notebook

November 15, 2024

1 Project Summary

The project titled "Enhancing Fashion Image Classification with AI for Industry Applications" aims to leverage artificial intelligence (AI) to improve fashion image classification, addressing a critical gap in the fashion industry. The global fashion market is projected to reach approximately $3 trillion by 2030, with a compound annual growth rate (CAGR) of 5.5% from 2021 to 2030 (Statista, 2023). In Kenya, the fashion industry is also burgeoning, contributing about 3% to the country's GDP and employing over 1.5 million people (Kenya National Bureau of Statistics, 2022). However, despite these promising statistics, the effective deployment of AI models for fashion image classification remains underexplored, particularly in high-stakes environments like e-commerce and fashion shows.

1.1 Business Understanding

The fashion industry is characterized by rapid changes in consumer preferences and trends, necessitating a robust mechanism for image classification to enhance design, marketing, and customer experience. Major fashion brands, including Gucci and Louis Vuitton, have recognized the need for more precise AI models to optimize their design processes and consumer interactions (Vogue Business, 2023). This project seeks to fill this gap by developing a model that can accurately classify fashion images, thereby improving operational efficiency and customer satisfaction.

1.2 Problem Statement

Despite advancements in AI, existing fashion image classification models often lack the precision required for real-world applications, leading to inefficiencies in design and marketing strategies. The inability to accurately classify images can result in misaligned inventory, poor customer experiences, and lost sales opportunities. This project addresses the need for a more effective classification model that can be utilized across various fashion industry applications.

1.3 Objectives

1.3.1 Main Objective

The primary objective of this project is to develop a robust fashion image classification model using the Fashion-MNIST dataset, enhancing its accuracy through advanced machine learning techniques and feature engineering. The model will be deployed via an interactive web application using Streamlit, making it accessible to stakeholders in the fashion industry.

1.3.2 Key Business Questions

1. How can AI-driven fashion image classification improve operational efficiency in design and marketing?
2. What machine learning techniques yield the highest accuracy for fashion image classification?
3. How can the developed model be integrated into existing fashion industry workflows?
4. What are the user experiences and feedback from stakeholders using the interactive application?
5. How can the model be adapted to cater to the unique needs of the Kenyan fashion market?

1.4 Stakeholders

Key stakeholders include: 1. Fashion designers 2. E-commerce platforms 3. Retailers 4. Consumers.

Additionally, industry leaders and creative directors from major fashion houses, as well as technology developers, will play a crucial role in the project's success.

1.5 Data Understanding

The project utilizes the Fashion-MNIST dataset, which comprises 70,000 grayscale images of clothing items, with 60,000 images designated for training and 10,000 for testing. Each image is flattened into a feature vector of 784 features, allowing for effective model training. This dataset serves as a foundational resource for developing a model that can accurately classify various clothing categories, ultimately contributing to enhanced consumer experiences and operational efficiencies in the fashion industry.

1.6 Metric of Success

- The "Metric Success" of the notebook is primarily evaluated through accuracy, loss, precision, recall, and F1-score.
- The model achieves an overall accuracy of 90%, indicating strong performance in classifying clothing categories.
- Loss values decrease over epochs, suggesting effective learning, while precision and recall metrics reveal nuanced performance; for instance, precision for "Sandal" is high at 0.98, but recall for "Shirt" is lower at 0.75, indicating some misclassifications.
- The confusion matrix highlights specific challenges, such as confusion between "Shirt" and "T-shirt/top," with 46 misclassifications.
- Overall, the classification report shows a balanced performance across classes, with a macro average F1-score of 0.90, suggesting the model is effective but has room for improvement in distinguishing visually similar items.

1.7 Import Dependancies

[1]:
```
import random
import numpy as np
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import gzip
import struct
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import cv2
from rembg import remove
from PIL import Image
from tensorflow.keras import layers, models, optimizers

# Set seeds for reproducibility
random.seed(0)
np.random.seed(0)
tf.random.set_seed(0)
```

```
2024-11-14 19:24:33.775429: I tensorflow/core/util/port.cc:153] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-11-14 19:24:33.789983: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
E0000 00:00:1731601473.805218  143161 cuda_dnn.cc:8310] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already
been registered
E0000 00:00:1731601473.809416  143161 cuda_blas.cc:1418] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
2024-11-14 19:24:33.824937: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
```

In order for the code's behavior to be predictable and repeatable, which is especially important for debugging and comparing results.

1.8 Data Loading

1.8.1 Function to extract images

[2]:
```
def extract_images(file_path):
    with gzip.open(file_path, 'rb') as f:
        magic, num, rows, cols = struct.unpack(">IIII", f.read(16))
        images = np.frombuffer(f.read(), dtype=np.uint8).reshape(num, rows,␣
 ↪cols)
    return images
```

Reading a gzip-compressed file containing image data, unpack the file's header to get the number of images and their dimensions, and then extracts and reshapes the image data into a 3D numpy array for easy access.

1.8.2 Function to extract labels

[3]:
```
def extract_labels(file_path):
    with gzip.open(file_path, 'rb') as f:
        magic, num = struct.unpack(">II", f.read(8))
        labels = np.frombuffer(f.read(), dtype=np.uint8)
    return labels
```

Reading a gzip-compressed file containing label data, unpack the file's header to get the number of labels, and then extract the label data into a numpy array for easy use

1.8.3 Extract and load the dataset

[4]:
```
train_images_path = 'data/train-images-idx3-ubyte.gz'
train_labels_path = 'data/train-labels-idx1-ubyte.gz'
test_images_path = 'data/t10k-images-idx3-ubyte.gz'
test_labels_path = 'data/t10k-labels-idx1-ubyte.gz'

train_images = extract_images(train_images_path)
train_labels = extract_labels(train_labels_path)
test_images = extract_images(test_images_path)
test_labels = extract_labels(test_labels_path)
```

Loading and processing training and test image data, along with their corresponding labels, from specified gzip-compressed files

[5]:
```
train_images.shape, train_labels.shape
```

[5]:
```
((60000, 28, 28), (60000,))
```

There are 60,000 training images of size 28x28 pixels and 60,000 corresponding labels.

[6]:
```
test_images.shape,  test_labels.shape
```

[6]:
```
((10000, 28, 28), (10000,))
```

There are 10,000 training images of size 28x28 pixels and 10,000 corresponding labels.

