ABSTRACT

- 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 names of the class given apparel items belongs to.This is a multiclass classification problem in which there are 10 apparel classes the items will be classified.
- The fashion training set consist of 70,000 images divided into 60,000 training and 10,000 testing samples.Dataset sample consist 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

OBJECTIVES

- This work is part of my experiments with Fashion-MNIST dataset using Convolutional Neural Network (CNN) which I have implemented using TensorFlow Keras APIs. The objective is to identify (predict) different fashion products from the given images using a CNN model. For regularization, I have used ‘dropout’ technique for this problem

INTRODUCTION

- The Fashion-MNIST clothing classification problem is a new standard dataset used in computer vision and deep learning.
- Although the dataset is relatively simple, it can be used as the basis for learning and practicing how to develop, evaluate, and use deep convolutional neural networks for image classification from scratch. This includes how to develop a robust test harness for estimating the performance of the model, how to explore improvements to the model, and how to save the model and later load it to make predictions on new data.
- Fashion-MNIST is a dataset of Zalando’s article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. It is split in 10,000 as test and 50,000 as train datasets.
- so, the end goal is to train and test the model usimg Convolution neural network.

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.

Fashion MNIST Data classification Project

STEP:1-IMPORT LIBRARIES

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

STEP:2- LOAD DATA

```
(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-datas
    29515/29515 [==============================] - 0s 0us/step
    Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
    26421880/26421880 [==============================] - 0s 0us/step
    Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
    5148/5148 [==============================] - 0s 0us/step
    Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
    4422102/4422102 [==============================] - 0s 0us/step
```

```
#Print the shape of data
```

```
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,   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,
```

```
y_train[0]
```

```
    9
```

```
class_labels = [    "T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal"
```

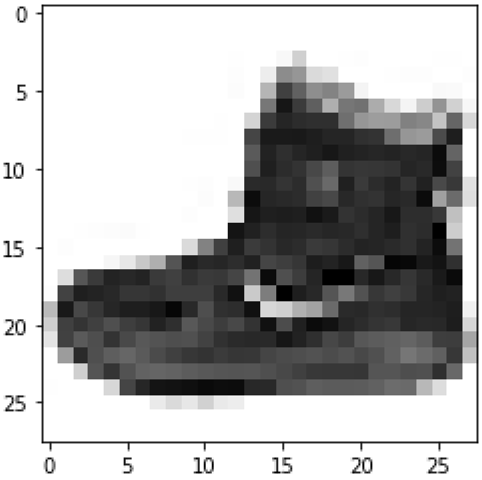
```
class_labels
```

```
    ['T-shirt/top',
     'Trouser',
     'Pullover',
     'Dress',
     'Coat',
     'Sandal',
     'Shirt',
     'Sneaker',
     'Bag',
     'Ankle boot']
```

```
# show image
```

