Trafic Sign Classifier

Said Zahrai
ABB Robotics

October 6, 2019

1 Introduction

These writeup describes my work on classification of trafic signs. The steps I have gone through in this project are the following:

1. Load the data set (see below for links to the project data set) *
2. Explore, summarize and visualize the data set
3. Use image processing to improve the quality of input data
4. Conduct data augmentation for those classes with limited number of images
5. Normalize the input data
6. Design, train and test a model architecture
7. Use the model to make predictions on new images
8. Analyze the softmax probabilities of the new images
9. Summarize the results in this document

These steps are presented in the following sections and important points are detailed .

2 Data loading and exploration

2.1 Dataset Summary

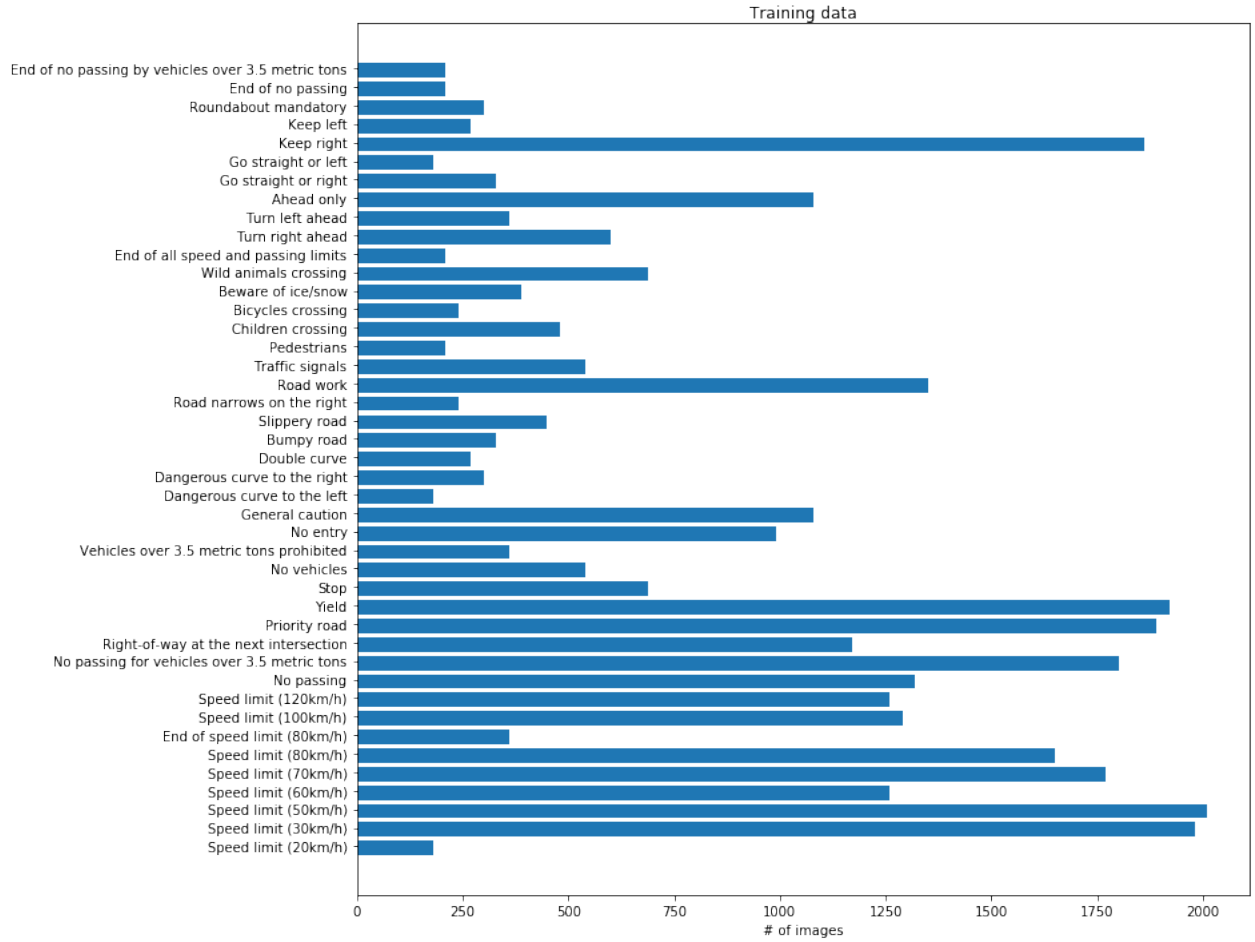
The data was loaded using `pickle` package and the text lables with `panda.read_csv`. Statistics were calculated simply by looking at shapes of the data. The results are presented in the table below.

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

2.2 Exploratory Visualization

Next, I looked at the statistics in the training data and visualized the numebr of images for each class of input data. For that, I used `numpy.histogram`. The result is shown in the blow figure.

Figure 1: Trainig data statistics.



As can be found in this figure, the number of images for each class can be as high as 2000 images and as low as about 200. This is not a suitable distribution of input data as it leads to higher weight in calculaiton of error for certain classes and low for others. In next section, I describe how this problem is tackled. Also, a noteworthy observation is the distribution of images in the validation and test data. The two figures below illustrate that and show that the distribution is not fully uniform. Although it has no effect on the training process, it indicates that the measures used for quality of training results can be biased.

Figure 2: Validation data statistics.

