# CarND Traffic Sign Classifier Writeup

**By James Marshall**

Note: The code for creating the bar graph for softmaxs, as well as the code from the data visualisation that prints 10 random images is attributed to Jeremy Shannon's sample project, as it was a much more organised presentation than my original methods. Jeremy's sample project can be found here.

## Dataset Summary and Exploration

Q1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

I used python methods to do a basic summary of the traffic sign dataset:

-Size of training set: 34,799
-Size of validation set: 4,410
-Size of test set: 12,630
-Shape of a traffic sign image: (32, 32, 3)
-Number of unique classes in the dataset: 43

Q2. Include an exploratory visualization of the dataset.

Here I have done a simple exploratory visualization of the dataset. I have made a bar chart showing the distribution of the classes in the training, validation and test sets, overlapping each other, to see if all three sets had similar ratios, which they seem to.

I have also printed out all of the classes that have a high occurrence rate, in this case, greater than 1,500 data points.

Lastly, I have printed out 10 random images from the training set to get an idea of where the images start from before normalisation.

## Design and Test a Model Architecture

Q1. Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques?

The first stage of preprocessing the images was the greyscale them, which was done by taking the average of each colour in each pixel. This was done as the decrease in training time was more beneficial than the colour of the sign, as the symbol on the sign was enough for the LeNet classifier to work.

I then normalized the images, giving them a mean of 0 and a range from [-1, 1]. I did this as reducing the distribution in the data from large values in the hundreds to [-1,1] makes it slightly easier to train as backpropagation uses multiplication with the learning rate, which could

lead to values in large distributions being over or under compensated. A greater explanation into normalization for CNN's is given here.

Examples of images before and after preprocessing are shown in the notebook.

Q2. Describe what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.)

My final architecture is the same architecture as the LeNet lab, with the same input/output sizes as shown in the lab. This gave a validation accuracy between 94% and 96%, which varied on each training run. The layers were set up in the following way:

1. 5x5 Convolution (32x32x1 Input, 28x28x6 Output)
2. ReLU activation
3. 2x2 Max Pooling (28x28x6 Input, 14x14x6 Output)
4. 5x5 Convolution (14x14x6 Input, 10x10x16 Output)
5. ReLU activation
6. 2x2 Max Pooling (10x10x16 Input, 5x5x16 Output)
7. Flatten (5x5x16 Input, 400 Output)
8. Fully Connected Layer (400 Input, 120 Output)
9. ReLU
10. Dropout (0.5 prob)
11. Fully Connected Layer (120 Input, 84 Output)
12. ReLU
13. Dropout (0.5 prob)
14. Fully Connected Layer (84 Input, 43 Output)

Q3. Describe how you trained your model.

I followed the LeNet lab closely, using the Adam Optimizer, and final parameter settings of:

- Batch Size: 128
- Epochs: 50
- Learning Rate: 0.008
- Mu: 0
- Sigma: 0.1
- Dropout Probability: 0.5

Q4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93.

I initially used the Gradient Descent Optimiser, to horrible results, with validation accuracies no more than 6-7%. Once I switched over to the Adam Optimizer, the validation accuracy went into the 90% range.

I used the mean (mu), stddev (sigma) and dropout probabilities from the LeNet labs, as they seemed to work quite well.

I then used trial and error to see what decimal place should be the first non-zero value and found the 3rd decimal place was the best in terms of accuracy vs time trade off. I then used trial and error on values between 0.005 and 0.01 to find a value that seemed to work the best, eventually settling on 0.008. I was originally using 100 epochs, but by watching the validation accuracy and loss, and keeping in mind overfitting, 50 epochs seemed to be a good number to epochs.

My final results were:

Validation accuracy: 95.6%

Test accuracy: 93.9%

## Test a Model on New Images

### Q1. Choose five German traffic signs found on the web and provide them in the report.

The 5 images I chose from the web have been plotted after being resized, but before grey scaling and normalization.

The first sign is different to the training dataset because it is not a real world picture, instead it is an image of just the traffic sign. This may affect the classification, but the clarity of the image may make it simple to classify.

The third, fourth and fifth signs may be a little harder to classify as the images aren't perfectly centered, which my training set was not augmented to include.

The second image may be quite hard to classify, as after resizing, the image is already quite unclear to the point where it is difficult for me to identify.

### Q2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set.

My model correctly predicted 3 out of 5 of the new images for an accuracy of 60%. The breakdown of the predictions are as follows:

1. Sign: 60km/h — Prediction: 50km/h
2. Sign: Children Crossing — Prediction: Children Crossing
3. Sign: No entry — Prediction: No entry
4. Sign: Right of way — Prediction: Right of way
5. Sign: Stop — Prediction: No passing

### Q3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction.

As shown in the notebook, the top 5 softmax predictions are shown on a bar graph, with the original image next to it for reference. All of the predictions seem quite certain of their classifications, with a maximum of 2 signs being shown as a possibility.

The first sign, while incorrect, seems to be able to identify the sign as a speed sign, however, it has a tough time recognising what the speed limit is. Hence, the two highest softmaxs are for the 50km/h and 30km/h signs respectively.

The second sign was correct, which is surprising as I thought this would be the hardest with the quality of the image, however it does think that “the end of no passing” sign is a slight possibility, with a probability of about 18%.

The third and fourth signs are both correct and very certain of the classifications with probabilities of 100% each.

The fifth sign is incorrect, which is surprising as the quality of the stop sign image is quite high, however, the model doesn’t even consider a stop sign to be a possibility, which seems to suggest that the model has a tough time identifying stop sign images. The other point about this particular image is that the model is very certain that the sign is a “no passing” sign, which may means that if the model returns a “no passing” sign as a prediction, maybe the result should not be trusted.