METHODOLOGY

Using CNN we can classify or identify an image. One image can be predicted by using previous image. If the pixel values are same as compared to previous one then the output is correct

We take small patches of the pixels to compare with present input. By doing this, the Convolutional Neural Network gets a lot better at seeing similarity than directly trying to match the entire image.

There are four layers in CNN to predict image

1. Convolution
2. Relu layer
3. Pooling layer
4. Fully Connectedness

Using epochs we can iterate the models. It shows accuracy of the prediction, it verifies the model for the number of times.

At last will find the confusion matrix and classification report

Using confusion matrix we can say that how many images are perfectly predicted and classified

For evaluating remaining images use cnn2 and cnn3 that repeat the verification using epochs

Following steps involves:-

Step1:->

Import Libraries

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

Step2:

Load Data

```
(X_train, y_train), (X_test, y_test)=tf.keras.datasets.fashion_mnist.load_dat
a()
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-
labels-idx1-ubyte.gz

32768/29515 [=================================] - 0s 0us/step

40960/29515 [=========================================] - 0s 0us/step

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-
images-idx3-ubyte.gz

26427392/26421880 [==============================] - 0s 0us/step

26435584/26421880 [==============================] - 0s 0us/step

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-
labels-idx1-ubyte.gz

16384/5148
[========================================================================================
=====] - 0s 0us/step

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-
images-idx3-ubyte.gz

4423680/4422102 [==============================] - 0s 0us/step

4431872/4422102 [==============================] - 0s 0us/step
X_train.shape,y_train.shape, "***************" , X_test.shape,y_test.shape
```

Step3:

Prepare Pixel Data:

We know that the pixel values for each image in the dataset are unsigned integers in the range between black and white, or 0 and 255.

```
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, 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)
```

```
X_train = X_train/255

X_test= X_test/255
```

But after dividing with 255 it now becomes with in value 1

Step4:
Define Model:

Next, we need to define a baseline convolutional neural network model for the problem.

The model has two main aspects: the feature extraction front end comprised of convolutional and pooling layers, and the classifier backend that will make a prediction.

For the convolutional front-end, we can start with a single convolutional layer with a small pool size (2,2) and a modest number of filters (32) followed by a max pooling layer. The filter maps can then be flattened to provide features to the classifier.

Given that the problem is a multi-class classification, we know that we will require an output layer with 10 nodes in order to predict the probability distribution of an image belonging to each of the 10 classes. This will also require the use of a softmax activation function. Between the feature extractor and the output layer, we can add a dense layer to interpret the features, in this case with 100 nodes.

All layers will use the ReLU activation function and the He weight initialization scheme, both best practices.

```
model=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.Flatten(),
                         keras.layers.Dense(units=128,activation='relu'),
                         keras.layers.Dense(units=10,activation='softmax')
```

```
])
```

Step5:

TRAIN Model:

In this section the model goes to evaluate 80% is for training the data and rest 20% is for testing

