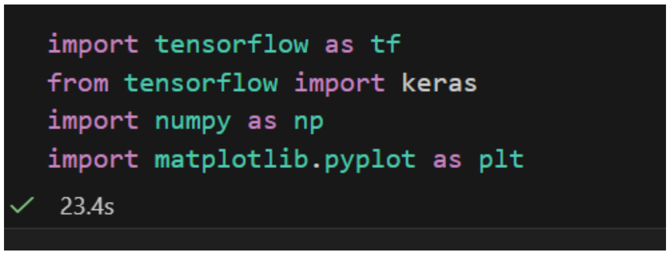
DOCUMENTATION

ARTIFICIAL NUERAL NETWORK

Artificial Neural Networks (ANNs) are computing systems inspired by the way the human brain processes information. They consist of interconnected units called neurons, which are organized into layers: an input layer that receives data, one or more hidden layers that process the information, and an output layer that produces the final result. Each connection between neurons has an associated weight, and a bias is added to help the model learn patterns more effectively. By applying activation functions and adjusting weights through a learning process called backpropagation, ANNs can identify complex relationships in data. They are widely used in applications such as image recognition, speech processing, medical diagnosis, and financial forecasting due to their ability to learn from experience and improve performance over time.

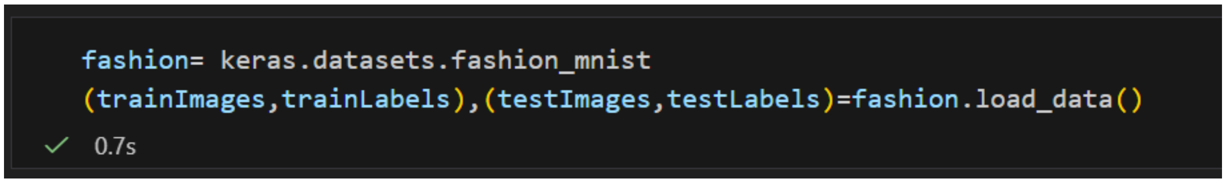
Import Required Libraries



```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
```
✓ 23.4s

This code imports the essential Python libraries commonly used in machine learning and deep learning projects. `tensorflow as tf` loads TensorFlow, the core framework used to build and train neural networks, while `from tensorflow import keras` imports Keras, a high-level API within TensorFlow that simplifies creating and managing deep learning models. numpy as np is included for efficient numerical computations, especially for handling arrays, matrices, and mathematical operations on data. Finally, `matplotlib.pyplot as plt` is imported to visualize data, model performance, and results through graphs such as line plots and loss/accuracy curves. Together, these libraries form the basic setup for developing, training, and evaluating neural network models.

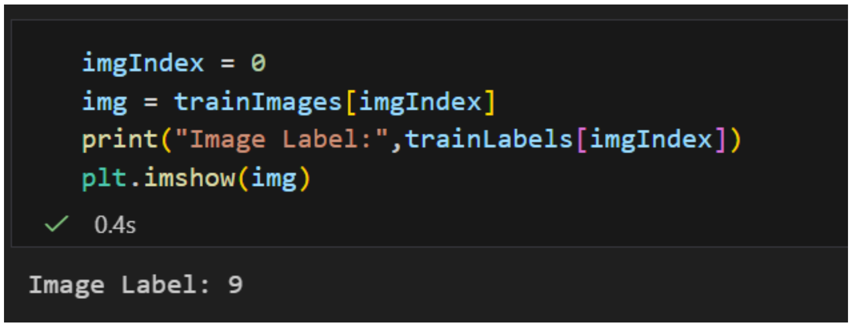
Loading the Fashion MNIST Dataset



```
fashion= keras.datasets.fashion_mnist
(trainImages,trainLabels),(testImages,testLabels)=fashion.load_data()
```
✓ 0.7s

This code loads the Fashion MNIST dataset using Keras, which is commonly used for training and testing Artificial Neural Networks in image classification tasks. The statement `keras.datasets.fashion_mnist` accesses the dataset, and the `load_data()` function automatically downloads it (if needed) and splits it into two parts: training data and testing data. `trainImages` contains the input images used to train the model, while `trainLabels` stores the corresponding class labels. Similarly, `testImages` and `testLabels` are used to evaluate how well the trained model performs on unseen data. This clear separation of training and testing datasets helps ensure that the ANN learns general patterns rather than simply memorizing the data.

