```
import tensorflow as tf  # --> Imports TensorFlow library for deep learning
import matplotlib.pyplot as plt  # --> Imports matplotlib for data visualization
from tensorflow import keras  # --> Imports Keras module from TensorFlow
import numpy as np  # --> Imports NumPy for numerical operations


# Load the Fashion MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()  # --> Loads training and test data from Fashion MNIST dataset


# There are 10 image classes in this dataset and each class has a mapping corresponding to the following labels:
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',  # --> Defines labels for each class index in the dataset
            'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']


# Visualize some samples
plt.figure(figsize=(10,10))  # --> Creates a figure of size 10x10 for plotting
for i in range(9):  # --> Loops through the first 9 training images
   plt.subplot(3,3,i+1)  # --> Creates a 3x3 grid of subplots
   plt.imshow(x_train[i], cmap='gray')  # --> Displays the i-th image in grayscale
   plt.title(class_names[y_train[i]])  # --> Sets title of the image based on its label
   plt.axis('off')  # --> Hides axis ticks for better visualization
plt.show()  # --> Displays the plotted images


# Preprocess the data by scaling the pixel values to be between 0 and 1, and reshaping the images
x_train = x_train.astype('float32') / 255.0  # --> Converts training images to float and scales pixel values to [0, 1]
x_test = x_test.astype('float32') / 255.0  # --> Converts test images to float and scales pixel values to [0, 1]
x_train = x_train.reshape(-1, 28, 28, 1)  # --> Reshapes training data to include channel dimension
x_test = x_test.reshape(-1, 28, 28, 1)  # --> Reshapes test data to include channel dimension
```

```
# Print shapes to verify
print("Training data shape:", x_train.shape)  # --> Prints shape of training image data
print("Test data shape:", x_test.shape)  # --> Prints shape of test image data
print("Training labels shape:", y_train.shape)  # --> Prints shape of training labels
print("Test labels shape:", y_test.shape)  # --> Prints shape of test labels

# Create the CNN model
model = keras.Sequential([  # --> Initializes a sequential CNN model
    keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),  # --> Adds a 2D convolution layer with 32 filters and ReLU activation
    keras.layers.MaxPooling2D((2,2)),  # --> Adds a 2D max pooling layer with 2x2 window
    keras.layers.Dropout(0.25),  # --> Adds dropout to reduce overfitting

    keras.layers.Conv2D(64, (3,3), activation='relu'),  # --> Adds a second convolutional layer with 64 filters
    keras.layers.MaxPooling2D((2,2)),  # --> Adds another max pooling layer
    keras.layers.Dropout(0.25),  # --> Adds another dropout layer

    keras.layers.Conv2D(128, (3,3), activation='relu'),  # --> Adds a third convolutional layer with 128 filters

    keras.layers.Flatten(),  # --> Flattens the 3D output to 1D for dense layers
    keras.layers.Dense(128, activation='relu'),  # --> Adds a fully connected layer with 128 units
    keras.layers.Dropout(0.25),  # --> Adds dropout before output layer
    keras.layers.Dense(10, activation='softmax')  # --> Adds output layer with 10 units and softmax activation for classification
])

model.summary()  # --> Prints model architecture and parameter summary

# Compile the model
```

```
model.compile(optimizer='adam',  # --> Sets Adam as the optimizer
       loss='sparse_categorical_crossentropy',  # --> Uses sparse categorical crossentropy as loss function
       metrics=['accuracy'])  # --> Tracks accuracy during training

# Train the model
history = model.fit(x_train, y_train,  # --> Trains the model on training data
          epochs=10,  # --> Sets number of training epochs to 10
          validation_data=(x_test, y_test))  # --> Validates model on test data after each epoch

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)  # --> Evaluates model performance on test data
print('Test accuracy:', test_acc)  # --> Prints the test accuracy

# Plot training history
plt.figure(figsize=(12, 4))  # --> Creates a figure of size 12x4 for plots
plt.subplot(1, 2, 1)  # --> Creates first subplot for accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')  # --> Plots training accuracy
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')  # --> Plots validation accuracy
plt.xlabel('Epoch')  # --> Labels x-axis as Epoch
plt.ylabel('Accuracy')  # --> Labels y-axis as Accuracy
plt.legend()  # --> Adds legend to the plot

plt.subplot(1, 2, 2)  # --> Creates second subplot for loss
plt.plot(history.history['loss'], label='Training Loss')  # --> Plots training loss
plt.plot(history.history['val_loss'], label='Validation Loss')  # --> Plots validation loss
plt.xlabel('Epoch')  # --> Labels x-axis as Epoch
plt.ylabel('Loss')  # --> Labels y-axis as Loss
plt.legend()  # --> Adds legend to the plot
plt.show()  # --> Displays the plots
```