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

Run the model for the few times using epochs function

```
model.fit(X_train,y_train,epochs=50,batch_size=512,verbose=1,validation_data=(X_Validation,y_Validation))
```

```
Epoch 1/50
94/94 [==============================] - 26s 262ms/step - loss: 0.6515 - accuracy: 0.7768 - val_loss: 0.4486 - val_accuracy: 0.8468
Epoch 2/50
94/94 [==============================] - 20s 209ms/step - loss: 0.4059 - accuracy: 0.8586 - val_loss: 0.3940 - val_accuracy: 0.8634
Epoch 3/50
94/94 [==============================] - 19s 207ms/step - loss: 0.3543 - accuracy: 0.8760 - val_loss: 0.3585 - val_accuracy: 0.8748
Epoch 4/50
94/94 [==============================] - 19s 204ms/step - loss: 0.3244 - accuracy: 0.8859 - val_loss: 0.3468 - val_accuracy: 0.8790
Epoch 5/50
94/94 [==============================] - 21s 229ms/step - loss: 0.2980 - accuracy: 0.8953 - val_loss: 0.3164 - val_accuracy: 0.8898
Epoch 6/50
94/94 [==============================] - 21s 221ms/step - loss: 0.2798 - accuracy: 0.9015 - val_loss: 0.3163 - val_accuracy: 0.8883
Epoch 7/50
94/94 [==============================] - 20s 209ms/step - loss: 0.2654 - accuracy: 0.9062 - val_loss: 0.3172 - val_accuracy: 0.8894
Epoch 8/50
94/94 [==============================] - 20s 212ms/step - loss: 0.2540 - accuracy: 0.9109 - val_loss: 0.2970 - val_accuracy: 0.8955
Epoch 9/50
94/94 [==============================] - 20s 208ms/step - loss: 0.2391 - accuracy: 0.9154 - val_loss: 0.2851 - val_accuracy: 0.9003
```

```
Epoch 10/50
94/94 [==============================] - 20s 209ms/step - loss: 0.2285 - accuracy: 0.9189 - val_loss: 0.2814 - val_accuracy: 0.9028
Epoch 11/50
94/94 [==============================] - 20s 211ms/step - loss: 0.2192 - accuracy: 0.9209 - val_loss: 0.2774 - val_accuracy: 0.9047
Epoch 12/50
94/94 [==============================] - 21s 222ms/step - loss: 0.2113 - accuracy: 0.9244 - val_loss: 0.2741 - val_accuracy: 0.9061
Epoch 13/50
94/94 [==============================] - 19s 204ms/step - loss: 0.2007 - accuracy: 0.9293 - val_loss: 0.2687 - val_accuracy: 0.9073
Epoch 14/50
94/94 [==============================] - 20s 209ms/step - loss: 0.1960 - accuracy: 0.9301 - val_loss: 0.2772 - val_accuracy: 0.9056
Epoch 15/50
94/94 [==============================] - 20s 211ms/step - loss: 0.1856 - accuracy: 0.9338 - val_loss: 0.2713 - val_accuracy: 0.9068
Epoch 16/50
94/94 [==============================] - 20s 211ms/step - loss: 0.1766 - accuracy: 0.9374 - val_loss: 0.2697 - val_accuracy: 0.9079
Epoch 17/50
94/94 [==============================] - 20s 208ms/step - loss: 0.1698 - accuracy: 0.9401 - val_loss: 0.2700 - val_accuracy: 0.9075
Epoch 18/50
94/94 [==============================] - 20s 216ms/step - loss: 0.1629 - accuracy: 0.9417 - val_loss: 0.2627 - val_accuracy: 0.9087
```

Step6:

Evaluate the model

model.evaluate(X_test, y_test)

313/313 [==============================] - 2s 7ms/step - loss: 0.3649 - accuracy: 0.9113
[0.3649125099182129, 0.911300003528595]

Get 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)


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_la
bels)
print(cr)
```

Save the model

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

Step-7:-

Build 2 Complex CNN

To evaluate more models build 2 complex cnn

```
cnn_model2 = keras.models.Sequential([
                        keras.layers.Conv2D(filters=32, kernel_size=3
, strides=(1,1), padding='valid',activation= 'relu', input_shape=[28,2
8,1]),
                        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='rel
u'),
                        keras.layers.Dropout(0.25),
                        keras.layers.Dense(units=256, activation='rel
u'),
                        keras.layers.Dropout(0.25),
                        keras.layers.Dense(units=128, activation='rel
u'),
                        keras.layers.Dense(units=10, activation='soft
max')
                        ])

