Submitted by Kyle Martin

Project 2
Traffic Sign Recognition

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

Dataset Exploration

Dataset Summary

The data I used to train and validate this classifier is from the German Traffic Sign Recognition Benchmark (GTSRB) dataset. It contains almost 52,000 labeled street sign images that are of 43 unique classes.

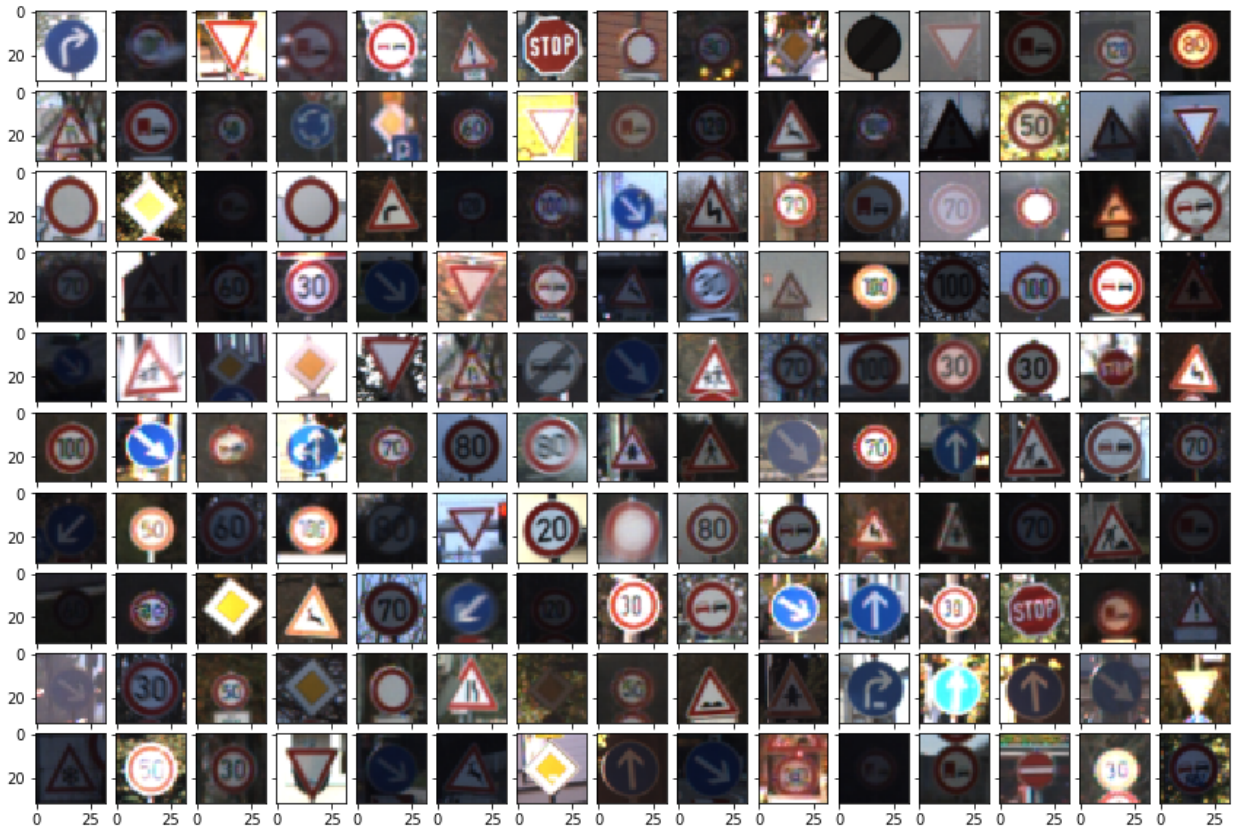
Each image is 32 px by 32 px.

This is how the dataset is split out into training, validation and test sets:

Training Set	67.1%
Validation Set	8.5%
Test Set	24.3%

Exploratory Visualization

In order to get an idea of what kind of images I was going to be working with, I wrote some code to pick 150 random images and display them.



Submitted by Kyle Martin

Design and Test a Model Architecture

Preprocessing

Before I feed the images into the my convolutional neural network (CNN), there are a number of techniques that are important to prepare the images for classification. I chose OpenCV to perform the processing steps described below.

There a number of issues that we need to solve before training the CNN.

