Capstone Project 2 - Traffic Sign Image Recognition using Convolutional Neural Network

Milestone Report 2

Model Architecture

The initial model architecture I used is based on the LeNet-5 architecture, which was originated in the 1990s for recognizing handwritten digits. The architecture includes the following layers:

Layer 1: Convolutional Layer
- Filter Size = 5 x 5
- 16 Filters
- 2 x 2 pooling
- ReLu Activation

Layer 2: Convolutional Layer
- Filter Size = 5 x 5
- 36 Filters
- 2 x 2 pooling
- ReLu Activation

Layer 3: Fully Connected Layer
- 516 Outputs
- ReLu Activation

Layer 4: Fully Connected Layer
- 360 Outputs
- ReLu Activation

Layer 5: Fully Connected Layer - Output Layer
- 43 Outputs (1 for each class)

The cost function optimized is the tensorflow softmax cross-entropy with logits function. The optimization method is the Adam Optimizer with a learning rate of .0001.

The model was trained on batch sizes of 64 images using an optimizer function. Each iteration in the optimizer function selects a random sample of 64 images from the training data, and passes them through the neural network and updates the model weights, and displays the training accuracy after each 100 iterations.

Results

The results of the base architecture were an overall accuracy of about 92% - 94% achieved after about 20,000 - 30,000 training iterations.

I tried different parameters for the number of filters per convolutional layer, the number of outputs in the fully connected layers, as well as the number of images per batch and the learning rate. None of these changes had much effect on the overall accuracy.

Second Model Architecture

With the first architecture peaking around 92-94 percent, the next step is to try a better overall architecture.

The second model architecture I used is based on the VGGNet architecture, which was first introduced in 2014. The VGGNet is much deeper than the LeNet-5 architecture, typically having 16-19 layers. VGGNet uses 3x3 filters, and 13 convolutional layers, with pooling occurring after layers 2, 4, 7, 10, and 13. These layers are followed by 2 fully connected layers and the output layer.

Since I am working with smaller images, I was not able to make the network as deep due to the many pooling layers. Therefore, I simplified the VGGNet Architecture to use 3 x 3 filters, and 6 convolutional layers, with pooling on layers 2, 4, and 6.

Layer 1: Convolutional Layer
- Filter Size = 3 x 3
- 32 Filters
- No Pooling
- ReLu Activation

Layer 2: Convolutional Layer
- Filter Size = 3 x 3
- 36 Filters
- 2 x 2 pooling
- ReLu Activation

Layer 3: Convolutional Layer
- Filter Size = 3 x 3
- 64 Filters
- No Pooling
- ReLu Activation

Layer 4: Convolutional Layer
- Filter Size = 3 x 3
- 64 Filters
- 2 x 2 pooling
- ReLu Activation

Layer 5: Convolutional Layer
- Filter Size = 3 x 3
- 128 Filters
- No Pooling
- ReLu Activation

Layer 6: Convolutional Layer
- Filter Size = 3 x 3
- 128 Filters
- 2 x 2 pooling
- ReLu Activation

