Fashion MNIST Image Classification and Deployment

Data Preprocessing and Feature Engineering

1. **Loading the Dataset**:
    - The Fashion MNIST dataset was loaded using `tf.keras.datasets.fashion_mnist.load_data()`.
    - The dataset contains grayscale images of fashion items.
2. **Normalization**:
    - Pixel values were normalized to the range [0, 1] using division by 255.0.
3. **Reshaping**:
    - Images were reshaped to add a channel dimension (28x28x1) for grayscale images.
4. **Resizing**:
    - Images were resized to 32x32 for compatibility with MobileNetV2 using `tf.image.resize()`.
5. **Conversion to RGB**:
    - Grayscale images were converted to RGB by repeating the single channel across three channels.
6. **Splitting Data**:
    - The training data was split into training and validation sets using `train_test_split()`.
7. **Data Augmentation**:
    - Applied data augmentation techniques such as rotation, width/height shifts, horizontal flip, and zoom using `ImageDataGenerator`.

Model Selection and Optimization Approach

1. **Model Architecture**:

   The model architecture utilized in the provided code is based on MobileNet. MobileNet is a lightweight and efficient convolutional neural network designed for mobile and embedded vision applications. Below is a summary of the model architecture:

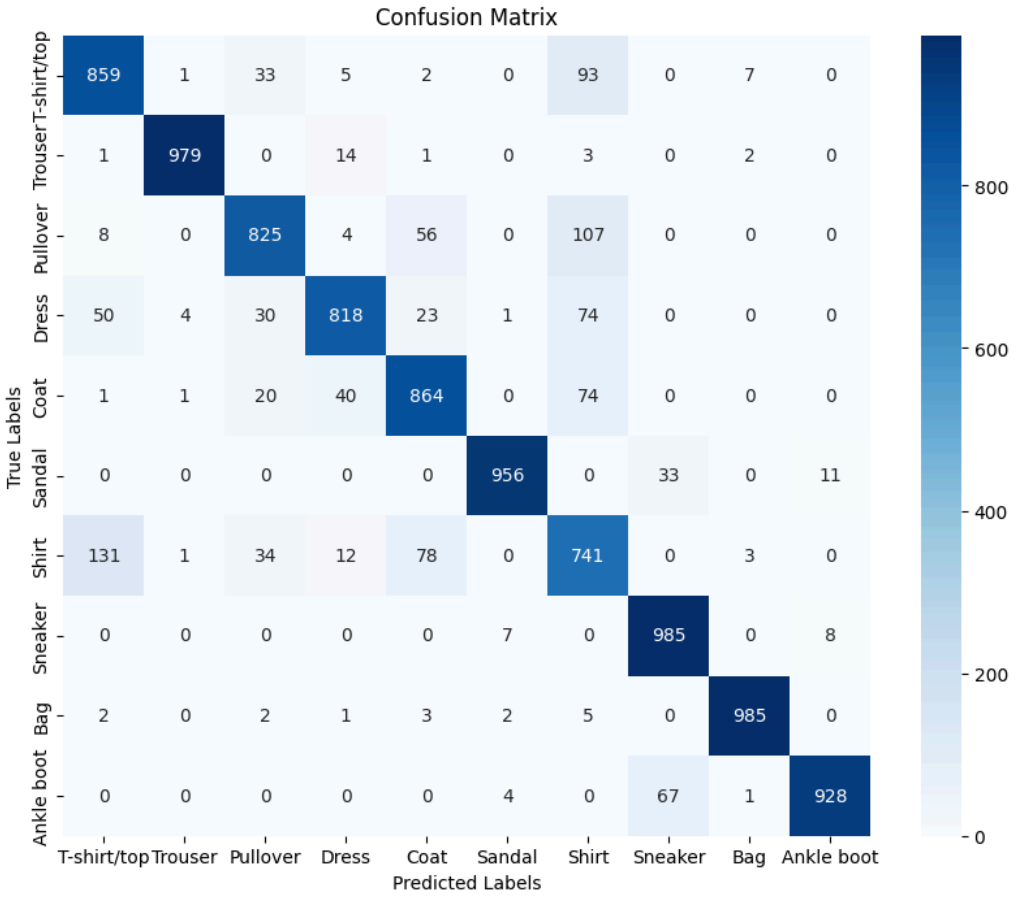
    1. **Input Layer**:
        - The input layer accepts images of shape `(32, 32, 3)`.
    2. **Base Model (MobileNet)**:
        - MobileNet is loaded with pre-trained weights on the ImageNet dataset.
        - `include_top=False` is used to exclude the top classification layer.
    3. **Additional Layers**:
        - **Flatten**: Flatten the output of the base model to a 1D array.
        - **Dense**: Fully connected dense layer with 256 units and ReLU activation.

