



# Fashion Apparel Classification using CNN

---

## Group members :

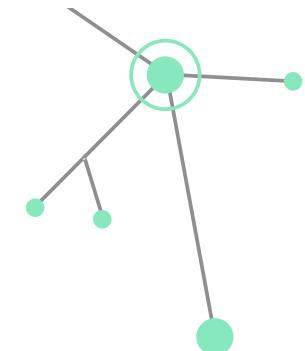
- Youssef ELMOUMEN **FVW745**
- Doha Djellit **VFOAXE**
- Thabet Asmaa A. H. Naima **ZWGD9E**
- Abed Al Hadi Ali **YTUK16**



# Introduction

## **Objective:**

Train a CNN that accurately classifies different types of apparel from the Fashion MNIST dataset.



## **Importance:**

Benefits in inventory management, trend analysis, and enhancing customer experience in the fashion retail industry

## **Dataset Details:**

Source: Zalando Research

Content: 70,000 grayscale images (28x28 pixels) of 10 fashion categories.

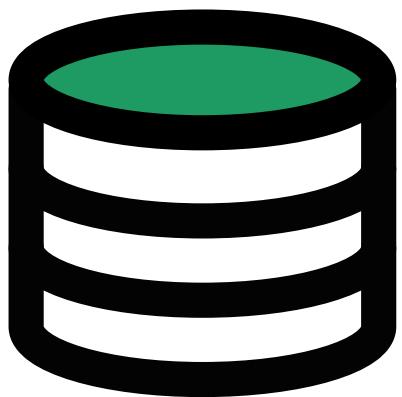
## **Classes:**

T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot



# Table of contents

- 01 Fashion MNIST**
- 02 Data loading**
- 03 Data Visualization & Exploration**
- 04 Data Preprocessing & Image Augmentation**
- 05 Model Testing & Visualization**
- 06 Comparison with Pretrained classifiers**
- 07 Conclusion**

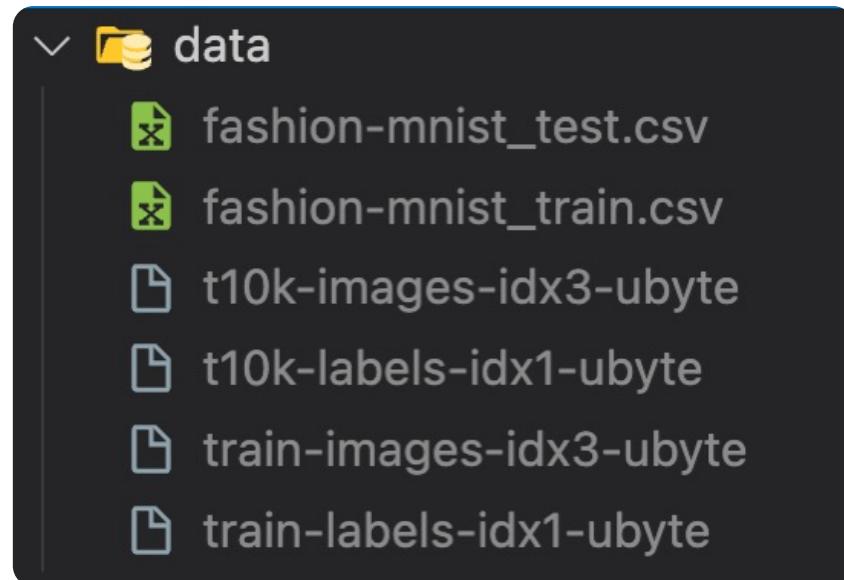


01

# Fashion MNIST Dataset

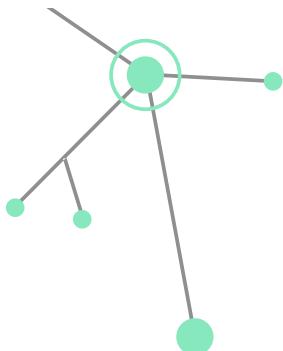
# About the dataset

- Developed by Zalando Research, a branch of Zalando SE, a European e-commerce company.
- Publicly released as an open-source dataset in 2017.
- Collection of 70,000 grayscale images in 10 categories.
- Available in a compressed format with training and test set labels and images.



# Fashion MNIST Properties

An MNIST-like dataset of 70,000 28x28 labeled fashion images



<b>Purpose and Usage:</b>	Designed as a more challenging replacement for the traditional MNIST dataset of handwritten digits. Commonly used for benchmarking machine learning algorithms for image classification.
<b>Dataset Composition</b>	Split into 60,000 training images and 10,000 test images. Uniform distribution of classes (each class has 7,000 images).
<b>Classes in the Dataset</b>	28x28 grayscale image
<b>Classes in the Dataset:</b>	T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot.
<b>Image details</b>	Images are 28x28 pixels, grayscale, with each pixel value ranging from 0 to 255.
<b>Features</b>	785 Feature
<b>Source</b>	<a href="https://www.kaggle.com/datasets/zalando-research/fashionmnist">https://www.kaggle.com/datasets/zalando-research/fashionmnist</a>

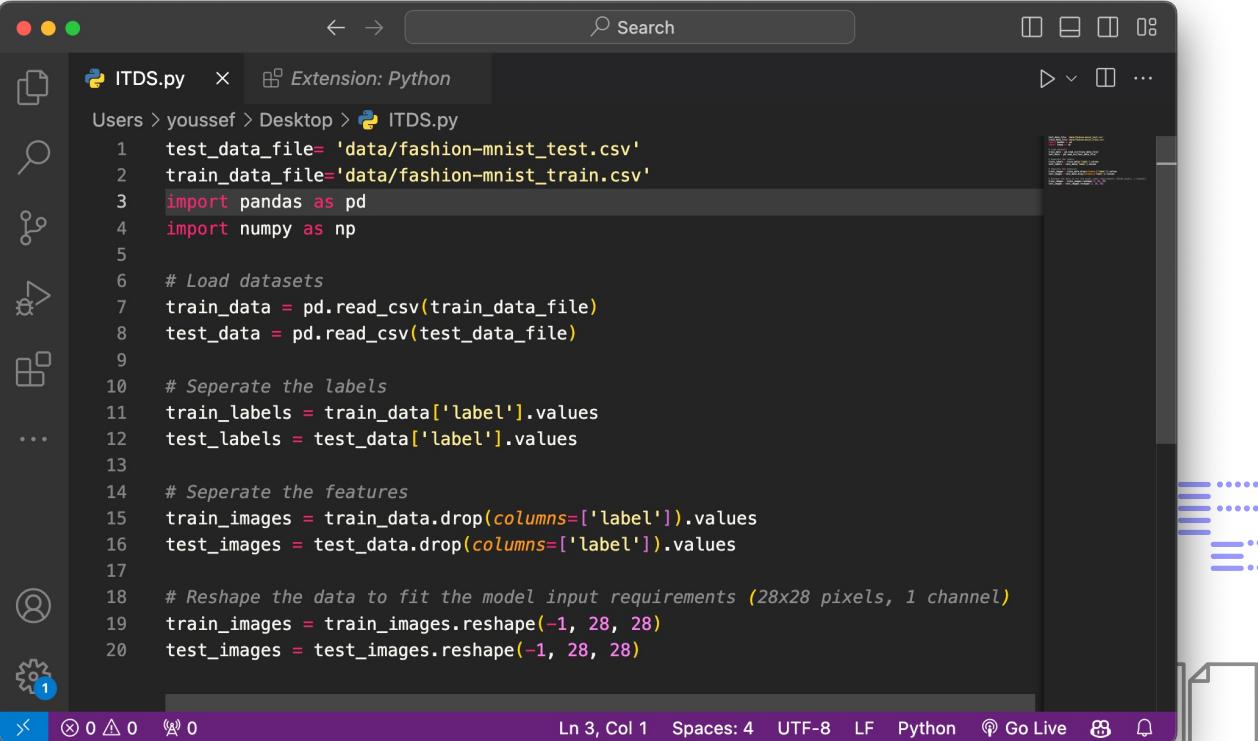


02  
**Data  
Loading**

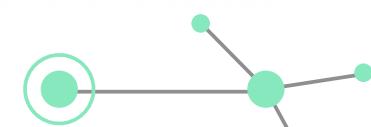
F

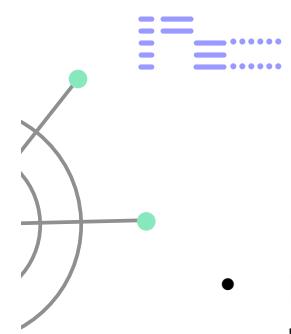
We begin by loading the Fashion MNIST dataset, which consists of 60,000 training images and 10,000 test images. Each image is a 28x28 pixel grayscale representation of a fashion item, categorized into 10 distinct classes. As mentioned before

- Pandas library used For reading CSV files and data manipulation
- NumPy For numerical operations
- and data manipulation

