STA 545 Final Project

Analysis and Modelling of Fashion-MNIST

Satish Varma Bhupathiraju
UB No: 50471172/University at Buffalo

Akshey Ram Murali
UB Number: 50442206/ University at Buffalo

Vaishnav Tammadwar
UB No: 50392972/University at Buffalo

Sriharsha Teja Nallamala
UB Number: 50442347/ University at Buffalo

Introduction:

The MNIST data set is a well-known large database of Handwritten digits that is commonly used for training of various image processing algorithms by researchers in the fields of Machine Learning and Artificial Intelligence. Because of its sheer size, this dataset is useful for predictive analytics, allowing deep learning to work its efficiently. MNIST dataset was later replaced by Fashion MNIST. Fashion- MNIST data set is a collection of fashion images and comprised of 70,000 gray scale images of items having height and width of 28 pixels. FMNIST data sets consists of 10 types of clothing image such as shoes, coat, pullover, dress, sandals, shirt, sneaker, bag. The class labels mapped to 0-9 integers listed below.

- ⇒ 0: T-shirt
- ⇒ 1: Trouser
- ⇒ 2: Pullover
- ⇒ 3: Dress
- ⇒ 4: Coat
- ⇒ 5: Sandal
- ⇒ 6: Shirt
- ⇒ 7: Sneaker
- ⇒ 8: Bag
- ⇒ 9: Ankle boot.

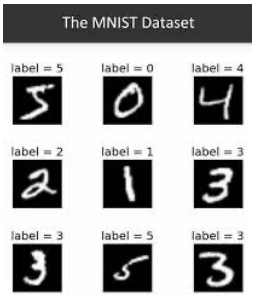


Figure (1): Images of MNIST Handwritten digits

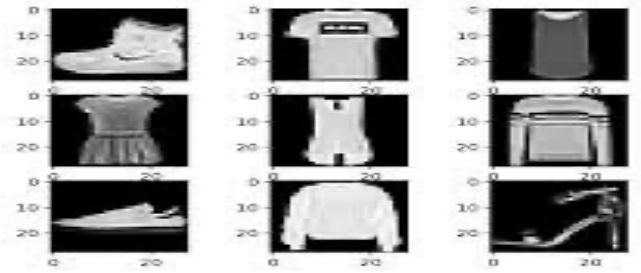


Figure (2): Images of Fashion MNIST fashion articles

Objective:

Experimenting various Machine Learning models on Fashion-MNIST dataset. The objective is to predict the fashion product from the given set of images by using the all the seven-machine learning algorithm implemented and comparing their results on the test data by the metrics defined and conclude the best ML model.

Exploratory Data Analysis:

Each image is 28 pixels in width and 28 pixels in height and a total of 784 pixels in total. Each pixel is associated with a single pixel value, indicating the lightness or darkness of the pixel. The pixel value is an integer between 0 to 255 values. The dataset consists of 785 columns in which the first column has the labels of the fashion article and associated with label of 0 to 9 integer value. The training data set has 60000 rows. whereas testing data is comprised of 10000 rows. The dataset is split into 85% training data and 15% as testing data. All the labels are equally spread in the given dataset by using seaborn function in python it has been plotted in figure (3). Each train and test data are assigned to one of the labels. By reshaping and setting the cmap to magma would get the corresponding images to label in figure (4).

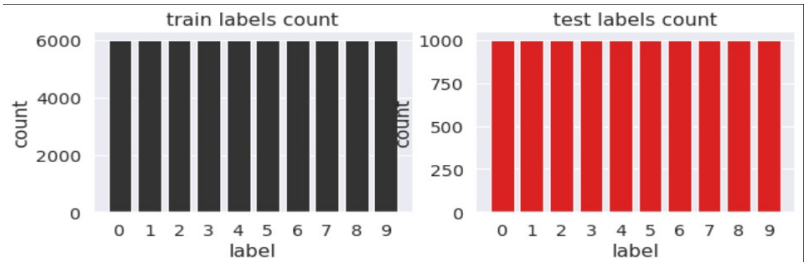


Figure (3): Image for spread of labels in dataset

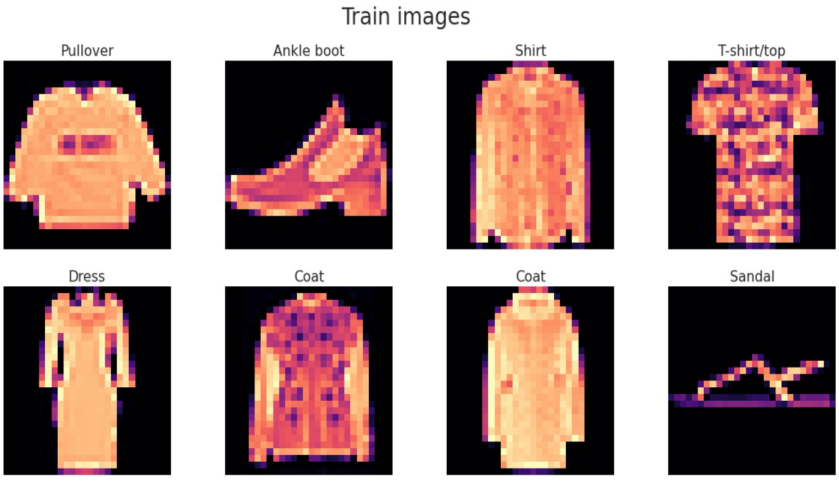


Figure (4): Images of fashion articles of dataset.

Data cleaning and Data Pre-Processing:

Given dataset has no null values. The data pre-processing is an essential part in data analysis. Image data require proper analysis for better results. The fashion Mnist raw train data was clumsy in the beginning. Moving forward with this raw data can adversely affects the accuracy. Firstly, the label column was dropped and then the data was converted to float followed by scaling down the pixels in the range of 0 to 1. Now the entire data will range between 0 to 1. This will improve our results. In order to further improve our pre-processing, dimensionality reduction was carried out using PCA function in sklearn. This is done to remove the highly correlated data in the dataset and after PCA, around 5% of the correlated data was removed. By this way, the pre-processing for our data was carried out.

Machine Learning Models:

Logistic Regression:

The logistic model is a statistical model that models the likelihood of an event occurring by making the event's log-odds a linear combination of one or more independent variables. The Logistic regression function from the sklearn is used here. The parameters here are solver which is generally the algorithm used in the problem. Liblinear is chosen here since it is good for mid-range datasets. Penalized logistic regression imposes a penalty to the logistic model for having too many variables. This results in shrinking the coefficients of the less contributive variables toward zero. This is also known as regularization. Penalty of l1, l2 was added and our model performed the same and gave the same accuracy for both with and without penalty (hyperparameters were checked and resulted in the same accuracy).

Support Vector Machine:

The support vector classifier is used to classify the pixel values to the corresponding labels. The SVC function from the sklearn is used for this data. The parameters used here are a regularization parameter value, kernel and gamma. The regularization parameter value is an integer, and the kernel is set to rbf, other kernels were tried and out of which the rbf kernel gave the best results. Gamma was set to auto and scale, the scale value resulted in high accuracy than default gamma parameter. The model yielded an accuracy 90.58%.

Convolution Neural Network:

Convolutional neural network is a deep learning neural network structure. A CNN's architecture is analogous to the connectivity pattern of the human brain. Just like the brain consists of billions of neurons, CNNs also have neurons arranged in a specific way. In fact, a CNN's neurons are arranged like the brain's frontal lobe, the area responsible for processing visual stimuli. In this

keras API has been used to implement the CNN for the fashion Mnist data. The data is loaded from the predefined function in keras and the data is split into train and test. Pre-processing is done by normalizing the data by scaling all the pixels in the range of 0 to 1. The architecture of CNN has three convolution layers, max-pooling and SoftMax layers. reLU and adam are used as activation functions and optimizers respectively. The model is trained for 10 epochs and the model yielded an accuracy of 91.4%. This accuracy is gained after several hyper parameters tuning and this set of hyper parameters gave the highest accuracy.

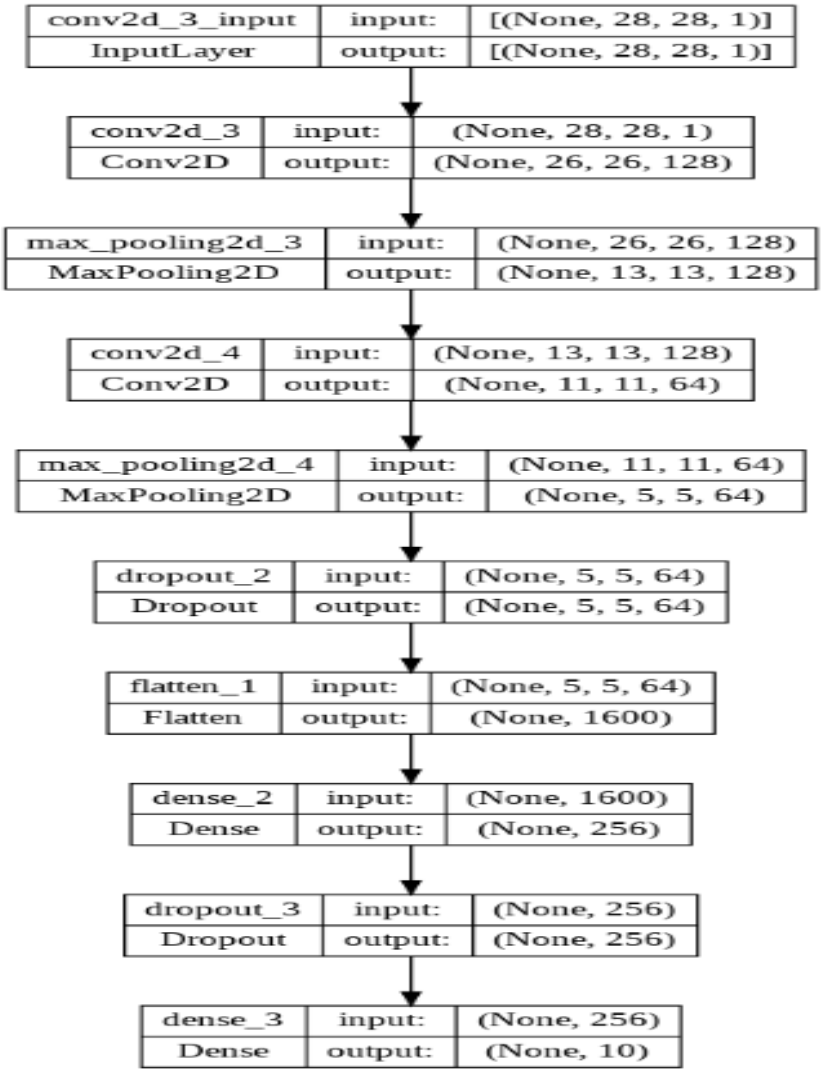


Figure (5): CNN Architecture flow chart

Decision Tree:

Decision Tree is represented as an upside-down tree with its root at the top. Data is split from the root and each non-leaf node contains a condition. We partition the data into resulting regions and repeat the same process of splitting on each region. The final leaf node contains a prediction. To build this model we are using the predefined function from sklearn (Decision Tree Classifier). Also, we are using a max depth of 13,16,19(depth of the tree) and we got the best accuracy for the

tree of max_depth = 13. To measure the quality of the split we are using gini impurity. Here 'pi' is the probability of an object being classified to a particular class.

