This project trained a convolutional neural network to predict German Traffic Signs. The data was first loaded, then summarized and visualized before training in the model. Summary statistics are given below:

Number of training examples = 34799

Number of validation examples = 4410

Number of testing examples = 12630

Image data shape = (32, 32, 3)

Number of classes = 43

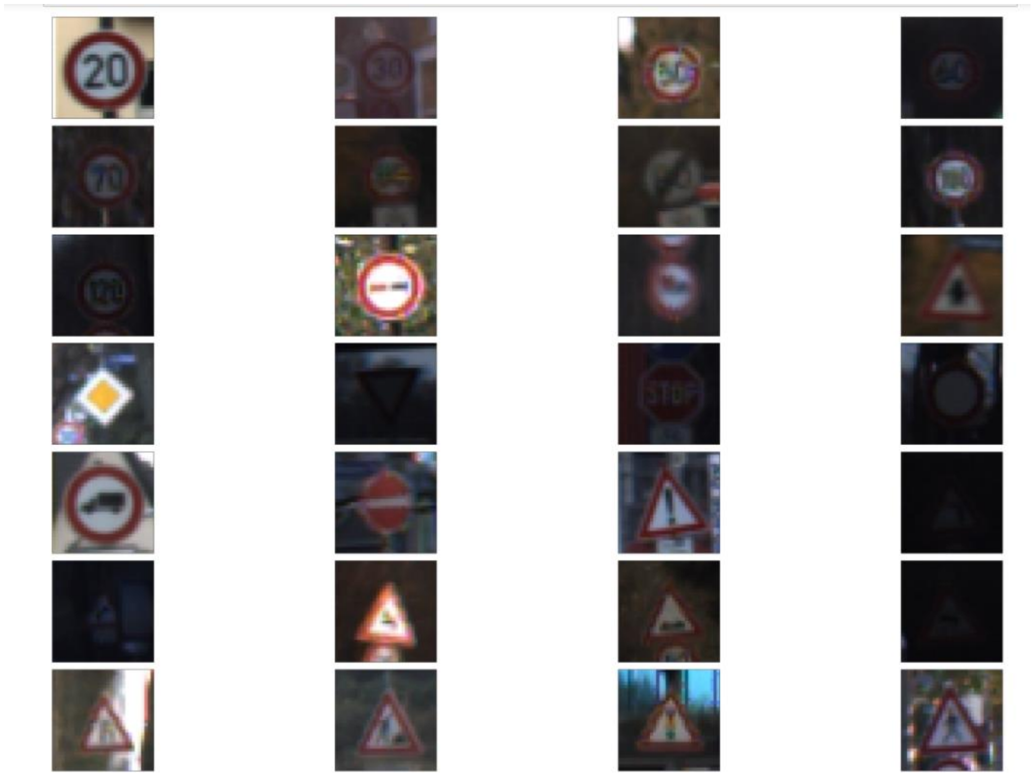
Number of classes in training set only = 43

Number of classes in validation set only = 43

Number of classes in testing set only = 43

Classes: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42}

For each of the 43 types of signs, I showed an image from the training set. This allowed me to observe the variances within the images. Some appear to be taken with better lighting; some appear blurry. I infer this will be a good set to train under. Here are some of the images from the training set.



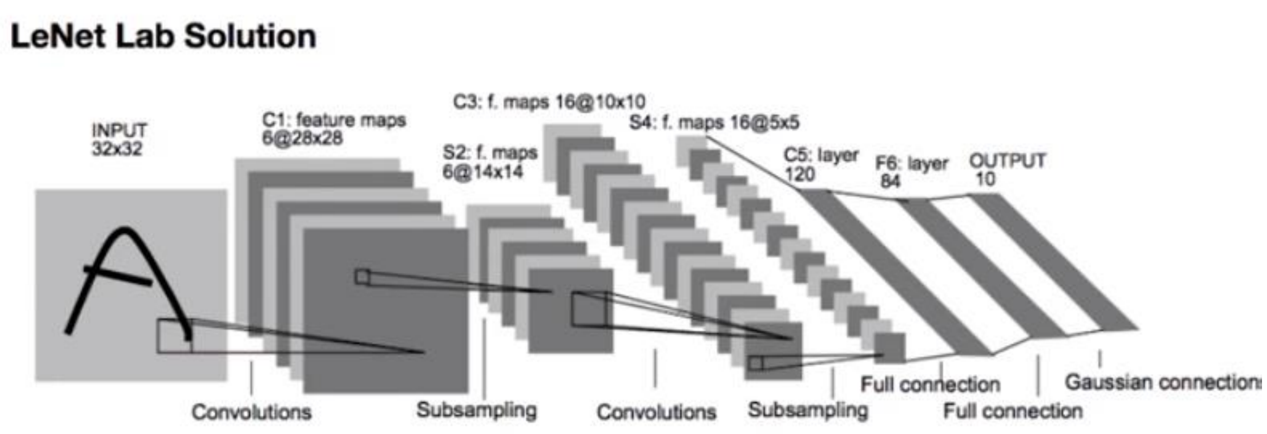
Next, I pre-processed the data. I remember the LeNet architecture working better with the grayscale. I also read in the published baseline model on this problem that the grayscale also provided with higher accuracy. I grayscaled the training and validation sets. I then normalized the grayscaled images so all values are between 0 and 1. This provides consistency for the network to train.