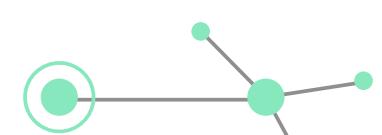


```
ITDS.py  X  Extension: Python
Users > youssef > Desktop > ITDS.py
1 test_data_file= 'data/fashion-mnist_test.csv'
2 train_data_file='data/fashion-mnist_train.csv'
3 import pandas as pd
4 import numpy as np
5
6 # Load datasets
7 train_data = pd.read_csv(train_data_file)
8 test_data = pd.read_csv(test_data_file)
9
10 # Separate the labels
11 train_labels = train_data['label'].values
12 test_labels = test_data['label'].values
13
14 # Separate the features
15 train_images = train_data.drop(columns=['label']).values
16 test_images = test_data.drop(columns=['label']).values
17
18 # Reshape the data to fit the model input requirements (28x28 pixels, 1 channel)
19 train_images = train_images.reshape(-1, 28, 28)
20 test_images = test_images.reshape(-1, 28, 28)
```





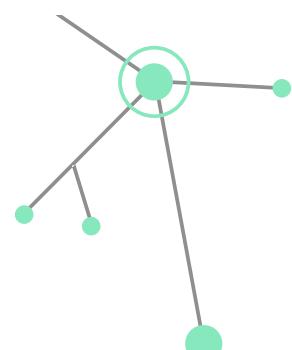
- **Data Sources:** Utilizes 'data/fashion-mnist\_train.csv' for training and 'data/fashion-mnist\_test.csv' for testing.
- **Data Loading:** CSV files are loaded into DataFrame objects for both training and test datasets.
- **Separating Labels and Features:** Labels are extracted into train\_labels and test\_labels, while pixel features are stored in train\_images and test\_images after removing the 'label' column.
- **Reshaping Data:** Images are reshaped to 28x28 pixels to match neural network input requirements, resulting in  $60000 \times 28 \times 28$  dimensions for training images and  $10000 \times 28 \times 28$  for test images.





03  
**Data  
Visualization &  
Exploration**

# Exploring the train data



```
ITDS.py
```

```
Users > youssef > Desktop > ITDS.py
1 train_data.head()
2 print("Shape:\n",train_data.shape, end="\n\n")
3 print("Columns:\n",train_data.columns, end="\n\n")
4 print("Data types:\n",train_data.dtypes, end="\n\n")
5 print("Missing values:\n",train_data.isnull().sum().sum(), end="\n\n")
6 print("Data description:\n",train_data.describe(), end="\n\n")
7 print("Label distribution:\n",train_data['label'].value_counts(),
end="\n\n")
```

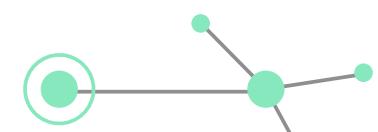
Ln 7, Col 78 Spaces: 4 UTF-8 LF Python ⚡ Go Live 🌐 🔔



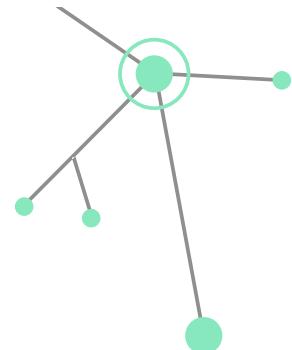
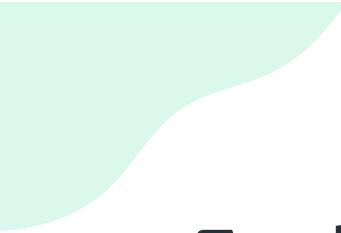
- The dataset consists of 60,000 samples, each with 785 columns.
- The columns include one 'label' column and 784 pixel columns (pixel1 to pixel784), corresponding to the 28x28 pixel values of the images.
- All columns are of integer data type (int64).
- There are no missing values in the dataset.
- The pixel values range from 0 to 255, with most of the pixels having values close to 0, suggesting that many pixels are background.

The screenshot shows a Jupyter Notebook interface with a dark theme. A code cell displays the following output:

```
Shape:  
(60000, 785)  
  
Columns:  
Index(['label', 'pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5', 'pixel6',  
       'pixel7', 'pixel8', 'pixel9',  
       ...,  
       'pixel775', 'pixel776', 'pixel777', 'pixel778', 'pixel779', 'pixel780',  
       'pixel781', 'pixel782', 'pixel783', 'pixel784'],  
      dtype='object', length=785)  
  
Data types:  
label      int64  
pixel1    int64  
pixel2    int64  
pixel3    int64  
pixel4    int64  
...  
pixel780  int64  
pixel781  int64  
pixel782  int64  
pixel783  int64  
pixel784  int64  
Length: 785, dtype: object  
  
Missing values:  
0  
  
Data description:  
label      pixel1      pixel2      pixel3      pixel4 \
```



# Exploring the test data



```
ITDS.py  X
Users > youssef > Desktop > ITDS.py
1  test_data.head()
2  print("Shape:\n",test_data.shape, end="\n\n")
3  print("Columns:\n",test_data.columns, end="\n\n")
4  print("Data types:\n",test_data.dtypes, end="\n\n")
5  print("Missing values:\n",test_data.isnull().sum().sum(), end="\n\n")
6  print("Data description:\n",test_data.describe(), end="\n\n")
7  print("Label distribution:\n",test_data['label'].value_counts(),
end="\n\n")
```

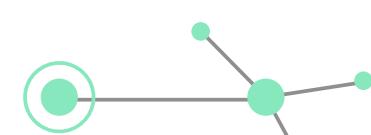
Ln 2, Col 46 Spaces: 4 UTF-8 LF Python ⚡ Go Live 🌐 🔔



- The dataset consists of 10,000 samples, each with 785 columns.
- The columns include one 'label' column and 784 pixel columns (pixel1 to pixel784), corresponding to the 28x28 pixel values of the images.
- All columns are of integer data type (int64).
- There are no missing values in the dataset.
- The pixel values range from 0 to 255, with most of the pixels having values close to 0, suggesting that many pixels are background.

The screenshot shows a Jupyter Notebook interface with a dark theme. The code cell displays the summary statistics for the first 10 columns of the dataset, labeled '0'. The columns are labeled 'label', 'pixel1' through 'pixel9', and 'pixel775' through 'pixel778'. The output shows the count, mean, standard deviation (std), minimum (min), 25th percentile (25%), median (50%), 75th percentile (75%), and maximum (max) for each column. The pixel columns show a range of values from 0 to 255, with most values clustered around 0, indicating background pixels. The 'label' column has a count of 10000, a mean of 4.5, and a standard deviation of approximately 2.87.

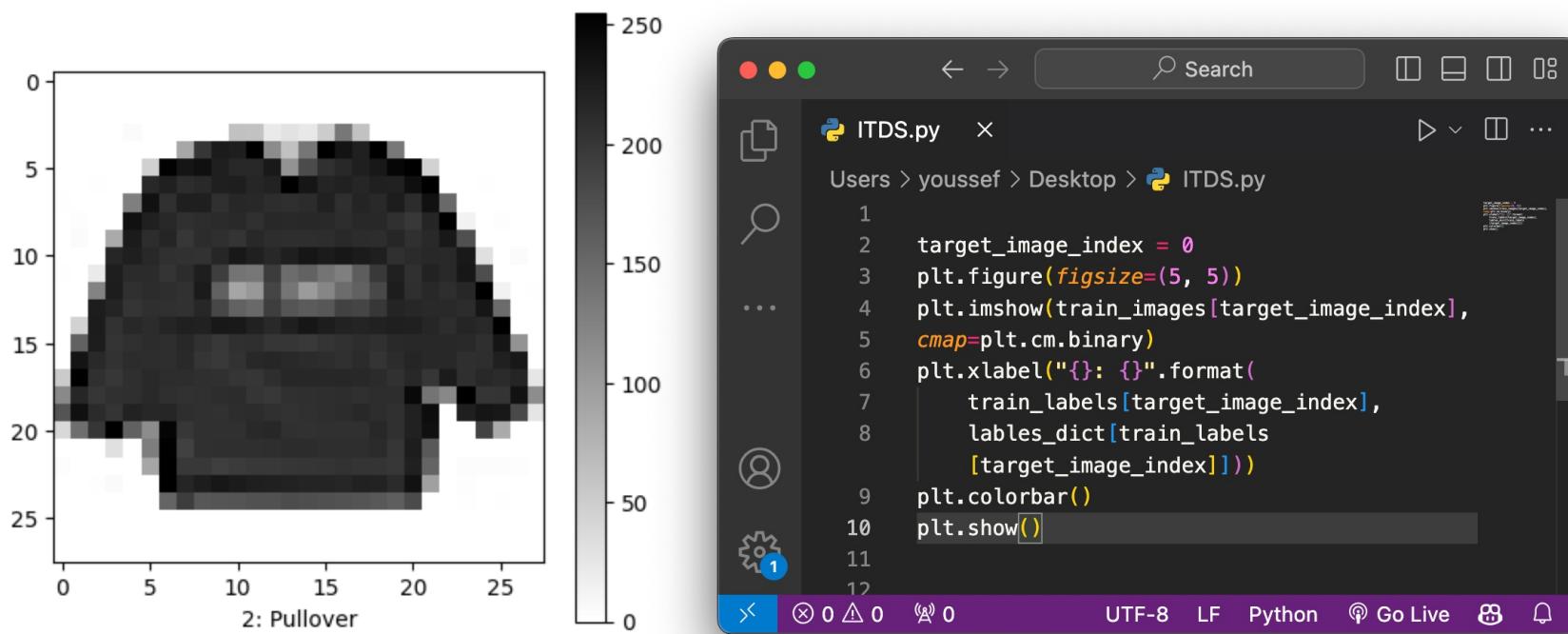
	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel775	pixel776	pixel777	pixel778
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	4.500000	0.000400	0.010300	0.052100	0.077000	0.208882	0.349200	0.826700	2.321200	5.457800	34.320800	23.071900	16.432000	17.870600
std	2.872425	0.024493	0.525187	2.494315	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	57.888679	49.049749	42.159665	44.140552
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	4.500000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	7.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	9.000000	2.000000	45.000000	218.000000	185.000000	227.000000	223.000000	247.000000	218.000000	244.000000	...	...	...	...