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2$$

We choose the attribute which has the least Gini Index as the root when building the decision tree.

Random Forest:

Consists of multiple decision trees where each individual tree in this splits out a class prediction and the best are considered. For implementing this model we are using a predefined function from sklearn(Random Forest Classifier). As for the parameters, in the n_estimators we are using 3 values (30,60,90) and also, we are using a max depth of 9,15,22. Out of all these parameters we get the highest accuracy of 88.03% when we use n_estimators= 90, max_depth=22. Now for the splitting of attributes we are using the same concept of Gini index which we used in the case of decision tree.

K Nearest Neighbor:

K-Nearest neighbor algorithm is a nonparametric classification algorithm. Contrast to it, there are many parameters used for classifying. Randomly select k centroids in the train set. Given the data point(image) find the k datapoints nearest from the train dataset. Choosing parameter K can be done by randomly choosing k and checking the accuracy and choosing the best. For FMINST dataset for K value the accuracy is seen highest. Calculate the distance between datapoint and the centroids using Euclidean distance and classify the datapoint(image) to the nearest datapoint(image). The predefined function from sklearn is used this model(K-Neighbors Classifier) From sklearn only K value is updated as 8 and rest of the parameters are default.

Naïve Bayes:

The "naïve" assumption that each pair of features is conditionally independent given the value of the class variable underlies a class of supervised learning algorithms collectively referred to as naïve Bayes methods. The Bayes' Theorem suggests a connection between given class variable y and dependent features vector x1 through xn.

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots, x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Calculates the probability for membership of a data point to each class. Assumes that probability of each pixel is a gaussian distribution and probability of each digit is equal.
The predefined function from sklearn is used this model (Gaussian NB)

Results:

Logistic Regression:

The Logistic Regression model is fit using the above parameters and predefined function from sklearn. After several hyperparameter tuning the best accuracy for this data is 84%.

----------THE CLASSIFICATION REPORT FOR LOGISTIC REGRESSION IS----------

	precision	recall	f1-score	support
0	0.80	0.82	0.81	590
1	0.97	0.97	0.97	585
2	0.74	0.75	0.74	579
3	0.79	0.87	0.83	567
4	0.73	0.73	0.73	596
5	0.92	0.92	0.92	635
6	0.67	0.55	0.61	612
7	0.89	0.91	0.90	630
8	0.93	0.93	0.93	607
9	0.94	0.94	0.94	599
accuracy			0.84	6000
macro avg	0.84	0.84	0.84	6000
weighted avg	0.84	0.84	0.84	6000

Figure (6): Classification report for Logistic regression

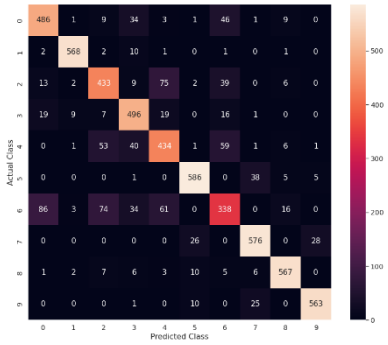


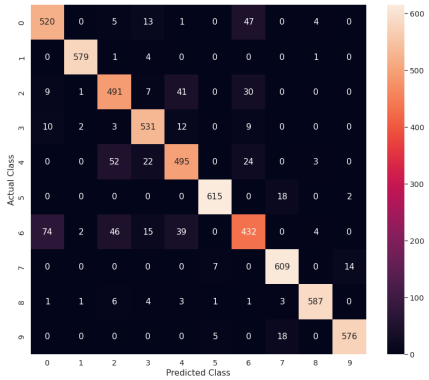
Figure (7): Confusion Matrix for Logistic regression

Support Vector Machine:

The SVC model is fit using the above parameters and predefined function from sklearn. After several hyperparameter tuning the best accuracy for this data is 90.05%.

	precision	recall	f1-score	support
0	0.80	0.82	0.81	590
1	0.97	0.97	0.97	585
2	0.74	0.75	0.74	579
3	0.79	0.87	0.83	567
4	0.73	0.73	0.73	596
5	0.92	0.92	0.92	635
6	0.67	0.55	0.61	612
7	0.89	0.91	0.90	630
8	0.93	0.93	0.93	607
9	0.94	0.94	0.94	599
accuracy			0.84	6000
macro avg	0.84	0.84	0.84	6000
weighted avg	0.84	0.84	0.84	6000

(8): Classification report for support vector machines



Figure

Figure (9): Confusion Matrix for support vector machines

KNN:

The kNN model is fit using the above parameters and predefined function from sklearn. After several hyperparameter tuning the best accuracy for this data is 86%.

--------------THE CLASSFICATION REPORT FOR KNN---------------

	precision	recall	f1-score	support
0	0.76	0.87	0.81	1493
1	0.99	0.97	0.98	1463
2	0.74	0.78	0.76	1449
3	0.89	0.89	0.89	1500
4	0.75	0.77	0.76	1527
5	0.98	0.90	0.93	1442
6	0.69	0.57	0.62	1538
7	0.90	0.94	0.92	1518
8	0.98	0.93	0.95	1544
9	0.92	0.96	0.94	1526
accuracy			0.86	15000
macro avg	0.86	0.86	0.86	15000
weighted avg	0.86	0.86	0.86	15000

Figure (10): Classification report for knn

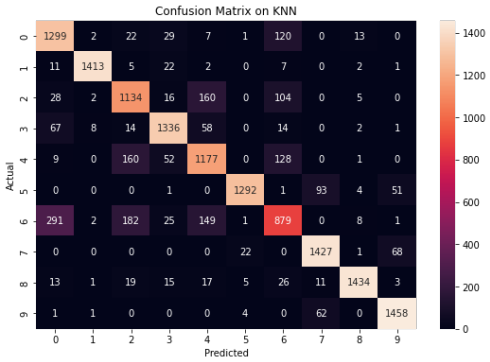


Figure (11): Confusion Matrix for knn

Naïve Bayes:

The Naïve Bayes model is fit using the above parameters and predefined function from sklearn. After several hyperparameter tuning the best accuracy for this data is 67%.

--------------CLASSIFICATION REPORT OF NAIVE BAYES-----------------

	precision	recall	f1-score	support
0	0.78	0.69	0.73	1493
1	0.98	0.92	0.95	1463
2	0.66	0.56	0.61	1449
3	0.69	0.87	0.77	1500
4	0.48	0.79	0.60	1527
5	0.76	0.54	0.63	1442
6	0.28	0.16	0.20	1538
7	0.65	0.89	0.75	1518
8	0.57	0.62	0.59	1544
9	0.89	0.63	0.73	1526
accuracy			0.67	15000
macro avg	0.68	0.67	0.66	15000
weighted avg	0.67	0.67	0.66	15000

Figure (12): Classification report for Naïve Bayes

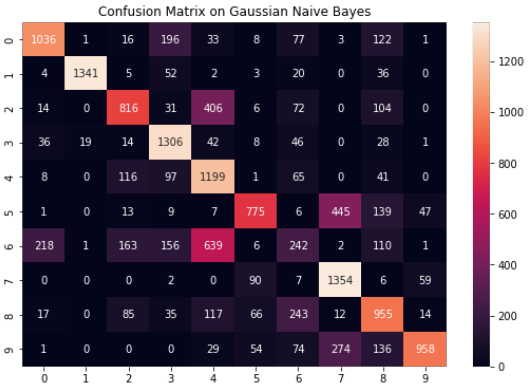


Figure (13): Confusion Matrix for Naive Bayes

Decision Tree:

The Decision Tree model is fit using the above parameters and predefined function from sklearn. After several hyperparameter tuning the best accuracy for this data is 82%.

```
----------CLASSFICATION REPORT OF DECISION TREE----------
              precision    recall  f1-score   support

           0       0.76      0.78      0.77      1000
           1       0.97      0.96      0.96      1000
           2       0.72      0.69      0.70      1000
           3       0.87      0.83      0.85      1000
           4       0.67      0.77      0.72      1000
           5       0.92      0.88      0.90      1000
           6       0.59      0.56      0.57      1000
           7       0.88      0.89      0.89      1000
           8       0.93      0.92      0.92      1000
           9       0.90      0.91      0.90      1000

    accuracy                           0.82     10000
   macro avg       0.82      0.82      0.82     10000
weighted avg       0.82      0.82      0.82     10000
```

Figure (14): Classification report for Decision Tree

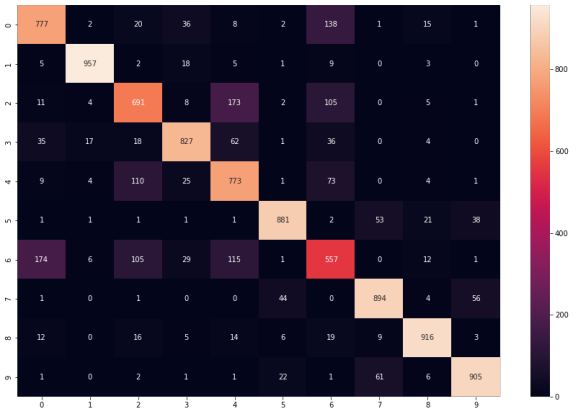


Figure (15): Confusion Matrix for Decision Tree

Random Forest:

The random forest model is fit using the above parameters and predefined function from sklearn. After several hyperparameter tuning the best accuracy for this data is 88%.

```
              precision    recall  f1-score   support

           0       0.81      0.87      0.84      1000
           1       0.99      0.97      0.98      1000
           2       0.80      0.81      0.80      1000
           3       0.90      0.93      0.91      1000
           4       0.80      0.86      0.83      1000
           5       0.98      0.95      0.96      1000
           6       0.75      0.60      0.67      1000
           7       0.92      0.93      0.93      1000
           8       0.96      0.98      0.97      1000
           9       0.93      0.95      0.94      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

Figure (16): Classification report for Random Forest

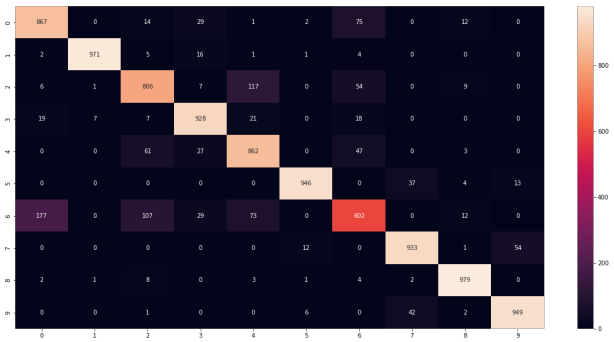


Figure (17): Confusion Matrix for Random Forest

Convolution Neural Network:

The CNN model is fit using the above parameters and predefined function from sklearn. After several hyperparameter tuning the best accuracy for this data is 91.4%

CLASSFICATION ANALYSIS

	precision	recall	f1-score	support
0	0.83	0.91	0.87	1000
1	1.00	0.98	0.99	1000
2	0.86	0.89	0.88	1000
3	0.93	0.90	0.92	1000
4	0.90	0.83	0.87	1000
5	0.98	0.98	0.98	1000
6	0.81	0.67	0.73	1000
7	0.95	0.97	0.96	1000
8	0.99	0.98	0.98	1000
9	0.97	0.96	0.96	1000
micro avg	0.92	0.91	0.92	10000
macro avg	0.92	0.91	0.91	10000
weighted avg	0.92	0.91	0.91	10000
samples avg	0.91	0.91	0.91	10000

Figure (18): Classification report for CNN

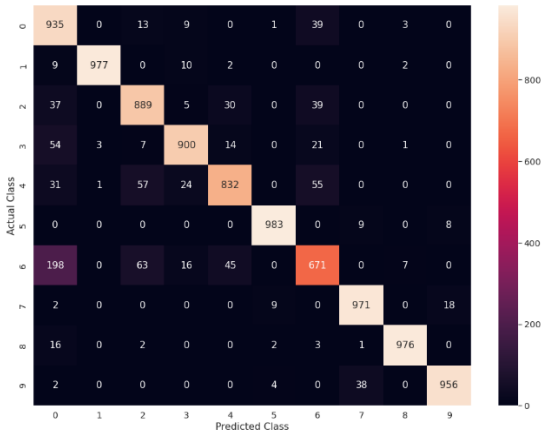


Figure (19): Confusion Matrix for CNN

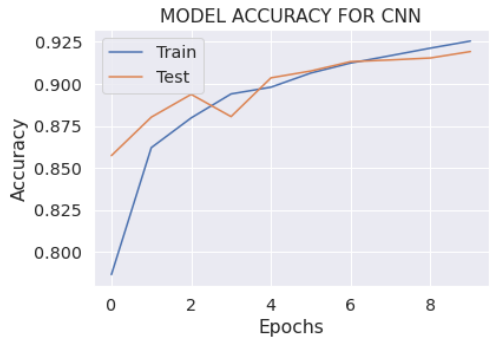


Figure (20): Model Accuracy for CNN

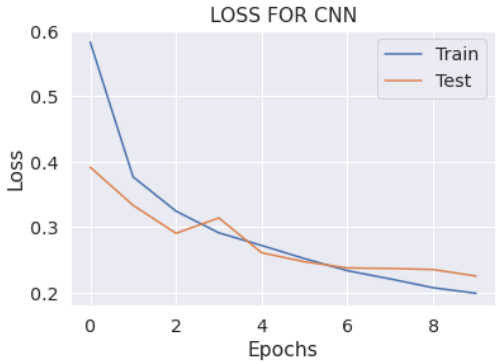


Figure (20): Model loss for CNN

Observations:

The fashion-MNIST data has been analyzed using all the above listed models, out of which CNN has attained the highest accuracy of 91.4% followed by the svc classifier. CNN has proved again it is the best model for image data.

Individual Work Contributions:

Name	Work Contributed	Contributed percentage
AKSHEY RAM MURALI	CNN, SVM, PRE-PROCESSING, PRESENTATION	25%
Satish Varma Bhupathiraju	EDA, LOGISTIC REGRESSION, REPORT	25%
Sriharsha Teja Nallamala	DECISION TREE, RANDOM FOREST, PRE-PROCESSING	25%
Vaishnav Tammadwar	NAÏVE BAYES , KNN, PRE-PROCESSING	25%

Presentation Q&A:

How to choose K value?

What is K value: K value is the number closest datapoints to check for the given datapoint. Calculate distance between the datapoints and find K nearest points and check the most frequent class and classify the datapoint to the class. For this dataset we have chosen K ranging from 5-15 and found accuracy is best when K value is given as 8.

Why low accuracy for naïve bayes classifier?

Because the pixels are independent of each other, so it is very hard to get desired shape. Hence the naïve bayes assumption that all the features are independent of each other for the fashion Mnist gives low accuracy compared to other models.

The SVC would have given higher accuracy than CNN when the gamma parameter is set to scale. Is That true?

Yes, the gamma parameter resulted in higher accuracy when it is set to ‘scale’ than auto. Unfortunately, the SVC did not beat CNN in spite of changing its parameter (refer code).

Was penalty term used for the logistic regression?

The penalty term of l2 is used for our model and also check without using the penalty. Fortunately, both gave very similar results with mild changes in the accuracy, difference was 0.0035%

Github Repository Link:

https://github.com/aksheyram/Group15-Fashion-MNIST

References:

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
https://scikit-learn.org/stable/modules/svm.html
https://scikit-learn.org/stable/modules/naive_bayes.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
https://scikit-learn.org/stable/modules/tree.html

APPENDIX:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn import model_selection, preprocessing
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Input, Dropout, MaxPooling2D, Conv2D,
Flatten, BatchNormalization
from keras.callbacks import LearningRateScheduler, ModelCheckpoint,
EarlyStopping
from keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

(trainX, trainy), (testX, testy) = fashion_mnist.load_data()

Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 2s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-
```

```
keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
```

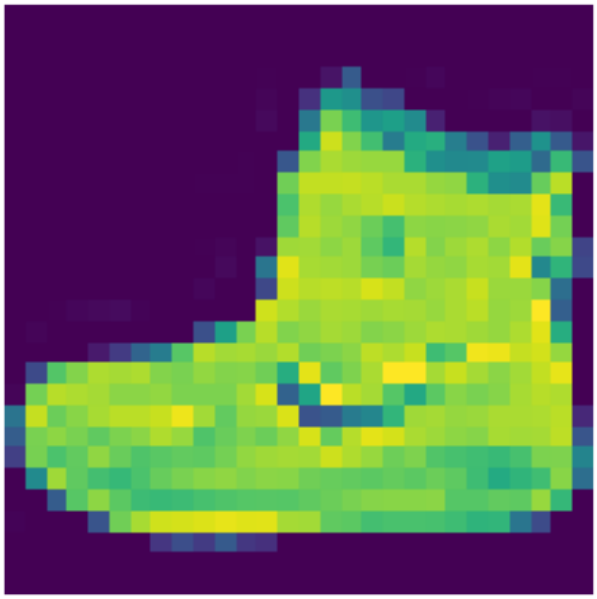
```
label = {
    0: "T-shirt/top",
    1: "Trouser",
    2: "Pullover",
    3: "Dress",
    4: "Coat",
    5: "Sandal",
    6: "Shirt",
    7: "Sneaker",
    8: "Bag",
    9: "Ankle boot"
}
```

```
print(trainX.shape)
print(testX.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
```

```
plt.axis('off')
plt.imshow(trainX[0])
```

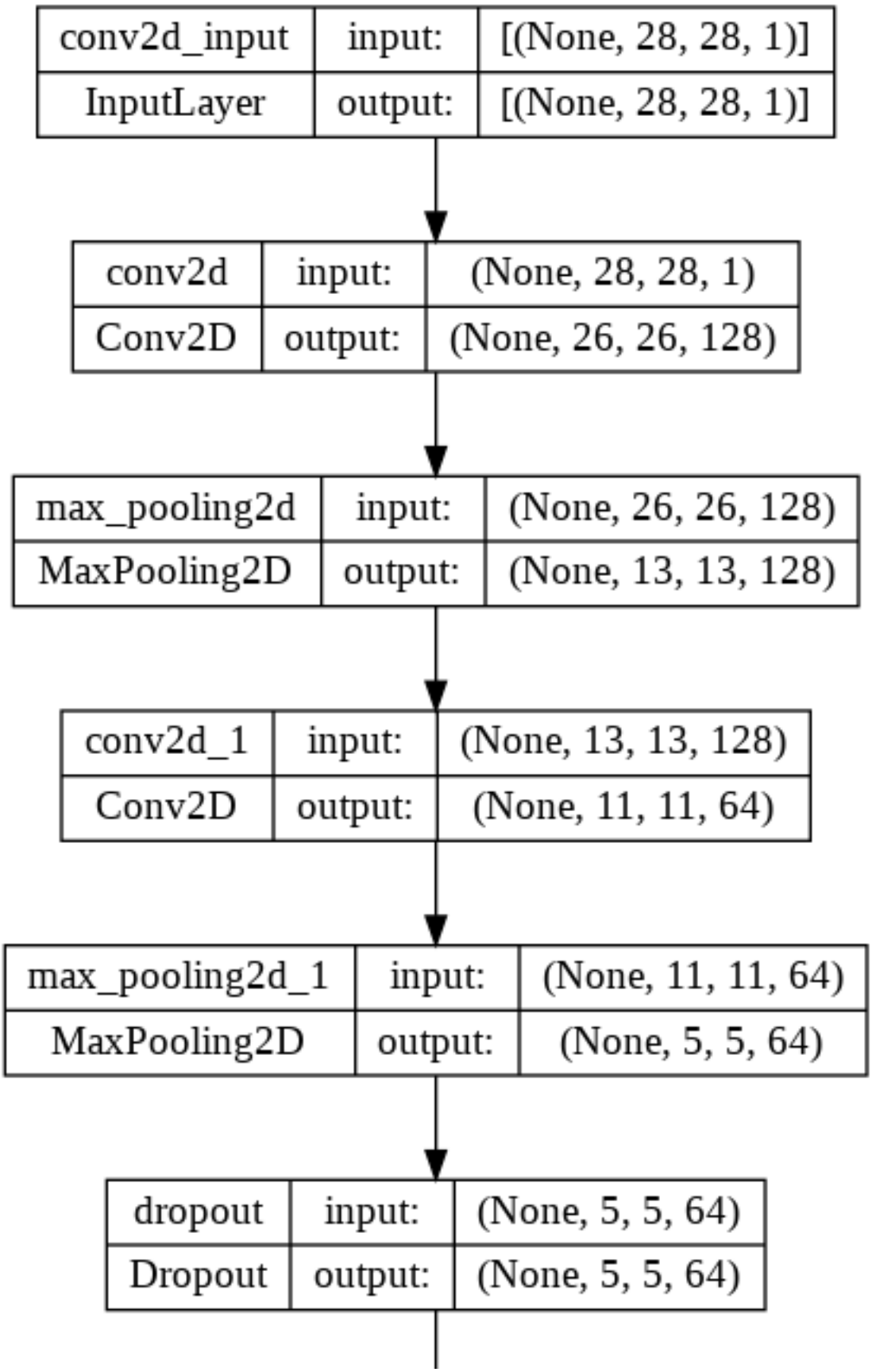
```
<matplotlib.image.AxesImage at 0x7f345d55b100>
```



```
new_trainX = np.copy(trainX).astype('float32')/255.0
new_testX = np.copy(testX).astype('float32')/255.0
```

```
X_train = new_trainX.reshape((new_trainX.shape[0], 28, 28, 1))
y_train = to_categorical(trainy)
```

```
X_test = new_testX.reshape((new_testX.shape[0], 28, 28, 1))
y_test = to_categorical(testy)

X_train.shape, y_train.shape

((60000, 28, 28, 1), (60000, 10))

cnn_model = Sequential([
    Conv2D(128,(3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D(2,2),
    Conv2D(64,(3,3), activation='relu'),
    MaxPooling2D(2,2),
    #Conv2D(32,(3,3), activation='relu'),
    #MaxPooling2D(2,2),
    Dropout(0.3),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(10, activation='softmax'),
])

keras.utils.plot_model(cnn_model, show_shapes=True)
```


conv2d_input
input:
[(None, 28, 28, 1)]
InputLayer
output:
[(None, 28, 28, 1)]
conv2d
input:
(None, 28, 28, 1)
Conv2D
output:
(None, 26, 26, 128)
max_pooling2d
input:
(None, 26, 26, 128)
MaxPooling2D
output:
(None, 13, 13, 128)
conv2d_1
input:
(None, 13, 13, 128)
Conv2D
output:
(None, 11, 11, 64)
max_pooling2d_1
input:
(None, 11, 11, 64)
MaxPooling2D
output:
(None, 5, 5, 64)
dropout
input:
(None, 5, 5, 64)
Dropout
output:
(None, 5, 5, 64)

```
cnn_model.compile(optimizer='adam',
             loss="categorical_crossentropy",
             metrics=['accuracy'])

cnn_model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 26, 26, 128)       1280

 max_pooling2d_6 (MaxPooling  (None, 13, 13, 128)      0
 2D)

 conv2d_7 (Conv2D)           (None, 11, 11, 64)        73792

 max_pooling2d_7 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 dropout_6 (Dropout)         (None, 5, 5, 64)          0

 flatten_3 (Flatten)         (None, 1600)              0

 dense_6 (Dense)             (None, 256)               409856

 dropout_7 (Dropout)         (None, 256)               0

 dense_7 (Dense)             (None, 10)                2570

=================================================================
Total params: 487,498
Trainable params: 487,498
Non-trainable params: 0
_________________________________________________________________
```

```
my_cnn = cnn_model.fit(X_train, y_train,
                   epochs=10,
                   batch_size=128,
                   validation_split=0.2)
```

