**Deep learning project**

**Fashion MNIST classification project**

****

**Presented by:-**

**Name-Debasis Pradhan**

**Artificial Intelligence -Microsoft -B3**

**Contents:-**

[**1.ABSTRACT:-** 3](#_Toc119712121)

[**2.OBJECTIVE:-** 4](#_Toc119712122)

[**3.INTRODUCTION:-** 6](#_Toc119712123)

[**4.METHODOLOGIES:-** 11](#_Toc119712124)

[**5.CODE:-** 13](#_Toc119712125)

[**6.Conclusion:-** 25](#_Toc119712126)

# **1.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.

# **2.OBJECTIVE:-**

The objective is to identify (predict) different fashion products from the given images using various best possible Machine Learning Models (Algorithms) and compare their results (performance measures/scores) to arrive at the best ML model.

I have used Fashion-MNIST dataset for this experiment with Machine Learning. Fashion-MNIST dataset is a collection of fashion articles images.

One of the most powerful and compelling types of AI is computer vision which we have almost surely experienced in any number of ways without even knowing.

Computer vision is the field of the human vision system and enabling computers to identify and process objects in images and videos in the same way that humans do. Until recently, computer vision only worked in limited capacity.

Thanks to advances in artificial intelligence and innovations in deep learning and neural networks, the field has been able to take great leaps in recent years and has been able to surpass humans in some tasks related to detecting and labelling objects.

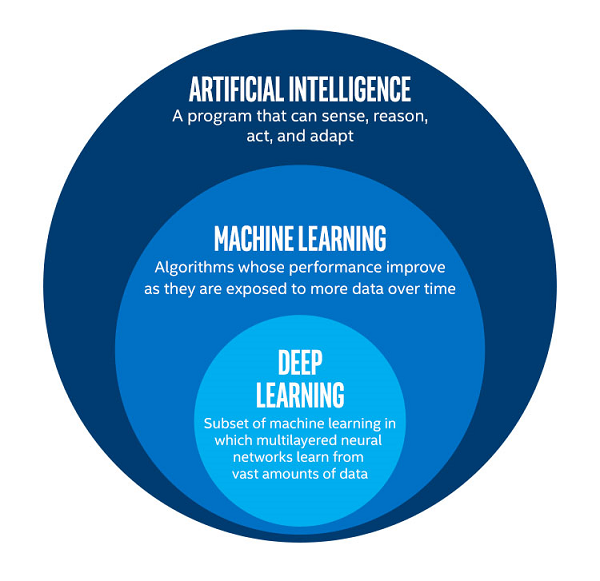
One of the driving factors behind the growth of computer vision is the amount of data we generate today that is then used to train and make computer vision better. Neural networks can now have millions of trainable parameters which makes technologies like diseases self-diagnoses or self-driving car possible.

The prime objective of this project is to implement a CNN to perform image classification on the famous fashion design MNIST dataset. In this, we will be implementing our own CNN architecture. The process will be divided into three steps: Data analysis, Model training, and Prediction.

**An overview of Deep Learning:**

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human brain so deep learning is also a kind of mimic of human brain. In deep learning, we don’t need to explicitly program everything. The concept of deep learning is not new. It has been around for a couple of years now. It’s on hype nowadays because earlier we did not have that much processing power and a lot of data. As in the last 20 years, the processing power increases exponentially, deep learning and machine learning came in the picture.

A formal definition of deep learning is- neurons.



# **3.INTRODUCTION:-**

**Convolutional Neural Networks:-**

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.

