**Multi Class Classification with Fashion MNIST dataset**

[**Introduction**](#_uyky0j11qotv) **1**

[**Objective**](#_ywj53t2znhf5) **1**

[**Data Exploration**](#_10zmwxb57ncn) **2**

[**Experimental Setup**](#_1iljuyxk2eox) **4**

[**HyperParameter Tuning**](#_9qekptny8d71) **4**

[**Performance Comparison**](#_l74cmcd4i67n) **6**

[Graphs and Metrics](#_4jdyk07cq0u) 7

[LR](#_tbbcivhp7r6z) 7

[LR(PCA)](#_tyjej1mdkde7) 8

[KNN](#_kkcgwfm4x83h) 9

[KNN(PCA)](#_h1ye2rn5gt77) 10

[CNN](#_iteomzfeaawl) 11

[RF](#_o0ft7dk7l8yu) 12

[RF(PCA)](#_8f35m2fe6csb) 13

[**Conclusion**](#_lm5oe0gkrxgz) **13**

[**Reference**](#_ah5vnyxjsg5o) **13**

[**Link to Video**](#_l9dqaytnm9x7) **14**

## **Introduction**

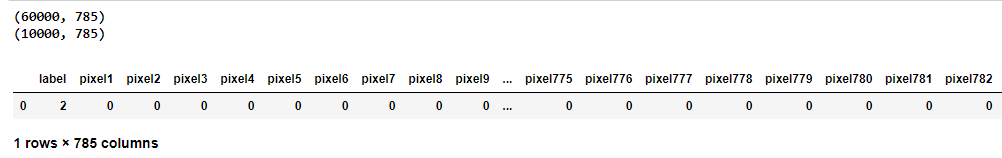
I have used the Fashion MNIST dataset for this project. I have chosen this dataset as there is a current demand for image based machine learning in the current market. The Fashion MNIST data set is a large dataset of images consisting of 60000x785 training images and 10000x785 test images. Each image is classified into 1 of 10 possible classes. As part of this experiment I am looking to achieve the following objectives.

## **Objective**

1. Explore multi class classification algorithms in the context of images.
2. Identify challenges of working with large datasets.
3. Hyperparameter tuning for different classification algorithms.
4. Impact of dimensionality reduction on classification algorithms.
5. Implement a Convolutional Neural Network for image classification.
6. Implement a Random Forest for image classification.
7. Run the algorithms in GCP.

## **Data Exploration**

The data consists of 60000 training images and 10000 test images. Each row of the data consists of 28 x 28 pixels(784) and 1 label column.