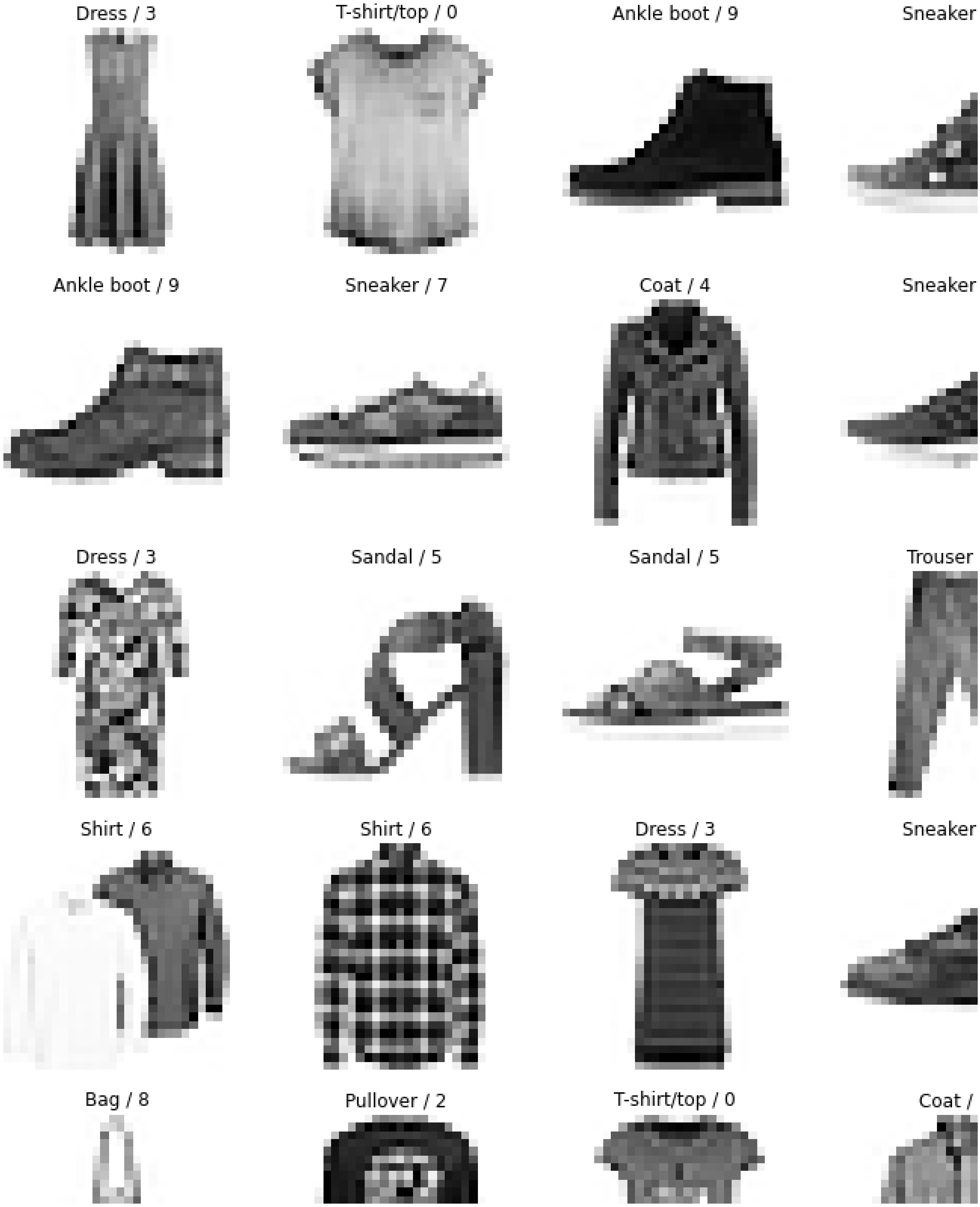
```
plt.imshow(X_train[0],cmap='Greys')
```

```
    <matplotlib.image.AxesImage at 0x7fc8407ae890>
```



```
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_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)


# feature scaling


X_train = X_train/255
X_test= X_test/255


# Split dataset


from sklearn.model_selection import  train_test_split
X_train,X_Validation,y_train,y_Validation=train_test_split(X_train,y_train,test_siz


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

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

Step:3-BUIDING the CNN MODEL**

```
model=keras.models.Sequential([
                         keras.layers.Conv2D(filters=32,kernel_size=3,strides=(1,1)
                         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=['acc
model.fit(X_train,y_train,epochs=10,batch_size=512,verbose=1,validation_data=(X_Val
```

```
Epoch 1/10
94/94 [==============================] - 25s 259ms/step - loss: 0.6195 - accur
Epoch 2/10
94/94 [==============================] - 23s 242ms/step - loss: 0.3722 - accur
Epoch 3/10
94/94 [==============================] - 22s 234ms/step - loss: 0.3260 - accur
Epoch 4/10
94/94 [==============================] - 30s 318ms/step - loss: 0.2970 - accur
Epoch 5/10
94/94 [==============================] - 28s 300ms/step - loss: 0.2775 - accur
Epoch 6/10
94/94 [==============================] - 30s 320ms/step - loss: 0.2633 - accur
Epoch 7/10
94/94 [==============================] - 22s 238ms/step - loss: 0.2448 - accur
Epoch 8/10
94/94 [==============================] - 26s 272ms/step - loss: 0.2303 - accur
Epoch 9/10
94/94 [==============================] - 23s 248ms/step - loss: 0.2207 - accur
Epoch 10/10
94/94 [==============================] - 25s 268ms/step - loss: 0.2122 - accur
<keras.callbacks.History at 0x7fc83d17a4d0>
```

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

