In [1]:

```
##Image recoginization use CNN :
#dataset : inbuilt dataset : fashion_mnist
#framework :  tensorflow and keras

#!pip install tensorflow
```

In [2]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import tensorflow
```

In [3]:

```
#inbuilt dataset : fashion_mnist which define in tensorflow.k
#library

(X_train,Y_train),(X_test,Y_test)=tensorflow.keras.datasets.f
```

In [4]:

```
X_train.shape,Y_train.shape #60000 no. of images and 28*28 pi
```

Out[4]:

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

In [5]:

```
X_train
```

Out[5]:

```
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],
        [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],
         [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=uint
8)
```

In [6]:

```
X_test.shape,Y_test.shape
```

Out[6]:

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

In [7]:

```
#access 1st row means 1st image
X_train[0]
```

Out[7]:

```
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,  2
3,   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, 16
1, 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, 12
7, 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, 21
5, 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, 22
4, 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, 21
0, 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, 20
8, 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, 21
5, 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, 21
8, 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, 21
6, 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, 22
4, 223, 219, 215, 224, 244,
        159,   0],
       [  0,   0,   0,   0,  18,  44,  82, 10
7, 189, 228, 220, 222, 217,
        226, 200, 205, 211, 230, 224, 234, 17
6, 188, 250, 248, 233, 238,
        215,   0],
       [  0,  57, 187, 208, 224, 221, 224, 20
8, 204, 214, 208, 209, 200,
        159, 245, 193, 206, 223, 255, 255, 22
1, 234, 221, 211, 220, 232,
        246,   0],
       [  3, 202, 228, 224, 221, 211, 211, 21
4, 205, 205, 205, 220, 240,
         80, 150, 255, 229, 221, 188, 154, 19
1, 210, 204, 209, 222, 228,
        225,   0],
       [ 98, 233, 198, 210, 222, 229, 229, 23
4, 249, 220, 194, 215, 217,
        241,  65,  73, 106, 117, 168, 219, 22
1, 215, 217, 223, 223, 224,
        229,  29],
       [ 75, 204, 212, 204, 193, 205, 211, 22
5, 216, 185, 197, 206, 198,
        213, 240, 195, 227, 245, 239, 223, 21
8, 212, 209, 222, 220, 221,
        230,  67],
       [ 48, 203, 183, 194, 213, 197, 185, 19
0, 194, 192, 202, 214, 219,
        221, 220, 236, 225, 216, 199, 206, 18
6, 181, 177, 172, 181, 205,
        206, 115],
       [  0, 122, 219, 193, 179, 171, 183, 19
6, 204, 210, 213, 207, 211,
        210, 200, 196, 194, 191, 195, 191, 19
8, 192, 176, 156, 167, 177,
        210,  92],
       [  0,   0,  74, 189, 212, 191, 175, 17
2, 175, 181, 185, 188, 189,
        188, 193, 198, 204, 209, 210, 210, 21
1, 188, 188, 194, 192, 216,
```

```
        170,   0],
       [  2,   0,   0,   0,  66, 200, 222, 23
7, 239, 242, 246, 243, 244,
        221, 220, 193, 191, 179, 182, 182, 18
1, 176, 166, 168,  99,  58,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,  4
0,  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)
```

In [8]:

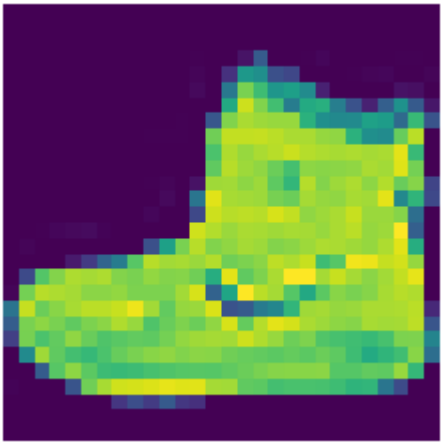
```
#class label of 1st image
Y_train[0]
```

Out[8]:

```
9
```

In [9]:

```
#to show image
plt.imshow(X_train[0])  #show 1st image dataset
plt.axis('off')
plt.show()
```



In [10]:

```
#to show image
plt.imshow(X_train[1])  #show 2nd image dataset
plt.axis('off')
plt.show()
```

In [11]:

```
#Loads the Fashion-MNIST dataset.
'''
This is a dataset of 60,000 28x28 grayscale images of 10 fash
along with a test set of 10,000 images. This dataset can be u
drop-in replacement for MNIST. The class labels are:

Label   Description
0   'T-shirt/top'
1   Trouser
2   Pullover
3   Dress
4   Coat
5   Sandal
6   Shirt
7   Sneaker
8   Bag
9   Ankle boot'''
```

Out[11]:

```
"\nThis is a dataset of 60,000 28x28 grayscale
images of 10 fashion categories,\nalong with a
test set of 10,000 images. This dataset can be
used as a\ndrop-in replacement for MNIST. The
class labels are:\n\nLabel\tDescription\n0\t'T
-shirt/top'\n1\tTrouser\n2\tPullover\n3\tDress
\n4\tCoat\n5\tSandal\n6\tShirt\n7\tSneaker\n8
\tBag\n9\tAnkle boot"
```

In [12]:

