ML Assignment 4 Report

Harish Chandra Thuwal
2017MCS2074

1. FIXED ALGORITHMS

A. K-Means

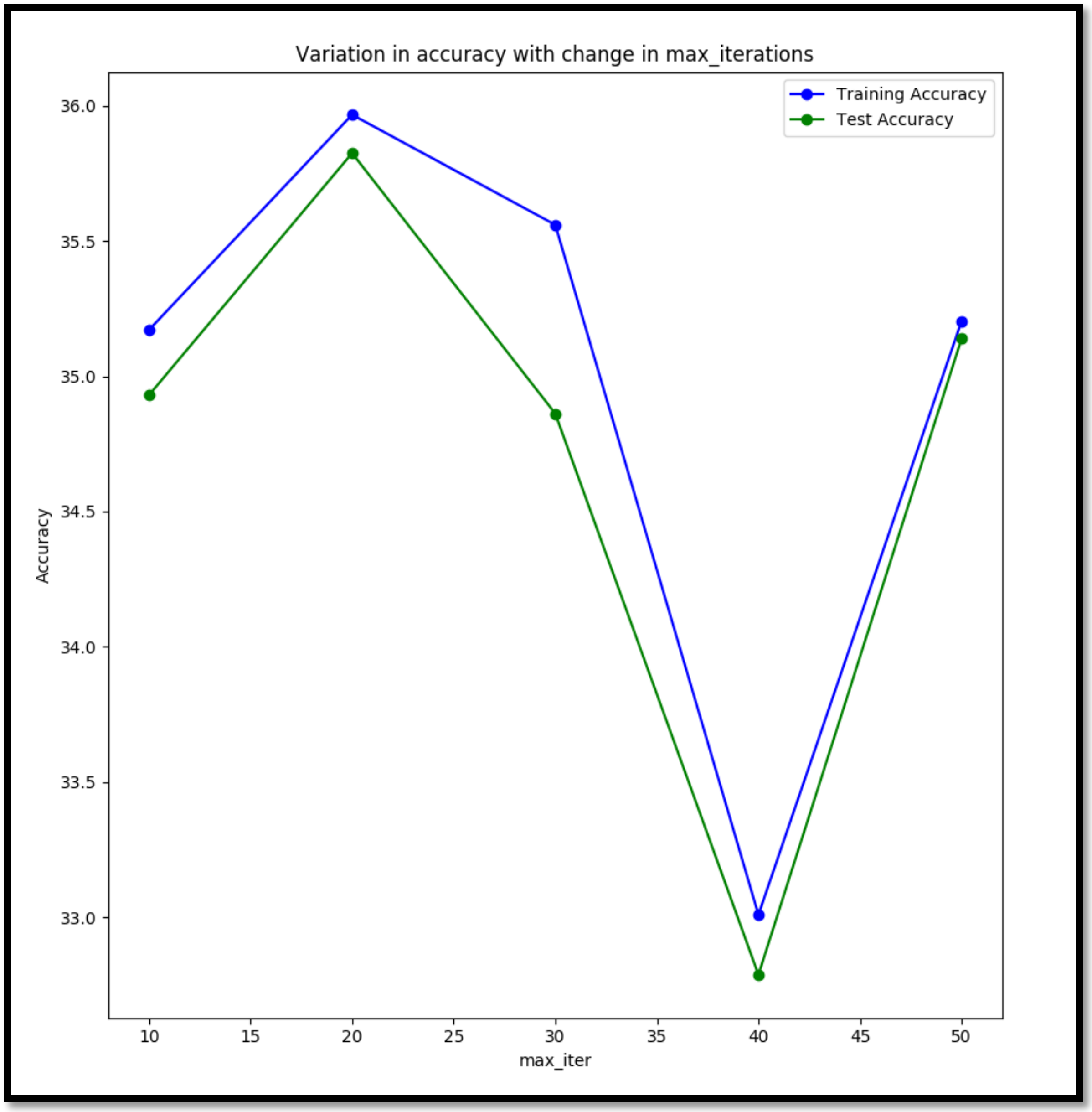
i) Implement K-Means using scikit learn with

n_init = 10, *n_clusters* = 20, rest set to default.