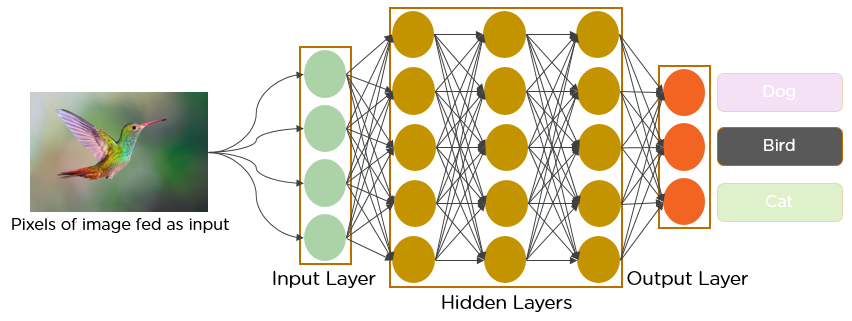
The whole network has a loss function and all the tips and tricks that we developed for neural networks still apply on Convolutional Neural Networks.

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.

Let’s take the example of automatic image recognition. The process of determining whether a picture contains a cat involves an activationfunction. If the picture resembles prior cat images the neurons have seenbefore**,** the label **“**cat**”** would be activated**.**

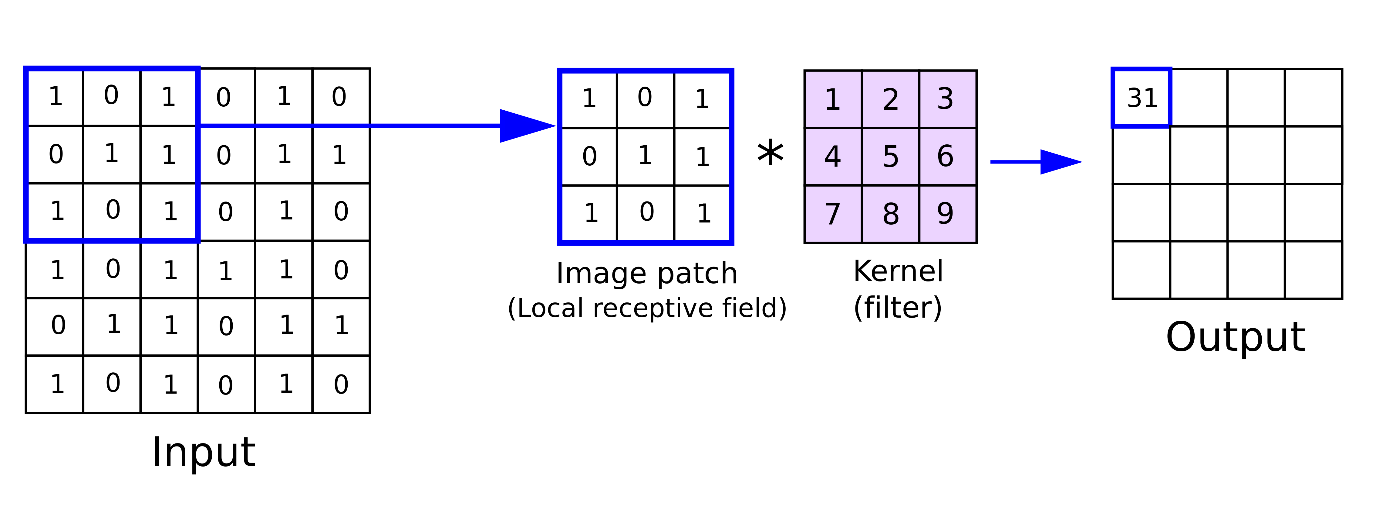
Hence**,** the more labeled images the neurons are exposed to, the better it learns how to recognize other unlabeled images. We call this the process of training neurons.



**Convolution:**

Convolution is one of the main building blocks of a CNN. The term [convolution](http://timdettmers.com/2015/03/26/convolution-deep-learning/) refers to the mathematical combination of two functions to produce a third function. It merges two sets of information.

In the case of a CNN, the convolution is performed on the input data with the use of a **filter** or **kernel**(these terms are used interchangeably)to then produce a **feature map**.



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).

There are 4 steps for convolution:

1.Line up the feature and the image

A picture containing table

Description automatically generated

2.Multiply each image pixel by corresponding feature panel.

A picture containing text, crossword puzzle

Description automatically generated

3.Add the values and find the sum.

4.Divide the sum by the total number of pixels in the feature.