1.9 Labels

Each training and test images is assigned to one of the following labels: - 0 T-shirt/top - 1 Trouser - 2 Pullover - 3 Dress - 4 Coat - 5 Sandal - 6 Shirt - 7 Sneaker - 8 Bag - 9 Ankle boot

1.10 Exploratory Data Analysis

[7]:
```
# Define the label names
label_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

# Create a figure to display images in a 2x5 grid
fig, axes = plt.subplots(2, 5, figsize=(12, 6))

# Find and plot one example of each label
for i in range(10):
    ax = axes[i // 5, i % 5]  # Determine the subplot index
    label_index = np.where(train_labels == i)[0][0]  # Find the first instance␣
 ↪of each label
    ax.imshow(train_images[label_index], cmap="Greys")
    ax.set_title(label_names[i])
    ax.axis('off')

# Display the grid
plt.tight_layout()
plt.show();
```

Creating a visual representation of one example image for each label in a 2x5 grid, with titles corresponding to the label names.

The images in the dataset are low-resolution, grayscale representations of various clothing items, designed to be simple yet distinctive enough for machine learning algorithms to recognize and classify different fashion items.

1.11 Data Processing

1.11.1 Feature Scaling

[8]:
```
# Normalize pixel values
train_images = train_images / 255.0
test_images = test_images / 255.0
```

Normalize the pixel values of the training and test images, scaling them to a range between 0 and 1 by dividing by 255. In order to improve the performance of machine learning models.

1.11.2 Re-Sizing

[9]:
```
# Reshape for CNN
train_images = train_images.reshape((train_images.shape[0], 28, 28, 1))
test_images = test_images.reshape((test_images.shape[0], 28, 28, 1))
```

Reshaping the training and test images to include a channel dimension, preparing them for input into a Convolutional Neural Network (CNN).

1.11.3 Split Data Set

[10]:
```
X_train, X_val, y_train, y_val = train_test_split(train_images, train_labels,␣
↪test_size=0.2, random_state=34)
```

Spliting the training data and labels into training and validation sets, with 20% of the data allocated for validation, to evaluate the model's performance.

[11]:
```
X_train.shape, y_train.shape
```

[11]:
```
((48000, 28, 28, 1), (48000,))
```

There are 48,000 training images with dimensions 28x28 pixels and 1 channel, along with 48,000 corresponding labels.

[12]:
```
X_val.shape, y_val.shape
```

[12]:
```
((12000, 28, 28, 1), (12000,))
```

There are 12,000 training images with dimensions 28x28 pixels and 1 channel, along with 12,000 corresponding labels.

1.12 Model Selection - CNN

[13]:
```
model = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=3, activation='relu',␣
 ↪input_shape=[28, 28, 1]),
    layers.MaxPooling2D(pool_size=2),
    layers.Conv2D(filters=64, kernel_size=3, activation='relu'),
    layers.MaxPooling2D(pool_size=2),
    layers.Conv2D(filters=128, kernel_size=3, activation='relu'),
    layers.Flatten(),
    layers.Dense(units=512, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(units=256, activation='relu'),
    layers.Dense(units=10, activation='softmax'),
])
```

```
/home/ikn/miniforge3/envs/learn-env/lib/python3.10/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2024-11-14 19:24:39.965703: E
external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:152] failed call to
cuInit: INTERNAL: CUDA error: Failed call to cuInit: CUDA_ERROR_NO_DEVICE: no
CUDA-capable device is detected
```

It defines a Convolutional Neural Network (CNN) model with three convolutional layers for feature extraction, followed by max pooling to reduce spatial dimensions, and includes two fully connected layers with a dropout layer to prevent overfitting. The final output layer uses softmax activation for classifying images into one of 10 categories.

[14]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 128)	73,856

```
flatten (Flatten)                    (None, 1152)                          0

dense (Dense)                        (None, 512)                     590,336

dropout (Dropout)                    (None, 512)                           0

dense_1 (Dense)                      (None, 256)                     131,328

dense_2 (Dense)                      (None, 10)                        2,570


Total params: 816,906 (3.12 MB)

Trainable params: 816,906 (3.12 MB)

Non-trainable params: 0 (0.00 B)
```

[15]:
```
optimizer = optimizers.Adam(learning_rate=0.0001)

model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

It initializes an Adam optimizer with a specified learning rate of 0.0001 and compiles the neural network model using this optimizer. The model is set to minimize the sparse categorical crossentropy loss function, which is suitable for multi-class classification problems with integer labels. Additionally, the model's performance will be evaluated using accuracy as a metric during training and testing.

1.13 Train Model

[16]:
```
history = model.fit(X_train, y_train, epochs=40, batch_size=512, verbose=1,␣
 ↪validation_data=(X_val, y_val))
```

```
Epoch 1/40
94/94              10s 95ms/step -
accuracy: 0.3324 - loss: 2.0057 - val_accuracy: 0.6815 - val_loss: 0.8610
Epoch 2/40
94/94              8s 85ms/step -
accuracy: 0.6830 - loss: 0.8521 - val_accuracy: 0.7409 - val_loss: 0.6817
Epoch 3/40
94/94              7s 76ms/step -
accuracy: 0.7390 - loss: 0.6886 - val_accuracy: 0.7646 - val_loss: 0.6061
Epoch 4/40
94/94              7s 74ms/step -
```