Accuracy of Clustering: 35.622 %

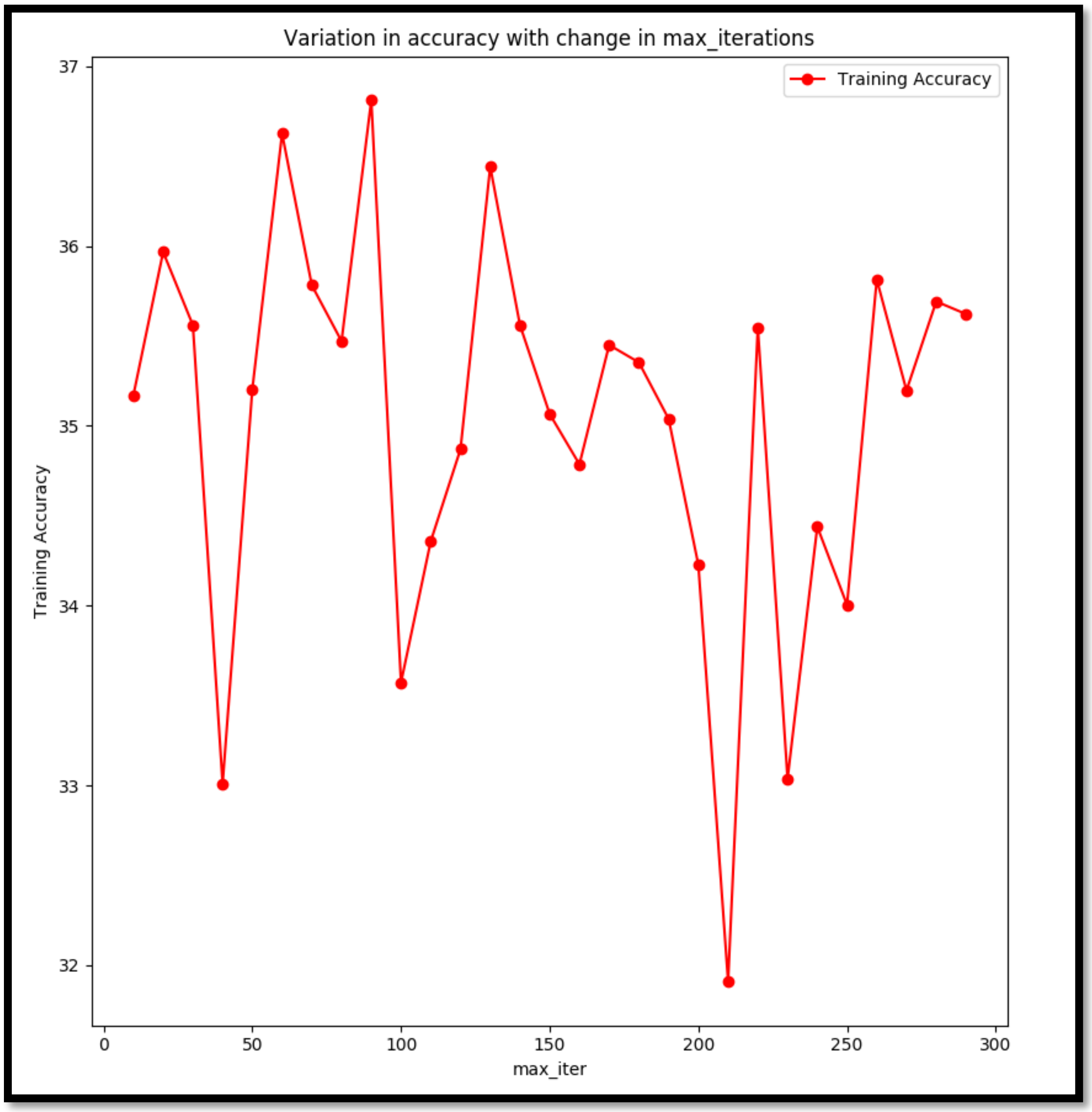
ii) Test accuracy (from Kaggle): 36.322 % [max_iter = 300(default)]

