Title of the Assignment: Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

```
import numpy as np
import matplotlib.pyplot as plt
import gzip
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical

# Step 1: Load the MNIST Fashion dataset
def load_mnist_images(filename):
    with gzip.open(filename, 'rb') as f:
        data = np.frombuffer(f.read(), np.uint8, offset=16)
    return data.reshape(-1, 28, 28)  # Reshape to (num_images, 28, 28)

def load_mnist_labels(filename):
    with gzip.open(filename, 'rb') as f:
        data = np.frombuffer(f.read(), np.uint8, offset=8)
    return data

# Paths to the downloaded dataset files

train_images_path = 'train-images-idx3-ubyte.gz'
train_labels_path = 'train-labels-idx1-ubyte.gz'

test_images_path = 't10k-images-idx3-ubyte.gz'
test_labels_path = 't10k-labels-idx1-ubyte.gz'

# Load the training and testing images and labels
train_images = load_mnist_images(train_images_path)
train_labels = load_mnist_labels(train_labels_path)
test_images = load_mnist_images(test_images_path)
test_labels = load_mnist_labels(test_labels_path)

# Step 2: Preprocess the data
train_images = train_images / 255.0
test_images = test_images / 255.0

# Step 3: Split the data (not necessary if already split)
# No need to split as it's already split in the dataset

# Step 4: Build the model
model = Sequential([
    Flatten(input_shape=(28, 28)),  # Flatten the 28x28 images into a 1D array
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
```

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

# Step 5: Train the model
model.fit(train_images, train_labels, epochs=10, batch_size=128,
validation_split=0.2)

# Step 6: Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
print("Test Accuracy:", test_accuracy)
```

OUTPUT:

```
Epoch 1/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.7359 - loss: 0.7945 - val_accuracy: 0.8307 - val_loss: 0.4739
Epoch 2/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8511 - loss: 0.4293 - val_accuracy: 0.8470 - val_loss: 0.4311
Epoch 3/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8609 - loss: 0.3864 - val_accuracy: 0.8704 - val_loss: 0.3683
Epoch 4/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8747 - loss: 0.3515 - val_accuracy: 0.8620 - val_loss: 0.3828
Epoch 5/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8845 - loss: 0.3194 - val_accuracy: 0.8649 - val_loss: 0.3738
Epoch 6/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8865 - loss: 0.3104 - val_accuracy: 0.8763 - val_loss: 0.3454
Epoch 7/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8928 - loss: 0.2917 - val_accuracy: 0.8726 - val_loss: 0.3551
Epoch 8/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8976 - loss: 0.2841 - val_accuracy: 0.8806 - val_loss: 0.3289
Epoch 9/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9012 - loss: 0.2690 - val_accuracy: 0.8858 - val_loss: 0.3283
Epoch 10/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9062 - loss: 0.2590 - val_accuracy: 0.8866 - val_loss: 0.3202
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8783 - loss: 0.3435
Test Accuracy: 0.8791000247001648
```