ABSTARCT

In this project, we have built a fashion apparel recognition using the Convolutional Neural Network (CNN) model. To train the CNN model, we have used the Fashion MNIST dataset. After successful training, the CNN model can predict the name of the class given apparel item belongs to. This is a multiclass classification problem in which there are 10 apparel classes the items will be classified.

The fashion training set consists of 70,000 images divided into 60,000 training and 10,000 testing samples. Dataset sample consists of 28x28 grayscale images, associated with a label from 10 classes.

So the end goal is to train and test the model using Convolution neural network

OBJECTIVE

This work is part of my experiments with Fashion-MNIST dataset using various Machine Learning algorithms/models. The objective is to identify (predict) different fashion products from the given images using various best possible Machine Learning Models (Algorithms) and compare their results (performance measures/scores) to arrive at the best ML model. I have also experimented with 'dimensionality reduction' technique for this problem.

INTRODUCTION

One of the classic problem that has been used in the Machine Learning world for quite sometime is the MNIST problem. The objective is to identify the digit based on image. But MNIST is not very great problem because we come up with great accuracy even if we are looking at few pixels in the image. So, another common example problem against which we test algorithms is Fashion-MNIST.

Fashion-MNIST is a dataset of Zalando's fashion article images —consisting of a **training set of 60,000** examples and **a test set of 10,000** examples. Each instance is a 28×28 grayscale image, associated with a label.

The 'target' dataset has 10 class labels, as we can see from above (0 – T-shirt/top, 1 – Trouser,,….9 – Ankle Boot).

Given the images of the articles, we need to classify them into one of these classes, hence, it is essentially a **'Multi-class Classification'** problem.

We will be using various Classifiers and comparing their results/scores.

METHODOLOGY

The Fashion MNIST dataset was developed as a response to the wide use of the MNIST dataset, that has been effectively "*solved*" given the use of modern convolutional neural networks.

Fashion-MNIST was proposed to be a replacement for MNIST, and although it has not been solved, it is possible to routinely achieve error rates of 10% or less. Like MNIST, it can be a useful starting point for developing and practicing a methodology for solving image classification using convolutional neural networks.

Instead of reviewing the literature on well-performing models on the dataset, we can develop a new model from scratch.

The dataset already has a well-defined train and test dataset that we can use.

In order to estimate the performance of a model for a given training run, we can further split the training set into a train and validation dataset. Performance on the train and validation dataset over each run can then be plotted to provide learning curves and insight into how well a model is learning the problem.

The Keras API supports this by specifying the "*validation_data*" argument to the *model.fit()* function when training the model, that will, in turn, return an object that describes model performance for the chosen loss and metrics on each training epoch.

```
1 # record model performance on a validation dataset during training
2 history = model.fit(..., validation_data=(valX, valY))
```

In order to estimate the performance of a model on the problem in general, we can use k-fold cross-validation, perhaps 5-fold cross-validation. This will give some account of the model's variance with both respect to differences in the training and test datasets and the stochastic nature of the learning algorithm. The performance of a model can be taken as the

mean performance across k-folds, given with the standard deviation, that could be used to estimate a confidence interval if desired.

We can use the KFold class from the scikit-learn API to implement the k-fold cross-validation evaluation of a given neural network model. There are many ways to achieve this, although we can choose a flexible approach where the KFold is only used to specify the row indexes used for each split.

```
# example of k-fold cv for a neural net
data = ...
# prepare cross validation
kfold = KFold(5, shuffle=True, random_state=1)
# enumerate splits
for train_ix, test_ix in kfold.split(data):
	model = ...