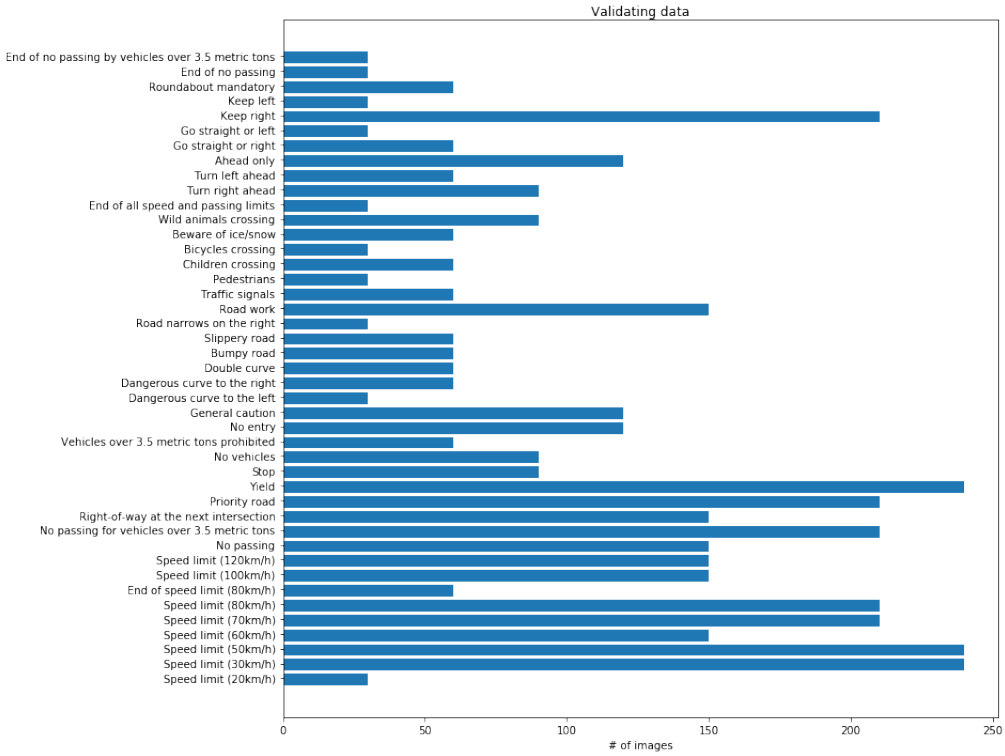
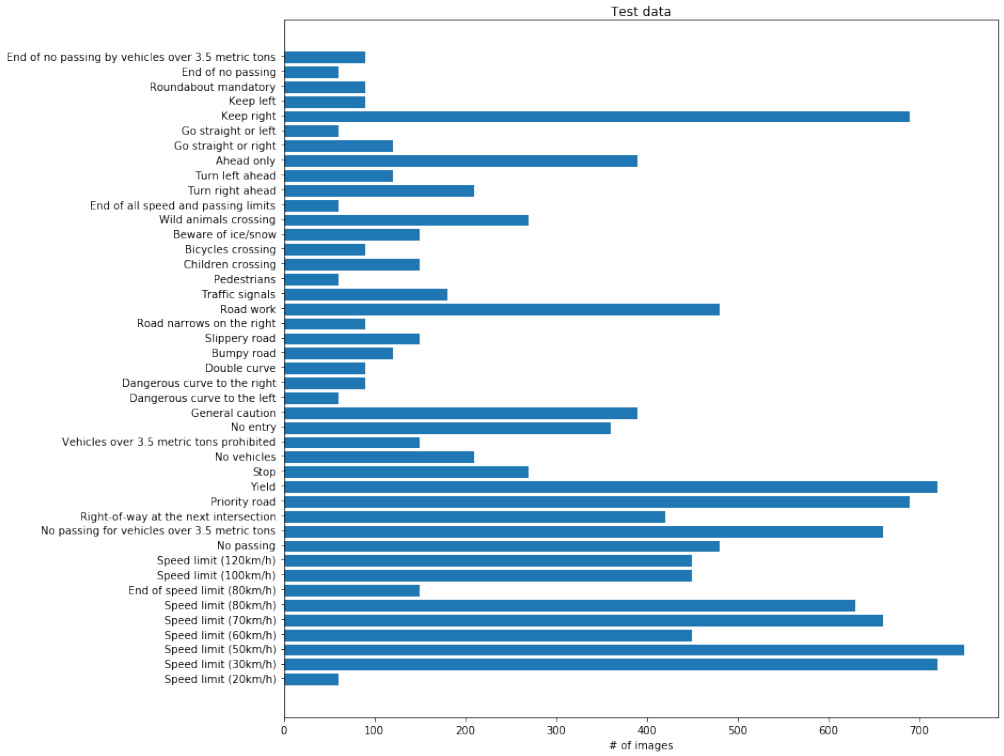


