Deep Learning Project Fashion MNIST Classification

Raneet Roy

Department of Computer Science and Engineering
Dr. B. C. Roy Engineering College

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.

Keywords image classification · MNIST

1. OBJECTIVE

The objective is to identify (predict) different fashion products from the given images using a Convolutional Neural Networks model. 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.

2. INTRODUCTION

The Fashion-MNIST dataset is proposed as a more challenging replacement dataset for the MNIST dataset. It is a dataset comprised of 60,000 small square 28×28 pixel grayscale images of items of 10 types of clothing, such as shoes, t-shirts, dresses, and more. The mapping of all 0-9 integers to class labels is listed below.

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

It is a more challenging classification problem than MNIST and top results are achieved by deep learning convolutional neural networks with a classification accuracy of about 90% to 95% on the hold out test dataset.

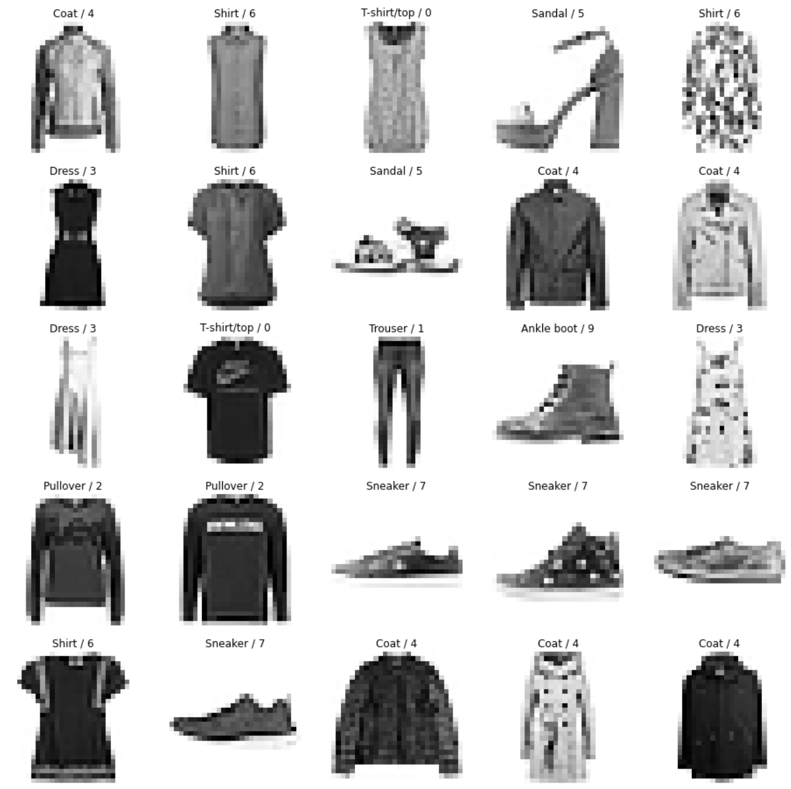


Fig. 1: Set of Images From the Fashion-MNIST Dataset

There are 60,000 examples in the training dataset and 10,000 in the test dataset and that images are indeed square with 28×28 pixels.

We cannot make use of fully connected networks when it comes to Convolutional Neural Networks because, we have considered an input of images with the size 28x28x3 pixels. If we input this to our Convolutional Neural Network, we will have about 2352 weights in the first hidden layer itself.

Any generic input image will at least have 200x200x3pixels in size. The size of the first hidden layer becomes a whooping 120000. If this is just the first hidden layer, imagine the number of neurons needed to process an entire complex image-set.This leads to over-fitting and isn't practical. Hence, we cannot make use of fully connected networks.

2.1 CNN :-

Convolutional Neural Networks, like neural networks, are made up of **neurons** with **learnable weights** and **biases**. Each **neuron** receives several **inputs**, takes a weighted **sum** over them, **pass** it through an **activation function** and responds with an **output**.
The whole network has a **loss function** and all the tips and tricks that we developed for neural networks still apply on **Convolutional Neural Networks.**
Neural networks, as its name suggests, is a **machine learning technique** which is modeled after the **brain** structure. It comprises of a network of **learning units** called neurons.
These **neurons** learn how to convert **input signals** (e.g. picture of a cat) into corresponding **output signals** (e.g. the label "cat"), forming the basis of automated recognition.
The goal is that if the **input signal** looks like **previous** images it has seen before, the **"image" reference** signal will be mixed into, or **convolved** with, the **input** signal. The resulting **output** signal is then passed on to the **next layer. So the computer understands every pixel. We take small patches of the pixels called filters and try to match them in the corresponding nearby locations to see if we get a match. By doing this, the Convolutional Neural Network gets a lot better at seeing similarity than directly trying to match the entire image.**

In this paper, we report a model that can achieve a very high accuracy on the MNIST test set without complex structural aspects or learning techniques. The model uses a set of convolution layers followed by a fully connected layer at the end, which is one of the commonly used model architectures. We use basic data augmentation schemes, translation and rotation. We train three models with similar architectures, and use majority voting between the models to obtain the final prediction. The three models have similar architectures, but have different kernel sizes in the convolution layers. Experiments show that combining models with different kernel sizes achieves better accuracy than combining models with the same kernel size.