To gain insights into the Fashion MNIST dataset, we visualize various apparel classes to understand their distribution and individual image characteristics.  
Matplotlib used For creating visualizations.

Sample representation of the data from the Fashion MNIST Dataset labled

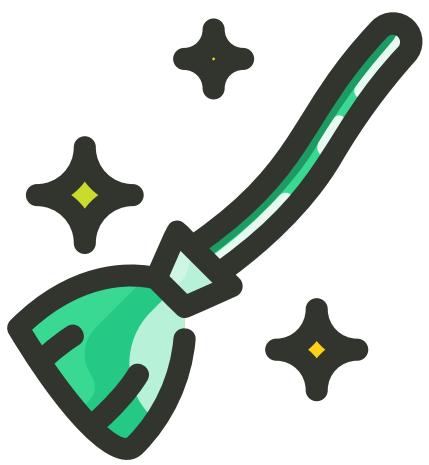


Single image representation with the color bar with its code snippet



Sample Images from the Fashion MNIST Dataset with Labels

- To gain insights into the Fashion MNIST dataset, we visualize
- various apparel classes to
- understand their distribution and
- individual image characteristics.
- Matplotlib used For creating
- visualizations.

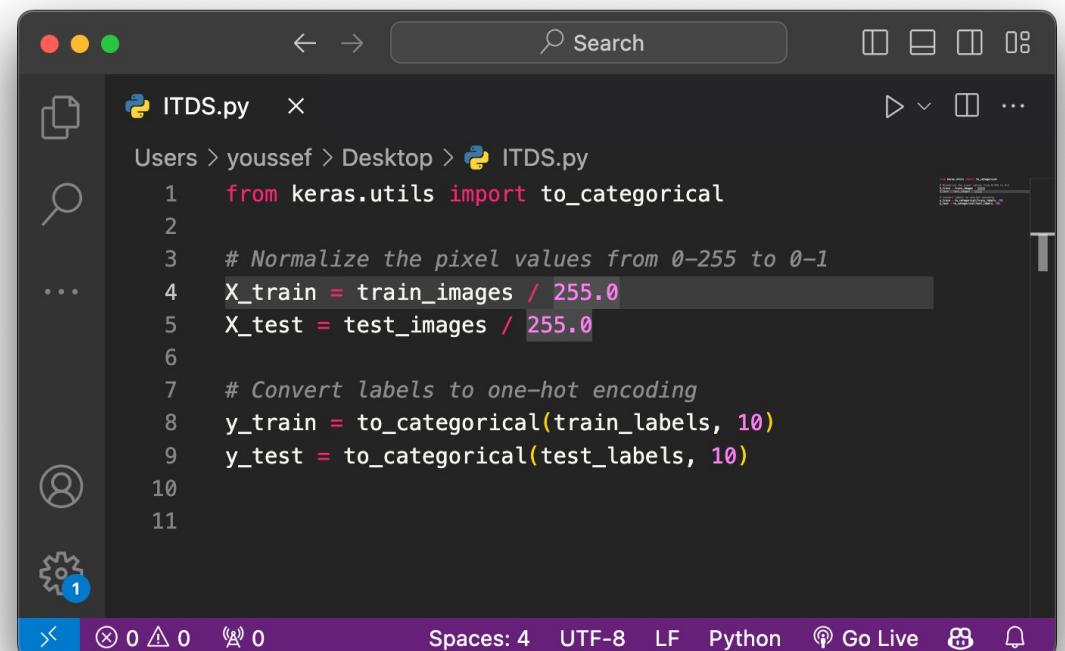


**04**

# **Data Preprocessing**

To prepare the dataset for training,  
We:

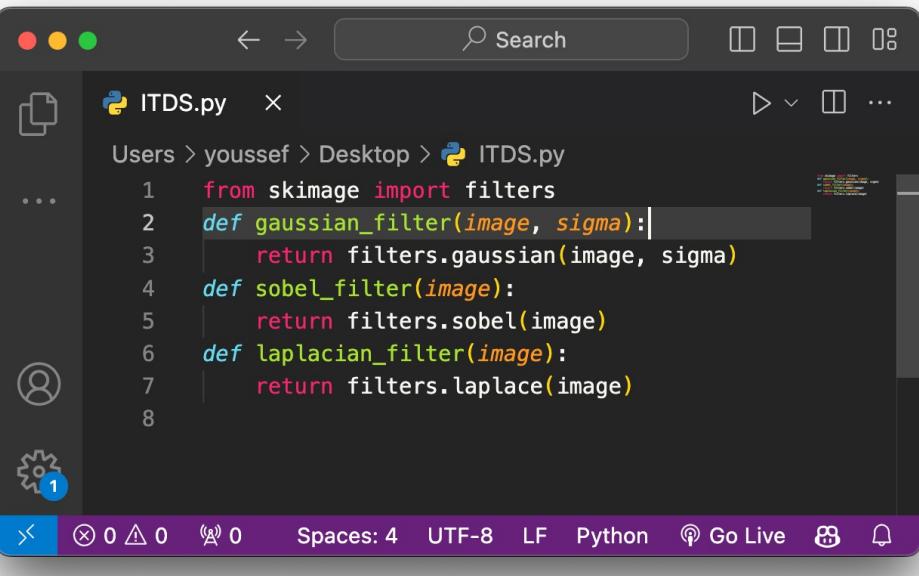
- didn't do any extra transformation as there were no null or missing values.
- normalized the image pixel values
- converted the labels using one Hotshot method



The screenshot shows a code editor window titled "ITDS.py". The file path is "Users > youssef > Desktop > ITDS.py". The code is as follows:

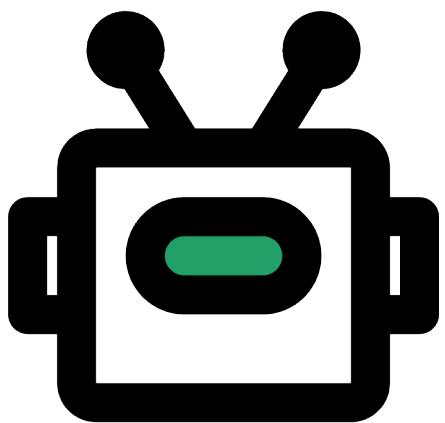
```
1  from keras.utils import to_categorical
2
3  # Normalize the pixel values from 0-255 to 0-1
4  X_train = train_images / 255.0
5  X_test = test_images / 255.0
6
7  # Convert labels to one-hot encoding
8  y_train = to_categorical(train_labels, 10)
9  y_test = to_categorical(test_labels, 10)
10
11
```

The code editor interface includes a sidebar with icons for file, search, and user, and a bottom status bar showing "Spaces: 4" and "Python".



```
ITDS.py
Users > youssef > Desktop > ITDS.py
1  from skimage import filters
2  def gaussian_filter(image, sigma):
3      return filters.gaussian(image, sigma)
4  def sobel_filter(image):
5      return filters.sobel(image)
6  def laplacian_filter(image):
7      return filters.laplace(image)
8
```

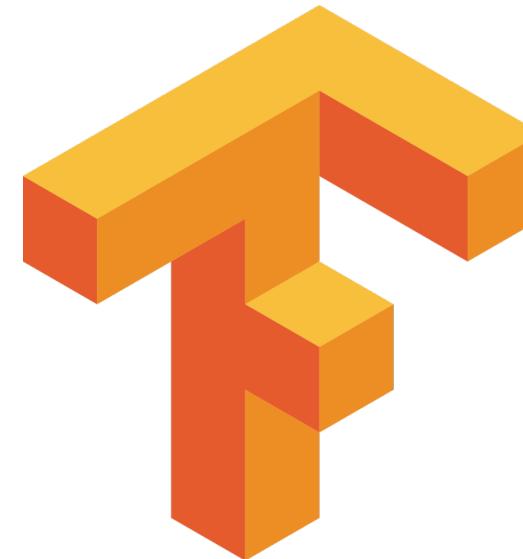
We also tested various filters, including Laplacian, Gaussian, and Sobel, to enhance the image clarity. However, we observed that the model achieved higher accuracy without the application of these filters.