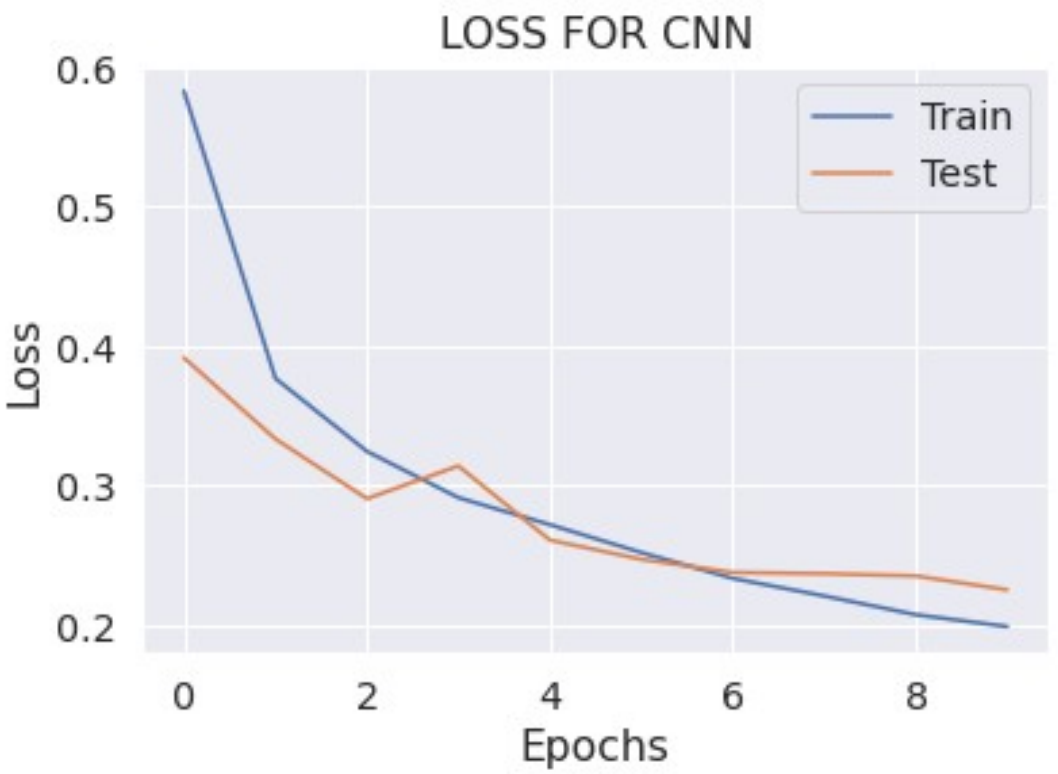
```
Epoch 1/10
375/375 [==============================] - 5s 11ms/step - loss: 0.5826
- accuracy: 0.7864 - val_loss: 0.3916 - val_accuracy: 0.8572
Epoch 2/10
375/375 [==============================] - 4s 10ms/step - loss: 0.3769
- accuracy: 0.8620 - val_loss: 0.3335 - val_accuracy: 0.8802
Epoch 3/10
375/375 [==============================] - 4s 11ms/step - loss: 0.3246
- accuracy: 0.8797 - val_loss: 0.2907 - val_accuracy: 0.8938
Epoch 4/10
```

```
375/375 [==============================] - 4s 10ms/step - loss: 0.2914
- accuracy: 0.8940 - val_loss: 0.3141 - val_accuracy: 0.8805
Epoch 5/10
375/375 [==============================] - 4s 11ms/step - loss: 0.2721
- accuracy: 0.8980 - val_loss: 0.2611 - val_accuracy: 0.9036
Epoch 6/10
375/375 [==============================] - 4s 11ms/step - loss: 0.2521
- accuracy: 0.9065 - val_loss: 0.2472 - val_accuracy: 0.9077
Epoch 7/10
375/375 [==============================] - 4s 11ms/step - loss: 0.2336
- accuracy: 0.9124 - val_loss: 0.2378 - val_accuracy: 0.9133
Epoch 8/10
375/375 [==============================] - 4s 10ms/step - loss: 0.2210
- accuracy: 0.9168 - val_loss: 0.2370 - val_accuracy: 0.9143
Epoch 9/10
375/375 [==============================] - 4s 11ms/step - loss: 0.2076
- accuracy: 0.9213 - val_loss: 0.2353 - val_accuracy: 0.9154
Epoch 10/10
375/375 [==============================] - 4s 10ms/step - loss: 0.1991
- accuracy: 0.9255 - val_loss: 0.2254 - val_accuracy: 0.9193
```

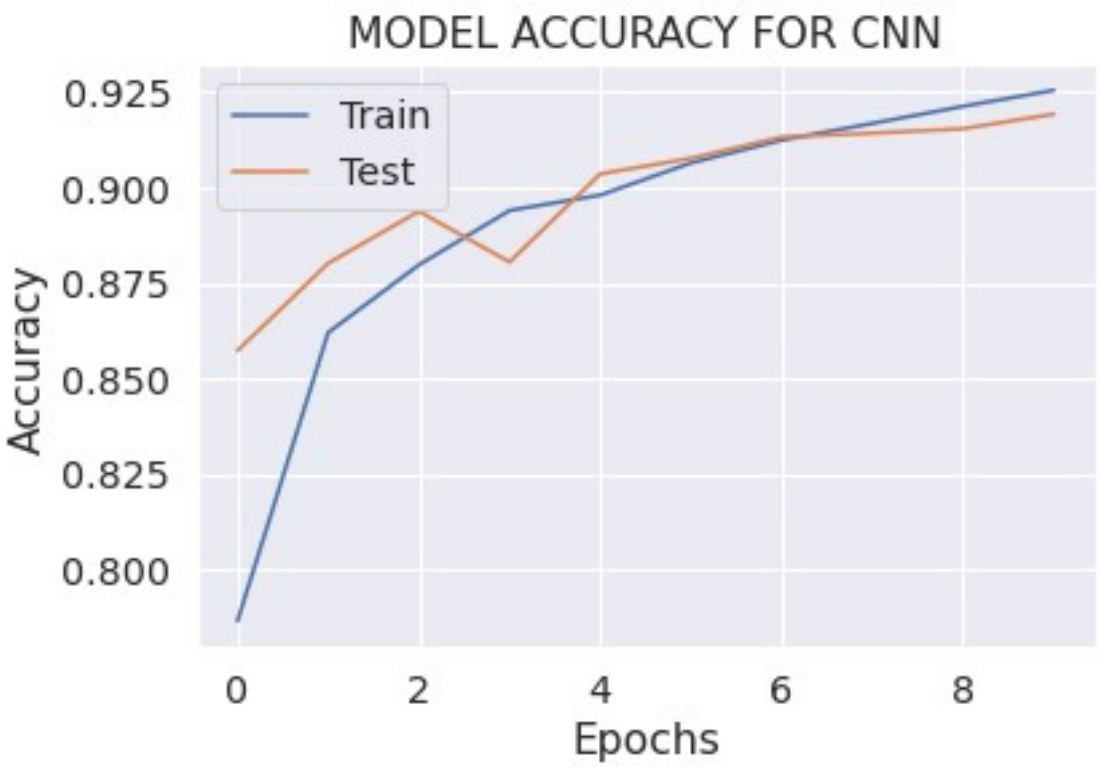
```
test_loss, test_acc = cnn_model.evaluate(X_test, y_test)
test_acc= test_acc*100
print("The test accuracy :",test_acc)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.2406
- accuracy: 0.9143
The test accuracy : 91.430002450943
```

```
plt.plot(my_cnn.history['loss'])
plt.plot(my_cnn.history['val_loss'])
plt.title("LOSS FOR CNN")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'])
plt.show()
```




```
plt.plot(my_cnn.history['accuracy'])
plt.plot(my_cnn.history['val_accuracy'])
plt.title("MODEL ACCURACY FOR CNN")
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'])
plt.show()
```

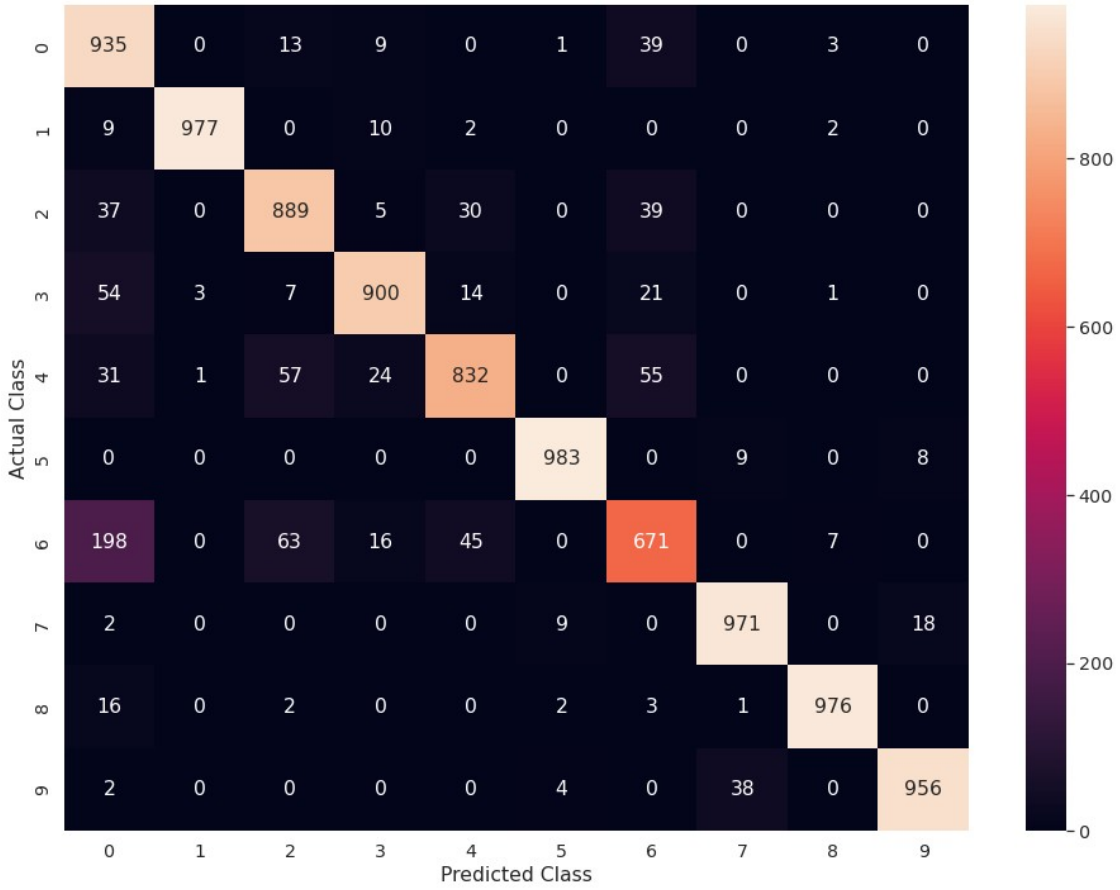



```
y_predictions = cnn_model.predict(X_test)
y_predictions = (y_predictions > 0.5)

print("CLASSFICATION ANALYSIS")
print(classification_report(y_test, y_predictions))
```

```
313/313 [==============================] - 1s 2ms/step
CLASSFICATION ANALYSIS
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      1000
           1       1.00      0.98      0.99      1000
           2       0.86      0.89      0.88      1000
           3       0.93      0.90      0.92      1000
           4       0.90      0.83      0.87      1000
           5       0.98      0.98      0.98      1000
           6       0.81      0.67      0.73      1000
           7       0.95      0.97      0.96      1000
           8       0.99      0.98      0.98      1000
           9       0.97      0.96      0.96      1000

   micro avg       0.92      0.91      0.92     10000
   macro avg       0.92      0.91      0.91     10000
weighted avg       0.92      0.91      0.91     10000
 samples avg       0.91      0.91      0.91     10000
```

```
label_test = []
label_pred = []
for val in y_test:
    label_test.append(np.argmax(val))
for val in y_predictions:
    label_pred.append(np.argmax(val))
conf_mat = confusion_matrix(label_test,label_pred)

sns.set(font_scale=1.3)
fig = plt.figure(figsize=(14,12))
graph = sns.heatmap(conf_mat, annot=True, fmt='d')
graph.set(xlabel='Predicted Class', ylabel='Actual Class')
```

```
[Text(116.5, 0.5, 'Actual Class'), Text(0.5, 80.5, 'Predicted Class')]
```