3. METHODOLOGY

The Fashion MNIST dataset was developed as a response to the wide use of the MNIST dataset, 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. 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 design of the test harness is modular, and we can develop a separate function for each piece. This allows a given aspect of the test harness to be modified or inter-changed, if we desire, separately from the rest.
We can develop this test harness with five key elements. They are the loading of the dataset, the preparation of the dataset, the definition of the model, the evaluation of the model, and the presentation of results.
At First we need to Import all the libraries as follows :- numpy, matplotlib.pyplot, seaborn, tensorflow, keras.

Then Download the data from Keras Datasets. We know that the images are all pre-segmented (e.g. each image contains a single item of clothing), that the images all have the same square size of 28×28 pixels, and that the images are grayscale. Therefore, we can load the images and reshape the data arrays to have a single color channel.

Then Show the image from numbers . 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.

A good starting point is to normalize the pixel values of grayscale images, e.g. rescale them to the range [0,1]. This involves first converting the data type from unsigned integers to floats, then dividing the pixel values by the maximum value.

Next, we need to define a baseline convolutional neural network model for the problem.
There are **four** layered **concepts** we should understand in Convolutional Neural Networks:

1. Convolution,
2. ReLu,
3. Pooling and
4. Full Connectedness (Fully Connected Layer).

3.1 CONVOLUTION OF AN IMAGE :-

Convolution has the nice property of being **translational invariant**. Intuitively, this means that **each** convolution filter represents a **feature** of interest (e.g **pixels in letters)** and the Convolutional Neural Network **algorithm** learns which **features** comprise the **resulting reference** (i.e. alphabet).

The 4 steps of convolution are as follows :-

1> Line up the feature and the image.
2> Multiply each image pixel by corresponding feature pixel.
3> Add the values and find the sum.
4> Divide the sum by the total number of pixels in the feature.
5> Put the final value in the center. Now, we can **move** this **filter** around and do the **same** at **any pixel** in the image.

3.2 ReLU LAYER:-

ReLU is an activation function. **Rectified Linear Unit** (ReLU) transform function only activates a node if the input is above a certain quantity, while the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent variable.

The main aim is to remove all the negative values from the convolution. All the positive values remain the same but all the negative values get changed to zero.

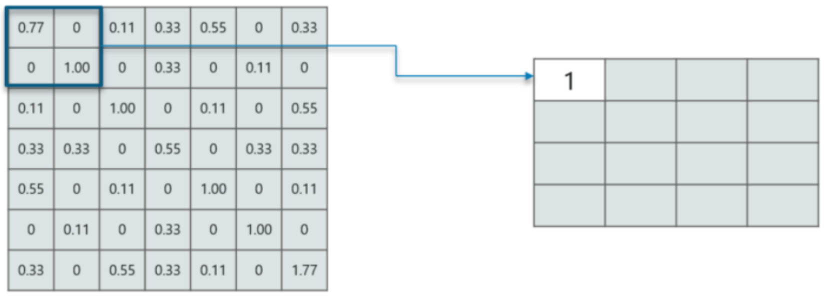
Similarly we do the same process to all the other feature images as well.

3.3 POOLING LAYER:-

In this layer we **shrink** the **image** stack into a **smaller size.** Pooling is done **after passing** through the **activation** layer. We do this by implementing the following 4 steps:

- Pick a **window size** (usually 2 or 3)
- Pick a **stride** (usually 2)
- **Walk** the window **across** the **filtered** images
- From each **window,** take the **maximum** value

Consider performing pooling with a window size of 2 and stride being 2 as well. So in this case, we took **window size** to be **2** and we got **4 values** to choose from. From those 4 values, the **maximum value** there is 1 so we pick 1. We **started out** with a **7×7** matrix but now the same matrix after **pooling** came down to **4×4.**

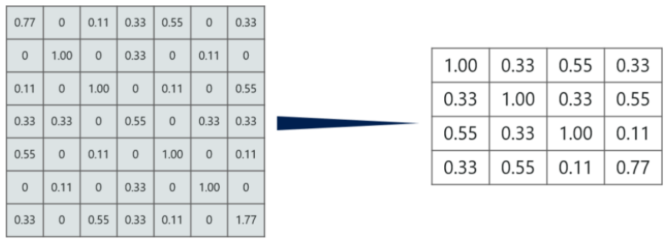


0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	1.77

1			

Fig. 2: A 4x4 matrix is formed from a 7x7 matrix

But we need to **move** the **window across** the **entire** image. The procedure is exactly as same as above and we need to repeat that for the entire image.