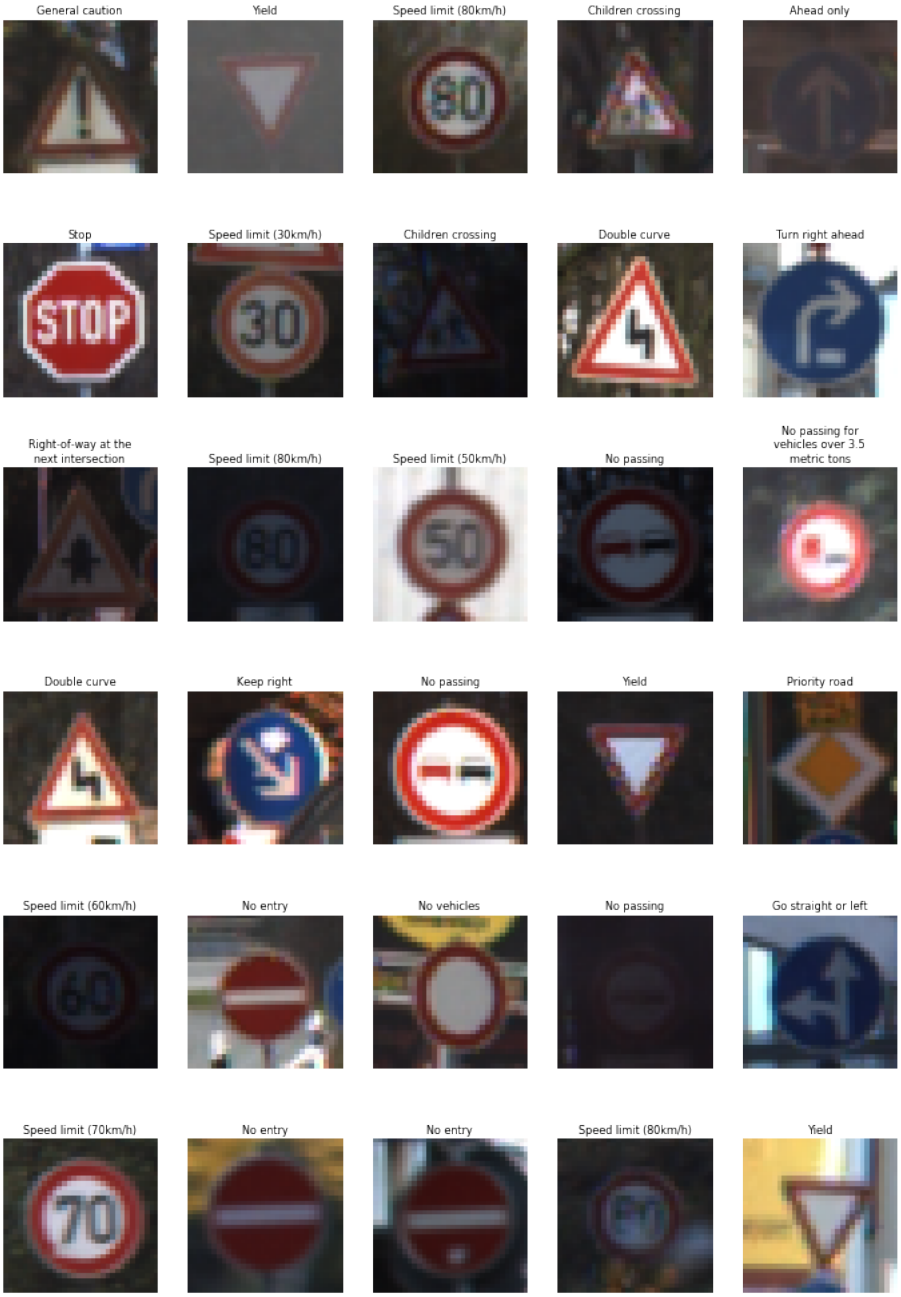
Figure 3: Test data statistics.



Finally, to have a feeling about the input data, I visualized some of the images in the training data as shown below. We can see here that the quality of the images vary a lot. There are some with good lighting, contrast and in general good visibility, while there are others that can hardly be visible. This is also a matter that is to be handled

in preprocessing of data, as discussed in next section.

Figure 4: Visualizing input images.



