Traffic Sign Recognition

Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

Build a Traffic Sign Recognition Project

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

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. You can use this template as a guide for writing the report. The submission includes the project code.

You're reading it! and here is a link to my project code

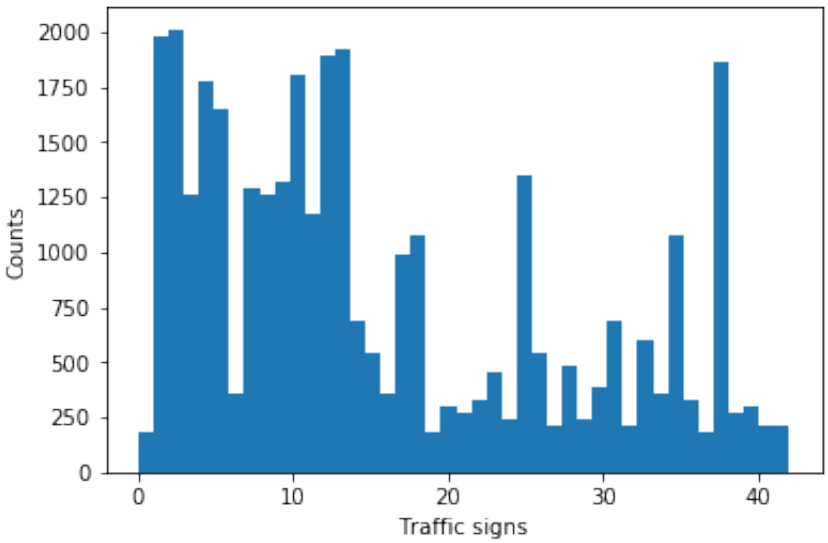
Dataset Exploration

Dataset Summary

To create summaries of the different datasets I used NumPy as well as plain Python, e.g. to get the shapes of the data or the length of unique class labels.

Exploratory Visualization

For visualization I used Matplotlib. I visualized different images from the training and test data to get a better understanding of what inputs the model would operate on. In addition, I created histograms for the labels of the different training sets to get an understanding how equally classes are distributed. It became evident that in fact the classes were quite unevenly distributed.



Design and Test a Model Architecture

Preprocessing

After experimenting and reading the paper from Pierre Sermanet and Yann Le Cun, I found that the usage of color channels did not significantly yield in better accuracy, thus, I decided to convert the images to grayscale.

Well-conditioned problems allow the gradient descent optimizer to converge much faster. Thus, I decided to additionally normalize the input data. After trying different normalization approaches (subtracting mean and dividing by mean, max-min, max-min and translating into -1/1 input space), I found that a max-min-transform worked best for me, i.e. x / 255.

Model Architecture

In general, I tried to stick as good as possible to the original LeNet-architecture, to see how far I could push it. After experimenting I found that introducing Dropout in the fully-connected layers 1 and 2 allowed me to eventually surpass the 93% validation accuracy line.

My final model consisted of the following layers:

Layer	Description
Input	32x32x1 grayscale image
Convolution 5x5	1x1 stride, valid padding, outputs 28x28x6
ReLU	
Max pooling	2x2 stride, outputs 14x14x6
Convolution 5x5	1x1 stride, valid padding, outputs 10x10x16
RELU	
Max pooling	2x2 stride, outputs 5x5x16
Flattening	outputs 1x400
Fully connected	outputs 1x120
ReLU	
Dropout	Keep-prob: 0.5
Fully connected	outputs 1x84
ReLU	
Dropout	Keep-prob: 0.5
Fully connected	outputs 1x43
Softmax	

Model Training

To train the model, I used the AdamOptimizer and tried out different hyperparameter settings:

- learning rate: 0.0001, 0.001, 0.01, 0.1
- epochs: 1 - 100 (step-size: 10)
- keep_prob: 0.5 - 0.9 (step-size: 0.1)
- batch_size: 64 - 512 (step-size: 2^x)

I found that a learning rate of 0.001, 20 epochs, a keep-prob of 0.5, and a batch-size of 256 yielded an appropriate balance of accuracy and runtime of the model.