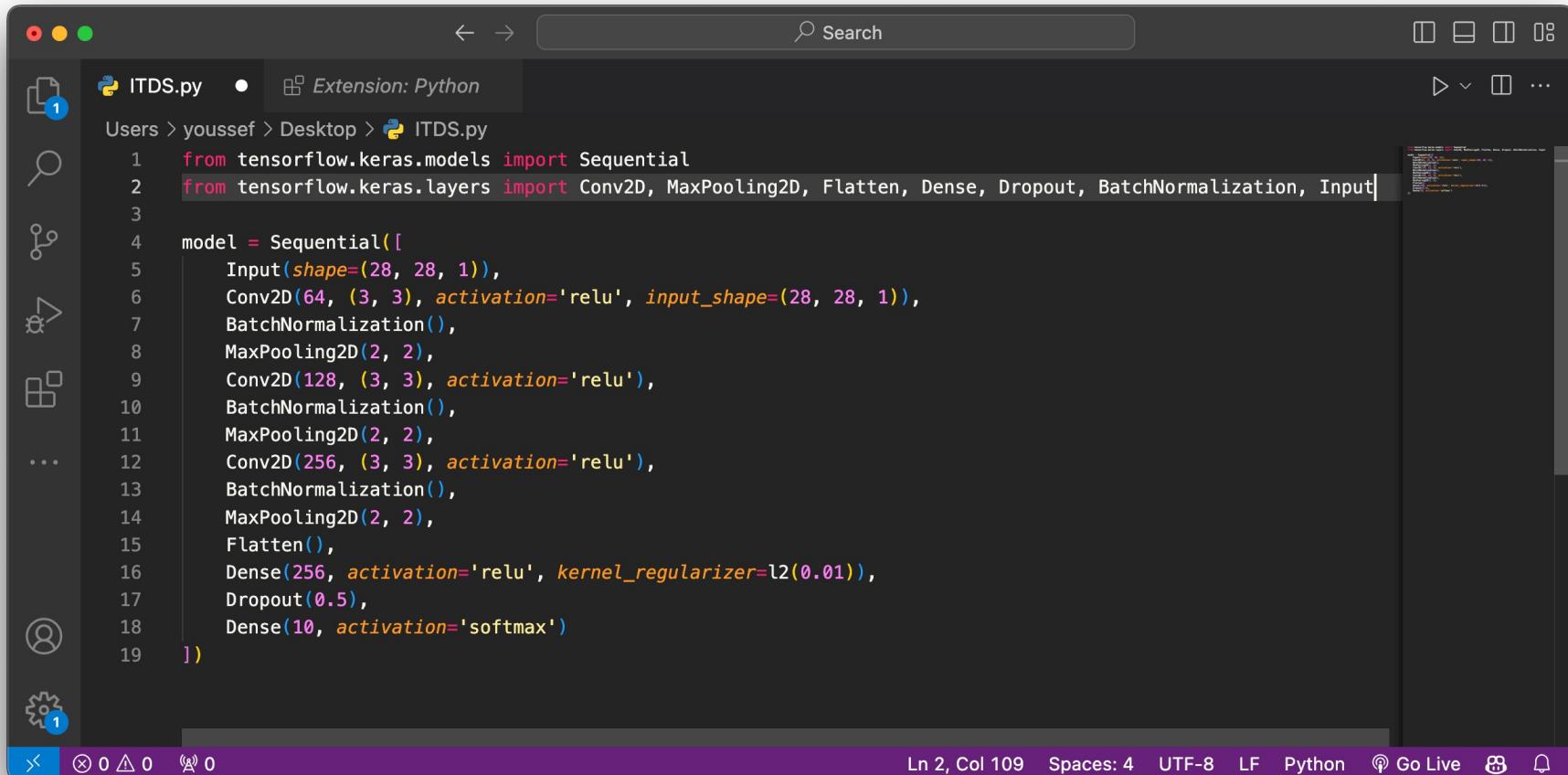
**05**  
**Model**  
**Training**

# Introduction

- The framework used in this project is : TensorFlow
- A Convolutional Neural Network (CNN) was designed to classify the fashion apparel images in the Fashion MNIST dataset.
- The architecture was refined through experimentation with various layers and hyperparameters.



# CNN Model Development



A screenshot of a Python code editor showing a file named ITDS.py. The code defines a Sequential model for image processing, utilizing various layers like Conv2D, MaxPooling2D, Flatten, Dense, and Dropout. The code is written in Python using the TensorFlow Keras API.

```
1  from tensorflow.keras.models import Sequential
2  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, Input
3
4  model = Sequential([
5      Input(shape=(28, 28, 1)),
6      Conv2D(64, (3, 3), activation='relu', input_shape=(28, 28, 1)),
7      BatchNormalization(),
8      MaxPooling2D(2, 2),
9      Conv2D(128, (3, 3), activation='relu'),
10     BatchNormalization(),
11     MaxPooling2D(2, 2),
12     Conv2D(256, (3, 3), activation='relu'),
13     BatchNormalization(),
14     MaxPooling2D(2, 2),
15     Flatten(),
16     Dense(256, activation='relu', kernel_regularizer=l2(0.01)),
17     Dropout(0.5),
18     Dense(10, activation='softmax')
19 ])
```



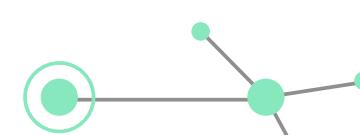
# CNN Model Development

## Overview

- Utilizes a Sequential model from TensorFlow's Keras API.
- Receives input images of size 28x28 pixels, with 1 channel (grayscale).

## Layers

- **Input Layer:**  
`Input(shape=(28, 28, 1))`: Defines the model to receive input images of size 28x28 pixels, with 1 channel (grayscale).





# CNN Model Development

- **First Convolutional Layer:**

64 filters of size 3x3.

Activation function: ReLU (Rectified Linear Unit).

Followed by Batch Normalization and Max Pooling (2x2).

- **Second Convolutional Layer:**

128 filters of size 3x3.

Activation function: ReLU.

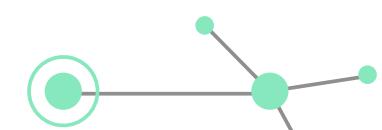
Includes Batch Normalization and Max Pooling (2x2)

- **Third Convolutional Layer:**

256 filters of size 3x3.

Activation function: ReLU.

Includes Batch Normalization and Max Pooling (2x2).





# CNN Model Development

- **Flattening Layer:**

Converts the 2D feature maps into a 1D feature vector.

- **Dense Layer:**

256 neurons.

Activation function: ReLU.

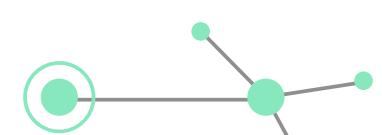
Includes L2 regularization (weight decay coefficient of 0.01) to combat overfitting.

Dropout layer with a rate of 0.5 to prevent overfitting.

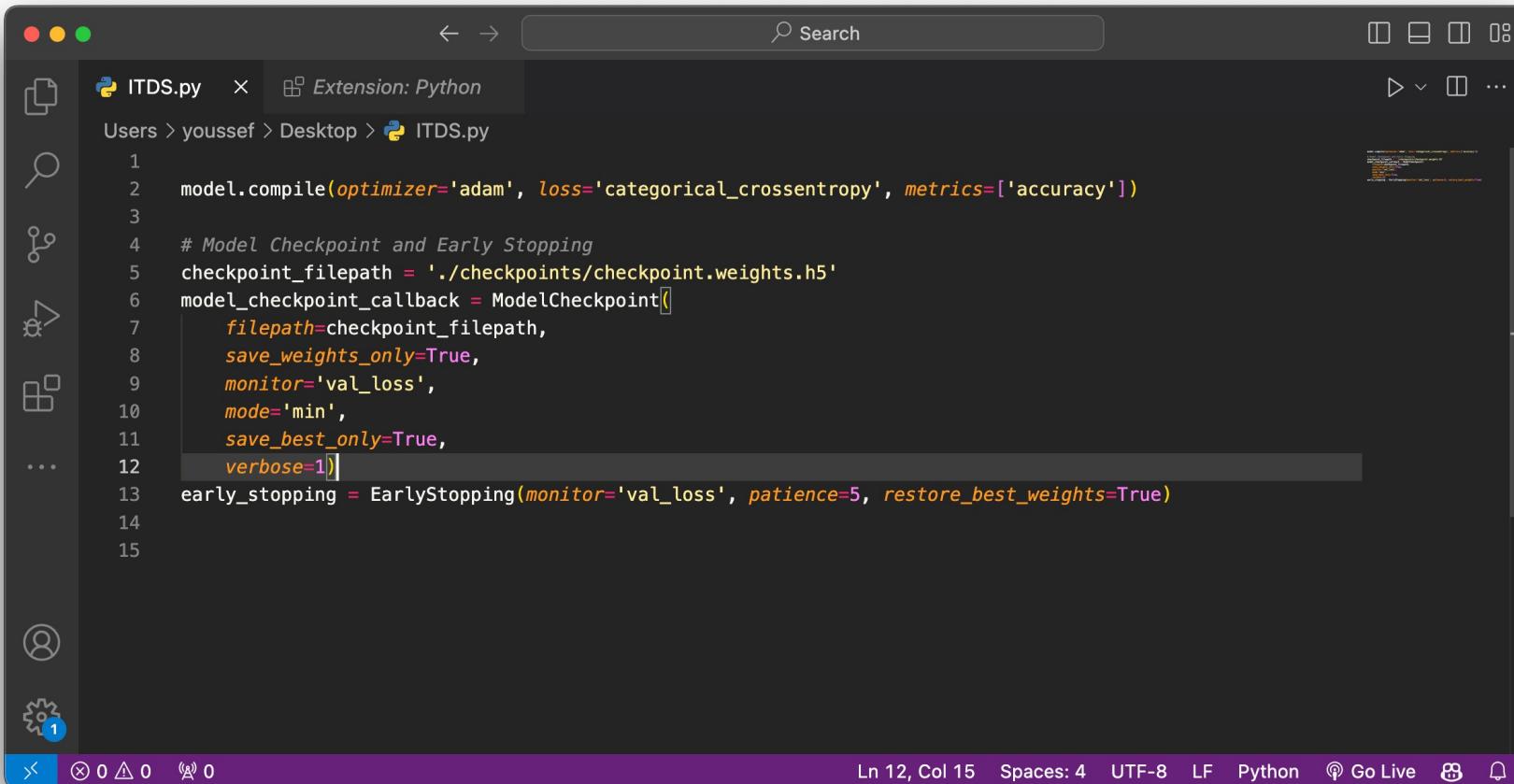
- **Output Layer:**

10 neurons (one for each clothing category).

