DEEP LEARNING PROJECT FASHION MNIST CLASSIFICATION

▻ 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 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 :-

Convolutional neural network, fashion-MNIST, image classification, MNIST.

▻ Introduction :-

Image Classification is one of the most popular tasks in Deep Learning where we are given a particular image and the model has to predict the class of the image. Deep Learning libraries such has TensorFlow and Pytorch and not to forget the very simple Keras API on the top of TensorFlow provide intuitive and easy methods to perform image classification at ease. Detecting Images and classifying them is a prime

application today in all industrial areas whether in Self Driving Vehicles to classify or traffic signals or in Medical Imaging and the list never ends. Isn't it amazing!!▻

We will use the very popular Fashion MNIST dataset to classify clothing, footwear and other related items using Keras API. So, lets dive in.

What actually is Fashion MNIST?

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 28×28 grayscale image, associated with a label from 10 classes. Fashion-MNIST is intended to serve as a direct drop-in replacement of the original MNIST dataset for benchmarking machine learning algorithms.
The original MNIST dataset is set of handwritten digits from 0–9, in grayscale with 28 x 28-pixel sizes.

▻ Methodology :-

1. Firstly we have to import all the necessary modules like numpy, matplotlib, keras, seaborn etc.
2. Now we have to load the dataset.
3. Now we have to set all the class labels.

4. We have 3-dimantional data. We convert it in 4-dimantional data.
5. Now we have to split the training data in training and validation data.
6. Now we have to build the Deep learning model using keras.
7. Now we have to compile the model.
8. Train the model using the training data.
9. We have to predict the outcomes using the test data.
10. Now we have to evaluate the model.
11. Create confusion matrix.
12. Now save the model.

CODE:

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

(x_train, y_train), (x_test, y_test) =
tk.datasets.fashion_mnist.load_data()

x_train.shape, y_train.shape

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

x_test.shape, y_test.shape

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

x_train

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=uint8)
```

```
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,
```

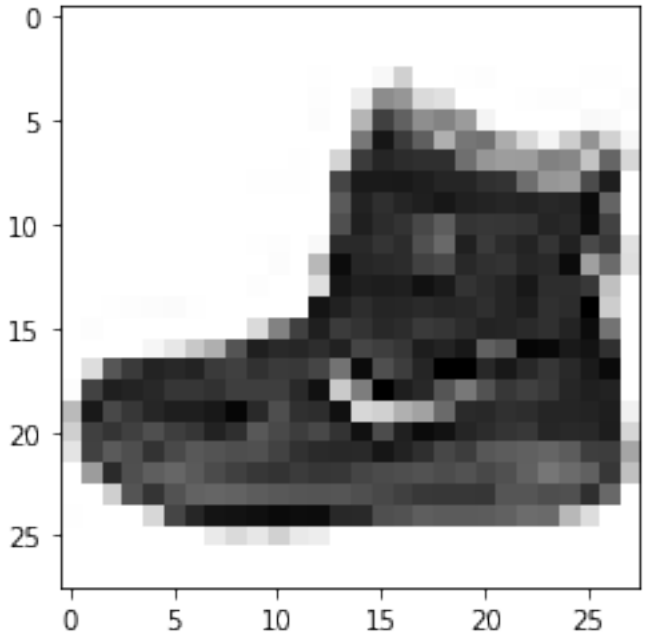
```
          0, 102, 204, 176, 134, 144, 123,  23,   0,   0,   0,   0,
12,
         10,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,
          0, 155, 236, 207, 178, 107, 156, 161, 109,  64,  23,  77,
130,
         72,  15],
       [  0,   0,   0,   0,   0,   0,   0,   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,
        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)
```

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

```
<matplotlib.image.AxesImage at 0x29b23e44df0>
```




```
class_labels = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
"Sandal", "Shirt","Sneaker","Bag","Ankle boot"]

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_test = np.expand_dims(x_test, -1)
```

```
x_train.ndim
```

```
4
```

```
x_train.shape
```

```
(60000, 28, 28, 1)
```

```
x_train = x_train/255
x_test = x_test/255
```

```
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train,
test_size=0.2, random_state=2020)

x_train.shape
```

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

```
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')
                        ])

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=30, batch_size=512, verbose=1,
validation_data=(x_val, y_val))
```

