

Build a Traffic Sign Recognition Project

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

Data Set Summary & Exploration

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

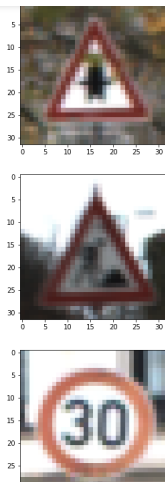
I used the numpy library to calculate summary statistics of the traffic signs data set:

- * The size of training set is 34799
- * The size of test set is 12630
- * The shape of a traffic sign image is 32x32x3
- * The number of unique classes/labels in the data set is 43

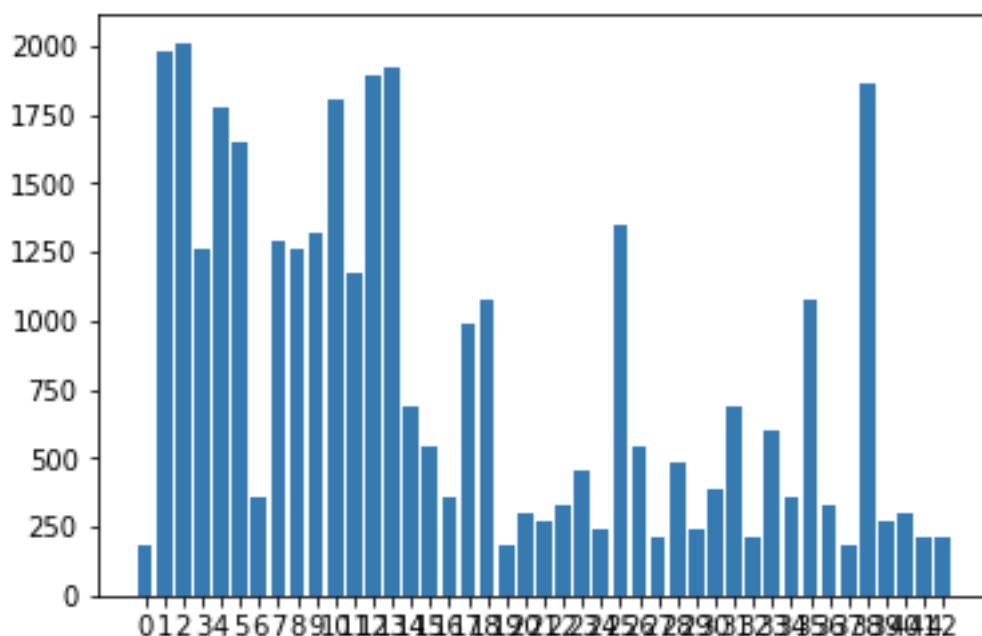
Exploratory visualization of the dataset

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

We first display 3 random traffic signs from the training set :



Then, the bar chart below shows the occurrences of each traffic sign in the training set:



The chart shows the occurrences of some signs is approx. 10 times that of others. The distribution is unbalanced and this could lead the model to better train on certain signs compared to others.

Design and Test a Model Architecture

Pre-processing

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 because this simplifies the input from 32x32x3 to 32x32x1, hence saving computing power and seemed to increase accuracy. One could argue that colour is not that important in recognizing a traffic sign and hence removing it allows the model to focus on more discriminating features.

As a last step, I normalized the image data because this allows to achieve more consistency throughout the dataset. This ensures signs taken with higher or lower contrast for example will be treated the same way by the model.

Model architecture

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

My final model consisted of the following layers:

Layer	Description
Input	32x32x1 greyscale image
Convolution 5x5	1x1 stride, valid padding outputs 28x28x6
RELU	
Max pooling	2x2 stride, valid padding, outputs 14x14x6
Convolution 5x5	1x1 stride, valid padding, outputs 10x10x16
RELU	
Max pooling	2x2 stride, valid padding, outputs 5x5x16
Flatten	Outputs 400
Fully connected	Outputs 120
RELU	
Dropout	
Fully connected	Outputs 84
RELU	
Dropout	
Fully connected	Outputs 10

It is based on the Lenet model with the addition of 2 dropout layers.

Model training

The code for training the model is located in the eighth cell of the ipython notebook.

To train the model, I used an Adam optimizer, a batch size of 128, 20 epochs and 0.001 learning rate. Although Adam optimizer is more expensive from the computing power angle, it produces better results than the gradient descent optimizer as it is able to optimize the step size. The number of epochs was increased to 20 until the accuracy reached a plateau.

Solution approach

The code for calculating the accuracy of the model is located in the ninth cell of the Ipython notebook.