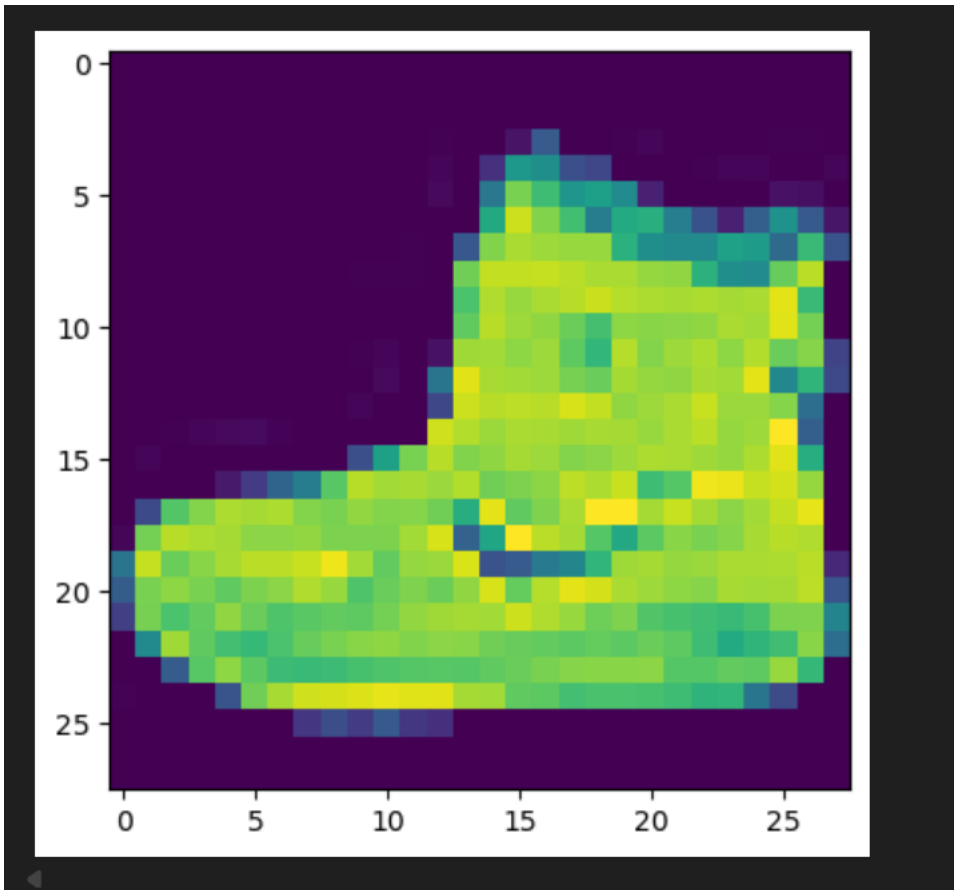
Displaying a Sample Training Image and Its Label



```
imgIndex = 0
img = trainImages[imgIndex]
print("Image Label:",trainLabels[imgIndex])
plt.imshow(img)
```

✓ 0.4s

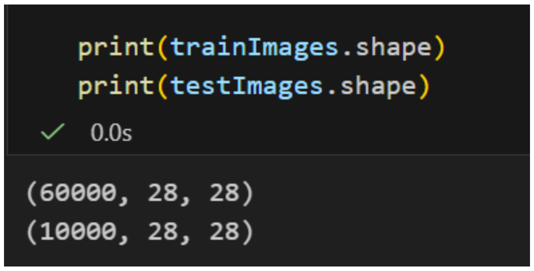
```
Image Label: 9
```



This code is used to display a single image from the training dataset and show its corresponding label. The variable `imgIndex` is set to `0`, which means the first image in the `trainImages` array is selected. That image is stored in the variable `img`, and its associated class label is accessed from `trainLabels` using the same index and printed to the output. The function `plt.imshow(img)` then visualizes the selected image using

Matplotlib. The printed result `Image Label: 9` indicates the numeric class assigned to that image in the Fashion MNIST dataset, which represents a specific clothing category. This step helps in understanding and verifying the data before training an Artificial Neural Network.