LOGISTIC REGRESSION

```
train = pd.read_csv('/content/fashion-
mnist_train.csv',low_memory=False)
test = pd.read_csv('/content/fashion-mnist_test.csv',
low_memory=False)
```

```
print(train.shape)
print(test.shape)

(60000, 785)
(10000, 785)

train.head()

   label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7
pixel8  \
0      2       0       0       0       0       0       0       0
0
1      9       0       0       0       0       0       0       0
0
2      6       0       0       0       0       0       0       0
5
3      0       0       0       0       1       2       0       0
0
4      3       0       0       0       0       0       0       0
0

   pixel9  ...  pixel775  pixel776  pixel777  pixel778  pixel779
pixel780  \
0       0  ...         0         0         0         0         0
0
1       0  ...         0         0         0         0         0
0
2       0  ...         0         0         0        30        43
0
3       0  ...         3         0         0         0         0
1
4       0  ...         0         0         0         0         0
0

   pixel781  pixel782  pixel783  pixel784
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 785 columns]

test.head()

   label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7
pixel8  \
0      0       0       0       0       0       0       0       0
9
1      1       0       0       0       0       0       0       0
0
```

```
2      2       0       0       0       0       0       0      14
53
3      2       0       0       0       0       0       0       0
0
4      3       0       0       0       0       0       0       0
0

   pixel9  ...  pixel775  pixel776  pixel777  pixel778  pixel779
pixel780  \
0       8  ...       103        87        56         0         0
0
1       0  ...        34         0         0         0         0
0
2      99  ...         0         0         0         0        63
53
3       0  ...       137       126       140         0       133
224
4       0  ...         0         0         0         0         0
0

   pixel781  pixel782  pixel783  pixel784
0         0         0         0         0
1         0         0         0         0
2        31         0         0         0
3       222        56         0         0
4         0         0         0         0

[5 rows x 785 columns]
```

```
#drop label column
X_train= train.drop(['label'],axis = 1)
y_test = test.drop(['label'],axis = 1)
print(y_test.shape)
```

```
(10000, 784)
```

```
X_test = train['label']
X_test.shape
```

```
(60000,)
```

```
#converting to float and dividing the data by 255 gives all the data
in the range of 0 to 1.
X_train = X_train.astype('float32')
y_test = y_test.astype('float32')
X_train /= 255.0
y_test /=255.0
X_train.head()
```

```
   pixel1  pixel2  pixel3    pixel4    pixel5  pixel6  pixel7
pixel8  \
```

```
0     0.0     0.0     0.0  0.000000  0.000000     0.0     0.0
0.000000
1     0.0     0.0     0.0  0.000000  0.000000     0.0     0.0
0.000000
2     0.0     0.0     0.0  0.000000  0.000000     0.0     0.0
0.019608
3     0.0     0.0     0.0  0.003922  0.007843     0.0     0.0
0.000000
4     0.0     0.0     0.0  0.000000  0.000000     0.0     0.0
0.000000

   pixel9  pixel10  ...  pixel775  pixel776  pixel777  pixel778
pixel779  \
0     0.0      0.0  ...  0.000000       0.0       0.0  0.000000
0.000000
1     0.0      0.0  ...  0.000000       0.0       0.0  0.000000
0.000000
2     0.0      0.0  ...  0.000000       0.0       0.0  0.117647
0.168627
3     0.0      0.0  ...  0.011765       0.0       0.0  0.000000
0.000000
4     0.0      0.0  ...  0.000000       0.0       0.0  0.000000
0.000000

   pixel780  pixel781  pixel782  pixel783  pixel784
0  0.000000       0.0       0.0       0.0       0.0
1  0.000000       0.0       0.0       0.0       0.0
2  0.000000       0.0       0.0       0.0       0.0
3  0.003922       0.0       0.0       0.0       0.0
4  0.000000       0.0       0.0       0.0       0.0

[5 rows x 784 columns]
```

```
X_train, X_val, y_train, y_val = train_test_split(X_train, X_test,
test_size=0.1)

pca = PCA(n_components=100, random_state=42)
X_train_pca =pca.fit_transform(X_train)
X_test_pca = pca.transform(X_val)
y_test_pca =pca.transform(y_test)
X_train_PCA1 = pd.DataFrame(X_train_pca)
X_test_PCA1 = pd.DataFrame(X_test_pca)

logistic = LogisticRegression(max_iter=200,
solver='liblinear',penalty='l2')
logistic.fit(X_train_PCA1, y_train)

LogisticRegression(max_iter=200, solver='liblinear')

y_train_lr = logistic.predict(X_train_PCA1)
y_pred_lr = logistic.predict(X_test_pca)
```

```
log_train = metrics.accuracy_score(y_train,y_train_lr )
log_accuracy = metrics.accuracy_score(y_val, y_pred_lr)

print("Train Accuracy score for logisitic Regression is:
{}".format(log_train))
print("Test Accuracy score for logistic regression is:
{}".format(log_accuracy))
```

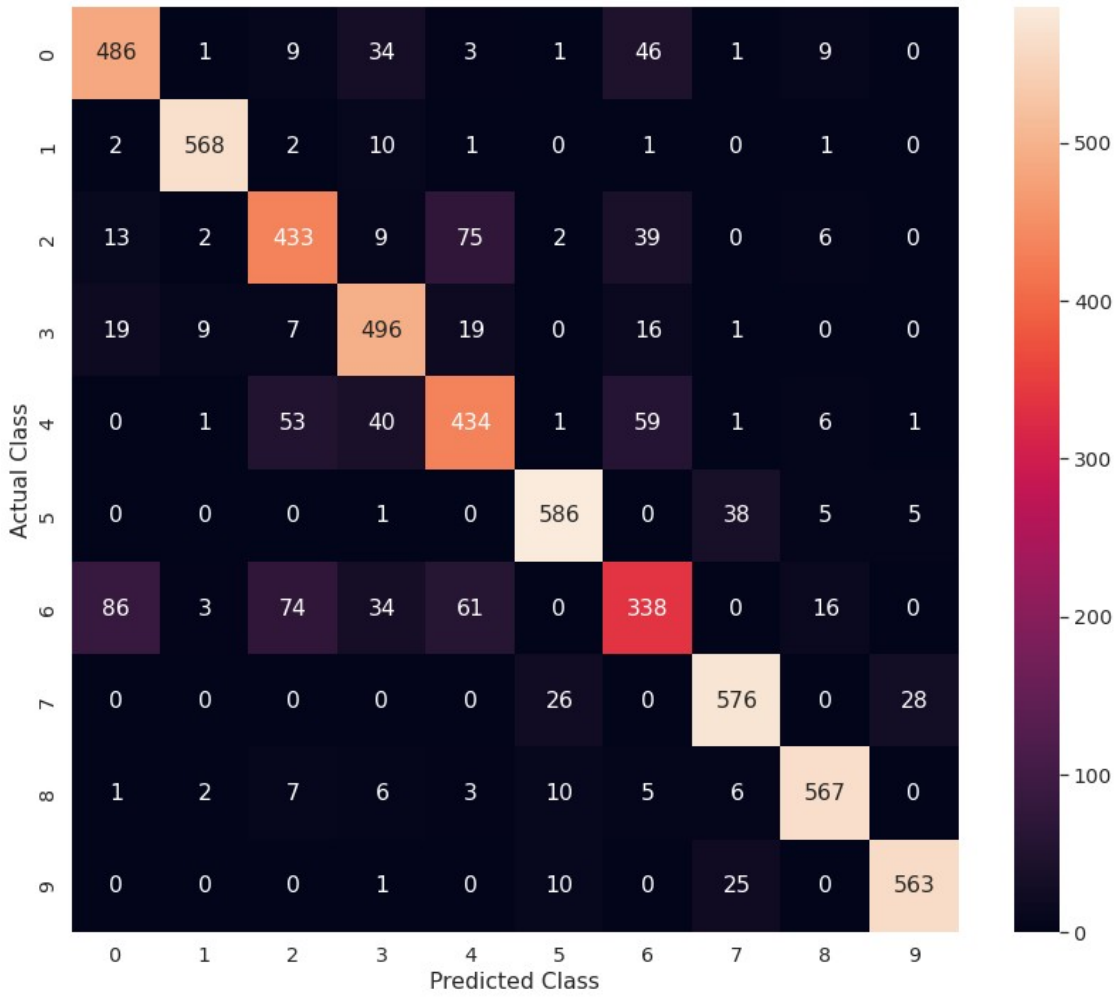
```
Train Accuracy score for logisitic Regression is: 0.8473888888888889
Test Accuracy score for logistic regression is: 0.8411666666666666
```

```
print("----------THE CLASSIFICATION REPORT FOR LOGISTIC REGRESSION
IS------------")
print(metrics.classification_report(y_val, y_pred_lr))
```

```
----------THE CLASSIFICATION REPORT FOR LOGISTIC REGRESSION
IS------------
              precision    recall  f1-score   support

           0       0.80      0.82      0.81       590
           1       0.97      0.97      0.97       585
           2       0.74      0.75      0.74       579
           3       0.79      0.87      0.83       567
           4       0.73      0.73      0.73       596
           5       0.92      0.92      0.92       635
           6       0.67      0.55      0.61       612
           7       0.89      0.91      0.90       630
           8       0.93      0.93      0.93       607
           9       0.94      0.94      0.94       599

    accuracy                           0.84      6000
   macro avg       0.84      0.84      0.84      6000
weighted avg       0.84      0.84      0.84      6000
```

```
con_matrix = pd.crosstab(pd.Series(y_val.values.flatten(),
name='Actual' ),pd.Series(y_pred_lr, name='Predicted'))
sns.set(font_scale=1.3)
fig = plt.figure(figsize=(14,12))
graph = sns.heatmap(con_matrix, annot=True, fmt='d')
graph.set(xlabel='Predicted Class', ylabel='Actual Class')
```

```
[Text(98.5, 0.5, 'Actual Class'), Text(0.5, 80.5, 'Predicted Class')]
```



