Project 3: LeNet Traffic Sign Classification

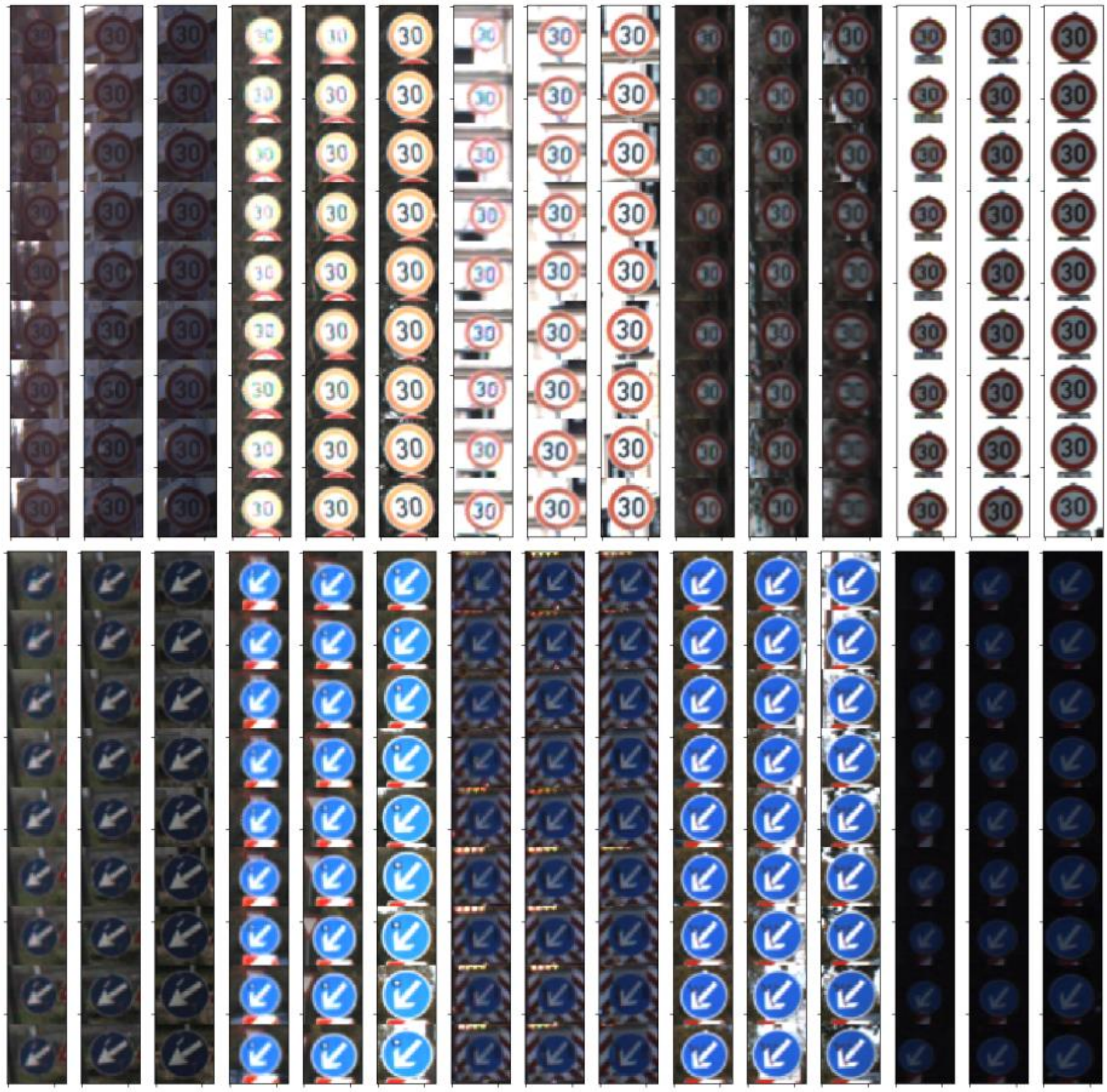
Sonntag, 5. Januar 2020 13:05

1. Dataset Exploration

The table below shows the possible classes of traffic signs as well as their occurences within in database:

ClassId	SignName	Training	Valid	Test
0	Speed limit (20km/h)	180	30	60
1	Speed limit (30km/h)	1980	240	720
2	Speed limit (50km/h)	2010	240	750
3	Speed limit (60km/h)	1260	150	450
4	Speed limit (70km/h)	1770	210	660
5	Speed limit (80km/h)	1650	210	630
6	End of speed limit (80km/h)	360	60	150
7	Speed limit (100km/h)	1290	150	450
8	Speed limit (120km/h)	1260	150	450
9	No passing	1320	150	480
10	No passing for vehicles over 3.5 metric tons	1800	210	660
11	Right-of-way at the next intersection	1170	150	420
12	Priority road	1890	210	690
13	Yield	1920	240	720
14	Stop	690	90	270
15	No vehicles	540	90	210
16	Vehicles over 3.5 metric tons prohibited	360	60	150
17	No entry	990	120	360
18	General caution	1080	120	390
19	Dangerous curve to the left	180	30	60
20	Dangerous curve to the right	300	60	90
21	Double curve	270	60	90
22	Bumpy road	330	60	120
23	Slippery road	450	60	150
24	Road narrows on the right	240	30	90
25	Road work	1350	150	480
26	Traffic signals	540	60	180
27	Pedestrians	210	30	60
28	Children crossing	480	60	150
29	Bicycles crossing	240	30	90
30	Beware of ice/snow	390	60	150
31	Wild animals crossing	690	90	270
32	End of all speed and passing limits	210	30	60
33	Turn right ahead	599	90	210
34	Turn left ahead	360	60	120
35	Ahead only	1080	120	390
36	Go straight or right	330	60	120
37	Go straight or left	180	30	60
38	Keep right	1860	210	690
39	Keep left	270	30	90
40	Roundabout mandatory	300	60	90
41	End of no passing	210	30	60
42	End of no passing by vehicles over 3.5 metric tons	210	30	90

In order to get a feeling of the data covered within the dataset, a file `"dataset_visualization.py"` was written. Beneath, pictures for "Speed limit 30 km/h" and "Keep left" are shown.



It can be seen, that there is a huge variance in lighting conditions. The pose of the traffic sign does not vary that much.

2. Design and Test Model Architecture

1. Image preprocessing

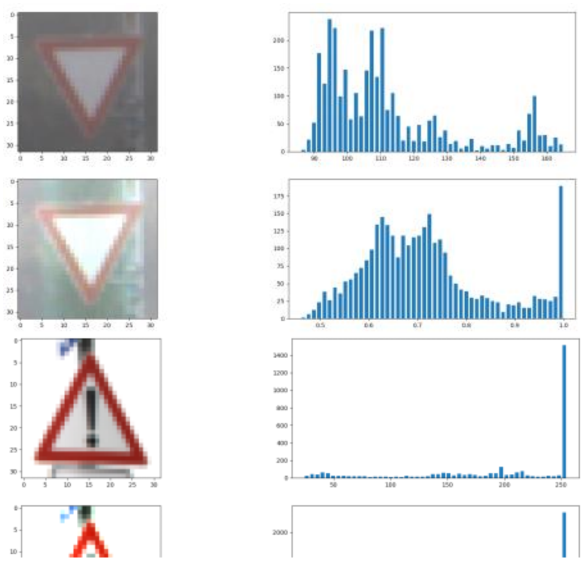
When using images in neural networks, the used data needs to be similar distributed, s.t. a classifier can be trained correctly. For normalizing the images, two approaches are compared.