```

CODE

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import keras

(X_train, y_train), (X_test, y_test)
=tf.keras.datasets.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 [==============================] - 0s 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
```

```
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

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

```
X_train[0]
```

```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,
          0,   0,  13,  73,   0,   0,   1,   4,   0,   0,   0,   0,   1,
          1,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
          0,  36, 136, 127,  62,  54,   0,   0,   0,   1,   3,   4,   0,
          0,   3],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,
```

```
   0, 102, 204, 176, 134, 144, 123,  23,   0,   0,   0,   0,  12,
  10,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0, 155, 236, 207, 178, 107, 156, 161, 109,  64,  23,  77, 130,
  72,  15],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,
  69, 207, 223, 218, 216, 216, 163, 127, 121, 122, 146, 141,  88,
 172,  66],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   1,   1,   0,
 200, 232, 232, 233, 229, 223, 223, 215, 213, 164, 127, 123, 196,
 229,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
 183, 225, 216, 223, 228, 235, 227, 224, 222, 224, 221, 223, 245,
 173,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
 193, 228, 218, 213, 198, 180, 212, 210, 211, 213, 223, 220, 243,
 202,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   3,   0,  12,
 219, 220, 212, 218, 192, 169, 227, 208, 218, 224, 212, 226, 197,
 209,  52],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,   0,  99,
 244, 222, 220, 218, 203, 198, 221, 215, 213, 222, 220, 245, 119,
 167,  56],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   4,   0,   0,  55,
 236, 228, 230, 228, 240, 232, 213, 218, 223, 234, 217, 217, 209,
  92,   0],
[  0,   0,   1,   4,   6,   7,   2,   0,   0,   0,   0,   0, 237,
 226, 217, 223, 222, 219, 222, 221, 216, 223, 229, 215, 218, 255,
  77,   0],
[  0,   3,   0,   0,   0,   0,   0,   0,   0,  62, 145, 204, 228,
 207, 213, 221, 218, 208, 211, 218, 224, 223, 219, 215, 224, 244,
 159,   0],
[  0,   0,   0,   0,  18,  44,  82, 107, 189, 228, 220, 222, 217,
 226, 200, 205, 211, 230, 224, 234, 176, 188, 250, 248, 233, 238,
 215,   0],
[  0,  57, 187, 208, 224, 221, 224, 208, 204, 214, 208, 209, 200,
 159, 245, 193, 206, 223, 255, 255, 221, 234, 221, 211, 220, 232,
 246,   0],
[  3, 202, 228, 224, 221, 211, 211, 214, 205, 205, 205, 220, 240,
  80, 150, 255, 229, 221, 188, 154, 191, 210, 204, 209, 222, 228,
 225,   0],
[ 98, 233, 198, 210, 222, 229, 229, 234, 249, 220, 194, 215, 217,
 241,  65,  73, 106, 117, 168, 219, 221, 215, 217, 223, 223, 224,
 229,  29],
[ 75, 204, 212, 204, 193, 205, 211, 225, 216, 185, 197, 206, 198,
 213, 240, 195, 227, 245, 239, 223, 218, 212, 209, 222, 220, 221,
 230,  67],
[ 48, 203, 183, 194, 213, 197, 185, 190, 194, 192, 202, 214, 219,
 221, 220, 236, 225, 216, 199, 206, 186, 181, 177, 172, 181, 205,
 206, 115],
```

```
[  0, 122, 219, 193, 179, 171, 183, 196, 204, 210, 213, 207, 211,
 210, 200, 196, 194, 191, 195, 191, 198, 192, 176, 156, 167, 177,
 210,  92],
[  0,   0,  74, 189, 212, 191, 175, 172, 175, 181, 185, 188, 189,
 188, 193, 198, 204, 209, 210, 210, 211, 188, 188, 194, 192, 216,
 170,   0],
[  2,   0,   0,   0,  66, 200, 222, 237, 239, 242, 246, 243, 244,
 221, 220, 193, 191, 179, 182, 182, 181, 176, 166, 168,  99,  58,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,  40,  61,  44,  72,  41,  35,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0]], dtype=uint8)
```