Table

Description automatically generated

A picture containing text, crossword puzzle, keyboard

Description automatically generated

**ReLU Layer:-**

ReLU is an activation function. But, what is an activation function?

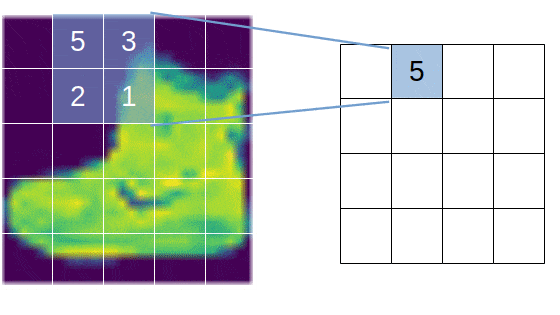
Rectified Linear Unit (ReLU) transform function only activates a node if the input is above a certain quantity, while the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent variable.

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 as shown in the image.

**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 your window across your filtered images
* From each window, take the maximum value



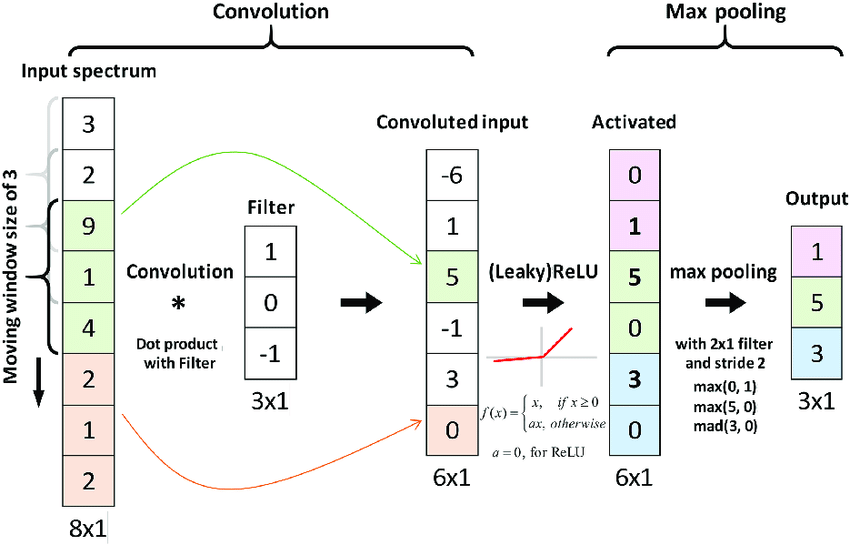
A picture containing text

Description automatically generated

**Max pooling:-**

**Max Pooling** is a pooling operation that calculates the maximum value for patches of a feature map, and uses it to create a down sampled (pooled) feature map. It is usually used after a convolutional layer. It adds a small amount of translation invariance - meaning translating the image by a small amount does not significantly affect the values of most pooled outputs.



****

# **4.METHODOLOGIES:-**

**Fashion MNIST Classification:-**

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**. We intend Fashion-MNIST to serve as a direct drop-in **replacement for the original MNIST** dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

**Steps:-**

1.Import Libraries

Lets import all the libraries we are going to require for this classification project. It is always good to put all the import statements at the begining of the file.

2.Load Data

Now lets use **pandas** library to read the train and test datasets in the respective csv files. We are going to use the **read\_csv** function which reads a csv file and returns a pandas **DataFrame** object.

3.Visualization

Now that we have loaded the data and also got somewhat acquainted with it lets visualize the actual images. We are going to use **Matplotlib** library for this.

4.Preprocess Data

Great! We have visualized the images. So now we can start preparing for creating our model. But before that we need to pre-process our data so that we can fit our model easily. Lets do that first.

Since we are dealing with image data and our task is to recognize and classify images our model should be a Convolutional Neural Network. For that our images should have at least 3 dimensions. But our images are flattened in one dimension, **784 pixel (28x28x1)** values per row. So we need to reshape the data into its original format.