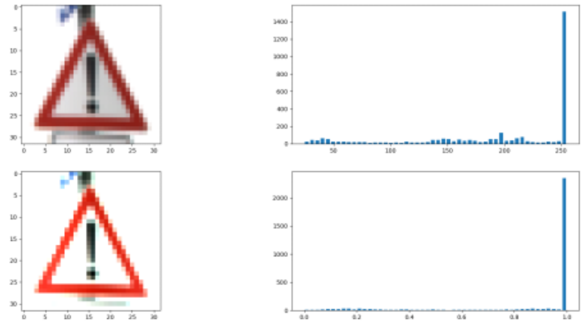
a. Image preprocessing with per-channel image normalization

In this approach, the mean and standard-deviation of each channel is created. The final image value x is then determined by following steps:

```
x = (x - means) / stds
x = np.clip(x, -1.0, 1.0)
# shift from [-1,1] to [0,1] with 0.5 mean
x = (x + 1.0) / 2.0
```

Beneath are two images (top: original with histogram, bottom: normalized with histogram)

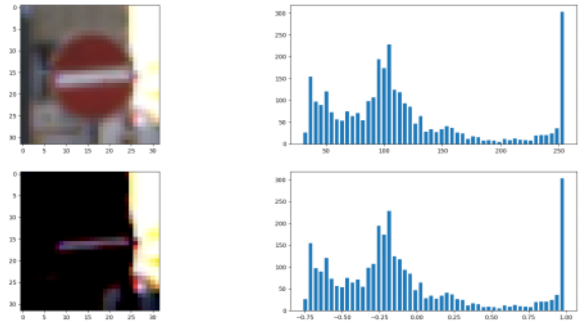




b. Image preprocessing with Udacity-proposed normalization

In this approach, the image values are tried to be centered around 0-mean due to a -128 shift. In order to get values in the boundaries [-1, 1], a division by 128 is done.

```
x = (x - 128.0) / 128
```



c. Image preprocessing with adaptive histogram and contrast limiting (CLAHE)

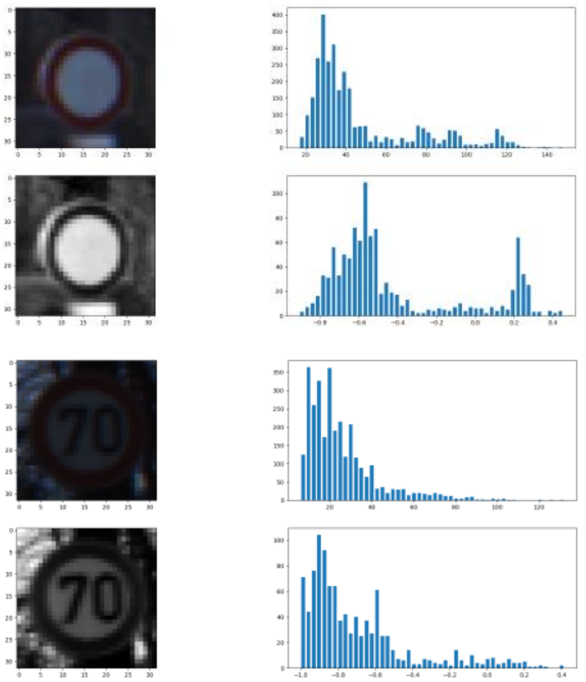
This approach equalizes the histogram of the image. But instead of equalizing the histogram of the whole image, the image is divided into small blocks, which are then equalized.
In order to reduce amplification of noise, the contrast is limited - e.g. if a histogram bin exceeds a given contrast limit, pixels are clipped and distributed uniformly to other bins.

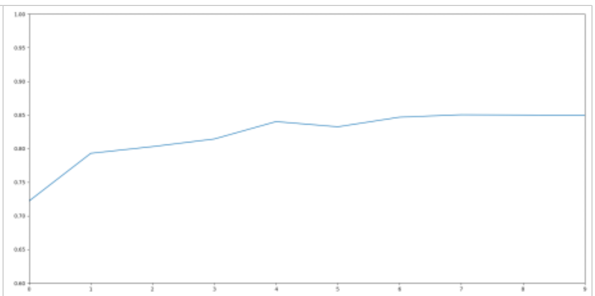
```
clahe = cv2.createCLAHE(clipLimit=2.5, tileGridSize=(4,4))
```

