Traffic Sign Recognition

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

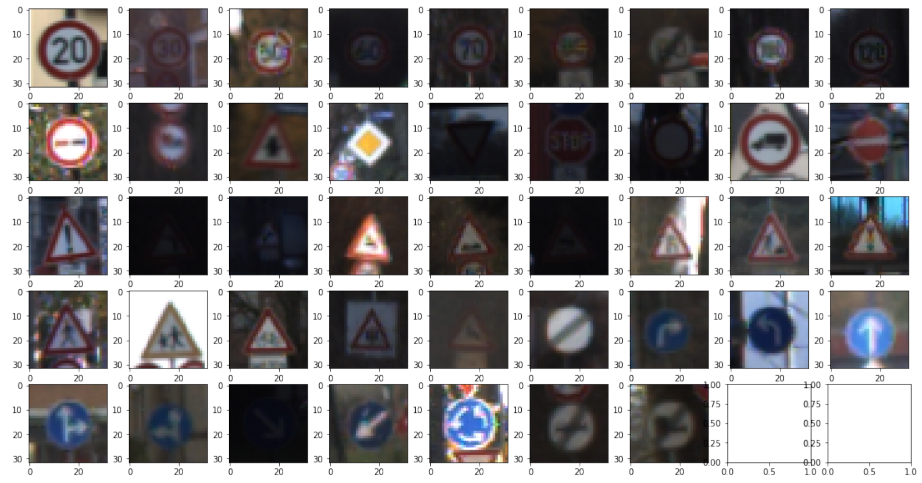


Figure 1: Traffic signs (43 different classes)

Step 1: Dataset Summary & Exploration

1. Basic summary of the data set

The dataset summary & exploration can be found beneath the caption *Step 1: Dataset Summary & Exploration* in the jupyter notebook.

The dataset is split into the three groups: Training set, validation set and testing set.

These are each given in the form of a pickle file, read in with the python pickle object and separate into features/images and labels.

It should be asserted that the length in the first part of the array, the number of images, is the same in the features array and in the labels array.

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

- The size of training set is 34799

- The size of the validation set is 4410
- The size of test set is 12630
- The shape of a traffic sign image is (32,32,3)
- The number of unique classes/labels in the data set is 43

2. Exploratory visualization of the dataset

Here is an exploratory visualization of the data set. It is a histogram showing the number of samples for each class:

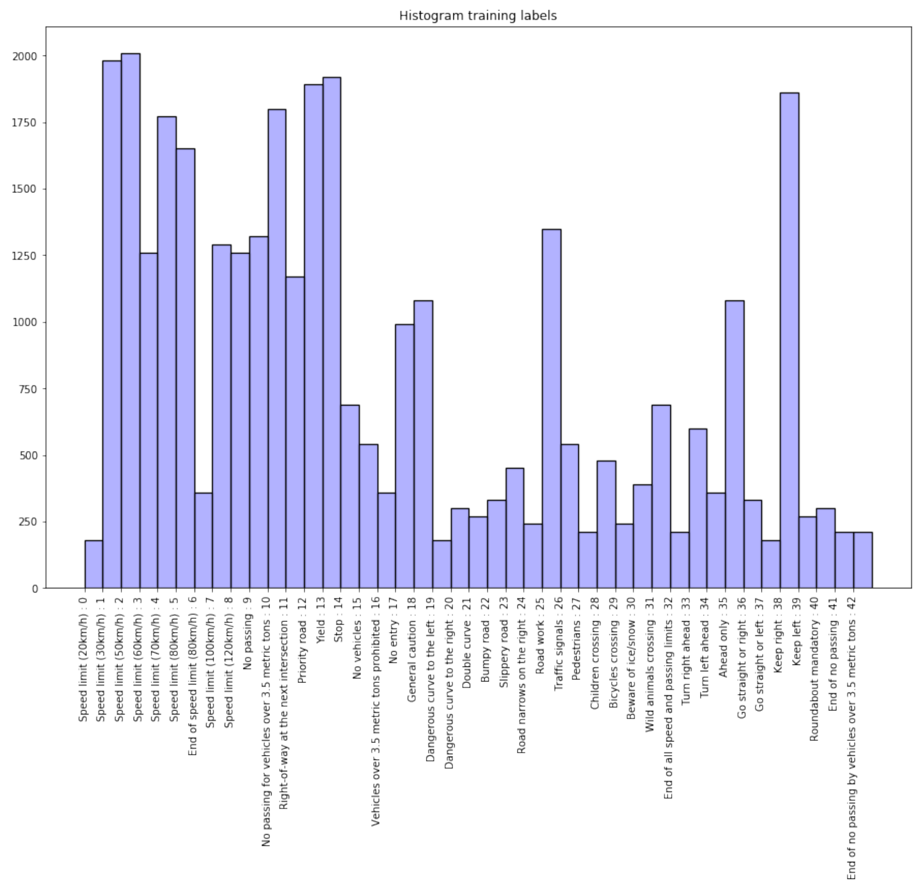


Figure 2: Histogram - training set

Five most occurring labels:

Amount	Description	Label number
2010.0	Speed limit (50km/h)	2.0
1980.0	Speed limit (30km/h)	1.0

Amount	Description	Label number
1920.0	Yield	13.0
1890.0	Priority road	12.0
1860.0	Keep right	38.0

Five least occurring labels:

Amount	Description	Label number
180.0	Speed limit (20km/h)	0.0
2010.0	Go straight or left	37.0
1980.0	Dangerous curve to the left	19.0
1920.0	End of all speed and passing limits	32.0
1890.0	Pedestrians	27.0

Mean of label distribution: 809.3

Median of label distribution: 540.0

Standard deviation of label distribution: 619.4

The histogram shows that there is a very big unbalance between the amount of training samples within the training set. This will have an effect for training the net obviously. Since it will be shown a lot more speed limit 50km/h signs than speed limit 20km/h signs, it will also adjust the weights more towards correctly classifying the 50km/h speed limit sign.

One solution for this problem is to generate additional pictures for the labels, occurring the least. The process of slightly changing existing pictures by blur, rotation, ranslation, brightness, warping, etc. is called augmentation.

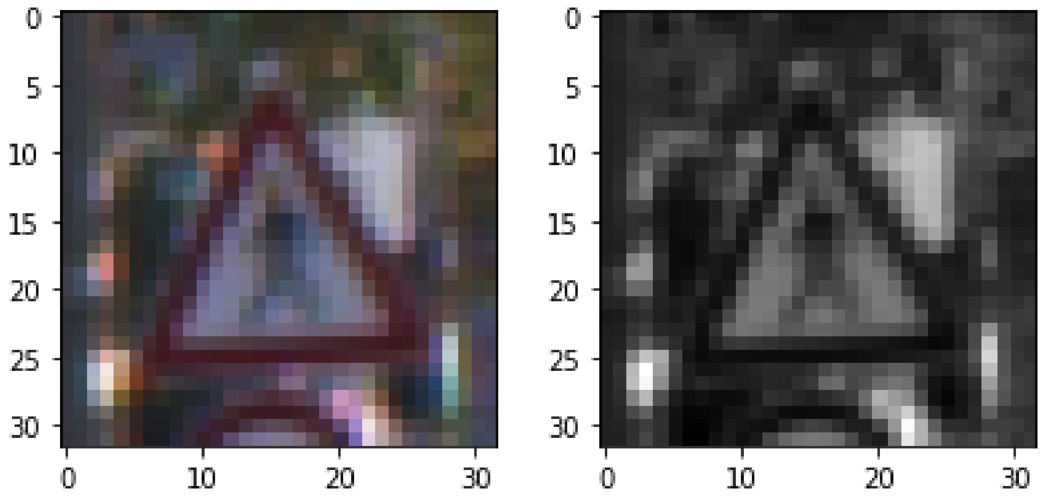
Design and Test a Model Architecture

1. Preprocessing

As a first step, I decided to convert the images to grayscale because it still holds most of the important information, while reducing the number of weights for the first convolutional layer significantly, thus improving the computational speed.

```python
# Grayscale
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])
```

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



As another step, I normalized the image data because it is much easier for a neural network to handle data, centers around zero with a given standard deviation, than handling data that has very big positive or negative values. A good way of explaining this, is to think about the activation functions. For example a sigmoid function has the highest gradient within a range of roughly -2 to +2. If the inputs are very high, for example +200 and +255, the activation function will be roughly 1 in both cases, meaning it does not make a difference.