Understanding the Shape of Training and Testing Images

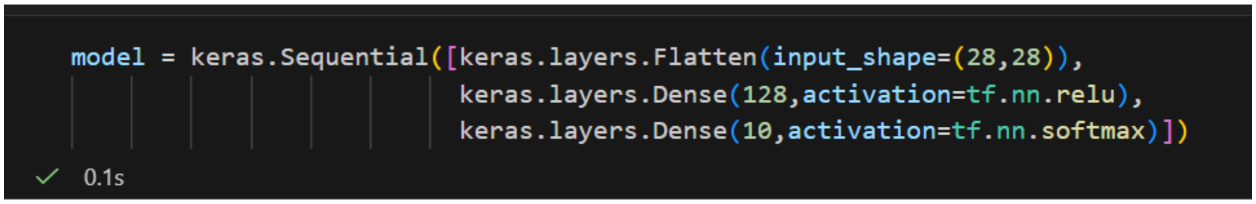


```
print(trainImages.shape)
print(testImages.shape)
✓ 0.0s

(60000, 28, 28)
(10000, 28, 28)
```

This code prints the dimensions (shape) of the training and testing image datasets. The output `(60000, 28, 28)` indicates that the training set contains 60,000 images, each with a height and width of 28×28 pixels. Similarly, `(10000, 28, 28)` shows that the test set contains 10,000 images, also of size 28×28 pixels. These images are stored as grayscale images, which is why there is no separate color channel dimension. Knowing the shape of the data is important because it helps in preparing the images correctly—such as reshaping or flattening them—before feeding them into an Artificial Neural Network for training and evaluation.

Building an Artificial Neural Network Model Using Keras

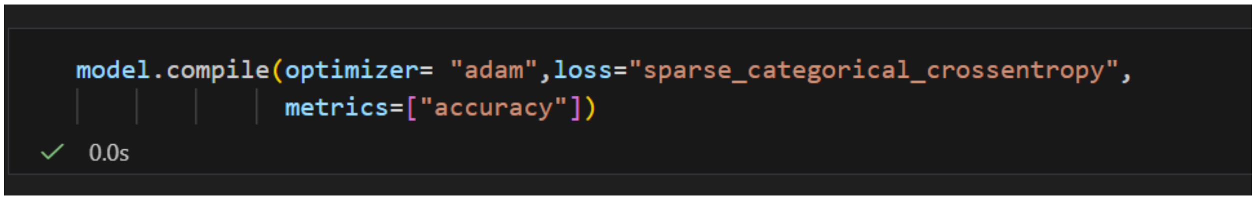


```
model = keras.Sequential([keras.layers.Flatten(input_shape=(28,28)),
                          keras.layers.Dense(128,activation=tf.nn.relu),
                          keras.layers.Dense(10,activation=tf.nn.softmax)])
✓ 0.1s
```

This code defines an Artificial Neural Network model using Keras' `Sequential` API, which creates a linear stack of layers. The first layer, `Flatten`, converts each 28×28 input image into a one-dimensional array so it can be processed by dense layers. The second layer is a Dense hidden layer with 128 neurons and the ReLU (Rectified Linear

Unit) activation function, which introduces non-linearity and helps the network learn complex patterns from the data. The final Dense layer has 10 neurons with the Softmax activation function, producing probability scores for each of the 10 clothing categories in the Fashion MNIST dataset. Together, these layers form a simple yet effective neural network for multi-class image classification.