Also we need to have three different sets of data for **training, validating** and **testing**. We already have different sets for training and testing. So we are going to split the training dataset further into two sets and will use one set of training and the other for validation.

**5.Create and Train the model**

* **Create the model**
* **compile the model**
* **Train the model**

**6.Evaluate the model**

* **Get the accuracy of the model**
* **Visualize the model's predictions**
* **Plot Confusion Matrix**
* **Classification Report**

**Hence we have successfully performed image classification on the fashion MNIST dataset.**

# **5.CODE:-**

**(1) Importing Libraries:-**

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

import keras

**(2) Load data:-**

>>>(x\_train,y\_train),(x\_test,y\_test)=tf.keras.datasets.fashion\_mnist.load\_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, 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)

>>> y\_train[0]

9

>>>class\_label = [ "T\_shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot" ]

>>>class\_label

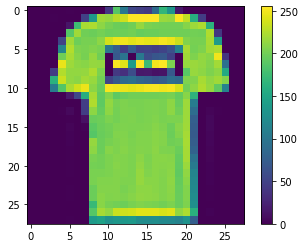
['T\_shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

>>>plt.figure()

>>>plt.imshow(x\_train[1])

>>>plt.colorbar()

<matplotlib.colorbar.Colorbar at 0x7fe5aebfa4d0>



>>>x\_train=x\_train/255

>>>x\_test=x\_test/255

>>>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. , 0.00392157, 0. , 0. , 0.05098039, 0.28627451, 0. , 0. , 0.00392157, 0.01568627, 0. , 0. , 0. , 0. , 0.00392157, 0.00392157, 0. ], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.01176471, 0. , 0.14117647, 0.53333333, 0.49803922, 0.24313725, 0.21176471, 0. , 0. , 0. , 0.00392157, 0.01176471, 0.01568627, 0. , 0. , 0.01176471], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.02352941, 0. , 0.4 , 0.8 , 0.69019608, 0.5254902 , 0.56470588, 0.48235294, 0.09019608, 0. , 0. , 0. , 0. , 0.04705882, 0.03921569, 0. ], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.60784314, 0.9254902 , 0.81176471, 0.69803922, 0.41960784, 0.61176471, 0.63137255, 0.42745098, 0.25098039, 0.09019608, 0.30196078, 0.50980392, 0.28235294, 0.05882353], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.00392157, 0. , 0.27058824, 0.81176471, 0.8745098 , 0.85490196, 0.84705882, 0.84705882, 0.63921569, 0.49803922, 0.4745098 , 0.47843137, 0.57254902, 0.55294118, 0.34509804, 0.6745098 , 0.25882353], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.00392157, 0.00392157, 0.00392157, 0. , 0.78431373, 0.90980392, 0.90980392, 0.91372549, 0.89803922, 0.8745098 , 0.8745098 , 0.84313725, 0.83529412, 0.64313725, 0.49803922, 0.48235294, 0.76862745, 0.89803922, 0. ], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.71764706, 0.88235294, 0.84705882, 0.8745098 , 0.89411765, 0.92156863, 0.89019608, 0.87843137, 0.87058824, 0.87843137, 0.86666667, 0.8745098 , 0.96078431, 0.67843137, 0. ], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.75686275, 0.89411765, 0.85490196, 0.83529412, 0.77647059, 0.70588235, 0.83137255, 0.82352941, 0.82745098, 0.83529412, 0.8745098 , 0.8627451 , 0.95294118, 0.79215686, 0. ], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.00392157, 0.01176471, 0. , 0.04705882, 0.85882353, 0.8627451 , 0.83137255, 0.85490196, 0.75294118, 0.6627451 , 0.89019608, 0.81568627, 0.85490196, 0.87843137, 0.83137255, 0.88627451, 0.77254902, 0.81960784, 0.20392157], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.02352941, 0. , 0.38823529, 0.95686275, 0.87058824, 0.8627451 , 0.85490196, 0.79607843, 0.77647059, 0.86666667, 0.84313725, 0.83529412, 0.87058824, 0.8627451 , 0.96078431, 0.46666667, 0.65490196, 0.21960784], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.01568627, 0. , 0. , 0.21568627, 0.9254902 , 