Traffic Sign Recognition

Data Set Summary & Exploration

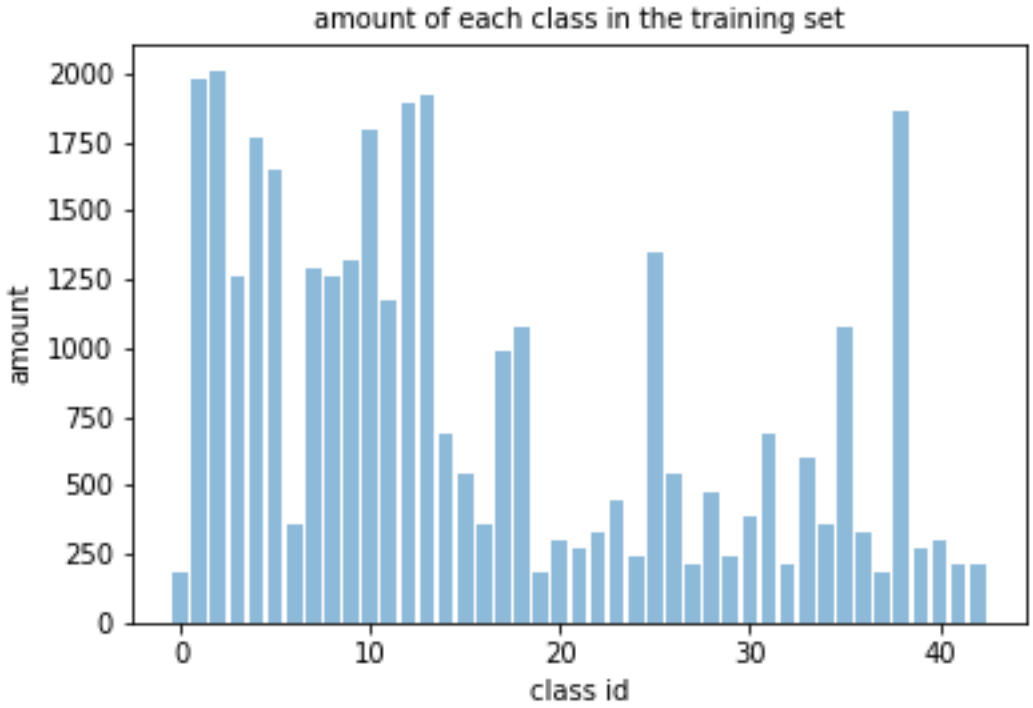
1. Dataset Summary

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

* The size of training set is: 34799 images
* The size of the validation set is 4410
* The size of test set is 12630 images
* The shape of a traffic sign image is 32*32 pixel
* The number of unique classes/labels in the data set is 43

2. Exploratory Visualization

Here is an exploratory visualization of the data set. The bar chart is showing how the data classes are contributed. The amount varies between 200 images for the 20 km/h sign and 2000 images for the 30 km/h sign. Also we see, that the classes in the dataset are not evenly distributed. It's hard to say, if this distribution of the sign represents the distribution on the street. But for the model, evenly distributed classes would be better, so that we can be sure, that we don't introduce a bias by the training sample.