```
accuracy: 0.7627 - loss: 0.6179 - val_accuracy: 0.7852 - val_loss: 0.5570
Epoch 5/40
94/94            7s 75ms/step -
accuracy: 0.7857 - loss: 0.5700 - val_accuracy: 0.7983 - val_loss: 0.5242
Epoch 6/40
94/94            7s 74ms/step -
accuracy: 0.7972 - loss: 0.5377 - val_accuracy: 0.8155 - val_loss: 0.4925
Epoch 7/40
94/94            7s 78ms/step -
accuracy: 0.8125 - loss: 0.5082 - val_accuracy: 0.8243 - val_loss: 0.4702
Epoch 8/40
94/94            8s 84ms/step -
accuracy: 0.8219 - loss: 0.4816 - val_accuracy: 0.8324 - val_loss: 0.4501
Epoch 9/40
94/94            8s 84ms/step -
accuracy: 0.8331 - loss: 0.4602 - val_accuracy: 0.8426 - val_loss: 0.4306
Epoch 10/40
94/94            8s 84ms/step -
accuracy: 0.8366 - loss: 0.4466 - val_accuracy: 0.8494 - val_loss: 0.4154
Epoch 11/40
94/94            8s 83ms/step -
accuracy: 0.8459 - loss: 0.4247 - val_accuracy: 0.8533 - val_loss: 0.4036
Epoch 12/40
94/94            8s 83ms/step -
accuracy: 0.8487 - loss: 0.4143 - val_accuracy: 0.8583 - val_loss: 0.3908
Epoch 13/40
94/94            8s 83ms/step -
accuracy: 0.8570 - loss: 0.3997 - val_accuracy: 0.8634 - val_loss: 0.3821
Epoch 14/40
94/94            8s 82ms/step -
accuracy: 0.8599 - loss: 0.3914 - val_accuracy: 0.8643 - val_loss: 0.3726
Epoch 15/40
94/94            8s 83ms/step -
accuracy: 0.8628 - loss: 0.3832 - val_accuracy: 0.8673 - val_loss: 0.3670
Epoch 16/40
94/94            8s 84ms/step -
accuracy: 0.8656 - loss: 0.3703 - val_accuracy: 0.8697 - val_loss: 0.3593
Epoch 17/40
94/94            8s 83ms/step -
accuracy: 0.8674 - loss: 0.3647 - val_accuracy: 0.8708 - val_loss: 0.3533
Epoch 18/40
94/94            8s 82ms/step -
accuracy: 0.8730 - loss: 0.3570 - val_accuracy: 0.8745 - val_loss: 0.3475
Epoch 19/40
94/94            8s 83ms/step -
accuracy: 0.8731 - loss: 0.3507 - val_accuracy: 0.8748 - val_loss: 0.3421
Epoch 20/40
94/94            8s 82ms/step -
```

```
accuracy: 0.8762 - loss: 0.3456 - val_accuracy: 0.8766 - val_loss: 0.3384
Epoch 21/40
94/94            8s 84ms/step -
accuracy: 0.8790 - loss: 0.3361 - val_accuracy: 0.8786 - val_loss: 0.3330
Epoch 22/40
94/94            8s 81ms/step -
accuracy: 0.8796 - loss: 0.3316 - val_accuracy: 0.8800 - val_loss: 0.3288
Epoch 23/40
94/94            8s 84ms/step -
accuracy: 0.8798 - loss: 0.3282 - val_accuracy: 0.8811 - val_loss: 0.3260
Epoch 24/40
94/94            8s 82ms/step -
accuracy: 0.8814 - loss: 0.3235 - val_accuracy: 0.8828 - val_loss: 0.3220
Epoch 25/40
94/94            8s 82ms/step -
accuracy: 0.8863 - loss: 0.3182 - val_accuracy: 0.8854 - val_loss: 0.3179
Epoch 26/40
94/94            8s 81ms/step -
accuracy: 0.8867 - loss: 0.3145 - val_accuracy: 0.8834 - val_loss: 0.3163
Epoch 27/40
94/94            8s 80ms/step -
accuracy: 0.8870 - loss: 0.3106 - val_accuracy: 0.8867 - val_loss: 0.3126
Epoch 28/40
94/94            8s 88ms/step -
accuracy: 0.8884 - loss: 0.3069 - val_accuracy: 0.8878 - val_loss: 0.3083
Epoch 29/40
94/94            8s 87ms/step -
accuracy: 0.8909 - loss: 0.3011 - val_accuracy: 0.8873 - val_loss: 0.3071
Epoch 30/40
94/94            9s 100ms/step -
accuracy: 0.8914 - loss: 0.2989 - val_accuracy: 0.8901 - val_loss: 0.3050
Epoch 31/40
94/94            10s 102ms/step -
accuracy: 0.8927 - loss: 0.2954 - val_accuracy: 0.8895 - val_loss: 0.3033
Epoch 32/40
94/94            8s 82ms/step -
accuracy: 0.8931 - loss: 0.2921 - val_accuracy: 0.8904 - val_loss: 0.3013
Epoch 33/40
94/94            8s 86ms/step -
accuracy: 0.8928 - loss: 0.2916 - val_accuracy: 0.8918 - val_loss: 0.3001
Epoch 34/40
94/94            7s 79ms/step -
accuracy: 0.8962 - loss: 0.2836 - val_accuracy: 0.8912 - val_loss: 0.2983
Epoch 35/40
94/94            7s 78ms/step -
accuracy: 0.8969 - loss: 0.2827 - val_accuracy: 0.8932 - val_loss: 0.2947
Epoch 36/40
94/94            8s 80ms/step -
```

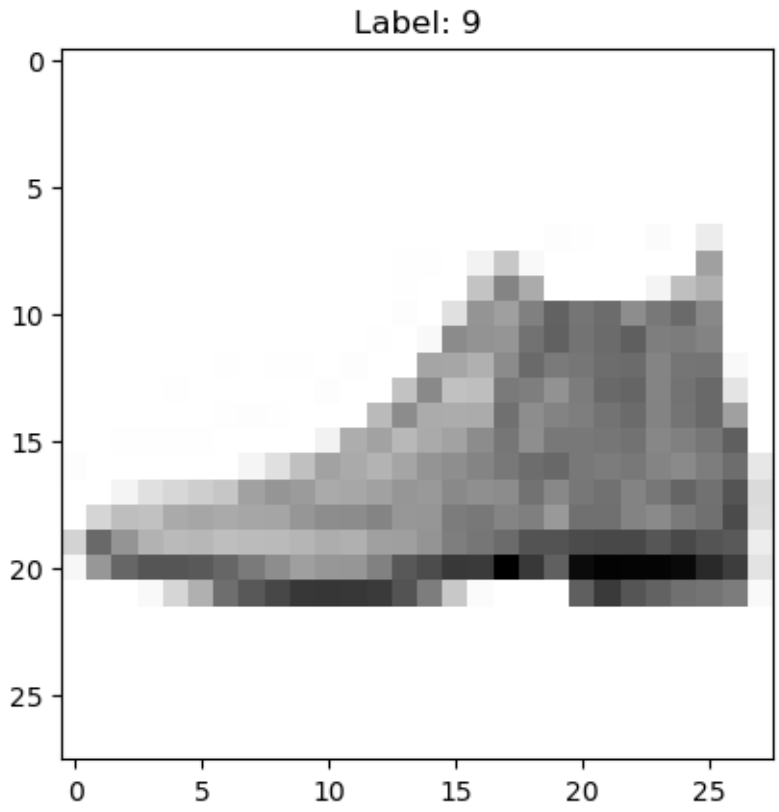
```
accuracy: 0.8994 - loss: 0.2805 - val_accuracy: 0.8926 - val_loss: 0.2956
Epoch 37/40
94/94             8s 86ms/step -
accuracy: 0.8993 - loss: 0.2766 - val_accuracy: 0.8953 - val_loss: 0.2911
Epoch 38/40
94/94             8s 82ms/step -
accuracy: 0.9011 - loss: 0.2733 - val_accuracy: 0.8957 - val_loss: 0.2886
Epoch 39/40
94/94             8s 85ms/step -
accuracy: 0.9015 - loss: 0.2719 - val_accuracy: 0.8968 - val_loss: 0.2887
Epoch 40/40
94/94             8s 85ms/step -
accuracy: 0.9016 - loss: 0.2688 - val_accuracy: 0.8968 - val_loss: 0.2869
```

Training the neural network model on the training data (X_train and y_train) for 40 epochs with a batch size of 512, while also evaluating it on the validation data (X_val and y_val).

The output shows the training progress, including accuracy and loss for both training and validation sets for each epoch. The model's accuracy improves over time, reaching around 90.16% on the training set and 89.68% on the validation set by the final epoch.

1.14 Test Model

[17]:
```python
# Display the test image
plt.imshow(test_images[0], cmap="Greys")
plt.title(f"Label: {test_labels[0]}")
plt.show()
```



[19]:
```
label = np.argmax(model.predict(np.expand_dims(test_images[0], axis=0)).
  ↪round(2))
print("The label number is ", label)
```