Activation function: Softmax for multi-class classification.



# CNN Model Development



```
ITDS.py  Extension: Python
Users > youssef > Desktop > ITDS.py
1
2     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
3
4     # Model Checkpoint and Early Stopping
5     checkpoint_filepath = './checkpoints/checkpoint.weights.h5'
6     model_checkpoint_callback = ModelCheckpoint(
7         filepath=checkpoint_filepath,
8         save_weights_only=True,
9         monitor='val_loss',
10        mode='min',
11        save_best_only=True,
12        verbose=1)
13     early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
14
15
```

# CNN Model Development

- **Compilation of the Model:**

Optimizer: Adam.

Loss function: Categorical Crossentropy.

Evaluation metric: Accuracy.

- **Training Process:**

Early stopping to terminate training when validation loss decreases, to prevent overfitting.

Model checkpointing to save the model with the best validation loss.

# Training and Results

The screenshot shows a Jupyter Notebook interface with the following details:

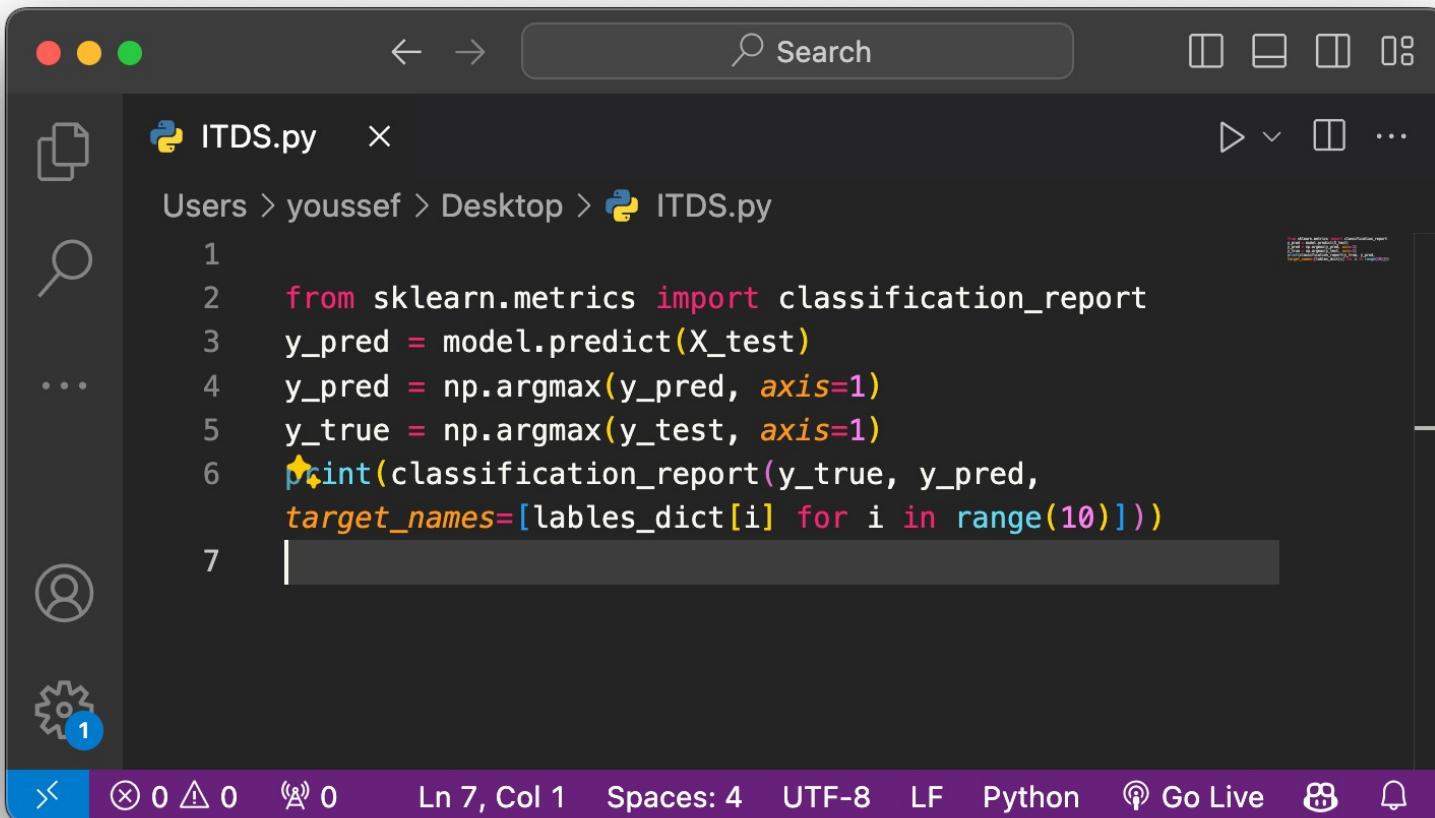
- File Bar:** DS\_Fashion\_Apparel\_Classification\_latest.ipynb M X, README.md
- Toolbar:** Code, Markdown, Run All, Restart, Clear All Outputs, Variables, Outline, datagen, Aa, ab, \* Y, 1 of 2, up, down, close.
- Left Sidebar:** Includes icons for file operations (New, Open, Save, etc.), search, cell selection (3), cell creation (4), cell execution count (5), cell output count (6), and a gear icon (7).
- Main Area:** Displays training logs for a Keras model. The logs show epochs from 1 to 10, with each epoch consisting of multiple steps. Metrics include accuracy, loss, and validation accuracy/loss. Checkpoints are saved at regular intervals. The final test accuracy is reported as 0.903.
- Bottom Bar:** Includes icons for main, file, cell, search, spaces, cell number, go live, prettier, and a settings gear.

```
... Epoch 1/10
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: super().__init__(activity_regularizer=activity_regularizer, **kwargs)
  1874/1875    0s 26ms/step - accuracy: 0.7724 - loss: 1.6123
Epoch 1: val_loss improved from inf to 0.53353, saving model to ./checkpoints/checkpoint.weights.h5
1875/1875    53s 28ms/step - accuracy: 0.7725 - loss: 1.6115 - val_accuracy: 0.8282 - val_loss: 0.5335
Epoch 2/10
1874/1875    0s 27ms/step - accuracy: 0.8666 - loss: 0.4568
Epoch 2: val_loss improved from 0.53353 to 0.39204, saving model to ./checkpoints/checkpoint.weights.h5
1875/1875    53s 28ms/step - accuracy: 0.8666 - loss: 0.4568 - val_accuracy: 0.8887 - val_loss: 0.3920
Epoch 3/10
1875/1875    0s 26ms/step - accuracy: 0.8837 - loss: 0.4033
Epoch 3: val_loss improved from 0.39204 to 0.39066, saving model to ./checkpoints/checkpoint.weights.h5
1875/1875    51s 27ms/step - accuracy: 0.8837 - loss: 0.4033 - val_accuracy: 0.8863 - val_loss: 0.3907
Epoch 4/10
1873/1875    0s 25ms/step - accuracy: 0.8968 - loss: 0.3662
Epoch 4: val_loss did not improve from 0.39066
1875/1875    50s 27ms/step - accuracy: 0.8968 - loss: 0.3663 - val_accuracy: 0.8719 - val_loss: 0.4445
Epoch 5/10
1873/1875    0s 27ms/step - accuracy: 0.9022 - loss: 0.3521
Epoch 5: val_loss did not improve from 0.39066
1875/1875    52s 28ms/step - accuracy: 0.9022 - loss: 0.3521 - val_accuracy: 0.8849 - val_loss: 0.4111
Epoch 6/10
1874/1875    0s 26ms/step - accuracy: 0.9136 - loss: 0.3229
Epoch 6: val_loss improved from 0.39066 to 0.34343, saving model to ./checkpoints/checkpoint.weights.h5
1875/1875    51s 27ms/step - accuracy: 0.9136 - loss: 0.3229 - val_accuracy: 0.9026 - val_loss: 0.3434
Epoch 7/10
1875/1875    0s 26ms/step - accuracy: 0.9240 - loss: 0.2934
...
Epoch 10: val_loss did not improve from 0.34343
1875/1875    50s 27ms/step - accuracy: 0.9369 - loss: 0.2519 - val_accuracy: 0.8941 - val_loss: 0.3795
313/313    2s 7ms/step - accuracy: 0.8976 - loss: 0.3522
Test accuracy: 0.903, loss: 0.343
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

## Training and Results

- Trained over 10 epochs with initial accuracy of 77.25% and loss of 1.6115.
- Achieved peak training accuracy of 93.69% with a loss reduced to 0.2519 by the final epoch.
- Best validation loss recorded at 0.34343 during Epoch 6, with the highest validation accuracy reaching 90.26% by the end of training.
- Final test performance demonstrated a robust accuracy of 90.3% and a loss of 0.343.

# Performance Evaluation - Code

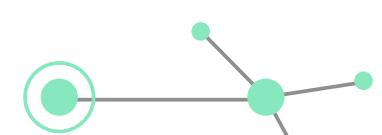