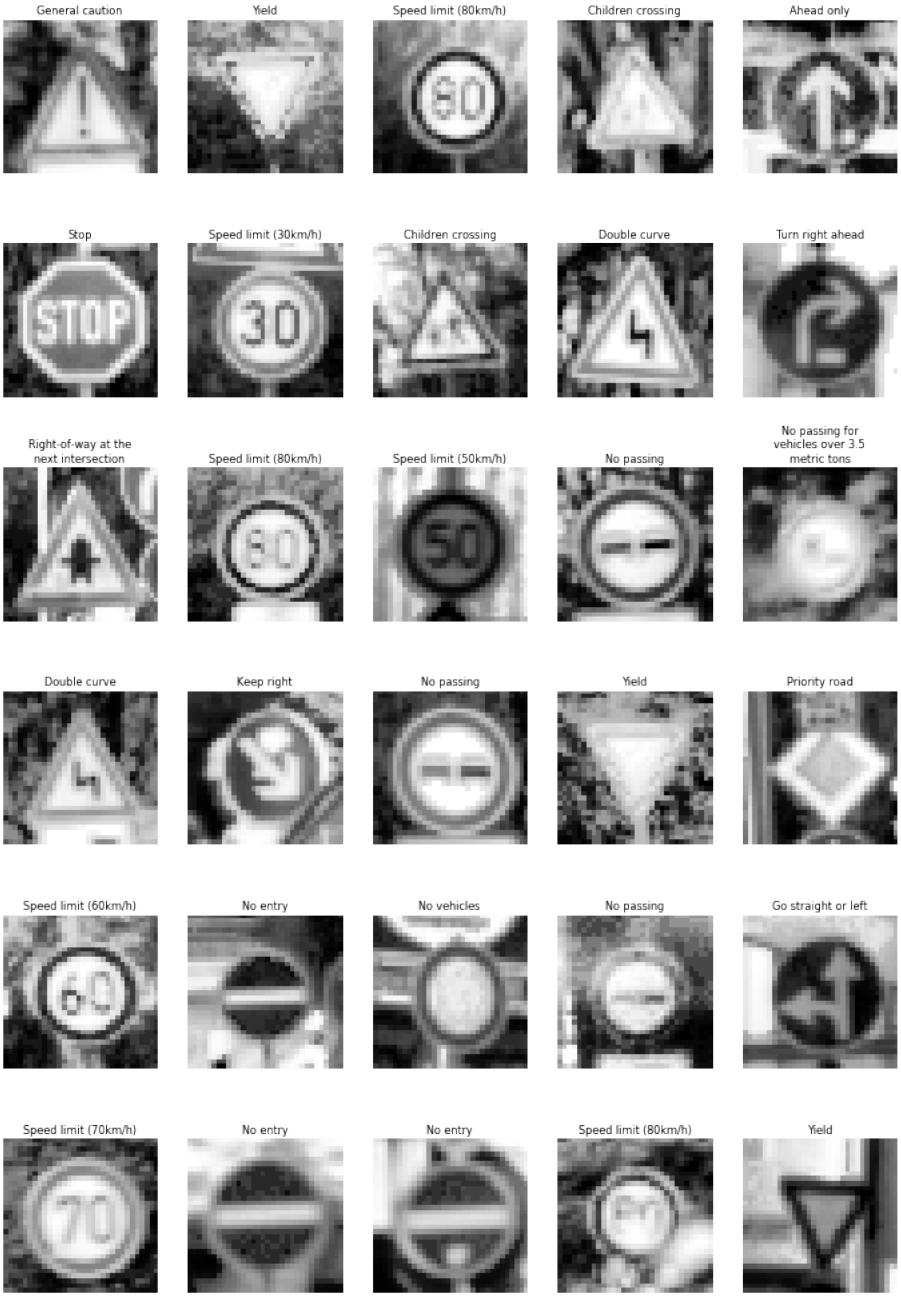
3 Design and Test a Model Architecture

3.1 Preprocessing

Initially, my thought was to use the color images as input and allow the network to choose a suitable color space. To equalize color images, I transfered them to YUP space, equalized them with `opencv.equalizeHist` and transfered back to RGB space. After some initial testing, it appeard that there was not much gain in using RGB images, compared to gray scale ones. It seemed that the input dimension was 3 times larger, but the useful information the same, or at least almost the same. Therefore, I decided to use gray scale input images and use `opencv.equalizeHist` to have comparable quality

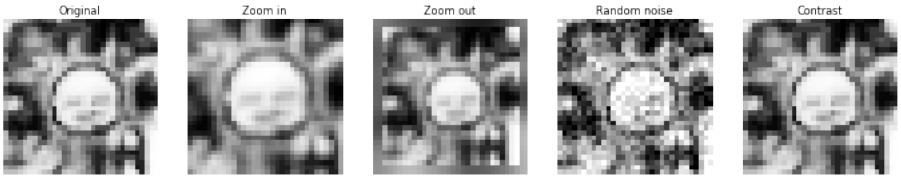
of input signals. The figure below shows the transformed version of above images. It is visible that the information is more visible to the eye and thereby even to the network.

Figure 5: Input pmages after histogram equalizing of gray scale.



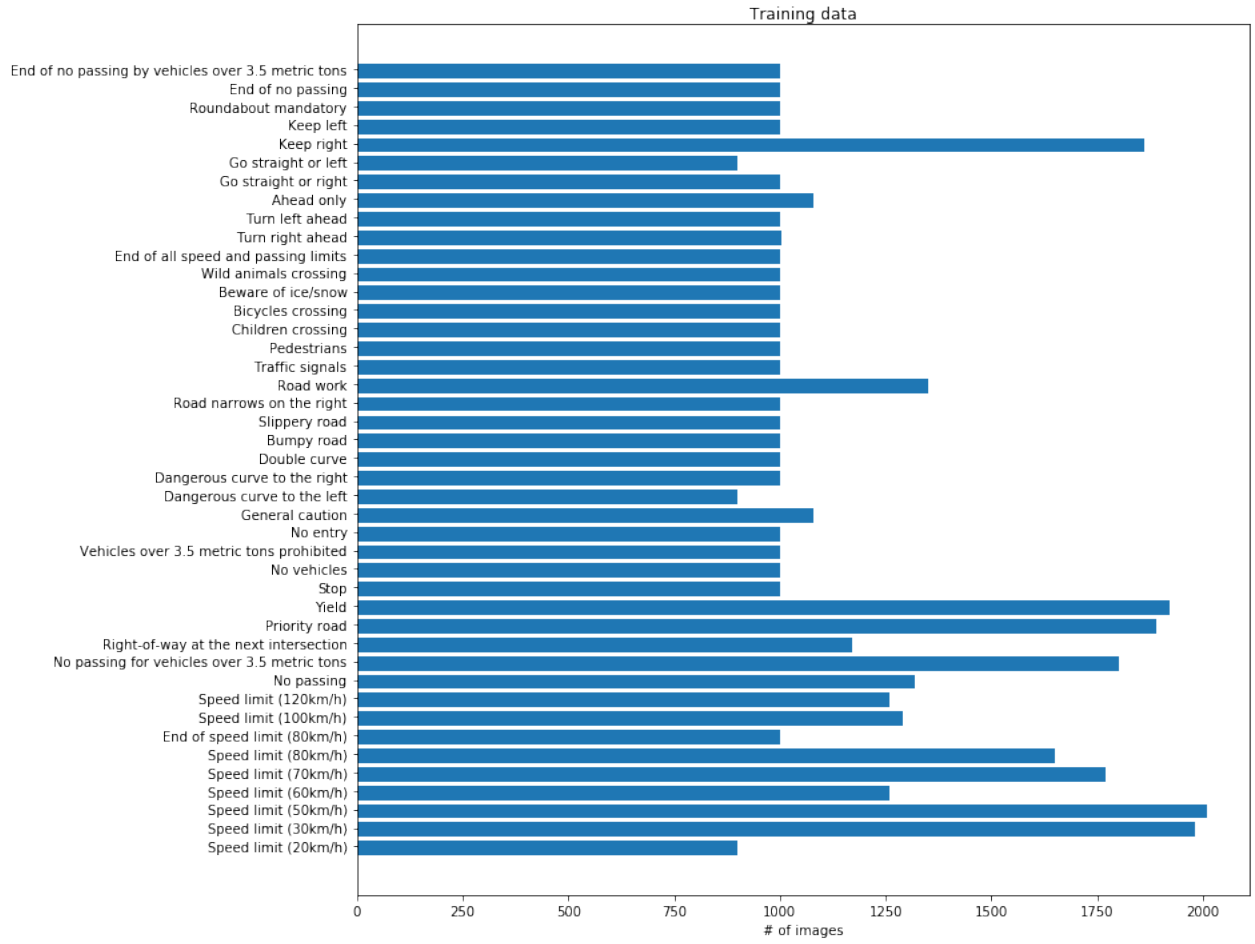
To improve the situation, I tried to conduct data augmentation by adding new images through manipulation of existing ones. To do that, when the number images for a class is lower than 1000, for each image, I add 4 more, one 10% larger, one 10% smaller, one with random noise and one with a modified contrast. The figure below illustrated the 4 more images together wirh the original one.