iii) Variation of accuracies with "*max_iter*" parameter



As can be seen from the previous plot, there seems to be no co-relation between the number of maximum iteration and the training/test accuracy. The training accuracy increases first followed by a sudden drop and then increases again as we increase maximum iterations from 10 to 50.

One thing that can be observed from the above plot is the fact that the accuracy on the test set closely follows the that of the training accuracy. So, I ran the kmeans algorithm varying *max_iter* from 10 to 300 with a step of 10, calculating train accuracy at every step. Following plot was obtained.



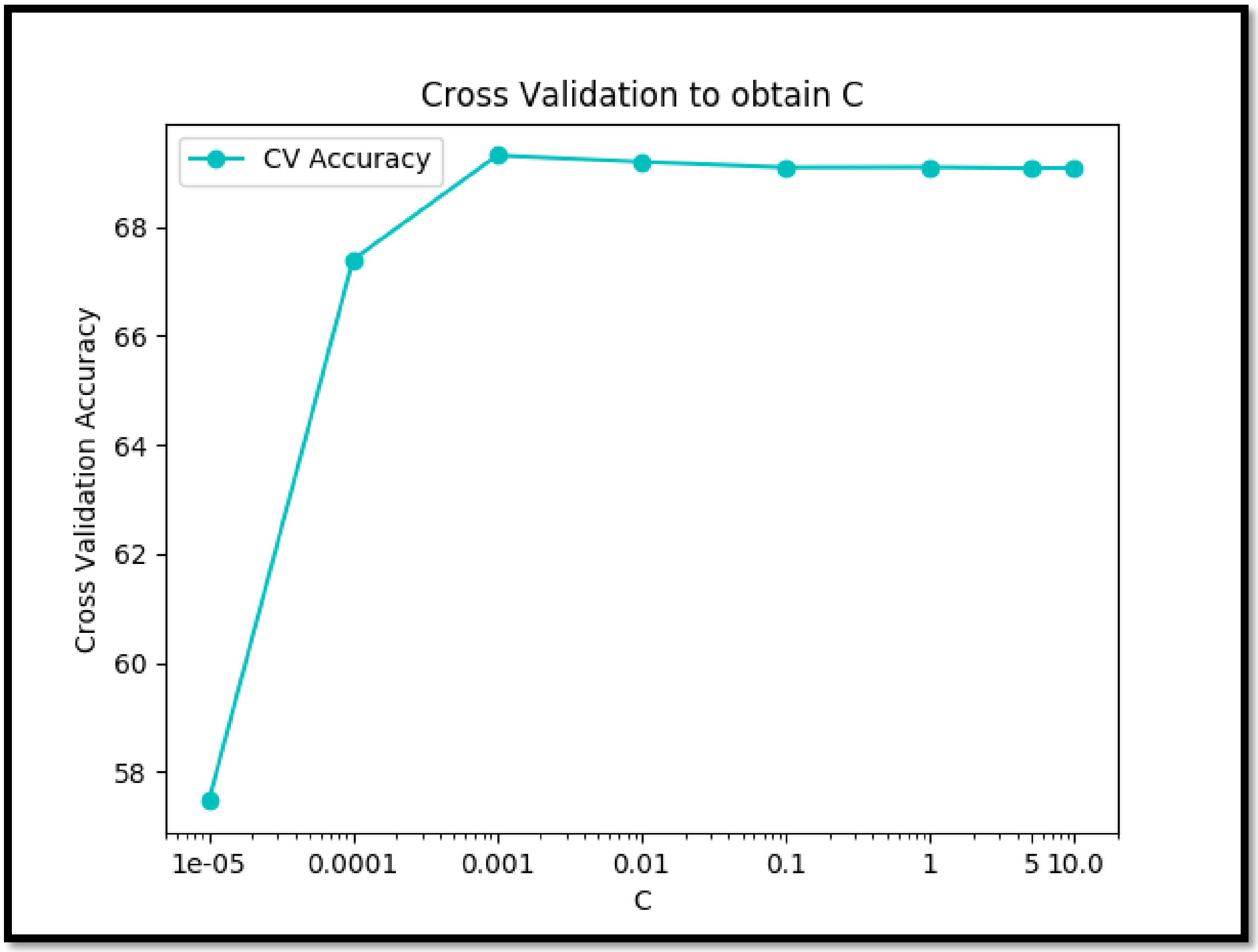
Based on above plot maximum training accuracy was obtained when max_iter was set to 90. Test accuracy using this model (max_iter = 90) was **36.685 %**