cnn_model2.compile(optimizer='adam', loss= 'sparse_categorical_crossen
tropy', metrics=['accuracy'])
```

Including more hidden layers and test it

Epoch 1/30
94/94 [==============================] - 25s 262ms/step - loss: 0.7085 - accuracy: 0.7284 - val_loss: 0.5226 - val_accuracy: 0.7979
Epoch 2/30
94/94 [==============================] - 24s 255ms/step - loss: 0.5161 - accuracy: 0.8083 - val_loss: 0.4410 - val_accuracy: 0.8372
Epoch 3/30
94/94 [==============================] - 27s 289ms/step - loss: 0.4433 - accuracy: 0.8377 - val_loss: 0.3961 - val_accuracy: 0.8535

Epoch 4/30
94/94 [==============================] - 25s 264ms/step - loss: 0.3926 - accuracy: 0.8554 - val_loss: 0.3733 - val_accuracy: 0.8618
Epoch 5/30
94/94 [==============================] - 25s 261ms/step - loss: 0.3569 - accuracy: 0.8701 - val_loss: 0.3354 - val_accuracy: 0.8761
Epoch 6/30
94/94 [==============================] - 24s 255ms/step - loss: 0.3299 - accuracy: 0.8792 - val_loss: 0.3193 - val_accuracy: 0.8828
Epoch 7/30
94/94 [==============================] - 25s 263ms/step - loss: 0.3161 - accuracy: 0.8838 - val_loss: 0.3111 - val_accuracy: 0.8842
Epoch 8/30
94/94 [==============================] - 26s 272ms/step - loss: 0.2941 - accuracy: 0.8910 - val_loss: 0.3034 - val_accuracy: 0.8869
Epoch 9/30
94/94 [==============================] - 24s 256ms/step - loss: 0.2824 - accuracy: 0.8947 - val_loss: 0.2910 - val_accuracy: 0.8941
Epoch 10/30
94/94 [==============================] - 24s 257ms/step - loss: 0.2707 - accuracy: 0.8984 - val_loss: 0.2875 - val_accuracy: 0.8973
Epoch 11/30
94/94 [==============================] - 24s 254ms/step - loss: 0.2600 - accuracy: 0.9031 - val_loss: 0.2833 - val_accuracy: 0.8985
Epoch 12/30
94/94 [==============================] - 24s 253ms/step - loss: 0.2516 - accuracy: 0.9068 - val_loss: 0.2778 - val_accuracy: 0.8998
Epoch 13/30
94/94 [==============================] - 24s 258ms/step - loss: 0.2415 - accuracy: 0.9098 - val_loss: 0.2720 - val_accuracy: 0.9026
Epoch 14/30
94/94 [==============================] - 25s 266ms/step - loss: 0.2350 - accuracy: 0.9129 - val_loss: 0.2706 - val_accuracy: 0.9017
Epoch 15/30
94/94 [==============================] - 24s 252ms/step - loss: 0.2298 - accuracy: 0.9147 - val_loss: 0.2725 - val_accuracy: 0.9026
Epoch 16/30
94/94 [==============================] - 25s 262ms/step - loss: 0.2204 - accuracy: 0.9172 - val_loss: 0.2633 - val_accuracy: 0.9068
Epoch 17/30
94/94 [==============================] - 24s 256ms/step - loss: 0.2150 - accuracy: 0.9208 - val_loss: 0.2761 - val_accuracy: 0.9044
Epoch 18/30
94/94 [==============================] - 24s 257ms/step - loss: 0.2172 - accuracy: 0.9196 - val_loss: 0.2631 - val_accuracy: 0.9086
Epoch 19/30
94/94 [==============================] - 25s 267ms/step - loss: 0.2056 - accuracy: 0.9237 - val_loss: 0.2615 - val_accuracy: 0.9107
Epoch 20/30
94/94 [==============================] - 24s 251ms/step - loss: 0.1978 - accuracy: 0.9257 - val_loss: 0.2702 - val_accuracy: 0.9055
Epoch 21/30
94/94 [==============================] - 24s 253ms/step - loss: 0.1907 - accuracy: 0.9290 - val_loss: 0.2616 - val_accuracy: 0.9072
Epoch 22/30

```
94/94 [==============================] - 24s 255ms/step - loss: 0.1874 - accuracy: 0.9307 - val_loss: 0.2696 - val_accuracy: 0.9101
Epoch 23/30
94/94 [==============================] - 24s 251ms/step - loss: 0.1831 - accuracy: 0.9327 - val_loss: 0.2726 - val_accuracy: 0.9095
Epoch 24/30
94/94 [==============================] - 24s 253ms/step - loss: 0.1778 - accuracy: 0.9348 - val_loss: 0.2802 - val_accuracy: 0.9060
Epoch 25/30
94/94 [==============================] - 25s 266ms/step - loss: 0.1705 - accuracy: 0.9359 - val_loss: 0.2682 - val_accuracy: 0.9118
Epoch 26/30
94/94 [==============================] - 24s 258ms/step - loss: 0.1691 - accuracy: 0.9368 - val_loss: 0.2750 - val_accuracy: 0.9117
Epoch 27/30
94/94 [==============================] - 24s 251ms/step - loss: 0.1623 - accuracy: 0.9384 - val_loss: 0.2665 - val_accuracy: 0.9131
Epoch 28/30
94/94 [==============================] - 24s 256ms/step - loss: 0.1583 - accuracy: 0.9402 - val_loss: 0.2847 - val_accuracy: 0.9082
Epoch 29/30
94/94 [==============================] - 24s 255ms/step - loss: 0.1544 - accuracy: 0.9431 - val_loss: 0.2748 - val_accuracy: 0.9119
Epoch 30/30
94/94 [==============================] - 25s 269ms/step - loss: 0.1518 - accuracy: 0.9429 - val_loss: 0.2860 - val_accuracy: 0.9091
<keras.callbacks.History at 0x7f5f8a36af10>
```

CODE:-