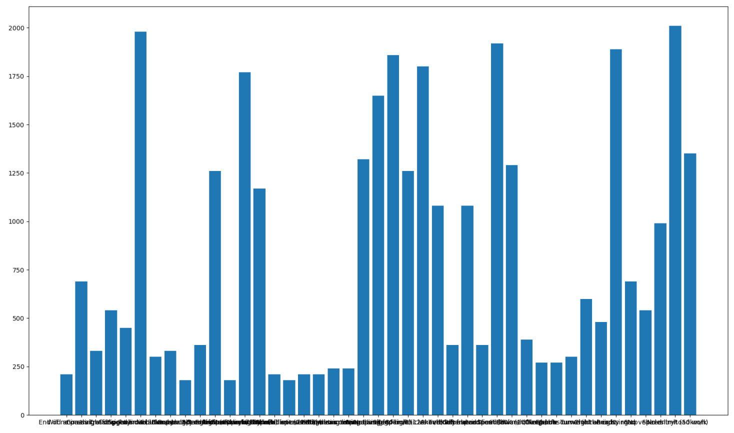
#Traffic Sign Recognition

## Data Set Summary & Exploration

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

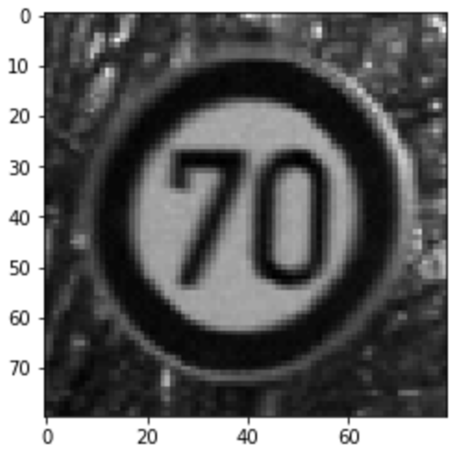
- The size of training set is 34799
- The size of the validation set is 4410
- The size of test set is 12630
- The shape of a traffic sign image is [32,32,3]
- The number of unique classes/labels in the data set is 43

Here is an exploratory visualization of the data set. It is a bar chart showing how many images for one traffic sign type.



## Design and Test a Model Architecture

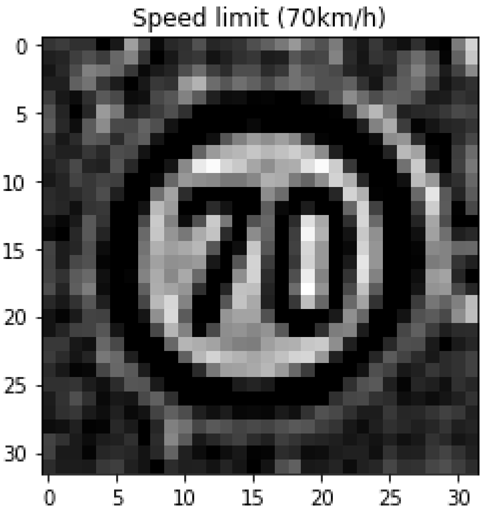
As a first step, I decided to convert the images to grayscale because even if the grayscale images loss some color features compared with color images but the important features for sign recognition like shape, texture remain besides grayscale images are easy for CNN to process and extract features. Here is an example of a traffic sign image before and after grayscaling.



But it is clear that there are a lot of noise on the image and the sign on image is not clear. So I decide to use median filter and gaussian filter to deal with the noise, median filter works well with pepper salt noise and guassian filter is good at bluring image. After going through the noise filter, I use another filter to sharpen the images which is shown below:

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

After going through the steps in the image preprocess pipeline, the output showns below:



We can see that the edge of the image is sharpened and the pixels in the boundery are strengthened. Besides I normaize the image at the last step to make it easy to train in CNN. My final model consisted of the following layers:

| Layer | Description |
|---|---|
| Input | 32x32x1 grayscale image |
| Convolution 5x5 | 1x1 stride, valid padding, outputs 28x28x16 |
| RELU | |
| Max pooling | 2x2 stride, valid padding, outputs 14x14x16 |
| Convolution 5x5 | 1x1 stride, calid padding, outputs 10x10x32 |
| RELU | |
| Max pooling | 2x2 stride, valid padding, outputs 5x5x32 |
| Fully connected | 800 neurons |
| Fully connected | 512 neurons |
| Fully connected | 128 neurons |
| Softmax | 43 neurons |