```
1/1             0s 35ms/step
The label number is  9
```

Predicts the label for the first image in the test_images array by expanding its dimensions to match the input shape expected by the model, making a prediction, and finding the class with the highest probability.

It then prints the predicted label, which in this case is 9.

[20]:
```
# Create a figure to display images in a 2x5 grid
fig, axes = plt.subplots(2, 5, figsize=(12, 6))

# Select 10 random indices
random_indices = np.random.choice(test_images.shape[0], 10, replace=False)

# Plot the images and their predicted labels
for i, idx in enumerate(random_indices):
    ax = axes[i // 5, i % 5]
```

```
    ax.imshow(test_images[idx].reshape(28, 28), cmap="Greys")
    label = np.argmax(model.predict(np.expand_dims(test_images[idx], axis=0)).
 ↪round(2))
    ax.set_title(label_names[label])
    ax.axis('off')

# Display the grid
plt.tight_layout()
plt.show()
```

```
1/1             0s 13ms/step
1/1             0s 12ms/step
1/1             0s 12ms/step
1/1             0s 12ms/step
1/1             0s 13ms/step
1/1             0s 12ms/step
1/1             0s 19ms/step
1/1             0s 13ms/step
1/1             0s 12ms/step
1/1             0s 13ms/step
```

1.15 Evaluate Model

[21]:
```
y_pred = model.predict(test_images).round(2)
y_pred
```

```
313/313             0s 1ms/step
```

[21]:
```
array([[0.  , 0.  , 0.  , …, 0.01, 0.  , 0.99],
       [0.  , 0.  , 0.99, …, 0.  , 0.  , 0.  ],
```