Then, I created the architecture for the training. I used the LeNet architecture. The only exception is that I added dropout. Dropout has been shown to significantly improve predictions in neural networks. It protects from overfitting and relying too heavily on one feature. I trained with a dropout rate of 50%. The published baseline model also included dropout. The LeNet architecture can be seen below.

The first layer in the convolutional neural network took the 32x32 image, ran six 5x5 pads over the image, and translated it into six feature maps of size 28x28. Strides of 1 going across and down were used. A relu activation was used. Next, there was a subsampling layer: a 2x2 box of stride 2 across and 2 down was max pooled throughout each of the six feature maps. The result is now six 14x14 maps. Another convolutional layer was added to the previous layer, turning six 14x14 maps into sixteen 10x10 feature maps by running sixteen 5x5x6 pads over the maps. Strides of 1 going across and down were used. A relu activation was used. Then, another 2x2 max pooling subsampling layer was added with strides of 2 across and 2 down. Now, the output is sixteen 5x5 feature maps. The feature maps were flattened, totaling to 400 features. These features will be fed into my neural network.

Also, there are other features that are fed into the network. The results after the first max pooling layer, the six 14x14 maps underlined above, were 2x2 max pooled once more, so now there are six 7x7 maps. The six 7x7 maps were flattened to 294 features, and fed into the network as well. This is a technique used in the published baseline model.

Now, there is a total of 694 features fed into the network. The 694 features are fed into the first hidden layer of 120 nodes and relu activated. This result is fed into another hidden layer of 84 nodes that is relu activated. This result is fed into the final layer, which has 43 nodes (for the 43 signs). Softmax was used to get the probabilities and an Adam Optimizer was used.



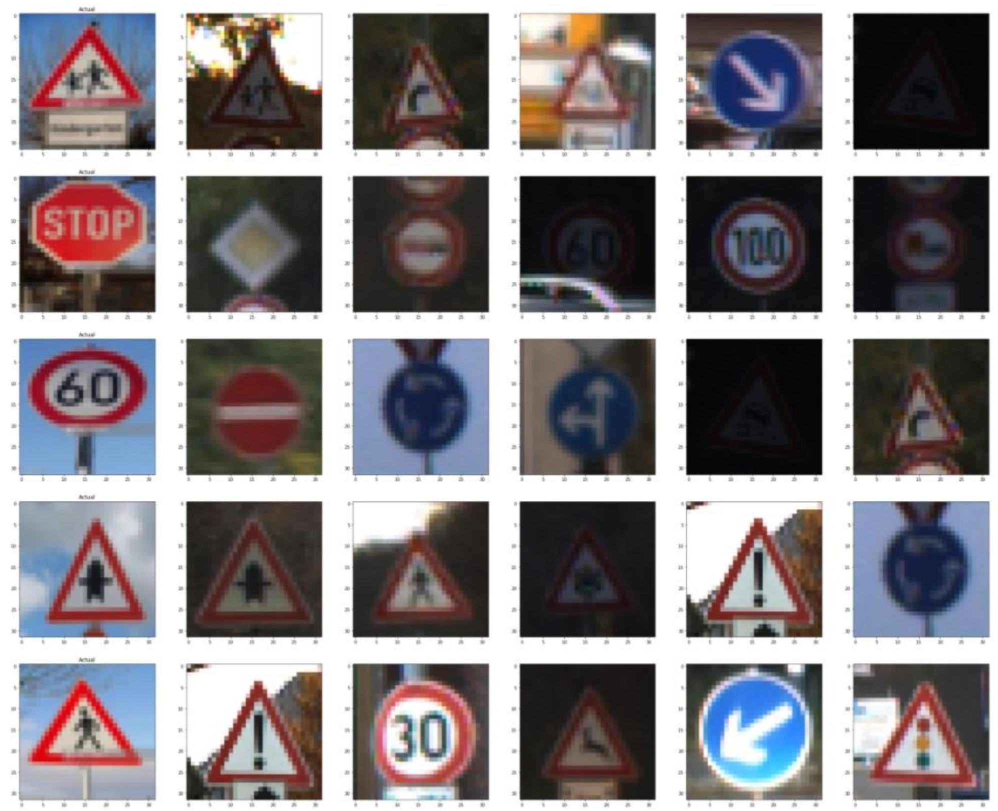
I used an Adam Optimizer. The learning rate was initially set at 0.001, but after 20 epochs, I changed the rate to half as much (0.0005). This decay of the learning rate has been shown to provide better predictions, because it does not allow the learning rate to remain too big to improve accuracy. I also used 30 epochs with a batch size of 128.