```
313/313 [==============================] - 3s 8ms/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.  , 0.98, 0.  ],
       [0.  , 1.  , 0.  , ..., 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.01, ..., 0.1 , 0.04, 0.  ]], dtype=float32)
```

```
y_test
```

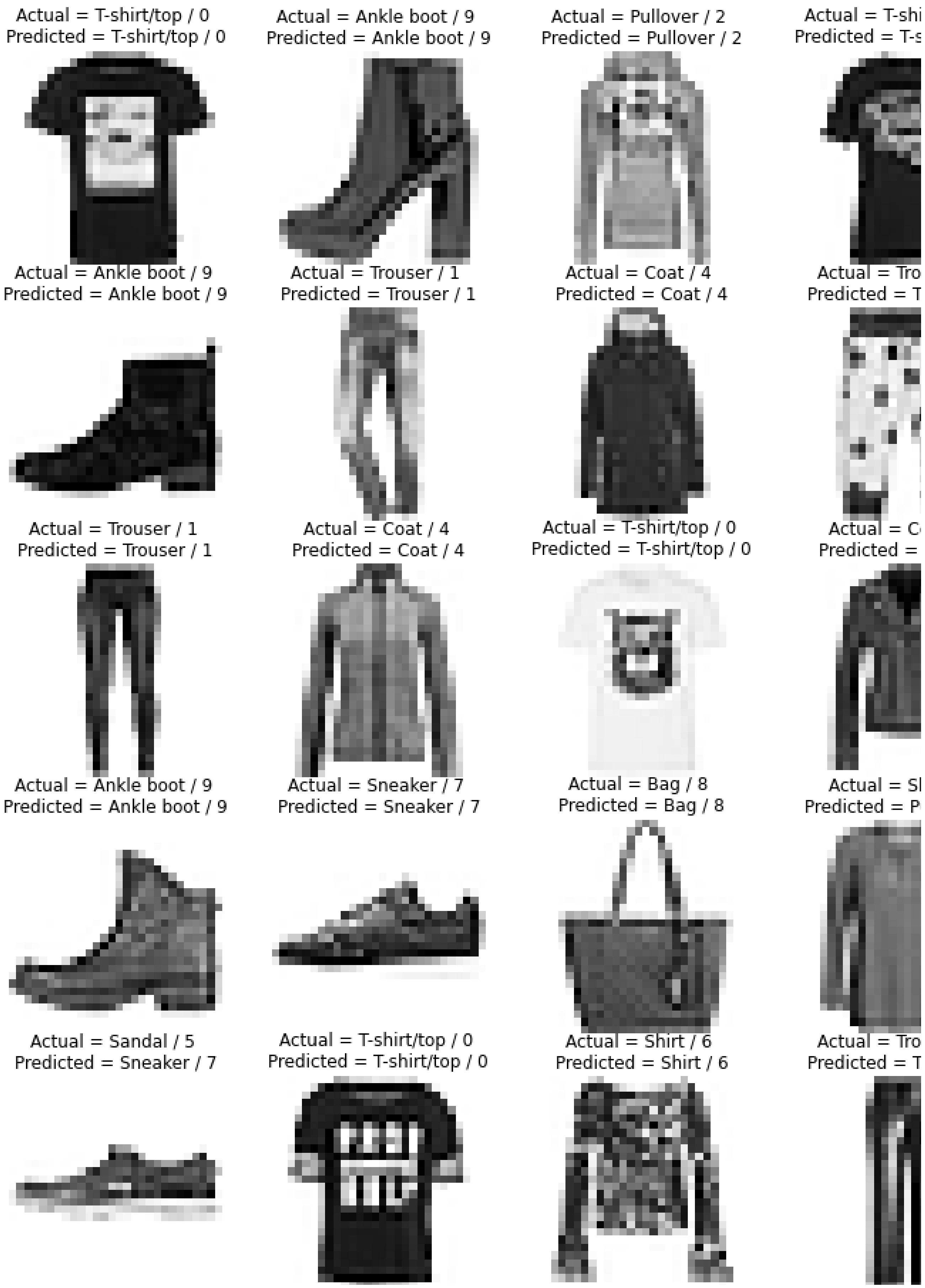
```
array([9, 2, 1, ..., 8, 1, 5], dtype=uint8)
```