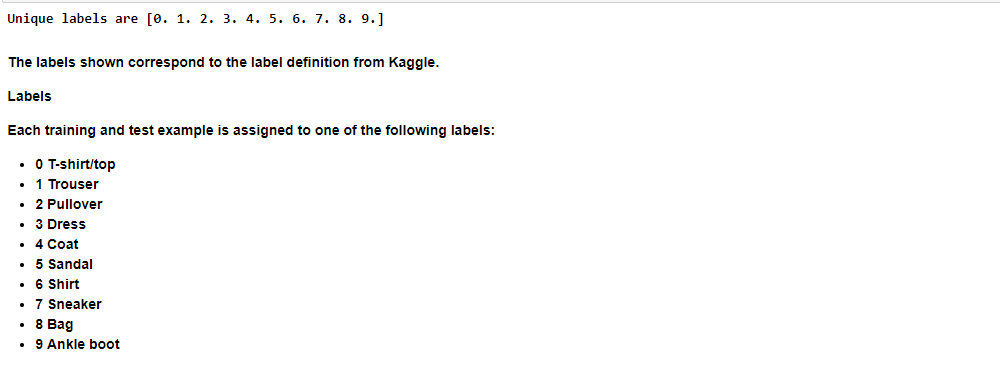


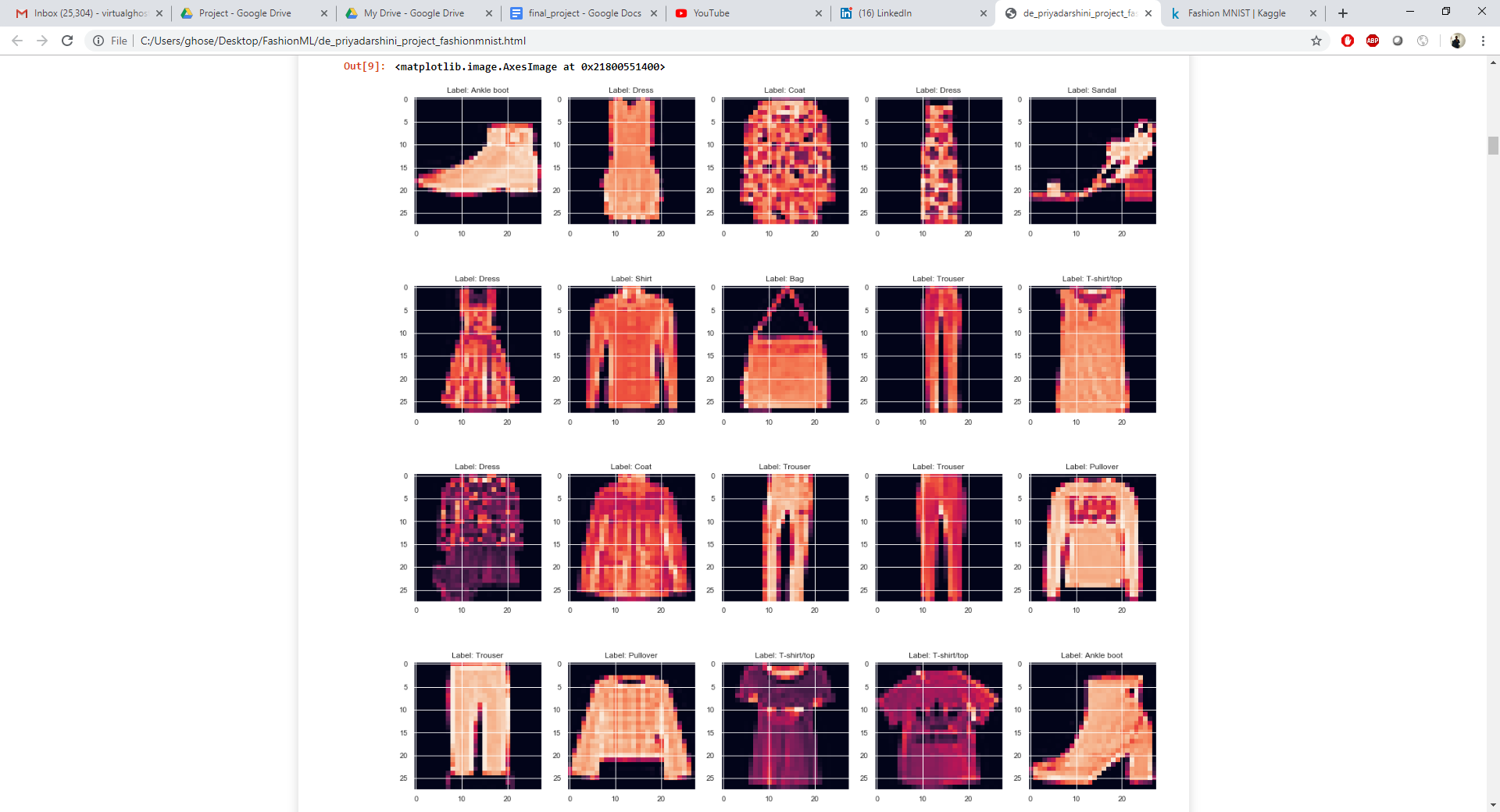
This data is a supervised dataset where the label column is the expected output for each 28x28 image.

The dataset does not consist of any null values.

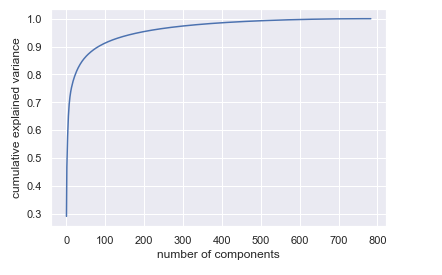


There are 10 unique labels provided from the dataset

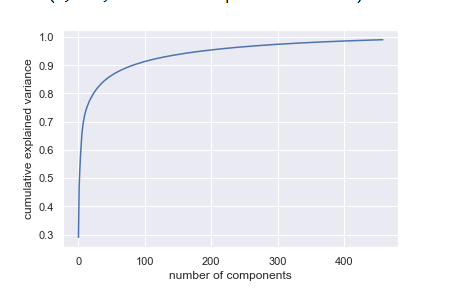




I applied Principal Component Analysis(PCA) on the Training dataset to identify the minimum number of components needed to explain the 99% variance of the data. The number of components was reduced by nearly 300 features to 440 components.



