Traffic Sign Classifier Writeup

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

Rubric Points
Here I will consider the [rubric points](https://review.udacity.com/#!/rubrics/481/view) individually and describe how I addressed each point in my implementation.

Data Set Summary & Exploration:

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

* Size of training set: 31,319
* Size of the validation set: 7,890
* Size of test set : 12, 630
* Shape of a traffic sign image: 32 x 32 x 3
* Number of unique classes/labels in the data set: 43

Design and Test a Model Architecture:

The only preprocessing that I did was to normalize the pixel values. It is always a good idea to have a mean of 0 and equal variance because a badly conditioned problem makes the optimizer do a lot of searching to find a good solution. A well conditioned problem makes it easier for the optimizer to do its job. Since we're dealing with images, all I had to do was take the pixel values of the images, which range from 0 to 255, subtract 127.5, then divide by 127.5. That made the values range from -1 to 1.

I opted not to convert the images to grayscale because I figured that the colors of the signs could be a helpful feature for my network to learn. Although, converting to grayscale might have led to my network training faster, in the end I think using the RGB channels led to more accurate results.

I decided to use the LeNet architecture, but I added a dropout layer after the first fully connected layer to reduce overfitting. This increased my validation accuracy from ~ 91% to ~ 97% and my test accuracy from ~90% to ~94%. It also allowed me to reduce the number of epochs from 50 to 25, respectively, which means my network trained much faster and was more accurate overall.

Layer	Description	Output
Input	32x32x3 RGB image	
1.Convolution	5x5 filter, a 1x1 stride, valid padding	28x28x6
RELU		
2.Max Pooling	2x2 kernel, a 2x2 stride, valid padding	14x14x6
3.Convolution	5x5 filter, a 1x1 stride, valid padding	10x10x16
RELU		
4.Max Pooling	2x2 kernel, a 2x2 stride, valid padding	5x5x16
Flatten		400

5.Fully Connected		120
RELU		
6.Dropout	keep_prob = 0.5	
7.Fully Connected		84
RELU		
8.Fully Connected		43

I trained my model using the following hyperparameters:

*Epochs: 25
*Batch Size: 128
*Learning Rate: 0.0008
*Keep Probability: 0.5
*Optimizer: AdamOptimizer
*Mean(mu): 0
*Standard Deviation(sigma): 0.1

I chose to use the LeNet architecture because convolutional neural networks work well with image classification. The weight-sharing architecture and translation invariance takes advantage of the structure of images to perform better.

The LeNet architecture takes in 32x32xC images. Since
the traffic sign images are already 32x32 pixels, there was no need to pad the images. The traffic sign data included 34,799 training images and 4,410 validation images, which is about 11% of the training images. As a good rule of thumb, it is a good idea to use 20% of training set for the validation set, so I decided to use the train_test_split function in the sklearn library to slice off an additional 10% from the training set to add to the validation set. Once uploaded, I proceeded to pre-process the data by normalizing it.

Before running the neural network, I had to adjust the last layer to make sure that the output matched the number of classes, which is 43. I got good results right off the bat using a learning rate of 0.001, 50 epochs, and a batch of size 128 but failed to reach a validation accuracy of 93%. I got better results using a learning rate of 0.0008, but the model still needed improvement. At this point, I decided to add a dropout layer, with a keep probability of 0.5, after the first convolutional layer to reduce overfitting.

My final model results were:
Validation set accuracy: 96.8%
Test set accuracy: 0.94.1%

Test a Model on New Images

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

Here are the results of the prediction:

Image	Prediction
Yield	Yield
Road Work	Road Work
Wild Animals Crossing	Wild Animals Crossing
Bumpy Road	Bumpy Road
Turn Right Ahead	Turn Right Ahead

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 94.1%.

The code for making predictions on my final model is located in the 16th cell of the Ipython notebook.

For the first image, the model is relatively sure that this is a yield sign (probability of ~1.0), and the image does contain a stop sign. The top five soft max probabilities were

Here is a chart of the top 5 predictions with corresponding probabilities of the first image:

Probability	Prediction
~1.0	Yield
~0.0	Priority road
~0.0	Keep right
~0.0	Turn left ahead
~0.0	No passing

Here is a chart of the top predictions with corresponding probabilities of the second image:

Probability	Prediction
~1.0	Road work
~0.0	Wild animals crossing
~0.0	Bicycles crossing
~0.0	Double curve
~0.0	Speed limit (30km/h)

Here is a chart of the top predictions with corresponding probabilities of the third image:

Probability	Prediction
~1.0	Wild animals crossing
~0.0	Double curve
~0.0	Beware of ice/snow
~0.0	Slippery road
~0.0	Road work

Here is a chart of the top predictions with corresponding probabilities of the fourth image:

Probability	Prediction
~0.64	Bumpy road
~0.33	Traffic signals
~0.02	Bicycles crossing
~0.0	Road work
~0.0	Road narrows on the right

We can see that for the fourth image, our classifier wasn't quite as confident as with the other images. This could be due to the fact that there wasn't an even distribution of images in each class. In out training set, there were more images of certain classes than of others. Because of that, our network might not have trained as well, and thus would not be as confident in its prediction.

Here is a chart of the top predictions with corresponding probabilities of the fifth image:

Probability	**Prediction**
~1.0	Turn right ahead
~0.0	Turn left ahead
~0.0	Ahead only
~0.0	Keep left
~0.0	Roundabout mandatory