0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	1.77

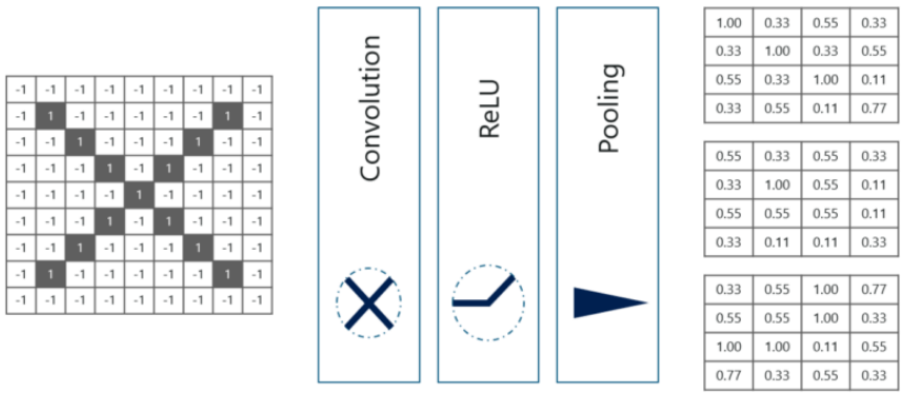
1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

Fig. 3: results obtained when we move the **window across** the **entire** image

This is for **one filter.** We need to do it for 2 other filters as well.

3.4 STACKING UP ALL THE LAYERS:-

to get the **time-frame** in one picture we're here with a **4×4** matrix from a **7×7** matrix after passing the input through 3 layers – **Convolution, ReLU** and **Pooling** as shown in the image:



-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

Fig. 4: A 4x4 matrix is formed out of the 7x7 matrix from all the 3 layers

So after the second pass we arrive at a 2×2 matrix as shown:

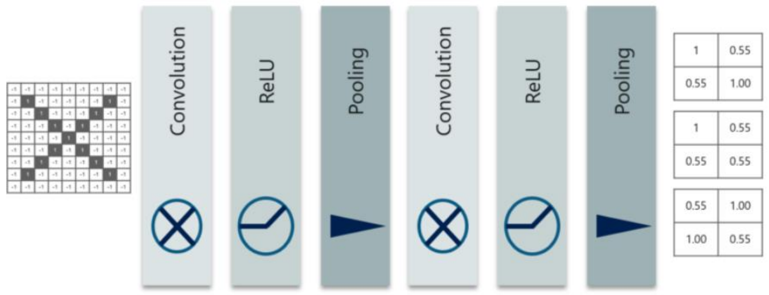
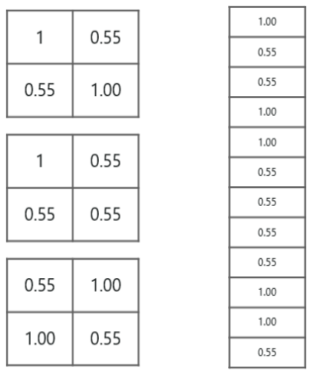


Fig. 5: 2x2 Matrixes are formed out of the 3 layers

The last layers in the network are **fully connected,** meaning that neurons of preceding layers are **connected** to **every neuron** in **subsequent** layers.

This **mimics high level reasoning** where all possible **pathways** from the **input** to **output** are considered.

Also, fully connected layer is the final layer where the classification actually happens. Here we take our filtered and shrinked images and put them into one single list as shown :



1	0.55
0.55	1.00

1	0.55
0.55	0.55

0.55	1.00
1.00	0.55

1.00
0.55
0.55
1.00
1.00
0.55
0.55
0.55
0.55
1.00
1.00
0.55

Fig. 6: filtered and shrinked images are put into one single list

When we feed in, **'trouser'** and **'shirt'** there will be **some element** in the vector that will be **high.** Consider the image below, as you can see for **'trouser'** there are **different elements** that are **high** and **similarly,** for **'shirt'** we have **different elements** that are **high:**