0.89411765, 0.90196078, 0.89411765, 0.94117647, 0.90980392, 0.83529412, 0.85490196, 0.8745098 , 0.91764706, 0.85098039, 0.85098039, 0.81960784, 0.36078431, 0. ], [0. , 0. , 0.00392157, 0.01568627, 0.02352941, 0.02745098, 0.00784314, 0. , 0. , 0. , 0. , 0. , 0.92941176, 0.88627451, 0.85098039, 0.8745098 , 0.87058824, 0.85882353, 0.87058824, 0.86666667, 0.84705882, 0.8745098 , 0.89803922, 0.84313725, 0.85490196, 1. , 0.30196078, 0. ], [0. , 0.01176471, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.24313725, 0.56862745, 0.8 , 0.89411765, 0.81176471, 0.83529412, 0.86666667, 0.85490196, 0.81568627, 0.82745098, 0.85490196, 0.87843137, 0.8745098 , 0.85882353, 0.84313725, 0.87843137, 0.95686275, 0.62352941, 0. ], [0. , 0. , 0. , 0. , 0.07058824, 0.17254902, 0.32156863, 0.41960784, 0.74117647, 0.89411765, 0.8627451 , 0.87058824, 0.85098039, 0.88627451, 0.78431373, 0.80392157, 0.82745098, 0.90196078, 0.87843137, 0.91764706, 0.69019608, 0.7372549 , 0.98039216, 0.97254902, 0.91372549, 0.93333333, 0.84313725, 0. ], [0. , 0.22352941, 0.73333333, 0.81568627, 0.87843137, 0.86666667, 0.87843137, 0.81568627, 0.8 , 0.83921569, 0.81568627, 0.81960784, 0.78431373, 0.62352941, 0.96078431, 0.75686275, 0.80784314, 0.8745098 , 1. , 1. , 0.86666667, 0.91764706, 0.86666667, 0.82745098, 0.8627451 , 0.90980392, 0.96470588, 0. ], [0.01176471, 0.79215686, 0.89411765, 0.87843137, 0.86666667, 0.82745098, 0.82745098, 0.83921569, 0.80392157, 0.80392157, 0.80392157, 0.8627451 , 0.94117647, 0.31372549, 0.58823529, 1. , 0.89803922, 0.86666667, 0.7372549 , 0.60392157, 0.74901961, 0.82352941, 0.8 , 0.81960784, 0.87058824, 0.89411765, 0.88235294, 0. ], [0.38431373, 0.91372549, 0.77647059, 0.82352941, 0.87058824, 0.89803922, 0.89803922, 0.91764706, 0.97647059, 0.8627451 , 0.76078431, 0.84313725, 0.85098039, 0.94509804, 0.25490196, 0.28627451, 0.41568627, 0.45882353, 0.65882353, 0.85882353, 0.86666667, 0.84313725, 0.85098039, 0.8745098 , 0.8745098 , 0.87843137, 0.89803922, 0.11372549], [0.29411765, 0.8 , 0.83137255, 0.8 , 0.75686275, 0.80392157, 0.82745098, 0.88235294, 0.84705882, 0.7254902 , 0.77254902, 0.80784314, 0.77647059, 0.83529412, 0.94117647, 0.76470588, 0.89019608, 0.96078431, 0.9372549 , 0.8745098 , 0.85490196, 0.83137255, 0.81960784, 0.87058824, 0.8627451 , 0.86666667, 0.90196078, 0.2627451 ], [0.18823529, 0.79607843, 0.71764706, 0.76078431, 0.83529412, 0.77254902, 0.7254902 , 0.74509804, 0.76078431, 0.75294118, 0.79215686, 0.83921569, 0.85882353, 0.86666667, 0.8627451 , 0.9254902 , 0.88235294, 0.84705882, 0.78039216, 0.80784314, 0.72941176, 0.70980392, 0.69411765, 0.6745098 , 0.70980392, 0.80392157, 0.80784314, 0.45098039], [0. , 0.47843137, 0.85882353, 0.75686275, 0.70196078, 0.67058824, 0.71764706, 0.76862745, 0.8 , 0.82352941, 0.83529412, 0.81176471, 0.82745098, 0.82352941, 0.78431373, 0.76862745, 0.76078431, 0.74901961, 0.76470588, 0.74901961, 0.77647059, 0.75294118, 0.69019608, 0.61176471, 0.65490196, 0.69411765, 0.82352941, 0.36078431], [0. , 0. , 0.29019608, 0.74117647, 0.83137255, 0.74901961, 0.68627451, 0.6745098 , 0.68627451, 0.70980392, 0.7254902 , 0.7372549 , 0.74117647, 0.7372549 , 0.75686275, 0.77647059, 0.8 , 0.81960784, 0.82352941, 0.82352941, 0.82745098, 0.7372549 , 0.7372549 , 0.76078431, 0.75294118, 0.84705882, 0.66666667, 0. ], [0.00784314, 0. , 0. , 0. , 0.25882353, 0.78431373, 0.87058824, 0.92941176, 0.9372549 , 0.94901961, 0.96470588, 0.95294118, 0.95686275, 0.86666667, 0.8627451 , 0.75686275, 0.74901961, 0.70196078, 0.71372549, 0.71372549, 0.70980392, 0.69019608, 0.65098039, 0.65882353, 0.38823529, 0.22745098, 0. , 0. ], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.15686275, 0.23921569, 0.17254902, 0.28235294, 0.16078431, 0.1372549 , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ]])

