Traffic Sign Classifier SDCND-Term1 Project2

Data Summary and Exploration

1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually

I used numpy library to compute the summary of dataset provided and the results are

Number of training examples = 34799

Number of validation examples= 4410

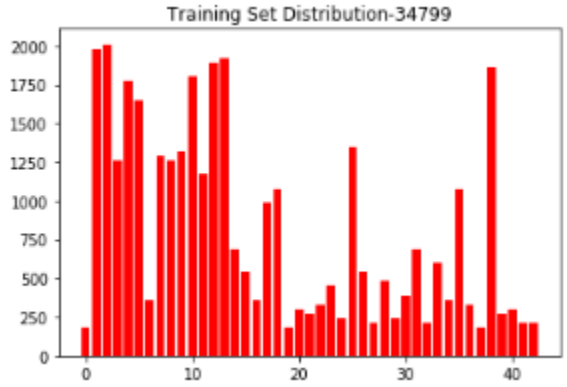
Number of testing examples = 12630

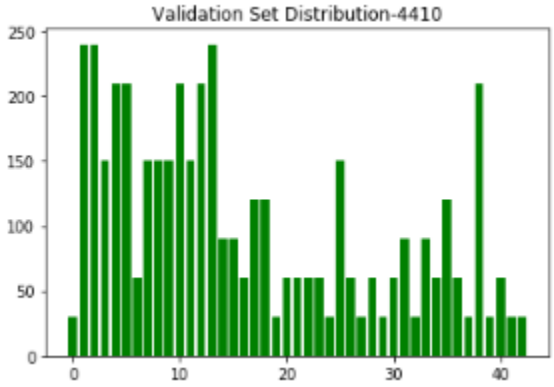
Image data shape = (32, 32, 3)

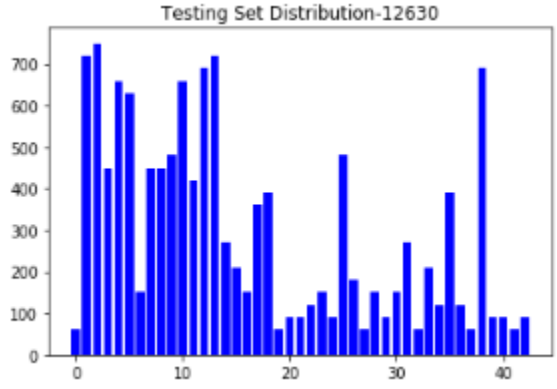
Number of classes = 43

2. Exploratory Visusalization of the dataset

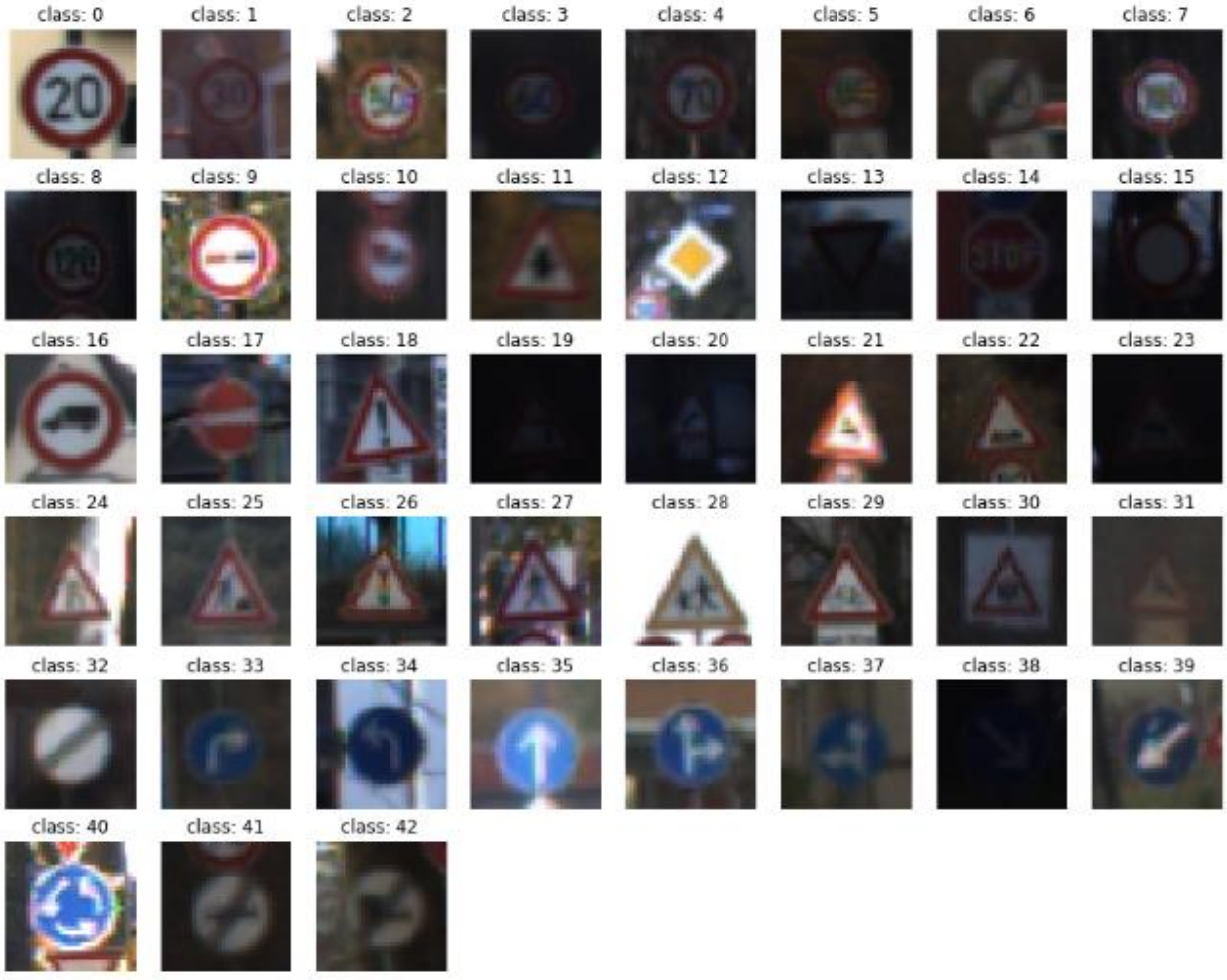
I used numpy and matplotlib to explore the dataset distribution for each classes in the provided set