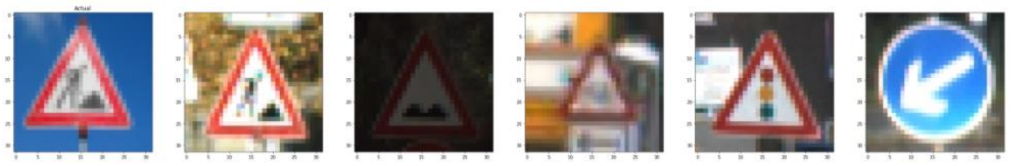
My final validation accuracy was 93.9%. My training accuracy was 99.9%. I felt comfortable enough testing on my training set and got a training accuracy of 93.0%.

I took many approaches to solving this problem. I used a combination of many techniques like l1 regularization, l2 regularization, dropout, grayscaling, augmenting, adding layers, changing the Optimizer. I was really frustrated that I was not getting the results I sought. Then, I decided to read the published baseline model. I used the trick of feeding the results from the first pooling into the fully connected layer. On my first try, I obtained the wanted results.

The LeNet architecture was used for digit recognition. Digits are like traffic signs in many ways. The key differences in numbers are the lines and curves in the numbers, like the lines and curves in the traffic signs. There are also numbers in the traffic signs.

I used six German sign images from the web. I imported the images and predicted the images all in one step. I was fairly surprised with the results. When I compared the actual sign with the predictions. I obtained an accuracy score of only 50% or half. This means three out of the six were incorrect. I initially concluded that my model overfit, but that does not explain the accuracy obtained from the training set. When comparing the top 5 answers to the actual, only 50% of the answers had a top 5 answer. These are startling results. The pictures can be seen here. The actuals are the first column. The predicted are shown next to the actuals, moving from increasing to decreasing probability as it moves to the right.





The results from the six online signs surprised me. I would assume that those signs would have been easy to predict. They are about the same lighting as the training signs. Also, there is just a sign in most of them. Some of the training examples were noted to have background images behind the sign. Also, the model is certain, even when the model is incorrect. The smallest probability is above 86%. The probabilities are shown below.

```
TopKV2(values=array([[ 0.94742841,  0.03566637,  0.01238833,  0.00188266,  0.00108992]], dtype=float32), indices=array(
0, 29, 38, 23]], dtype=int32))
TopKV2(values=array([[  9.27608550e-01,   7.21371323e-02,   2.53961072e-04,
          3.13821289e-07,   3.80373039e-10]], dtype=float32), indices=array([[12,  9,  3,  7, 10]], dtype=int32))
TopKV2(values=array([[  8.60330343e-01,   8.51142406e-02,   5.42926043e-02,
          1.89720929e-04,   1.56791375e-05]], dtype=float32), indices=array([[17, 40, 37, 23, 20]], dtype=int32))
TopKV2(values=array([[  1.00000000e+00,   1.48984669e-09,   6.88723330e-13,
          4.11141905e-16,   2.15074487e-18]], dtype=float32), indices=array([[11, 27, 30, 18, 40]], dtype=int32))
TopKV2(values=array([[  1.00000000e+00,   2.20794028e-09,   3.73319764e-10,
          2.60161532e-12,   1.29080969e-12]], dtype=float32), indices=array([[18,  1, 31, 39, 26]], dtype=int32))
TopKV2(values=array([[  9.98417854e-01,   1.56880717e-03,   7.32142598e-06,
          5.96474501e-06,   6.86892765e-10]], dtype=float32), indices=array([[25, 22, 29, 26, 39]], dtype=int32))
```