Traffic Sign Classifier: Reflection

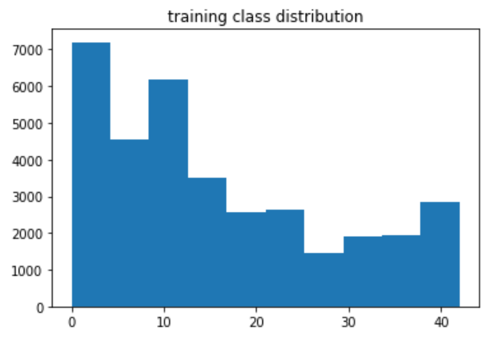
1. Summary of the data set

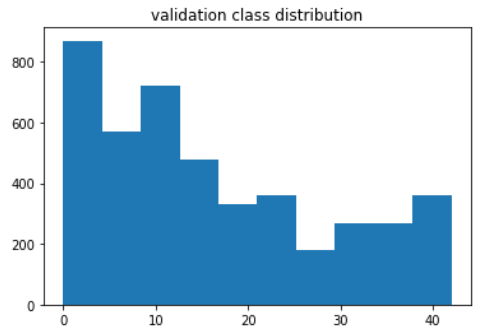
I used the standard python functions “len” and “shape” to calculate summary statistics of the traffic signs data set:

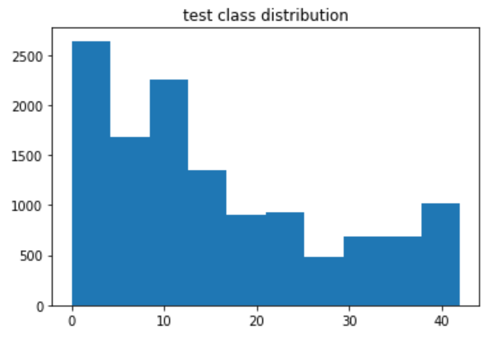
```
Number of training examples = 34799
Number of validation examples = 4410
Number of testing examples = 12630
Image data shape = (32, 32, 1)
Number of classes = 43
```

2. Exploratory visualization of the dataset.

Here is an exploratory visualization of the data set. It is a bar chart showing how the data is distributed over different classes:







As can be seen from figures above the data over all sets are unequally distributed over different class, for example classes 0 – 10 have significantly more examples than the classes 20 – 30.

3. Preprocessing image data

I used normalization and conversion to grey scale for preprocessing.

Normalization is done by subtracting the value of 128 from each pixel and divided by 128. Normalization is important because neural networks work better when inputs are in the same scale. I tried both version without and with normalization and normalization provided higher accuracy.

I also tried classification with and without conversion to the grey scale and it turned out that conversion to the grey scale brings more than 2 % increase in accuracy. It seems that colour is not a significant feature for classifying traffic signs, probably because different lightening conditions and/or bad image quality could sometimes lead to changes in traffic signs colour and confuse the classifier.

4. Final model architecture

My final model is a well-known LeNet – 5, a convolutional neural network used successfully in digits recognition and other tasks as depicted in the figure below. The only difference to the "classical" LeNet is that there are 43 outputs for 43 traffic signs instead of 10 for digits and dropout.

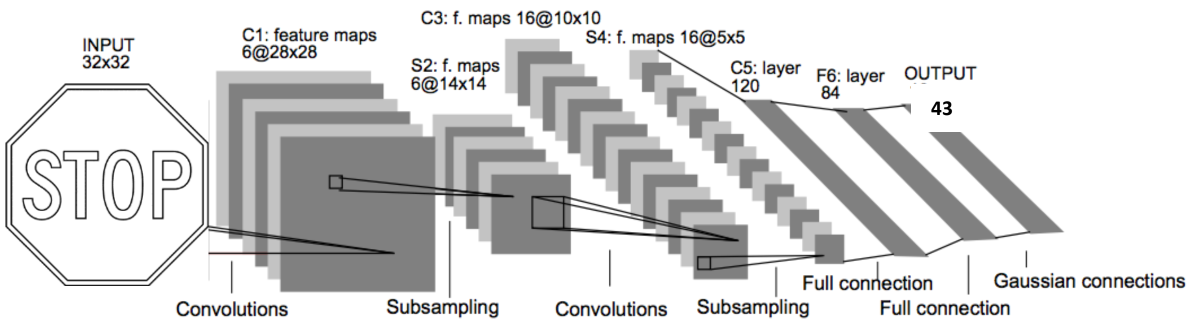


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

I tried different variation of the net above like decreasing/increasing the number of layers, changing filter size in convolutional layers and number of neurons in fully connected layer etc. but it didn't bring much improvement. Only dropout for the architecture above brought desired accuracy.

My final model consisted of the following layers:

Input : 32x32x1 Gray image

Convolution: 5x5 filter, 1x1 stride, no padding, output 28x28x6

RELU

Max pooling Layer: (Subsampling) 2x2 kernel, 2x2 stride, output 14x14x6

Convolution Layer: 5x5 filter, 1x1 stride, no padding, output 10x10x16

RELU

Max pooling: (Subsampling) 2x2 kernel , 2x2 stride, output 5x5x16

Fully connected: Input: 400, Output: 120

Fully connected: Input: 120, Output: 84

Fully connected: Input: 84, Output: 43

Softmax layer is used in a later stage to evaluate probabilities of new traffic signs.

5. Model Training

To train the model, I used the following hyperparameters:

- **Learning rate** = 0.001
- **Cost function:** Cross entropy
- **Optimizer**: Adam optimizer
- **Number of epochs**: 25
- **Mini batch size**: 128

The discussion can include the type of optimizer, the batch size, number of epochs and any hyperparameters such as learning rate.