Original Dataset PCA



Dataset after applying 99% variance PCA

I have also normalized the data in the dataset by dividing all the value in the columns by the maximum value of a pixel in the dataset.

## **Experimental Setup**

I have divided the dataset into 3 parts.

1. Training
2. Validation
3. Testing

Training and Validation sets are achieved by doing a 80 - 20 split of the 60000 training images.

Testing dataset consists of 10000 images.

The training and validation dataset is used to train and create 7 different models.

The testing dataset is used to compare the performance of the developed models.

The models developed are as follows:

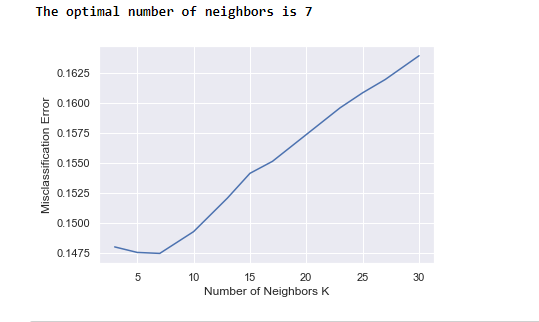
1. Logistic Regression
2. Logistic Regression on the PCA dataset.
3. K nearest neighbour
4. K nearest neighbour on the PCA dataset.
5. 3 layer Convoluted Neural Network.
6. Random forest
7. Random on the PCA dataset.

## **HyperParameter Tuning**

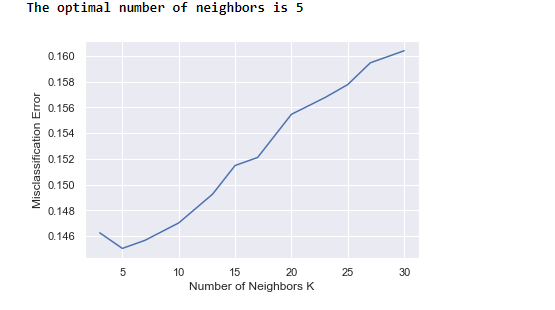
I have applied hyper parameter tuning to the following algorithm.

1. K nearest neighbour
2. CNN
3. Random forest

I have applied a custom tuning using Minimum classification error for KNN to determine the optimum number of neighbours from a list of possible neighbour selection([3,5,7,10,13,15,17,20,23,25,27,30]).



For Non PCA data



For PCA data

For CNN and Random forest, I have applied GridSearch on a set of possible features.

For CNN, I have optimized the following parameters on the basis of accuracy.



For Random forest, I have optimized the following parameters based on accuracy.



