Traffic Sign Recognition Project

The goals / steps of this project are the following:

- Loading the data set (see below for links to the project data set)
- Exploring, summarizing and visualizing the data set
- Designing, training and testing a model architecture
- Using the model to make predictions on new images
- Analyzing the softmax probabilities of the new images
- Summarizing the results with a written report

Submission Files

This submission consists of two files:

- A writeup file detailing how the different points of the rubric were addressed (the present file).
- The project code.

Dataset Exploration

Dataset Summary

The code for the data summary is contained in the second cell of the IPython notebook. I used the numpy library to calculate summary statistics of the traffic signs data set. Here are the results:

- Size of training set: **34799**
- Size of test set: **12630**
- Shape of a traffic sign image: **(32,32,3)**
- Number of unique classes/labels in the data set: **43**

Exploratory Visualization

The code for the exploratory visualization of the dataset is contained in the third cell of the IPython notebook. The code produces a bar chart showing the distribution of the different labels/classes within the train, test and validation sets (Fig.1).

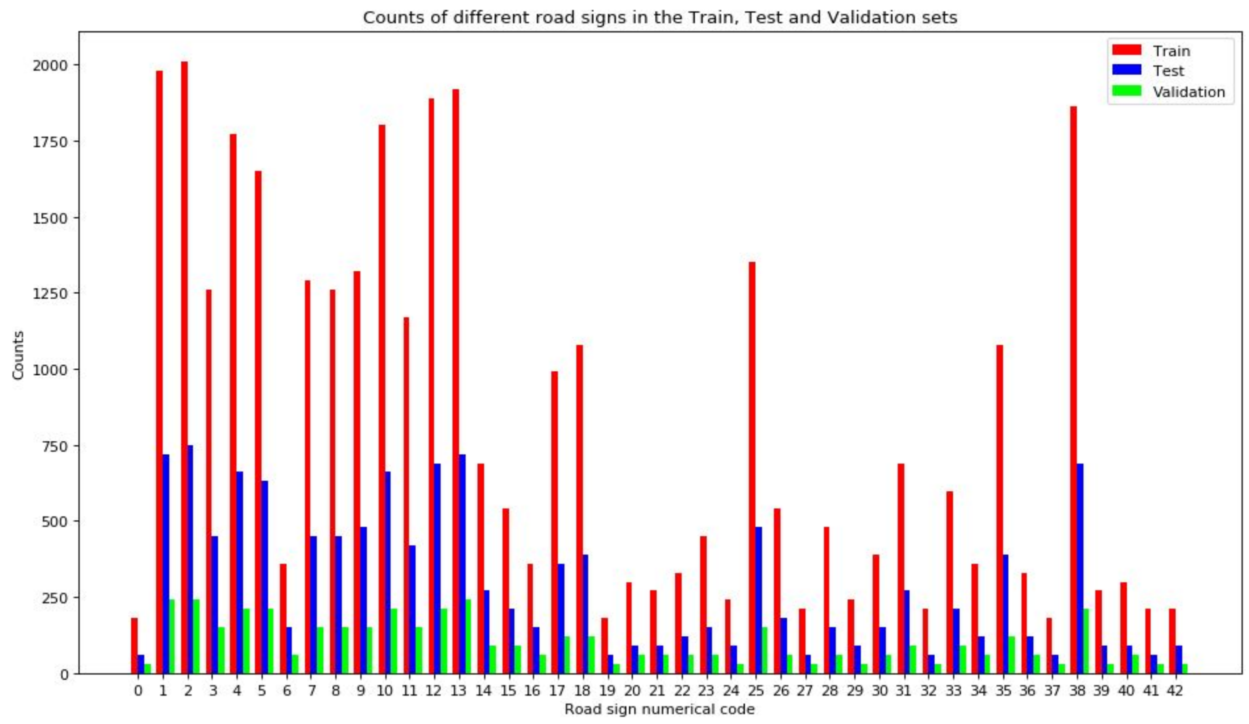


Figure 1 - Distribution of the different labels/classes within the train, test and validation sets.

The bar-chart shows two main aspects of the data:

- The proportion of the different classes is kept across the different sets. The distribution for the train, test and validation sets look the same, apart from a scaling factor.
- The distributions are skewed. In other terms the instances of some classes are far more numerous than those of other classes.

The fact that the distributions are skewed can lead our classification model to perform badly with respect to the less represented classes, even if the overall accuracy is high. This issue has been addressed and described in the next session.

Design and Test a Model Architecture

Preprocessing

The code for this step is contained in the fourth code cell of the IPython notebook.

As a first step, I decided to convert the images to grayscale. This was done in the attempt to minimize spurious shade/hue differences due to factors such as the age and conditions of the traffic signs (e.g. whether they are discoloured or worn), how the light is reflected back and ambient light conditions.

Secondly I normalized the grayed pictures in an attempt to compensate for different light conditions, such as dark and bright.

Here is an example of a traffic sign image before and after grayscaling and normalization.

