**CMSC 828C: Statistical Pattern Recognition**

**Project 2**

***SVM with PCA/LDA Reduction and CNN***

**Sneha Nayak (UID: 115586284)**

**November 16, 2019**

**Introduction**

This project involves the implementation of

1. SVM with linear, polynomial and RBF kernels with PCA and LDA reduction
2. Convolution Neural Networks based on my own architecture

The data set that we have used for this project is the Fashion MNIST dataset with 6000 training and 10,000 test images. The image size is 1x28x28.

We first deduce the SVM’s accuracy in the following steps:

1. then subsequently find its accuracy with Principal Component analysis
2. followed by Linear Discriminant Analysis
3. We compare these accuracies and study them.

This is done in **Chapter 2**.

Then we create a neural network and explain its architecture in **Chapter 3**. Here we tweak various parameters and compare the accuracies accordingly.

For execution of codes, please follow steps in **Readme.txt**

**Chapter 1**

Dimensionality Reduction

Principal Component Analysis

The code for the implementation of PCA is in pca.py.

Here we use scikit.learn inbuilt library to implement PCA. After testing multiple times, the ideal number of principal components was chosen to be 50, since this gives the most accuracy in case of SVM and also since 50 principal components give us enough information to give an overview of the entire dataset.

Linear Discriminant Analysis

The code for the implementation of LDA is in lda.py.

Here we use scikit.learn inbuilt library to implement LDA

It is seen that for values of dimensions less than 9, the accuracy begins to detriment in both the cases.

Values above 9 tend to give errors with the inbuilt LDA library since the value of dimension should be min (features, classes-1).

**Chapter 2**

SVM

To perform SVM, we use python’s inbuilt library, scikit.svm.

In order to reduce the computational time, we perform dimensionality reduction.

We first center the data about the mean and then perform the following dimensionality reductions.

For svm we first perform the following:

1. LDA Reduction
2. Linear SVM: We find that the accuracy turns out to be **81.69 %**
3. Linear Kernel SVM: We find that the accuracy turns out to be **82.53 %**
4. RBF Kernel SVM: We find that the accuracy turns out to be **83.7 %**
5. Polynomial Kernel SVM: We find that the accuracy turns out to be **83.37 %**
6. PCA Reduction

We then move on to carry out dimensionality reduction by deducing 45 Principal components and then projecting the data onto these 10 principal components.

1. Linear SVM: We find that the accuracy turns out to be **81.84 %**
2. Linear Kernel SVM: We find that the accuracy turns out to be **83.65 %**
3. RBF Kernel SVM: We find that the accuracy turns out to be **87.76 %**
4. Polynomial Kernel SVM: We find that the accuracy turns out to be **87.44 %**

Analysis

We see that normalizing and centering the data about the mean for the training and test data gives us better results in the case of PCA and LDA as opposed to not normalizing them.

Not normalizing the data has no effect on SVM in the case of LDA. However, in the case of PCA, it gives very poor results and takes a very long time to fit the data. The accuracy when the data is not normalized and centered in the case of RBF kernel with PCA reduction gives out an accuracy of 10.76%.

