```
try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass
```

```
TensorFlow 2.x selected.
```

Thasina Tabashum, Farhad Mokter

```
from __future__ import absolute_import, division, print_function, unicode_literals

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

```
2.1.0
```

```
fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-
32768/29515 [==================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-l
8192/5148 [================================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-i
4423680/4422102 [==============================] - 0s 0us/step
```

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

```
train_images.shape

len(train_labels)
```

```
60000
```

Each label is an integer between 0 and 9:

```
train_labels
```

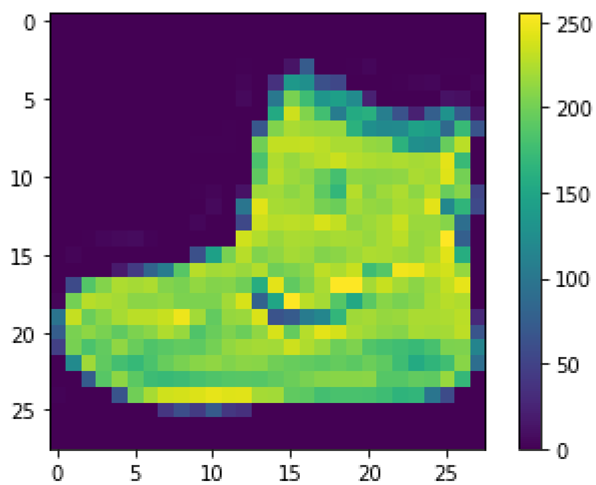
```
array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

```
test_images.shape
```

```
len(test_labels)
```

```
10000
```

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```




```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```


```
train_images = train_images / 255.0

test_images = test_images / 255.0


plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

Adding Regularizer

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='elu',
    kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    keras.layers.Dense(10, activation='relu')
])
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
demo =model.fit(train_images, train_labels, validation_split=0.33, epochs=10, verbose=2)
```

