Traffic Sign Recognition

The goals / steps of this project are the 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

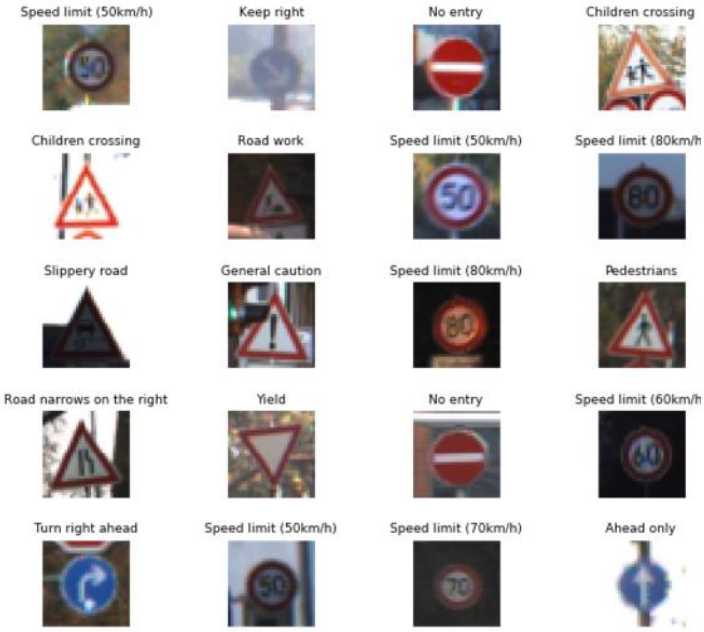
1. Data Set Summary and Exploration

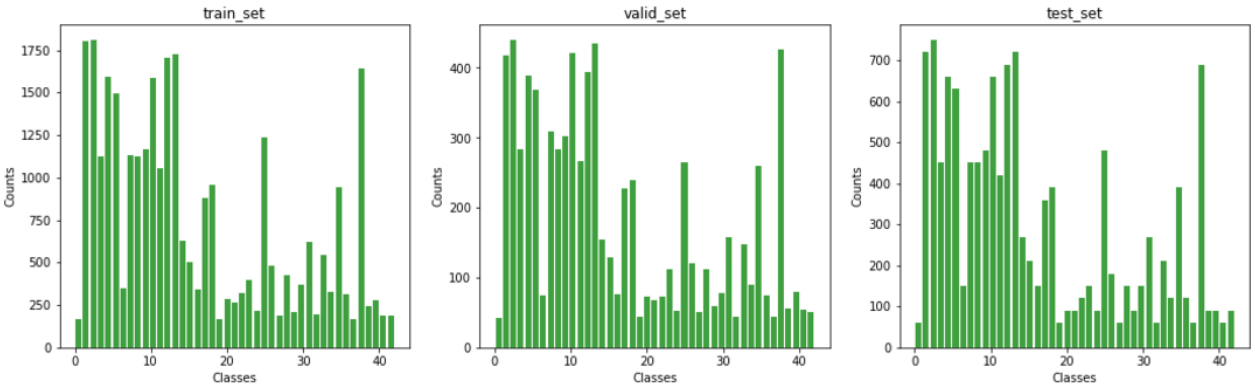
The loaded train and validation set from original dataset are concatenated, shuffled and then separated to train and validation with 80% as trainset and 20% as validation set. The loaded test set is kept as it was.

I used the numpy library to calculate summary statistics of the traffic signs data set:

- The size of training set is 31367
- The size of the validation set is 7842
- The size of test set is 12630
- The shape of a traffic sign image is (32, 32, 3)
- The number of unique classes/labels in the data set is 43.

Here is an example of traffic sign images form trainset and their labels.





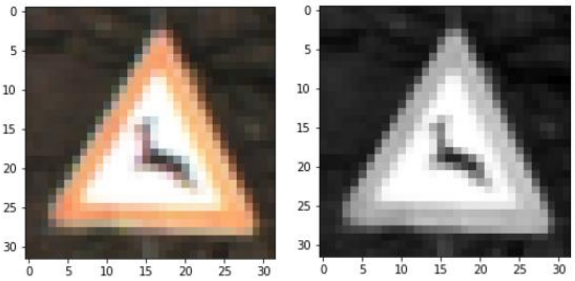
As it is obvious in the histograms above, the train, validation and test sets are not distributed equally in different classes. To make a better training I have added augmented images to the classes, which had sample below some threshold.