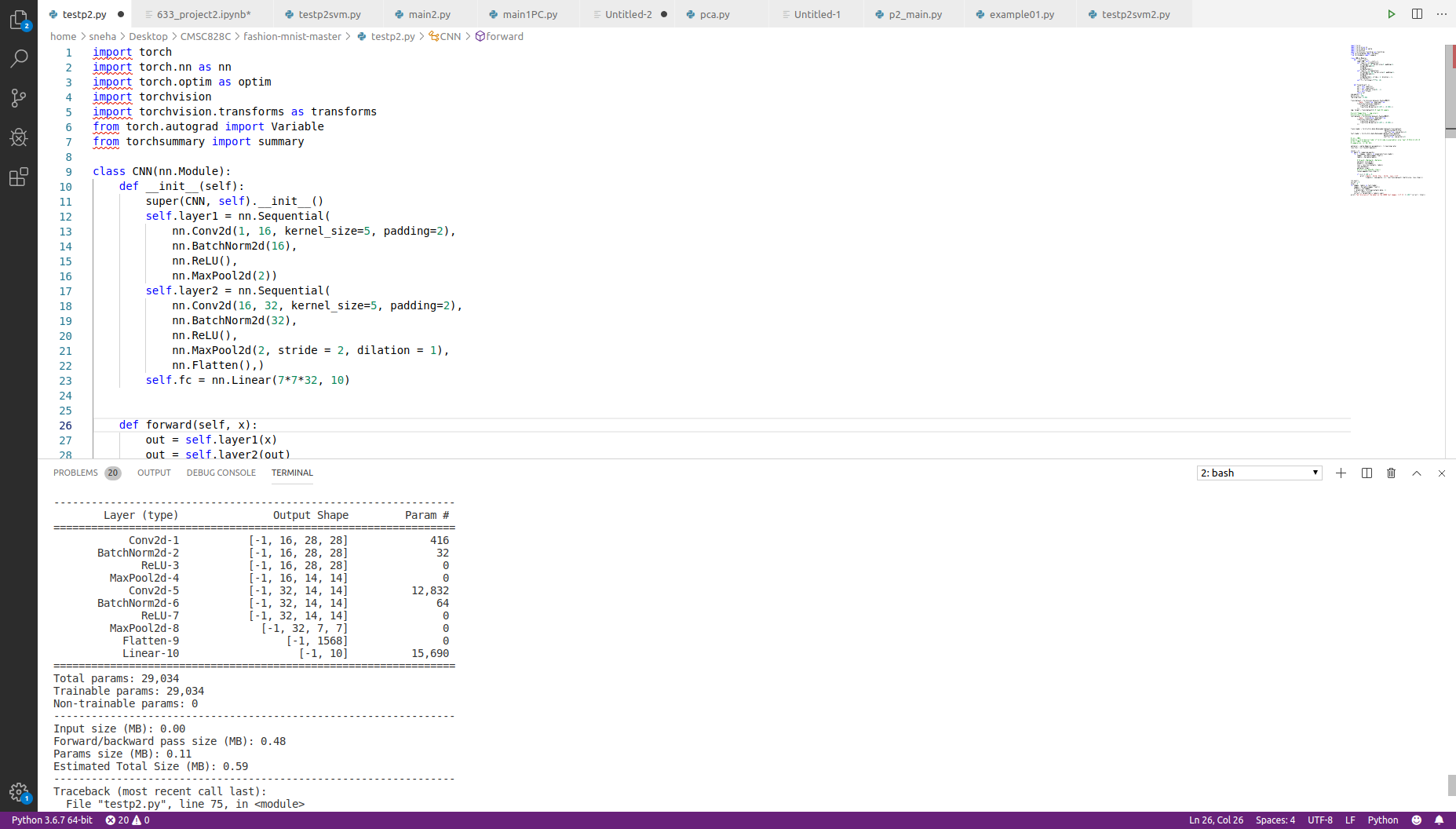
From the above accuracies, it can be seen that the combination that gives the best results is the RBF svm Kernel with PCA Reduction, with an accuracy of 87.76%.

The code for this is found in **svm.py**

**Chapter 3**

Convolutional Neural Network

The architecture that I’ve made use of here is as follows:



The accuracy comes out to be in between the range of **90% - 91%**

Methods Employed

The following was considered while building the model:

1. The data is first loaded from torch vision’s inbuilt datasets and is transformed into tensor form. The training and test set are normalized and then worked on.
2. The learning rate that is set here is 0.002 and the batch size is set to 100.
3. The optimizer that we use here is the Adam optimizer.
4. Here there is a forward pass as well as a backward pass. The loss function that we have used for this model is the Cross Entropy Loss function
5. The number of Epochs that this runs for is 6.
6. torchsummary was used to give an overview of the parameters as well as input/output dimensions from each layer.
7. The code for this is in the file cnn.py

Analysis

On testing the architecture with a Dropout layer as the 4th layer, it is seen that there is a decrease in the accuracy of about 77%. Also trying to increase the strides in the 1st layer to 2, decreased the accuracy to about 75.38%.

It is also seen that as the number of epochs increases, the model tends to give better results.

**Chapter 4**

Conclusion

We can draw the following conclusions based on the experiments performed on the dataset using SVM and Deep Learning,

1. From the experiments it is found that the classifier with the highest overall accuracy is the Neural network with a 91% accuracy.
2. The loss at the first epoch is about 0.43 and decreases to about 0.093 by the end of the 6th epoch, showing that the increase in epochs tends to decrease in the loss criterion.
3. The highest accuracy in the case of SVM is the RBF SVM kernel with PCA Dimensionality reduction, with 87.7% accuracy.
4. Normalizing and centering the data gives better accuracies.
5. The optimal number of components to be used is close to 50 for PCA.

**Chapter 5**

References

1] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick HaffnerGradient-based learning applied to document recognition, Proceedings of the IEEE, 1998, pp 2278-2324.1

[2] Karen Simonyan, and Andrew Zisserman,Very Deep Convolutional Networks for Large-Scale Image Recognition, International Conference on Learning Representations, 2015.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun,Deep Residual Learning forImage Recognition, IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[4] Gleason, J. (2019, November). Introduction To PyTorch. CMSC828C. College Park.