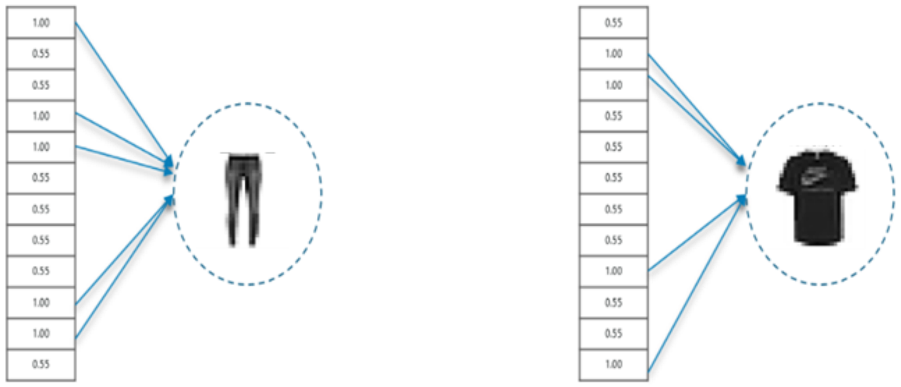


Fig 7: Comparison of values in the fully connected layer for a Trouser vs a Shirt.

When the **1st, 4th, 5th, 10th** and **11th** values are **high,** we can classify the image as **'trouser'.** The concept is similar for the other **alphabets** as well – when certain **values** are arranged the way they are, they can be **mapped** to an **actual** letter or a **number** which we **require.**

In this paper we use 3 model architectures to classify Fashion MNIST dataset . In the conclusion section we show how one of the model architectures perform better than the other. The model architectures are shown in figure8.

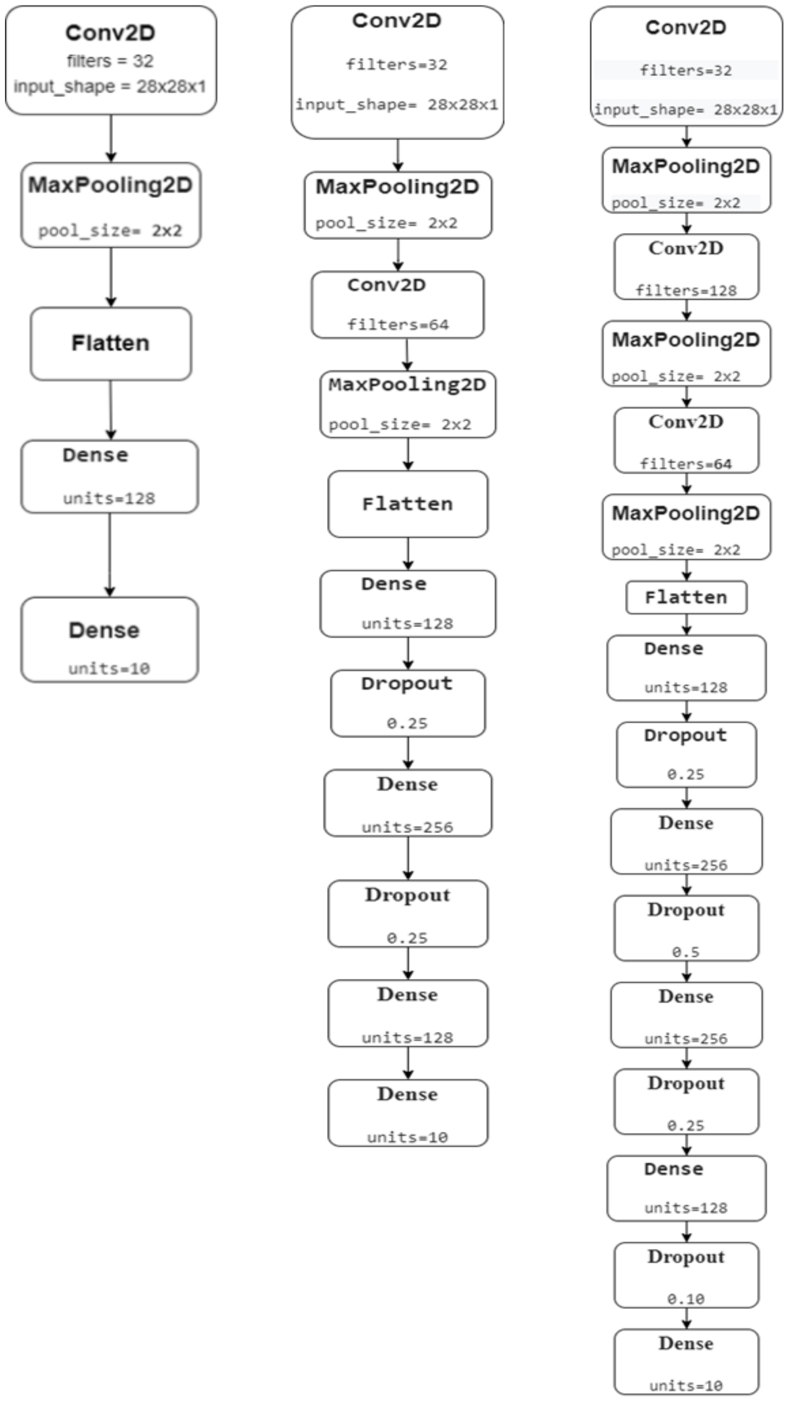


Fig. 8: The 3 Model Architectures

3.5 TESTING AND EVALUATION OF CNN MODEL:-

We have a **12 element** vector obtained after **passing** the **input** of a **random letter** through all the **layers** of our **network.**

