# **Fashion MNIST Classification**

Fashion MNIST Classification is the image classification dataset and it consists of grayscale images of 10 different types of clothing items.There are 0 to 9 labels and these labels are easily distinguishable datas.I have done this problem in 2 approaches ie, In approach 1 I have used simple feedforward neural network and In approach 2 I have used CNN**.**

**APPROACH 1:**

In the first approach I have used **simple feedforward neural network** (TENSOR FLOW) because the images in the dataset are easily distinguishable,images are grayscale and there are only 10 labels and also it is a basic dataset.So I don't need to implement the complex neural network like CNN.

**In this approach I have got 87 % accuracy.**

**APPROACH 2:**

In the second approach I have used the CNN model because if we add more labels or if the images are RGB then we should use the CNN model because CNN is basically used for image classification,it will capture spatial patterns (like edges, textures, and shapes).Most importantly if the complexity of the images increases then we should use the CNN model.

**In this approach I have got 90 % accuracy.**

**APPROACH 1:**

### **Step by step process of feedforward neural network used in this model:**

1. **Input Layer**:
   * Each image has 28x28 pixels (784 total pixels), where each pixel has a value representing between 0 and 255.The input layer **flattens** the image, this will change the 28x28 grid into single line , so it can be fed into the network.
2. **Hidden Layer**:
   * In this layer the model starts learning patterns in the images.The neural network has **neurons**. Each neuron will try to find the patterns like edges or shapes.
   * The hidden layer uses an **activation function** called **ReLU** (Rectified Linear Unit), this relu function helps the neurons to decide which patterns to focus on.
3. **Output Layer**:
   * The output layer has 10 neurons because there are 10 different categories.
   * For model prediction I have used the **softmax activation function**.This function is mostly used in the output layer .

## Neural network: basic classification using tf.keras

Here I train a neural network model to classify images of clothing, like sneakers and shirts.

### **Step 1: Import Libraries**

* First, I import the libraries:
  + **TensorFlow/Keras**: To build and train the neural network.
  + **NumPy**: For numerical computations.
  + **Matplotlib**: To visualize data

# TensorFlow and tf.keras

import tensorflow as tf

from tensorflow import keras

# Helper libraries

import numpy as np

import matplotlib.pyplot as plt

print(tf.\_\_version\_\_)

2.17.0

### **Step 2: Load the Fashion MNIST Dataset**

* The **Fashion MNIST** dataset contains 70,000 grayscale images of 10 different types of clothing for example T-shirt, pants, etc.
* **Keras** provides function to load the dataset
* I used 60,000 images to train the network and 10,000 images for testing

fashion\_mnist = keras.datasets.fashion\_mnist

(train\_images, train\_labels), (test\_images, test\_labels) = fashion\_mnist.load\_data()

Loading the dataset returns four NumPy arrays:

* The train\_images and train\_labels arrays are the *training set*—the data the model uses to learn.
* The model is tested against the *test set*, the test\_images, and test\_labels arrays.

**The images are 28x28 NumPy arrays, with pixel values are between 0 and 255. The *labels* are ranging from 0 to 9.**

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

Each image is mapped to a single label. Since the *class names* are not included with the dataset.

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

### **Step 3: Exploring the Data**

There are 60,000 images in the training set, with each image represented as 28 x 28 pixels:

train\_images.shape

(60000, 28, 28)

Likewise, there are 60,000 labels in the training set:

len(train\_labels)

60000

There are 10,000 images in the test set. Again, each image is represented as 28 x 28 pixels:

test\_images.shape

(10000, 28, 28)

And the test set contains 10,000 images labels:

len(test\_labels)

10000

### **Step 4: Preprocess the Data**

* Neural networks perform better when the input data is normalized. In this case, pixel values (which range from 0 to 255) are scaled to a range of 0 to 1.

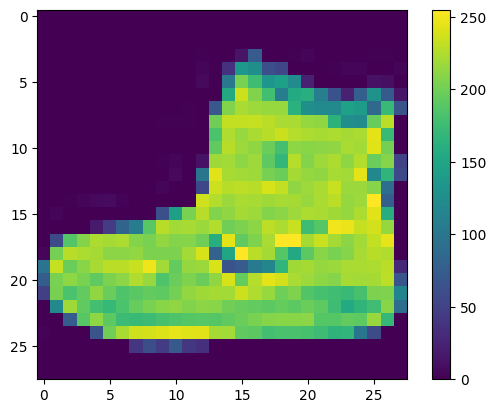
plt.figure()

plt.imshow(train\_images[0])

plt.colorbar()

plt.grid(False)

plt.show()



train\_images = train\_images / 255.0

test\_images = test\_images / 255.0

Display the first 25 images from the *training set* and display the class name below each image.

plt.figure(figsize=(10,10))

for i in range(25):

plt.subplot(5,5,i+1)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.imshow(train\_images[i], cmap=plt.cm.binary)

plt.xlabel(class\_names[train\_labels[i]])

plt.show()