```
[0.  , 1.  , 0.  , …, 0.  , 0.  , 0.  ],
…,
[0.  , 0.  , 0.  , …, 0.  , 0.99, 0.  ],
[0.  , 1.  , 0.  , …, 0.  , 0.  , 0.  ],
[0.  , 0.  , 0.  , …, 0.4 , 0.03, 0.  ]], dtype=float32)
```

- Predictions for all images in the test_images dataset using the trained model.
- The output y_pred is a 2D numpy array where each row corresponds to an image and each column represents the probability of that image belonging to a specific class, rounded to two decimal places.
- The first image has a high probability (0.99) of belonging to the 9th class.

[22]:
```
model.evaluate(test_images, test_labels)
```

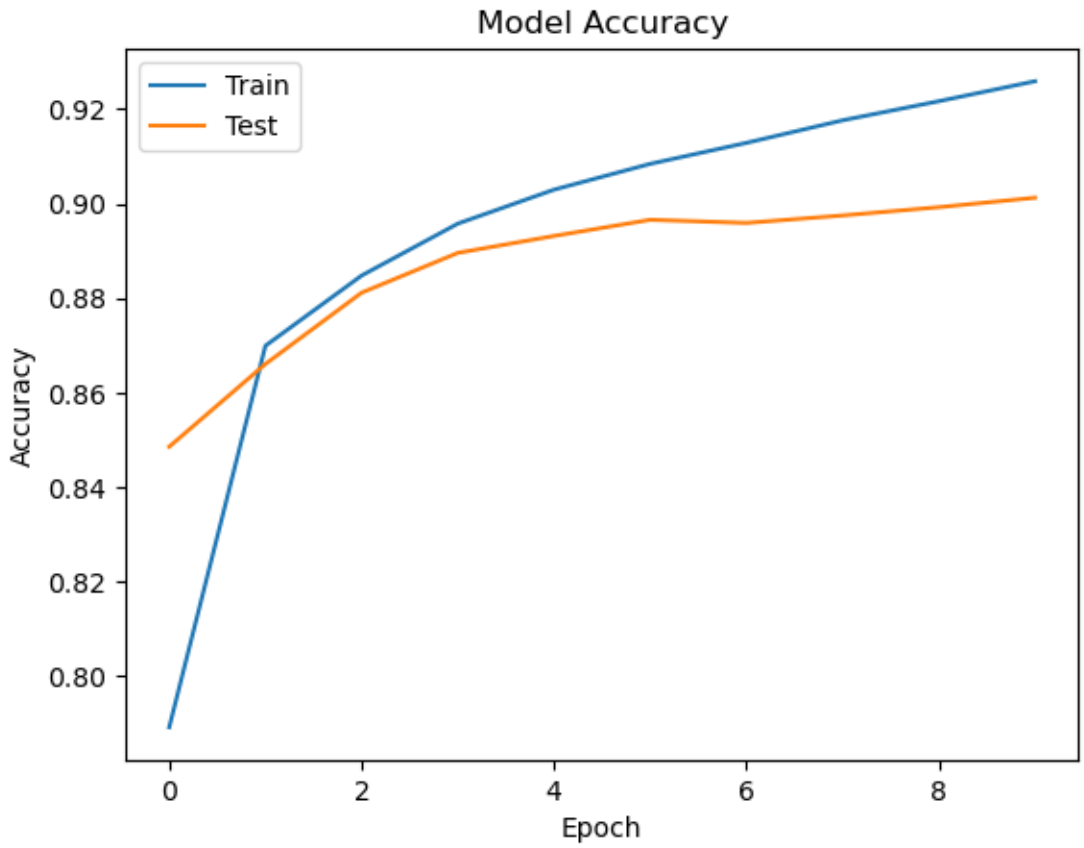
```
313/313             0s 1ms/step -
accuracy: 0.9031 - loss: 0.2820
```

[22]:
```
[0.2831084132194519, 0.899399995803833]
```

Evaluate the performance of the trained model on the test dataset (test_images and test_labels). The model achieved an accuracy of approximately 90.31% and a loss of about 0.2820 on the test set, indicating how well the model generalizes to unseen data.

1.15.1 Plot training & validation accuracy values

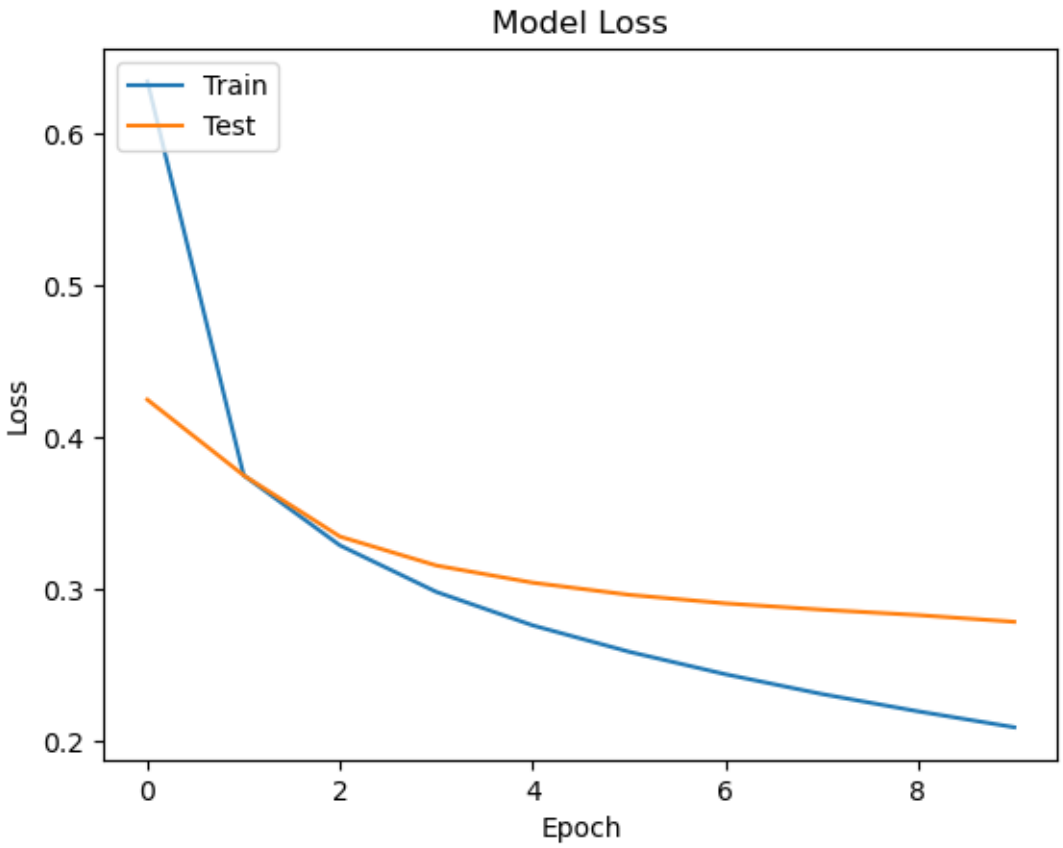
[23]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



- Generate a line plot to visualize the training and validation accuracy of the model over each epoch. Using matplotlib to plot the accuracy values stored in the history object.
- The plot shows the trend of increasing accuracy for both training and validation sets, helping to assess the model's learning progress and identify any potential overfitting.

1.15.2 Plot training & validation loss values

[24]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

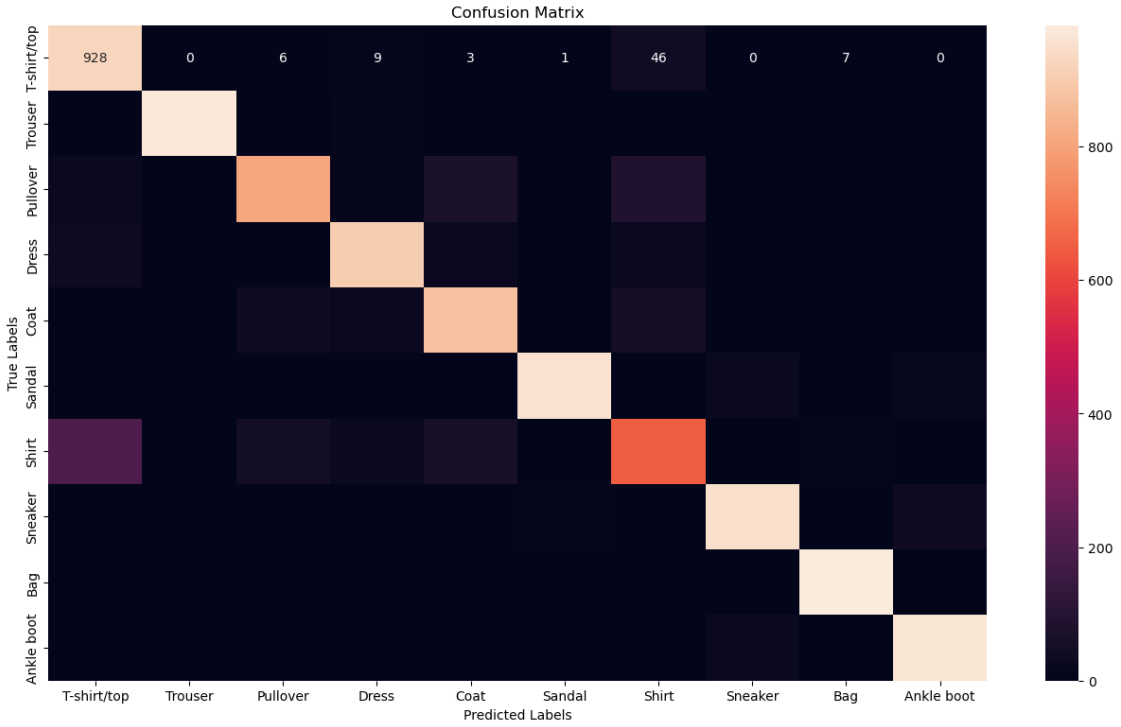


- Generate a line plot to visualize the loss values of the model during training and validation phases over each epoch. It plots the training loss and validation loss on the y-axis against the number of epochs on the x-axis.
- The graph helps to identify trends in loss reduction and can highlight any overfitting, by comparing the decrease in loss for both training and validation sets.
- The plot shows how well the model is learning and generalizing over time.

1.16 Confussion Matrics

[25]:
```
y_pred_labels = [np.argmax(label) for label in y_pred]