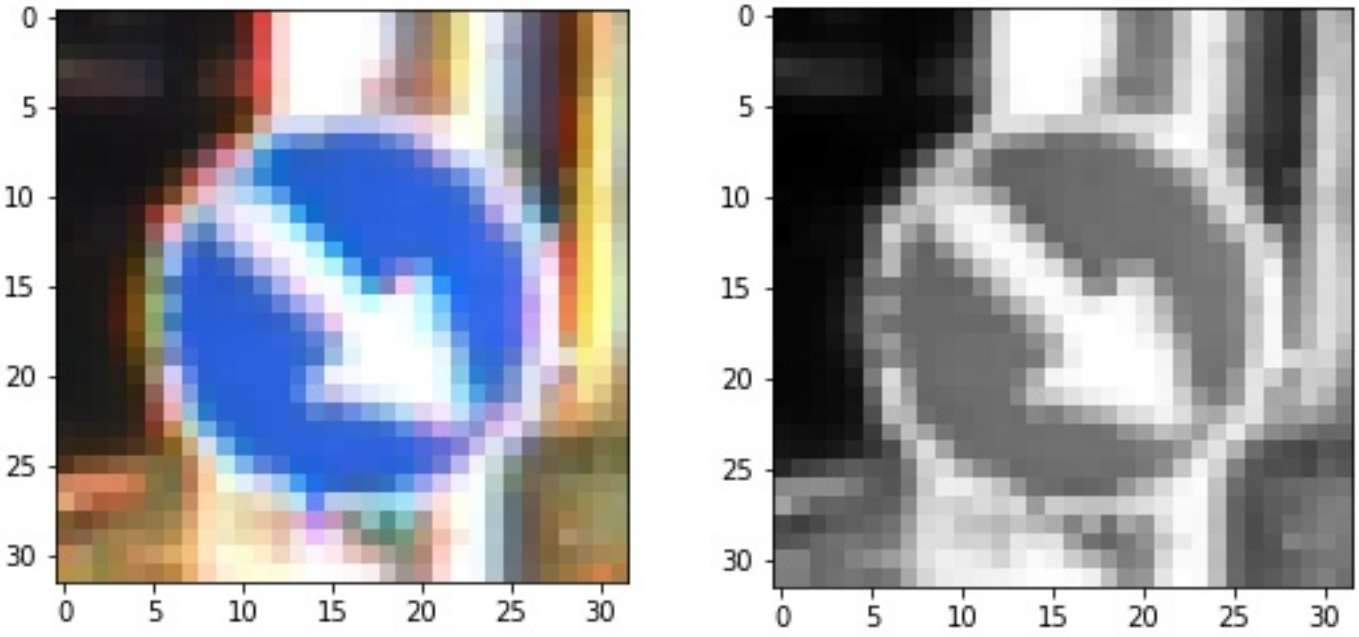


Design and Test a Model Architecture

1. Preprocessing

As a first step, I decided to convert the images to grayscale because on my personal laptop the training time will decrease by the factor of 3. For a bigger network, which I used, for training on the workspace GPU, I stay with the original colour depth of the image.

Here is an example of a traffic sign image before and after applying the gray scaling.



For every model, I normalized the image data to values between 1 and 0. The normalisation is necessary, since there will be “millions” of multiplications during the training of a model. The normalisation makes sure, that the weights for each node will not get to big, which will stop the training procedure. In my approach, I set the normalisation between 1 and 0 by x-min_value/ (max_value - min_value) for max_value = 255 and min_value = 0

I decided to generate additional data because the size of our training set is quite small. Therefor I used a ImageGenerator from the Keras library, where I only had to set specific arguments. I decided, that I want to randomly pick images and rotate them by max 10 % and randomly shift them by max. 10 % of the height or width. Those data are generated on the fly during the training process, so I can’t show before and after images here.

2. Model Architecture

My final model consisted of the following layers (Keras output, just for this summary):