The normalization can be done as follows:

```
X_train = (X_train.astype(np.int16) - 128) / 128
X_valid = (X_valid.astype(np.int16) - 128) / 128
X_test = (X_test.astype(np.int16) - 128) / 128
```

A pixel is usually read in as an unsigned int with 8 bit, to represent the necessary range from 0 to 255 for each of the RGB layers. Here we subtract 128 and divide by 128, to center the values around zero within the range of -1 to +1. Since an unsigned int with 8 bit can only show positive values, we need to change the type to a signed int. Here 16 bit are chosen, to for robustness of the pixels at the edge.

The training, validation and testing set are then re-assigned the new normalized values.

2. Model architecture

My final model consisted of the following layers:

Layer	Description
Input	32x32x1 RGB image

Layer	Description
Convolution 5x5	1x1 stride, valid padding, outputs 28x28x12
Relu	
Max pooling	2x2 stride, outputs 14x14x12
Convolution 5x5	1x1 stride, valid padding, outputs 10x10x32
Relu	
Max pooling	2x2 stride, outputs 5x5x32
Fully connected	Flatten to 5*5*32=800 neurons
Fully connected	2048 neurons
Fully connected	1024 neurons
Fully connected	43 neurons = number of classes
Softmax	Get probabilities from logits

The CNN shown is based on the LeNet architecture from Yan Lecun from 98. Adjustments have been made in the parameters for the fully connected layers to compensate for the higher amount of classes (43 classes here vs. 10 classes in LeNet) and in the depth of the convolutions, since there might be more patterns to find in traffic signs than in letters. The net is scaling the depth from 1 grayscale layer to 12 depth layers in the first convolution and 32 layers in the second convolution. This allows for finding more details of the traffic sign images, compared to the letters of LeNet.

Relu activations are used to avoid the problem of vanishing gradients. When i.e. sigmoid activations are used, the gradient gets smaller and smaller with increasing or decreasing x.

Max pooling layers are used to avoid overfitting and reduce the computational cost. This is done with a down-sampling filter of 2x2. Example: Input layer = 10x10x32 –> Output layer = 5x5x32.

Additional measures for regularizing the network where:

- A dropout in the fully connected layer
- Batch normalization
- L2 regularization

After playing around with these techniques, I could find out that batch normalization is very powerful alone, which lead me to not using dropout and L2 at all.

3. Training

To train the model, I used an the adam optimizer, since it shows very good results (Here: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/). I set the batch size to a rather low value (of 64), since it improves the validation accuracy. It is a good technique to set the exact value to a power of 2, because the GPU ram, where the model is trained is also almost always a number with a power of 2 and the batches are loaded directly into memory.

The number of epochs where set to something between 10 and 50 to allow the network to get close to the optimum. In the training I saw that the necessary >93% validation accuracy could already be reached within the first couple of iterations.

The learning rate was set to a low value of 0.001, since this was also used in the lectures. I played around with the learning rate and saw that bigger learning rates lead me to worse results.

For a loss function, the mean of the cross entropy was used.

4. Validation accuracy

The model was trained and validated with the validation set in batches. As expected it can be shown that, training accuracy increases as the model is trained while at the same time it is important to see an increasing validation accuracy. If this is not the case, the model would not be able to generalize over the seen training data. In my case I could reach a validation accuracy of ca. 97%.

My final model results were:

- training set accuracy of ca. 99%
- validation set accuracy of ca. 97%
- test set accuracy of ca. 97%

For this problem the LeNet architecture from Yan Lecun from 98 was chosen. I believe this architecture can be transferred to other image classification problems as well, since the convolutional approach in combination with a fully connected layer classifier is very powerful, as can be shown by the fact that over 20 years later these types of networks are still used in state of the art architectures.

The transfer is possible from my point of view, because as letters and numbers, traffic signs are also build from patterns like horizontal/ vertical/diagonal lines, quarter/half/full circles, etc. A convolutional neural network (CNN) will adjust its weights to whatever the labels are telling it to go towards.