```
Epoch 1/30
94/94 [==============================] - 21s 215ms/step - loss: 0.0550
- accuracy: 0.9829 - val_loss: 0.3154 - val_accuracy: 0.9125
Epoch 2/30
94/94 [==============================] - 18s 195ms/step - loss: 0.0463
- accuracy: 0.9869 - val_loss: 0.3255 - val_accuracy: 0.9123
Epoch 3/30
94/94 [==============================] - 21s 227ms/step - loss: 0.0435
- accuracy: 0.9877 - val_loss: 0.3279 - val_accuracy: 0.9124
Epoch 4/30
94/94 [==============================] - 31s 336ms/step - loss: 0.0449
- accuracy: 0.9859 - val_loss: 0.3444 - val_accuracy: 0.9101
Epoch 5/30
94/94 [==============================] - 34s 365ms/step - loss: 0.0384
- accuracy: 0.9896 - val_loss: 0.3359 - val_accuracy: 0.9125
Epoch 6/30
94/94 [==============================] - 35s 376ms/step - loss: 0.0330
- accuracy: 0.9917 - val_loss: 0.3512 - val_accuracy: 0.9133
Epoch 7/30
94/94 [==============================] - 26s 274ms/step - loss: 0.0336
- accuracy: 0.9912 - val_loss: 0.3516 - val_accuracy: 0.9136
Epoch 8/30
94/94 [==============================] - 23s 243ms/step - loss: 0.0303
- accuracy: 0.9922 - val_loss: 0.3814 - val_accuracy: 0.9062
Epoch 9/30
94/94 [==============================] - 19s 199ms/step - loss: 0.0287
- accuracy: 0.9926 - val_loss: 0.3691 - val_accuracy: 0.9120
Epoch 10/30
94/94 [==============================] - 15s 161ms/step - loss: 0.0278
- accuracy: 0.9929 - val_loss: 0.3810 - val_accuracy: 0.9104
Epoch 11/30
94/94 [==============================] - 14s 153ms/step - loss: 0.0240
- accuracy: 0.9946 - val_loss: 0.3858 - val_accuracy: 0.9115
Epoch 12/30
94/94 [==============================] - 14s 149ms/step - loss: 0.0235
- accuracy: 0.9944 - val_loss: 0.3878 - val_accuracy: 0.9122
Epoch 13/30
94/94 [==============================] - 14s 150ms/step - loss: 0.0194
- accuracy: 0.9960 - val_loss: 0.3966 - val_accuracy: 0.9126
Epoch 14/30
94/94 [==============================] - 14s 151ms/step - loss: 0.0196
- accuracy: 0.9958 - val_loss: 0.3936 - val_accuracy: 0.9134
Epoch 15/30
94/94 [==============================] - 14s 148ms/step - loss: 0.0163
- accuracy: 0.9969 - val_loss: 0.4131 - val_accuracy: 0.9114
```

```
Epoch 16/30
94/94 [==============================] - 14s 151ms/step - loss: 0.0166
- accuracy: 0.9967 - val_loss: 0.4111 - val_accuracy: 0.9099
Epoch 17/30
94/94 [==============================] - 15s 164ms/step - loss: 0.0158
- accuracy: 0.9971 - val_loss: 0.4184 - val_accuracy: 0.9130
Epoch 18/30
94/94 [==============================] - 14s 153ms/step - loss: 0.0125
- accuracy: 0.9982 - val_loss: 0.4183 - val_accuracy: 0.9119
Epoch 19/30
94/94 [==============================] - 14s 151ms/step - loss: 0.0104
- accuracy: 0.9989 - val_loss: 0.4311 - val_accuracy: 0.9119
Epoch 20/30
94/94 [==============================] - 15s 157ms/step - loss: 0.0101
- accuracy: 0.9989 - val_loss: 0.4397 - val_accuracy: 0.9118
Epoch 21/30
94/94 [==============================] - 16s 176ms/step - loss: 0.0099
- accuracy: 0.9989 - val_loss: 0.4429 - val_accuracy: 0.9112
Epoch 22/30
94/94 [==============================] - 21s 224ms/step - loss: 0.0125
- accuracy: 0.9973 - val_loss: 0.4894 - val_accuracy: 0.9053
Epoch 23/30
94/94 [==============================] - 17s 186ms/step - loss: 0.0111
- accuracy: 0.9982 - val_loss: 0.4627 - val_accuracy: 0.9119
Epoch 24/30
94/94 [==============================] - 16s 165ms/step - loss: 0.0082
- accuracy: 0.9993 - val_loss: 0.4551 - val_accuracy: 0.9119
Epoch 25/30
94/94 [==============================] - 16s 171ms/step - loss: 0.0069
- accuracy: 0.9994 - val_loss: 0.4723 - val_accuracy: 0.9096
Epoch 26/30
94/94 [==============================] - 16s 167ms/step - loss: 0.0186
- accuracy: 0.9956 - val_loss: 0.5022 - val_accuracy: 0.9034
Epoch 27/30
94/94 [==============================] - 15s 163ms/step - loss: 0.0175
- accuracy: 0.9959 - val_loss: 0.4911 - val_accuracy: 0.9095
Epoch 28/30
94/94 [==============================] - 15s 163ms/step - loss: 0.0080
- accuracy: 0.9990 - val_loss: 0.4750 - val_accuracy: 0.9127
Epoch 29/30
94/94 [==============================] - 15s 161ms/step - loss: 0.0054
- accuracy: 0.9997 - val_loss: 0.4817 - val_accuracy: 0.9134
Epoch 30/30
94/94 [==============================] - 15s 161ms/step - loss: 0.0039
- accuracy: 0.9999 - val_loss: 0.4864 - val_accuracy: 0.9118

