David Clark
3/26/2017

P2 Reflection - Udacity Self-Driving Car Nanodegree

Overview

Project #2 is about using deep learning, specifically convolutional neural networks, to classify images of German traffic signs. This is an important problem, for being able to classify an image of a traffic sign according to its type will allow a self-driving car to make important decisions.

My code for this project is publicly available and can be found here:
https://github.com/SealedSaint/CarND-Term1-P2

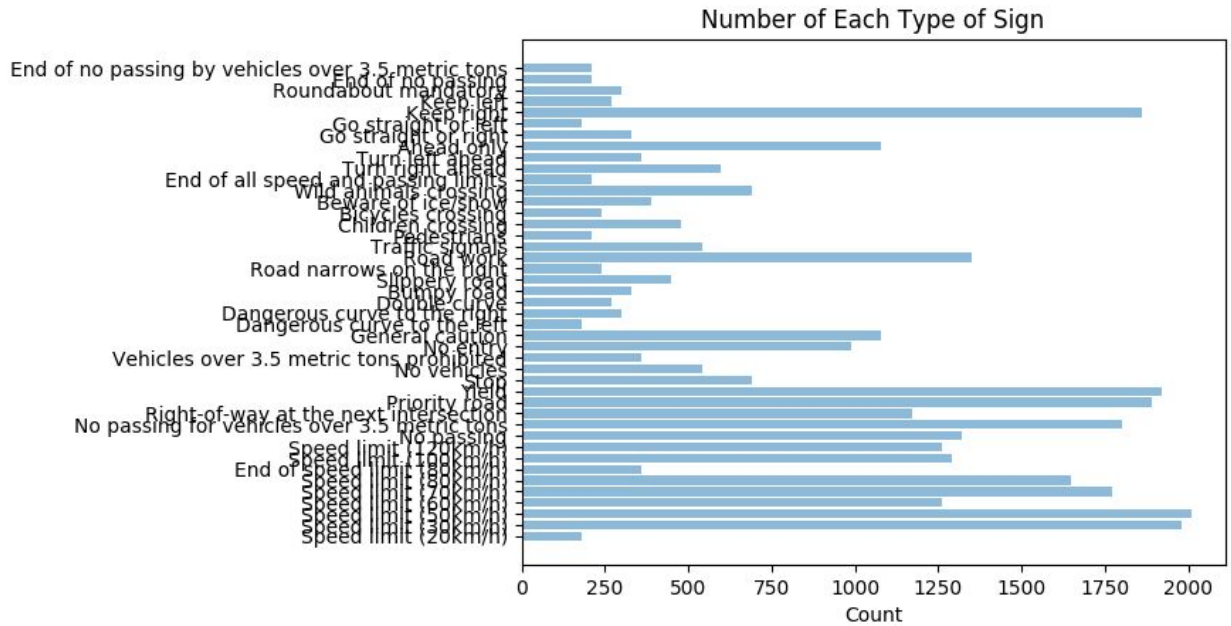
Exploring the Dataset

In Step 1, I explore the dataset and identify some relevant information:

- There 34,799 training images and 12,630 testing images
- The images are indeed 32x32 pixels
- There are 43 different classes of traffic signs labeled 0-42

Exploratory Visualization

Also in Step 1 I print a bar chart showing the distribution of the types of traffic signs. Signs like speed limits, stop, and yield had many more examples than some other less common signs like “go straight or left”. The distribution is shown below.



David Clark
3/26/2017

At the end of Step 1, to prepare for pre-processing, I display a sample image and print the min and max pixel values to ensure we are dealing with image data in the 0-255 pixel range.

Pre-Processing the Data

I did very minimal pre-processing. With about 35,000 training images, I didn't feel like augmenting the data was necessary to get good results, so I used only the raw images from the dataset.

I kept the images in RGB instead of converting to grayscale, because I felt like the color information was not only valuable but important. One major way to identify signs is by the colors they include, and these colors are often simply used and easily distinguishable. I figured these traits should be picked up by an appropriate model.

The two pre-processing steps I performed are:

1. Shuffling
2. Normalizing

I pre-shuffled the data just to make sure it was good and mixed - no groupings of sign types to oddly influence the development of the model. Finally, I normalized the data to [0, 1] by dividing the numpy array by 255.

Dividing the Data to Train, Validate, and Test

Conveniently, the data came pre-divided into training, validation, and test sets. All I had to do was pre-process each of these sets accordingly.

The Model

My model can best be described as the standard LeNet architecture with a few tweaks. The two major tweaks I made are:

1. Account for color images instead of just grayscale
2. Add a third convolutional layer

The table below shows each layer in the model. I used convolutions, activations, maxpooling, and linear combinations to classify the 32x32 color images into one of 43 traffic sign categories.