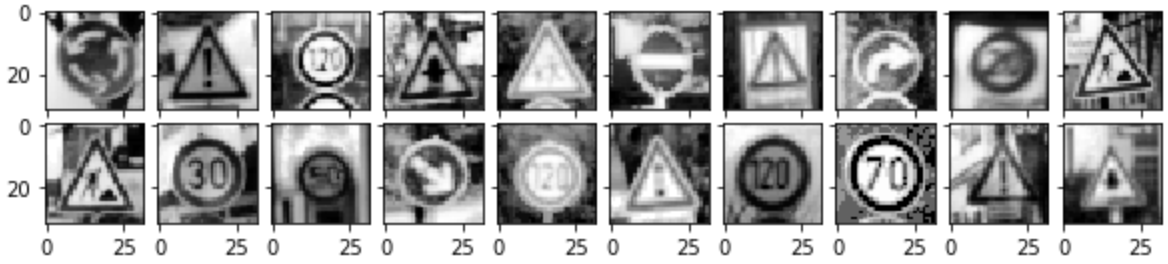
First, the architecture I chose requires an input image to be 32x32x1, meaning it must be 32px wide, 32px tall, and 1 channel. Grayscaling the image will help us here by

Second, the provided images have a varying levels of brightness and color distribution. Applying a histogram normalization algorithm will help with leveling the brightness levels.

Third, we need to ensure the contrast between all the images is averaged so to reduce noise and allowing our CNN to focus on shapes rather than noise around boundaries of objects. We do this using a statistical technique called zero-centering.

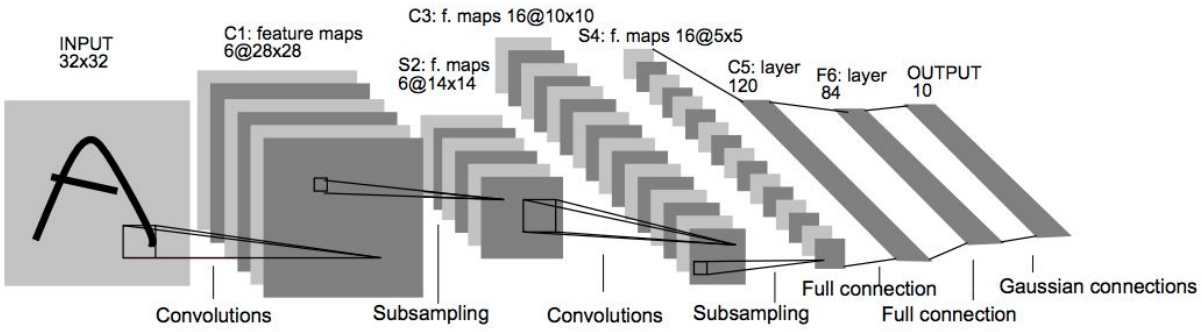
Finally, incoming images may not be exactly 32x32 so we resize each image to 32x32.

Here's a few training images after preprocessing:



Model Architecture

The submission provides details of the characteristics and qualities of the architecture, such as the type of model used, the number of layers, the size of each layer. Visualizations emphasizing particular qualities of the architecture are encouraged.



I chose the classic LeNet architecture to classify the German street signs because it's relatively simple to implement and fast to train and the original author used it on this exact problem.

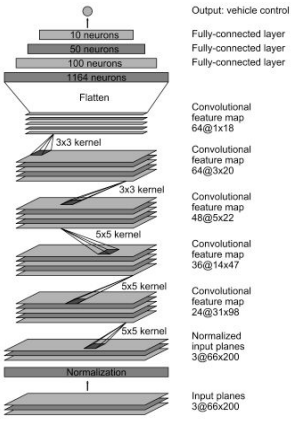
Here's the architecture layers and sizes in descending order:

Layer	Description
Input	32x32x1 RGB Image
Convolution 1	32x32x1 Input, 28x28x6 Output, 5x5 Window, Stride of 1x1, Valid Padding
RELU	28x28x6 Output, Layer activation
Max Pool 1	28x28x6 Input, 14x14x6 Output, 2x2 Window, Stride of 2x2, Valid Padding
Convolution 2	28x28x6 Input, 14x14x6 Output, 5x5 Window, Stride of 1x1, Valid Padding
RELU	14x14x6 Output, Layer activation
Max Pool 2	14x14x6 Input, 5x5x16 Output, 2x2 Window, Stride of 2x2, Valid Padding

Flatten	5x5x16 Input, 400x1 Output
Fully Connected 1	400x1 Input, 120x1 Output
Fully Connected 2	120x1 Input, 84x1 Output
Output	84x1 Input, 10x1 Output

If I had more time, I'd like to experiment with stacking 2 or 3 more convolutional layers before and after pooling layers.