For the baseline model for prototyping I used this LeNet-architecture:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 6)         456
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_3 (Dense)              (None, 43)                3655
=================================================================
Total params: 64,811
Trainable params: 64,811
Non-trainable params: 0
_________________________________________________________________
```

It consists of 5 layers, 2 convolutional layers and 3 fully connected layers and has in total 64,811 parameters. I included one Dropout-Layer, where the dropout is set to 1.0 during the prototyping. The input shape is a 32x32x1 image, where a 5x5 filter with valid padding is applied with 6 filters for the convolutional layer. Afterwards I uses a 2x2 Max Pooling, followed by another convolutional layer with 16 5x5 filters and valid padding. After another Max Pooling the output shape of the image is 5x5x16. Afterwards, those output is reshaped to an 1d array and fully connected to a layer 120 nodes. Afterwards two fully connected layers follows with 80 nodes and ,as the number of image classes, 43 nodes. The activation function between the layers are expect the last one, a relu function. For the last layer it's a softmax function.

Here is my simplified AlexNet-architecture. I had to adapt parameters, since the raw image shape is bigger in the original version i than in this example.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	2432
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 32)	0
dropout_1 (Dropout)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	51264
max_pooling2d_2 (MaxPooling2	(None, 8, 8, 64)	0
dropout_2 (Dropout)	(None, 8, 8, 64)	0
conv2d_3 (Conv2D)	(None, 8, 8, 96)	55392
conv2d_4 (Conv2D)	(None, 8, 8, 96)	83040
conv2d_5 (Conv2D)	(None, 8, 8, 64)	55360
flatten_1 (Flatten)	(None, 4096)	0
dense_1 (Dense)	(None, 1024)	4195328
dropout_3 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 128)	131200
dropout_4 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 43)	5547

Total params: 4,579,563
Trainable params: 4,579,563
Non-trainable params: 0