Figure 6: Data augmentation by transformation of input.



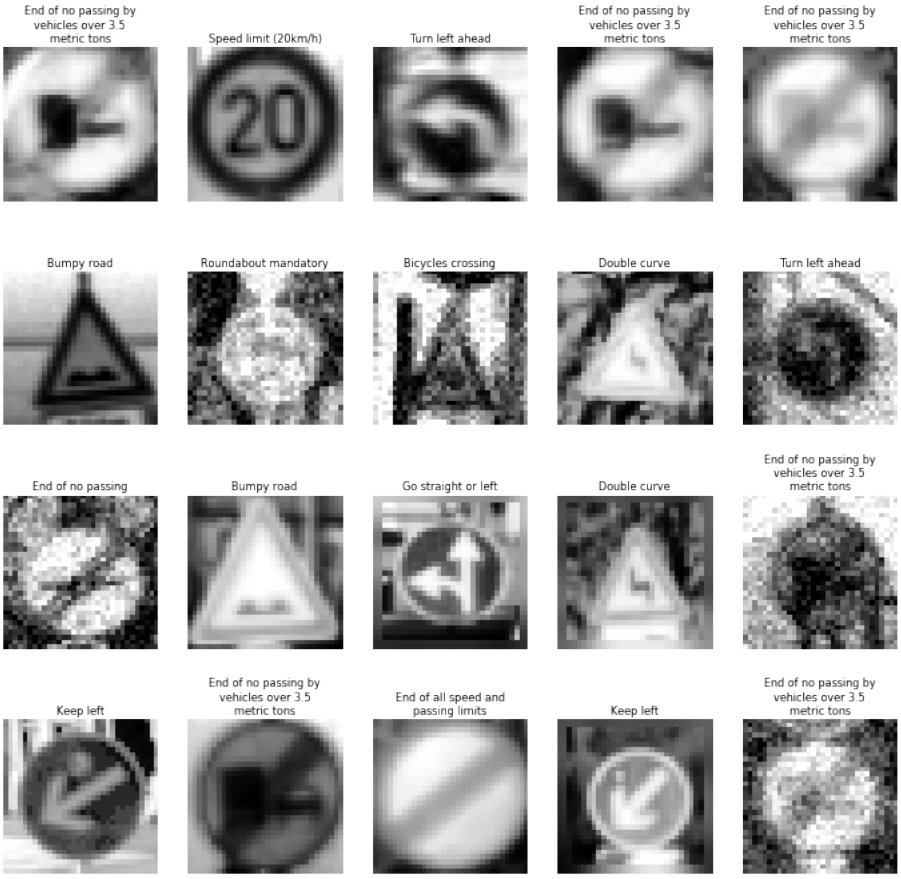
This method was applied to the training set of data resulting in the folowing number of images in the training set, which apparently has a better distribution between different classes.

Figure 7: Train data statistics after augmentation.



The folowing figure shows an example of the images included in the training data after augmentation.

Figure 8: Examples of input data after augmentation.



Finally, before the images were fed to the network, they were normalized so that the values are below one. I tried different normalization schemes, starting from the one suggested in the project description, which normalized the values to $[-1, +1]$, but finally I followed a proposal I found in internet nomalizing the values to $[0.1, 0.9]$. Intuitively, I felt this will be a good way as it can lead that we stay within limits.

3.2 Model Architecture

	Original LeNet	Modified for sign classification
Input	$32 \times 32 \times 1$	$32 \times 32 \times 1$
Convolution layer 1	$32 \times 32 \times 1 \rightarrow 28 \times 28 \times 6$	$32 \times 32 \times 1 \rightarrow 28 \times 28 \times 16$
Activation	ReLu	ReLu
Max pool	$28 \times 28 \times 6 \rightarrow 14 \times 14 \times 6$	$28 \times 28 \times 16 \rightarrow 14 \times 14 \times 16$
Convolution layer 2	$14 \times 14 \times 6 \rightarrow 10 \times 10 \times 16$	$14 \times 14 \times 16 \rightarrow 10 \times 10 \times 32$
Max pool	$10 \times 10 \times 16 \rightarrow 5 \times 5 \times 16$	$10 \times 10 \times 32 \rightarrow 5 \times 5 \times 32$
Activation	ReLu	ReLu
Flattening	$5 \times 5 \times 16 \rightarrow 400$	$5 \times 5 \times 32 \rightarrow 800$
Keep probability	1.0	0.8
Fully connected 1	$400 \rightarrow 120$	$800 \rightarrow 240$
Activation	ReLu	ReLu
Keep probability	1.0	0.5
Fully connected 2	$120 \rightarrow 84$	$240 \rightarrow 120$
Activation	ReLu	ReLu
Keep probability	1.0	0.5
Fully connected 3	$84 \rightarrow 43$	$120 \rightarrow 43$
Output	43	43

The above table presents the original LeNet network that I started with and the final one that I felt happy about. Between the two a relatively large number of cases was tested, but not in a systematic manner. Therefore, maybe a smaller network could also provide similar accuracy.

3.3 Model Training

The weights were initialized with random number with truncated normal distribution with the mean value $\mu = 0$ and the standard deviation $\sigma = 0.1$. All biases were sent to 0 at the initial state.