```
1 from sklearn.metrics import classification_report
2 y_pred = model.predict(X_test)
3 y_pred = np.argmax(y_pred, axis=1)
4 y_true = np.argmax(y_test, axis=1)
5 print(classification_report(y_true, y_pred,
6 target_names=[labels_dict[i] for i in range(10)]))
7
```



# Performance Evaluation - Report

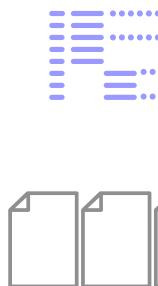
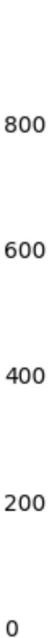
	precision	recall	f1-score	support
T-shirt/top	0.87	0.84	0.86	1000
Trouser	1.00	0.98	0.99	1000
Pullover	0.82	0.89	0.85	1000
Dress	0.88	0.91	0.89	1000
Coat	0.86	0.81	0.83	1000
Sandal	0.99	0.96	0.97	1000
Shirt	0.74	0.71	0.73	1000
Sneaker	0.94	0.96	0.95	1000
Bag	0.96	0.98	0.97	1000
Ankle boot	0.96	0.96	0.96	1000
accuracy			0.90	10000
macro avg	0.90	0.90	0.90	10000
weighted avg	0.90	0.90	0.90	10000



# Performance Evaluation

## Confusion Matrix

		Confusion matrix									
		T-shirt/top -	Trouser -	Pullover -	Dress -	Coat -	Sandal -	Shirt -	Sneaker -	Bag -	Ankle boot -
Actual	T-shirt/top -	844	1	25	37	0	0	81	0	12	0
	Trouser -	0	978	5	15	1	0	1	0	0	0
	Pullover -	9	0	887	10	36	0	53	0	5	0
	Dress -	14	1	16	913	35	0	19	0	1	1
	Coat -	0	0	86	15	810	1	86	0	2	0
	Sandal -	1	0	0	0	0	961	1	29	4	4
	Shirt -	95	2	60	52	63	0	711	0	17	0
	Sneaker -	0	0	0	0	0	4	0	957	2	37
	Bag -	2	0	4	0	2	1	6	0	985	0
	Ankle boot -	0	0	0	0	0	7	0	32	0	961





Per



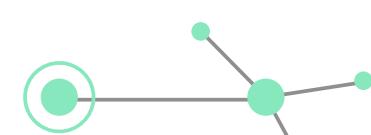
## Performance Evaluation - Analysis

**Overall Accuracy:** The model achieved a high overall accuracy of 90%, showing it performs well across all categories.

**Top Performers:** The Trouser category excelled with perfect precision of 100% and very high recall of 98%. The Sandal, Bag, and Ankle boot categories also performed excellently, with both precision and recall over 95%.

**Areas for Improvement:** The Shirt category had the weakest results, with a precision of 74% and a recall of 71%. This indicates a need for further adjustments to the model to improve its ability to correctly identify shirts.

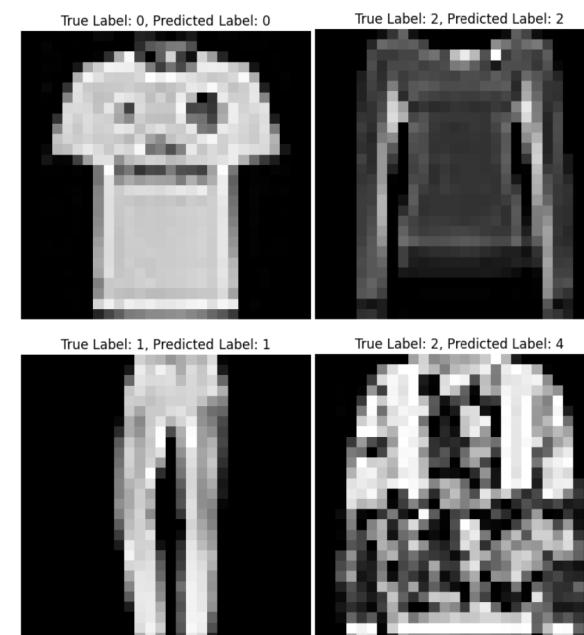
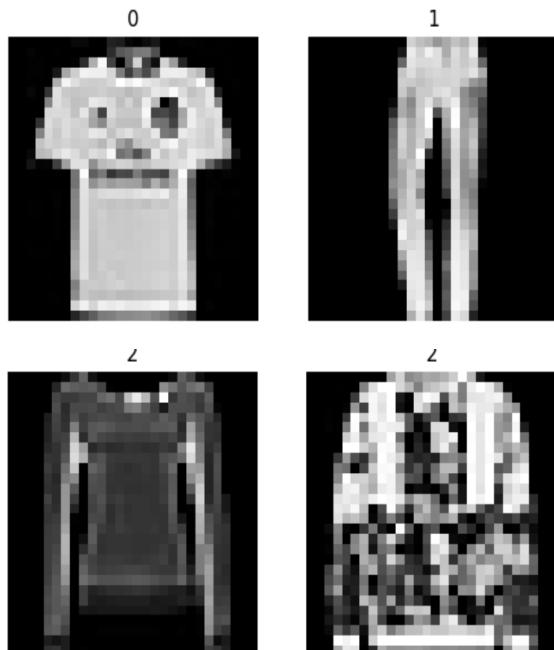
**Balanced Metrics:** Categories like Dress, Sandal, and Sneaker showed a good balance between precision and recall, leading to high f1-scores, which measure the accuracy of the model's predictions.





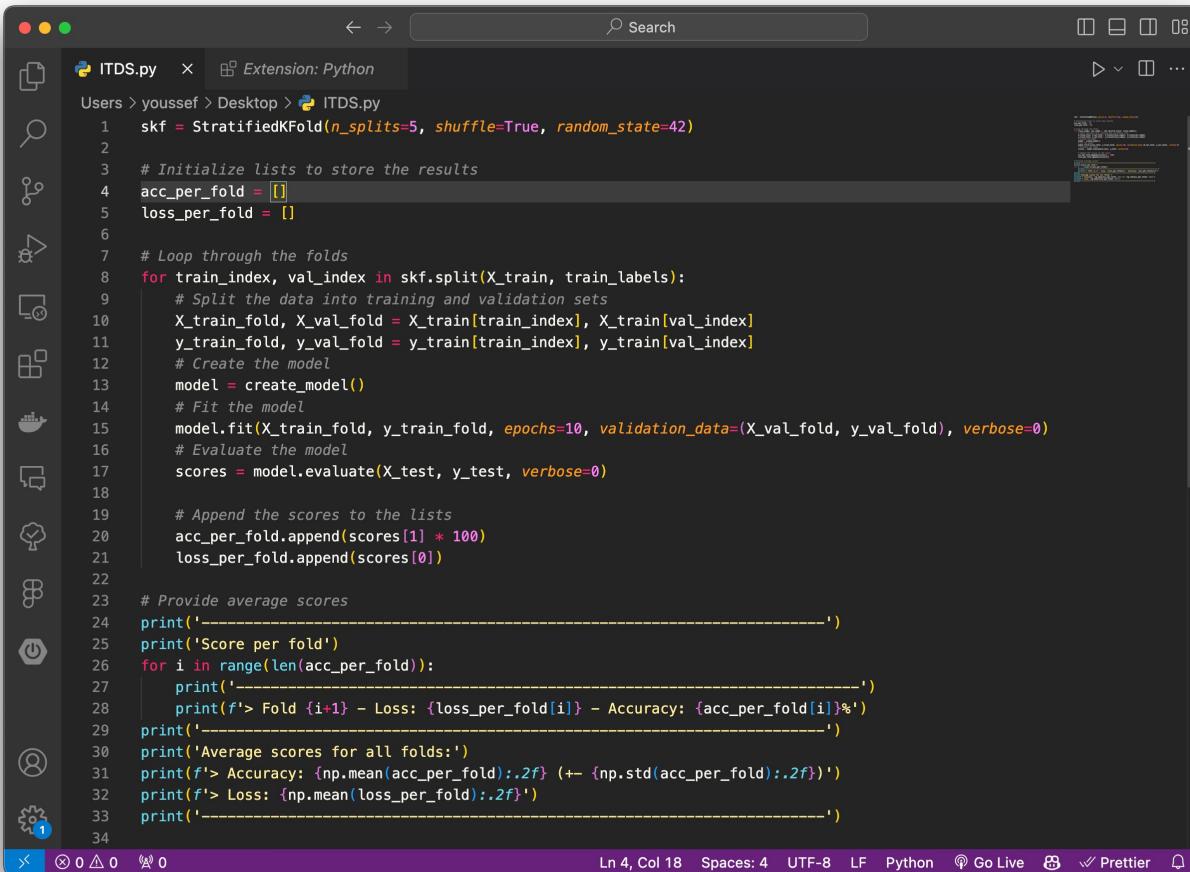
# Model Testing - Prediction results

The CNN model demonstrates high accuracy in classifying fashion items, with a few misclassifications

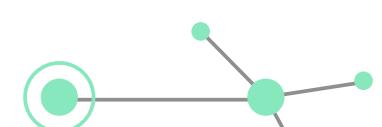


True vs. Predicted Labels

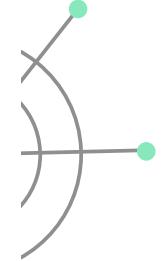
# Model Testing – Cross Validation