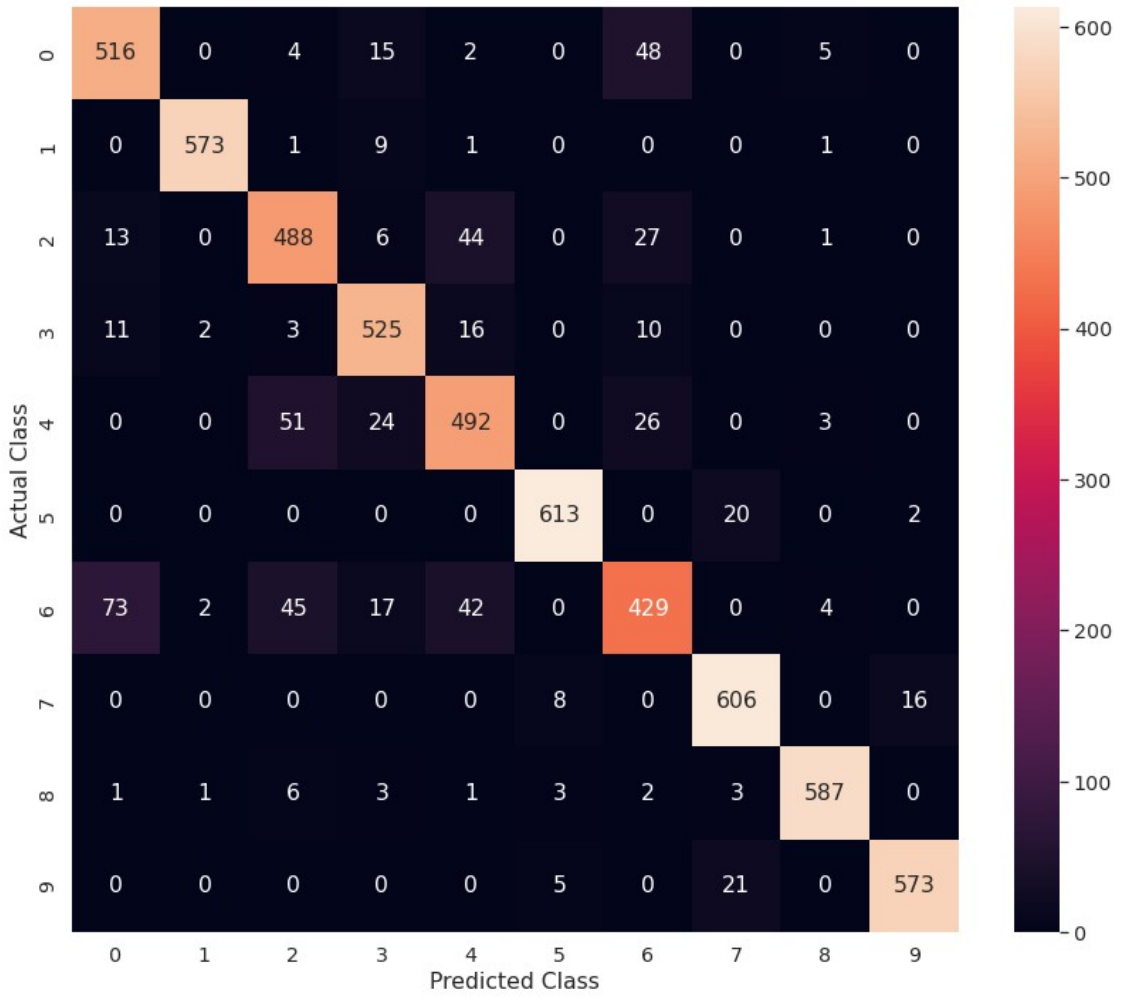
SUPPORT VECTOR MACHINE

```
from sklearn.svm import SVC
svc = SVC(C=13,kernel='rbf',gamma="auto",probability = True)
svc.fit(X_train_PCA1, y_train)
y_train_svc = svc.predict(X_train_PCA1)
y_pred_svc = svc.predict(X_test_pca)
a = metrics.accuracy_score(y_val, y_pred_svc)

print("Test Accuracy score for SVC is: {}".format(a))

Test Accuracy score for SVC is: 0.9003333333333333

print(metrics.classification_report(y_val, y_pred_lr))

              precision    recall  f1-score   support

           0       0.80      0.82      0.81       590
           1       0.97      0.97      0.97       585
           2       0.74      0.75      0.74       579
```

```
              3       0.79      0.87      0.83       567
              4       0.73      0.73      0.73       596
              5       0.92      0.92      0.92       635
              6       0.67      0.55      0.61       612
              7       0.89      0.91      0.90       630
              8       0.93      0.93      0.93       607
              9       0.94      0.94      0.94       599

       accuracy                           0.84      6000
      macro avg       0.84      0.84      0.84      6000
   weighted avg       0.84      0.84      0.84      6000
```

```
con_matrix = pd.crosstab(pd.Series(y_val.values.flatten(),
name='Actual' ),pd.Series(y_pred_svc, name='Predicted'))
sns.set(font_scale=1.3)
fig = plt.figure(figsize=(14,12))
graph = sns.heatmap(con_matrix, annot=True, fmt='d')
graph.set(xlabel='Predicted Class', ylabel='Actual Class')
```

```
[Text(98.5, 0.5, 'Actual Class'), Text(0.5, 80.5, 'Predicted Class')]
```



SVC WITH GAMMA SET TO 'SCALE'

```
svc = SVC(C=13,kernel='rbf',gamma="scale",probability = True)
svc.fit(X_train_PCA1, y_train)
y_train_svc = svc.predict(X_train_PCA1)
y_pred_svc = svc.predict(X_test_pca)
a = metrics.accuracy_score(y_val, y_pred_svc)

print("Test Accuracy score for SVC is: {}".format(a))
```

```
Test Accuracy score for SVC is: 0.9058333333333334
```

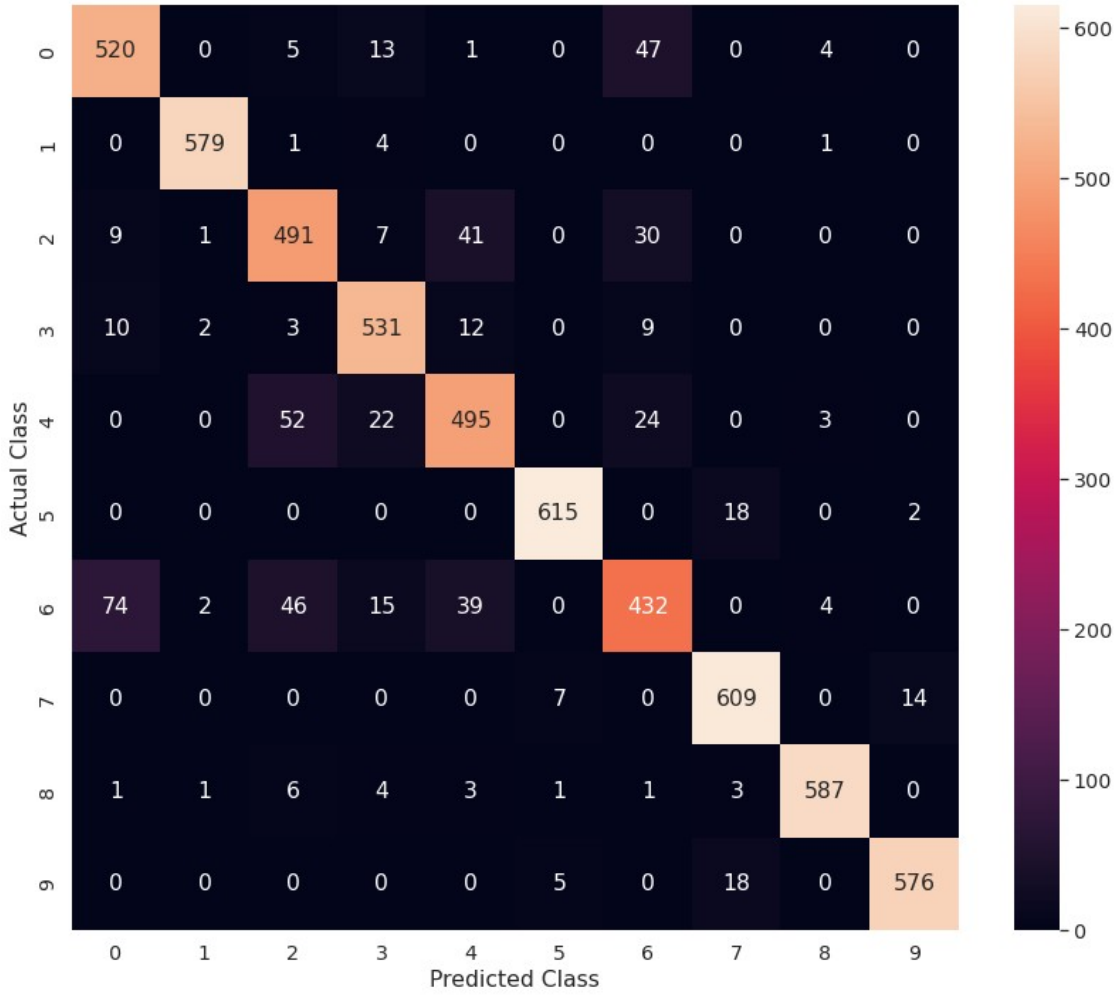
```
print(metrics.classification_report(y_val, y_pred_lr))
```

```
              precision    recall  f1-score   support

           0       0.80      0.82      0.81       590
           1       0.97      0.97      0.97       585
           2       0.74      0.75      0.74       579
           3       0.79      0.87      0.83       567
```

```
              4       0.73      0.73      0.73       596
              5       0.92      0.92      0.92       635
              6       0.67      0.55      0.61       612
              7       0.89      0.91      0.90       630
              8       0.93      0.93      0.93       607
              9       0.94      0.94      0.94       599

    accuracy                           0.84      6000
   macro avg       0.84      0.84      0.84      6000
weighted avg       0.84      0.84      0.84      6000
```

```
con_matrix = pd.crosstab(pd.Series(y_val.values.flatten(),
name='Actual' ),pd.Series(y_pred_svc, name='Predicted'))
sns.set(font_scale=1.3)
fig = plt.figure(figsize=(14,12))
graph = sns.heatmap(con_matrix, annot=True, fmt='d')
graph.set(xlabel='Predicted Class', ylabel='Actual Class')
```

```
[Text(98.5, 0.5, 'Actual Class'), Text(0.5, 80.5, 'Predicted Class')]
```