Adam optimizer was used with a learning rate of 0.001 to reduce the mean cross entropy. During the course of the work, I made many changes in the learning rate, batch size and numeber of epoches. I got a feeling that the learning rate of 0.001 was working quite well and therefore decided to keep it at that value. I could not see any visible changes related to the batch size, which I changed between 64 and 128. This needs to be investigsated in more details.

My observation regarding the number of epoches, was that when the network could model well, the validation accuracy increased fast and reached a high value. Thereafter, it was fluctuating around that value. A larger number of epoches did not seem to have any positive impact.

With this network, I could have quite successful convergence and prediction. The validaiton rate was reached 98% after only 13 epoches and thereafter was fluctuating. I let the training go on for 30 epoches to see the effects, but the observation was that the accuracy did not improve, but the validation was fluctuating between 97.5% and 98.5%. The final test accuracy, after 30 epoches was 94.8%.

3.4 Solution approach

The process to find the solution was somewhat adhoc as it was the first time I set up a network and trained it. Initially, I was not sure whether not reaching the required accuracy is related to a bug in my code or any of hyperparameters and therefore, I needed to test a large number of cases until I somewhat understood the important parameters to focus on.

The input data was prepared in a straight forward manner. The statistics were calculated and data was augmented to reach a reasonable level. Further, as it was evident that the quality of images is varying very much and unless they are tsaken care of, it will become more difficult for the network to be trained. Naturally, the normalization will help, but I found using histogram equalizer would give images that are better comparable and this could help at very low cost.

Without dropout, I sometimes could reach the accuracy of 93%, but just repeating the training could give a different results. Throughout the work, I learned that the network easily overfits the training data and therefore, the valiation accuracy cannot improve unless a regularization tecnique is applied. As a regularization technique, dropout appears as a very attractive method. It is easy to implement and, as it reduces the size of the network, it leads to faster iterations. Although in my case I understood its value at a late stage, I think it should be included in the initial design of the network. The *keep probabilities* was first set to 50% for all layers, but I found the suggested values from literature, i.e. 80% for input layer of the fully connected part and then 50% for hidden layers, gave a better result.

I started with the same network architecture and size as the lab. Initially, I increased the size as I thought the input was richer in features than the hand-written digits we were working with. However, I noticed that increase of the size of the network without regularization was not improving the results. On the other hand, I have a feeling that too small network together with high degree of drop-out will have difficulties te span all necessary features of input data. The selected network seemed to classify the images well, but it could be further optimized.

Initialization of the weight and biases was done in the same way as in the lab. My conclusion from that lab was that truncated normal distribution with $\mu = 0$ and $\sigma = 0.1$.

I made some variation of learning rate, batch size and number of epoches. Regarding the learning rate, I concluded that 0.001 worked quite well for Adam optimizer. The number of epoches does not seem to have a strong effect once it is large enough. Based on my experience from this project, once the network has reached what it best can do, further iterations do not improve the results in average, but they could have impact on a specific class. I chose batch size to be 64, but it is not proven to be the best choice.

4 Test a Model on New Images

4.1 Acquiring New Images

Figure 9: Sign images found on internet. Upper line is the oroginal image and lower the resized one to 32 times 32.