```
ITDS.py  Extension: Python
Users > youssef > Desktop > ITDS.py
1  skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
2
3  # Initialize lists to store the results
4  acc_per_fold = []
5  loss_per_fold = []
6
7  # Loop through the folds
8  for train_index, val_index in skf.split(X_train, train_labels):
9      # Split the data into training and validation sets
10     X_train_fold, X_val_fold = X_train[train_index], X_train[val_index]
11     y_train_fold, y_val_fold = y_train[train_index], y_train[val_index]
12
13     # Create the model
14     model = create_model()
15
16     # Fit the model
17     model.fit(X_train_fold, y_train_fold, epochs=10, validation_data=(X_val_fold, y_val_fold), verbose=0)
18     # Evaluate the model
19     scores = model.evaluate(X_test, y_test, verbose=0)
20
21     # Append the scores to the lists
22     acc_per_fold.append(scores[1] * 100)
23     loss_per_fold.append(scores[0])
24
25     # Provide average scores
26     print('-----')
27     print('Score per fold')
28     for i in range(len(acc_per_fold)):
29         print(f'> Fold {i+1} - Loss: {loss_per_fold[i]} - Accuracy: {acc_per_fold[i]}%')
30     print('-----')
31     print('Average scores for all folds:')
32     print(f'> Accuracy: {np.mean(acc_per_fold):.2f} (+- {np.std(acc_per_fold):.2f})')
33     print(f'> Loss: {np.mean(loss_per_fold):.2f}')
34
-----
```



PE



The following are the result of the cross validation of our CNN

Score per fold

> Fold 1 - Loss: 0.4180976450443268 - Accuracy: 88.99999856948853%

> Fold 2 - Loss: 0.41137993335723877 - Accuracy: 88.88000249862671%

> Fold 3 - Loss: 0.5078486800193787 - Accuracy: 86.83000206947327%

> Fold 4 - Loss: 0.45873013138771057 - Accuracy: 88.30000162124634%

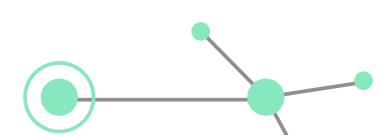
> Fold 5 - Loss: 0.44572535157203674 - Accuracy: 88.26000094413757%

Average scores for all folds:

> Accuracy: 88.25 (+- 0.77)

> Loss: 0.45

PE





**06**

## **Comparison with other Classifiers**

PE

# SVC Classifier

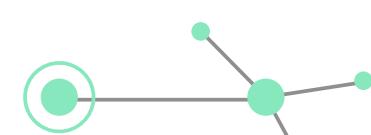
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000 entries, 0 to 59999
Columns: 785 entries, label to pixel784
dtypes: int64(785)
memory usage: 359.3 MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Columns: 785 entries, label to pixel784
dtypes: int64(785)
memory usage: 59.9 MB
None
Evaluation Metrics for SVC Classifier:
-----
Accuracy: 0.8921
Balanced Accuracy: 0.8920999999999999
F1 Score: 0.8915003945780919
Log Loss: 0.29818199856446
```

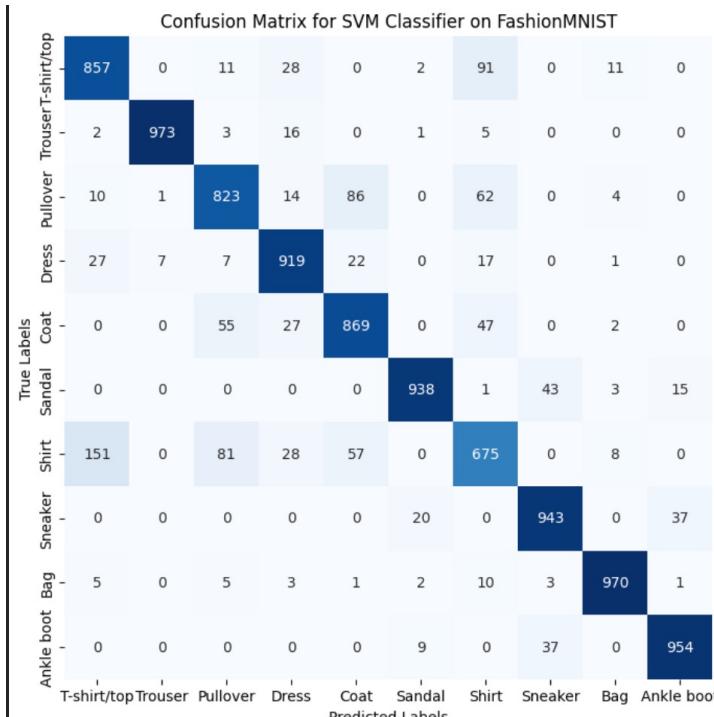
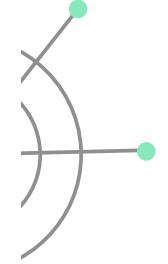
Training Metrics

	precision	recall	f1-score	support
T-shirt/top	0.81	0.86	0.84	1000
Trouser	0.99	0.97	0.98	1000
Pullover	0.84	0.82	0.83	1000
Dress	0.89	0.92	0.90	1000
Coat	0.84	0.87	0.85	1000
Sandal	0.97	0.94	0.95	1000
Shirt	0.74	0.68	0.71	1000
Sneaker	0.92	0.94	0.93	1000
Bag	0.97	0.97	0.97	1000
Ankle boot	0.95	0.95	0.95	1000
accuracy			0.89	10000
macro avg	0.89	0.89	0.89	10000
weighted avg	0.89	0.89	0.89	10000

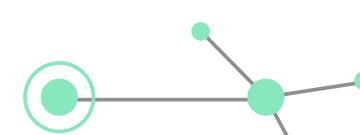
Classification report

PE





Confusion Matrix



# K-Neighbors Classifier

	precision	recall	f1-score	support
T-shirt/top	0.77	0.87	0.82	1000
Trouser	0.99	0.96	0.98	1000
Pullover	0.75	0.81	0.78	1000
Dress	0.91	0.88	0.90	1000
Coat	0.79	0.80	0.79	1000
Sandal	1.00	0.82	0.90	1000
Shirt	0.68	0.58	0.63	1000
Sneaker	0.87	0.94	0.91	1000
Bag	0.98	0.95	0.97	1000
Ankle boot	0.88	0.96	0.92	1000
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

Classification report

Evaluation Metrics for KNeighbors Classifier:

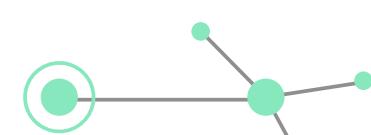
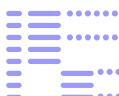
Accuracy: 0.8589

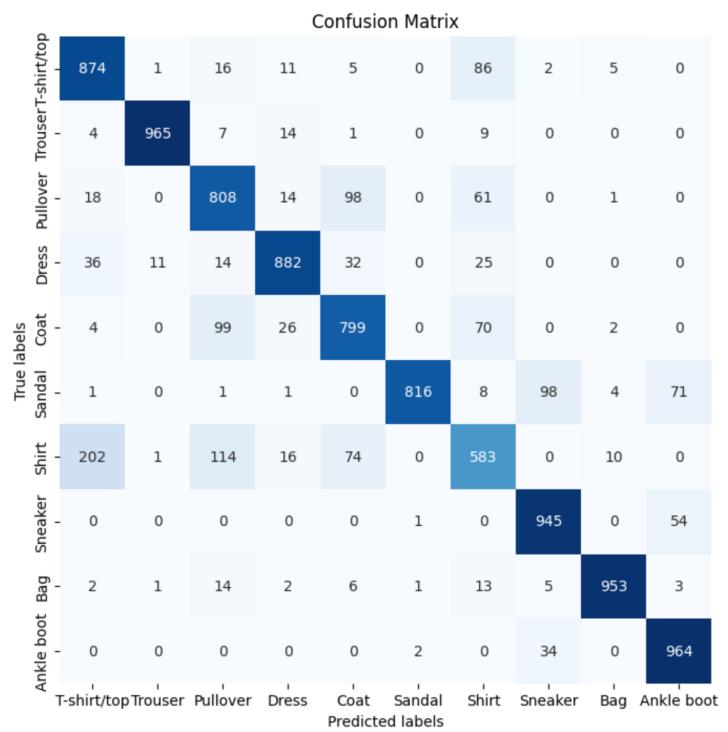
Balanced Accuracy: 0.8589

F1 Score: 0.8580386825779729

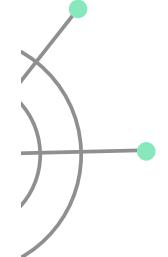
Log Loss: 1.749503487525264

Training Metrics





F



# Gaussian Naive Bayes Classifier

	precision	recall	f1-score	support
T-shirt/top	0.83	0.59	0.69	1000
Trouser	0.69	0.94	0.79	1000
Pullover	0.61	0.32	0.42	1000
Dress	0.46	0.64	0.53	1000
Coat	0.38	0.77	0.51	1000
Sandal	0.89	0.29	0.44	1000
Shirt	0.32	0.04	0.07	1000
Sneaker	0.50	0.98	0.66	1000
Bag	0.83	0.71	0.77	1000
Ankle boot	0.92	0.63	0.75	1000
accuracy			0.59	10000
macro avg	0.64	0.59	0.56	10000
weighted avg	0.64	0.59	0.56	10000

