Self-Driving Car Engineer Nanodegree

Deep Learning

Project: Build a Traffic Sign Recognition Classifier

This project deals with classifying Traffic Signals using a Deep Learning Technique (Convolutional Neural Network).

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.

- Preprocessing the data.

- Design, train and test model architecture.

- Use the model to make predictions on new images.

- Analyze the softmax probabilities of the new images.

Step 2: Explore, Summarize and Visualize the Data

- The data is explored by studying the dimension of arrays of different types of samples.

- Then respective histogram is plotted to study the frequency of each of the classes in three types of data.

- Grey Scaling has not been included because it does not increase the efficiency of the model.

Step 3: Preprocessing the Data

- This step involves normalizing the data and assigning the respective labels of y.

- The data is shuffled and then the some part of the training data is split into the validation set.

- The dimensions of the data are published at the end of this step.

- The gap of -0.5 to 0.5 is chosen because this ensures the intermediate numbers generated by matrix multiplication will not grow to be too large.

- One hot encoded the labels because Classifying traffic signs involves dealing with multiple classes where binary classification is extended for all the classes.

- Grey Scaling has been excluded as it did not increase the accuracy of the model.

- To increase the amount of training data, i transformed and augmented the image, but accuracy was not the best and it took more than 15 minutes for 1 epoch.

Step 4: Designing the Neural Network Architecture

This step is defining the architecture of Neural Network which has very similar values of that of Classroom's LeNet Neural Network Architecture. The hyperparameters are chosen accordingly by varying the learning rates, epochs and changing the Optimizer method.

The overview of the architecture is as follows:

- Used LeNet architecture.

- Conv layers having 6 and 16 filters of size 5x5 on 3 channels having strides of (1,1,1,1).

- Max pool layer of strides (1,2,2,1)

- 3 Fully connected layers connecting 400x120x84x43 nodes

- InputLayer: Size (N, 32, 32, 3)

- N normalized images of size (32, 32, 3)

- ConvLayer1: Size (N, 28, 28, 6)

- Filter bias values of size (6) and weight values of size (5, 5, 3, 6) with strides (1,1) on height and width of image and VALID padding will produce N matrices of size (28, 28, 6).

- ReluLayer1: Size (N, 28, 28, 6)

- MaxPoolinglayer1: Size (N, 14, 14, 6)

- Applied max pooling of size (2,2) with strides (2,2) on height and width of an image proudces N matrices of size (14,14,6)

- ConvLayer2: Size (N, 10, 10, 16)

- Filter bias values of size (16) and weight values of size (5, 5, 6, 16) with strides (1,1) on height and width of image and VALID padding will produce N matrices of size (10, 10, 16).

- ReluLayer2: Size (N, 10, 10, 16)

- MaxPoolinglayer2: Size (N, 5, 5, 16)

- Applied max pooling of size (2,2) with strides (2,2) on height and width of an image proudces N matrices of size (5,5,16)

- FCLayer1: Size (400, 120)

- FCLayer2: Size (120, 84)

- FCLayer3: Size (84, 43)

This is the final layer that produces values for 43 class labels.

Hyperparameter Tuning

- Used AdamOptimizer with learning rate 0.001.Tried 0.0001 and 0.005

- Batch size is 128. Tried with 64 and 128 but 128 looks to be the best for accuracy.

- Ran on 20 epochs. Tried with 5,10,15 and 20.

- Dropout 50% on fully connected layer.

Steps followed for designing the Neural Network Architecture

- Normalized the dataset

- One hot encoded the labels

- Splitted training set into train and validation set

- Then tried augmented data but and it does not improve the performance.

- Tried several epochs and batch size configuration but did not improve.

- Tried initially with inception module but took very long time to run the model. Choosing the parameters were very difficult. So shifted to traditional LeNet architecture.

- Finally Implemented LeNet architecture for initial run and ran it for batch_size=128 and epochs=20. Results are validation accuracy = 98% and test accuracy = 90%.

Conclusion:

LeNet is already a proven architecture to classify images with a very small parameter values which fits well and thus does a pretty good job in combination with dropout for overfitting prevention.Overall the accuracy for all the validation, test and downloaded image dataset are pretty good with this simple yet effective architecture.

Step 5: Testing on New Images

- I have collected around 9 images from Google Search.

- The images are converted to 32x32x3 height and width scale.

- The images include that of Road Work, Stop sign, No passing, Left Ahead, Yield and others.

- The images are visualized as follows:

Results

- 8 out of 9 images are classified correctly which results in 88.9% accuracy.

- It is compared to test data accuracy of 90%. The validation data accuracy is 98%.

- The last image is wrongly classified as No passing of vehicles over 3.5 metric tons.It might be because of the background of the image which is not clear and image is wrongly classified.

Analysis of Softmax Probabilities:

- The images correctly classified are very certain yielding 100% probable.

- The yield is correctly classified but it is only 69.21% probable.

- I have chosen k value in the top_k function to be 2 to get 2 probabilities answer.

- I guess if it is not finding out the classification, it tends to produce no passing for vehicles for 3.5 metric tons.

- Some of the images are very simple to classify and thus classifier provides good accuracy.

- Future work is to work with very distinctive traffic signs.