For Non PCA data



For PCA data

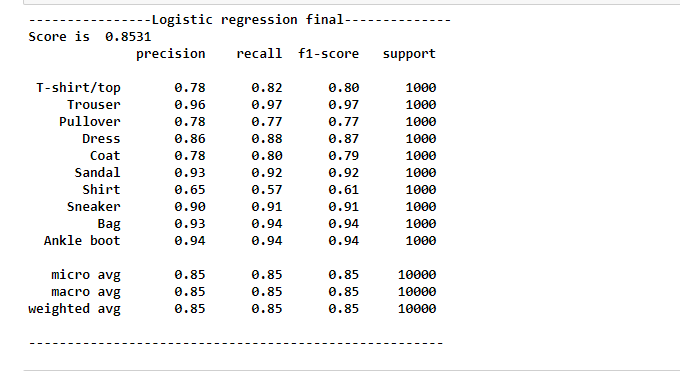
## **Performance Comparison**

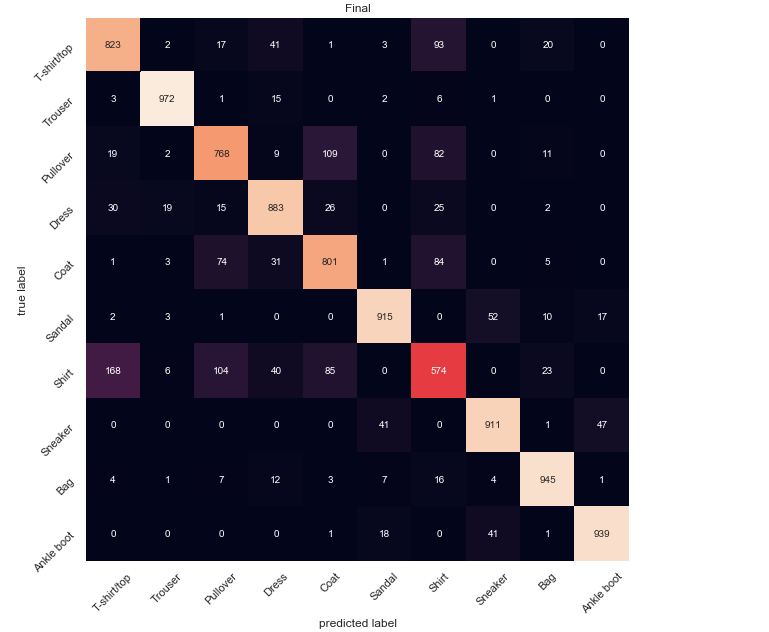
Performance Comparison was achieved by comparing the metrics of all 7 models against the testing dataset(10000 images). I have captured the weighted average of accuracy, precision, recall and F1 score below.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | LR | LR(PCA) | KNN | KNN(PCA) | CNN | RF | RF(PCA) |
| Accuracy | 0.8531 | 0.8436 | 0.8545 | 0.8582 | 0.929 | 0.881 | 0.8524 |
| Precision | 0.85 | 0.84 | 0.86 | 0.86 | 0.93 | 0.88 | 0.85 |
| Recall | 0.85 | 0.84 | 0.85 | 0.86 | 0.93 | 0.88 | 0.85 |
| F1 score | 0.85 | 0.84 | 0.85 | 0.86 | 0.93 | 0.88 | 0.85 |