```
Train on 40199 samples, validate on 19801 samples
Epoch 1/10
40199/40199 - 5s - loss: 1.4310 - accuracy: 0.5847 - val_loss: 1.2688 - val_accuracy: 0.
Epoch 2/10
40199/40199 - 4s - loss: 1.2623 - accuracy: 0.6283 - val_loss: 1.3317 - val_accuracy: 0.
Epoch 3/10
40199/40199 - 4s - loss: 1.0898 - accuracy: 0.6466 - val_loss: 1.0761 - val_accuracy: 0.
Epoch 4/10
40199/40199 - 4s - loss: 1.0685 - accuracy: 0.6499 - val_loss: 1.0675 - val_accuracy: 0.
Epoch 5/10
40199/40199 - 4s - loss: 1.0596 - accuracy: 0.6520 - val_loss: 1.0733 - val_accuracy: 0.
Epoch 6/10
40199/40199 - 4s - loss: 1.0544 - accuracy: 0.6522 - val_loss: 1.0702 - val_accuracy: 0.
Epoch 7/10
40199/40199 - 4s - loss: 1.0485 - accuracy: 0.6529 - val_loss: 1.0879 - val_accuracy: 0.
Epoch 8/10
40199/40199 - 4s - loss: 1.0439 - accuracy: 0.6550 - val_loss: 1.0380 - val_accuracy: 0.
Epoch 9/10
40199/40199 - 4s - loss: 1.0425 - accuracy: 0.6539 - val_loss: 1.0397 - val_accuracy: 0.
Epoch 10/10
40199/40199 - 4s - loss: 1.0358 - accuracy: 0.6555 - val_loss: 1.0894 - val_accuracy: 0.
```

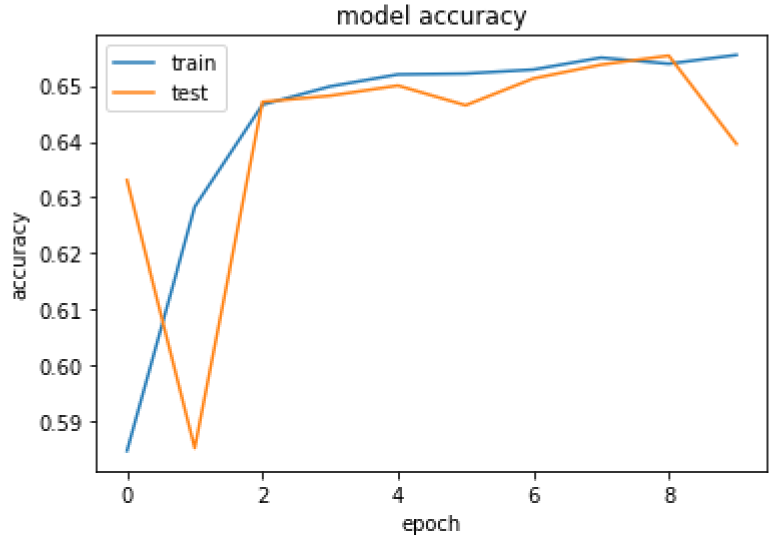
```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print('\nTest accuracy:', test_acc)
```

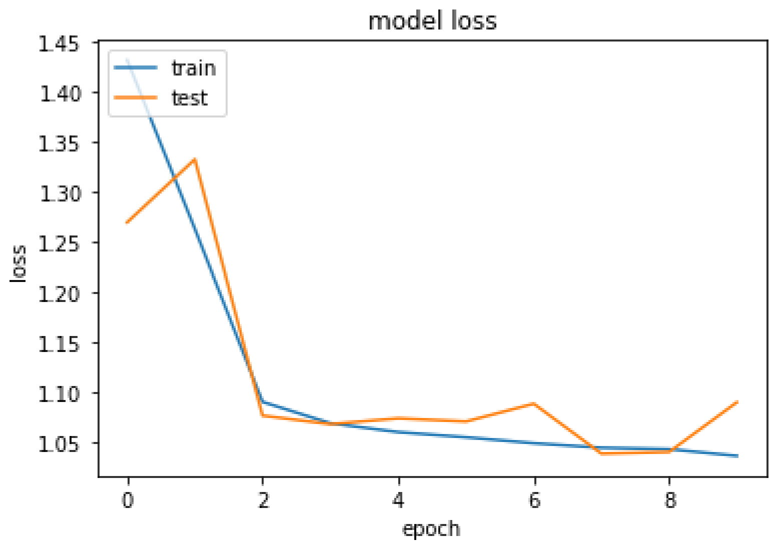
```
10000/10000 - 0s - loss: 1.1060 - accuracy: 0.6358

Test accuracy: 0.6358
```

```
print(demo.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
plt.plot(demo.history['accuracy'])
plt.plot(demo.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(demo.history['loss'])
plt.plot(demo.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```






```
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
```

```
predictions = probability_model.predict(test_images)
```

```
predictions[10]
```

```
array([0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1], dtype=float32)
```

```
np.argmax(predictions[10])
```

```
0
```

```
test_labels[10]
```

```
4
```

Graph this to look at the full set of 10 class predictions.

