Traffic Sign Classifier

Data summary:
Number of training examples = 34799
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43

Exploratory visualization on the dataset:
I started with first viewing a random sample from the dataset along with the class that it belongs to.
Then I modified it to view one random sample from each of the 43 classes and then correlated those to the class names in the csv file

Preprocessing techniques used and why these techniques were chosen:

1. Turn the images to grayscale to minimize training time. It might also help the training neglect the color information as most signs were of similar colors (red and white with some signs as yellow) and would thus not help distinguish one sign from another perhaps. Although in testing I found that was getting comparable validation accuracy even with color images. So, I suppose efficiency would be the prime reason here.
2. Normalize the images to [-1,1] range to help reduce mean of the overall data and improve training
3. I then shuffled the data so the training does get effected by the ordering of the data

Model Architecture:
I used the LeNet model architecture as discussed in the lesson
The first convolutional layer has a 5x5 filter with an input depth of 1 (grayscale images) and output depth 6. The vertical and horizontal stride is set to 1. So our output height/width will be 28x28x6 (as per the formula (outHeight = inHeight - filterSize +1) / stride). Then apply an activation to it and run through a pooling layer with a filter size of 2x2 and that reduces the output to 14x14x6
The second convolutional layer is set up in a similar way with the output size now 10x10x16 (14-5+1). Another round of activation and pooling. After flattening the output from the second convolutional layer we then then pass it through two fully connected layers with relu activations and a dropout added after each layer.
At the end we then have a fully connected output layer that the the output equal to the number of classes in our dataset (43)

This table better explains the whole architecture

Layer	Description
Input	32x32x1 grayscale image
Convolution	5x5 filter, 1x1 stride, valid padding, outputs 28x28x6
Activation	relu
Max pooling	2x2 stride, outputs 14x14x6
Convolution	5x5 filter, 1x1 stride, valid padding, outputs 10x10x16
Activation	relu
Max pooling	2x2 stride, outputs 5x5x16
Flatten	Size 400
Fully connected	input 400, output 120
Activation	relu
Dropout	70% keep probability
Fully connected	input 120, output 84
Activation	relu
Dropout	70% keep probability
Fully connected	input 84, output 43

The final accuracies were as follows:

- Validation set accuracy of 95.3%
- Test set accuracy of 93.4%

Architecture discussion:

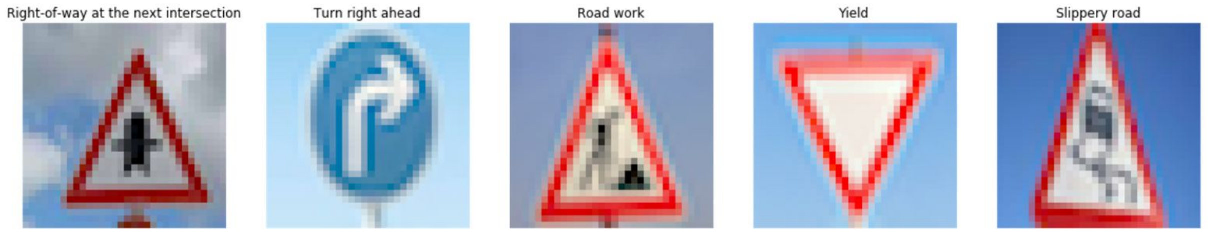
The basic architecture is the one shown in the lesson.

To begin with the LeNet model seemed to do a decent job and was giving an accuracy of ~89%. I tweaked the learning rate and the batch size to balance the training time and the accuracy increased a bit but not by a whole lot. That is when I experimented with switching back to color images and found that the accuracy was still pretty much the same but the training time had obviously gone up. So I switched back to grayscale. Then I added dropout with a 50% keep probability and saw a decent rise in the accuracy. I experimented more with its values and finally settled on a value of 70%. I ran out of time and have not experimented with adding another convolutional layer to the model. This could perhaps give better results. Perhaps augmenting data to include transformed images (in scale, translation and even rotation) will give the model more information to learn from and perform better on various images of traffic signs. I think as a starting point this architecture is a decent fit.

Training and testing:
Optimizer - AdamOptimizer
Epochs - 20
Batch size - 112
Learning rate - 0.001
mu = 0
sigma = 0.12
Keep probability = 0.7
The training was most affected by the keep probability and the learning rate - I experimented with a lot of values but settled with the ones above that gave a validation accuracy of ~95% and was not too slow to train either (I did not have access to a gpu). The batch size was also modified with bigger values producing lesser accuracy in fewer epochs - I presume if left to train for a more number of epochs the accuracy would perhaps improve.
The test set accuracy was 93.4%.

Testing on new images:

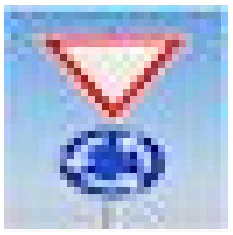
I used five new images to test and they are visualized in the project. They are shown below:



The last image of a “Slippery Road” sign did not do well for some reason (I think it is a little skewed and not as front on as the other images and that’s why the it failed to classify it correctly). It was predicted to be a “General Caution” sign with a high probability and interestingly the correct class did not even show up in the top 5 softmax probabilities at all.

The other four images were classified correctly with great probability. The first image (right of way at next intersection) is somewhat dark and has clouds in the background and could have been a tough case. Initially, all the images had a lot more background in them and the test did not do well for quite a few them. I then cropped the excess background out and that made a huge difference.

The Yield sign was actually a composite of two signs (I later cropped out the roundabout part):



And was predicted to be a “Yield” sign. Again, if the data can be augmented to include such composite signs with another class added to label such signs then perhaps the network will cover wider cases as this.