2. Design and Test a Model Architecture

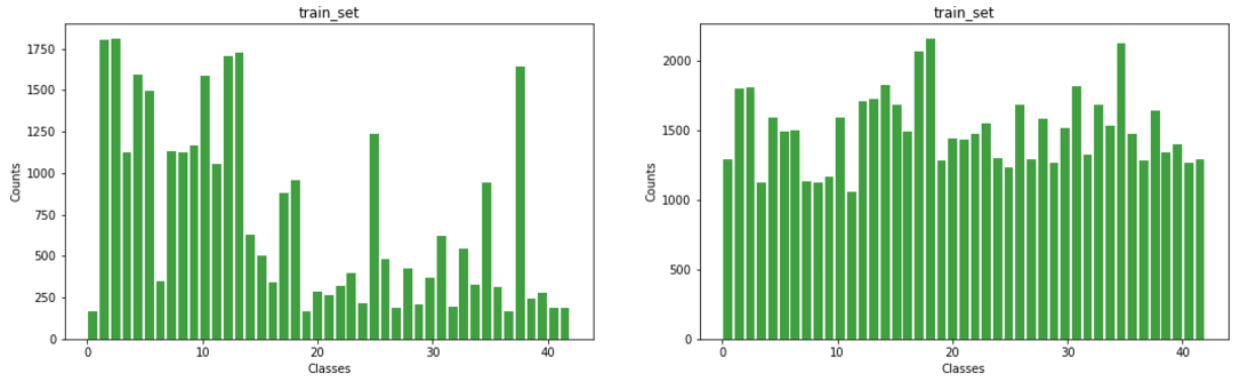
I have prepared 2 types of data, gray scale and normalized RGB. Normalization is done by:

$$(data - 128)/128$$

And gray scale achieved by dot product of RGB image to the [0.2989, 0.587, 0.114] vector resulting in a 32 × 32 gray scale image. I found out for the various architectures of the deep network the grayscale images resulted in higher accuracy of validation set, so I decided to use the gray scale images for the final model. Here is an example of a traffic sign image before and after grayscaling:



As described in the previous section the histogram of the data sets showed that there are unequal distribution of images in various classes. To solve this problem I have added augmented images to the classes with lower number of samples.



Histograms in the image above show the grayscale train set before and after up-sampling. The original train and validation sets and also the normalized ones up-sampled too, but experimentation on these various dataset showed that the up-sampled grayscale reached higher accuracy. The Keras library used for data augmentation. Based on the experiments I have done on various types of augmentation I have found these parameters and values more are effective: rotation_range=15, height_shift_range=0.05, shear_range=0.1, zoom_range=0.1, width_shift_range=0.05.

First I used the model like LeNet but found out increasing the depth of the convolution and size of fully connected layers can lead to higher accuracy. While training with this architecture the difference between the train and validation sets accuracy was the indication of over-fitting, so I added the dropout layers between the fully connected layers. The value of the dropout layer is a hyper-parameter, which I chose values of 0.8 and 0.7 for the first and second dropout layers, respectively. The table blow shows the final selected architecture for the model.

Layer	**Description**
Input	32, 32, 1 gray image
Convolution 5 × 5	1 × 1 stride, valid padding, output 28 × 28 × 10
RELU	
Max pooling	2 × 2 stride, valid padding, output 14 × 14 × 10
Convolution 5 × 5	1 × 1 stride, valid padding, output 10 × 10 × 20
RELU	
Max pooling	2 × 2 stride, valid padding, output 5 × 5 × 10
Fully connected	Input 500, output 200
RELU	
Dropout	Keep prob 0.8
Fully connected	Input 200, output 100
RELU	
Dropout	Keep prob 0.7
Fully connected	Input 100, output 43

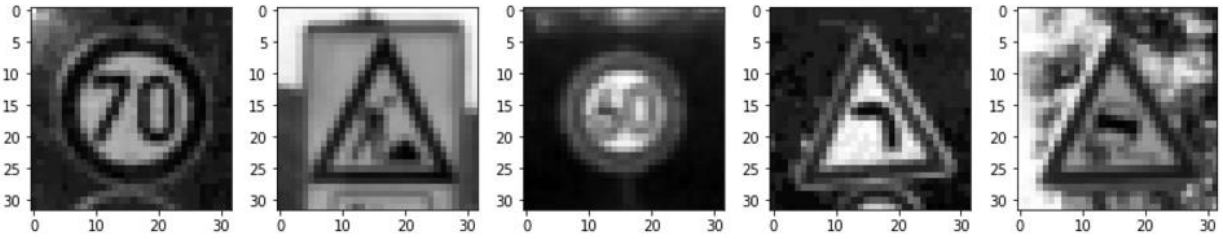
Other hyper-parameters are number of epochs, batch size and learning rate. Batch sizes of 64, 128, 320 are tried, and 128 showed more accurate result in reasonable time. I have started with learning rate of 0.001 and decreased it gradually whenever the accuracy of validation set was flatten out. To do that I first trained the model for 20 epochs with learning rate of 0.001, then 10 epochs for rate of 0.0001 and 20 epochs for rate of 0.00001. This training resulted in:

- Training set accuracy of 99.6%
- Validation set accuracy of 97.4%
- Test set accuracy of 93.8%

The problem with the LeNet that was selected at the beginning was that it could not reach high accuracy of validation set well above 93%, even with high number of epochs, therefore increased the depth of convolutional layers and size of fully connected layers were used that could improve the result. This improvement is the indication of that the model by this changes can learn more sophisticated features out of image and be more precise in the classification.