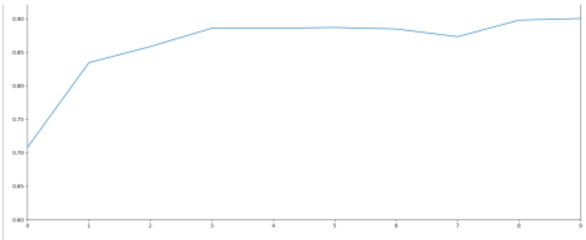
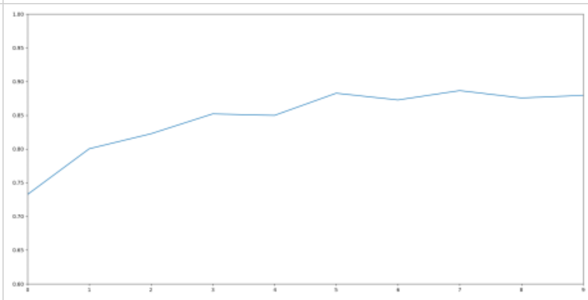
After applying CLAHE, the pixel values are again centered to have 0-mean:

```
x = (x - 128.0) / 128
```

Following pictures show example outputs before and after applying CLAHE.

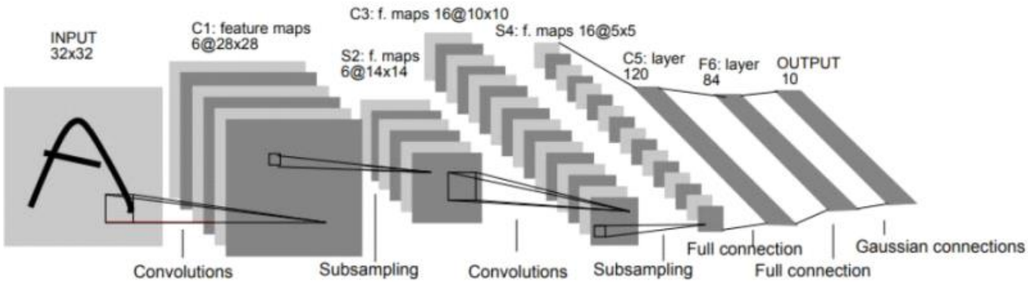


With per-channel image normalization (x - means) / standard_deviation	Final Validation Accuracy: 0.849 Final Test Accuracy: 0.847	
With Udacity-proposed image normalization (x - 128) / 128	Final Validation Accuracy: 0.900 Final Test Accuracy: 0.897	

		
With histogram normalization	Final Validation Accuracy: 0.915 Final Test Accuracy: 0.909	
Without per-channel image normalization	Final Validation Accuracy: 0.879 Final Test Accuracy: 0.883	

The best results could be calculated with histogram-normalization using CLAHE. Therefore this approach is used.

2. Model Architecture



The input of the new traffic sign dataset was of size [32x32x3], s.t. the first layer depth needed to be adopted. Furthermore, the new dataset consists of the three RGB-color-layers. To handle this, the dimensions of the weights needed to be adopted. When experimenting with the given LeNet, it was very interesting to see, that the settings of the truncated_normal in order to initialize the weights was very important:

```
tf.truncated_normal(shape=(5,5,3,6), mean = mu, stddev = sigma)
```

If mean and standard deviation are not explicitly given, a standard deviation of 1 is assumed. Since the standard deviation of the images after normalization is ca. 0.5. When the LeNet is run with a standard deviation of 1, the final accuracy is < 0.1. When tuning this parameter to 0.5, the final accuracy is 0.75. Only when setting the standard deviation to 0.1, the results are above 0.85.