Compiling the Neural Network Model

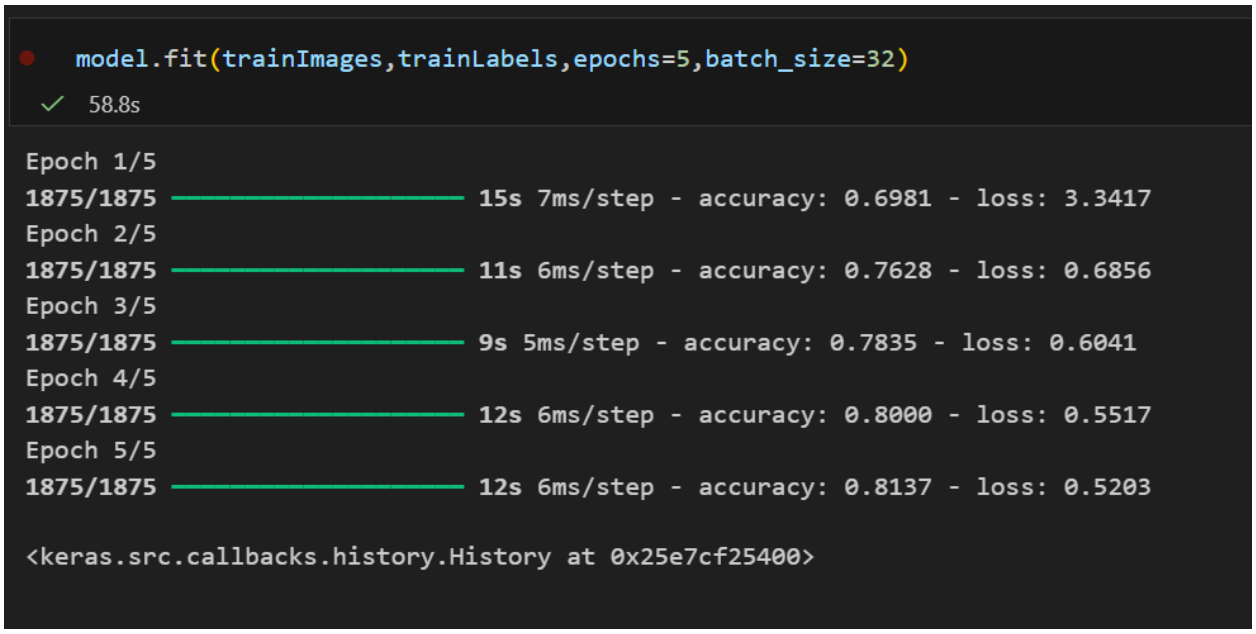


```
model.compile(optimizer= "adam",loss="sparse_categorical_crossentropy",
                            metrics=["accuracy"])
```

```
✓ 0.0s
```

This code compiles the neural network by specifying how the model should learn and how its performance should be evaluated. The `optimizer="adam"` sets the Adam optimization algorithm, which efficiently adjusts the model's weights during training to minimize error. The `loss="sparse_categorical_crossentropy"` defines the loss function used for multi-class classification when the class labels are provided as integers rather than one-hot encoded vectors. The `metrics=["accuracy"]` parameter tells the model to measure accuracy during training and testing, indicating how many predictions are correct. Compiling the model is a crucial step because it prepares the network for training by defining the learning strategy and evaluation criteria.

Training the Neural Network Model



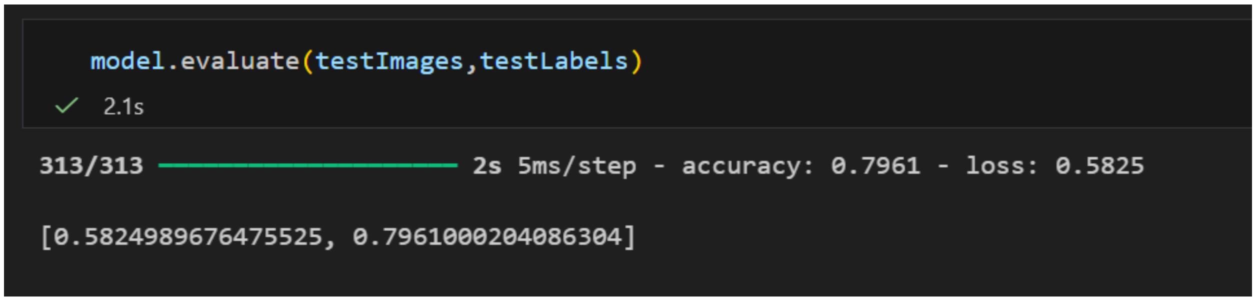
```
model.fit(trainImages,trainLabels,epochs=5,batch_size=32)
✓ 58.8s

Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 15s 7ms/step - accuracy: 0.6981 - loss: 3.3417
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.7628 - loss: 0.6856
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.7835 - loss: 0.6041
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 6ms/step - accuracy: 0.8000 - loss: 0.5517
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 6ms/step - accuracy: 0.8137 - loss: 0.5203

<keras.src.callbacks.history.History at 0x25e7cf25400>
```

This code trains the neural network using the training images and their corresponding labels. The `model.fit()` function starts the learning process, where the model iterates over the entire training dataset 5 times (epochs) and updates its weights in small groups of 32 samples (batch size) at a time. During each epoch, the model reports performance metrics such as accuracy and loss, showing how well it is learning. As seen in the output, the accuracy steadily improves from about 69.8% to 81.4%, while the loss decreases, indicating that the model is learning patterns more effectively with each epoch. The returned `History` object stores these training metrics and can later be used to visualize or analyze the model's learning progress.