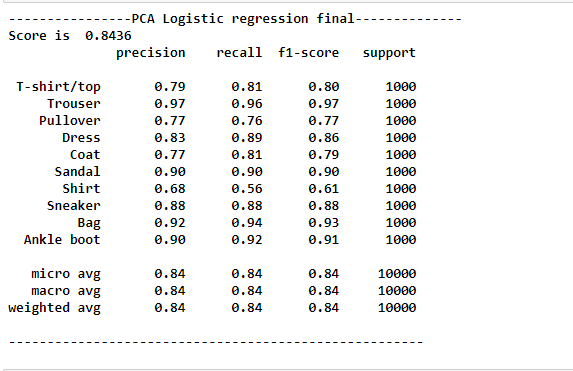
### Graphs and Metrics

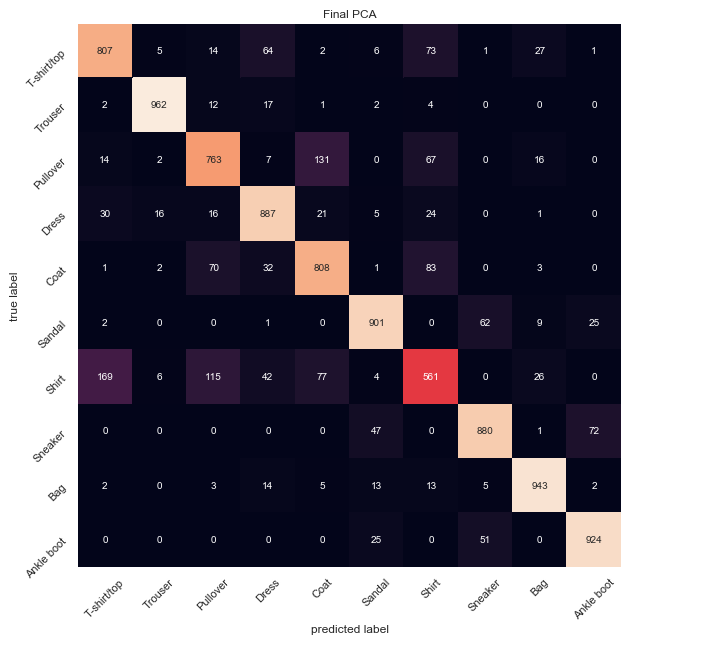
#### LR



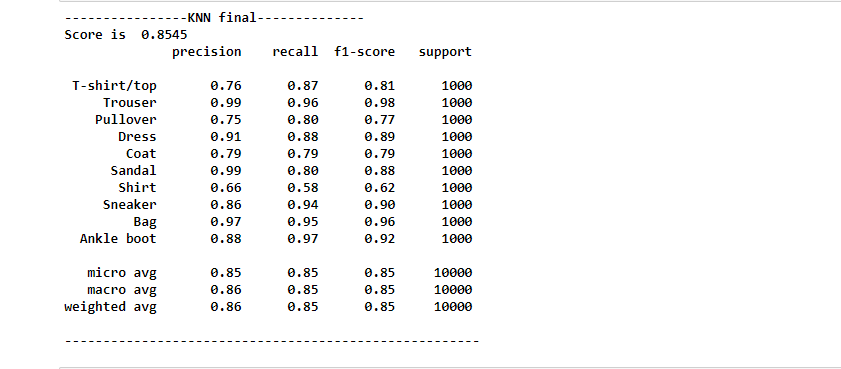


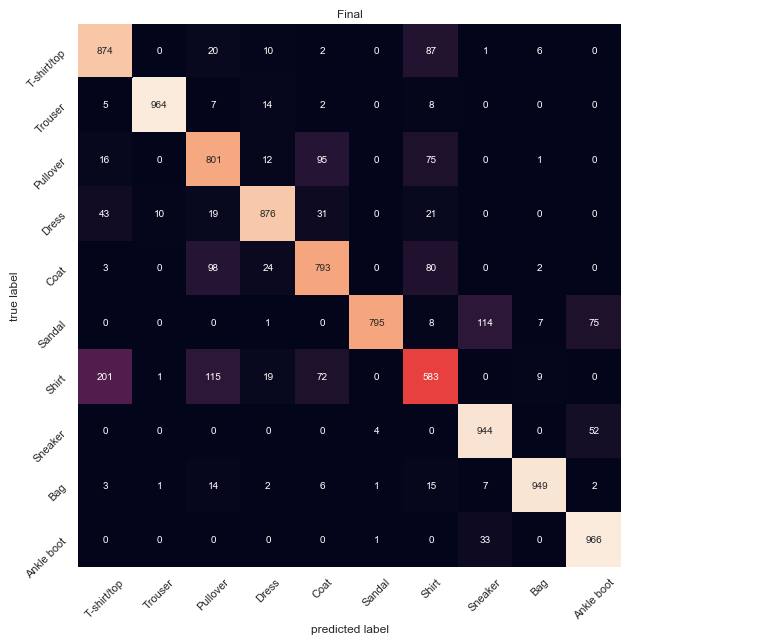
#### LR(PCA)