<keras.callbacks.History at 0x29b2b8e7070>

model.predict(np.expand_dims(x_test[0], axis=0)).round(2)

1/1 [==============================] - 0s 277ms/step
```

```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]], dtype=float32)
```

```
np.argmax(model.predict(np.expand_dims(x_test[0], axis=0)).round(2))
```

```
1/1 [==============================] - 0s 70ms/step
```

```
9
```

```
y_test[0]
```

```
9
```

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

```
313/313 [==============================] - 2s 7ms/step
```

```
array([[0., 0., 0., ..., 0., 0., 1.],
       [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., ..., 0., 0., 0.]], dtype=float32)
```

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

```
313/313 [==============================] - 3s 8ms/step - loss: 0.4978
- accuracy: 0.9129
```

```
[0.4978380799293518, 0.9128999710083008]
```

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

```
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>
```

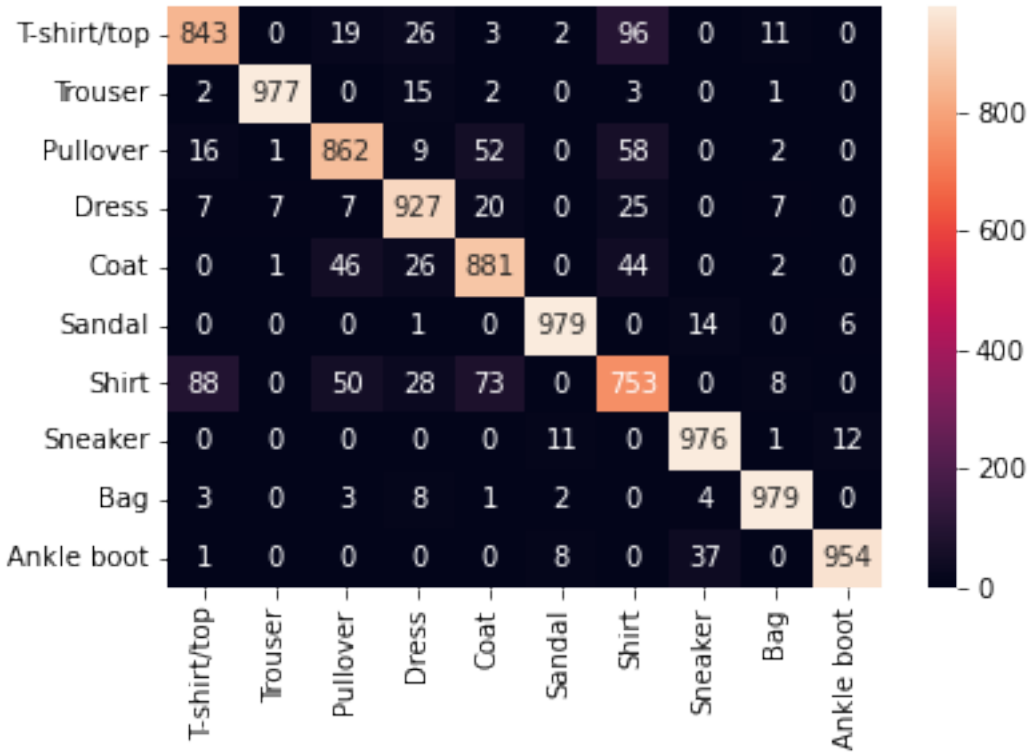
```
sns.heatmap(cm, annot=True, fmt='d',xticklabels=class_labels,
yticklabels=class_labels)
```

```
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       0.88      0.84      0.86      1000
     Trouser       0.99      0.98      0.98      1000
    Pullover       0.87      0.86      0.87      1000
       Dress       0.89      0.93      0.91      1000
        Coat       0.85      0.88      0.87      1000
      Sandal       0.98      0.98      0.98      1000
       Shirt       0.77      0.75      0.76      1000
     Sneaker       0.95      0.98      0.96      1000
         Bag       0.97      0.98      0.97      1000
  Ankle boot       0.98      0.95      0.97      1000

    accuracy                           0.91     10000
   macro avg       0.91      0.91      0.91     10000
weighted avg       0.91      0.91      0.91     10000
```




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

▻ Conclusion :-

We have successfully created and tested DEEP LEARNING MODEL FOR FASHION MNIST CLASSIFICATION WITH 91.29 %.