Evaluating the Model on Test Data



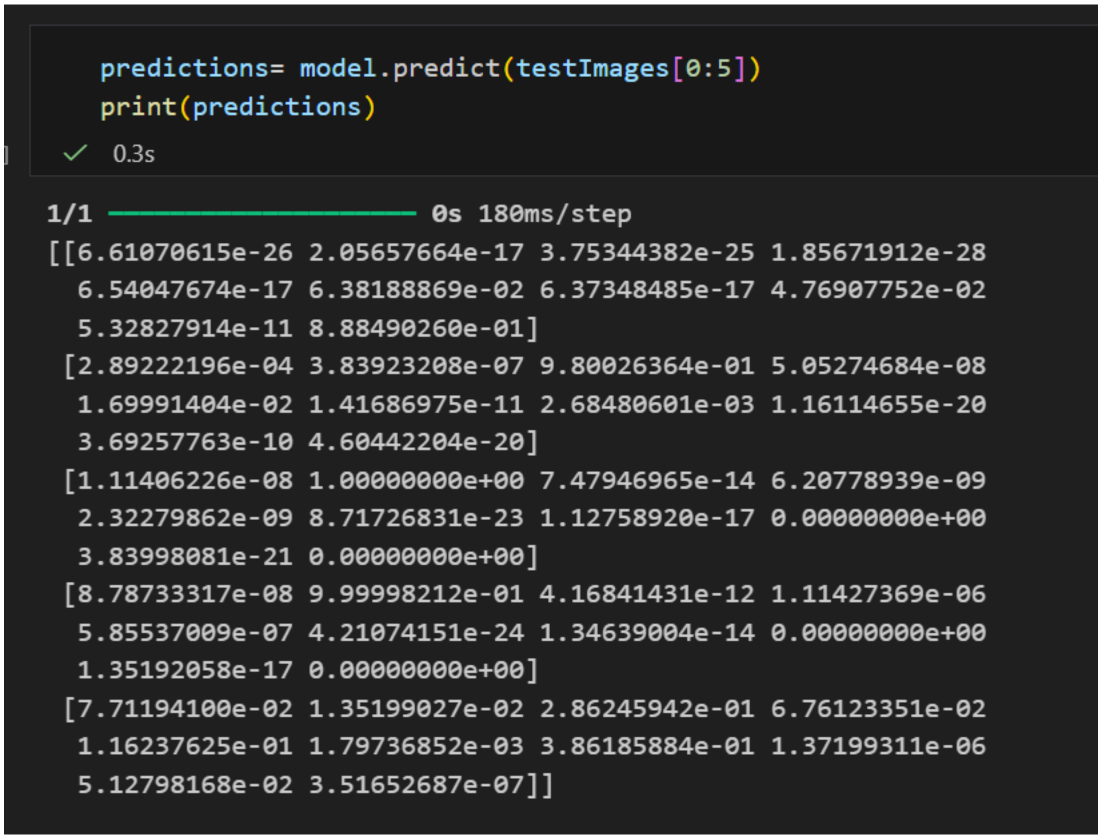
```
model.evaluate(testImages,testLabels)
✓ 2.1s

313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.7961 - loss: 0.5825

[0.5824989676475525, 0.7961000204086304]
```

This code evaluates the trained neural network using the test dataset, which contains images the model has never seen before. The `model.evaluate()` function computes the loss and accuracy based on the test images and their true labels. In the output, the value `0.5825` represents the test loss, indicating how much error the model makes on unseen data, while `0.7961` (about 79.6% accuracy) shows that the model correctly classifies nearly 80% of the test images. This evaluation step is important because it helps assess how well the model generalizes to new data and whether it is performing reliably beyond the training dataset.