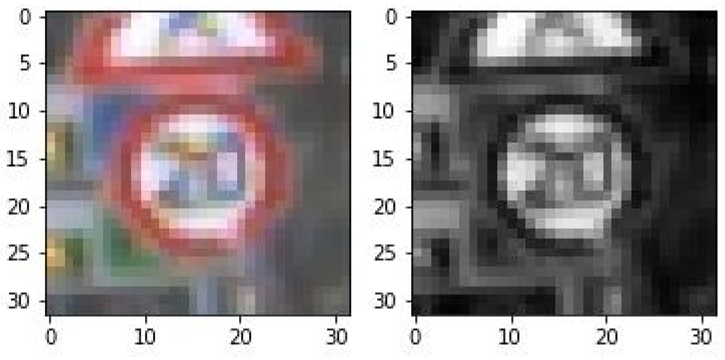


Figure 2 - An image from the training set before (left) and after (right) grayscaling and normalization.

As mentioned in the previous section, an imbalanced training set can lead the classifier to perform badly with respect to the less represented classes, even if the overall accuracy is high. To minimize this problem I have "augmented" the training dataset by generating additional data samples. Such additional images were created by applying affine transformation s to the images already present in the dataset. At the end of the data augmentation each class was represented by the same number of occurrences.

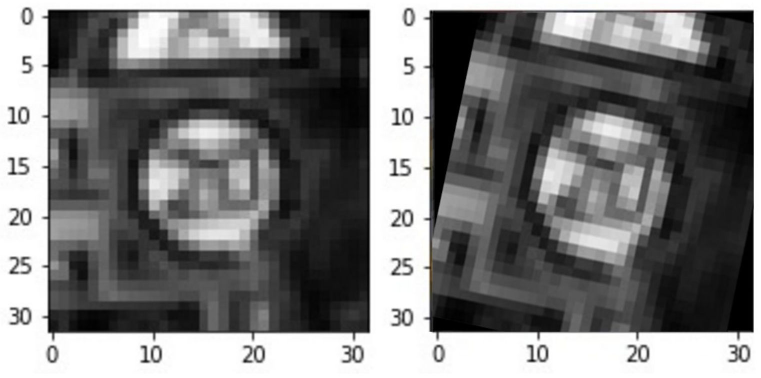


Figure 3 - An image from the training set before (left) and after (right) affinity transformations are applied.

Model Architecture

The code for my final model is located in the fifth cell of the ipython notebook.

My starting point for the convolutional neural network (CNN) used in this project was LeNet. LetNet had previously given good results at recognising elements in the MINST dataset. However its architecture showed its limitations when tried on the german traffic signs dataset, where the performance was not as good as requested. To allow the CNN to capture the wider variety of pictorial elements in traffic signs images, I decided to increase the number of filters in the first two convolutional layers of the original architecture and add a third convolutional layer. Another relevant change to the original LeNet architecture consisted of replacing the maximal pooling layers with average pooling layers. By doing so I aimed to minimize the loss of information while reducing the dimensionality of the input.

My final model consists of the layers listed in the table below:

Layer	**Description**
Input	32x32x1 (Gray-scale image)
Convolution Layer	Kernel: 5x5 Stride: 1x1 Padding: ‘VALID’ Output: 28x28x15
Activation Layer	ReLUs
Average pooling	Kernel: 2x2 Stride: 2x2 Padding = ‘VALID’ Output: 14x14x15
Convolution Layer	Kernel: 5x5 Stride: 1x1 stride Padding: ‘VALID’ Outputs 10x10x30
Activation Layer	ReLUs
Convolution Layer	Kernel: 5x5 Stride: 1x1 stride Padding: ‘VALID’ Outputs 6x6x60
Activation Layer	ReLUs
Average pooling	Kernel: 2x2 Stride: 2x2 Padding = ‘VALID’ Output: 3x3x60
Fully connected	Input: 540 Output: 120
Fully connected	Input: 120 Output: 84
Fully connected	Input: 84 Output: 43

Tabke 1 - List of layers in the Convolutional Neural Network (CNN).

Fig.4 shows a graphical representation of the CNN implemented for this assignment.

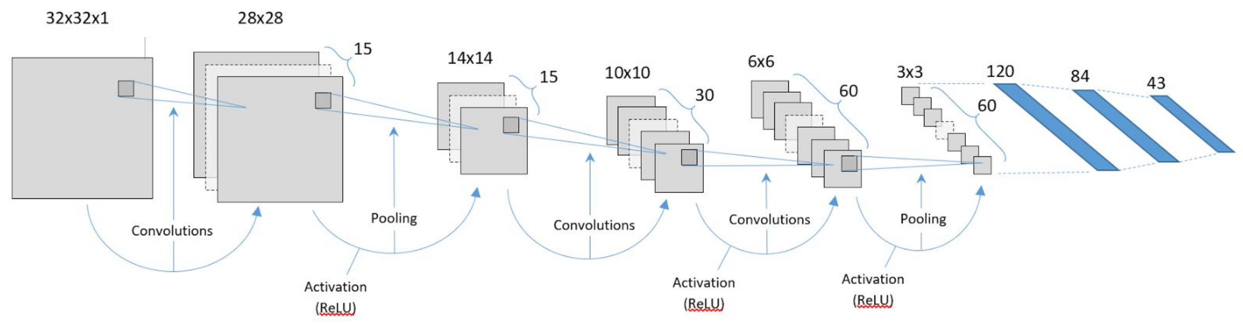


Figure 4 - Scheme of the convolutional neural network used to classify german traffic sign.