This architecture consists of 8 layers with 5 convolutional layers and 3 fully connected layers and has in total 4,579,563 trainable parameters. I included several Dropout-Layer, where the dropout is set to 0.5 during the training. The input shape is a 32x32x3 image, where a 32 5x5 filter with same padding is applied for the convolutional layer. Afterwards I uses a 2x2 Max Pooling, followed by another convolutional layer with 64 5x5 filters and same padding. After another Max Pooling the output shape of the image is 8x8x64. Here I follow the AlexNet approach and add 3 convolutional layers behind each other with 3x3 filters and same padding. The number of filters vary between 96 and 64. Afterwards I flatten the layer and use a fully connected layer with 1024 nodes. Then there are two fully connected layers following with 128 nodes and ,as the number of image classes, 43 nodes. The activation function between the layers are expect the last one, a relu function. For the last layer it’s a softmax function.

3. Model Training

My results from the first dry run on my laptop with the LeNet-Architecture were quite satisfying, so I didn't spend much time here. Also my first run on the AlexNet-architecture was also quite good, so I really tweaked the epochs a bit.

For the model training, I used an AdamOptimizer with the standard initiation. The AdamOptimizer is some kind of standard approach, which works quite well for most of the time. Also I didn't want to tweak the optimiser parameters (e.g. momentum), since fine-tuning takes a lot of time and my model was already "good" enough. The most important parameter is anyway the learning_rate. Since everything worked from the beginning, I decided to keep the learning rate with 0.001. Also I didn't touched the batch_size of 128. I set up my epochs to 10, but with my ImageGenerator, which runs twice over the initial images, it's 20 epochs in total.

Since I trained every model on my laptop (yes, also AlexNet - sic!), I used DropOff of 1.0; on the GPU I reduced it to 0.5 and used more epochs. Since the computation is quite expensive and I could run them only over Night + work time, I took the risk for a potential overfitting.

4. Solution Approach