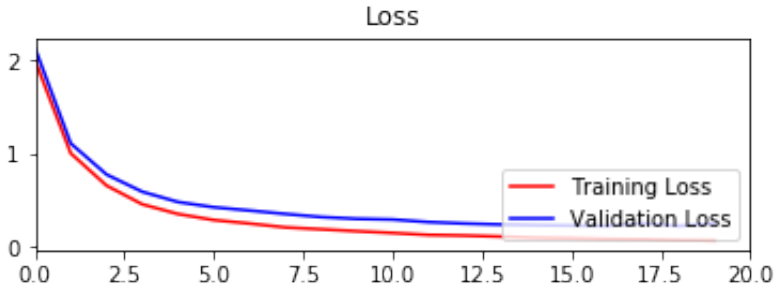
Solution Approach

The LeNet-architecture (and convolutional neural networks in general) have been successfully applied in

the domain of visual classification, e.g. for classifying hand-written digits on the MNIST-dataset. The network learns hierarchical relationships from the inputs and uses these relationships to inform its classification decisions.

As a result, I decided to use the general LeNet-architecture and tweak it to achieve a validation accuracy > 93%. I tried different things, such as classifying with all RGB-color channels on, trying with a lower learning rate, sampling more data by rotating and shifting the original inputs. However, what gave me an eventual boost was the application of Dropouts on the fully-connected layers 1 and 2, because beforehand I was running into serious overfitting problems and in addition was never able to achieve a validation accuracy > 88%. The additional data did not necessarily increase the validation accuracy, thus, I decided to keep the model and learning process as lean as possible. However, I think it would be a great performance boost to - instead of generating new data for all the classes - generate data for all the underrepresented classes (with less than, say, 250, examples).

I collected loss values during training and found that both training and validation loss were generally decreasing together without too much of a difference between the two.

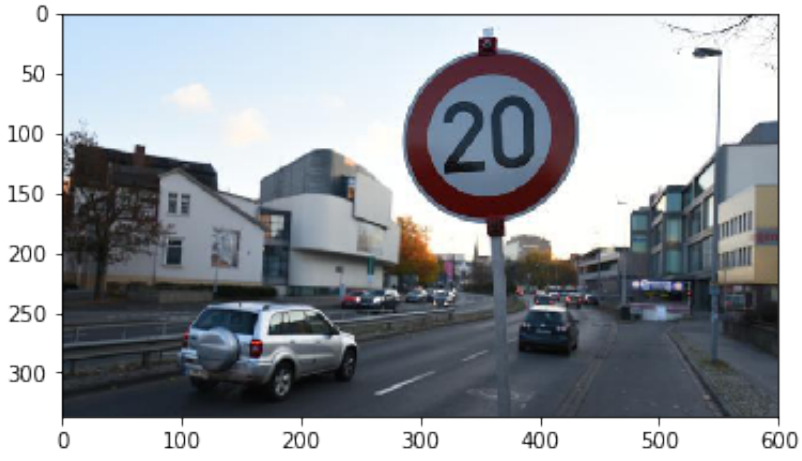


This was also confirmed by a final test accuracy of 91.67%.