Also I visualized the single image from each class



3.Design and Test a Model Architecture

Preprocessing:

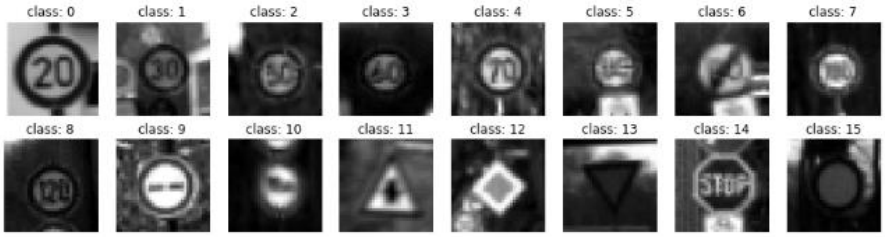
I used Open CV libraries for preprocessing the images to feed to CNN

Converting the images to gray scale and normalizing the images to keep the mean around zero (-1 to 1).

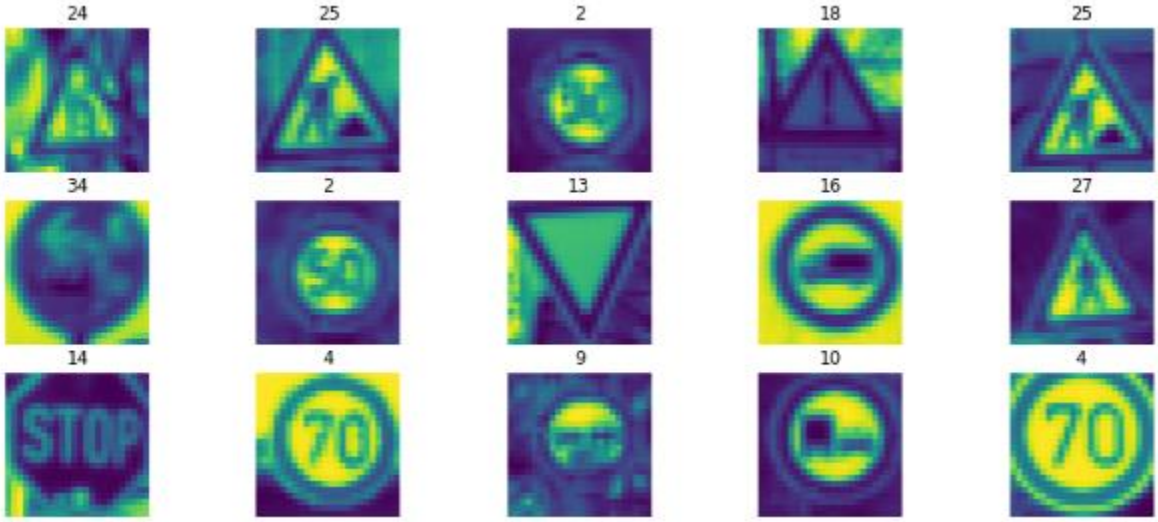
Then select the region of interest in the image that is where most of feature extraction is easier and is reshaped around 24 x 24 x 1

