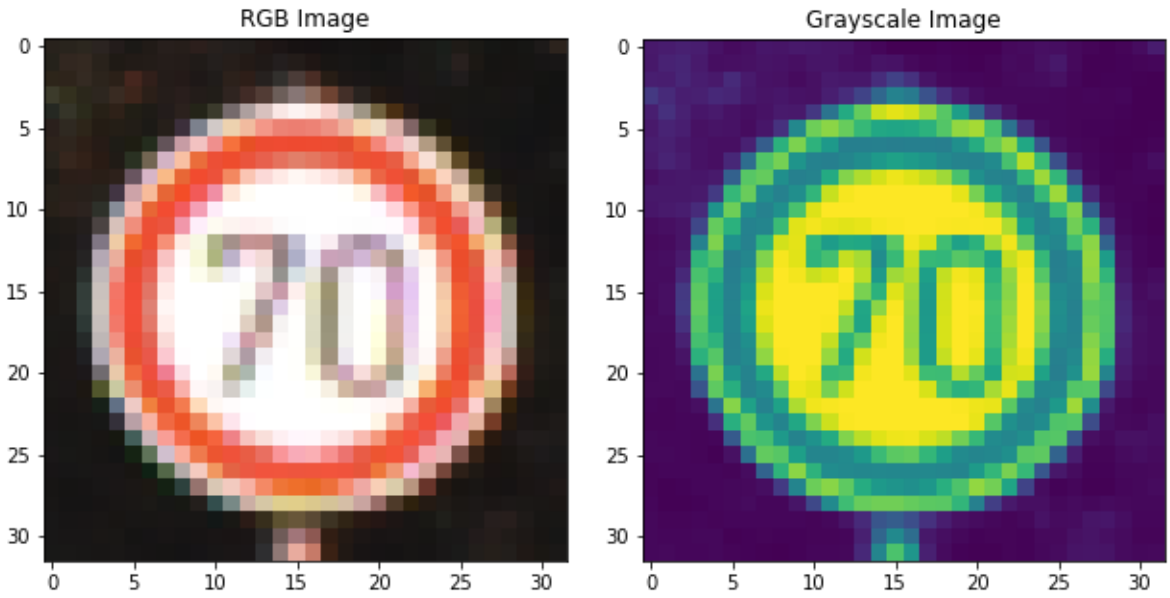
It can also be observed from the plots that all the three datasets (namely Training, Validation and Testing) have approximately similar distribution.



Design and Test Model Architecture:

Preprocessing:

First RGB images in the training, validation and test datasets are converted to Grayscale using the "numpy.dot()" function available with the numpy library. The function performs a vector dot multiplication. The below image show RGB image and its gray scale image.



The images converted to Grayscale are then normalized. On each image an operation "(image – 128.)/128." is performed. Below are the men values of all the images in respective datasets before and after normalization.

- Mean of Training Dataset before Normalization: 81.9164679387
- Mean of Validation Dataset before Normalization: 82.7522688698
- Mean of Testing Dataset before Normalization: 81.2767774761

- Mean of Training Dataset after Normalization: -0.360027594229
- Mean of Validation Dataset after Normalization: -0.353497899455
- Mean of Testing Dataset after Normalization: -0.365025175968

Model Architecture:

A modified version of the LeNet architecture mentioned in the paper by Pierre Sermanet and Yann LeCun is implemented in the project. The paper has a Convolutional Neural Network with three "Convolutional" layers and one fully connected layer. But my implementation has Six Layers in total.

There are Six Layers in my implementation of the Neural Network. First three layers are convolutional layers and the other three are fully connected layers. Each Convolutional Layer internally has two convolutional layers followed by a RELU Activation and Max Pooling Layers.