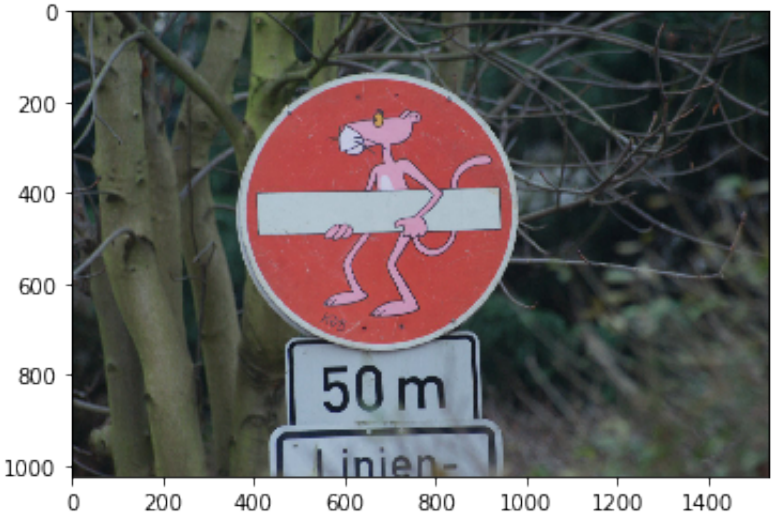
Test a Model on New Images

Acquiring New Images

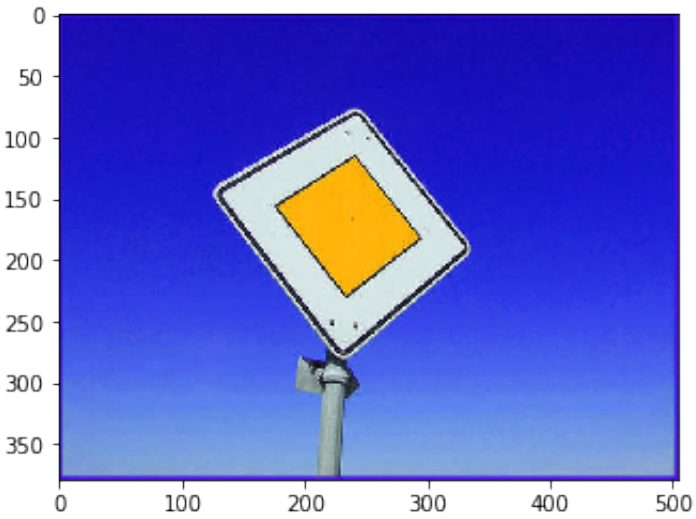
Here are five German traffic signs that I found on the web:




0
200
400
600
800
1000
0
200
400
600
800
1000
1200
1400
1600

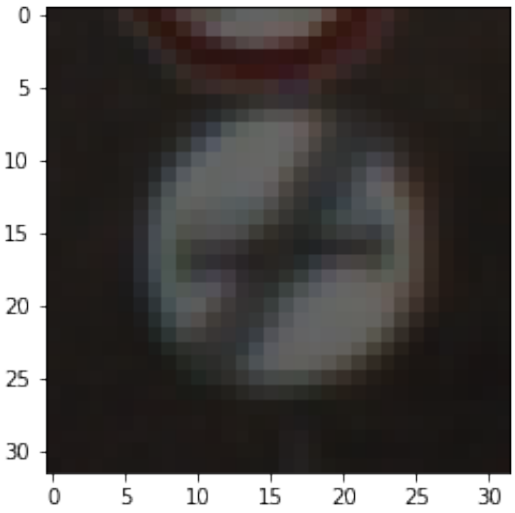

0
200
400
600
800
1000
50 m
0
200
400
600
800
1000
1200
1400


0
50
100
150
200
250
0
100
200
300
400
500



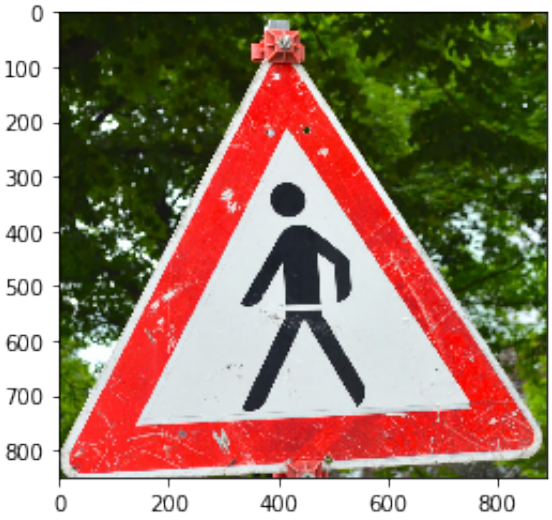
Apart from the fact that the angle e.g. for images 1, 2, and 5 is slightly skewed, i.e. we are not looking straight onto the traffic sign, the first classifications with the network were horrible. :-) But this was rather because the network has been trained on "close-ups" of the signs, i.e. the signs as such were really the focus-point of the image.