Validation set error is evaluated after each epoch and test set accuracy is evaluated at the training end (after 25 epochs).

6. Finding required accuracy

To find required accuracy I started with original LeNet architecture and RGB images and look at the training set and validation set accuracy.

The initial accuracy of the training set was about 98% and of validation set 89% (with 10 trainings epochs). This was clear indication of **overfitting**, since training set accuracy was much higher than validation set accuracy.

In order to reduce overfitting and improve accuracy I tried different strategies in an iterative manner:

- Reduced the number of neurons and layers - > this brought only 1-2 % improvement.
- **Dropout**: this was the **main contribution for achieving the required accuracy**. I tried different dropout rates 50%, 20%, 15% and 10% at it turned out that (at least for about 10 training epochs) the dropout rate of 15% i.e. "*keep_prob*" parameter set to 0.85 provided the best results of about 94% of validation set accuracy.
- I could achieve further accuracy improvement for about 2.5% by using grey instead of colour images.

My final model results were:

- training set accuracy of 99.76%
- validation set accuracy of 96.70%
- test set accuracy of 94.00%

High training set accuracy indicates that the original LeNet architecture is powerful enough i.e. has enough parameters (weights) to approximate desired function well (mapping for input images to traffic sign classes).

This should be no surprise, since the LeNet was developed for image recognition and in all image recognition tasks we have similar hierarchical features like edges, shapes, figures etc. that could be extracted using convolutional networks.

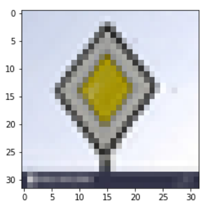
So the model complexity was high enough (indicated by low bias) and it was up to regularization to reduce the overfitting. It turned out that introducing dropout provided required accuracy.

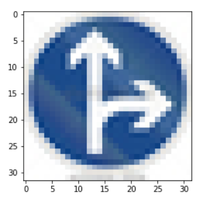
Since test set accuracy is still more than 5% lower than training set accuracy and validation set accuracy 3% lower than training set accuracy, there is further potential in reducing overfitting and improving accuracy by:

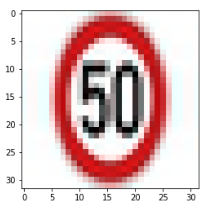
- Further regularizations like introduction of weight penalties in the cost function
- Better image pre-processing using different filters and other algorithms
- Enlarging the training sets by obtaining new traffic signs images and/or augmented existing ones

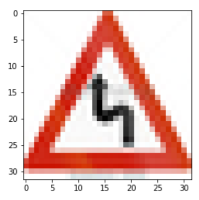
7. Testing Model on New Images

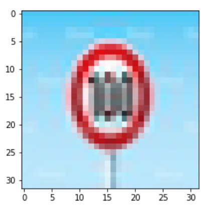
Here are five German traffic signs that I found on the web:











The speed limit images (50 km/h and 100 km/h) might be difficult to classify because the numbers are not clearly distinguishable from each other.

Here are the results of the prediction:

Image	Prediction
Priority road	Priority road
Speed limit (50km/h)	Speed limit (20km/h)
Speed limit (100km/h)	Speed limit (100km/h)
Go straight or right	Go straight or right
Double curve	Double curve

The model was able to correctly guess 4 of the 5 traffic signs, which gives an accuracy of 80%. This was expected according to the accuracy on the test set of 94%. Only the traffic sign “Speed limit 50 km/h” was incorrectly classified, since the numbers were not clearly distinguishable in the image, for example number 5 can be easily misinterpreted as 2 or 3.

8. Softmax Probabilities

The following table provides 5 highest softmax probabilities for each traffic sign:

Image	Prediction	Softmax Probabilities [%]
Priority road	Priority road	99.4, 0.47, 0.10, 0.03, 0.002
Speed limit (50km/h)	Speed limit (20km/h)	47.68, 25.35, 24.91, 1.63, 0.25
Speed limit (100km/h)	Speed limit (100km/h)	94.39, 4.63, 0.69, 0.24, 0.02
Go straight or right	Go straight or right	99.05, 0.36, 0.26, 0.20, 0.04
Double curve	Double curve	93.10, 2.75, 2.53, 0.50, 0.04

As can be seen from table above, for the correctly classified traffic signs softmax probabilities are above 90%.

In the case of incorrectly classified sign: “Speed limit 50 km/h”, all softmax probabilities are below 50%. The highest probability was 47.68% for the traffic sign “Speed limit 20 km/h”. The second highest probability was 25.35% for the traffic sign “Speed limit 50 km/h” which would be correct classification. The third highest probability was 24.91% for the traffic sign “Speed limit 30 km/h”. As stated above, the speed limit traffic signs might be difficult to classify because the single numbers are not clearly distinguishable. In the above example, it is clear from softmax probabilities that the image was recognized as speed limit traffic sign: 4 most probable classifications are speed limits signs: 20 km/h, 50 km/h, 30 km/h and 70 km/h, which sums up to more than 99% softmax probability. But what exactly was the speed limit was more difficult to estimate. For example number 5 can be easily misinterpreted as 2 or 3, as indicated by softmax probabilities. Maybe some better image preprocessing would provide more clear images that could be easier to classify. Or some kind of hierarchical classification would help: At first the traffic sign might be classified as a “speed limit” sign, and then by the next classifier should be evaluated what is exactly the speed limit.

9. Visualizing the Neural Network's State with Test Images

Network state visualisation was not possible because variables like “conv1” and “conv2” were not visible outside of the LeNet function. Maybe I should define them as global variables or there are some other ways to access them after restoring the model?