```
#class_labels user defined list object
class_labels=['T-shirt/top','Trouser','Pullover','Dress','Coa
'Shirt','Sneaker','Bag','Ankle boot']
print(class_labels)
```

```
['T-shirt/top', 'Trouser', 'Pullover', 'Dres
s', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Ba
g', 'Ankle boot']
```

In [13]:

```
#To show 25 images randomly
plt.figure(figsize=(16,16))
j=1
for i in np.random.randint(0,1000,30):
    plt.subplot(6,5,j);j=j+1
    plt.imshow(X_train[i]) #0-255
    plt.axis('off')
    plt.title("Record No. {}-{}-{}".format(i+1,class_labels[Y
```

In [14]:

```
#CNN means Convolutional neural network CNN :
#Note : In CNN , we have to give 4 dimension input data(image

#check dimension of dataset
print("Dimension of Training data : ",X_train.ndim)
#We have 3 dimensional dataset
print("Shape of Training Data : ",X_train.shape)
```

```
Dimension of Training data :  3
Shape of Training Data :  (60000, 28, 28)
```

In [15]:

```
#change the dimesion of training data
#We have to converts 3D dimension dataset into 4D dimension d
#so we use inbuilt method of numpy :  expand_dims()

X_train=np.expand_dims(X_train,-1)#expand_dims(data,axis)

#check dimension of training data after expand dimension
print("Dimension : ",X_train.ndim)
#Check shape of training data after expand dimension
print(X_train.shape)
```

```
Dimension :  4
(60000, 28, 28, 1)
```

In [16]:

```
#change the dimesion of training data
#We have to converts 3D dimension dataset into 4D dimension d
#so we use inbuilt method of numpy :  expand_dims()

X_test=np.expand_dims(X_test,-1)#expand_dims(data,axis)

#check dimension of training data after expand dimension
print("Dimension : ",X_test.ndim)
#Check shape of training data after expand dimension
print(X_test.shape)
```

```
Dimension :  4
(10000, 28, 28, 1)
```

In [17]:

```
#feature scaling : -
#Feature Scaling : -
##Feature Scaling on input data(training data and testing dat
#apply min_max_scaler() means value  between 0 to 1
#here min value=0 and max value=255
X_train=X_train/255
X_test=X_test/255
```

In [18]:

```
#X_train[0] #access 1st image
```

In [19]:

```
#training error>=testing error :  model is perfect means mode
#overfit
#Split Dataset (To split training dataset into (80%  train da
#20% :- validation data for check overfitting model)
#means take 80% data for training and 20% for validation from
#Y_train
#call train_test_split
from sklearn.model_selection import train_test_split
X_train,X_val,Y_train,Y_val=train_test_split(X_train,Y_train,
                                             random_state=1)
```

In [20]:

```
X_train.shape
```

Out[20]:

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

In [21]:

```
X_val.shape
```

Out[21]:

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

In [22]:

```
#Convolutional Neural Network - model Building
model=tensorflow.keras.models.Sequential([
 tensorflow.keras.layers.Conv2D(filters=32,kernel_size=3,stri
                padding='valid',activation='relu',input_shape
    tensorflow.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tensorflow.keras.layers.Flatten(),
    tensorflow.keras.layers.Dense(units=128,activation='relu'
     tensorflow.keras.layers.Dense(units=10,activation='softm
])
```

In [23]:

```
#check summary
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
_________________________________________________________________
```

In [24]:

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

In [25]:

```
#train model
model.fit(X_train,Y_train,epochs=10,batch_size=512,verbose=1,
          validation_data=(X_val,Y_val))
```

```
Epoch 1/10
94/94 [==============================] - 20s 2
00ms/step - loss: 0.6383 - accuracy: 0.7908 -
val_loss: 0.4124 - val_accuracy: 0.8548
Epoch 2/10
94/94 [==============================] - 18s 1
93ms/step - loss: 0.3717 - accuracy: 0.8711 -
val_loss: 0.3451 - val_accuracy: 0.8764
Epoch 3/10
94/94 [==============================] - 17s 1
85ms/step - loss: 0.3245 - accuracy: 0.8877 -
val_loss: 0.3355 - val_accuracy: 0.8826
Epoch 4/10
94/94 [==============================] - 18s 1
91ms/step - loss: 0.2954 - accuracy: 0.8970 -
val_loss: 0.3032 - val_accuracy: 0.8914
Epoch 5/10
94/94 [==============================] - 18s 1
89ms/step - loss: 0.2772 - accuracy: 0.9027 -
val_loss: 0.2885 - val_accuracy: 0.8947
Epoch 6/10
94/94 [==============================] - 17s 1
84ms/step - loss: 0.2614 - accuracy: 0.9082 -
val_loss: 0.2770 - val_accuracy: 0.8982
Epoch 7/10
94/94 [==============================] - 17s 1
85ms/step - loss: 0.2477 - accuracy: 0.9112 -
val_loss: 0.2857 - val_accuracy: 0.8962
Epoch 8/10
94/94 [==============================] - 17s 1
86ms/step - loss: 0.2345 - accuracy: 0.9162 -
val_loss: 0.2631 - val_accuracy: 0.9066
Epoch 9/10
94/94 [==============================] - 18s 1
91ms/step - loss: 0.2237 - accuracy: 0.9197 -
```