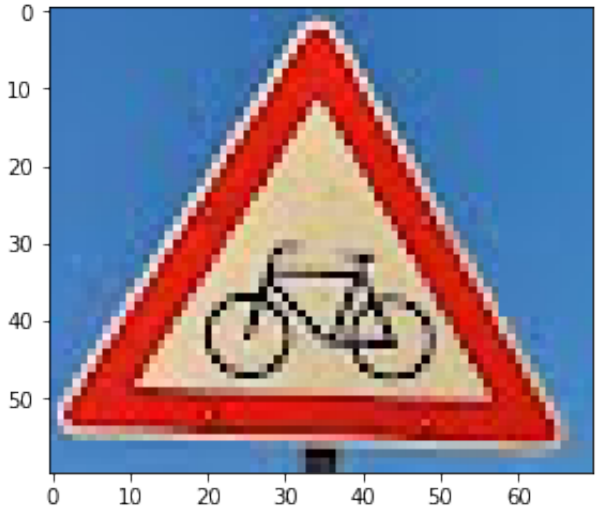
One example:



Hence, I decided to crop the images and really just select the sub-area in which the traffic sits.

Examples after selecting areas of interest:





Performance on New Images

Here are the results of the prediction:

Image	Prediction
Speed limit (20km/h)	Speed limit (30km/h)
Pedestrians	Right-of-way at the next intersection
No entry	No entry
Bicycles crossing	Dangerous curve to the left
Priority road	Priority Road

The model was able to correctly guess 2 of the 5 traffic signs, which gives an accuracy of 40%. Compared to the training accuracy of 91.67% these results do not seem overly good. A reason for this lower accuracy might be that the new test images differ from the training/validation/test images by being taken from a different angle or are distorted in the perspective. Similarly, this difference could be a result from the class-imbalance of the training data which gives certain classes more weight in favour of others.

Model Certainty - Softmax Probabilities

For the first image, the model is relatively sure that this is a speed limit of 30 km/h (probability of 0.959). However, the test input was a 20km/h sign. Considering the initial frequency distribution of the signs, it might be possible that a reason for this is the over-representation of 30km/h signs vs. the underrepresentation of the 20km/h sign.

The top five soft max probabilities were:

Probability	Prediction	Correct
0.959	Speed limit (30km/h)	No
0.040	Speed limit (50km/h)	No
0.001	Speed limit (20km/h)	Yes
0.000	Wild animals crossing	No
0.000	End of speed limit (80km/h)	No

For the second image, the model is relatively sure that this is a Right-of-way at the next intersection (probability of 0.972). However, the test input was a pedestrians sign (which looks at least similar). Considering the initial frequency distribution of the signs, it might be possible that a reason for this is again the class-imbalance. (+ possible difficulties of the network to deal with images that are taken from a different angle).

The top five soft max probabilities were:

Probability	Prediction	Correct
0.972	Right-of-way at the next intersection	No
0.023	Pedestrians	Yes
0.004	Beware of ice/snow	No
0.000	Road narrows on the right	No
0.000	General caution	No

For the third image, the network is absolutely sure that this is a no entry sign (probability of 1) and it is right.

The top five soft max probabilities were:

Probability	Prediction	Correct
1.	No entry	Yes
0.000	Turn left ahead	No
0.000	Roundabout mandatory	No
0.000	Road narrows on the right	No
0.000	Stop	No

For the fourth image, the network is not really sure sure about the decision.

The top five soft max probabilities were:

Probability	Prediction	Correct
0.292	Dangerous curve to the left	No
0.180	Bicycles crossing	Yes
0.167	Slippery road	No
0.161	Wild animals crossing	No
0.091	Road work	No

For the fifth image, is again absolutely sure about it's decision and is right.

The top five soft max probabilities were:

Probability	Prediction	Correct
1.000	Priority road	No
0.000	Roundabout mandatory	Yes
0.000	Yield	No
0.000	No entry	No
0.000	Speed limit (30km/h)	No

In conclusion: Mixed results which could be improved by correction of image distortion, adding more training examples (shifted, tilted, etc.) and/or improving on class-imbalance. A good aspect for now is that the original images were at least always in the top 5 of the softmax-outputs.