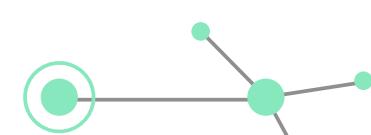
Classification report

Evaluation Metrics for Gaussian Naive Bayes Classifier:

-----  
Accuracy: 0.5914  
Balanced Accuracy: 0.5914  
F1 Score: 0.5627387972343205  
Log Loss: 14.219901360032864

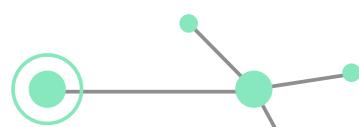
Training Metrics

F



## Confusion Matrix

		Confusion Matrix for GaussianNB on FashionMNIST									
		T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
Actual	T-shirt/top	594	42	25	196	87	0	16	0	40	0
	Trouser	0	940	16	33	2	1	8	0	0	0
	Pullover	4	7	324	68	553	0	21	0	23	0
	Dress	6	311	6	644	25	0	5	0	3	0
	Coat	0	33	37	155	765	0	1	0	9	0
	Sandal	1	1	1	2	0	289	6	648	14	38
	Shirt	113	32	103	236	432	0	38	0	46	0
	Sneaker	0	0	0	0	0	3	0	978	0	19
	Bag	1	2	17	79	161	3	22	3	711	1
	Ankle boot	0	0	0	1	0	27	2	328	11	631



F

# Logistic Regression

	precision	recall	f1-score	support
T-shirt/top	0.81	0.82	0.81	1000
Trouser	0.95	0.98	0.96	1000
Pullover	0.77	0.76	0.77	1000
Dress	0.88	0.87	0.87	1000
Coat	0.75	0.82	0.79	1000
Sandal	0.95	0.88	0.91	1000
Shirt	0.68	0.58	0.63	1000
Sneaker	0.90	0.93	0.92	1000
Bag	0.93	0.95	0.94	1000
Ankle boot	0.92	0.95	0.93	1000
accuracy			0.85	10000
macro avg	0.85	0.85	0.85	10000
weighted avg	0.85	0.85	0.85	10000

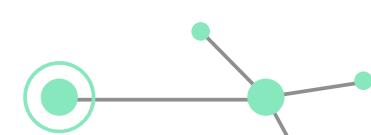
Classification report

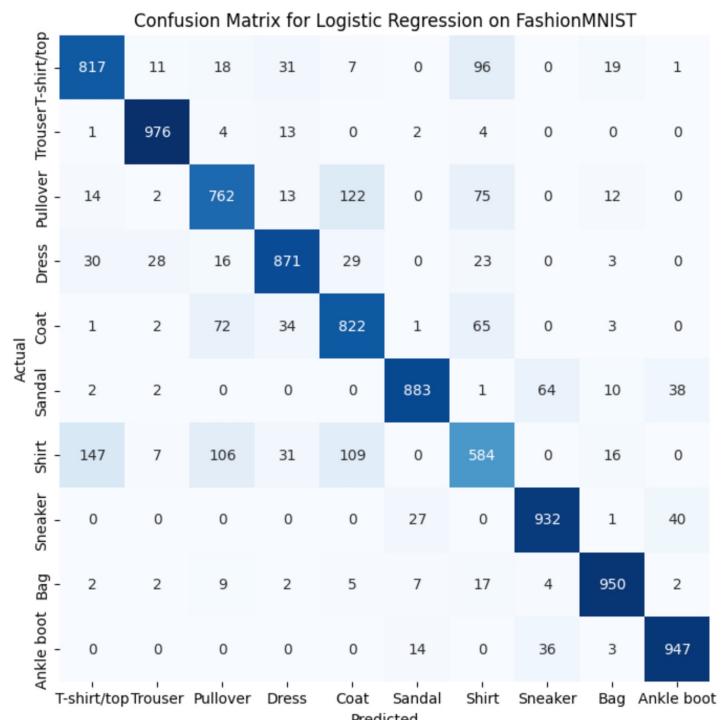
Evaluation Metrics for Logistic Regression Classifier:

Accuracy: 0.8544  
Balanced Accuracy: 0.8544  
F1 Score: 0.8529011356168574  
Log Loss: 0.44334444146632257

Training Metrics

F





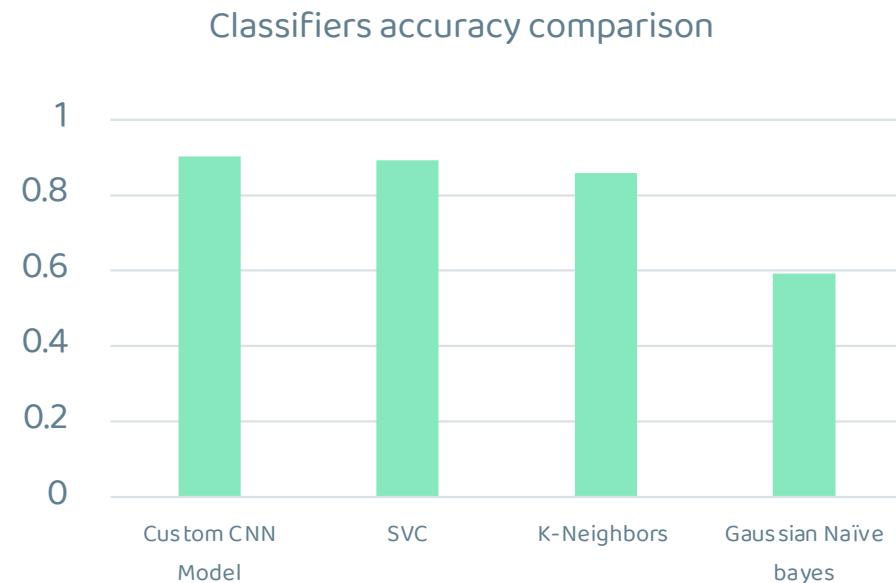
Confusion Matrix

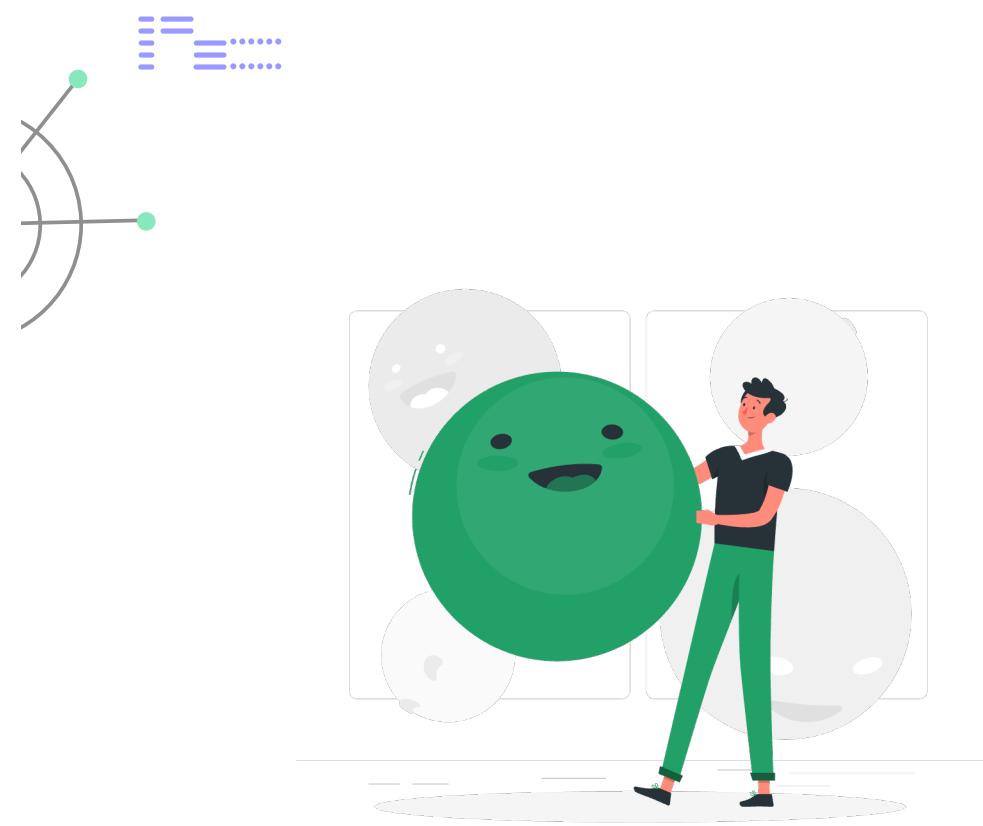


**07**  
**Conclusion**

After comparing our CNN classifier with other pretrained ones like SVC, K-Neighbors, Gaussian Naïve bayes, and the logistic regression ones  
We found that ours is more accurate than them

Classifier	Accuracy
Custom CNN Model	0.9030
SVC	0.8921
K-Neighbors	0.8589
Gaussian Naïve bayes	0.5914
Logistic regression	0.8544





# Thanks!