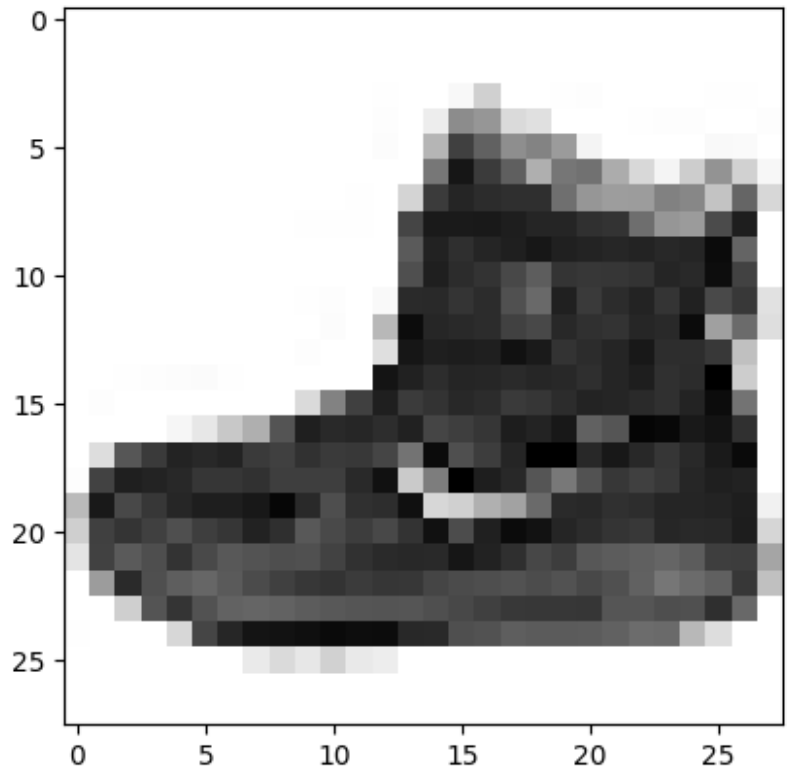
```
y_train[0]
```

```
9
```

```
class_labels=[    "T-
shirt/top","Trouser","Pullover","Dress","Coat","Sandal","Shirt",'Sneaker','Ba
g','Ankle boot']
```

```
plt.imshow(X_train[0],cmap='Greys')
```

```
<matplotlib.image.AxesImage at 0x7f28428e6710>
```




```
plt.figure(figsize=(16,16))
j=1
for i in np.random.randint(0,1000,25):
    plt.subplot(5,5,j); j=j+1
    plt.imshow(X_train[i],cmap="Greys")
    plt.axis('off')
    plt.title('{} / {}'.format(class_labels[y_train[i]],y_train[i]))
```


```
X_train.ndim
```

```
3
```

```
X_train=np.expand_dims(X_train,-1)
```

```
X_train.ndim
```

```
4
```

```
X_test=np.expand_dims(X_test,-1)
```

```
X_test=X_test/255
X_train=X_train/255
```

```
from sklearn.model_selection import train_test_split
X_train,X_Validation,y_train,y_Validation=train_test_split(X_train,y_train,te
st_size=0.2,random_state=2020)
```

```
X_train.shape,X_Validation.shape,y_train.shape,y_Validation.shape
```

```
((48000, 28, 28, 1), (12000, 28, 28, 1), (48000,), (12000,))
```

```
model=keras.models.Sequential([

keras.layers.Conv2D(filters=32,kernel_size=3,strides=(1,1),padding='valid',ac
tivation='relu',input_shape=[28,28,1]),
                         keras.layers.MaxPooling2D(pool_size=(2,2)),
                         keras.layers.Flatten(),
                         keras.layers.Dense(units=128,activation='relu'),
                         keras.layers.Dense(units=10,activation='softmax')
])


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 flatten (Flatten)           (None, 5408)              0

 dense (Dense)               (None, 128)               692352

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 693,962
Trainable params: 693,962
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics
=['accuracy'])

model.fit(X_train,y_train,epochs=10,batch_size=512,verbose=1,validation_data=
(X_Validation,y_Validation))
```

```
Epoch 1/10
94/94 [==============================] - 27s 272ms/step - loss: 0.6406 -
accuracy: 0.7817 - val_loss: 0.4444 - val_accuracy: 0.8413
Epoch 2/10
94/94 [==============================] - 27s 283ms/step - loss: 0.3895 -
accuracy: 0.8643 - val_loss: 0.3766 - val_accuracy: 0.8676
Epoch 3/10
```

```
94/94 [==============================] - 25s 272ms/step - loss: 0.3362 -
accuracy: 0.8834 - val_loss: 0.3382 - val_accuracy: 0.8819
Epoch 4/10