>>>plt.imshow(x\_train[0],cmap='Greys')

<matplotlib.image.AxesImage at 0x7fe5aec52050>



>>>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\_label[y\_train[i]],y\_train[i]))



>>>x\_train.ndim

3

>>>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

((48000, 28, 28), (12000, 28, 28), (48000,), (12000,))

**(3) Building the CNN model:-**

>>> 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')

])

>>> 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=20,batch\_size=512,verbose=1,validation\_data=(x\_validation,y\_validation))

Epoch 1/20

94/94 [==============================] - 21s 226ms/step - loss: 0.2759 - accuracy: 0.9031 - val\_loss: 0.3023 - val\_accuracy: 0.8959

Epoch 2/20

94/94 [==============================] - 21s 226ms/step - loss: 0.2568 - accuracy: 0.9084 - val\_loss: 0.2889 - val\_accuracy: 0.8982

Epoch 3/20

94/94 [==============================] - 23s 246ms/step - loss: 0.2429 - accuracy: 0.9131 - val\_loss: 0.2825 - val\_accuracy: 0.9037

Epoch 4/20

94/94 [==============================] - 21s 226ms/step - loss: 0.2343 - accuracy: 0.9149 - val\_loss: 0.2766 - val\_accuracy: 0.9022

Epoch 5/20

94/94 [==============================] - 21s 225ms/step - loss: 0.2222 - accuracy: 0.9202 - val\_loss: 0.2703 - val\_accuracy: 0.9040

Epoch 6/20

94/94 [==============================] - 21s 225ms/step - loss: 0.2068 - accuracy: 0.9250 - val\_loss: 0.2635 - val\_accuracy: 0.9090

Epoch 7/20

94/94 [==============================] - 21s 225ms/step - loss: 0.1995 - accuracy: 0.9283 - val\_loss: 0.2679 - val\_accuracy: 0.9068

Epoch 8/20

94/94 [==============================] - 21s 225ms/step - loss: 0.1894 - accuracy: 0.9328 - val\_loss: 0.2625 - val\_accuracy: 0.9093

Epoch 9/20

94/94 [==============================] - 21s 225ms/step - loss: 0.1805 - accuracy: 0.9358 - val\_loss: 0.2582 - val\_accuracy: 0.9118