My final model results were:
* training set accuracy of 0.999
* validation set accuracy of 0.942
* test set accuracy of 0.949

For building the model, prototyping and testing I used the LeNet from the Udacity course. Therefor I also transformed the images to grey scale. After very short hyperparamter tuning I reached a validation set of 0.924 and a test set accuracy of 0.916. Instead of going more into details for fine-tuning, I decided to build a bigger network. I choose AlexNet architecture, because it's the smallest one of the well-known architectures (VGG, GoogLeNet, ResNet). Also I was curious about AlexNet, since this was the architecture, which started the Deep Learning hype and I didn't know much about it.

My AlexNet adaption (the images are smaller in the traffic sign dataset) was tuned on the GPU on the workspace. But I didn't figure out, how to download my model from the workspace, so I was retraining the model on my machine. One change here is the dropoff, which I set to 1.0 on my machine, because of limited calculation resources.

One of my concern is overfitting, because I reached quite fast 99,99 % accuracy on the training dataset. So I was optimising the model on the validation dataset, which is definitely a bad practice. In the next evaluation, I would use a cross-validation approach for the training. Also I will use my dropoff-layers properly. In the end, I used only 10 epochs, where the validation accuracy was quite stable. So I decided, that overfitting is in that state only a small issue and run it on the test set with an accuracy of 94,9 %.

Test a Model on New Images

1. Acquiring New Images

I recorded traffic data from my German hometown one month ago. At that time I put a GoPro behind the windshield and was driving around. I extracted 7 traffic signs out of my recording.

Here are seven German traffic signs:




The first image might be difficult to classify because it's not recorded directly from the front, but more from the side. So there is a distortion into it. The 2nd one, is not hard to detect. The 3rd one, has some stains on it and a slight rotation, so that there could be some problems. The 4th is difficult, because there are a lot of stickers on it (typical Berlin "problem"). For 5th - 7th, I just picked 3 signs for 30 km/h. Here I see the problem in the recognition of the digit. So, will the model detect the 30 or will it go for 20?