Model Training

The code for training the model is located in the seventh, eighth and ninth cells of the IPython notebook.

To train the model, I used the following:

Optimizer:	**AdamOptimizer**
Learning rate:	**0.008**
Batch size:	**1024**
Number of epochs:	**10**

By increasing the batch size from 128 to 1024 I aimed to make the model converging more quickly by reducing its wandering in the parameter landscape. An increased Learning Rate (from 0.001 to 0.008) also resulted in a faster convergence while allowing the model to move away from local minima. In this regard it should be mentioned that 0.008 is the initial learning rate, as the Adam's algorithm (on which the AdamOptimizer is based) computes adaptive learning rates for each parameter.

Solution approach

Adjustment to the initial CNN architecture (LeNet) and the parameter values were made progressively and kept if they led to better performance on the validation set.

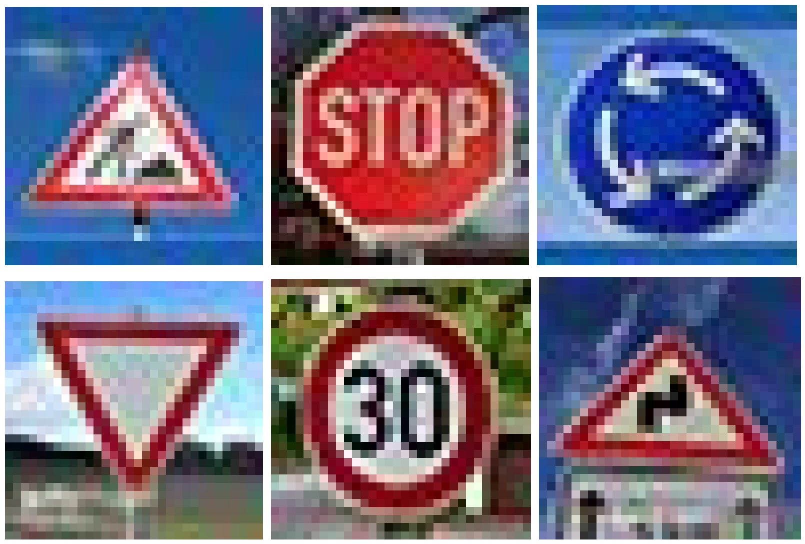
Such list of changes to the initial CNN architecture and parameter settings have been discussed in the previous sections. Here I report the accuracy of the trained model on the train, validation and test set:

- Training set accuracy: **0.938**
- Validation set accuracy: **0.898**
- Test set accuracy: **0.941**

Test a Model on New Images

Acquiring New Images

Below are six German traffic signs that I found on the web.



The pictures were preprocessed so as to make them readable by the CNN and consistent with the images used to train the model. In particular they were grayscaled and normalized.

Performance on New Images

The code for making predictions on these six new images is located in the thirteenth cell of the IPython notebook.

Here are the results of the prediction:

Image	**Prediction**
Road work	Road work
Stop	Stop
Roundabout mandatory	Roundabout mandatory
Yield	Yield
Speed limit (30km/h)	Speed limit (30km/h)
Double curve	Dangerous curve to the left

Table 2 - Comparison between the real and the predicted class for the new five traffic signs.

The model was able to correctly guess five out of the six new traffic signs, which gives an accuracy of 83%. Although the accuracy is lower than for the test set, one has to consider that statistical fluctuations play here a prominent role, given the very small number of samples.

Model Certainty - Softmax Probabilities

The code for the calculation of the softmax probabilities is located in the fourteenth cell of the Ipython notebook.

The probabilities are reported in the table below.

The model is certain about what traffic signs are represented by the first, second and fourth images. High probability are also associated with the third (0.84) and fifth (0.89) traffic signs, both classified correctly.

The model failed however to recognise the sixth traffic sign. The corresponding softmax probability is the lowest within this limited dataset, suggesting that the model is less confident at classifying this traffic sign compared to the other five.

Probability	Prediction	Correct Class Label
1.00	**Road work**	**25**
1.00	**Stop**	**14**
0.84	**Roundabout mandatory**	**40**
1.00	**Yield**	**13**
0.89	**Speed limit (30km/h)**	**1**
0.77	**Dangerous curve to the left**	**17**

Table 3 - Softmax probability calculated by the model on a set of six new traffic signs images. Green and red cells are associated with correct and incorrect predictions respectively.