Epoch 10/20

94/94 [==============================] - 21s 226ms/step - loss: 0.1752 - accuracy: 0.9366 - val\_loss: 0.2609 - val\_accuracy: 0.9116

Epoch 11/20

94/94 [==============================] - 21s 226ms/step - loss: 0.1639 - accuracy: 0.9418 - val\_loss: 0.2555 - val\_accuracy: 0.9108

Epoch 12/20

94/94 [==============================] - 24s 255ms/step - loss: 0.1580 - accuracy: 0.9439 - val\_loss: 0.2581 - val\_accuracy: 0.9120

Epoch 13/20

94/94 [==============================] - 21s 225ms/step - loss: 0.1560 - accuracy: 0.9442 - val\_loss: 0.2592 - val\_accuracy: 0.9146

Epoch 14/20

94/94 [==============================] - 21s 226ms/step - loss: 0.1456 - accuracy: 0.9487 - val\_loss: 0.2574 - val\_accuracy: 0.9122

Epoch 15/20

94/94 [==============================] - 21s 225ms/step - loss: 0.1397 - accuracy: 0.9502 - val\_loss: 0.2661 - val\_accuracy: 0.9103

Epoch 16/20

94/94 [==============================] - 21s 225ms/step - loss: 0.1326 - accuracy: 0.9537 - val\_loss: 0.2568 - val\_accuracy: 0.9144

Epoch 17/20

94/94 [==============================] - 21s 226ms/step - loss: 0.1269 - accuracy: 0.9557 - val\_loss: 0.2686 - val\_accuracy: 0.9133

Epoch 18/20

94/94 [==============================] - 21s 226ms/step - loss: 0.1214 - accuracy: 0.9574 - val\_loss: 0.2698 - val\_accuracy: 0.9105

Epoch 19/20

94/94 [==============================] - 21s 225ms/step - loss: 0.1209 - accuracy: 0.9576 - val\_loss: 0.2632 - val\_accuracy: 0.9152

Epoch 20/20

94/94 [==============================] - 21s 226ms/step - loss: 0.1087 - accuracy: 0.9627 - val\_loss: 0.2702 - val\_accuracy: 0.9111

<keras.callbacks.History at 0x7fe5a3143fd0>

>>>y\_pred=model.predict(x\_test)

>>>y\_pred.round(2)

313/313 [==============================] - 3s 8ms/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. , 0.01, 0. ]], dtype=float32)

>>>y\_test

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

>>>model.evaluate(x\_test,y\_test)

313/313 [==============================] - 3s 9ms/step - loss: 0.2729 - accuracy: 0.9096

[0.2728726267814636, 0.909600019454956]

>>>plt.figure(figsize=(16,16))

j=1

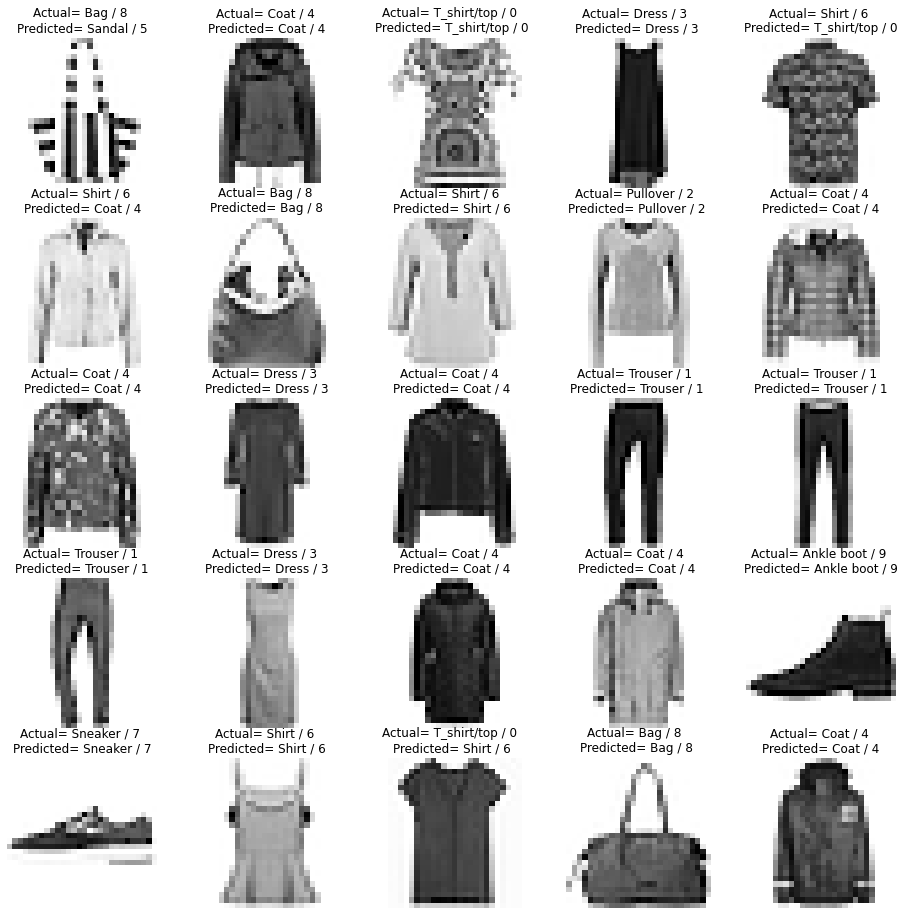
for i in np.random.randint(1000,2000,25):