All activations used are ReLU's.

David Clark
3/26/2017

Layer	Description
Input	Pass in the RGB images of shape 32x32x3
Convolution with Activation	Transform the space from 32x32x3 to 28x28x18 using a 5x5 filter. No maxpooling is used here to preserve space for future convolutions.
Convolution with Activation	Transform the space from 28x28x18 to 24x24x48 using a 5x5 filter.
Maxpool	Condense the space to 12x12x48
Convolution with Activation	Perform a final convolution turning the 12x12x48 space into an 8x8x96 space using a 4x4 filter. (A 4x4 filter was used instead of 5x5 to avoid shrinking the space too far.)
Maxpool	Condense the space to 4x4x96, bringing the total volume from 6,144 down to 1,536.
Flatten	Flatten the space down to one dimension of size 1,536 in preparation for the fully-connected layers.
Fully-Connected with Activation	Calculate linear combinations of the features bringing the space down to size 360.
Fully-Connected with Activation	Calculate linear combinations of the features bringing the space down to size 252.
Output	Calculate linear combinations of the 252 features to produce estimates for each of the 43 classes of traffic signs.

Why this Model?

While there is nothing strange or revolutionary about my model, it's worth visiting how I arrived at this architecture.

I started with the the standard LeNet model and first adjusted for three color channels. This produced similar results to the grayscale images. I expected additional color data to improve the model, but it didn't.

To adjust for the additional color channels, I basically multiplied the output sizes of each layer by three. I figured with more data to digest there needed to be more space to capture those additional relationships. This expanding of the model increased the validation accuracy from 88% to 93% - quite an improvement.

David Clark
3/26/2017

Finally I reasoned that there is probably more abstraction to identify in a traffic sign as opposed to a handwritten digit. To accommodate for this I added a third convolutional layer and adjusted the rest of the model accordingly. Adding a third convolutional layer further improved the validation accuracy from 93% to 97%.

Training the Model

After designing the model, I built the training pipeline and executed it with the model. I one-hot encoded the labels, used softmax cross entropy to determine loss, and used the Adam optimizer.

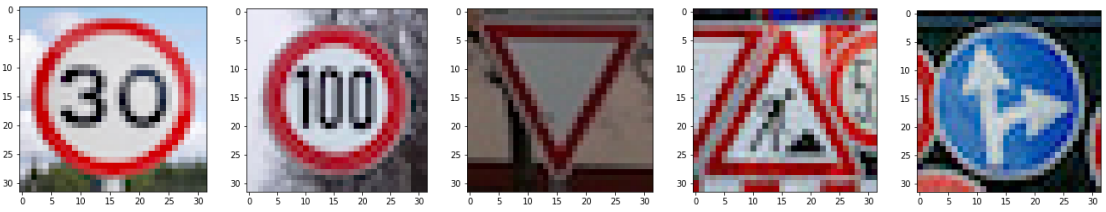
The learning rate was set to 0.001, the batch size to 128, and the number of epochs to 10.

After each epoch the model was tested against the validation set. I set the model to save when a new validation record was reached. In hindsight, I don't think this was the best choice. I made the assumption that lower validation meant a worse network, but I don't think that's the case. Even with slightly lower validation, the network probably continued to improve in future epochs.

The final results from training were a validation accuracy of 97% and a training accuracy of 95%. While those are close enough, I was a tad disappointed the testing accuracy was lower. This probably indicates a bit of overfitting in the model, but I'm not certain how big of an indicator a 2% difference is.

Testing the Network on My Own Images

As a helpful exercise I downloaded five different pictures of German traffic signs from the web to see how my model would classify them. The five images, after pre-processing, are shown below.



The results of the model's predictions for these five images (in order) are summarized in the table below.

David Clark
3/26/2017

Actual Sign	Predicted Sign	Softmax Probability
30 speed limit	30 speed limit	77%
100 speed limit	100 speed limit	100%
Yield	Yield	100%
Road work	Road work	100%
Go straight or right	Keep right	68%

As seen from the table, the model had a total performance of 80% on these five signs. It's comforting to me that the model mistook the "go straight or right" sign for a "keep right" sign, since those are at least close in nature.

I was surprised the model identified the road work sign so well since portions of other signs can be seen in the image (including a full '5' from one). I wonder if the other sign edges in the last sign image, the 'go straight or right' sign, affected the result. I was unable to find a better image online, though, since it's a less common sign. Also, being a less common sign, the model was trained on proportionally less "go straight or right" images. Perhaps that contributed to the model failing the classification too.