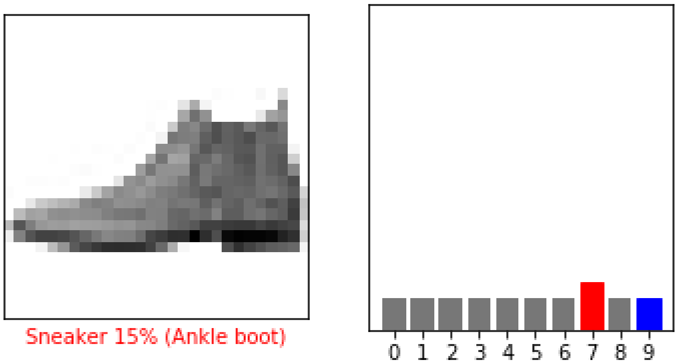
```
def plot_image(i, predictions_array, true_label, img):
  predictions_array, true_label, img = predictions_array, true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)

def plot_value_array(i, predictions_array, true_label):
  predictions_array, true_label = predictions_array, true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

Verify predictions

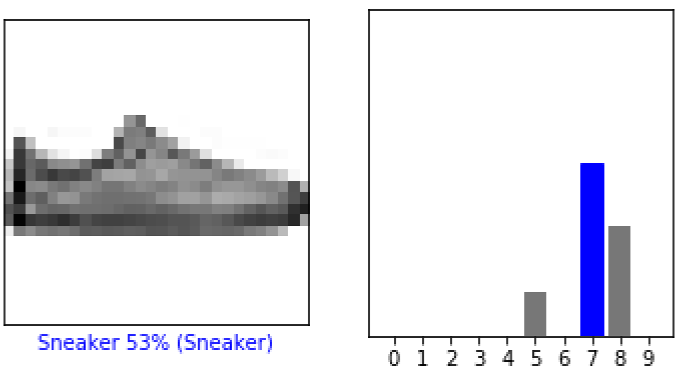
With the model trained, you can use it to make predictions about some images.

Let's look at the 0th image, predictions, and prediction array. Correct prediction labels are blue and in number gives the percentage (out of 100) for the predicted label.

```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```

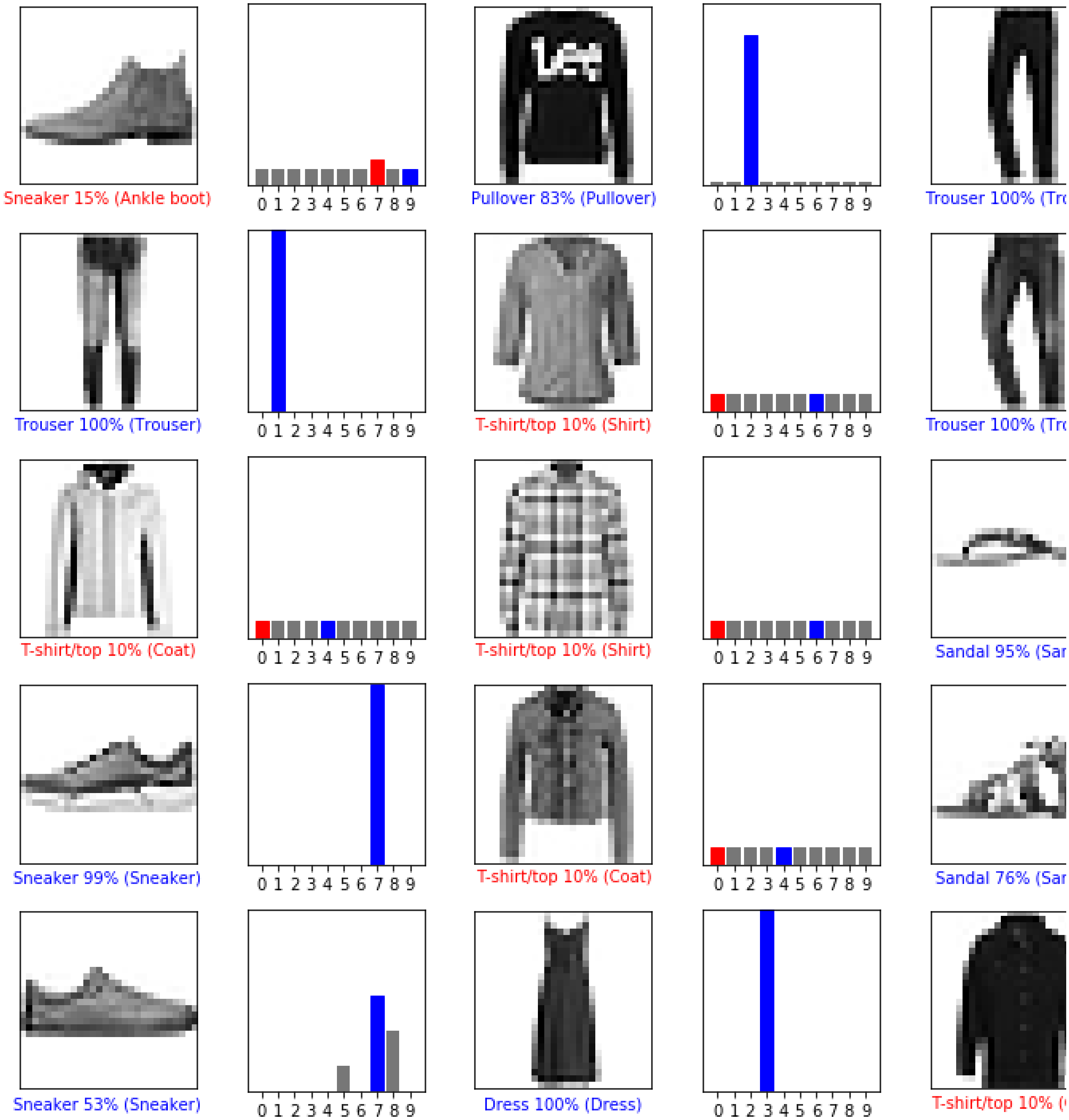



```
i = 12
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```



Let's plot several images with their predictions. Note that the model can be wrong even when very co