The gaussian blur and weighing is applied to make a blended image over

Sample normalized images after grayscale conversion



Random images after complete preprocessing



Model Architecture:

Statistical Parameters: mean=0; Standard deviation=0.1; Drop out probability=0.9

Convolution Layer 1	Input: 24x24x1; Output: 20x20x20; Kernal=[5,5,1,20];Depth20;Stride=[1,1,1,1];padding=VALID
Relu	
Dropout	
Max Pooling	Input: 20x20x20; Output: 10x10x20; Kernal=2; Stride=[1,2,2,1];padding=VALID
Convolution Layer 2	Input: 10x10x20; Output: 6x6x40; Kernal=[5,5,20,40]; Depth40;Stride=[1,1,1,1];padding=VALID
Relu	
Dropout	
Max Pooling	Input: 6x6x40; Output: 3x3x40; Kernal=2; Stride=[1,2,2,1];padding=VALID
Flatten	Input: 3x3x40; Output: 360
Fully Connected Layer 1	Input=360, Output=260
Relu	
Dropout	
Fully Connected Layer 2	Input=260; Output=160
Relu	
Fully Connected Layer 3	Input=160; Output=100
Relu	
Dropout	
Fully Connected Layer 4	Input=100; Output=80
Relu	
Fully Connected Layer 5	Input=80; Output=50
Relu	
Dropout	
Output Layer	Input=50; Output=43

How I arrived at this model?

I thought of reducing the region of the image from the dataset provided to only the region which covers most of the features of the signal and eliminate the rest of the image corner areas. So I finally arrived at an image of 24x24x1. I cannot directly apply LeNET architecture. I experimentally restricted with two convolution steps and finish the rest of the output with fully connected layers as I tried to see how the CPU only computation power for neural networks.My first attempt was to check dropout at all layers of the network to see what happens. And I achieved a Validation set accuracy of 94% and Testing set accuracy of 92%. I initially used a drop out rate of 0.0005 and my accuracy was not going much better than validation 94% with about 40 EPOCHS and std.dev=0.05. Then after tuning the learning rate to 0.0009 and std. deviation to 0.1 and drop out probability to 0.8 I gained like 95.1% validation accuracy. Finally I reduced the drop out only at intermittent stages of fully connected layer to avoid over fitting, and learning rate to 0.001 to optimize the gradient descent and std dev to 0.1 and EPOCHS to 35 and a drop out rate of 0.9. I achieved the best validation accuracy of 96.3% and testing set accuracy of 93.6%.

5.Model Training:

Batch Size: 100

EPOCHS: 35

Learning rate: 0.001

Results:

Testing Accuracy: 96.3%

Validation Accuracy:93.6%

Testing on Random Images:100%

6.Load and Test on New Images:

Loading random Images from Internet:

The respective classes: y_new=[39,40,28,17,13]

7. **The Classes Identified and Softmax Probabilities in the order represented above**:

```
test_index: [39 24 31 25 29]
test_value: [  1.00000000e+00   1.00362114e-11   2.10899115e-12   7.41128025e-13
   2.29807831e-15]
test_index: [40  1  6  8 12]
test_value: [  9.99943852e-01   3.65515079e-05   9.20279399e-06   6.20318406e-06
   2.98920327e-06]
test_index: [28 29 20 36 35]
test_value: [  1.00000000e+00   6.87562351e-15   1.11546947e-16   9.51780896e-18
   8.31701310e-19]
test_index: [17 34  3  9 37]
test_value: [  1.00000000e+00   4.99246777e-09   3.18443327e-10   1.63493427e-10
   1.72397551e-13]
test_index: [13  8 12 35 38]
test_value: [  1.00000000e+00   6.13476278e-32   3.24589138e-32   6.66165460e-34
   7.24279272e-38]
```

Features Significant for easier detection:

All the images used are relatively easier for the algorithm to find, because they had only simple features like edge, shapes and can be easily identified from the gradient differences. Hence all the images I fed the network in internet were easily identified with almost 100% accuracy and if looked at the soft max probabilities it can be easily witnessed.

I tried feeding the random images from internet like warning for cows falling from above, people at work, such images with complex features and network achieved accuracy of only 50-80%. And it requires training with additional dataset with augmentation of existing images so that network can identify images taken at varying positions.

Conclusion:

My network worked fine with 93.6% test accuracy and 96.3% validation accuracy. But the network classified this set of images with 100% accuracy, when I tried testing for few other images the network accuracy was 100%. So the network training is not robust., I have to try augmentation of images to create additional data set samples and modify the network architecture and increasing the EPOCHS and utilize GPU computing in AWS to reach about 99% validation accuracy.