KNN

```
traindata = pd.read_csv("fashion-mnist_train.csv")
testdata = pd.read_csv("fashion-mnist_test.csv")

data_train=traindata.iloc[:,1:785]/ 255.0
label_train=pd.DataFrame([traindata.iloc[:,0]]).T
data_test=testdata.iloc[:,0:784]/ 255.0

label_train.value_counts()
```

```
label
0        6000
1        6000
2        6000
3        6000
4        6000
5        6000
6        6000
7        6000
8        6000
9        6000
dtype: int64
```

```
categoryMap=
{0 :'T-shirt/Top',
1 :'Trouser',
2 :'Pullover',
3 :'Dress',
4 :'Coat',
5 :'Sandal',
6 :'Shirt',
7 :'Sneaker',
8 :'Bag',
9 :'Ankle boot'}

label_train['category']=label_train['label'].map(categoryMap)

l_train=pd.DataFrame([traindata.iloc[:,0]]).T
X_train, X_val, Y_train, Y_val = train_test_split(data_train, l_train,
test_size = 0.25, random_state=255)

np.mean(X_train.values),np.std(X_train.values),np.mean(X_val.values),n
p.std(X_val.values)
```

```
(0.28573853674803273,
 0.3525878811647684,
 0.2872052621048414,
 0.35347813709187687)
```

```
X_train=StandardScaler().fit_transform(X_train)
X_val=StandardScaler().fit_transform(X_val)
np.mean(X_train),np.std(X_train),np.mean(X_val),np.std(X_val)
```

```
(-7.554551592506054e-19,
 1.0000000000000002,
 -1.4778805541404805e-18,
 0.9999999999999989)
```

```
column_name=['pixel'+str(i) for i in range(1,785)]
X_train = pd.DataFrame(X_train,columns =column_name)
X_val = pd.DataFrame(X_val,columns =column_name)
```

```
pca = PCA(n_components=0.9,copy=True, whiten=False)
X_train = pca.fit_transform(X_train)
X_val = pca.transform(X_val)
print(pca.explained_variance_ratio_)
```

```
[0.22021041 0.14384998 0.0547689  0.05131468 0.04063138 0.03007163
 0.02747667 0.02328081 0.016965   0.01312661 0.01167599 0.00966141
 0.00899778 0.00863649 0.00742566 0.0073279  0.0065662  0.00632188
 0.00623672 0.00579956 0.00518253 0.0051314  0.00473594 0.00455951
 0.00437728 0.00416881 0.00393644 0.00392189 0.00382522 0.0037333
 0.00371573 0.00351303 0.00336553 0.00328974 0.00328232 0.00319012
 0.0030556  0.00294426 0.0028811  0.00282458 0.0027157  0.00266539
 0.00258201 0.00254747 0.00247826 0.00245517 0.0023832  0.00227583
 0.00224023 0.0021398  0.00210534 0.00209826 0.00202363 0.00201152
 0.00197738 0.00195216 0.00191349 0.00184274 0.00183835 0.00179505
 0.00176205 0.00174474 0.00172148 0.00168596 0.00162911 0.00157656
 0.00155699 0.00151357 0.00148351 0.00147275 0.00144672 0.00144151
 0.00142568 0.00141295 0.00137972 0.00136985 0.00134535 0.00130561
 0.00129231 0.00126802 0.0012613  0.00123772 0.00121467 0.00119871
 0.00117982 0.00115049 0.00114735 0.00112809 0.00111563 0.00110085
 0.00109581 0.00107948 0.00106858 0.0010493  0.0010227  0.00100247
 0.00099479 0.00098044 0.00097036 0.0009612  0.00094554 0.00093938
 0.0009296  0.0009156  0.00091294 0.00090078 0.00088439 0.00088154
 0.00086715 0.00085786 0.00085174 0.00083828 0.00082762 0.0008172
 0.00080943 0.00080201 0.00079327 0.00079122 0.00078554 0.00077271
 0.00076404 0.00075755 0.00075261 0.00074189 0.00073685 0.00072887
 0.00071958 0.00070972 0.00070701 0.00070049 0.00069595 0.00068553
 0.00067991 0.00066419 0.00065905 0.00065627]
```

```
pcn=X_train.shape[1]
pcn
```

```
136
```

```
X_train = pd.DataFrame(X_train,columns =column_name[0:pcn])
X_val = pd.DataFrame(X_val,columns =column_name[0:pcn])
```

```
k = 6
cross_val = KFold(k, shuffle=True, random_state=1)
```

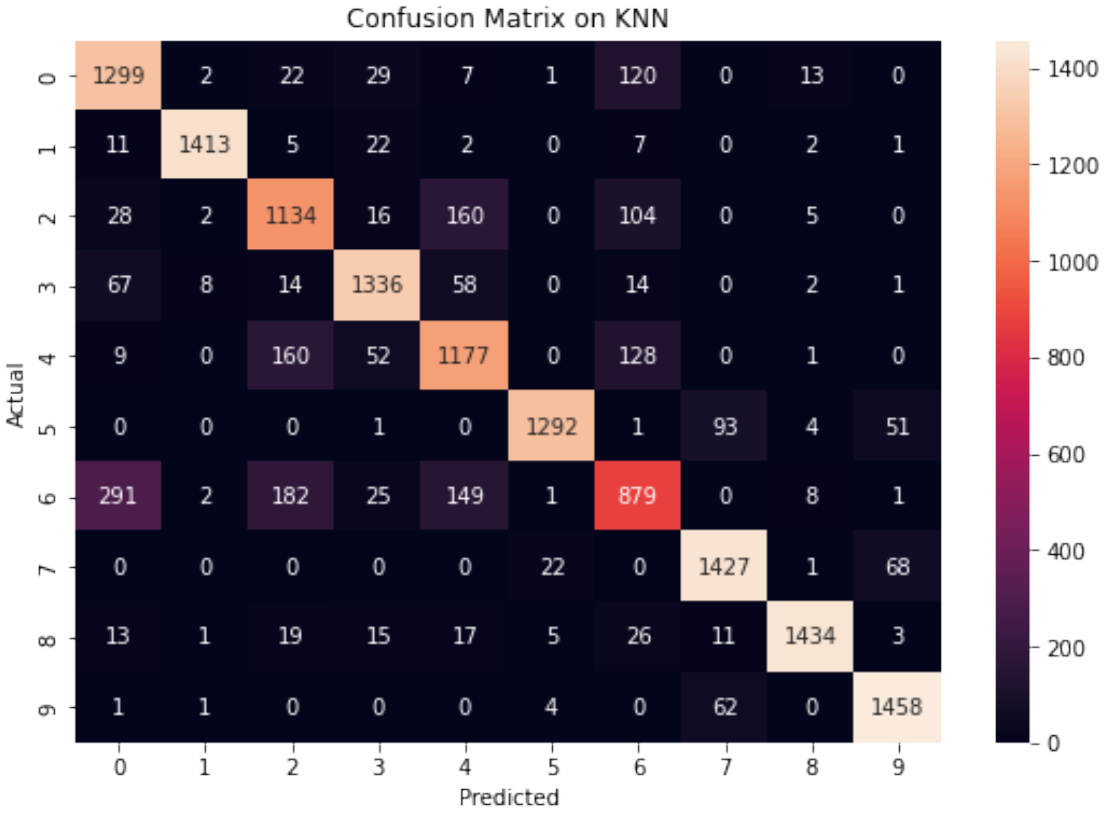
```
fold_count = 1

# For training epochs
epochs = 32

# For loss & acc plotting
histories = []

# For testing/evaluation acc scores
eval_scores = []

# For callbacks
es_callbacks = keras.callbacks.EarlyStopping(monitor="val_loss",
                                             mode="min",
                                             verbose=1,
                                             patience=4)

knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(X_train, Y_train.values.ravel())
y_train_prd = knn.predict(X_train)
y_val_prd = knn.predict(X_val)
acc_train_knn=accuracy_score(Y_train,y_train_prd )
acc_val_knn=accuracy_score(Y_val,y_val_prd)
print("accuracy on train set:{:.4f}\naccuracy on validation set:
{:.4f}".format(acc_train_knn,acc_val_knn))
```

```
accuracy on train set:0.8908
accuracy on validation set:0.8566
```

```
print("--------------THE CLASSFICATION REPORT FOR KNN---------------")
print(metrics.classification_report(Y_val, y_val_prd))
```

```
--------------THE CLASSFICATION REPORT FOR KNN---------------
              precision    recall  f1-score   support

           0       0.76      0.87      0.81      1493
           1       0.99      0.97      0.98      1463
           2       0.74      0.78      0.76      1449
           3       0.89      0.89      0.89      1500
           4       0.75      0.77      0.76      1527
           5       0.98      0.90      0.93      1442
           6       0.69      0.57      0.62      1538
           7       0.90      0.94      0.92      1518
           8       0.98      0.93      0.95      1544
           9       0.92      0.96      0.94      1526

    accuracy                           0.86     15000
   macro avg       0.86      0.86      0.86     15000
weighted avg       0.86      0.86      0.86     15000
```

```
con_matrix = pd.crosstab(pd.Series(Y_val.values.flatten(),
name='Actual' ),pd.Series(y_val_prd, name='Predicted'))
plt.figure(figsize = (9,6))
plt.title("Confusion Matrix on KNN")
sns.heatmap(con_matrix, annot=True, fmt='g')
plt.show()
```



NAIVE BAYES

```
NB = GaussianNB()
NB.fit(X_train, Y_train.values.ravel())
y_train_prd = NB.predict(X_train)
y_val_prd = NB.predict(X_val)
acc_train_nb=accuracy_score(Y_train,y_train_prd )
acc_val_nb=accuracy_score(Y_val,y_val_prd)
print("accuracy on train set:{:.4f}\naccuracy on validation set:
{:.4f}".format(acc_train_nb, acc_val_nb))
```

```
accuracy on train set:0.6654
accuracy on validation set:0.6655
```

