

Traffic Sign Recognition

The goals / steps of this project are the following:

- Load the data set (see below for links to the project 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
- Summarize the results with a written report

Data Set Summary & Exploration

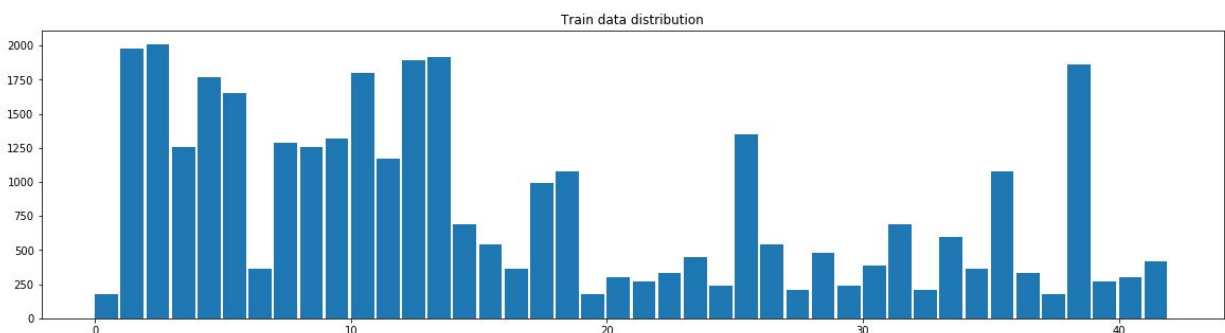
A basic summary of the data set

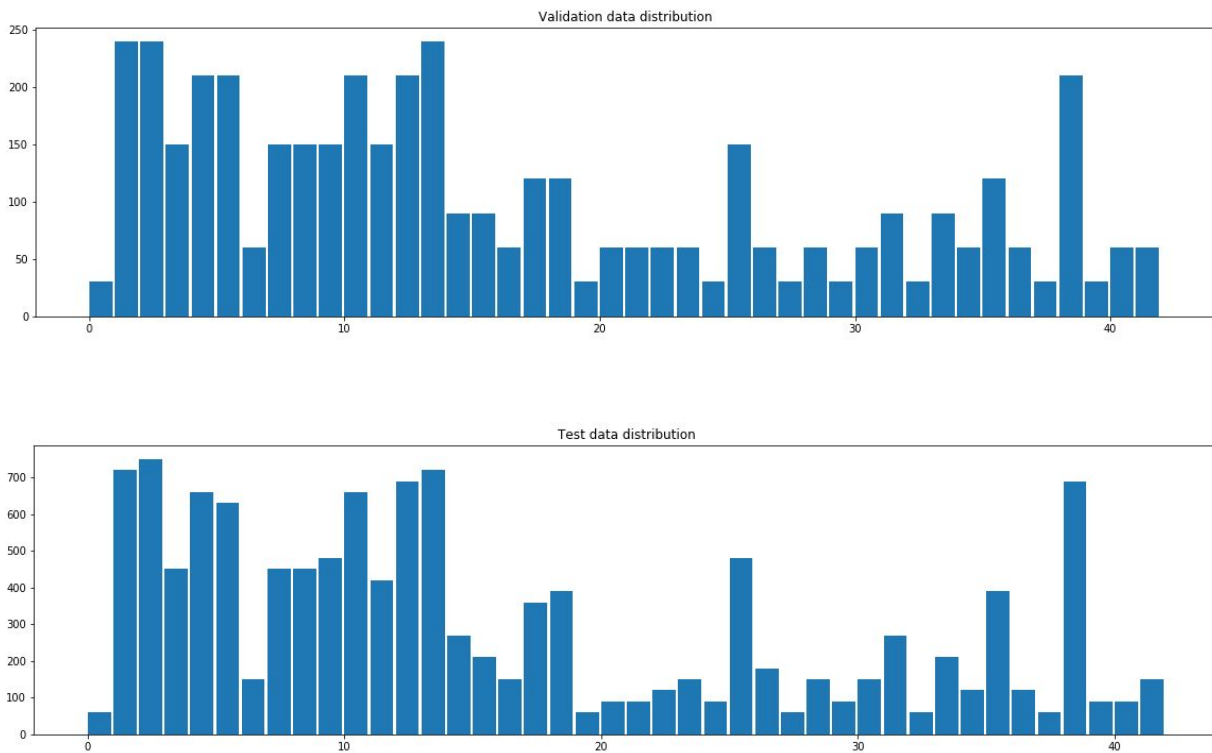
I used the pandas library to calculate summary statistics of the traffic signs data set:

- 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

An exploratory visualization of the dataset

I used matplotlib for data visualization. We can notice that all datasets are skewed in the same way, this fact would be crucial during data preprocessing.