B) PCA + SVM

Scaled the data using ***sklearn.preprocessing.scale*** (which performs standardization of the data) before applying PCA.

Both the test and train data were transformed were projected to a 50-dimensional space using PCA.

Cross Validation on the set ***{1e-5, 1e-3, 1e-2, 1e-4, 1e-1, 1, 5, 10.0}*** *was performed to obtain optimum value of parameter C for multi-class linear SVM classifier.*



As can be seen from the above plot maximum cross validation accuracy of (69.31) was obtained at C = 0.001. Thus, Linear SVM model was trained by setting the parameter C = 0.001 and the accuracy obtained by this model on the train and test set are **70.12 %** and **69.842 %** respectively.

It's clear that PCA followed by SVM gave a substantial improvement as compared to the K-means algorithm. The best test set accuracy obtained using linear SVM is almost double to that obtained by K-means.

Also, the accuracy obtained using PCM + SVM were slightly better than SVM alone. Using PCM reduced the dimensionality and therefore the training time of SVM.

Few other configurations were also implemented results of which are as follows.

Kernel	Parameters	Trainset Accuracy	Test Set Accuracy
Linear	Default	64.368	63.859
Linear	C=0.001 (using CV)	70.12	69.842
Rbf	decision_func = "ovo"	99.064	81.682
Rbf	decision_func = "ovr"	99.087	81.795

Rbf kernel SVM boosted the accuracy to 81.795 % which is 125% more than the K-means algorithm. Decision boundary "ovo" or "ovr" made little to no difference to the final accuracies but "ovr" version was significantly faster than the "ovo" version as later involves O(m^2) classifiers as compared to just O(m) classifiers required by the former. Here m is the number of classes.

C) Neural Network (NN)

Framework Used: Pytorch
Implemented a fully connected NN architecture with one hidden layer Followed by one output layer.