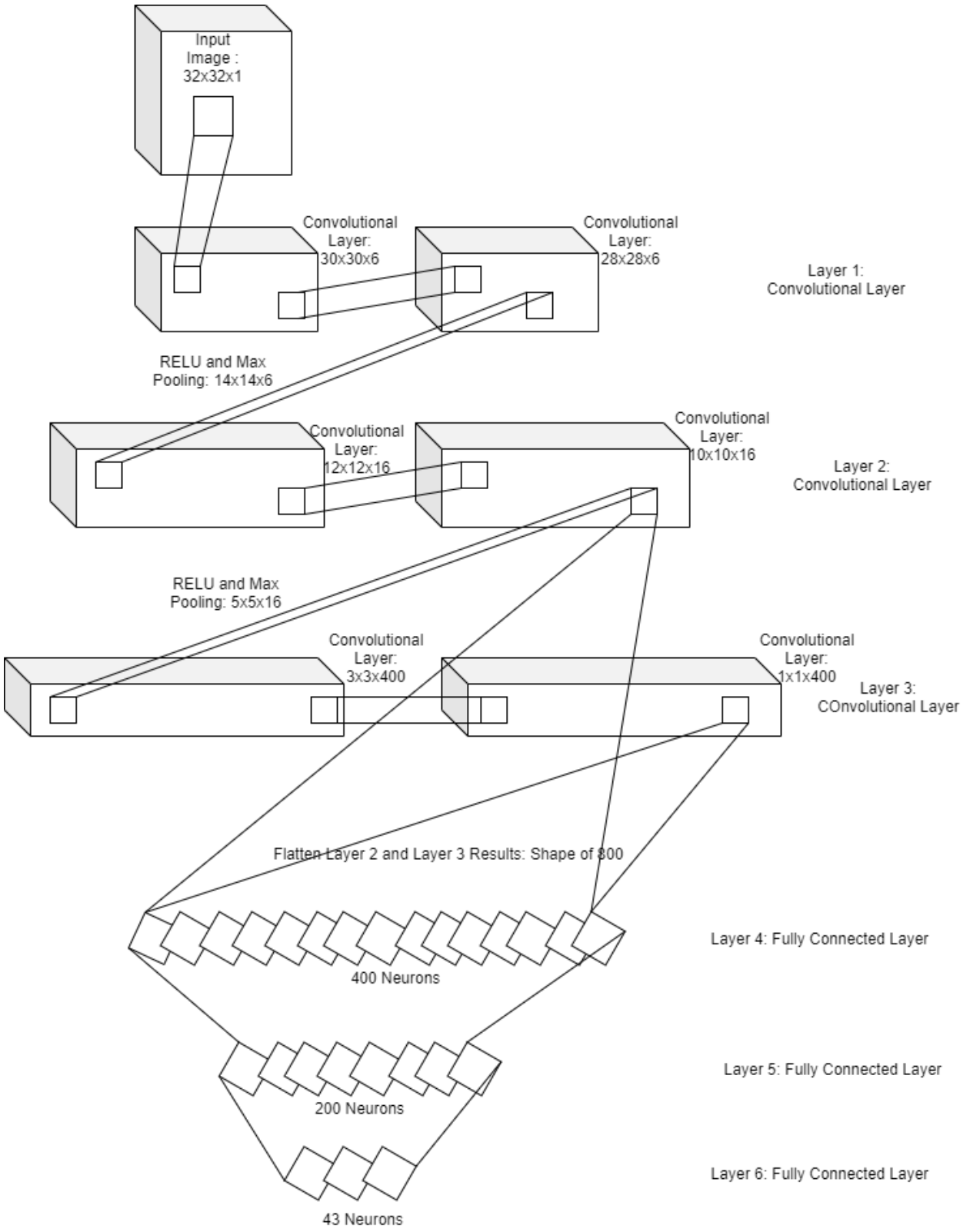
NOTE: The two consecutive convolutional layers instead of one are helping in capturing as many features as possible.

NOTE: In all the below Layers a Mean of "0" variance of "0.1" is used.

- **Layer 1:** has two Convolutional Layers followed by an RELU Activation and Max Pooling Layers

1. Convolutional Layers with input shape “32x32x1” and output shape “30x30x6”. The layer has a filter shape of “(3, 3, 1, 6)” and a stride of 1x1 and uses VALID Padding.
2. Convolutional Layers with input shape “30x30x6” and output shape “28x28x6”. The layer has a filter shape of “(3, 3, 6, 6)” and a stride of 1x1 and uses VALID Padding.
3. RELU Activation Layer.
4. Max Pooling layer with input shape “28x28x6” and output shape “14x14x6”. Uses a stride of 2x2 with Valid Padding.