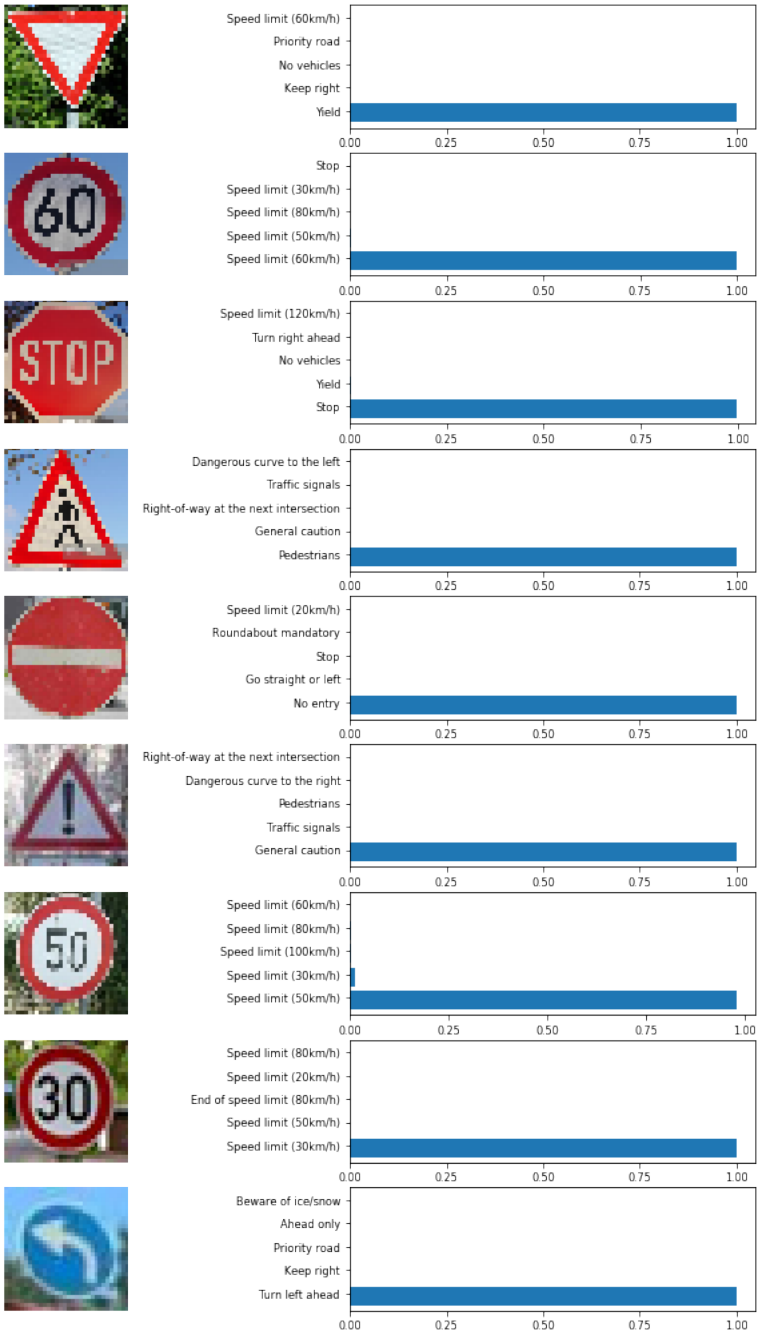
4.2 Performance on New Images

The above nine images were used to test the prediction of the model. In these nine signs, three are speed limits with numbers 30, 50 and 60 that are not easy to distinguish between. Pedestrains and general caution are also very similar, with both triangular shapes and a vertical blach shape inside. Despite the fact that this set of signs was not easy to make predictions, all of the signs were predicted correctly, so that the performance on this set of data was 100%.

4.3 Model Certainty - Softmax Probabilities

As shown in the figures below, also the certainity of prediction is very good. The only visible second choice is 30 km/H (1.2%) versus 50 km/H (98%), i.e. with a significant difference.

Figure 10: Predicted labels for 9 test images.



4.4 Feature maps

4.4.1 Triangular shape

Figure 11: Input imge

Figure 12: Feature maps - Output from first convolutional layer.

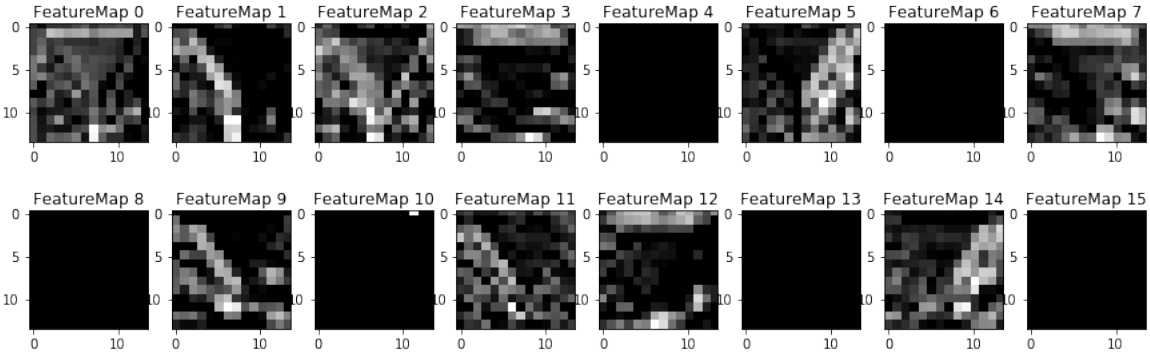
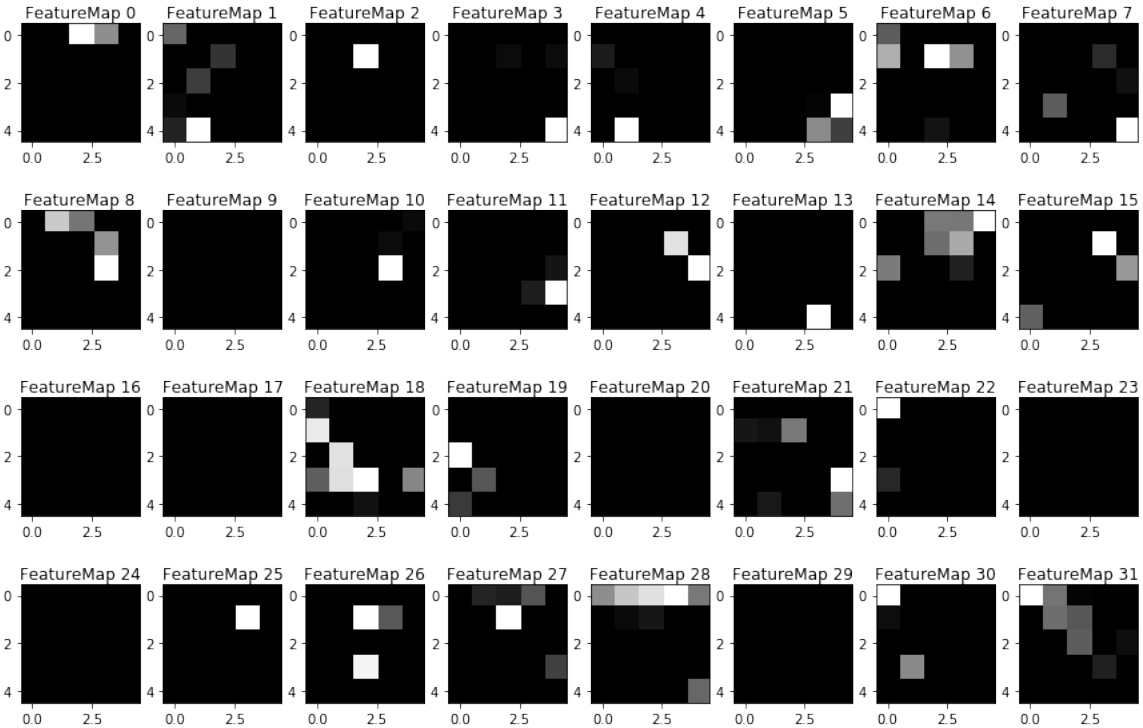


Figure 13: Feature maps - Output from second convolutional layer.



4.4.2 Circular shape and numebrs

Figure 14: Input imge

Figure 15: Feature maps - Output from first convolutional layer.