The training, validation and testing accuracy of ca. 97% after some optimization shows that this model is still very much up to date for this kind of problem.

Test a Model on New Images

1. Additional test images

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

The first four images might be difficult to classify because they are warped. All of the images have background noise that is different from those in the training

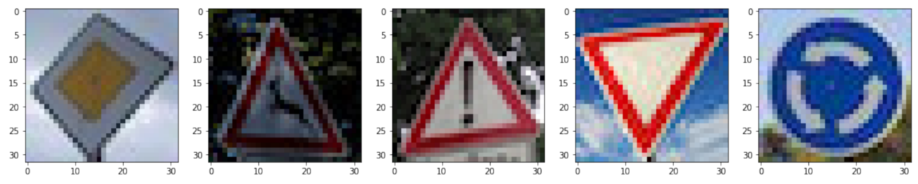


Figure 3: Traffic signs from the internet

set, which might be difficult as well for the network. Additionally the brightness is different in all of the pictures. Especially picture 2 is quite dark and might therefore be difficult to classify.

2. Classification results

Here are the results of the prediction:

Image	Prediction
Priority road	Priority road
Wild animals crossing	Wild animals crossing
General caution	General caution
Yield	Yield
Roundabout	Roundabout

The model was able to correctly guess 5 of the 5 traffic signs, which gives an accuracy of 100%. This compares favorably to the accuracy on the test set of 97%

3. Probability for classification results

For the third and fourth image the model is very sure about the classified image with a probability ov > 80%. For the last image, showing the roundabout, the net is not sure at all. A probability of only 13-14% for the image showing a roundabout, while the other probable candidates for the net are:

- Speed limit 100km/h (approx. 11%)
- Speed limit 120 km/h (approx. 10%)
- Priority road (approx. 8%)
- Yield (approx. 6%)

The probability for the first and second image being a priority road and a wild animals crossing sign respectively is ok with >25% and >35%.

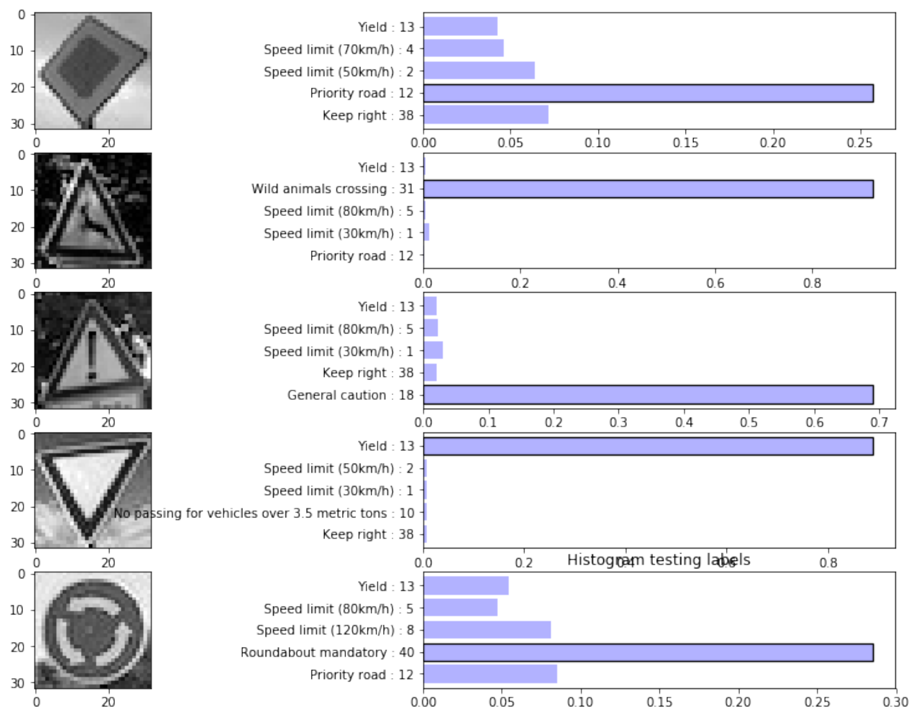


Figure 4: Traffic signs from the internet