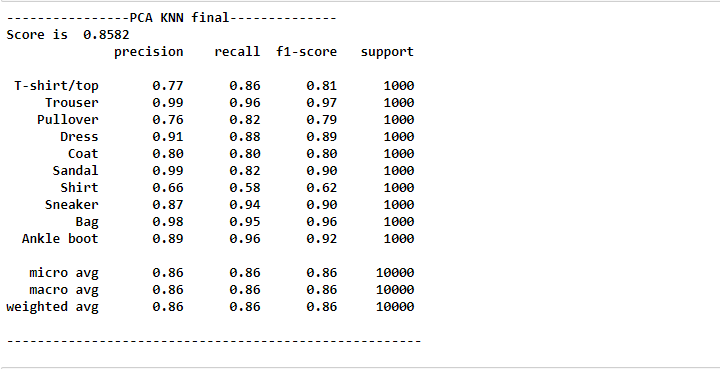


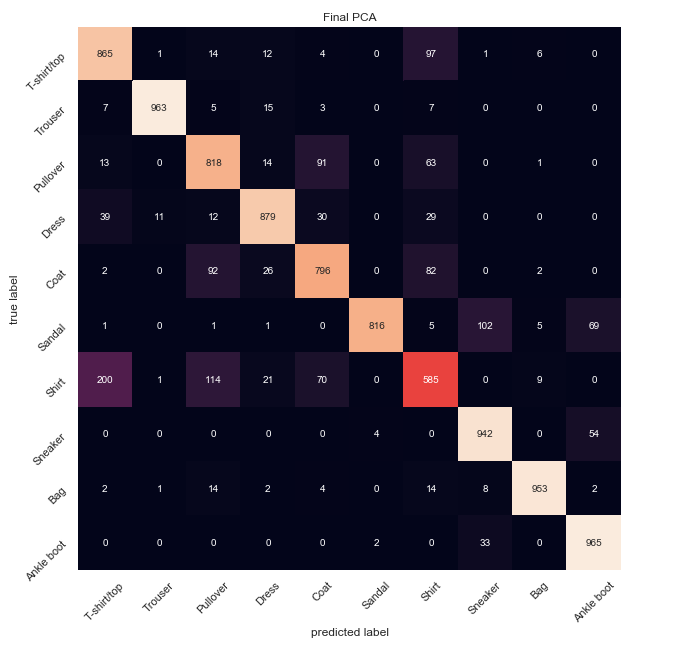
#### KNN



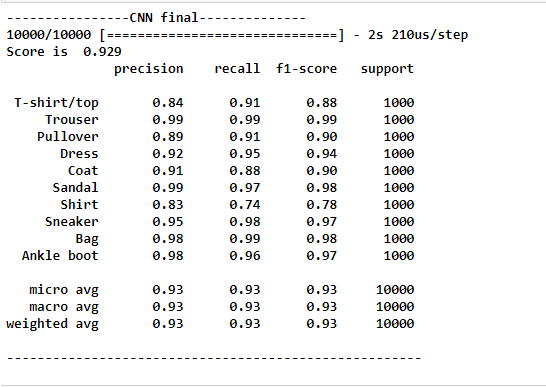


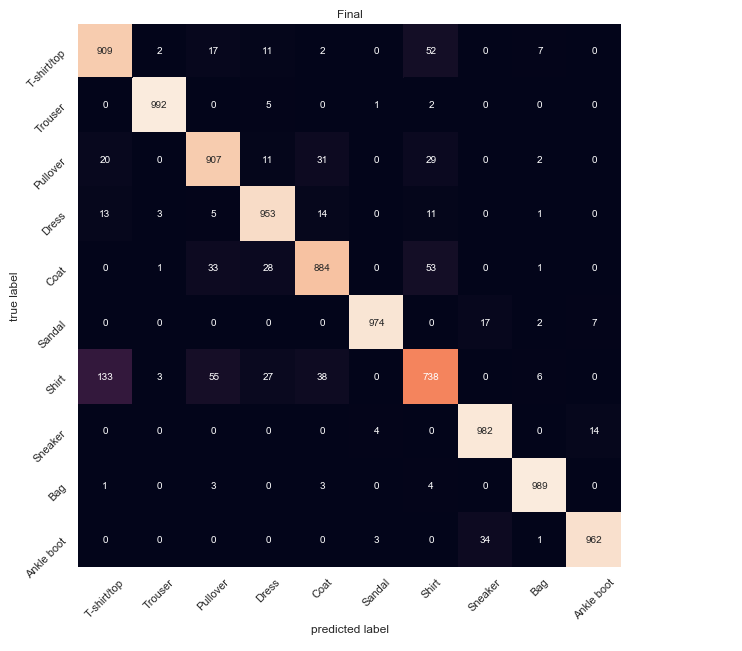
#### KNN(PCA)