```
model.evaluate(X_test, y_test)
```

```
313/313 [==============================] - 3s 8ms/step - loss: 0.2772 - accura
[0.27724429965019226, 0.9006999731063843]
```

```
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]]
  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]]
    plt.axis('off')
```

Actual = Sneaker / 7
Predicted = Sneaker / 7

Actual = Dress / 3
Predicted = Dress / 3

Actual = Sandal / 5
Predicted = Sandal / 5

Actual = Bag / 8
Predicted = Bag / 8

Actu
Predic

```
"""## Confusion Matrix"""
```

```
    '## Confusion Matrix'
```

```
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 1152x648 with 0 Axes>
```

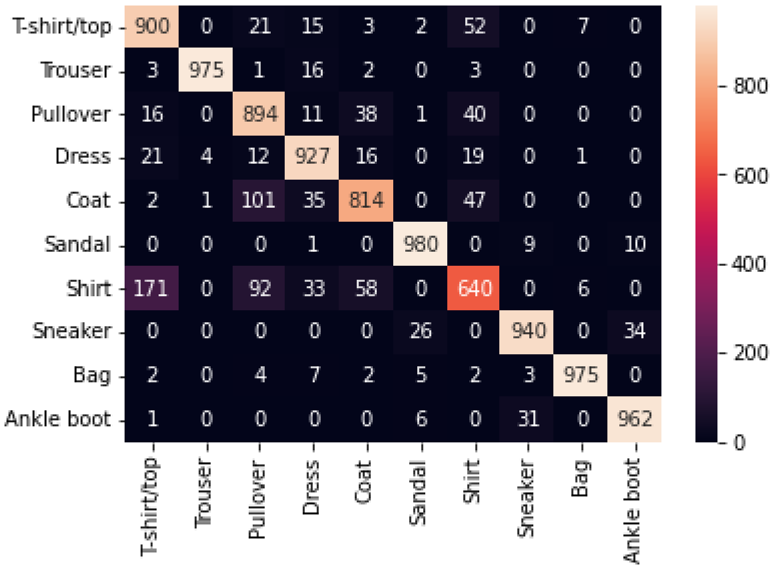
```
sns.heatmap(cm, annot=True, fmt='d',xticklabels=class_labels, yticklabels=class_lab

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       0.81      0.90      0.85      1000
     Trouser       0.99      0.97      0.98      1000
    Pullover       0.79      0.89      0.84      1000
       Dress       0.89      0.93      0.91      1000
        Coat       0.87      0.81      0.84      1000
      Sandal       0.96      0.98      0.97      1000
       Shirt       0.80      0.64      0.71      1000
     Sneaker       0.96      0.94      0.95      1000
         Bag       0.99      0.97      0.98      1000
  Ankle boot       0.96      0.96      0.96      1000

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



Predicted = Coat / 4 Predicted = Bag / 8 Predicted = Sandal / 5 Predicted = Trouser / 1 Pre

```
"""# Save Model"""
```

```
    '# Save Model'
```