Generating Predictions for Test Images

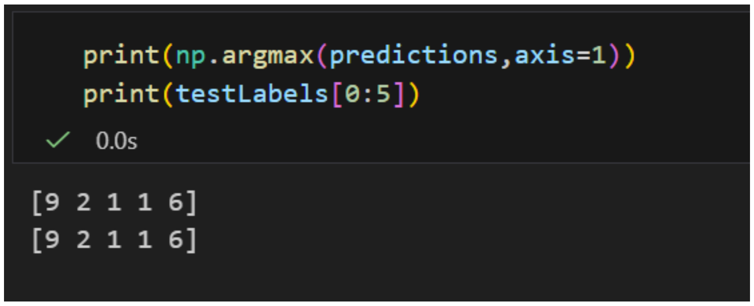


```
predictions= model.predict(testImages[0:5])
print(predictions)
✓ 0.3s

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 180ms/step
[[6.61070615e-26 2.05657664e-17 3.75344382e-25 1.85671912e-28
  6.54047674e-17 6.38188869e-02 6.37348485e-17 4.76907752e-02
  5.32827914e-11 8.88490260e-01]
 [2.89222196e-04 3.83923208e-07 9.80026364e-01 5.05274684e-08
  1.69991404e-02 1.41686975e-11 2.68480601e-03 1.16114655e-20
  3.69257763e-10 4.60442204e-20]
 [1.11406226e-08 1.00000000e+00 7.47946965e-14 6.20778939e-09
  2.32279862e-09 8.71726831e-23 1.12758920e-17 0.00000000e+00
  3.83998081e-21 0.00000000e+00]
 [8.78733317e-08 9.99998212e-01 4.16841431e-12 1.11427369e-06
  5.85537009e-07 4.21074151e-24 1.34639004e-14 0.00000000e+00
  1.35192058e-17 0.00000000e+00]
 [7.71194100e-02 1.35199027e-02 2.86245942e-01 6.76123351e-02
  1.16237625e-01 1.79736852e-03 3.86185884e-01 1.37199311e-06
  5.12798168e-02 3.51652687e-07]]
```

This code uses the trained neural network to generate predictions for the first five images in the test dataset. The `model.predict()` function outputs a set of probability values for each image, where each value corresponds to one of the 10 possible clothing classes in the Fashion MNIST dataset. The `predictions` array therefore contains five rows, one for each test image, and each row has 10 probability scores that sum to 1. By printing these predictions, we can see how confident the model is about each class and which class it considers most likely for each image. This step is useful for understanding the model's behavior and interpreting its classification results in more detail.

Comparing Predicted Classes with Actual Labels



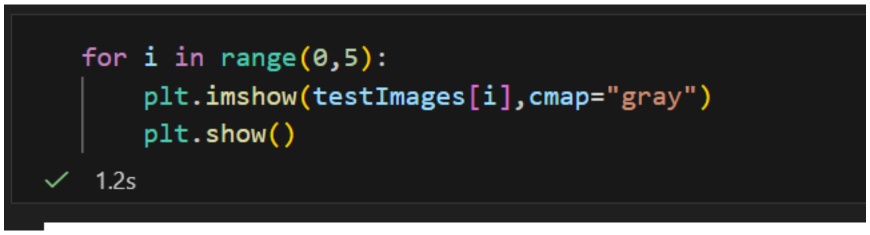
```
print(np.argmax(predictions,axis=1))
print(testLabels[0:5])
```

```
✓ 0.0s
```

```
[9 2 1 1 6]
[9 2 1 1 6]
```

This code compares the model's predicted classes with the true labels for the first five test images. The function `np.argmax(predictions, axis=1)` selects the index of the highest probability from each prediction array, converting the probability outputs into final class labels. The second line prints the actual labels from `testLabels` for the same five images. Since both outputs are identical—`[9 2 1 1 6]`—it shows that the model has correctly classified all five test samples. This comparison helps verify the accuracy of the model's predictions at an individual image level and builds confidence in its performance.

Displaying Test Images Using a Loop



```
for i in range(0,5):
    plt.imshow(testImages[i],cmap="gray")
    plt.show()
```

```
✓ 1.2s
```

This code uses a `for` loop to display the first five images from the test dataset one by one. The loop runs from index 0 to 4, and in each iteration `plt.imshow(testImages[i], cmap="gray")` visualizes the selected image in grayscale, which matches the original format of the Fashion MNIST images. The `plt.show()` function is called inside the loop so that each image is displayed separately. This step is useful for visually inspecting the test images and comparing them with the model's predictions to better understand how the neural network interprets different clothing items.