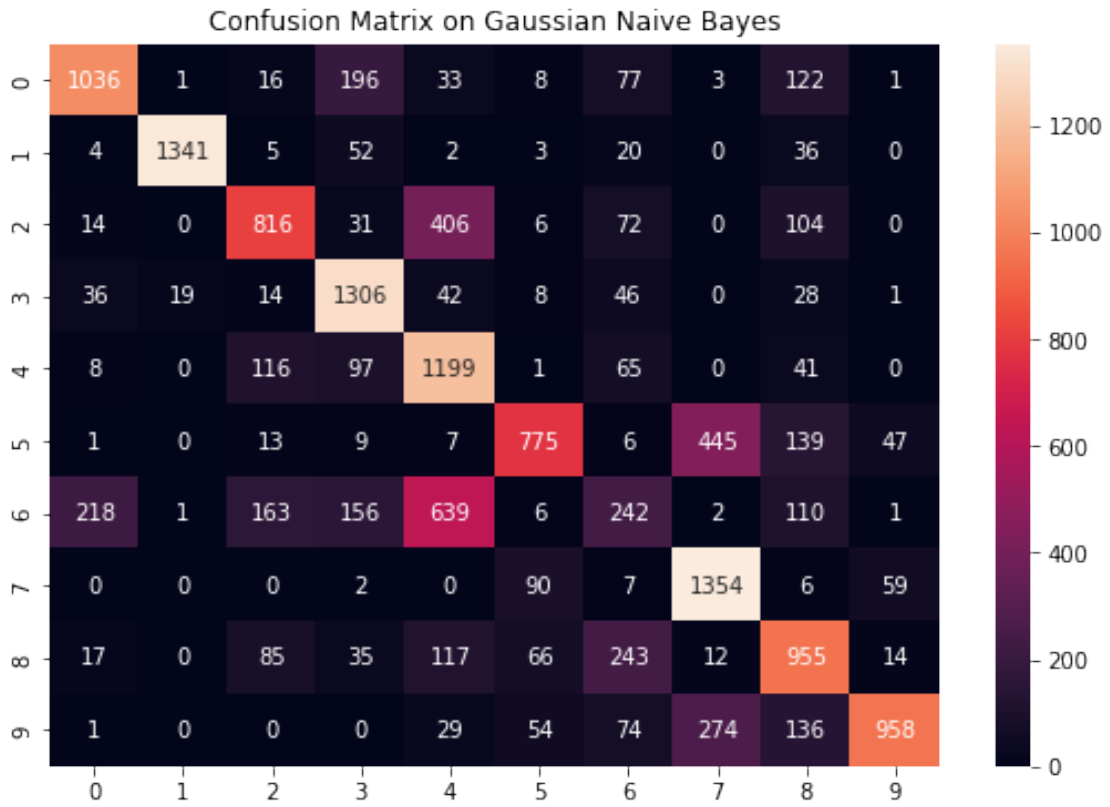
```
y_pred = NB.predict(X_val)
y_pred
```

```
array([0, 4, 8, ..., 7, 0, 7])
```

```
print("--------------CLASSIFICATION REPORT OF NAIVE
BAYES-----------------")
print(metrics.classification_report(Y_val, y_pred))
```

```
--------------CLASSIFICATION REPORT OF NAIVE BAYES-----------------
              precision    recall  f1-score   support

           0       0.78      0.69      0.73      1493
           1       0.98      0.92      0.95      1463
           2       0.66      0.56      0.61      1449
           3       0.69      0.87      0.77      1500
           4       0.48      0.79      0.60      1527
           5       0.76      0.54      0.63      1442
           6       0.28      0.16      0.20      1538
           7       0.65      0.89      0.75      1518
           8       0.57      0.62      0.59      1544
           9       0.89      0.63      0.73      1526

    accuracy                           0.67     15000
   macro avg       0.68      0.67      0.66     15000
weighted avg       0.67      0.67      0.66     15000
```

```
con_matrix = pd.crosstab(pd.Series(Y_val.values.flatten(),
name='Actual' ),pd.Series(y_val_prd, name='Predicted'))
plt.figure(figsize = (9,6))
plt.title("Confusion Matrix on Gaussian Naive Bayes")
cm = confusion_matrix(Y_val, y_pred)
sns.heatmap(cm, annot=True, fmt='g')
plt.show()
```



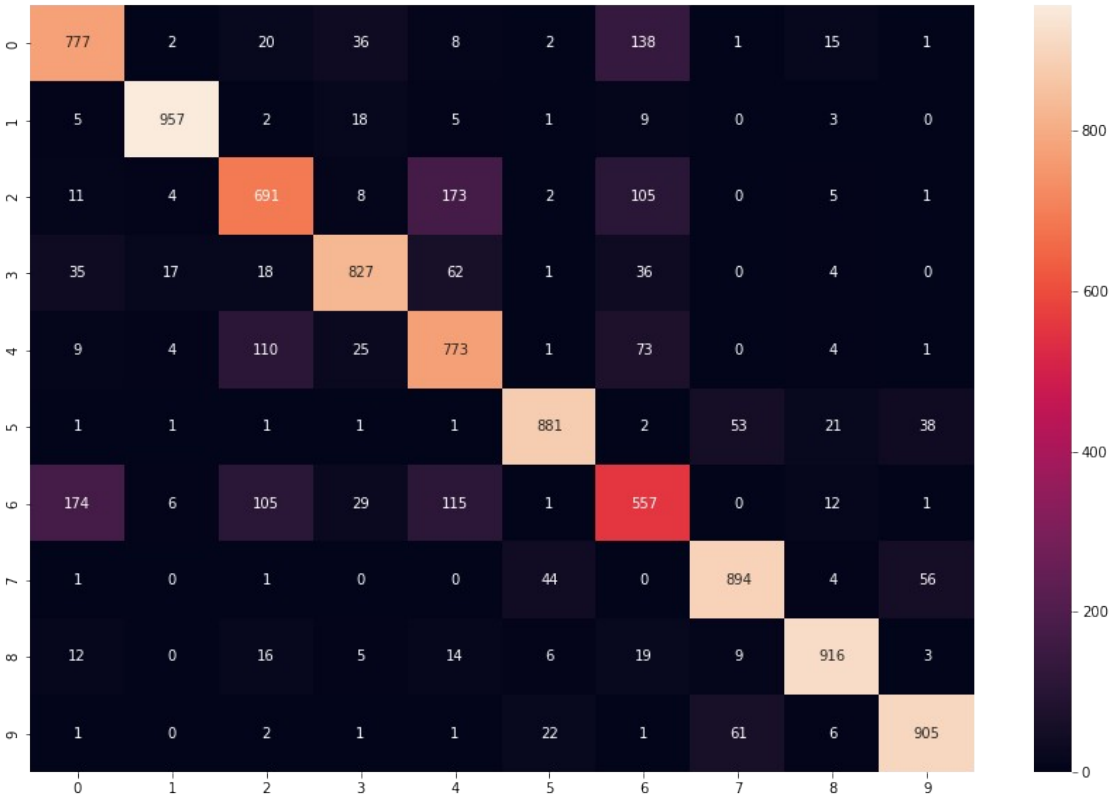
DECISION TREE

```
X_test = testdata.drop(['label'], axis = 1)
y_test = testdata.label
X = traindata.drop(['label'], axis = 1)
y = traindata.label

parameters = {'max_depth' : [13,16,19]}

tree_clf = DecisionTreeClassifier()
grid_search_cv_clf_tree = GridSearchCV(tree_clf, parameters, cv = 5)
grid_search_cv_clf_tree.fit(X, y)

GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [13, 16, 19]})

y_pred=grid_search_cv_clf_tree.predict(X_test)

print("----------CLASSFICATION REPORT OF DECISION TREE----------")
print(metrics.classification_report(y_test, y_pred))

----------CLASSFICATION REPORT OF DECISION TREE----------
              precision    recall  f1-score   support

           0       0.76      0.78      0.77      1000
           1       0.97      0.96      0.96      1000
```

```
              2       0.72      0.69      0.70      1000
              3       0.87      0.83      0.85      1000
              4       0.67      0.77      0.72      1000
              5       0.92      0.88      0.90      1000
              6       0.59      0.56      0.57      1000
              7       0.88      0.89      0.89      1000
              8       0.93      0.92      0.92      1000
              9       0.90      0.91      0.90      1000

       accuracy                           0.82     10000
      macro avg       0.82      0.82      0.82     10000
   weighted avg       0.82      0.82      0.82     10000
```

```
cm_tree=confusion_matrix(y_test, y_pred)
plt.figure(figsize=(15,10))
sns.heatmap(cm_tree, annot=True, fmt='g')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f33e01dc880>
```



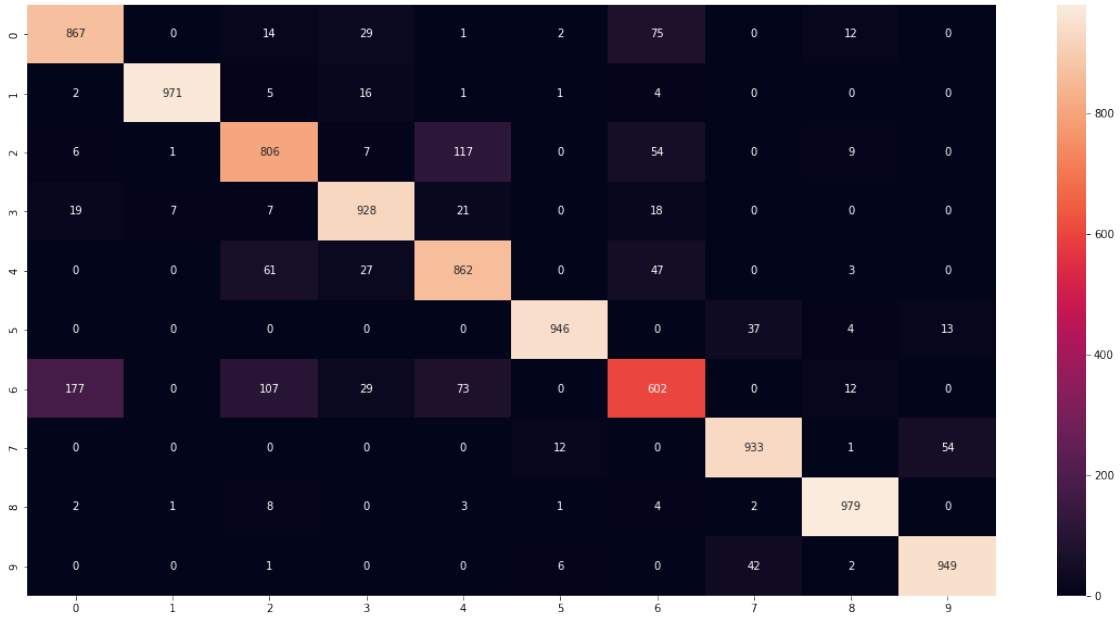

```
best_clf_tree = grid_search_cv_clf_tree.best_estimator_
```

```
a= best_clf_tree.score(X_test,y_test)*100
print("The accuracy for Decision tree is:{:.2f} ".format(a))
```

```
The accuracy for Decision tree is:81.78
```

RANDOM FOREST

```
clf_rf = RandomForestClassifier()

clf_rf.fit(X,y)

RandomForestClassifier()

b= clf_rf.score(X_test,y_test)*100
print("The accuracy for Random Forest is: {:.2f} ".format(b))

The accuracy for Random Forest is: 88.29

parameters = {'n_estimators':[30, 60, 90],
              'max_depth': [9, 15, 22]}

grid_search_cv_clf_rf = GridSearchCV(clf_rf, parameters, cv = 4)

grid_search_cv_clf_rf.fit(X, y)

GridSearchCV(cv=4, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [9, 15, 22],
                         'n_estimators': [30, 60, 90]})

y_pred_rf=grid_search_cv_clf_rf.predict(X_test)
print(metrics.classification_report(y_test, y_pred_rf))

              precision    recall  f1-score   support

           0       0.81      0.87      0.84      1000
           1       0.99      0.97      0.98      1000
           2       0.80      0.81      0.80      1000
           3       0.90      0.93      0.91      1000
           4       0.80      0.86      0.83      1000
           5       0.98      0.95      0.96      1000
           6       0.75      0.60      0.67      1000
           7       0.92      0.93      0.93      1000
           8       0.96      0.98      0.97      1000
           9       0.93      0.95      0.94      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000


cm=confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(20,10))
sns.heatmap(cm, annot=True,fmt='g')

<matplotlib.axes._subplots.AxesSubplot at 0x7f33ce217640>
```




```
best_clf_rf = grid_search_cv_clf_rf.best_estimator_
b=best_clf_rf.score(X_test,y_test)*100
print("The accuracy for Random Forest is: {:.2f} ".format(b))
```

```
The accuracy for Random Forest is: 88.43
```