- **Layer 2:** has two Convolutional Layers followed by an RELU Activation and Max Pooling Layers
    1. Convolutional Layers with input shape “14x14x6” and output shape “12x12x16”. The layer has a filter shape of “(3, 3, 6, 16)” and a stride of 1x1 and uses VALID Padding.
    2. Convolutional Layers with input shape “12x12x16” and output shape “10x10x16”. The layer has a filter shape of “(3, 3, 16, 16)” and a stride of 1x1 and uses VALID Padding.
    3. RELU Activation Layer.
    4. Max Pooling layer with input shape “10x10x16” and output shape “5x5x16”. Uses a stride of 2x2 with Valid Padding.
- **Layer 3:** has two Convolutional Layers followed by an RELU Activation Layer
    1. Convolutional Layers with input shape “5x5x16” and output shape “3x3x400”. The layer has a filter shape of “(3, 3, 16, 400)” and a stride of 1x1 and uses VALID Padding.
    2. Convolutional Layers with input shape “3x3x400” and output shape “1x1x400”. The layer has a filter shape of “(3, 3, 400, 400)” and a stride of 1x1 and uses VALID Padding.
    3. RELU Activation Layer.
- **Flatten Layer 2 and Layer 3 outputs and Concatenate:** the outputs from layer 2 and layer 3 are flattened to have a shape of (400) each, and they are concatenated to get a shape of (800)
- **Dropout:** TensorFlow dropout function is applied on the output of the above layer. While training the probability to keep values is set as 0.5 and while testing and actual running the model it is set to 1.0.
- **Layer 4:** fully connected layer with input feature shape of (800) and output of shape (400)
- **Dropout:** TensorFlow dropout function is applied on the output of the above layer. While training the probability to keep values is set as 0.5 and while testing and actual running the model it is set to 1.0.
- **Layer 5:** fully connected layer with input feature shape of (400) and output of shape (200)
- **Dropout:** TensorFlow dropout function is applied on the output of the above layer. While training the probability to keep values is set as 0.5 and while testing and actual running the model it is set to 1.0.
- **Layer 6:** fully connected layer with input feature shape of (200) and output of shape (43)



Model Training:

The above mentioned Convolutional Neural Network is trained with an Epoch Size of 50, Batch Size of 128. Used TensorFlow's AdamOptimizer as the optimization technique.

Used a learning rate of "0.0001" while training the model.

Solution Approach:

The trained model obtained the below accuracy on training, validation and test data sets.

- Accuracy with training dataset: 1.0
- Accuracy with validation dataset: 0.97
- Accuracy with test dataset: 0.957

Test Model on New Images:

Acquiring New Images:

I have downloaded five German Traffic Sign Images from google images. All the images are placed in the folder "CarND-Traffic-Sign-Classifier-Project/New_Images/". Each Image is named after it's the class it belongs. Below are the images with their names over them.

./New_Images/class_13.jpg

./New_Images/class_18.jpg

./New_Images/class_14.jpg

./New_Images/class_35.jpg

./New_Images/class_41.jpg

Image Name	Description of Image Class
New_Images/class_13.jpg	Yield
New_Images/class_18.jpg	General caution
New_Images/class_14.jpg	Stop
New_Images/class_35.jpg	Ahead only
New_Images/class_41.jpg	End of no passing

All the above images are resized to a shape of (32x32x1) using cv2.resize() function. Also, it is to be noted that I choose the images with classes of high, medium and low number of training examples, as shown in the distribution plots above.

Also, the model should be able to classify all the above images.

Performance on New Images:

The model correctly classified all the images. The Testing dataset Accuracy of the model is 0.941 and on the New Images it is 1.00.

Model Certainty – Softmax Probabilities:

Below is the list of top 5 probabilities for each class of Image displayed above.

- **Image with class 13 ("New_Images/class_13.jpg"):**
    a. Has top probabilities [1. 0. 0. 0. 0.] at indices [13 0 1 2 3]
- **Image with class 18 ("New_Images/class_18.jpg"):**
    a. Has top probabilities [1.00000000e+00   4.47113049e-25   5.80988738e-27
    1.04988869e-28   8.46202531e-32] at indices [18 26 27 38 1]
- **Image with class 14 ("New_Images/class_14.jpg"):**
    a. Has top probabilities [1.00000000e+00   1.09426905e-10   4.10612350e-15
    3.92176890e-15   9.89801740e-17] at indices [14 34 17 35 33]
- **Image with class 35 ("New_Images/class_35.jpg"):**
    a. Has top probabilities [1.00000000e+00   2.45586734e-14   1.87837201e-15
    8.45162485e-20   4.76281097e-20] at indices [35 29 36 25 3]
- **Image with class 41 ("New_Images/class_41.jpg"):**
    a. Has top probabilities [9.99998212e-01   1.81371604e-06   6.89297774e-09
    5.31766124e-12   2.60213536e-13] at indices [41 42 6 32 23]

It can be seen from the above list of top five probabilities for image of each class that for the classes (13, 18, 14, 35) the model got 1.0 as the top probability, which means the model is highly certain while classifying the images.

But, in case of the class 41, the top probability is not 1.0, but is slightly less. This can be attributed to the fewer examples provided for the class in the training dataset.

Possible Improvements to the Implementation:

As it can be seen from the class distribution plots above, some classes have many examples, and some have less examples. So, we can generate new data using augmentation techniques like warping, rotating, changing brightness etc. Doing so would improve the model accuracy by a big margin as the model gets enough examples to train on.