94/94 [==============================] - 40s 426ms/step - loss: 0.3059 -
accuracy: 0.8908 - val_loss: 0.3240 - val_accuracy: 0.8863
Epoch 5/10
94/94 [==============================] - 37s 392ms/step - loss: 0.2845 -
accuracy: 0.8987 - val_loss: 0.3064 - val_accuracy: 0.8928
Epoch 6/10
94/94 [==============================] - 41s 432ms/step - loss: 0.2687 -
accuracy: 0.9046 - val_loss: 0.2982 - val_accuracy: 0.8952
Epoch 7/10
94/94 [==============================] - 29s 301ms/step - loss: 0.2524 -
accuracy: 0.9096 - val_loss: 0.2844 - val_accuracy: 0.9011
Epoch 8/10
94/94 [==============================] - 28s 295ms/step - loss: 0.2353 -
accuracy: 0.9154 - val_loss: 0.2831 - val_accuracy: 0.9016
Epoch 9/10
94/94 [==============================] - 25s 269ms/step - loss: 0.2250 -
accuracy: 0.9179 - val_loss: 0.2734 - val_accuracy: 0.9032
Epoch 10/10
94/94 [==============================] - 25s 270ms/step - loss: 0.2166 -
accuracy: 0.9204 - val_loss: 0.2617 - val_accuracy: 0.9063

<keras.callbacks.History at 0x7f2836717190>

y_pred = model.predict(X_test)
y_pred.round(2)

313/313 [==============================] - 3s 10ms/step

array([[0.  , 0.  , 0.  , ..., 0.01, 0.  , 0.99],
       [0.  , 0.  , 1.  , ..., 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , ..., 0.  , 0.  , 0.  ],
       ...,
       [0.  , 0.  , 0.  , ..., 0.  , 1.  , 0.  ],
       [0.  , 1.  , 0.  , ..., 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.01, ..., 0.26, 0.08, 0.  ]], dtype=float32)

y_test

array([9, 2, 1, ..., 8, 1, 5], dtype=uint8)

model.evaluate(X_test, y_test)

313/313 [==============================] - 2s 6ms/step - loss: 0.2747 -
accuracy: 0.8961

[0.27465856075286865, 0.8960999846458435]

plt.figure(figsize=(16,16))
```

```
j=1
for i in np.random.randint(0, 1000,25):
  plt.subplot(5,5, j); j+=1
  plt.imshow(X_test[i].reshape(28,28), cmap = 'Greys')
  plt.title('Actual = {} / {} \nPredicted = {} /
{}'.format(class_labels[y_test[i]], y_test[i],
class_labels[np.argmax(y_pred[i])],np.argmax(y_pred[i])))
  plt.axis('off')
```


```
plt.figure(figsize=(16,30))

j=1
for i in np.random.randint(0, 1000,60):
  plt.subplot(10,6, j); j+=1
  plt.imshow(X_test[i].reshape(28,28), cmap = 'Greys')
  plt.title('Actual = {} / {} \nPredicted = {} /
```

```
{}'.format(class_labels[y_test[i]], y_test[i],
class_labels[np.argmax(y_pred[i])],np.argmax(y_pred[i])))
  plt.axis('off')
```