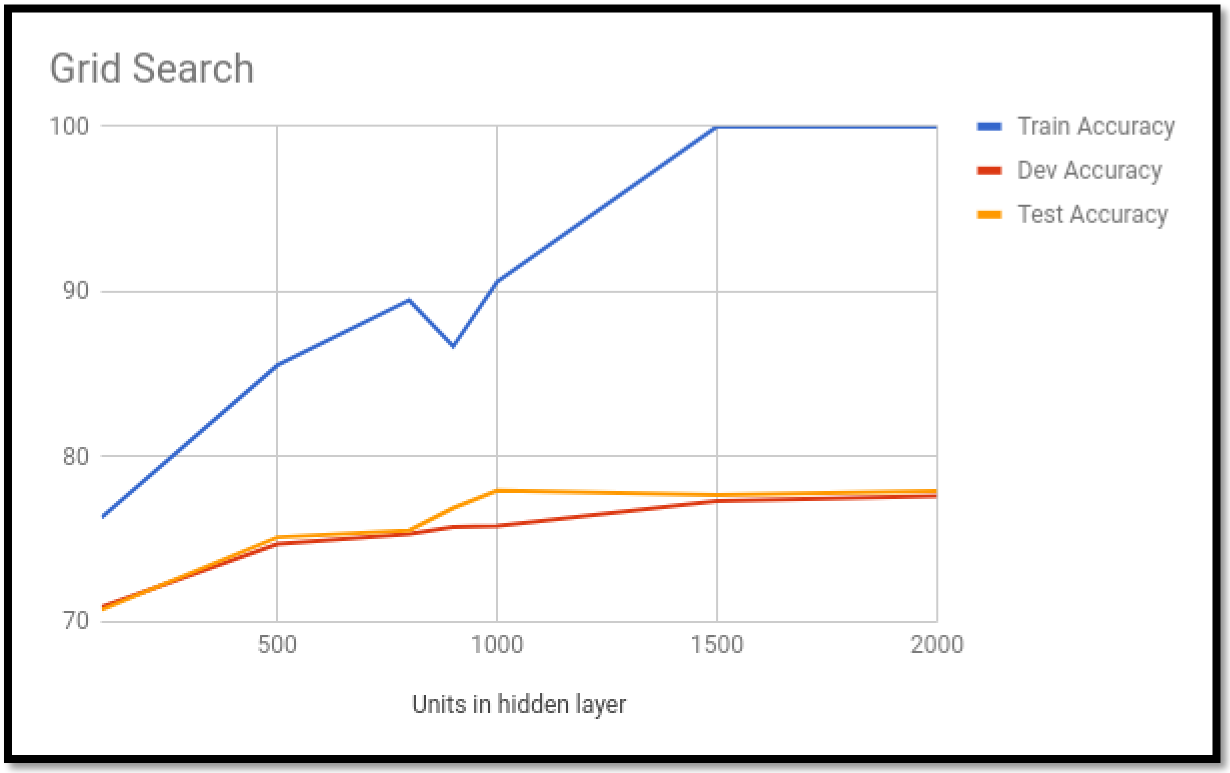
(0): Linear: 784 x hidden
(1): Sigmoid
(2): Linear: hidden x 20

Followed by a *softmax*.
Loss: Negative of log likelihood
Optimizer: adam

Grid Search to find best number of hidden units.

Units in hidden layer	Train Accuracy	Dev Accuracy	Test Accuracy
100	76.32	70.90	70.735
500	85.55	74.72	75.130
800	89.48	75.33	75.547
900	86.69	75.74	76.90
1000	90.61	75.81	77.952
1500	100.00	77.33	77.680
2000	100.00	77.64	77.940

Although the dev accuracies continued to increase on increasing the number of hidden layers but the test accuracies reached a peak accuracy of 77.952% at 1000 hidden units. After that the test accuracy started to decrease. This can be attributed to the fact that on further increasing the number of units in hidden layer to 1500/2000 the model overfitted to give 100 percent training accuracy.



The accuracy obtained by the neural model containing **just a single hidden** layer was better than both the K-Means and Linear SVM (36.685, 69.842)!!

However, using just a single hidden layer model wasn't able to surpass the Rbf SVM (81.795).

D) Convolutional Neural Network (NN)

Framework Used: Pytorch

CNN Architecture

```
Dropout p=0.2
conv:
   Conv2d: 1 x num_filters x kernel_size
   ReLU
   MaxPool2d: kernel_size=(2, 2), stride=(2, 2))
   Dropout: p=0.2
intermediate_fc: [ (num_filters * 14 * 14) x fc_size ]
output_layer: fc_size x 20
```

Dropouts for regularization. Padding in Conv2d to ensure no decrease in image dimensions by Convolution Layer. MaxPool layer with kernel size 2x2 to reduce the image to its 1/4th. Finally, an intermediate fully connected layer and the output layer followed by a softmax.