Layer 7: Fully Connected Layer
- 512 Outputs
- ReLu Activation

Layer 8: Fully Connected Layer
- 256 Outputs
- ReLu Activation

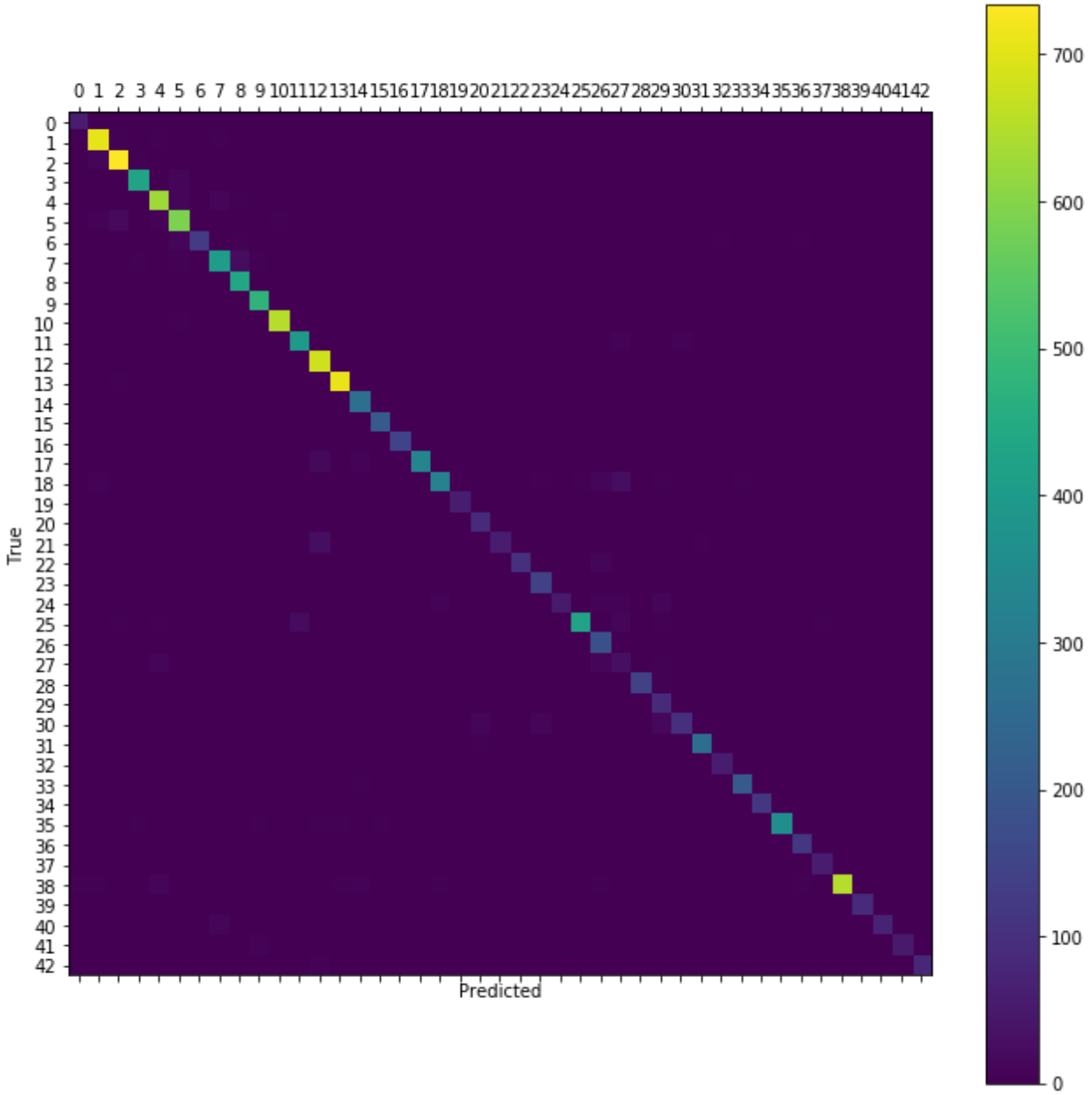
Layer 9: Fully Connected Layer - Output Layer
- 43 Outputs (1 for each class)

I used the same optimizer and cost function in for the second architecture. Only the neural network layers and parameters listed above differed from the first architecture.

Results - 2nd Architecture

The second architecture is showing improved results over the first, with an accuracy of about 96% after 30,000 training iterations. I have not yet tried tuning the hyperparameters such as the number of filters per layer, or the output size for the fully connected layers.

The confusion matrix below the number of images classified correctly in the test data. Since the test set is imbalanced much like the training set originally was, there are more correct examples of the images that are more common, showing in yellow. This visual does not show any major misclassifications.



Final Steps

One final step is to try to tweak the parameters of the second architecture to see if the results can still be improved even further.

Also, while I have been using overall accuracy, I would like to break that down into accuracy by class to see if there are particular classes that the model is struggling more with.

The one last thing that could be done to try to improve the results would be to do more data preprocessing, such as histogram equalization. Another option would be to try adding more

variability to the additional training images that were generated. The generated images had a small random rotation added, but they could also have small amounts of stretching or skewing.