Actual = Sandal / 5
Predicted = Sandal / 5
Actual = Dress / 3
Predicted = Dress / 3
Actual = Dress / 3
Predicted = Dress / 3
Actual = Bag / 8
Predicted = Bag / 8
Actual = Sneaker / 7
Predicted = Sneaker / 7
Actual = Trouser / 1
Predicted = Trouser / 1
Actual = Pullover / 2
Predicted = Pullover / 2
Actual = Bag / 8
Predicted = Bag / 8
Actual = Dress / 3
Predicted = Dress / 3
Actual = Sandal / 5
Predicted = Sandal / 5
Actual = Sneaker / 7
Predicted = Sneaker / 7
Actual = Pullover / 2
Predicted = Pullover / 2
Actual = Pullover / 2
Predicted = Pullover / 2
Actual = Dress / 3
Predicted = Dress / 3
Actual = Dress / 3
Predicted = Dress / 3
Actual = Pullover / 2
Predicted = Pullover / 2
Actual = Dress / 3
Predicted = Dress / 3
Actual = T-shirt/top / 0
Predicted = T-shirt/top / 0
Actual = Trouser / 1
Predicted = Trouser / 1
Actual = Coat / 4
Predicted = Coat / 4
Actual = Coat / 4
Predicted = Coat / 4
Actual = Dress / 3
Predicted = Dress / 3
Actual = T-shirt/top / 0
Predicted = T-shirt/top / 0
Actual = Dress / 3
Predicted = Coat / 4
Actual = Dress / 3
Predicted = Dress / 3
Actual = Dress / 3
Predicted = Dress / 3
Actual = T-shirt/top / 0
Predicted = T-shirt/top / 0
Actual = Coat / 4
Predicted = Coat / 4
Actual = Trouser / 1
Predicted = Trouser / 1
Actual = Dress / 3
Predicted = Dress / 3
Actual = T-shirt/top / 0
Predicted = T-shirt/top / 0
Actual = Sneaker / 7
Predicted = Sneaker / 7
Actual = Bag / 8
Predicted = Bag / 8
Actual = Ankle boot / 9
Predicted = Ankle boot / 9
Actual = Trouser / 1
Predicted = Trouser / 1
Actual = Bag / 8
Predicted = Bag / 8
Actual = Shirt / 6
Predicted = Shirt / 6
Actual = Ankle boot / 9
Predicted = Ankle boot / 9
Actual = Pullover / 2
Predicted = Pullover / 2
Actual = Ankle boot / 9
Predicted = Ankle boot / 9
Actual = Sneaker / 7
Predicted = Sneaker / 7
Actual = Bag / 8
Predicted = Bag / 8
Actual = Ankle boot / 9
Predicted = Ankle boot / 9
Actual = Sandal / 5
Predicted = Sandal / 5
Actual = Coat / 4
Predicted = Dress / 3
Actual = T-shirt/top / 0
Predicted = T-shirt/top / 0
Actual = Pullover / 2
Predicted = Shirt / 6
Actual = T-shirt/top / 0
Predicted = T-shirt/top / 0
Actual = Dress / 3
Predicted = Dress / 3
Actual = T-shirt/top / 0
Predicted = T-shirt/top / 0
Actual = Bag / 8
Predicted = Bag / 8
Actual = Dress / 3
Predicted = Dress / 3
Actual = Bag / 8
Predicted = Bag / 8
Actual = T-shirt/top / 0
Predicted = T-shirt/top / 0
Actual = Coat / 4
Predicted = Coat / 4
Actual = Trouser / 1
Predicted = Trouser / 1
Actual = Dress / 3
Predicted = Dress / 3
Actual = Pullover / 2
Predicted = Pullover / 2
Actual = T-shirt/top / 0
Predicted = T-shirt/top / 0
Actual = Trouser / 1
Predicted = Trouser / 1

```
from sklearn.metrics import confusion_matrix
plt.figure(figsize=(16,9))
y_pred_labels = [ np.argmax(label) for label in y_pred ]
cm = confusion_matrix(y_test, y_pred_labels)
```

```
<Figure size 1600x900 with 0 Axes>
```