### **Step 4: Build the Neural Network**

* A **simple neural network** is made of layers of neurons that are fully connected (each neuron is connected to every neuron in the previous layer).
* The structure includes:
  1. **Input Layer**: Flattening the 28x28 image into a 1D array of 784 values.
  2. **Hidden Layer(s)**: Fully connected (Dense) layers with an activation function like **ReLU**.
  3. **Output Layer**: A dense layer with 10 neurons (one for each clothing class) and a **softmax** activation function to output probabilities.

model = keras.Sequential([

keras.layers.Flatten(input\_shape=(28, 28)),

keras.layers.Dense(128, activation=tf.nn.relu),

keras.layers.Dense(10, activation=tf.nn.softmax)

])

### **Step 5: Compile the Model**

### **Optimizer**: adam (used for training neural networks).

### **Loss Function**: sparse\_categorical\_crossentropy (used for multiclass classification problem).

### **Metric**: accuracy (used to evaluate the model’s performance).

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

## **Train the model**

model.fit(train\_images, train\_labels, epochs=5)

## **Evaluate accuracy**

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)

print('Test accuracy:', test\_acc)

Test accuracy: 0.8751000165939331

## Make predictions

predictions = model.predict(test\_images)

predictions[0]

np.argmax(predictions[0])

9

test\_labels[0]

9

**Why I have used simple feedforward neural network:**

* The images are grayscale (single colour)
* Data resolution is low (28x28 pixels)
* There are only 10 categories of clothing items (e.g., shirts, shoes, bags). The limited number of classes makes classification easier compared to tasks with many more categories .
* Clothing categories like "T-shirt/top" and "Sneaker" are visually significant, making it easier for even a basic model to distinguish between them.

Alternative for using **simple feedforward neural network ,**we can use CNN because it is specifically designed for image-related tasks.

**Pros and cons for simple feedforward neural network:**

**Pros:**

* It is simple and straightforward, making it quick to set up and train.
* works well for small datasets like Fashion MNIST, where images are low-resolution and contain easily distinguishable objects.
* The model isn't complex,It don't require much computational power for advanced models like CNNs.

**Cons:**

* Feedforward networks don't capture spatial patterns (like edges, textures, and shapes) as well as convolutional neural networks (CNNs).
* This model works for simple datasets, it struggles with higher-resolution dataset like (RGB) images.
* Compared to CNNs, it has lower performance for image classification, especially as the complexity of the images increases.

**APPROACH 2:**

I have used TensorFlow to create and train the Convolutional Neural Network.NumPy is used for numerical computations and imported matplotlib to visualize the dataset.

**# Step 1: Import Libraries**

**import tensorflow as tf**

**from tensorflow import keras**

**import numpy as np**

**import matplotlib.pyplot as plt**

I have loaded the dataset using keras and here I am going to train and evaluate the CNN model. The train\_images and train\_labels are used for training, test\_images and test\_labels are used for model evaluation and prediction.

**# Step 2: Load the Fashion MNIST Dataset**

**fashion\_mnist = keras.datasets.fashion\_mnist**

**(train\_images, train\_labels), (test\_images, test\_labels) = fashion\_mnist.load\_data()**

The pixel values are initially between 0 and 255 to make the data more efficient I normalize the pixel values to 0 and 1.

# Step 3: Preprocess the Data

train\_images = train\_images / 255.0

test\_images = test\_images / 255.0

Here i have done the data preprocessing because CNN expect the input data in a specific shape (height, width, color channels).

train\_images = train\_images.reshape((train\_images.shape[0], 28, 28, 1))

test\_images = test\_images.reshape((test\_images.shape[0], 28, 28, 1))

I have used keras.Sequential to build a sequential model so each layer will add one after the other. Conv2Dis used to create Convolutional layer and it specifies the input shape(28,28,1) which indicates the dimension of image.

MaxPooling2D is used to reduce the spatial dimensions but it remains the most important features.

Flatten is used to reduce the 3D to 1D vector.

The CNN layers (Conv2D + MaxPooling2D) extract spatial features from the images and the dense layers perform the classification task .

# Step 4: Build the CNN Model

model = keras.Sequential([

keras.layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

keras.layers.MaxPooling2D((2, 2)),

keras.layers.Conv2D(64, (3, 3), activation='relu'),

keras.layers.MaxPooling2D((2, 2)),

keras.layers.Conv2D(64, (3, 3), activation='relu'),

keras.layers.Flatten(),

keras.layers.Dense(64, activation='relu'),

keras.layers.Dense(10, activation='softmax')

])

I have used Adam optimizer ,it is used to update the weights during the training and I have used loss function for multi-class classification problems when the labels are integers .

# Step 5: Compile the Model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Step 6: Train the Model

model.fit(train\_images, train\_labels, epochs=5)

# Step 7: Evaluate the Model

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)

print(f'Test accuracy: {test\_acc}')

accuracy: 0.9062 - loss: 0.2716

Test accuracy: 0.9057999849319458

# Step 8: Make Predictions

predictions = model.predict(test\_images)

# Example: Predict the first test image

print(f"Prediction for first test image: {np.argmax(predictions[0])}")

print(f"True label for first test image: {test\_labels[0]}")

Prediction for first test image: 9

True label for first test image: 9