# Create the confusion matrix
cm = confusion_matrix(test_labels, y_pred_labels)

# Plot the confusion matrix using seaborn
plt.figure(figsize=(16, 9))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=label_names,␣
 ↪yticklabels=label_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
```

```
plt.title('Confusion Matrix')
plt.show()
```



- High Accuracy for Certain Classes: The model excels at classifying "T-shirt/top" and "Trouser" with 928 and 946 correct predictions respectively.
- Misclassification Patterns: The model struggles to distinguish "Shirt" from "T-shirt/top" (46 instances) and "Pullover" from "Coat" (42 instances).
- Confusion Between Similar Items: Noticeable confusion exists between "Coat" and "Pullover" (42 instances) and "Shirt" and "T-shirt/top" (46 instances).
- Least Confused Classes: "Trouser" and "Ankle boot" are easily distinguishable, with very few misclassifications.
- Overall Performance: The majority of predictions are correct, but the model has specific challenges with visually similar items.

[26]:
```
cr = classification_report(test_labels, y_pred_labels, target_names=label_names)
print(cr)
```

```
              precision    recall  f1-score   support

 T-shirt/top       0.77      0.93      0.84      1000
     Trouser       0.99      0.97      0.98      1000
    Pullover       0.89      0.81      0.84      1000
```

```
              precision    recall  f1-score   support

       Dress       0.90      0.90      0.90      1000
        Coat       0.84      0.87      0.86      1000
      Sandal       0.98      0.96      0.97      1000
       Shirt       0.75      0.65      0.70      1000
     Sneaker       0.94      0.95      0.95      1000
         Bag       0.97      0.98      0.98      1000
  Ankle boot       0.95      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
```

The classification report evaluates the model's performance for each clothing category by showing precision, recall, and F1-score metrics. It indicates an overall accuracy of 90%, with strong performance across most categories, though "Shirt" has a relatively lower precision and recall.

1.17 Save Model

[27]:
```
# Save the model using the native Keras format
model.save('./model/trained_fashion_mnist_model.keras')
```

1.18 Model Deployment

[28]:
```
# Dependancies
# ! pip install rembg
```

[29]:
```
# Load images
img1 = Image.open("test/test1.jpeg")
img2 = Image.open("test/test2.jpeg")
img3 = Image.open("test/test3.jpeg")
img4 = Image.open("test/test4.jpeg")

# Display images
fig, axs = plt.subplots(2, 2)  # Create a 2x2 grid for displaying images

# Display each image in the subplot
axs[0, 0].imshow(img1)
axs[0, 0].set_title('Image 1')
axs[0, 1].imshow(img2)
axs[0, 1].set_title('Image 2')
axs[1, 0].imshow(img3)
axs[1, 0].set_title('Image 3')
axs[1, 1].imshow(img4)
axs[1, 1].set_title('Image 4')

