Traffic Sign Classifier Project

Brent Wylie, March 29th, 2017

The goals / steps of this project are the following:

- Load the 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

Dataset exploration

For this project, I dynamically calculated the number of training and testing examples, the example dimensions, and the number of unique classes in the dataset. These metrics are printed in the jupyter notebook printout.

I analyzed the training, validation and testing set, as well as the combined set of all of these to get an idea of what the most frequent and infrequent traffic sign categories were within the data set. I plotted a histogram showing the occurrences of each category in the datasets, as well as some examples of the 5 most uncommon categories. I chose to visualize the most uncommon categories as I expected the network to perform worse on categories with fewer examples.

Design and Test a Model Architecture

In the preprocessing step, I chose to convert the images to grayscale in order to reduce network complexity by reducing the number of channels in the network to provide a lighter computational load during training and inference. It also allowed for easier porting of the LeNet lab to be used for this purposes. Other preprocessing steps I did were to reshape to 32,32,1 to allow for CNN processing, equalizing the histogram of the image, improving overall contrast, and to center normalize the data set so each image starts from a fairly consistent starting point and we avoid any particular images from dominating the learning.

The architecture I used was the LeNet network we built in the previous lesson. This was a natural choice as I already had the code, and the only tweak really necessary to get it to work was to change the output layer to have correct number of labels. Simply porting the LeNet model was enough to see about 85- 91% validation accuracy. I decided to add two sections of dropout following the first and second fully connected layers in order to promote robustness and reduce overfitting. The model has 10 layers, not including input. See Below for summary of layers and sizes.

```
Input (32x32x1)
Convolution 1 (5x5x1 filter, valid padding, stride of 1s) Output = 28x28x6
Relu 1
Max pool (2x2 filter, 2x2 stride, valid padding) Output = 14x14x6
Convolution 2 (5x5x6 filter, valid padding, stride of 1s) Output = 10x10x16
Relu 2
Max pool (2x2 filter, 2x2 stride, valid padding) Output = 5x5x16
Flatten (5x5x16 to 400 nodes)
Fully connected layer (Relu, with dropout added) Output = 120
Fully connected layer (Relu, with dropout added) Output = 84
Fully Connected output mapping Input = 84, Output =43
```

I used the following hyperparameters: A dropout probability of 20% (keep probability of 80%), a batch size of 64, a learning rate of 0.002 and 10 Epochs. Using the Adam optimizer included with tensorflow and the above hyperparameters, I was able to reach about 94% accuracy in validation. Due to oscillation on accuracy figures between epochs I expect there is still some overtraining going on, but at an acceptable minimum.

This solution consistently performs at 94% or higher on the validation set.

Test a Model on New Images

Using the information about the most uncommon categories in the dataset, I chose the 5 most uncommon traffic signs, found examples of them, and loaded them into the notebook and plotted them. The 100 and 120km/h speed limit signs are fairly recognizable, but the other 3 categories (no passing, pedestrians, and end no passing for vehicles over 3.5 metric tons) were much less intelligible. I chose the most uncommon categories with the intuition that less examples would mean higher chance of bad classification. However, I would need to generate a histogram of the average top 1 softmax value for each category in order to know generally if this was true.

The network performed rather well on these images, being able to correctly predict the category for the five images. The accuracy of the predictions on these images was therefore 100%, whereas the accuracy of the network on the full test set provided (X_test,y_test), was 92.1%.

I output the softmax probabilities of the 5 new images, and for the most part the top 1 score was above 90% for each of the five examples, with the exception of the pedestrian image which scored 87.4%. Every top score corresponded to the correct label as well, of course (otherwise the prediction would be wrong). Similarly, the next best score for all but the pedestrian sign was orders of magnitude smaller. The pedestrian sign had about a 6.8% confidence that the sign was actually a ‘traffic signals’ (label 26) sign, which is a similar shape and color with different center icon. It is also worth noting that ‘pedestrians’ (label 27) had only 210 occurrences in the training set and tied label category 42 (end of no passing for large vehicles) for rarest sign in training set. The traffic signals sign which was the next best guess (at only 6.8%) had 540 occurrences in the training set, so from the training set point of view, it was twice as likely to be a traffic signal sign as it was to be a pedestrian warning sign. The other labels chosen all had 1000 plus examples. It is likely that more examples of the pedestrian sign and possibly the traffic signal sign would push this categorization above 90%.

Final comments

Overall I was surprised at how well this architecture worked for being a port of the LeNet digit recognition architecture. Only a small amount of dropout was needed to vastly improve the robustness, and I expect more examples of certain categories in the dataset (namely all categories with sub 500 examples) would improve the classification results. I expect that the network could learn to be slightly more robust with color data, provided that intensity information is also presented well. Independently considering R G and B values would likely not improve performance, but using YUV or adding a grayscale channel to an RGB architecture would likely do better. During this project, I also learned of the german sign **detection** dataset, and thought it might be interesting to try to combine the two, using classical computer vision approaches to detect where the street sign is, and then using this CNN to classify them.