3. Test a Model on New Images

Here are five German traffic signs that I loaded from data set downloaded from the web:



The loaded images are RGB and have different seizes, which changes to grayscale 32 × 32. I have loaded various sets of images (set of 5 images), predicted the labels and in most cases the accuracy was 100% and rarely it was 80%. Achieving higher accuracy on the new images comparing to the test set, is indication of the fact that the model is generalized well on the train set and didn't have over- or under-fitting. Based on my exploration on the new images, the contrast of the images are better comparing to the train and test sets and also there are less blurry images, which can be the reason to get the better accuracy on the new images.

I also calculated the precision, recall and f1-score for the predicted labels of the test set and it shows for some classes like 27 (pedestrians) the precision, recall and f1-score is low, meaning the model having difficulty to correctly predict this class correctly.

Afterward the softmax probabilities of each prediction is calculated. The table below show the top 5 softmax probabilities for each image along with the sign type of each probability.

1st image		**2nd image**	
Probability	prediction	Probability	prediction
9.99e-1	Speed limit (70km/h)	1	Road work
3.3e-5	Speed limit (30km/h)	5.7e-30	Bicycles crossing
7.97e-6	Speed limit (20km/h)	0	Wild animals crossing
6.93e-6	Keep left	0	Beware of ice/snow
2.5e-6	Stop	0	Double curve

3th image		**4th image**	
Probability	prediction	Probability	prediction
9.99e-1	Speed limit (60km/h)	9.99e-1	Dangerous curve to the left

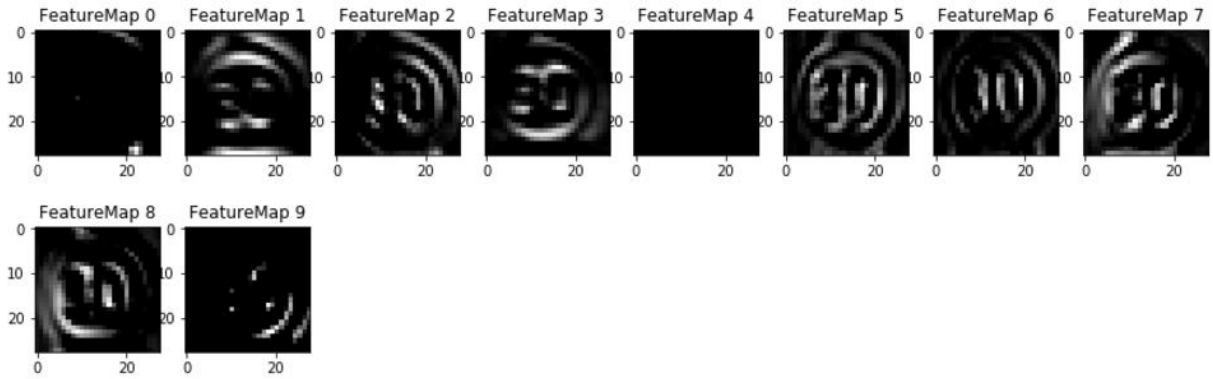
4.49e-6	Speed limit (80km/h)	3.43e-6	Dangerous curve to the right
2.74e-8	Speed limit (50km/h)	1.92e-6	Slippery road
2.83e-10	Children crossing	2.86e-7	Traffic signals
1.83e-10	Ahead only	1.89e-7	Road narrows on the right

5th image	
Probability	prediction
1	Slippery road
4.39e-30	Dangerous curve to the right
0	Dangerous curve to the left
0	Children crossing
0	Speed limit (20km/h)

These softmax probability results show that the model can predict the classes with high certainty.

4. Visualizing the Neural Network

I have visualized the output of first convolutional layer before maxpooling as shown below:



To access the conv1 output I have returned it from LeNet function that beside logits it give the conv1 output too. By feeding conv1 to the outputFeatureMap function can get the feature maps. Based on the visualized feature maps that lighten up, the conv1 layer mostly picking up the various types of edges.