Then I experimented with dropout-layers and a rate of :

- When introducing dropout with rate=0.5 after layer 1 convolution, the final validation accuracy was 0.920, test accuracy 0.898.
- **When introducing dropout with rate=0.25 after layer 1 convolution, the final validation accuracy was 0.927, test accuracy 0.912.**
- When introducing dropout with rate=0.5 after layer 1 max pooling, the final validation accuracy was 0.841, test accuracy 0.837.
- When introducing dropout with rate=0.5 after layer 2 convolution, the final validation accuracy was 0.888, test accuracy 0.889.
- When introducing dropout with rate=0.5 after layer 2 max pooling, the final validation accuracy was 0.875, test accuracy 0.872.
- When introducing dropout with rate=0.5 after layer 3 multiplication, the final validation accuracy was 0.881, test accuracy 0.867.
- When introducing dropout with rate=0.5 after layer 4 multiplication, the final validation accuracy was 0.882, test accuracy 0.878.
- When introducing dropout with rate=0.5 after layer 5 multiplication, the final validation accuracy was 0.488, test accuracy 0.484.
- When introducing additional dropout with rate=0.5 after layer1 and 2, the final validation accuracy was 0.856, test accuracy 0.842.
- When introducing additional dropout with rate=0.25 after layer 1 convolution and layer 4 matrix multiplication, the final val. acc was 0.916, test acc. 0.884.

Finally, I could not see any advantage in my net accuracy when introducing an dropout.

2. Solution Approach / Optimizing the LeCun-Net

First, I tried to change some of the hyper-parameters. When changing one hyper-parameter, the others will be constant, s.t. a comparison is possible. The standard-paramter contain: Batch_size = 128; Learning_rate = 0.001; EPOCHS = 10

Hyper-Parameter	Value	Final Validation Accuracy	Final Test Accuracy
Batch_size	64	0.911	0.900
	128	0.915	0.909
	256	0.892	0.897
	512	0.858	0.866
Learning_rate	0.001	0.900	0.897
	0.005	0.928	0.916
	0.01	0.906	0.884

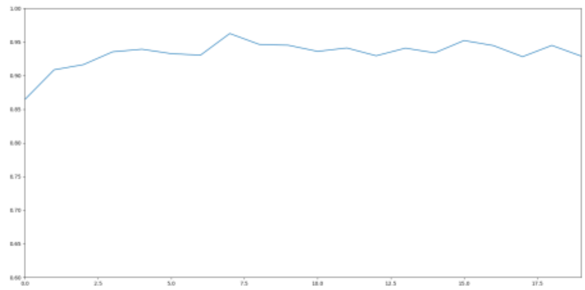
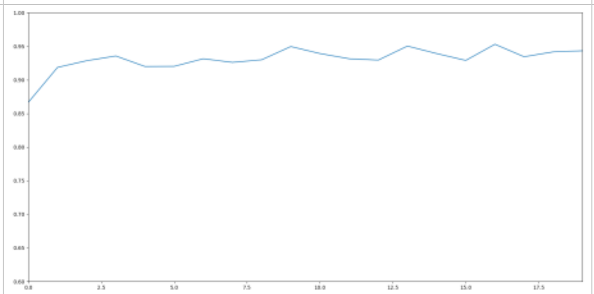
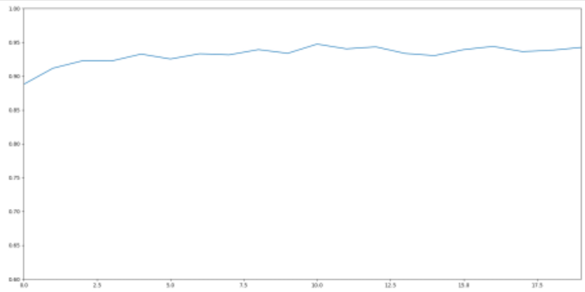
	0.05	< 0.1	< 0.1