But, **in order to** check what **we've obtained** is right or wrong, we **make predictions** based on the **output** data by comparing the **obtained values** with list of 'trouser' and 'shirt'!

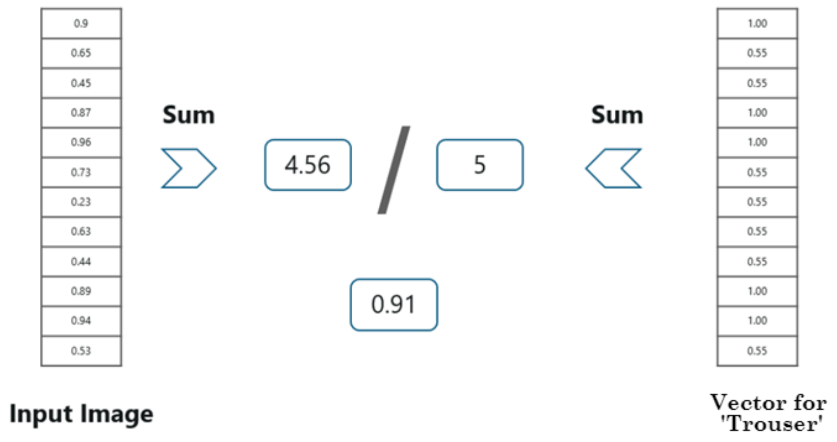


Fig. 9: This represents the probability match for trouser is 0.91

We just **added** the values we which found out as high (1st, 4th, 5th, 10th and 11th) from the **vector table** of **Trouser** and we got the sum to be **5.** We did the **exact same thing** with the **input image** and got a value of **4.56**.

When we **divide** the **value** we have a **probability match** to be **0.91.**

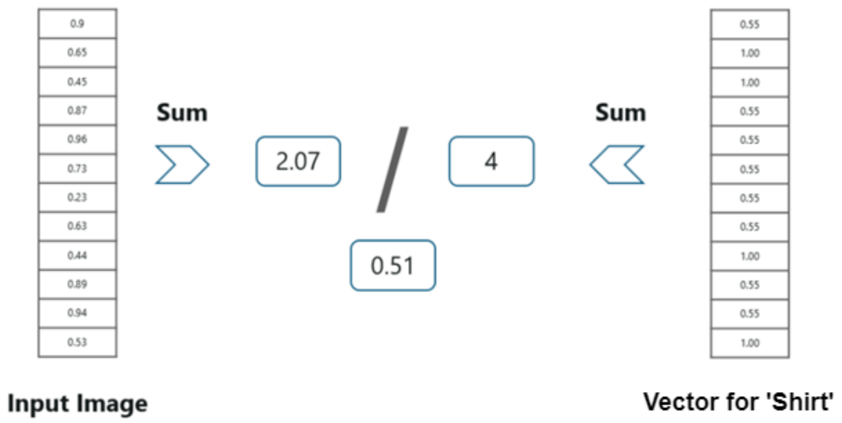


Fig. 10: This represents the probability match for shirt is 0.51

The output is 0.51 with this table, probability being 0.51 is less than 0.91. So we can conclude that the resulting input image is an 'trouser'.

3.5.1 CONFUSION MATRIX FOR MULTI-CLASS CLASSIFICATION

Confusion Matrix is used to know the performance of a Machine learning classification. It is represented in a matrix form.
Confusion Matrix gives a comparison between Actual and predicted values.
The confusion matrix is a N x N matrix, where N is the number of classes or outputs.
For 2 class ,we get 2 x 2 confusion matrix.
For 3 class ,we get 3 x 3 confusion matrix.
Confusion Matrix has 4 terms to understand True Positive(TP),False Positive(FP),True Negative(TN) and False Negative(FN).