```
val_loss: 0.2592 - val_accuracy: 0.9069
Epoch 10/10
94/94 [==============================] - 18s 1
87ms/step - loss: 0.2120 - accuracy: 0.9250 -
val_loss: 0.2535 - val_accuracy: 0.9082
```

Out[25]:

```
<keras.callbacks.History at 0x18df176cdf0>
```

In [26]:

```
#training and testing score
model.evaluate(X_train,Y_train)
model.evaluate(X_val,Y_val)
```

```
1500/1500 [==============================] - 8
s 5ms/step - loss: 0.1987 - accuracy: 0.9293
375/375 [==============================] - 2s
6ms/step - loss: 0.2535 - accuracy: 0.9082
```

Out[26]:

```
[0.25351160764694214, 0.9081666469573975]
```

In [27]:

```
#test the model
Y_pred=model.predict(X_test).round(2)
```

```
313/313 [==============================] - 2s
5ms/step
```

In [28]:

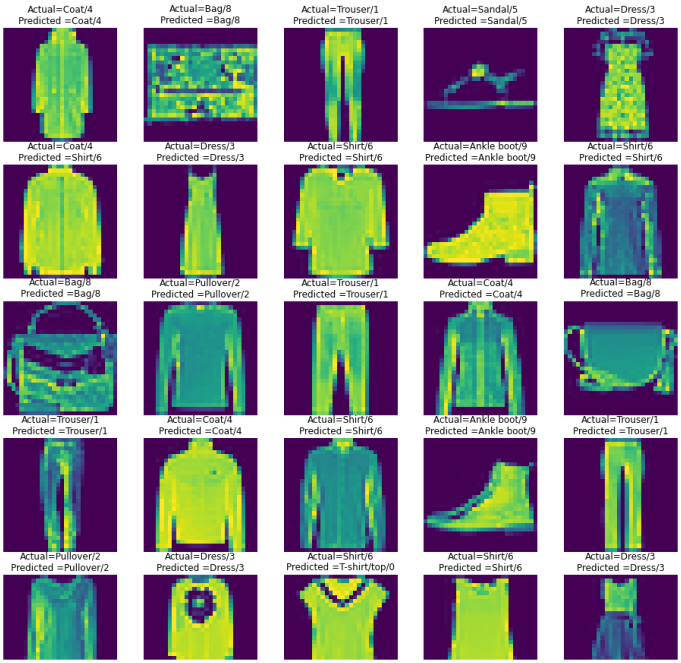
```
Y_pred
```

Out[28]:

```
array([[0.  , 0.  , 0.  , ..., 0.01, 0.  , 0.9
9],
       [0.  , 0.  , 1.  , ..., 0.  , 0.  , 0.
],
       [0.  , 1.  , 0.  , ..., 0.  , 0.  , 0.
],
       ...,
       [0.  , 0.  , 0.  , ..., 0.  , 0.99, 0.
],
       [0.  , 1.  , 0.  , ..., 0.  , 0.  , 0.
],
       [0.  , 0.  , 0.  , ..., 0.22, 0.06, 0.
]], dtype=float32)
```

In [29]:

```
#visualize
#To show 25 images randomly
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_test[i].reshape(28,28)) #0-255
    plt.axis('off')
    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])))
```



In [30]:

```
#generate report
Y_pred1=[np.argmax(i) for i in Y_pred]
```

In [31]:

```
Y_pred1[0:20]
```

Out[31]:

```
[9, 2, 1, 1, 6, 1, 4, 6, 5, 7, 4, 5, 5, 3, 4,
1, 2, 4, 8, 0]
```

In [33]:

```
class_labels
```

Out[33]:

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

In [34]:

```
#classification report
from sklearn.metrics import classification_report,confusion_m
print(classification_report(Y_test,Y_pred1))
```

```
              precision    recall  f1-score
support

           0       0.83      0.87      0.85
1000
           1       0.99      0.97      0.98
1000
           2       0.83      0.85      0.84
1000
           3       0.87      0.94      0.90
1000
           4       0.85      0.87      0.86
1000
           5       0.98      0.97      0.97
1000
           6       0.80      0.66      0.72
1000
           7       0.95      0.96      0.96
1000
           8       0.97      0.98      0.98
1000
           9       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
```

In [36]:

```
cm=confusion_matrix(Y_test,Y_pred1)
import seaborn as sns
plt.figure(figsize=(16,9))
sns.heatmap(cm,annot=True,fmt="d",xticklabels=class_labels,
            yticklabels=class_labels) #fmt means format and d

plt.xlabel("Predicted output")
plt.ylabel("Actual output")
plt.show()
```