I'd also love to experiment with Nvidia's proposed architecture in their [End to End Learning for Self-Driving Cars] paper.



Model Training

To train the LeNet architecture, I had to make several decisions about the hyperparameters and techniques used.

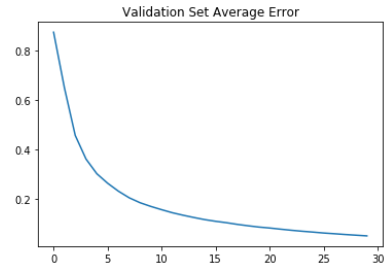
Optimizer

I chose the Adam Optimizer because because it's been shown to have good overall performance and it also reduces the number of hyperparameters compared to stochastic gradient descent optimization.

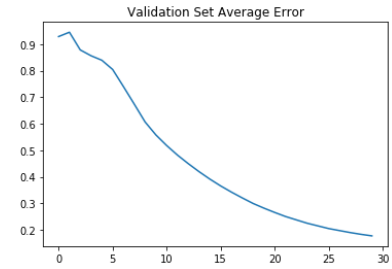
Batch Size

With a fixed epoch of 30 and a learn rate of 0.0001, I tested several batch sizes to see how it affected validation accuracy. It seemed like the larger I made the batch size, the slower networked learned. I suppose it would eventually converge given enough epochs. It seems like there's a proportional relationship between batch size and epochs needed to reach a fixed validation accuracy.

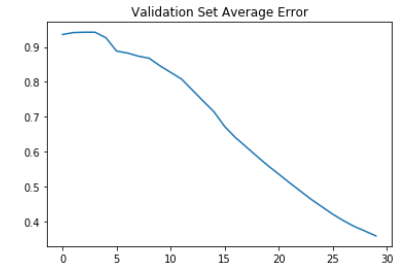
In the end, I chose a batch size of 128 because it seemed to have the fastest convergence to 95% validation accuracy given 30 epochs.



128 Batch Size



1024 Batch Size



2048 Batch Size

Epochs

I determined an epoch of 30 was resulted in 95% validation accuracy in just about 5 minutes of training on an AWS g2.2xlarge instance. I also tried extending the epochs to

40 which resulted in a 97% validation accuracy. I wanted to avoid overfitting so I decided 30 epochs was a good place.

Learning Rate

I chose a learning rate of 0.0001 through experimentation.

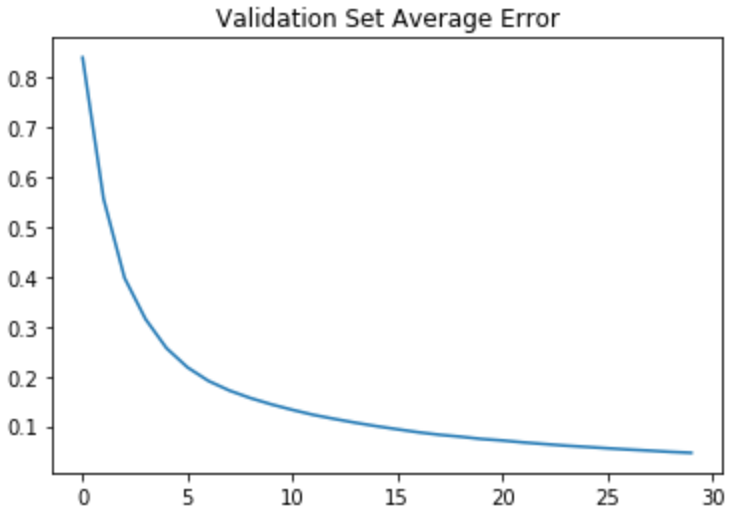
Parameter Initialization Parameters

I randomly sampled from a truncated normal distribution with a mean of 0 and standard deviation of 0.05. These were chosen through experimentation.