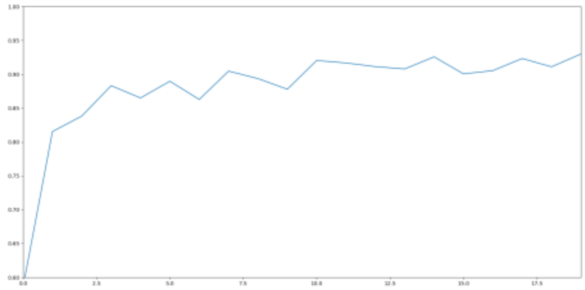
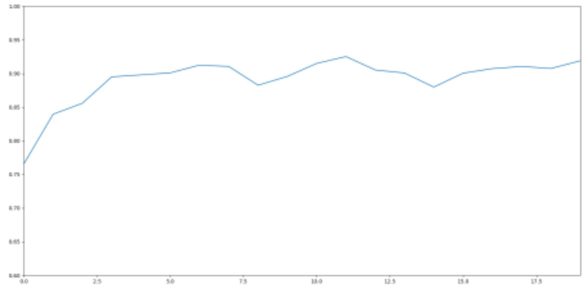
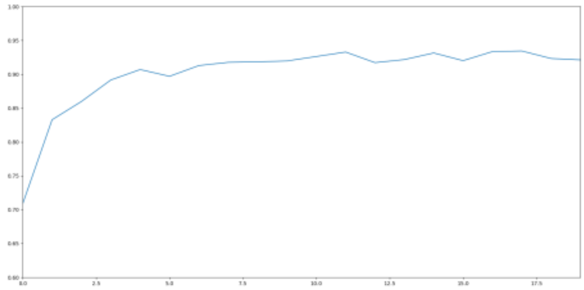
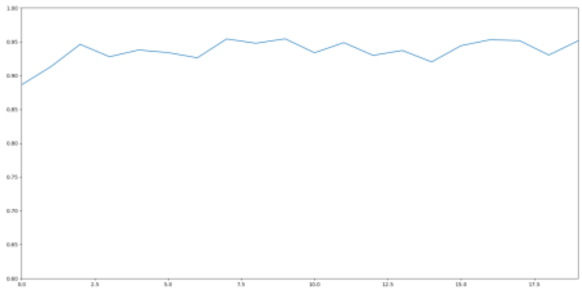
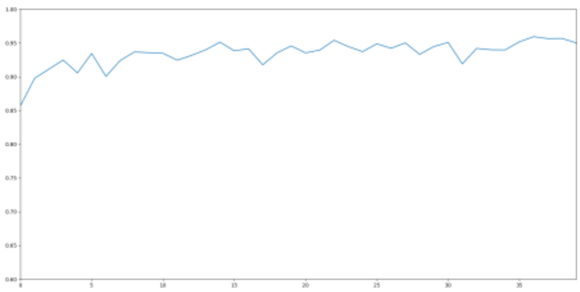
Due to these experiments, following parameters seem to be best:
Batch_size = 64; EPOCHS = 10; Learning_rate = 0.005
The final validation accuracy was 0.899, while the test accuracy was 0.905.

The given model did not get a higher accuracy > 0.93, s.t. I needed to adapt the model architecture. In order to compensate the higher input-complexity (LeNet was designed for digits out of 10 possible classes [0..9], the new net needs to interpret 43 possible classes from light-and background-variant images), I introduced larger layer-sizes:

ID	Layer	Description
0	Input	32x32x1 (when using histogram normalization) 32x32x3 (when using Udacity-normalization)
1	Convolution Activation Max Pooling	Output 28x28x12 with 5x5-filter. Activation is done with ReLU. Output 14x14x12 with 2x2-Pooling-Kernel.
2	Convolution Activation Max Pooling Flattening	Output 10x10x24 with 5x5-filter. Activation is done with ReLU. Output 5x5x24 with 2x2-Pooling-Kernel. Output 5 * 5 * 24 = 600.
3	Fully Connected Activation	Output = 120. Activation is done with ReLU.
4	Fully Connected Activation	Output = 84. Activation is done with ReLU.
5	Fully Connected	Output = 43. (Number of Traffic Sign Classes)

For those larger layer-sizes, the batch-size of 128 seemed to be too low, so I have experimented with taking batch sizes of 196 and 256.