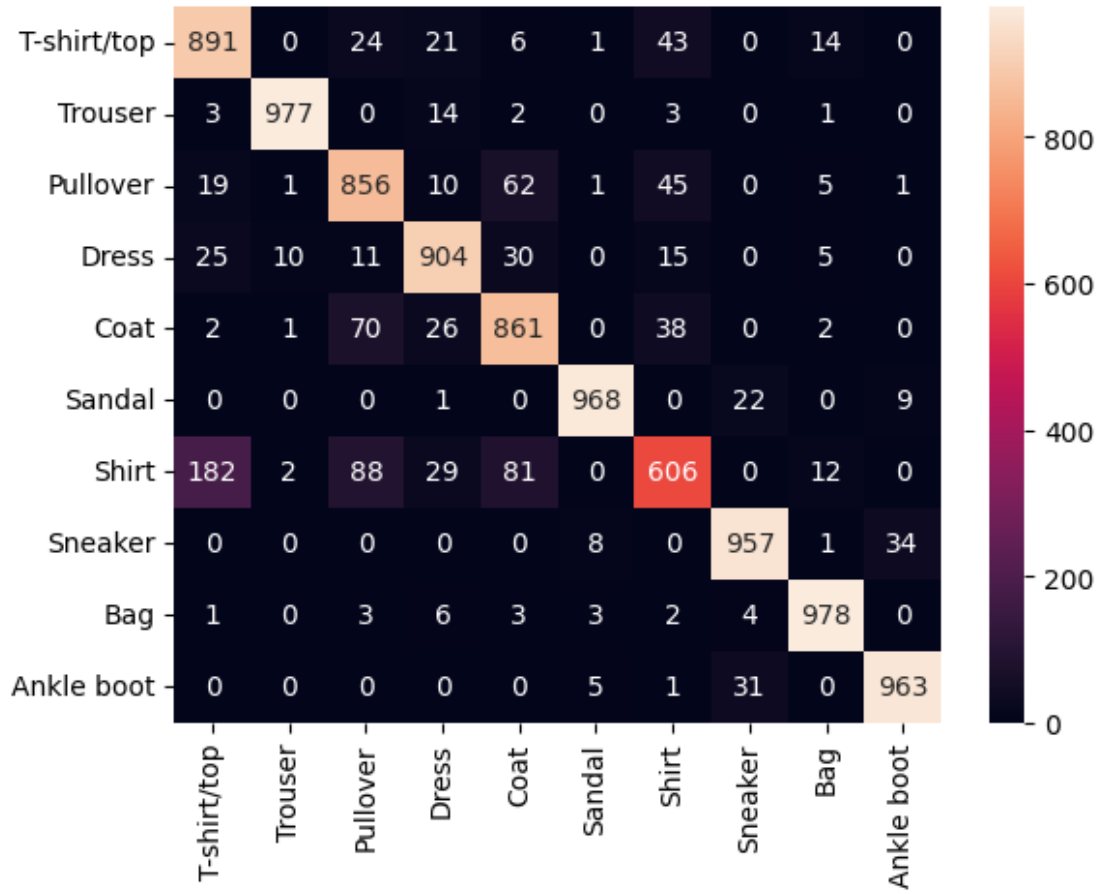
```
sns.heatmap(cm, annot=True, fmt='d',xticklabels=class_labels,
yticklabels=class_labels)
```

```
from sklearn.metrics import classification_report
cr= classification_report(y_test, y_pred_labels, target_names=class_labels)
print(cr)
```

```
              precision    recall  f1-score   support

 T-shirt/top       0.79      0.89      0.84      1000
     Trouser       0.99      0.98      0.98      1000
    Pullover       0.81      0.86      0.83      1000
       Dress       0.89      0.90      0.90      1000
        Coat       0.82      0.86      0.84      1000
      Sandal       0.98      0.97      0.97      1000
       Shirt       0.80      0.61      0.69      1000
     Sneaker       0.94      0.96      0.95      1000
         Bag       0.96      0.98      0.97      1000
  Ankle boot       0.96      0.96      0.96      1000

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.89     10000
weighted avg       0.90      0.90      0.89     10000
```




```
model.save('fashion_mnist_cnn_model.h5')
```

```
cnn_model2 = keras.models.Sequential([
                        keras.layers.Conv2D(filters=32, kernel_size=3,
strides=(1,1), padding='valid',activation= 'relu', input_shape=[28,28,1]),
                        keras.layers.MaxPooling2D(pool_size=(2,2)),
                        keras.layers.Conv2D(filters=64, kernel_size=3,
strides=(2,2), padding='same', activation='relu'),
                        keras.layers.MaxPooling2D(pool_size=(2,2)),
                        keras.layers.Flatten(),
                        keras.layers.Dense(units=128, activation='relu'),
                        keras.layers.Dropout(0.25),
                        keras.layers.Dense(units=256, activation='relu'),
                        keras.layers.Dropout(0.25),
                        keras.layers.Dense(units=128, activation='relu'),
                        keras.layers.Dense(units=10, activation='softmax')
                        ])
```

```
# complie the model
cnn_model2.compile(optimizer='adam', loss= 'sparse_categorical_crossentropy',
metrics=['accuracy'])
```

```
#Train the Model
```

```
cnn_model2.fit(X_train, y_train, epochs=20, batch_size=512, verbose=1,
validation_data=(X_Validation, y_Validation))

cnn_model2.save('fashion_mnist_cnn_model2.h5')

"""######## very complex model"""

#Building CNN model
cnn_model3 = keras.models.Sequential([
                        keras.layers.Conv2D(filters=64, kernel_size=3,
strides=(1,1), padding='valid',activation= 'relu', input_shape=[28,28,1]),
                        keras.layers.MaxPooling2D(pool_size=(2,2)),
                        keras.layers.Conv2D(filters=128, kernel_size=3,
strides=(2,2), padding='same', activation='relu'),