Final Model Results

With the given model parameters and hyperparameters above here are the final results:

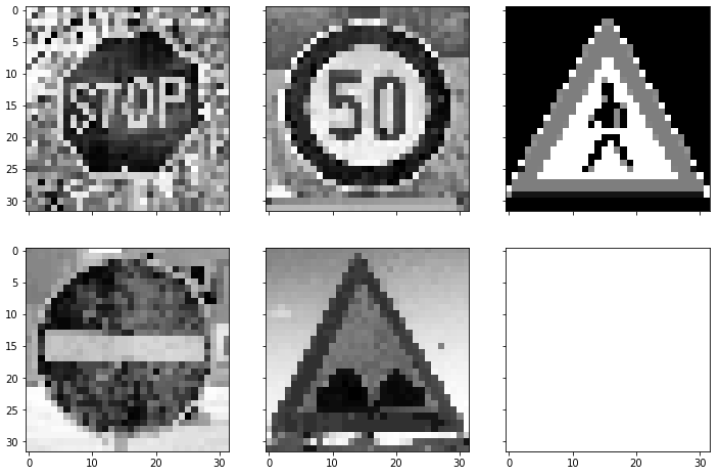
Train Accuracy	95.4%
Test Accuracy	95.4%
Validation Accuracy	95.4%
New Images Accuracy	80%



Test a Model on New Images

Acquiring New Images

I found 5 random Germans street signs off of Google Images and I cropped them similar to what the training set had. I loaded them into my Jupyter Notebook and plotted them.



Performance on New Images

When I ran them through the trained network, it got ⅘ correct which is 80%. The validation accuracy was around 95%. The image that the network has trouble on was the pedestrian sign. This could be for a number of reasons. The first reason is that this is image was wasn't actually from a real picture, it was a graphic from Wikipedia. Perhaps some of the details that the classifier is looking for are not present in this perfect graphic. The second reason is that the model could be overfitted and it's no longer able to generalize. The third reason is that there might not have seen enough examples of this sign and the model is underfit for this class.

Image	Prediction
Stop Sign	Stop Sign
Speed Limit 50	Speed Limit 50

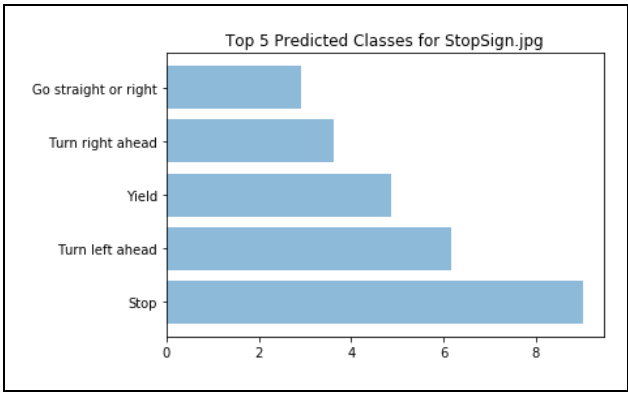
Pedestrians	Right of way at next intersection
No Entry	No Entry
Bumpy Road	Bumpy Road

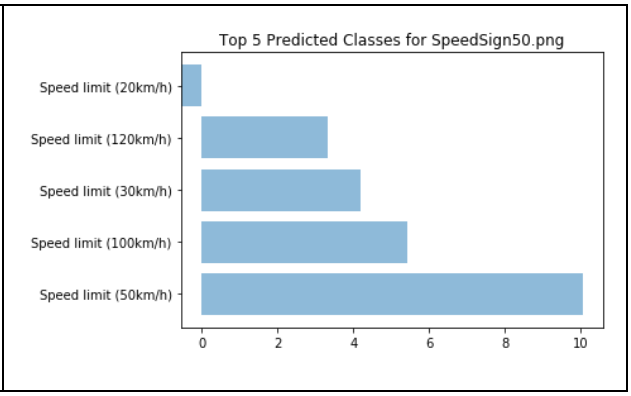
Model Certainty - Softmax Probabilities

The top five softmax probabilities of the predictions on the captured images are outputted. The submission discusses how certain or uncertain the model is of its predictions.

After running the images through the model, it's useful to see the top probable classes for each image. To help with this, I grabbed the top 5 probable classes for each image and plotted them with a bar chart. Ideally the model should have a very high confidence for the top probable class and low confidence for the other probable classes.

Happily this turns out to be the case for the correctly identified signs. For the pedestrians sign that was incorrectly predicted, the correct class was the second highest probable classes which leads me to believe the model is slightly underfitted for this sign class.





Submitted by Kyle Martin