Project 2

Traffic Sign Recognition Classifier

Goals

- Design a Neural network in Tensorflow to recognize traffic signs.

Reflections

My pipeline consisted of the following steps:

1. Load the training, validation and test data
2. Print summary info about the dataset
3. Show randomly chosen 10 images from the training data with its label
4. Preprocess the training data: Augmenting the training data by adding +15 and -15 degrees rotated images for every image. Also normalizing the color values of RGB from 0 to 255 into -0.5 to +0.5 in order to prevent the weights from going too large during training.
5. Setup Tensorflow using a modified layers size of LeNet architecture as the following:

Layer 1: Convolutional. The output shape should be 28x28x18.

Activation. ReLu activation function.

Pooling. The output shape should be 14x14x18.

Layer 2: Convolutional. The output shape should be 10x10x48.

Activation. ReLu activation function.

Pooling. The output shape should be 5x5x48.

Flatten. Flatten the output shape of the final pooling layer such that it's 1D instead of 3D. The easiest way to do is by using `tf.contrib.layers.flatten`, which is already imported. This will yield 1200x1 outputs.

Layer 3: Fully Connected. This should have 360 outputs.

Activation. ReLu activation function.

Layer 4: Fully Connected. This should have 172 outputs.

Activation. ReLu activation function.

Layer 5: Fully Connected (Logits). This should have 43 outputs.

6. Setting up a training pipeline (softmax cross entropy with logits, finding out the mean which is the loss operation, and setting up the optimizer as the Adam optimizer)

7. Defining the evaluate function

8. Training the model with 100 epochs which gave **Training accuracy of 100%** and **Validation accuracy of 96.1%.**

9. Plotting the Training and validation accuracies for all the 100 epochs.

10. Evaluate the performance on the test set which gave **Test Accuracy of 94.7%**

11. Plotting the confusion matrix between the true labels and the predicted labels

12. Loading 5 images from the internet and testing them on the trained model, which gave 80% accuracy. All were predicted correct but one because that one wasn't a known sign for the trained network. That sign probably not part of the German traffic signs. But the prediction even though wasn't correct but pretty close.

13. Printing out the top 5 softmax probabilities for those 5 images tested in the previous step.

The training of the LeNet network was done with 100 epochs and batch size of 128. After 50 to 60 epochs the training reached its highest training and validation accuracy. The Adam optimizer was used, since it is known for better performance for such types of neural networks. I have experimented a bit with dropouts but did not give better accuracy, so did not use it.

Experimented with learning rate but found that the best performance and accuracy was on around 0.001. Tried to make the learning rate changed from higher number (0.01) then after every 10 epochs decreased by ten times less, but that did not give me better accuracy, so returned to use the 0.001 learning rate. Some other students found that decreasing the sigma gave them better accuracy; I did not tried that yet.

<u>Potential shortcoming and possible improvements:</u>

One way to increase the accuracy of the network is to augment even more the training dataset with more types of augmentation. Like geometrical transformation and illumination augmentation. That is because traffic signs are with variable illuminations and seen with different angle views.

Others achieved better performance with dropping out the color information and using the grey color space instead. I did not go with that root since I thought that colors are giving important info on the type of the traffic signs, but they argued that the shape alone are enough and without the RGB color the input will be with 3rd less input dimension which will enhance the performance and prediction accuracy. I did not have time to test that.

Some other tried to use different networks other than LeNet like the new Dense Net where each layer will get its input not only from the output of its previous layer but from the outputs of all the previous layers. This will prevent the known problem of gradient decaying which appears when the gradient descent will go through a lot of layers.

I tried to use dropout but did not give better accuracy (it was less by few percents). Probably that was because my network was not large enough to make the regularization techniques

like L2 regularization or droput as useful like where if we had a larger network. These could in theory prevent overfit of relatively large network for small datasets.