To train the model, I used adam optimizer which contains momentum method to avoid the training process to stay at the local optimum. Besides I use exponetial decay to the learning rate which decays the learnig rate dynamically during the training process in order to find a good path to converge. The batch size is set to be 128, epochs is set to be 30. My final model results were:

- validation set accuracy of 0.962
- test set accuracy of 0.952

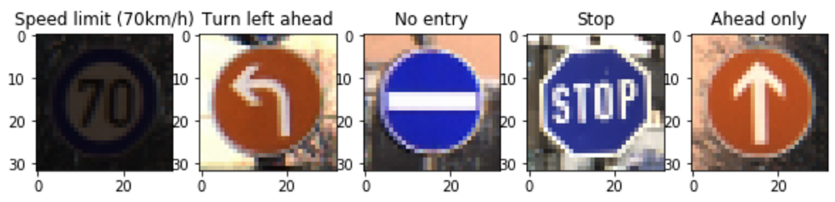
At the every beginning, I have chosen a model called multiscale CNN which is first published by Yann LeCun by the famous paper Traffic sign recognition with multi-scale Convolutional Networks (http://s3.amazonaws.com/academia.edu.documents/31151276/sermanet-ijcnn-11.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1496379972&Signature=bmRot0%2BWva%2BzkmUC content-disposition=inline%3B%20filename%3DTraffic_Sign_Recognition_with_Multi-Scal.pdf).

However due to parameter tuning, limited size on dataset and computation limitations, I only get an accuracy of 90% on test set. Actually if the data set is big enough and computation ability is powerful enough, combining with fine tuning, it is possible to make a more complex model based on multiscale CNN to achieve a better accuracy, here is a reference which reaches an incredible accuracy 98.8% based on multiscale CNN: German sign classification using deep learning neural networks (https://chatbotslife.com/german-sign-classification-using-deep-learning-neural-networks-98-8-solution-d05656bf51ad).

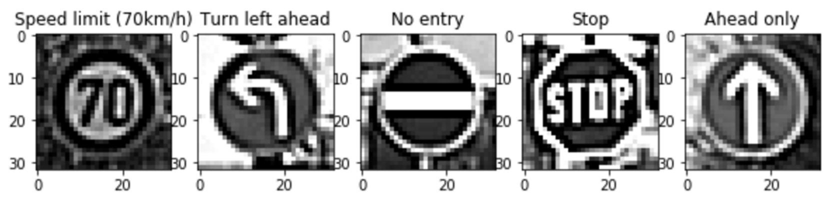
So at last, I decided to use a CNN model based on Lenet, but a problem of Lenet is learning rate is a constant, during the training process I find that the model performed better with a decay learning rate. So I added an exponential decay with initial learning rate 0.0005 and a decay parameter 0.9. Finally, the model achieved an accuracy of 95.2% on the test dataset.Another improvement is adding dropout layer to the fully connected layer. Because I found that in the previous training process, validation accuracy is nearly 97.5% but the test accuracy is only 94%, so I guess the model is overfitting that is why I add the dropout layer.

### Test a Model on New Images

Here are five German traffic signs that I found on the web:



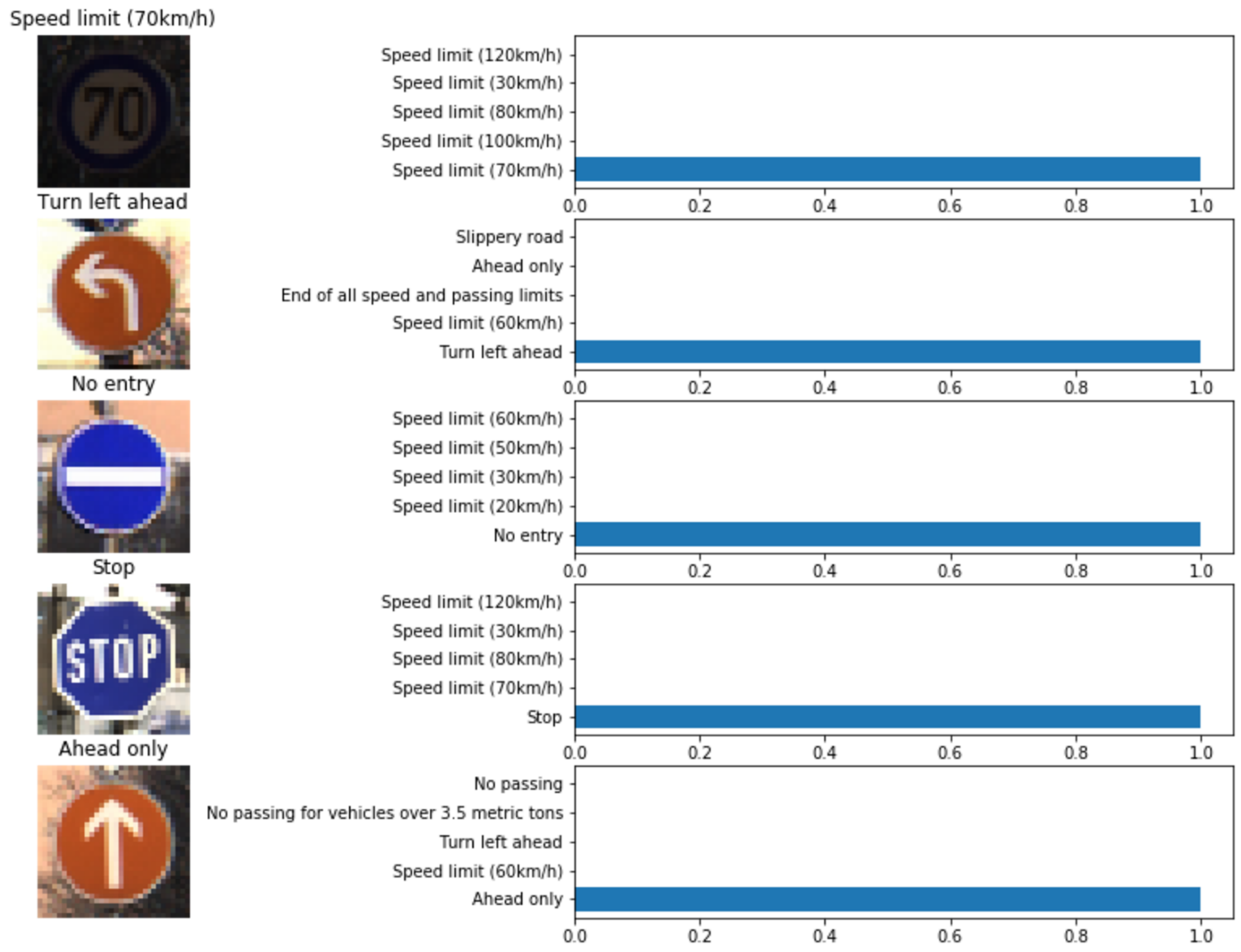
Here are the five traffic signs processed by the pipline:



The first image might be difficult to classify because it is dark and not clear enough, so in order to make it clear and clean we need to use some techniques to reduce the noise and sharpen bounderies. Here are the results of the prediction:

| Image | Prediction |
|---|---|
| Speed limit (70km/h) | Speed limit (70km/h) |
| Turn left ahead | Turn left ahead |
| No entry | No entry |
| Stop | Stop |
| Ahead only | Ahead only |

The model was able to correctly guess 5 of the 5 traffic signs, which gives an accuracy of 100%. This compares favorably to the accuracy on the test set of 95.2% is better but the test images are only 5. Here is the five top probabilities for each of the five images:



We can see that the model is pretty sure for each of the five and the probability of the top one is 100%. So we can see that the model is really successful at least on these five images.

```
In [ ]:
```