My final model results were:

- * training set accuracy of 0.996
- * validation set accuracy of 0.937
- * test set accuracy of 0.920

As suggested, I have started with the Lenet-5 architecture which is quite successful with handwriting recognition. Similarly, traffic signs consist of relatively simple patterns and hence this architecture can be a suitable starting point.

Running the Lenet-5 model as it is on the traffic signs dataset produced an 83% accuracy on the validation set. So, it is indeed a good starting point but still needs improvements to reach a satisfactory accuracy.






In order to achieve a 93% accuracy on the validation set, I have taken an iterative approach:

1. Before any modification to the architecture, I applied grayscaling. This improved the accuracy by 3%.
2. Then, I tried normalization with different parameters. The best improvement I got was with a normalization between -0.5 and 0.5. This improved the accuracy by 4%, reaching 90%.
3. I then inserted dropout layers, right after the RELU at different stages of the architecture. After several combinations, it appeared the dropout functions were most efficient if applied at the final stages of the architecture, at the level of the fully connected layers. If applied at the start of the architecture, the accuracy drastically drops. Finally, I added a dropout layer after each fully connected layer (except the last one of course as its output is the logits)
4. The 2 added dropout layers need the `keep_prob` parameter. I started with 0.5 and then tried other values (0.4, 0.6, 0.7, 0.8, 0.9) and noticed that setting it to 0.8 produced better results. This allowed to reach 92% accuracy.
5. Finally, as I noticed accuracy was still globally improving at the 10th and final EPOCH, I decided to increase the EPOCH number to 20, which allowed to achieve the target accuracy 0.937.

Test a Model on New Images

Acquiring new images

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

	Speed limit 60km/h This speed limit sign should be easy to classify, although the model might confuse it with 50 speed limit sign as they are quite close.
	Children crossing The shape of the picture (rectangle and not square) and the text in the centre may be a difficulty.
	Double curve No particular difficulty here.
	Stop sign The trees in the background and text at the bottom of the sign may be a difficulty.
	Wild animals No particular difficulty here.

Performance on new images

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

Here are the results of the prediction:

Image	Prediction
3. Speed limit (60km/h)	3. Speed limit (60km/h)
28. Children crossing	28. Children crossing
21. Double curve	11. Right-of-way at the next intersection
14. Stop	14. Stop
31. Wild animals crossing	25. Road work

The model was able to correctly guess 3 of the 5 traffic signs, which gives an accuracy of 60%. This is below the test set accuracy which was more than 90%.

The double curve and wild animals crossing signs were supposed to be the easiest to guess as they are plain signs without any noise. However, it turns out that the model fails in identifying these.

This is most probably because the model has not learned on any of those simple signs and is more efficient with images with noise similar to those of the training set.

The human perception of a difficulty to recognize a sign (noise such as trees, text) is definitely different from the difficulties that this model will face.

Model Certainty - Softmax Probabilities

The code for making predictions on my final model is located in the 11th cell of the Ipython notebook.

For the 2 first images, the model is 99% sure of its predictions:

Probability	Prediction
0.99	3. Speed limit (60km/h)
0	1. Speed limit (30km/h)
0	2. Speed limit (50km/h)
0	14. Stop
0	38. Keep right

It is interesting to see that the second and third best probabilities also correspond to speed limit signs, which are very similar to our image.

Probability	Prediction
0.99	28. Children crossing
0	31. Wild animals crossing
0	8. Speed limit (120km/h)
0	24. Road narrows on the right
0	11. Right-of-way at the next intersection

For the third image, double curve, the model is certain it corresponds to another sign. We can note the second best probability, although close to 0, corresponds to the correct sign.

Probability	Prediction
0.99	11. Right-of-way at the next intersection
0	21. Double curve
0	30. Beware of ice/snow
0	34. Turn left ahead
0	24. Road narrows on the right

For the fourth image, the model is relatively sure (89%) to have found the correct sign. We can note here that the speed limit signs and the stop sign seem to 'look' similar to the model (confirmed by the first image for which these signs also appear in the top 5)

Probability	Prediction
0.89	14. Stop
0.07	1. Speed limit (30km/h)
0.02	2. Speed limit (50km/h)
0.01	5. Speed limit (80km/h)
0	4. Speed limit (70km/h)

For the last image, wild animals crossing, unlike the previous images, the model is unsure of what is. Indeed, the 3 first probabilities are all quite high and the correct prediction corresponds to the 3 best probability.

Probability	Prediction
0.47	25. Road work
0.38	22. Bumpy road
0.15	31. Wild animals crossing
0	23. Slippery road
0	29. Bicycles crossing