# Hide axes
```

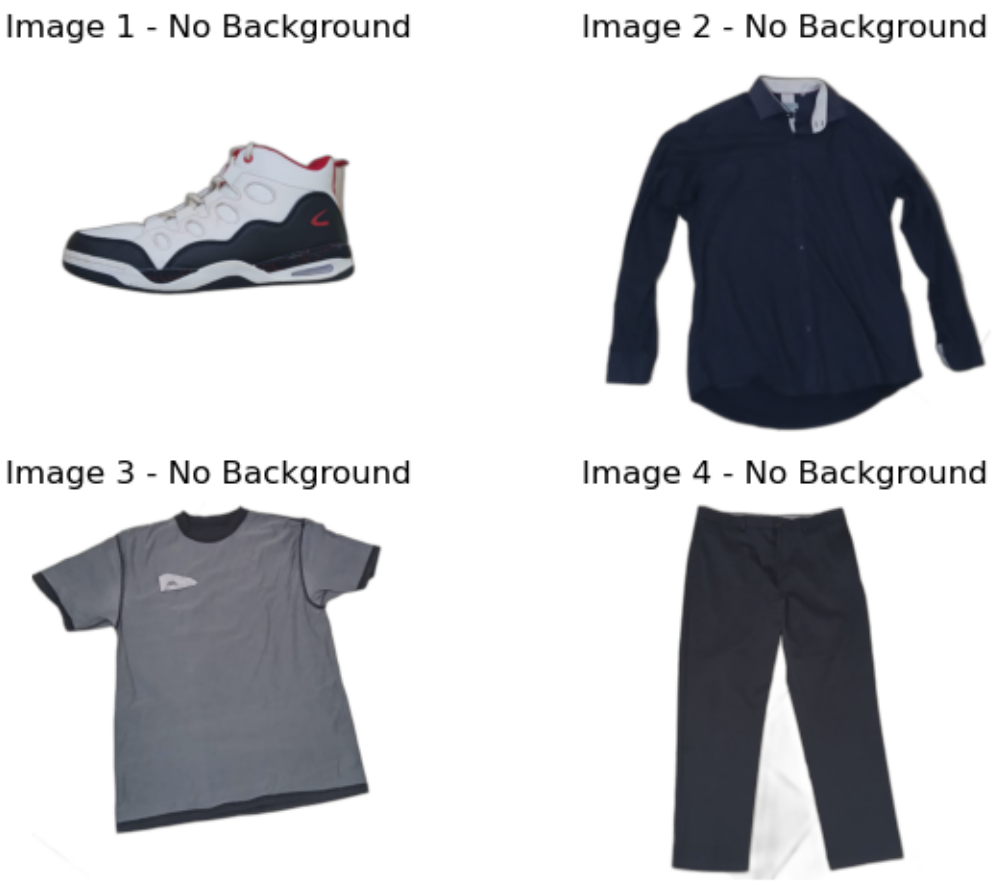
```
for ax in axs.flat:
    ax.axis('off')

plt.show();
```

Loading and displaying four images in a 2x2 grid using Matplotlib. Each image is shown in its own subplot with a title ("Image 1" to "Image 4"), and the axes are hidden for a cleaner presentation.

1.18.1 Remove background-image

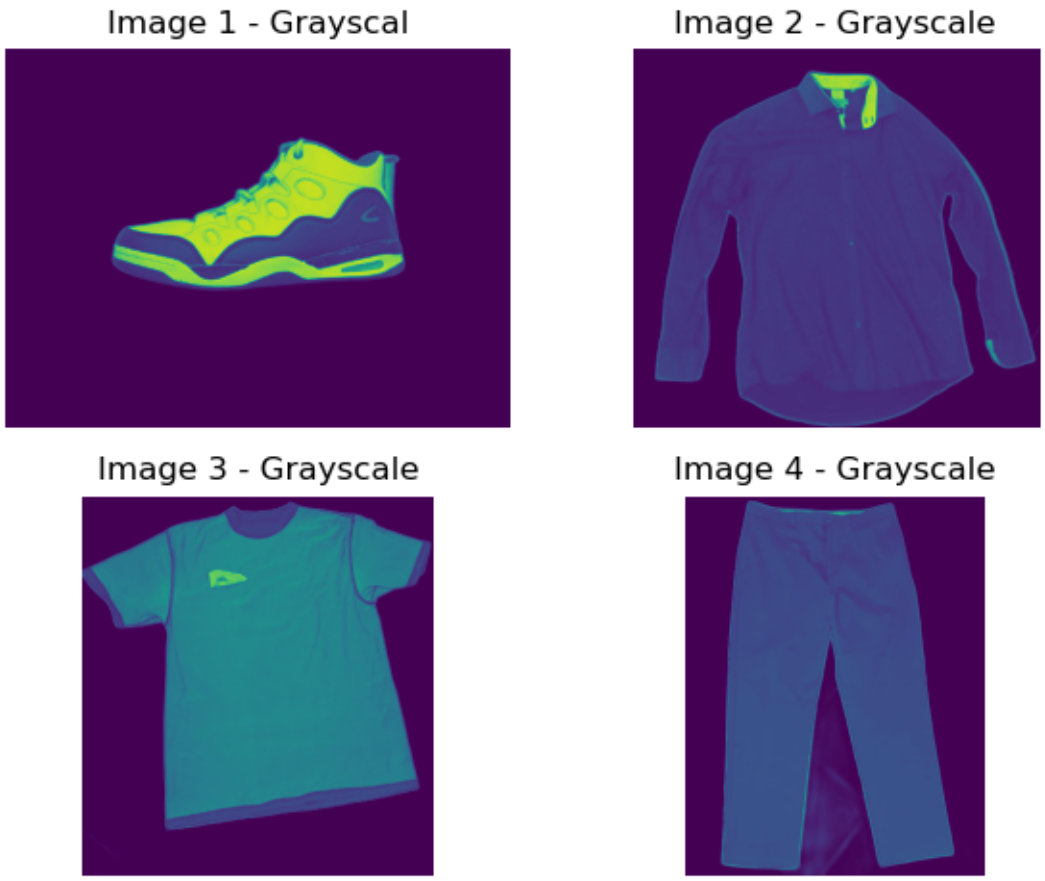
[30]:
```
# Remove backgrounds
img1_no_bg = remove(img1)
img2_no_bg = remove(img2)
img3_no_bg = remove(img3)
img4_no_bg = remove(img4)

# Display images without background
fig, axs = plt.subplots(2, 2)

# Display each image in the subplot
axs[0, 0].imshow(img1_no_bg)
axs[0, 0].set_title('Image 1 - No Background')
```

```
axs[0, 1].imshow(img2_no_bg)
axs[0, 1].set_title('Image 2 - No Background')
axs[1, 0].imshow(img3_no_bg)
axs[1, 0].set_title('Image 3 - No Background')
axs[1, 1].imshow(img4_no_bg)
axs[1, 1].set_title('Image 4 - No Background')

# Hide axes
for ax in axs.flat:
    ax.axis('off')

plt.tight_layout()
plt.show();
```



Removing the backgrounds from four images and then displays them in a 2x2 grid layout without their original backgrounds. Each subplot is titled accordingly, and the axes are hidden for a cleaner visual presentation.

[31]:
```
# Convert images to grayscale
img1_gray = img1_no_bg.convert('L')
img2_gray = img2_no_bg.convert('L')
img3_gray = img3_no_bg.convert('L')
img4_gray = img4_no_bg.convert('L')

# Display grayscale images
fig, axs = plt.subplots(2, 2)

# Display each grayscale image in the subplot
axs[0, 0].imshow(img1_gray)
axs[0, 0].set_title('Image 1 - Grayscal')
axs[0, 1].imshow(img2_gray)
axs[0, 1].set_title('Image 2 - Grayscale')
axs[1, 0].imshow(img3_gray)
axs[1, 0].set_title('Image 3 - Grayscale')
axs[1, 1].imshow(img4_gray)
axs[1, 1].set_title('Image 4 - Grayscale')