As a future step, if I have a gpu and more time, I will try to implement a segmentation run (or figure out how to train yolo3) and run it against my traffic video.

2. Performance on New Images

My examples, where not perfect. One image couldn't classified correct, so I got 85.7 % on my final model right. I will go a bit more into detail for the wrong classification in the next subchapter. Interestingly one my smaller LeNet, I got 100 % correct.

Here are the results of the prediction:

Image	Predition
Stop sign	Stop sign
Yield sign	Yield sign
Ahead only	Ahead only
Keep right	Keep right
Speed limit (30km/h)	Speed limit (30km/h)
Speed limit (30km/h)	End of no passing by vehicles over 3.5 metric tons
Speed limit (30km/h)	Speed limit (30km/h)

3. Model Certainty - Softmax Probabilities

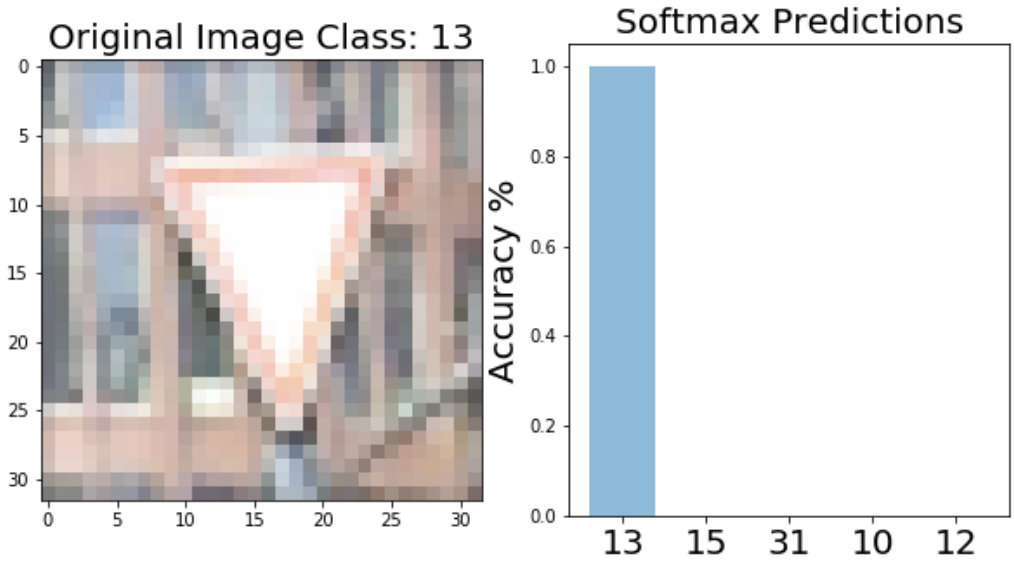
The code for making predictions on my final model is located in the 26th cell of the Ipython notebook.

For the first image, the model is relatively sure that this is a stop sign (probability of 0.991), and the image does contain a stop sign. The top five soft max probabilities were distributed according to this table:

Predition	Probability
Stop sign	9.91487861e-01
Speed limit (20km/h)	4.56377398e-03
Priority road	3.57533153e-03
No entry	1.01205595e-04
Bicycles crossing	9.82292549e-05

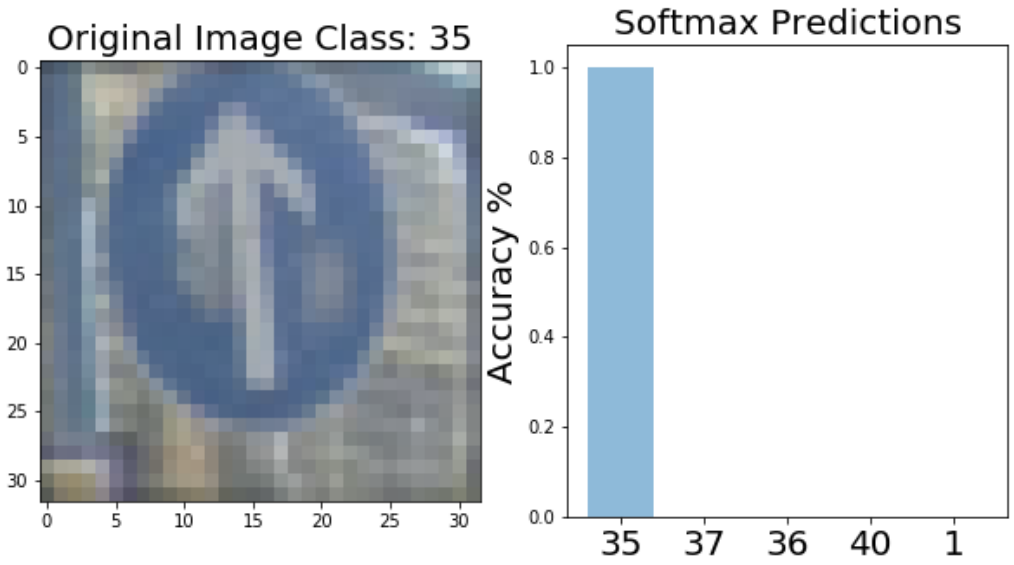
For the second image, the model is relatively sure that this is a yield sign (probability of 0.999), and the image does contain a yield sign. The top five soft max probabilities were distributed according to this table:

Predition	**Probability**
Yield sign	9.99999881e-01
No vehicles	1.08072186e-07
Wild animals crossing	1.09797895e-08
No passing for vehicles over 3.5 metric tons	3.97224964e-09
Priority road	2.95474034e-10



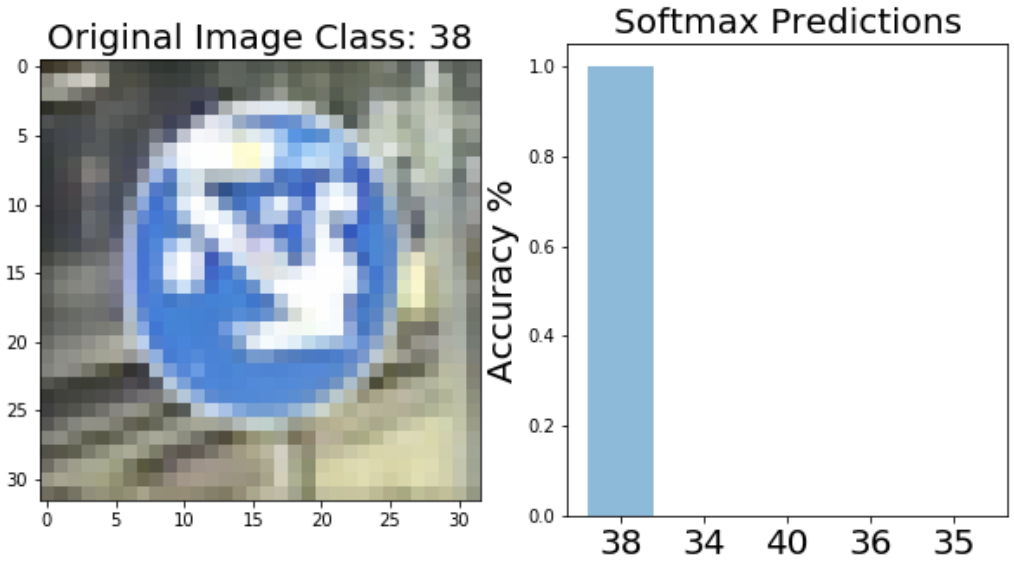