    - **Output Layer**: Fully connected dense layer with 10 units and softmax activation for predicting the classes of Fashion MNIST items.
2. **Optimizer**:
    - Compiled the model with the Adam optimizer and a custom learning rate of 0.001.
3. **Loss Function**:
    - Used `sparse_categorical_crossentropy` as the loss function.
4. **Early Stopping**:
    - Applied early stopping to monitor validation loss and restore the best weights if no improvement is seen in 5 epochs.
5. **Training**:
    - Trained the model for 20 epochs with data augmentation.
6. **Evaluation**:
    - Evaluated the model on the test set and achieved a test accuracy of approximately 89.4%.

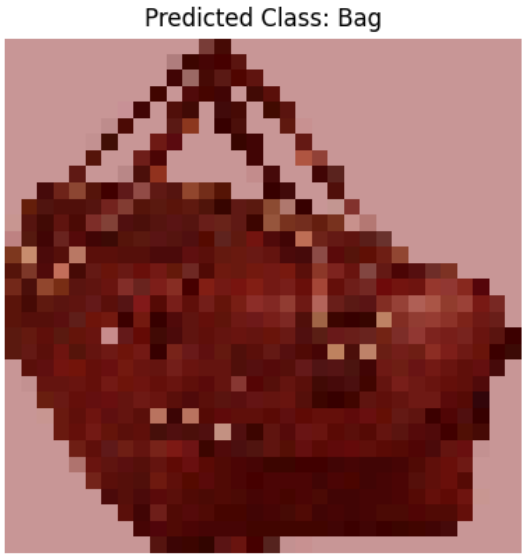
7. **Classification Report:**

Classes	Precision	Recall	F1-Score	Support
T-shirt/top	0.82	0.86	0.84	1000
Trouser	0.99	0.98	0.99	1000
Pullover	0.87	0.82	0.85	1000
Dress	0.91	0.82	0.86	1000
Coat	0.84	0.86	0.85	1000
Sandal	0.99	0.96	0.97	1000
Shirt	0.68	0.74	0.71	1000
Sneaker	0.91	0.98	0.94	1000
Bag	0.99	0.98	0.99	1000
Ankle boot	0.98	0.93	0.95	1000
Accuracy			0.89	10000
Macro avg	0.9	0.89	0.89	10000
Weighted avg	0.9	0.89	0.89	10000

8. **Confusion Matrix:**



9. **Grad-CAM Visualization for test_images/test1.png**



Deployment Strategy and API Usage Guide

1. **Model Saving**:
    - Saved the trained model using `model.save("fashion_mnist_mobilenet.h5")`.
2. **Evaluation Script**:
    - Created an evaluation script to load the model and test data, predict classes, generate a classification report, and plot a confusion matrix.
3. **API Usage with FastAPI Backend**:
    - **Endpoint**: `/predict`
    - **Method**: `POST`
    - **Authentication**: Basic authentication with username and password
    - **Request Body**: Multipart form-data with the following fields:
        - `file`: The image file to be uploaded for prediction
    - **Response**: JSON object with the predicted class and confidence score