Design and Test a Model Architecture

Image preprocessing

CNN backpropagation alg uses gradient descent method, an engineer should normalize data due to this fact, the alg would find minimum much faster with normalized data.

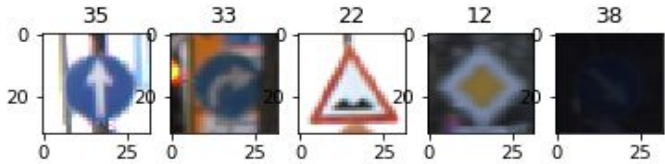
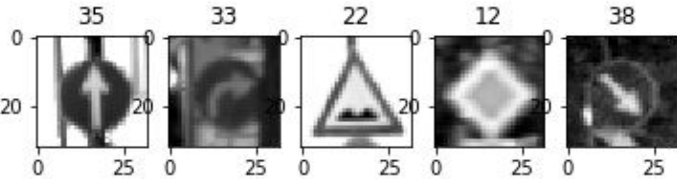
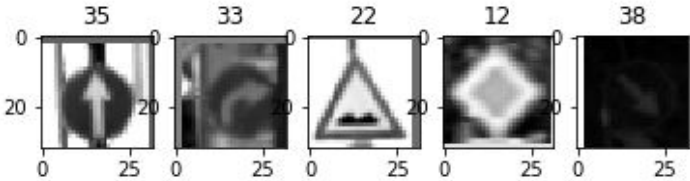
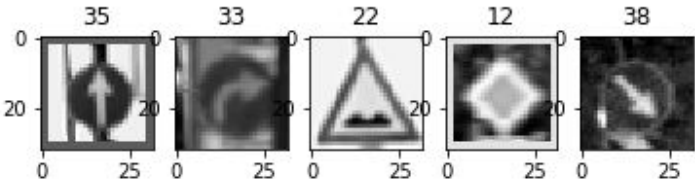
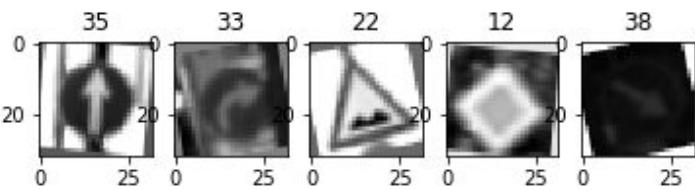
Color is not a crucial feature, we can remove it with the grayscale procedure.

During the first training procedure of LeNet CNN validation accuracy was about 0.85. We could improve CNN performance if we had enough data. LeNet CNN is not sensitive to minor data variations, whence I used augmentation procedure for data generation. Results could be seen below.

All datasets have similar distributions, so data balancing is not important. We could just generate more data and don't think balancing task if we repeat it enough times the augmented dataset would have the same distribution as validation and test datasets.

Augmenting procedure has such steps:

- Randomly take image from train dataset.
- Apply transition, scale and rotation.
- Repeat 100K times.

Original images		Randomly chosen images from train dataset
Grayscaled, normalized		<code>cv2.cvtColor(image, cv2.COLOR_RGB2GRAY).</code> $(\text{pixel} - 128) / 128$
Transition		Randomly shift images for $[-2, 2]$ pixels
Scale		Randomly resize images in ratios = $[0.875, 1, 1.125]$.
Augmented data		Chain all transformations with random rotation on $[-15, 15]$ degrees

Model architecture

My final model consisted of the following layers:

Layer	Description
Input	32x32x1 Grayscale image
Convolution 5x5	1x1 stride, valid padding, outputs 28x28x24
RELU	
Max pooling	2x2 stride, outputs 14x14x24
Convolution 5x5	1x1 stride, valid padding, outputs 10x10x48
RELU	
Max pooling	2x2 strides, outputs 5x5x48
Flatten, 1200	Outputs 120
RELU	
Dropout	
Fully connected, 120	Outputs 84
RELU	
Dropout	
Fully connected, 84	Outputs 43

Model training

My first approach was simple LeNet architecture from character recognition lab. Character and sign recognition are similar problems, whence it is a good idea to try LeNet approach.