For the third image, the model is relatively sure that this is an ahead only sign (probability of 0.999), and the image does contain an ahead only sign. The top five soft max probabilities were distributed according to this table:

Predition	**Probability**
Ahead only	9.99949336e-01
Go straight or left	4.45825244e-05
Go straight or right	2.90473758e-06
Roundabout mandatory	2.88680371e-06
Speed limit (30km/h)	1.13499262e-07



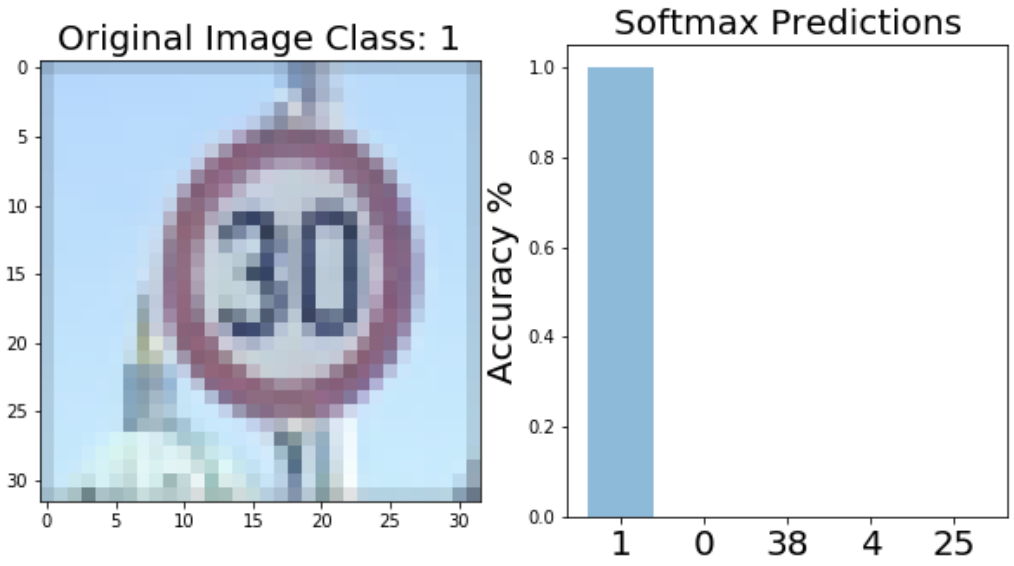
For the forth image, the model is relatively sure that this is a keep right sign (probability of 1.000), and the image does contain a keep right sign. The top five soft max probabilities were distributed according to this table:

Predition	**Probability**
Keep right	1.00000000e+00,
Turn left ahead	1.21193350e-11
Roundabout mandatory	3.40903282e-12
Go straight or right	6.35889453e-13
Ahead only	1.51280169e-15



For the fifth image, the model is relatively sure that this is a speed limit (30km/h) sign (probability of 0.999), and the image does contain a speed limit (30km/h). The top five soft max probabilities were distributed according to this table:

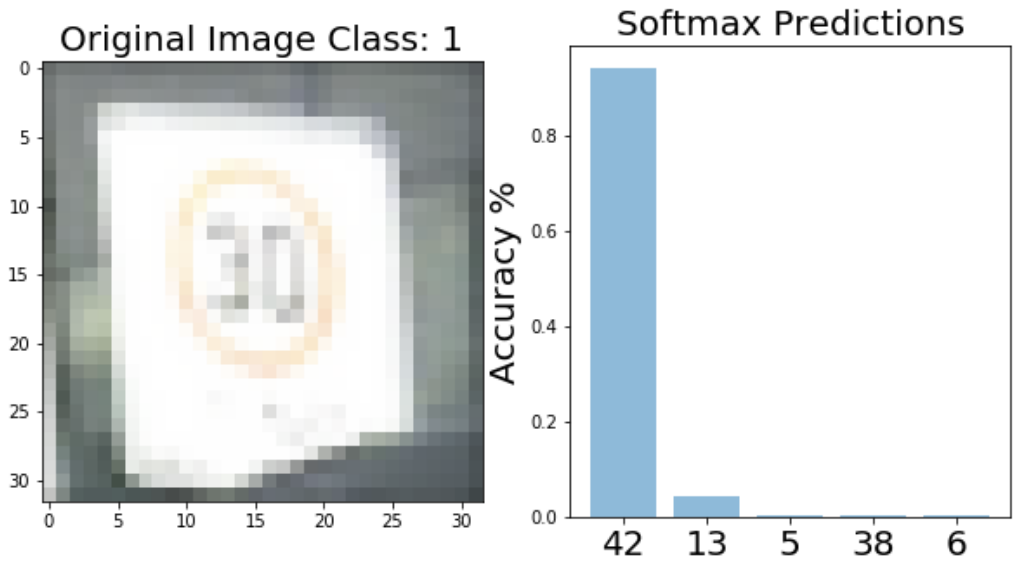
Predition	**Probability**
Speed limit (30km/h)	9.99999881e-01
Speed limit (20km/h)	8.04459646e-08
Keep right	7.24743154e-09
Speed limit (70km/h)	7.48376014e-11
Road work	2.13726554e-11



For the sixth image, the model is relatively sure that this is a end of no passing by vehicles over 3.5 metric tons sign (probability of 0.943), and the image does contain a speed limit (30km/h). So the model was totally wrong with this example. This 30km/h sign is a bit different to the others, so I assume, it's not part of the training set. Also this image is quite bright, so that probably the red circle and the digits aren't detected well.

The top five soft max probabilities were distributed according to this table:

Predition	**Probability**
End of no passing by vehicles over 3.5 metric tons	9.43065822e-01
Yield	4.37621288e-02
Speed limit (80km/h)	3.92386317e-03
Keep right	3.75657273e-03
End of speed limit (80km/h)	2.78717326e-03



For the seventh image, the model is relatively sure that this is a speed limit (30km/h) sign (probability of 0.999), and the image does contain a speed limit (30km/h). The top five soft max probabilities were distributed according to this table:

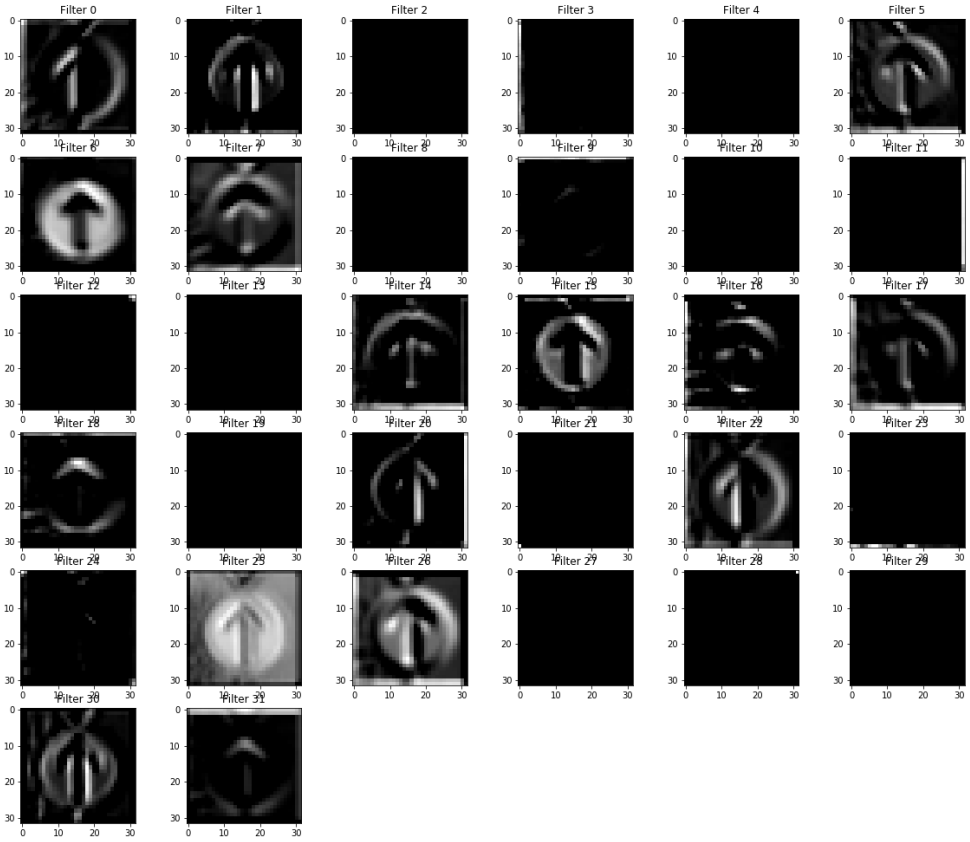
Predition	**Probability**
Speed limit (30km/h)	9.99997139e-01
Keep right	2.85261103e-06,
Speed limit (20km/h)	2.40170159e-08
Speed limit (80km/h)	3.29253025e-09
Road work	2.04464379e-09

1. Discuss the visual output of your trained network's feature maps. What characteristics did the neural network use to make classifications?

In my example for the feature maps, I used the go ahead sign from the beginning. Since I have 5 convolutional layers in my model, I describe only the layers 1, 2 and 5.

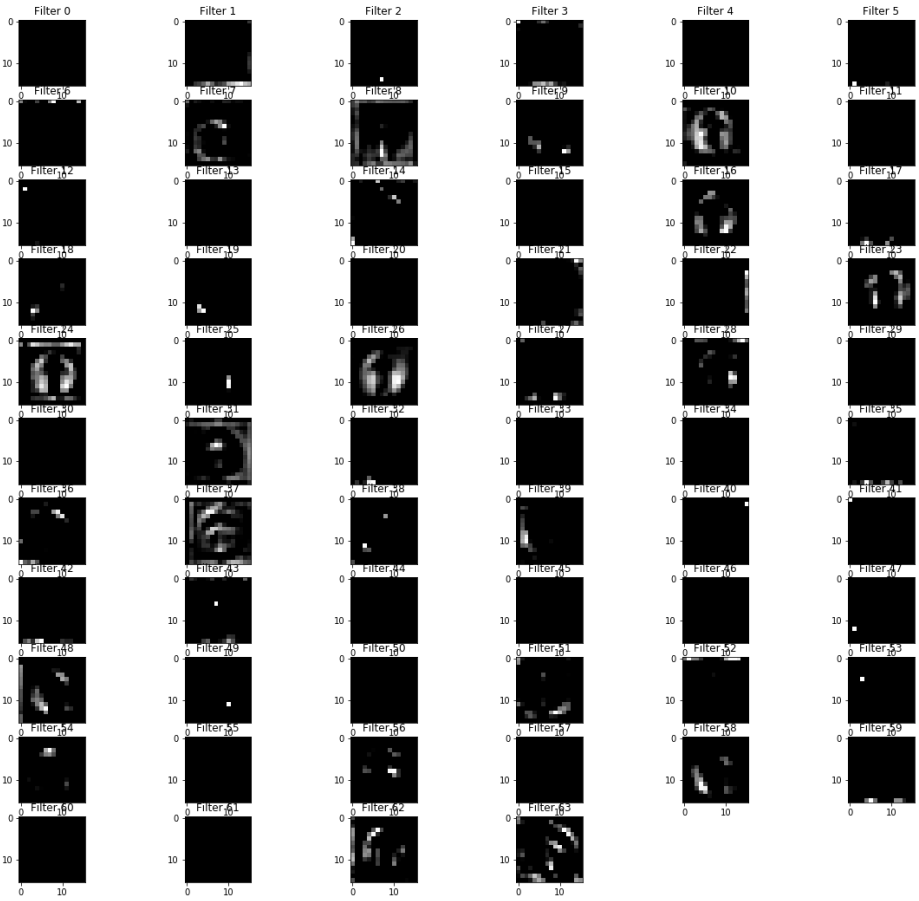
Layer 1:

On layer 1 are 32 filters. Half of them are totally black. Here we need several images to see, if this filter doesn't apply or the nodes died. Nodes can die (all weights are zero) through the training, since there are a lot of multiplications with numbers < 1. E.g. there is a "natural" tendency in a neural network, that the nodes die. Besides that, the other 16 filter are quite impressive. Each of those 16 filter is different. E.g. Filter 7 could be filtering the colour blue, while filter 2 looks a bit like the sobalx filter from the previous courses. In this layer, the filter are still looking like the sign. In the next layer, the shape of the sign and arrow will more and more disappear. All in all, the working filter are quite unique and it would be impossible to figure out the filter weights manual.

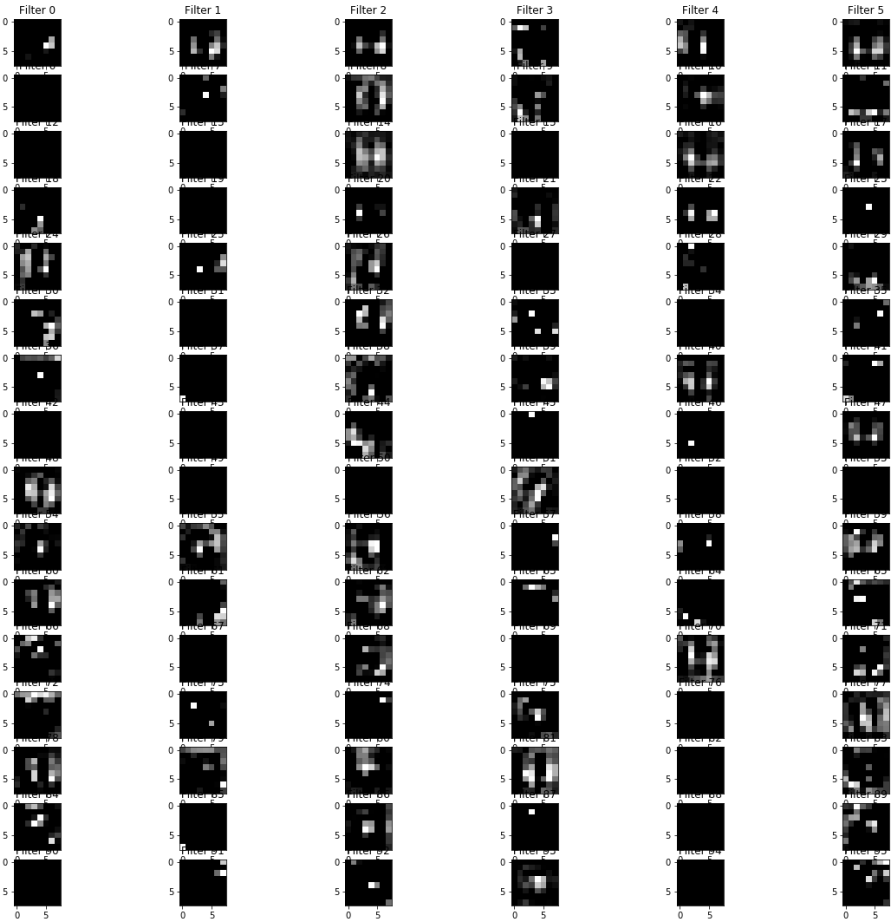


Layer 2:

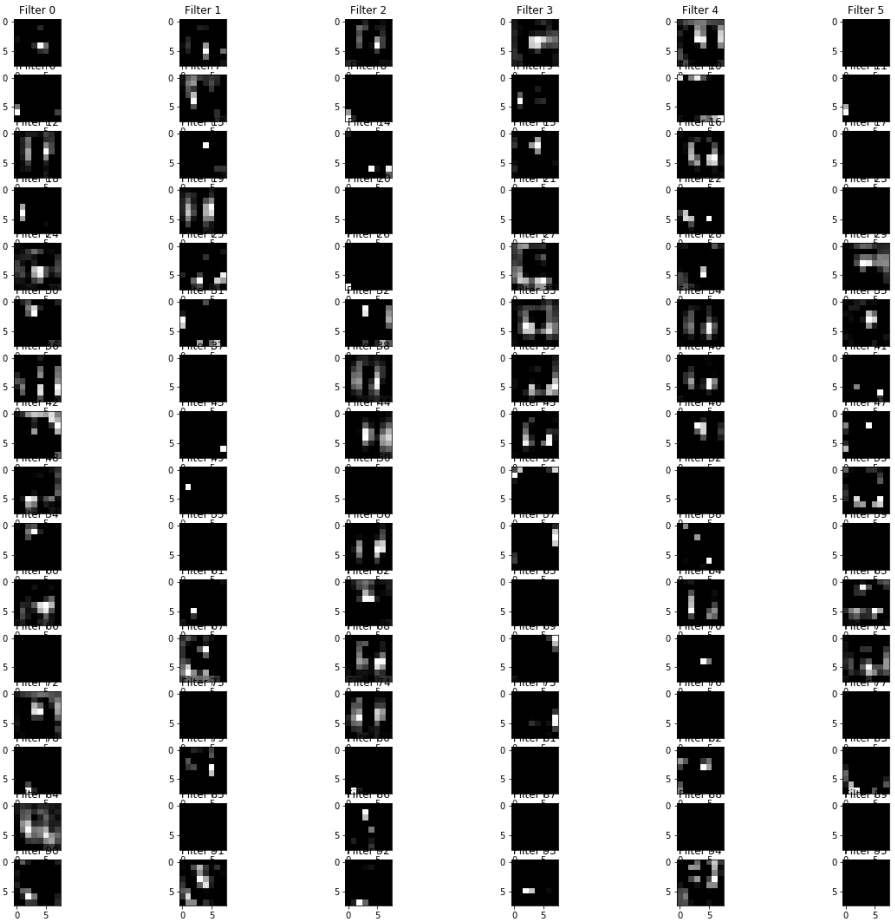
In the 2nd layer I use 64 filters. The filters got more abstract, but some of them still show some shape of the round sign or the arrow. Also there are some of them, which are only black. Other ones are quite abstract and it's not clear, what they are "see" e.g. image 64.



Layer 3:



Layer 4:



Layer 5:

In layer 5 I use 64 filters and the output is quite abstract. It's quite hard to imagine, what e.g Filter 0 with the left dot sees. For other filters, I can image, that it sees an arrow or more common, the blue space of the sign. But overall, it's not obvious, what information each of the filters represent.