                        keras.layers.MaxPooling2D(pool_size=(2,2)),
                        keras.layers.Conv2D(filters=64, kernel_size=3,
strides=(2,2), padding='same', activation='relu'),
                        keras.layers.MaxPooling2D(pool_size=(2,2)),
                        keras.layers.Flatten(),
                        keras.layers.Dense(units=128, activation='relu'),
                        keras.layers.Dropout(0.25),
                        keras.layers.Dense(units=256, activation='relu'),
                        keras.layers.Dropout(0.5),
                        keras.layers.Dense(units=256, activation='relu'),
                        keras.layers.Dropout(0.25),
                        keras.layers.Dense(units=128, activation='relu'),
                        keras.layers.Dropout(0.10),
                        keras.layers.Dense(units=10, activation='softmax')
                        ])

# complie the model
cnn_model3.compile(optimizer='adam', loss= 'sparse_categorical_crossentropy',
metrics=['accuracy'])

#Train the Model
cnn_model3.fit(X_train, y_train, epochs=50, batch_size=512, verbose=1,
validation_data=(X_Validation, y_Validation))

cnn_model3.save('fashion_mnist_cnn_model3.h5')

cnn_model3.evaluate(X_test, y_test)


Epoch 1/20
94/94 [==============================] - 30s 304ms/step - loss: 1.0358 -
accuracy: 0.6133 - val_loss: 0.5824 - val_accuracy: 0.7786
Epoch 2/20
94/94 [==============================] - 33s 357ms/step - loss: 0.5480 -
accuracy: 0.7928 - val_loss: 0.4628 - val_accuracy: 0.8281
```

```
Epoch 3/20
94/94 [==============================] - 29s 313ms/step - loss: 0.4558 -
accuracy: 0.8322 - val_loss: 0.4100 - val_accuracy: 0.8474
Epoch 4/20
94/94 [==============================] - 28s 300ms/step - loss: 0.4060 -
accuracy: 0.8520 - val_loss: 0.3646 - val_accuracy: 0.8632
Epoch 5/20
94/94 [==============================] - 29s 313ms/step - loss: 0.3686 -
accuracy: 0.8647 - val_loss: 0.3426 - val_accuracy: 0.8742
Epoch 6/20
94/94 [==============================] - 30s 317ms/step - loss: 0.3381 -
accuracy: 0.8754 - val_loss: 0.3272 - val_accuracy: 0.8798
Epoch 7/20
94/94 [==============================] - 29s 314ms/step - loss: 0.3188 -
accuracy: 0.8839 - val_loss: 0.3105 - val_accuracy: 0.8832
Epoch 8/20
94/94 [==============================] - 28s 301ms/step - loss: 0.3005 -
accuracy: 0.8896 - val_loss: 0.3106 - val_accuracy: 0.8851
Epoch 9/20
94/94 [==============================] - 29s 307ms/step - loss: 0.2886 -
accuracy: 0.8936 - val_loss: 0.2976 - val_accuracy: 0.8899
Epoch 10/20
94/94 [==============================] - 30s 317ms/step - loss: 0.2773 -
accuracy: 0.8982 - val_loss: 0.2948 - val_accuracy: 0.8904
Epoch 11/20
94/94 [==============================] - 28s 302ms/step - loss: 0.2647 -