# Hide axes
for ax in axs.flat:
    ax.axis('off')

plt.tight_layout()
plt.show();
```



Converting four images with removed backgrounds to grayscale and then displays them in a 2x2 grid layout. Each subplot contains one grayscale image with a corresponding title, and the axes are hidden for a cleaner presentation.

[32]:
```
# Resize the image to 28x28 pixels
img1_resize = img1_gray.resize((28, 28))
img2_resize = img2_gray.resize((28, 28))
img3_resize = img3_gray.resize((28, 28))
img4_resize = img4_gray.resize((28, 28))
```

Resizing four grayscale images (with removed backgrounds) to 28x28 pixels each, preparing them for prediction.

[33]:
```
# Convert the image to a numpy array and normalize pixel values
img1_array = np.array(img1_resize) / 255.0
img2_array = np.array(img2_resize) / 255.0
img3_array = np.array(img3_resize) / 255.0
img4_array = np.array(img4_resize) / 255.0
```

Converting the resized 28x28 grayscale images into numpy arrays and normalizes their pixel values

to a range of 0 to 1 by dividing by 255.

[34]:
```
# Expand dimensions to match the input shape expected by the model
img1 = np.expand_dims(img1_array, axis=(0, -1))
img2 = np.expand_dims(img2_array, axis=(0, -1))
img3 = np.expand_dims(img3_array, axis=(0, -1))
img4 = np.expand_dims(img4_array, axis=(0, -1))
```

Expanding the dimensions of the normalized image arrays to match the input shape expected by the model, adding new dimensions for batch size and channel.

[35]:
```
# Predict the labels
label1 = np.argmax(model.predict(img1).round(2))
label2 = np.argmax(model.predict(img2).round(2))
label3 = np.argmax(model.predict(img3).round(2))
label4 = np.argmax(model.predict(img4).round(2))

print("The label number for img1 is", label1)
print("The label number for img2 is", label2)
print("The label number for img3 is", label3)
print("The label number for img4 is", label4)

# Print the corresponding label names
label_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

print("The predicted label for img1 is", label_names[label1])
print("The predicted label for img2 is", label_names[label2])
print("The predicted label for img3 is", label_names[label3])
print("The predicted label for img4 is", label_names[label4])
```

```
1/1             0s 13ms/step
1/1             0s 13ms/step
1/1             0s 13ms/step
1/1             0s 13ms/step
The label number for img1 is 7
The label number for img2 is 6
The label number for img3 is 0
The label number for img4 is 1
The predicted label for img1 is Sneaker
The predicted label for img2 is Shirt
The predicted label for img3 is T-shirt/top
The predicted label for img4 is Trouser
```

Uses a trained neural network model to predict the labels of four preprocessed images. It identifies the class with the highest probability for each image and then prints both the numeric label and the corresponding class name.

The predictions are “Sneaker,” “Shirt,” “T-shirt/top,” and “Trouser” for the respective images.

2 Conclusion

2.1 Insights

- High Overall Accuracy: The model achieves an impressive overall accuracy of 90%, indicating its effectiveness in classifying clothing items.
- Loss Reduction: The training and validation loss consistently decrease over epochs, demonstrating that the model is learning and improving its predictions.
- Class-Specific Performance: Precision and recall metrics reveal that while the model excels in categories like "Sandal" and "Trouser," it struggles with "Shirt," which has a lower recall of 0.75.
- Confusion Patterns: The confusion matrix indicates significant misclassifications between visually similar classes, particularly between "Shirt" and "T-shirt/top," highlighting areas for improvement.
- Balanced F1-Score: The macro average F1-score of 0.90 suggests a well-balanced performance across different classes, although some categories require more attention.

2.2 Model Performance Overview

The model demonstrates strong performance metrics, with a training accuracy reaching approximately 90.16% and a validation accuracy of 89.68% by the final epoch. The confusion matrix and classification report provide insights into specific class performance, revealing both strengths and weaknesses in the model's predictions.

2.3 Recommendations

- Data Augmentation: Implement data augmentation techniques to enhance the training dataset, particularly for classes with lower performance, such as "Shirt."
- Hyperparameter Tuning: Experiment with different hyperparameters, such as learning rates and batch sizes, to further optimize model performance.
- Ensemble Methods: Consider using ensemble methods to combine predictions from multiple models, which may improve overall accuracy and robustness.
- Class Rebalancing: If certain classes are underrepresented, techniques like oversampling or synthetic data generation could help balance the dataset.
- Regularization Techniques: Apply regularization methods to prevent overfitting, especially if the model shows signs of diverging performance between training and validation datasets.

2.4 Limitations

- Class Confusion: The model exhibits confusion between visually similar classes, which can lead to misclassifications and reduced overall performance.
- Limited Dataset Size: The performance may be constrained by the size and diversity of the training dataset, potentially affecting the model's ability to generalize to unseen data.

- Single Model Approach: Relying on a single model architecture may limit the exploration of more complex or specialized models that could yield better results.
- Evaluation Metrics: While accuracy is a useful metric, it may not fully capture the model's performance, especially in cases of class imbalance.
- Computational Resources: The training process may require significant computational resources, which could limit accessibility for some users.

2.5 Future Works

- Model Architecture Exploration: Investigate more advanced architectures, such as convolutional neural networks (CNNs) or transfer learning models, to enhance classification performance.
- Incorporate Additional Data: Expand the dataset by including more diverse clothing items or variations to improve the model's robustness and generalization capabilities.
- Real-Time Testing: Implement real-time testing and deployment of the model to evaluate its performance in practical applications, such as fashion recommendation systems.
- User Feedback Integration: Develop a feedback loop where user interactions and corrections can be used to continuously improve the model's accuracy over time.
- Cross-Validation: Utilize cross-validation techniques to ensure the model's performance is consistent across different subsets of the data, providing a more reliable evaluation of its effectiveness.

2.6 References

- Statista. (2023). "Global Fashion Market - Statistics & Facts." Retrieved from Statista.
- Kenya National Bureau of Statistics. (2022). "Economic Survey 2022." Retrieved from KNBS.
- Vogue Business. (2023). "The Future of AI in Fashion." Retrieved from Vogue Business.
- Elle. (2024). "Alessandro Michele on AI and Creativity." Retrieved from Elle Magazine.
- Harper's Bazaar. (2023). "Supermodels on AI in Fashion." Retrieved from Harper's Bazaar.