Using fashoin_mnist dataset and training the model with SimpleRNN.

```python
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models

(x_train, y_train), (x_test, y_test) = datasets.fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test /255.0

model = models.Sequential([
    layers.SimpleRNN(64, input_shape=(28,28), activation='relu'),
    layers.Dense(10, activation='softmax')
])

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

history = model.fit(x_train, y_train, epochs = 3, validation_data = (x_test, y_test), batch_size = 64)
test_loss, test_acc = model.evaluate(x_train, y_train, batch_size = 64)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [================================] - 0s 0us/step
40960/29515 [======================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
26435584/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
16384/5148 [===============================================================================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_1 (SimpleRNN)    (None, 64)                5952

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 6,602
Trainable params: 6,602
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
938/938 [==============================] - 9s 9ms/step - loss: 0.8554 - accuracy: 0.6805 - val_loss: 0.6252 - val_accuracy: 0.7804
Epoch 2/3
938/938 [==============================] - 8s 9ms/step - loss: 0.5704 - accuracy: 0.7990 - val_loss: 0.6056 - val_accuracy: 0.8002
Epoch 3/3
938/938 [==============================] - 8s 9ms/step - loss: 0.5054 - accuracy: 0.8218 - val_loss: 0.5479 - val_accuracy: 0.8019
```

Replace SimpleRNN with LSTM. The result is better than the preview's result.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 64)                23808

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 24,458
Trainable params: 24,458
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
938/938 [==============================] - 22s 22ms/step - loss: 0.7025 - accuracy: 0.7412 - val_loss: 0.4904 - val_accuracy: 0.8172
Epoch 2/3
938/938 [==============================] - 21s 22ms/step - loss: 0.4374 - accuracy: 0.8388 - val_loss: 0.4298 - val_accuracy: 0.8441
Epoch 3/3
938/938 [==============================] - 21s 22ms/step - loss: 0.3901 - accuracy: 0.8559 - val_loss: 0.4022 - val_accuracy: 0.8534
938/938 [==============================] - 9s 9ms/step - loss: 0.3674 - accuracy: 0.8625
```

Add more LSTM layers.

```python
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models

(x_train, y_train), (x_test, y_test) = datasets.fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test /255.0

model = models.Sequential([
    layers.LSTM(64, input_shape=(28,28), activation='relu', return_sequences = True),
    layers.LSTM(64, activation = 'relu'),
    layers.Dense(10, activation='softmax')
])

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

history = model.fit(x_train, y_train, epochs = 3, validation_data = (x_test, y_test), batch_size = 64)
test_loss, test_acc = model.evaluate(x_train, y_train, batch_size = 64)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 28, 64)            23808

 lstm_2 (LSTM)               (None, 64)                33024

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 57,482
Trainable params: 57,482
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
938/938 [==============================] - 44s 45ms/step - loss: 0.7053 - accuracy: 0.7400 - val_loss: 0.5401 - val_accuracy: 0.8059
Epoch 2/3
938/938 [==============================] - 42s 45ms/step - loss: 0.4432 - accuracy: 0.8347 - val_loss: 0.4516 - val_accuracy: 0.8272
Epoch 3/3
938/938 [==============================] - 42s 45ms/step - loss: 0.3867 - accuracy: 0.8548 - val_loss: 0.3927 - val_accuracy: 0.8559
938/938 [==============================] - 15s 16ms/step - loss: 0.3565 - accuracy: 0.8673
```

Add extra Dense layers. It helps push accuracy a little bit more.

```python
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models

(x_train, y_train), (x_test, y_test) = datasets.fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test /255.0

model = models.Sequential([
    layers.LSTM(64, input_shape=(28,28), activation='relu', return_sequences = True),
    layers.LSTM(64, activation = 'relu'),
    layers.Dense(32, activation = 'relu'),
    layers.Dense(10, activation='softmax')
])

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

history = model.fit(x_train, y_train, epochs = 3, validation_data = (x_test, y_test), batch_size = 64)
test_loss, test_acc = model.evaluate(x_train, y_train, batch_size = 64)
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (LSTM)               (None, 28, 64)            23808

 lstm_4 (LSTM)               (None, 64)                33024

 dense_4 (Dense)             (None, 32)                2080

 dense_5 (Dense)             (None, 10)                330

=================================================================
Total params: 59,242
Trainable params: 59,242
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
938/938 [==============================] - 45s 46ms/step - loss: 0.7112 - accuracy: 0.7472 - val_loss: 0.4932 - val_accuracy: 0.8121
Epoch 2/3
938/938 [==============================] - 41s 44ms/step - loss: 0.4314 - accuracy: 0.8410 - val_loss: 0.4300 - val_accuracy: 0.8462
Epoch 3/3
938/938 [==============================] - 41s 44ms/step - loss: 0.3793 - accuracy: 0.8596 - val_loss: 0.3879 - val_accuracy: 0.8569
938/938 [==============================] - 15s 16ms/step - loss: 0.3479 - accuracy: 0.8692
```

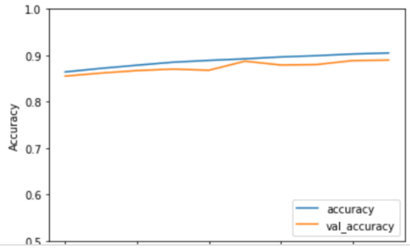
Add dropouts to avoid overfitting.