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

Build 2 complex CNN

```
#Building CNN model
cnn_model2 = keras.models.Sequential([
                         keras.layers.Conv2D(filters=32, kernel_size=3, strides=(1,
                         keras.layers.MaxPooling2D(pool_size=(2,2)),
                         keras.layers.Conv2D(filters=64, kernel_size=3, strides=(2,
                         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', metri

#Train the Model
cnn_model2.fit(X_train, y_train, epochs=20, batch_size=512, verbose=1, validation_d

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,
                         keras.layers.MaxPooling2D(pool_size=(2,2)),
                         keras.layers.Conv2D(filters=128, kernel_size=3, strides=(2
                         keras.layers.MaxPooling2D(pool_size=(2,2)),
                         keras.layers.Conv2D(filters=64, kernel_size=3, strides=(2,
                         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', metri

#Train the Model
cnn_model3.fit(X_train, y_train, epochs=50, batch_size=512, verbose=1, validation_d

cnn_model3.save('fashion_mnist_cnn_model3.h5')

cnn_model3.evaluate(X_test, y_test)

    Epoch 1/20
    94/94 [==============================] - 27s 282ms/step - loss: 1.0217 - accur
    Epoch 2/20
    94/94 [==============================] - 26s 280ms/step - loss: 0.5494 - accur
    Epoch 3/20
    94/94 [==============================] - 26s 280ms/step - loss: 0.4628 - accur
    Epoch 4/20
    94/94 [==============================] - 27s 285ms/step - loss: 0.4130 - accur
    Epoch 5/20
    94/94 [==============================] - 26s 282ms/step - loss: 0.3728 - accur
    Epoch 6/20
    94/94 [==============================] - 29s 310ms/step - loss: 0.3494 - accur
    Epoch 7/20
    94/94 [==============================] - 27s 282ms/step - loss: 0.3285 - accur
    Epoch 8/20
    94/94 [==============================] - 28s 294ms/step - loss: 0.3093 - accur
    Epoch 9/20
    94/94 [==============================] - 27s 282ms/step - loss: 0.2978 - accur
    Epoch 10/20
    94/94 [==============================] - 26s 282ms/step - loss: 0.2813 - accur
    Epoch 11/20
    94/94 [==============================] - 26s 282ms/step - loss: 0.2708 - accur
    Epoch 12/20
    94/94 [==============================] - 26s 281ms/step - loss: 0.2627 - accur
    Epoch 13/20
    94/94 [==============================] - 26s 281ms/step - loss: 0.2527 - accur
    Epoch 14/20
    94/94 [==============================] - 26s 281ms/step - loss: 0.2442 - accur
    Epoch 15/20
    94/94 [==============================] - 26s 281ms/step - loss: 0.2371 - accur
    Epoch 16/20
    94/94 [==============================] - 26s 280ms/step - loss: 0.2281 - accur
    Epoch 17/20
    94/94 [==============================] - 26s 280ms/step - loss: 0.2236 - accur
    Epoch 18/20
    94/94 [==============================] - 26s 281ms/step - loss: 0.2141 - accur
    Epoch 19/20
    94/94 [==============================] - 29s 308ms/step - loss: 0.2082 - accur
    Epoch 20/20
    94/94 [==============================] - 26s 280ms/step - loss: 0.2061 - accur
    Epoch 1/50
    94/94 [==============================] - 60s 635ms/step - loss: 1.2046 - accur
    Epoch 2/50
    94/94 [==============================] - 57s 607ms/step - loss: 0.6001 - accur
    Epoch 3/50
    94/94 [==============================] - 57s 608ms/step - loss: 0.4922 - accur
```

```
Epoch 4/50
94/94 [==============================] - 57s 612ms/step - loss: 0.4131 - accur
Epoch 5/50
94/94 [==============================] - 60s 640ms/step - loss: 0.3686 - accur
Epoch 6/50
94/94 [==============================] - 64s 678ms/step - loss: 0.3370 - accur
Epoch 7/50
94/94 [==============================] - 60s 643ms/step - loss: 0.3077 - accur
Epoch 8/50
94/94 [==============================] - 58s 616ms/step - loss: 0.2929 - accur
Epoch 9/50
94/94 [==============================] - 59s 630ms/step - loss: 0.2778 - accur
```

CONCLUSION

With a complex sequential model with multiple convolution layers and 50 epochs for the training, we obtained an accuracy ~0.91 for test prediction. After investigating the validation accuracy and loss, we understood that the model is overfitting. We retrained the model with Dropout layers to the model to reduce overfitting. We confirmed the model improvement and with the same number of epochs for the training we obtained with the new model an accuracy of ~0.93 for test prediction