  plt.subplot(5,5,j);j+=1

  plt.imshow(x\_test[i].reshape(28,28),cmap='Greys')

  plt.axis('off')

  plt.title("Actual= {} / {} \nPredicted= {} / {}".format(class\_label[y\_test[i]],y\_test[i],class\_label[np.argmax(y\_pred[i])],np.argmax(y\_pred[i])))



>>>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\_label, yticklabels=class\_label)

from sklearn.metrics import classification\_report

cr=classification\_report(y\_test,y\_pred\_labels,target\_names=class\_label)

print(cr)

precision recall f1-score support

T\_shirt/top 0.79 0.91 0.85 1000

Trouser 0.99 0.98 0.98 1000

Pullover 0.81 0.91 0.86 1000

Dress 0.92 0.90 0.91 1000

Coat 0.88 0.83 0.86 1000

Sandal 0.97 0.98 0.98 1000

Shirt 0.83 0.68 0.75 1000

Sneaker 0.95 0.97 0.96 1000

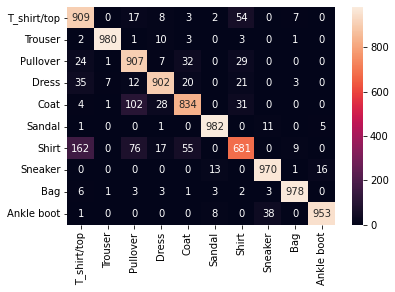
Bag 0.98 0.98 0.98 1000

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\_cnn\_model.h5")

**Building 2 complex CNN model:-**

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')

])

# compile 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')

"""###### very complex model"""

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')

])

# complie 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))

cnn\_model3.save('fashion\_mnist\_cnn\_model3.h5')

cnn\_model3.evaluate(x\_test, y\_test)

# **6.Conclusion:-**

In this tutorial, we learned how to develop a convolutional neural network for clothing classification from scratch.

Specifically, we learned:

* How to develop a test harness to develop a robust evaluation of a model and establish a baseline of performance for a classification task.
* How to explore extensions to a baseline model to improve learning and model capacity.
* How to develop a finalized model, evaluate the performance of the final model, and use it to make predictions on new images.

Deep Learning has wider applications in many industries including the fashion industry. The algorithm could be used to resolve categorization problems often experience in this sector of the economy. Feed-forward NN performs better than Logistic regression in image classification with some tweaks to the algorithm. This requires more training, fine-tuning of hyperparameters and activation functions. Other advanced Deep Learning techniques such as Convolution Neural Network(CNN), Residual Network(ResNET) and Generative Adversarial Network (GAN) give better result.