ECE 4200: Final Report

Name: Kranthi Kumar M

ID: km853

Kaggle ID: kranthi017

For the data, first I had to use h5py library to load the data.hdf5 data and separate into train and test data. Using Pandas library I loaded the train\_labels.csv file and encoded the labels into integers between 0 and 9. As described in the IEEE research paper I started off with extracting features of the IQ dataset. Primarily I converted the 1024 X 2 data to 1024 X 1 data by converting into complex, I + jQ. Then applied fft on the complex data. Then I normalized the data by dividing the real and complex parts into their respective maximum values. Then, as suggested, I extracted 36 features of higher order moments of the data. More specifically I used the formulas

, 

I extracted M20, M21, M22, M40, M41, M42, M43, M60, M61, M62, M63, C20, C21, C22, C40, C41, C42, C43, C60, C61, C62, and C63. Using these features I applied the following models.

* SVM: linear, poly (degree:2, 3, 4), Gaussian
* Bagging: decision trees
* Boosting: Decision trees
* Gradient boosted trees, XG boost

However, none of these models gave accuracy higher than 40% with my extracted features. Going through the IEEE paper’s results suggested these models fail for S/N ratio lower than 5 dB. While neural network models do better for the same SNR values. I then ventured onto using neural networks. As per the paper’s suggestion I used Convolutional Neural Networks for training the models. The layers of which were the same as prescribed in the paper; 7 convolutional layers combined with maxpooling to half the data length at every stage using relu activation followed by two fully connected layers with scaled exponential linear unit activation. In between I also employed dropout layers to minimize overfitting of the data. For this model, I simply fit in the raw I/Q data initially, and then used normalized data (to unit variance) later on. Within this model I changed parameters to find the best accuracy fit with the given data. That is, I varied epochs between 70 – 200 with various steps, changed the kernel size and filter size of the convolutional layers, varied the dropout ratio from 0.1 to 0.5, added padding, with and without bias vector and bias regularization, varied the loss function (categorical cross entropy and such), varied the batch size etc. Finally, I could get the best fit model at 150 epochs with 0.25 dropout, added padding, without bias regularization and using normalized data.