accuracy: 0.9020 - val_loss: 0.2856 - val_accuracy: 0.8953
Epoch 12/20
94/94 [==============================] - 28s 300ms/step - loss: 0.2525 -
accuracy: 0.9068 - val_loss: 0.2693 - val_accuracy: 0.9032
Epoch 13/20
94/94 [==============================] - 28s 300ms/step - loss: 0.2445 -
accuracy: 0.9104 - val_loss: 0.2671 - val_accuracy: 0.9052
Epoch 14/20
94/94 [==============================] - 29s 314ms/step - loss: 0.2382 -
accuracy: 0.9116 - val_loss: 0.2912 - val_accuracy: 0.8917
Epoch 15/20
94/94 [==============================] - 28s 300ms/step - loss: 0.2326 -
accuracy: 0.9130 - val_loss: 0.2753 - val_accuracy: 0.9013
Epoch 16/20
94/94 [==============================] - 29s 311ms/step - loss: 0.2249 -
accuracy: 0.9167 - val_loss: 0.2610 - val_accuracy: 0.9058
Epoch 17/20
94/94 [==============================] - 29s 311ms/step - loss: 0.2169 -
accuracy: 0.9193 - val_loss: 0.2921 - val_accuracy: 0.8962
Epoch 18/20
94/94 [==============================] - 30s 315ms/step - loss: 0.2109 -
accuracy: 0.9222 - val_loss: 0.2722 - val_accuracy: 0.9042
Epoch 19/20
94/94 [==============================] - 28s 300ms/step - loss: 0.2049 -
```

```
accuracy: 0.9229 - val_loss: 0.2574 - val_accuracy: 0.9072
Epoch 20/20
94/94 [==============================] - 29s 312ms/step - loss: 0.1988 -
accuracy: 0.9253 - val_loss: 0.2666 - val_accuracy: 0.9091
Epoch 1/50
94/94 [==============================] - 68s 706ms/step - loss: 1.2005 -
accuracy: 0.5231 - val_loss: 0.6106 - val_accuracy: 0.7594
Epoch 2/50
94/94 [==============================] - 66s 699ms/step - loss: 0.5867 -
accuracy: 0.7786 - val_loss: 0.4819 - val_accuracy: 0.8128
Epoch 3/50
94/94 [==============================] - 67s 716ms/step - loss: 0.4862 -
accuracy: 0.8182 - val_loss: 0.4240 - val_accuracy: 0.8377
Epoch 4/50
94/94 [==============================] - 67s 719ms/step - loss: 0.4237 -
accuracy: 0.8453 - val_loss: 0.3753 - val_accuracy: 0.8582
Epoch 5/50
27/94 [=======>......................] - ETA: 45s - loss: 0.3767 - accuracy:
0.8662
```

CONCLUSION

With our final CNN model, we could achieve a training accuracy of 94% and **test accuracy of 93%** confirming that model is fine **with no overfitting**.

If you remember, with **Machine Learning** model (XGBoost) I had achieved a test accuracy of **84.72 %**, and with **Deep Learning** model (CNN) here I could achieve a test accuracy of **93 %**. Thus, we got around *8% improvement in accuracy* by using Deep Learning.

Though, in this case, we got a good improvement in accuracy score (8%), still there may be a chance to improve performance further, by say, increasing the number of convolutional layers (and neurons/filters) or trying out different combinations of different layers.