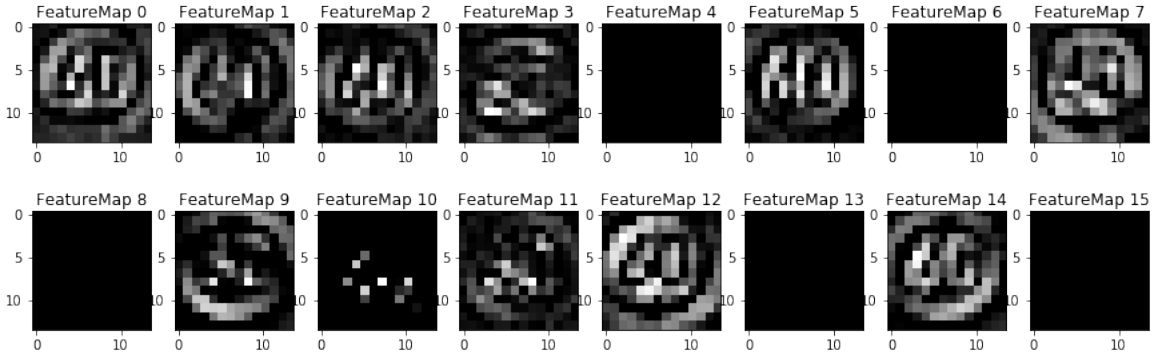
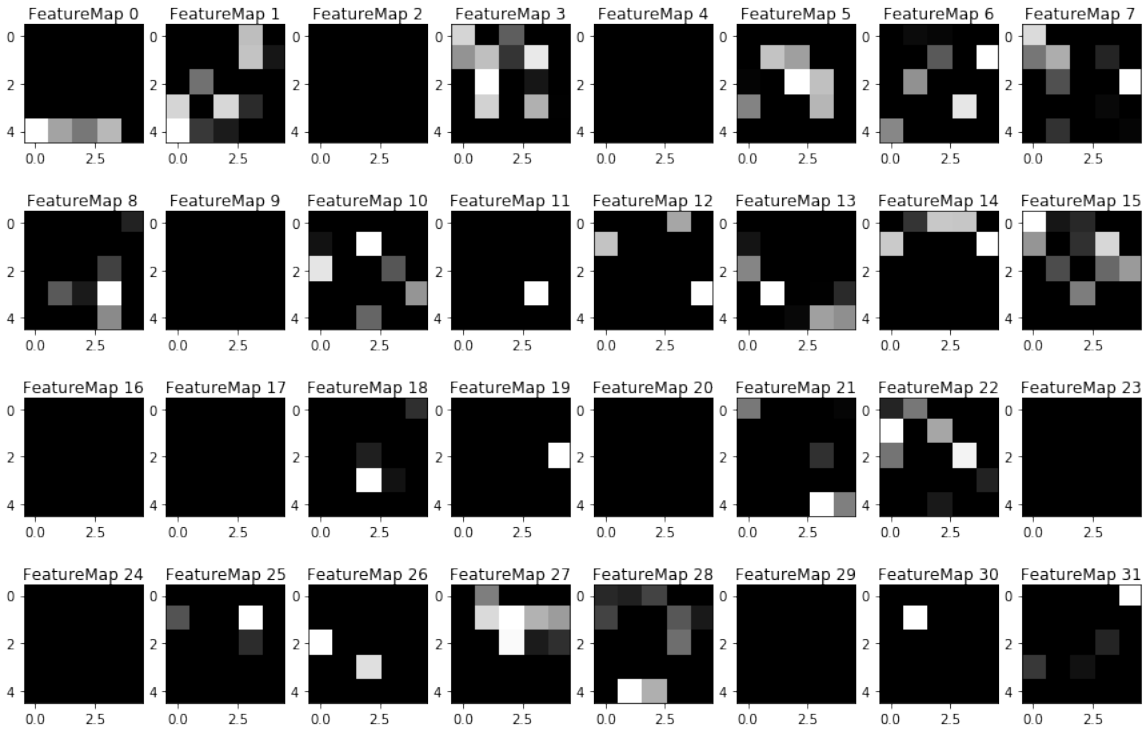


Figure 16: Feature maps - Output from second convolutional layer.



4.4.3 Octagonal with text

Figure 17: Input imge

Figure 18: Feature maps - Output from first convolutional layer.

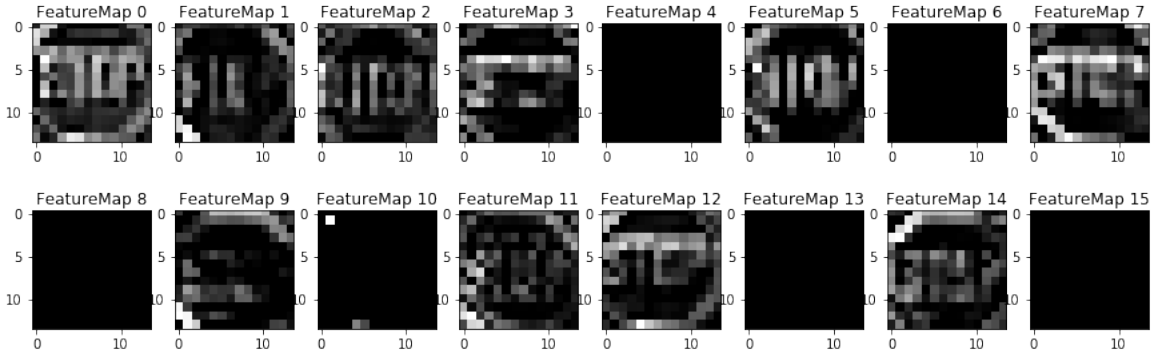


Figure 19: Feature maps - Output from second convolutional layer.