Backend API with FastAPI

The FastAPI backend includes endpoints for user registration, sign-in, and image prediction. Below is the summary of each endpoint:

1. **Register Endpoint**:
    - **Endpoint**: `/register`
    - **Method**: `POST`
    - **Request Body**: JSON object with `username` and `password`
    - **Response**: Message indicating successful registration or error details
2. **Sign-In Endpoint**:
    - **Endpoint**: `/signin`
    - **Method**: `POST`
    - **Request Body**: Basic authentication with `username` and `password`
    - **Response**: Message indicating successful sign-in or error details
3. **Predict Endpoint**:
    - **Endpoint**: `/predict`
    - **Method**: `POST`
    - **Request Body**: Multipart form-data with the `file` field for the image
    - **Response**: JSON object with the predicted class and confidence score

Curl Test Results for Following Endpoints

Register Endpoint

Request Code:

powershell

```
$headers = @{
    "Content-Type" = "application/json"
}

$body = @{
    "username" = "testuser"
    "password" = "Test@1234"
} | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:8000/register" -Method POST -Headers $headers -Body $body
```

Response:

plaintext

```
StatusCode        : 200
StatusDescription : OK
Content           : {"message":"User registered successfully!"}
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 43
                    Content-Type: application/json
                    Date: Wed, 12 Mar 2025 11:56:44 GMT
```

Server: uvicorn

{"message":"User registered successfully!"}

Forms : {}

Headers : {[Content-Length, 43], [Content-Type, application/json], [Date, Wed, 12 Mar 2025 11:56:44 GMT], [Server, uvicorn]}

Images : {}

InputFields : {}

Links : {}

ParsedHtml : System.__ComObject

RawContentLength : 43

Sign-In Endpoint

Request Code:

powershell

```
$credPair = [Convert]::ToBase64String([Text.Encoding]::ASCII.GetBytes("testuser:Test@1234"))
$headers = @{
    "Authorization" = "Basic $credPair"
}

Invoke-WebRequest -Uri "http://localhost:8000/signin" -Method POST -Headers $headers
```

Response:

plaintext

StatusCode : 200

StatusDescription : OK

Content : {"message":"User signed in successfully!"}

RawContent : HTTP/1.1 200 OK

Content-Length: 42

Content-Type: application/json

Date: Wed, 12 Mar 2025 11:58:31 GMT

Server: uvicorn

{"message":"User signed in successfully!"}

Forms : {}

Headers : {[Content-Length, 42], [Content-Type, application/json], [Date, Wed, 12 Mar 2025 11:58:31 GMT], [Server, uvicorn]}

Images : {}

InputFields : {}

Links : {}

ParsedHtml : System.__ComObject

RawContentLength : 42

Predict Endpoint

Request Code:

powershell

```
$credPair = [Convert]::ToBase64String([Text.Encoding]::ASCII.GetBytes("testuser:Test@1234"))

curl.exe -X POST "http://localhost:8000/predict" -H "Authorization: Basic $credPair" -F "file=@C:\Projects\ImageClassification_DeccanAI\test_images\test1.png"
```

Response:

plaintext

{"class":"Bag","confidence":0.7172699570655823}

These commands will help you test the FastAPI endpoints using PowerShell and `curl`

Streamlit Frontend Usage

The Streamlit frontend provides a user interface for registering, signing in, and uploading images for prediction. Below is the usage guide:

1. **Register**:
    - Open the `Register` page.
    - Enter a valid `username` and `password`.
    - Click the `Register` button to create a new user account.
2. **Sign In**:
    - Open the `Sign In` page.
    - Enter the registered `username` and `password`.
    - Click the `Sign In` button to authenticate the user.
3. **Predict**:
    - After signing in, open the `Predict` page.
    - Upload an image file (`png`, `jpg`, `jpeg`).
    - Click the `Predict` button to send the image to the backend for prediction.
    - The predicted class and confidence score will be displayed on the page.