Initial architecture performs well, but it was not perfect. Validation accuracy was about 0.85 and test accuracy was about 0.83. Due to fact that validation accuracy was below training accuracy, overfitting case, I decided to use dropout layers, which were designed for solving such cases. Another common technique is to use broader network architecture with dropouts.

I used softmax cross-entropy as output function. Then I calculated mean and used Adam Optimizer for parameters optimization.

Due to the fact of network increasing, I decided to increase epochs parameter, we need more steps to train bigger network.

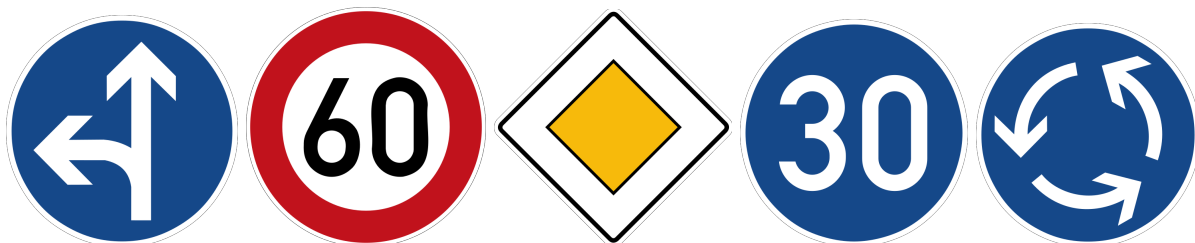
Final hyperparameters: batch size = 128, epochs =40 and probability of keep = 0.5 for dropouts, learning rate = 0.001.

My final model results were:

- training set accuracy of 0.991
- validation set accuracy of 0.985
- test set accuracy of 0.962


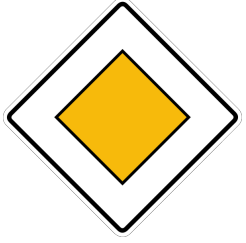


Test a Model on New Images

Here are five German traffic signs that I found on the [wiki](#):



I chose signs that network familiar with, so there could not be any problems with recognition. I outputted top 5 softmax probabilities for each image below:

	<p>Go straight or left : 99.601%</p> <p>Children crossing : 0.080%</p> <p>Ahead only : 0.075%</p> <p>Dangerous curve to the left : 0.044%</p> <p>Right-of-way at the next intersection : 0.044%</p>
--	---

	<p>Speed limit (60km/h) : 100.000%</p> <p>Speed limit (80km/h) : 0.000%</p> <p>Speed limit (30km/h) : 0.000%</p> <p>Speed limit (20km/h) : 0.000%</p> <p>Speed limit (50km/h) : 0.000%</p>
	<p>Priority road : 100.000%</p> <p>No vehicles : 0.000%</p> <p>Keep right : 0.000%</p> <p>Yield : 0.000%</p> <p>Speed limit (50km/h) : 0.000%</p>
	<p>General caution : 99.773%</p> <p>Traffic signals : 0.224%</p> <p>Road narrows on the right : 0.003%</p> <p>Speed limit (30km/h) : 0.000%</p> <p>Road work : 0.000%</p>
	<p>Roundabout mandatory 92.072%</p> <p>Priority road 7.853%</p> <p>Speed limit (100 km/h) 0.075%</p> <p>End of no passing by vehicles over 3.5 metric tons 0.000%</p> <p>Keep right 0.000%</p>

Predictions are correct and straightforward. If an image belongs to train dataset network could recognize it with high accuracy.

The NN's state visualization

I used written code for first activation layer visualization. It is very useful exercise and helps understand network architecture. We can notice how first convolutional layer decomposes an image into several feature maps. On this step, we could recognize some features, but if we try more deep layers it would be much harder for a human to understand it. We should not describe NNs as human brain model, but rather a thing that was inspired by it.