Loss: Negative of log likelihood
Optimizer: adam
Hyperparameters: num_filters, kernel_size, fc_size

Performed a grid search over the following parameter Values:

num_filters:	[3,5,16,32,64,128]
kernel_size:	[3,5,9]

Best Parameters obtained by grid search:
[num_filters: 128, filter_size: 9, fc_size: 32]

The following table shows the best hyperparameters and corresponding accuraies for a fixed value of "num_filters":

Num_filters	Kernel_size	fc_size	Train Accuracy	Dev accuracy
3	3	32	65.37	72.92
5	5	64	72.21	77.77
16	5	32	78.56	82.15
32	5	32	79.47	82.58
64	5	32	80.79	82.86
128	5	32	82.69	82.75
128	9	32	85.04	83.34

The best set of parameters gave an accuracy of **83.162 %** on the test set which is higher than any of the previous method used.

One important observation is the fact that with just 5 filters of kernel size 5 CNN architecture gave a dev set accuracy of 77.77 %. To achieve similar performance in the part B the fully connected NN architecture required a whopping 1000 hidden units. This shows that CNN can perform as good (and better) as NN with just a fraction of the number of parameters required by NN.

E) Overall Comparison

K-Means algorithm failed miserably at the given task and was only able to achieve a maximum train and test accuracy of 36.812% and 36.685%.

PCA + Linear SVM performed significantly well as compared to the K-Means giving a 69.842 test accuracy. Both the K-Means and SVM took about the same time to train but the later performed significantly well.

Note: PCA + Rbf Kernel gave 81.795 test set accuracy. Took almost twice the time as compared to Linear SVM

Fully Connected NN with just a single hidden layer of 100 units was able to outperform the Linear SVM by about 7-8 percent. Also, the time taken by the NN network to train was at least similar if not less than of that of Linear SVM. (On GPU it takes little to no time as compared to SVM)

With the CNN architecture and hyperparameter tuning, CNN outperformed all other methods giving an accuracy of 83.34%. Also, although CNN took lot more time per epoch than NN but was able to achieve similar results in less number of epochs with only a fraction of number of parameters of NN.

2. KAGGLE COMPETITION

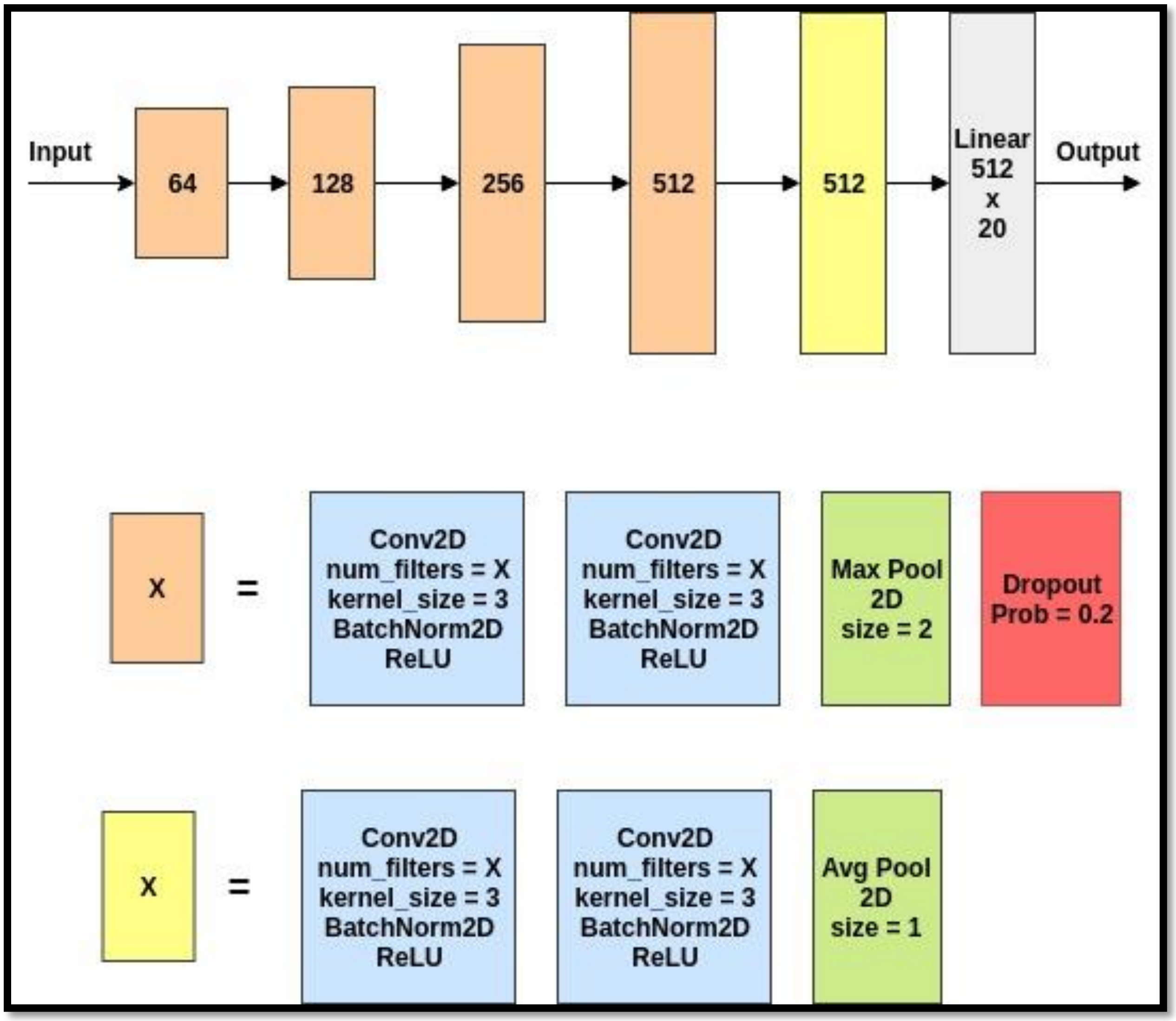
Link to model: https://drive.google.com/open?id=1YxAjua8Re-x7WBJgciTQbW_QohTqEVG5