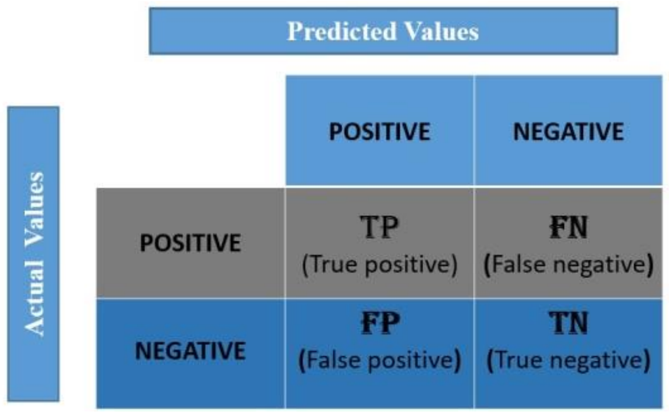


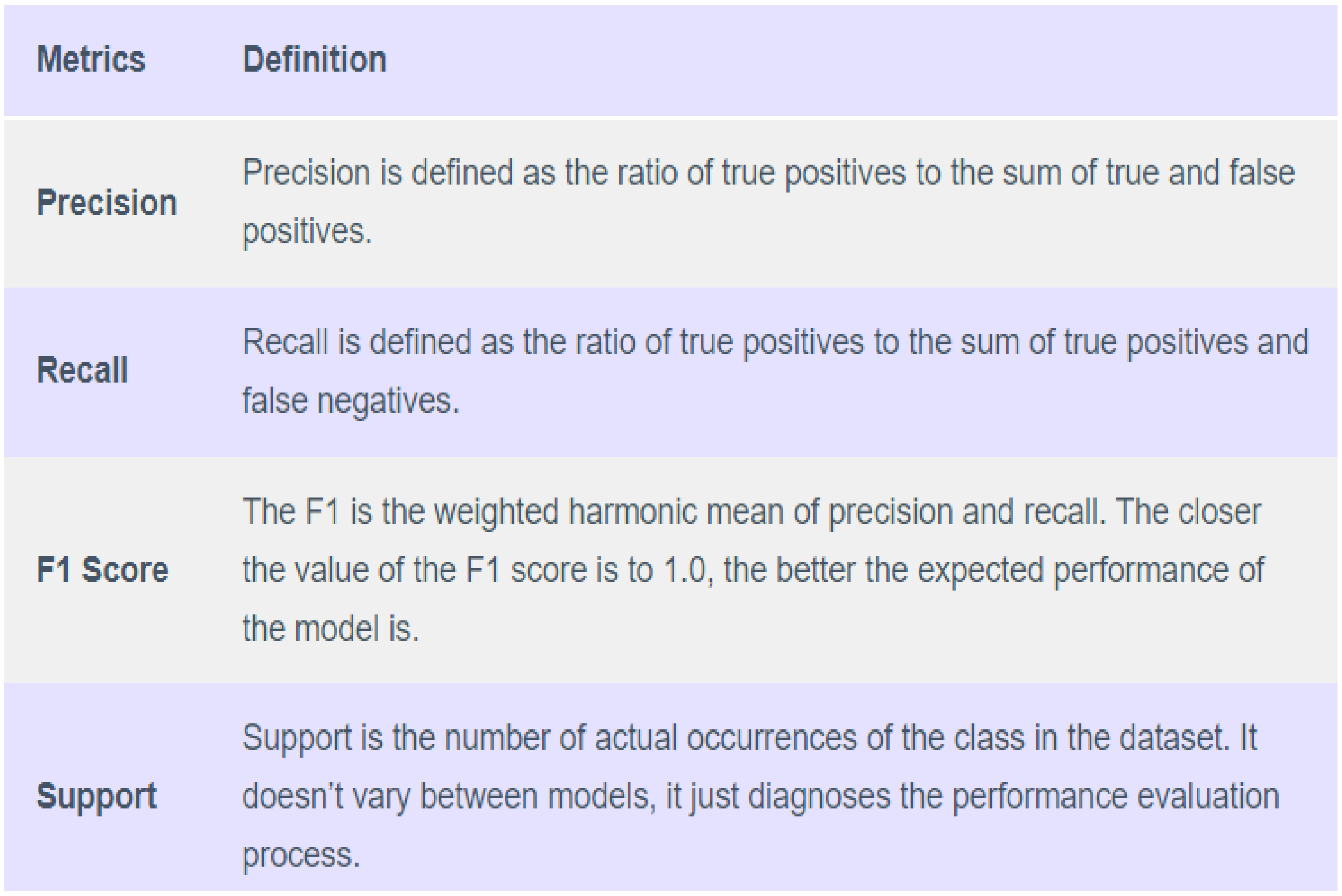
Fig. 11: Cross of predicted and actual values

How to calculate FN, FP, TN, and TP:

FN: The False-negative value for a class will be the sum of values of corresponding rows except for the TP value.
FP: The False-positive value for a class will be the sum of values of the corresponding column except for the TP value.
TN: The True Negative value for a class will be the sum of values of all columns and rows except the values of that class that we are calculating the values for.
TP: The True positive value is where the actual value and predicted value are the same.

