```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np


# Load the Fashion MNIST dataset
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.fashion_mnist.load_data()

# Normalize the pixel values to be between 0 and 1
train_images = train_images / 255.0
test_images = test_images / 255.0

# Reshape data 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))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
model = models.Sequential()

# First convolutional layer
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())

# Second convolutional layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())

# Third convolutional layer
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.BatchNormalization())

# Flatten the output and add a fully connected layer
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))  # Regularization to avoid overfitting

# Output layer with 10 classes (for the 10 clothing categories)
model.add(layers.Dense(10, activation='softmax'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

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



history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

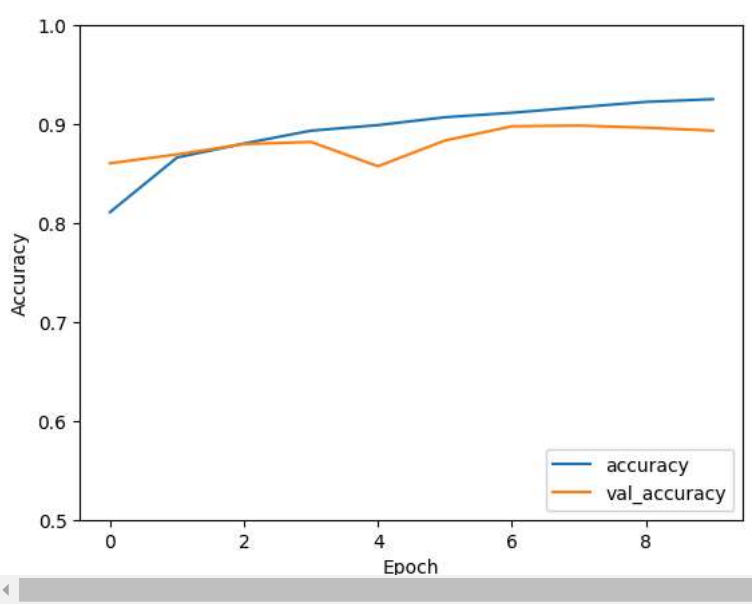
```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 74s 37ms/step - accuracy: 0.7580 - loss: 0.6977 - val_accuracy: 0.8604 - val_loss: 0.3806
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 79s 36ms/step - accuracy: 0.8624 - loss: 0.3835 - val_accuracy: 0.8694 - val_loss: 0.3521
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 67s 35ms/step - accuracy: 0.8782 - loss: 0.3339 - val_accuracy: 0.8798 - val_loss: 0.3289
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 81s 35ms/step - accuracy: 0.8925 - loss: 0.2980 - val_accuracy: 0.8820 - val_loss: 0.3277
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 83s 36ms/step - accuracy: 0.8987 - loss: 0.2764 - val_accuracy: 0.8574 - val_loss: 0.4171
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 84s 37ms/step - accuracy: 0.9083 - loss: 0.2541 - val_accuracy: 0.8834 - val_loss: 0.3217
```

```
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 69s 37ms/step - accuracy: 0.9135 - loss: 0.2357 - val_accuracy: 0.8978 - val_loss: 0.2954
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 80s 35ms/step - accuracy: 0.9184 - loss: 0.2198 - val_accuracy: 0.8985 - val_loss: 0.3005
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 83s 36ms/step - accuracy: 0.9219 - loss: 0.2113 - val_accuracy: 0.8964 - val_loss: 0.2942
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 68s 36ms/step - accuracy: 0.9270 - loss: 0.1979 - val_accuracy: 0.8935 - val_loss: 0.3095
```

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.8955 - loss: 0.3188
Test accuracy: 0.8934999704360962
```

```
# Plot training & validation accuracy
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
plt.show()
```




```
# Define class names for Fashion MNIST dataset
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

```
# Function to plot 10 random images with their predictions
def plot_random_images_and_classify(model, images, labels):
    # Select 10 random indices
    random_indices = np.random.choice(images.shape[0], 10, replace=False)

    plt.figure(figsize=(10, 10))

    for i, idx in enumerate(random_indices):
        # Select the image and the corresponding label
        img = images[idx]
        true_label = labels[idx]

        # Reshape and expand dimensions for prediction
        img_reshaped = np.expand_dims(img, axis=0)

        # Predict the class of the image
        prediction = model.predict(img_reshaped)
        predicted_label = np.argmax(prediction)

        # Plot the image
        plt.subplot(5, 5, i+1)
        plt.xticks([])
        plt.yticks([])
```

```
            plt.grid(False)

            # Plot the image in grayscale
            plt.imshow(img.squeeze(), cmap=plt.cm.binary)

            # Set the title: Predicted label vs True label
            plt.title(f"Predicted: {class_names[predicted_label]}\nTrue: {class_names[true_label]}",
                      color='green' if predicted_label == true_label else 'red')

    plt.tight_layout()
    plt.show()


# Test the function with the test dataset
plot_random_images_and_classify(model, test_images, test_labels)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 318ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
```

Start coding or generate with AI.