```python
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models

(x_train, y_train), (x_test, y_test) = datasets.fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test /255.0

model = models.Sequential([
    layers.LSTM(64, input_shape=(28,28), activation='relu', return_sequences = True),
    layers.LSTM(64, activation = 'relu', dropout = 0.2, recurrent_dropout = 0.2),
    layers.Dense(32, activation = 'relu'),
    layers.Dropout(rate = 0.1),
    layers.Dense(10, activation='softmax')
])

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

history = model.fit(x_train, y_train, epochs = 3, validation_data = (x_test, y_test), batch_size = 64)
test_loss, test_acc = model.evaluate(x_train, y_train, batch_size = 64)
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_5 (LSTM)               (None, 28, 64)            23808

 lstm_6 (LSTM)               (None, 64)                33024

 dense_6 (Dense)             (None, 32)                2080

 dropout (Dropout)           (None, 32)                0

 dense_7 (Dense)             (None, 10)                330

=================================================================
Total params: 59,242
Trainable params: 59,242
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
938/938 [==============================] - 74s 75ms/step - loss: 0.7887 - accuracy: 0.7030 - val_loss: 0.5164 - val_accuracy: 0.8058
Epoch 2/3
938/938 [==============================] - 69s 74ms/step - loss: 0.4937 - accuracy: 0.8208 - val_loss: 0.4575 - val_accuracy: 0.8313
Epoch 3/3
938/938 [==============================] - 70s 75ms/step - loss: 0.4441 - accuracy: 0.8392 - val_loss: 0.4269 - val_accuracy: 0.8455
938/938 [==============================] - 14s 15ms/step - loss: 0.4023 - accuracy: 0.8554
```

Visualize the accuracy.
The result without adding dropout layers.

```
Epoch 3/10
938/938 [==============================] - 43s 46ms/step - loss: 0.3240 - accuracy: 0.8787 - val_loss: 0.3473 - val_accuracy: 0.8671
Epoch 4/10
938/938 [==============================] - 42s 45ms/step - loss: 0.3079 - accuracy: 0.8851 - val_loss: 0.3447 - val_accuracy: 0.8703
Epoch 5/10
938/938 [==============================] - 42s 45ms/step - loss: 0.2962 - accuracy: 0.8891 - val_loss: 0.3440 - val_accuracy: 0.8678
Epoch 6/10
938/938 [==============================] - 43s 46ms/step - loss: 0.2851 - accuracy: 0.8924 - val_loss: 0.3117 - val_accuracy: 0.8873
Epoch 7/10
938/938 [==============================] - 42s 45ms/step - loss: 0.2765 - accuracy: 0.8965 - val_loss: 0.3222 - val_accuracy: 0.8791
Epoch 8/10
938/938 [==============================] - 42s 45ms/step - loss: 0.2686 - accuracy: 0.8992 - val_loss: 0.3158 - val_accuracy: 0.8801
Epoch 9/10
938/938 [==============================] - 43s 46ms/step - loss: 0.2586 - accuracy: 0.9028 - val_loss: 0.3006 - val_accuracy: 0.8886
Epoch 10/10
938/938 [==============================] - 42s 45ms/step - loss: 0.2524 - accuracy: 0.9050 - val_loss: 0.2965 - val_accuracy: 0.8897
938/938 [==============================] - 15s 16ms/step - loss: 0.2336 - accuracy: 0.9117
```



The result that added dropout layers.

```
history = model.fit(x_train, y_train, epochs = 10, validation_data = (x_test, y_test), batch_size = 64)
test_loss, test_acc = model.evaluate(x_train, y_train, batch_size = 64)


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()
```

```
Epoch 4/10
938/938 [==============================] - 70s 75ms/step - loss: 0.2470 - accuracy: 0.9069 - val_loss: 0.3004 - val_accuracy: 0.8944
Epoch 5/10
938/938 [==============================] - 71s 75ms/step - loss: 0.2397 - accuracy: 0.9102 - val_loss: 0.2988 - val_accuracy: 0.8964
Epoch 6/10
938/938 [==============================] - 70s 75ms/step - loss: 0.2341 - accuracy: 0.9118 - val_loss: 0.3000 - val_accuracy: 0.8921
Epoch 7/10
938/938 [==============================] - 71s 76ms/step - loss: 0.2302 - accuracy: 0.9136 - val_loss: 0.2920 - val_accuracy: 0.8941
Epoch 8/10
938/938 [==============================] - 70s 75ms/step - loss: 0.2230 - accuracy: 0.9157 - val_loss: 0.3013 - val_accuracy: 0.8935
Epoch 9/10
938/938 [==============================] - 71s 76ms/step - loss: 0.2218 - accuracy: 0.9174 - val_loss: 0.2922 - val_accuracy: 0.8963
Epoch 10/10
938/938 [==============================] - 70s 75ms/step - loss: 0.2210 - accuracy: 0.9173 - val_loss: 0.2873 - val_accuracy: 0.9006
938/938 [==============================] - 15s 15ms/step - loss: 0.1959 - accuracy: 0.9245
```