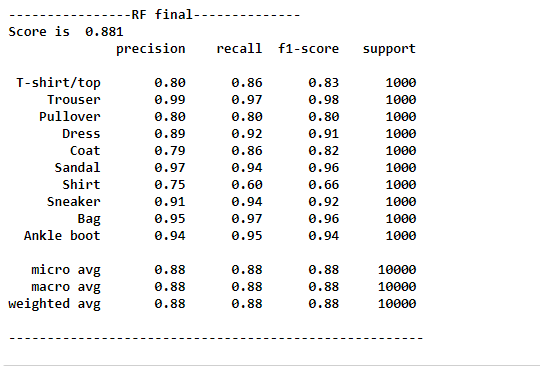


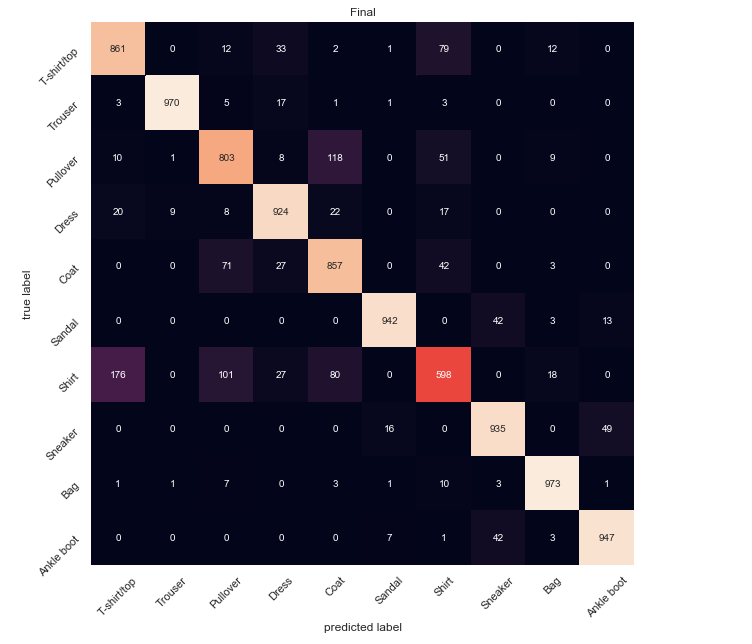
#### CNN



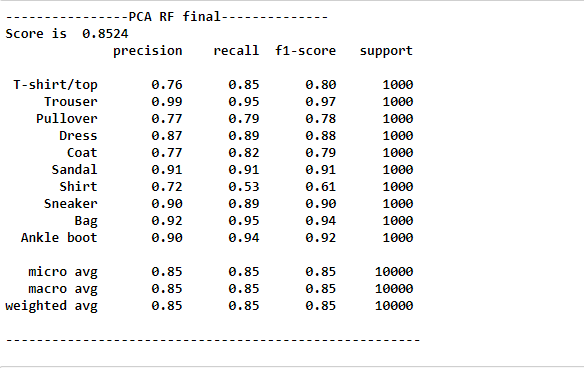


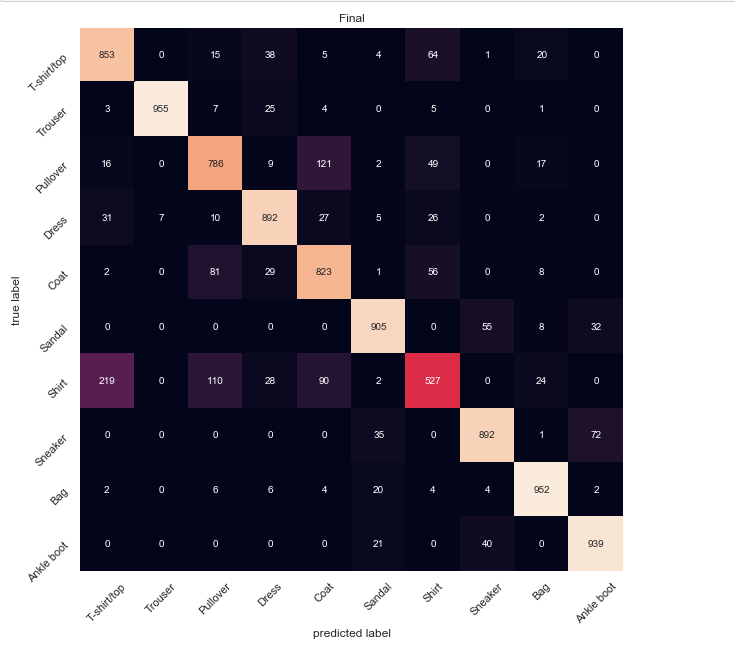
#### RF





#### RF(PCA)





## **Conclusion**

There were a number of observations/learnings that I was able to achieve as part of this experiment. They are detailed below:

1. KNN was the slowest of all the algorithms. KNN performed slower as the dataset size increased and the number of neighbours increased.
2. 3 layer CNN was the best performing algorithm on all metrics.
3. Hyper parameter tuning while time consuming allowed me to achieve much better results.
4. PCA based dataset allowed for much faster model training compared to the original dataset at the cost of a very small loss in performance(if at all).
5. I was able to perform all the training in cloud except hyper parameter tuning. Since the parameter tuning was very slow, the cloud datalab instance kept timing out.

## **Reference**

1. <https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html>
2. <https://www.geeksforgeeks.org/multiclass-classification-using-scikit-learn/>
3. <https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html>
4. <https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/#parameter-tuning-with-cross-validation>
5. <https://www.pyimagesearch.com/2019/02/11/fashion-mnist-with-keras-and-deep-learning/>
6. <https://www.pyimagesearch.com/2016/08/15/how-to-tune-hyperparameters-with-python-and-scikit-learn/>
7. <https://chrisalbon.com/deep_learning/keras/tuning_neural_network_hyperparameters/>
8. <https://www.kaggle.com/zalando-research/fashionmnist>

## **Link to Video**

## 