3.5.2 Classification Report

It is one of the performance evaluation metrics of a classification-based machine learning model. It displays the model’s precision, recall, F1 score and support. It provides a better understanding of the overall performance of our trained model. To understand the classification report of a machine learning model, we need to know all of the metrics displayed in the report.



Metrics	Definition
Precision	Precision is defined as the ratio of true positives to the sum of true and false positives.
Recall	Recall is defined as the ratio of true positives to the sum of true positives and false negatives.
F1 Score	The F1 is the weighted harmonic mean of precision and recall. The closer the value of the F1 score is to 1.0, the better the expected performance of the model is.
Support	Support is the number of actual occurrences of the class in the dataset. It doesn’t vary between models, it just diagnoses the performance evaluation process.

4. CODE

#At first, we need to import the libraries
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import keras
```

#Then, we load the data.
```
 (X_train, y_train), (X_test, y_test)=tf.keras.datasets.fashion_mnist.load_data()
```

#Now, print the shape of data.
```
print(X_train.shape,y_train.shape, X_test.shape, y_test.shape)

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

#Now, show the image.
```
 plt.imshow(X_train[0],cmap='Greys')

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

#Now Feature Scaling.
```
X_train = X_train/255
X_test= X_test/255
```

#Then 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_size=0.2,rando
m_state=2020)
print(X_train.shape,X_Validation.shape,y_train.shape,y_Validation.shape)
```

#Now, Buiding the CNN model – Model Architecture 1: -

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

                        keras.layers.Conv2D(filters=32,kernel_size=3,strides=(1,1),padding='valid',activa
            tion='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.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))


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

 y_test

 model.evaluate(X_test, y_test)


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')
```

#'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_labels)
print(cr)
```

#""" Save Model"""

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

#Lastly , we need to build 2 complex CNN.

#Building CNN model – Model Architecture 2: -

```
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')
                    ])
```

#Compiling 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')

cnn_model2.evaluate(X_test, y_test)
```

#Building CNN model – Model Architecture 3: -

```
from keras.callbacks import EarlyStopping
es = Earlystopping(monitor=”val_loss”, mode = “min”, verbose = 1)

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')
            ])
```

#Compile the model: -

```
cnn_model3.compile(optimizer='adam', loss= 'sparse_categorical_crossentropy', metrics=['accuracy'])
```

#Train the Model: -

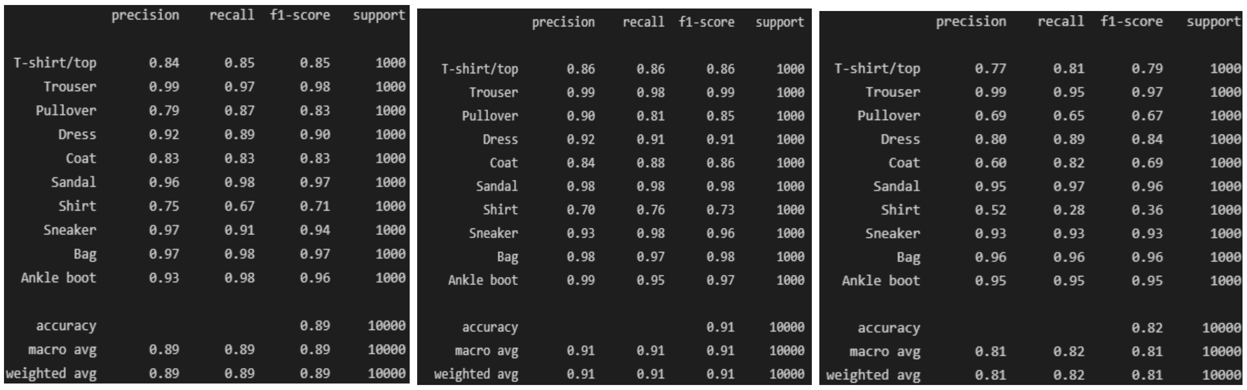
```
cnn_model3.fit(X_train, y_train, epochs=50, batch_size=512, verbose=1, validation_data=(X_Validation, y_Validation), callbacks = [es])

cnn_model3.save('fashion_mnist_cnn_model3.h5')

cnn_model3.evaluate(X_test, y_test)
```

5. RESULTS and DISCUSSION

In the first CNN model, we could achieve a **training accuracy of 92.07%** and **test accuracy of 89%** confirming that model is fine **with no overfitting**. The most correctly predicted class is trouser where 981 out of 1000 trouser images were predicted correctly.

In the second CNN model, we could achieve a **training accuracy of 92.45%** and **test accuracy of 91%** confirming that model is fine **with no overfitting**. The most correctly predicted class are sandal and sneaker which has a value of 980/1000 each. The number of images predicted correctly for the other classes are also either higher than or almost equal to the first CNN model's predictions confirming the higher accuracy here.

In the third CNN model, we use early stopping to avoid overfitting. Hence while training , instead of running for 50 complete epochs , it ran for x epochs. We could achieve a **training accuracy of 78%** and **test accuracy of 82%** confirming that model is fine **with no overfitting which is the worst amongst the other two models**. The most correctly predicted class are sandal which has a value of 967/1000.