Architecture	Validation Accuracy during training epochs	Final Validation Accuracy	Final Test Accuracy
Batch-Size = 196 Learning-Rate = 0.005		0.929	0.909
Batch-Size = 256 Learning-Rate = 0.005		0.943	0.921
Batch-Size = 300 Learning-Rate = 0.005		0.942	0.92
Batch-Size = 300, Learning-Rate = 0.005 L2-Regularization with Beta = 0.01	No picture taken	0.920	0.901

Batch-Size = 300, Learning-Rate = 0.001 L2-Regularization with Beta = 0.01		0.930	0.901
Batch-Size = 300, Learning-Rate = 0.002 L2-Regularization with Beta = 0.01		0.919	0.908
Batch-Size = 300, Learning-Rate = 0.001, L2-Regularization with Beta = 0.001		0.921	0.908
Batch-Size = 300, Learning-Rate = 0.005, L2-Regularization with Beta = 0.001		0.951	0.929
Batch-Size = 300, Learning-Rate = 0.005, L2-Regularization with Beta = 0.001, Epochs = 40		0.950	0.935
Batch-Size = 300, Learning-Rate = 0.005, L2-Regularization with Beta = 0.001, Epochs = 40 (run with local Jupyter Notebook)	See notebook-html	0.944	0.937

The final validation accuracy with batch size of 300 was at 0.944, the test accuracy 0.937.

3. Test a model on new images

In this section, new images are classified with the above described LeNet neural network.

New test images	Source	Expected Output	Output	Comment

	https://www.noz.de/deutschland-welt/niedersachsen/artikel/651883/hochstens-mit-80-km-h-gegen-den-baum-gericht-entscheidet	5	0	Due to the small image size of 32x32, some of the spatial information (like numbers) get lost. To identify the shape is easier, than identifien the numbers within the sign. Top5-Soft-Max-probabilities: [0 6 1 40 38] Probability for 0 was 0.88.
	https://www.rbb24.de/politik/thema/2017/abgasalarm/beitraege/Tempo-30-in-Berlin-Luftreinhaltung.html	1	1	Probability for 1 was 1.
	https://www.moz.de/landkreise/maerkisch-oderland/seelow/artikel7/dg/0/1/1726205/	1	1	Special about this sign: perspective is warped. Traffic sign was correctly identified. Probability for 1 was 1.
	https://www.bussgeldkatalog.org/geschwindigkeitsbegrenzung-aufgehoben/	6? (close to 6 - End of 80 or 32 - End of Speed Limits)	6	Special about this sign: dataset does not contain this class; output should be close to nearest signs like "End-of-80km/h" or "End-of-Speed-Limits". Top5-Soft-Max-probabilities: [6 36 20 3 41]] Probability for 6 was 1.
	https://www.ka-news.de/region/karlsruhe/Karlsruhe~/A5-bei-Karlsruhe-Grosse-Verwirrung-um-Ende-des-80er-Tempolimits;art6066,1683710	5	37	Special about this sign: traffic sign is rotated very much. Since there is no color-information used any more, this traffic sign seems to have something in common with "go straight or left" Top5-Soft-Max-probabilities: [37 40 34 26 35] Probability for 37 was 0.97.
	https://www.oeamtc.at/news/oeamtc-begruesst-geplante-aufhebung-von-ig-l-hunderter-fuer-e-autos-25287936	7	14	Special about this sign: picture taken from Traffic Sign Bridge in Germany. Those signs do have a black background instead of the white color - which is not trained by the datast. This traffic sign was identified as a "Stop"-sign - probably because the pixel-intensity of the circle and text is high in comparison to the rest (same holds true for white text and white border of Stop-sign. Top5-Soft-Max-probabilities: [14 39 13 37 38] Probability for 14 was 0.75.
	https://www.autobild.de/artikel/vorfahrt-gewaehren-44209.html	13	13	Probability for 13 was 1.

I wanted to test the model on new, unknown data. My evaluation had shown, that the model has some problems identifying the nu mbers on limitation signs (e.g. 80 km/h). Furthermore, completely different looking traffic signs (e.g. black background, white text with 100 km/h) are not classified correctly, si nce any relations to the originial, trained version (white background, black text) are not possible. It was very interesting to see the "correct" classification of the end -of-30 km/h - sign. Since this sign was unknown to the model, it showed, that its classification is near to End-of-80 km/h.