Libraries Used:

- ***python3***
- ***pytorch***: Deep Learning Framework
- ***sklearn***: for splitting train into train and dev set, accuracy_score
- ***tqdm***: to display progress bar
-

ARCHITECTURE

The following architecture is used and is inspired from VGG13 model. Let's Call it **VGG13_hc.**


Input
64
128
256
512
512
Linear 512 x 20
Output
X
=
Conv2D num_filters = X kernel_size = 3 BatchNorm2D ReLU
Conv2D num_filters = X kernel_size = 3 BatchNorm2D ReLU
Max Pool 2D size = 2
Dropout Prob = 0.2
X
=
Conv2D num_filters = X kernel_size = 3 BatchNorm2D ReLU
Conv2D num_filters = X kernel_size = 3 BatchNorm2D ReLU
Avg Pool 2D size = 1

Vgg13_hc differs from original vgg13 model in the following ways:

1. Dropout Layer after every Max Pool Layers.
2. No Max Pool Layer in the Last Conv2D block.

PREPROCESSING

Normalize the train and test data to zero mean and unit variance

data = (data – data.mean(axis=0)) / data.std(axis=0)

NOTE: **Log Softmax** is performed over the output of the architecture and **Negative Log Likelihood Loss** is used as the Loss Function.

HYPERPARAMETERS:

1. Train on complete trainset for 10 epochs.
2. Batch_size = 64
3. Adam Optimizer, learning rate = 0.001
4. Use Data Loader with shuffle = True (pytorch)

Using The code uploaded on moodle:
Train and test set folders should be inside a directory dataset. The directory dataset should be alongside the kaggle.py file

To Train: *python3 kaggle.py -t model_file_name*
This will train the model above describe model for 10 epochs and save it as "model_file_name"

To Predict: python3 kaggle.py -p model_file_name output_file_name
This will load the model saved as "model_file_name" and use it to predict the test data and dump the labels in the kaggle format as "output_file_name"