```
              precision    recall  f1-score   support

 T-shirt/top       0.84      0.85      0.85      1000
     Trouser       0.99      0.97      0.98      1000
    Pullover       0.79      0.87      0.83      1000
       Dress       0.92      0.89      0.90      1000
        Coat       0.83      0.83      0.83      1000
      Sandal       0.96      0.98      0.97      1000
       Shirt       0.75      0.67      0.71      1000
     Sneaker       0.97      0.91      0.94      1000
         Bag       0.97      0.98      0.97      1000
  Ankle boot       0.93      0.98      0.96      1000

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

```
              precision    recall  f1-score   support

 T-shirt/top       0.86      0.86      0.86      1000
     Trouser       0.99      0.98      0.99      1000
    Pullover       0.90      0.81      0.85      1000
       Dress       0.92      0.91      0.91      1000
        Coat       0.84      0.88      0.86      1000
      Sandal       0.98      0.98      0.98      1000
       Shirt       0.70      0.76      0.73      1000
     Sneaker       0.93      0.98      0.96      1000
         Bag       0.98      0.97      0.98      1000
  Ankle boot       0.99      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
```

```
              precision    recall  f1-score   support

 T-shirt/top       0.77      0.81      0.79      1000
     Trouser       0.99      0.95      0.97      1000
    Pullover       0.69      0.65      0.67      1000
       Dress       0.80      0.89      0.84      1000
        Coat       0.60      0.82      0.69      1000
      Sandal       0.95      0.97      0.96      1000
       Shirt       0.52      0.28      0.36      1000
     Sneaker       0.93      0.93      0.93      1000
         Bag       0.96      0.96      0.96      1000
  Ankle boot       0.95      0.95      0.95      1000

    accuracy                           0.82     10000
   macro avg       0.81      0.82      0.81     10000
weighted avg       0.81      0.82      0.81     10000
```

Fig. 12: represents the tables for the 3 models with their test accuracies , macro avgs and weighted avgs respectively.

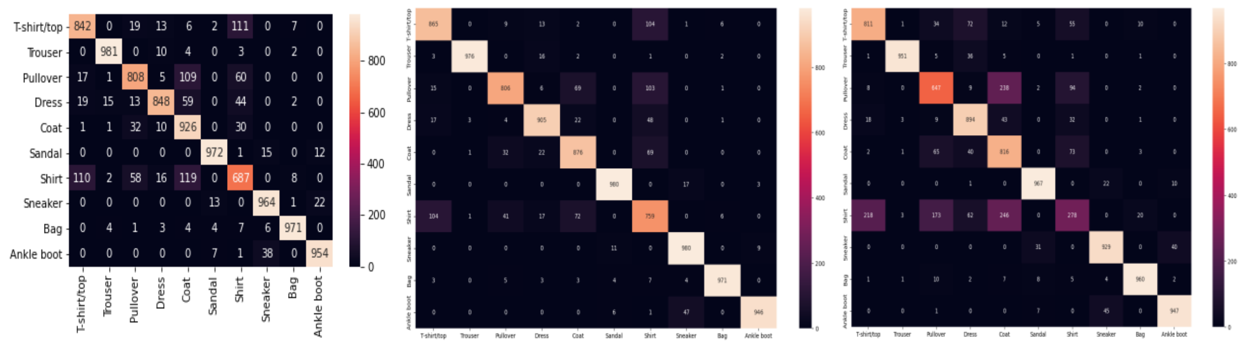


Fig. 13: represents the predicted values for the different items for the 3 models .

6. CONCLUSION

In this report, we address the problem of distinguishing clothing elements in fashion images using a CNN. This is accomplished using MNIST dataset, which contains many images, together with CNN algorithms for accurate and effective image classification.

The best model out of the 3 was the 2nd CNN model, where we could achieve a **training accuracy of 92.45%** and **test accuracy of 91%** confirming that model is fine **with no overfitting**. The number of images predicted correctly for the other classes are also either higher than or almost equal to the first CNN model's predictions confirming the higher accuracy here.

Identifying an article type in the fashion industry is a crucial task. This model can serve as a building block for a service which shows related fashion articles based on a given image.