```
# Plot the first X test images, their predicted labels, and the true labels.
# Color correct predictions in blue and incorrect predictions in red.
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```



Use the trained model

Finally, use the trained model to make a prediction about a single image.

```
# Grab an image from the test dataset.
img = test_images[1]

print(img.shape)
```

```
print(img.shape)
```

```
(28, 28)
```

`tf.keras` models are optimized to make predictions on a *batch*, or collection, of examples at once. A
single image, you need to add it to a list:

```
# Add the image to a batch where it's the only member.
img = (np.expand_dims(img,0))

print(img.shape)
```

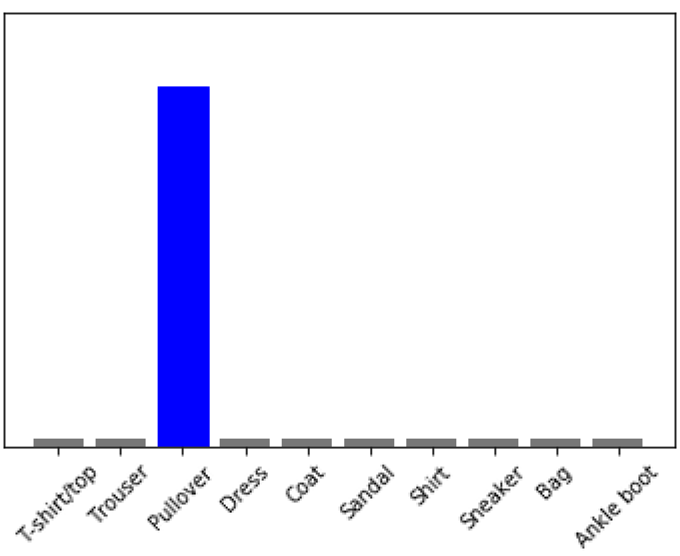
```
(1, 28, 28)
```

Now predict the correct label for this image:

```
predictions_single = probability_model.predict(img)

print(predictions_single)
```

```
[[0.01910141 0.01910141 0.8280874  0.01910141 0.01910141 0.01910141
  0.01910141 0.01910141 0.01910141 0.01910141]]
```

```
plot_value_array(1, predictions_single[0], test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
```



`keras.Model.predict` returns a list of lists—one list for each image in the batch of data. Grab the pre

```
np.argmax(predictions_single[0])
```

```
2
```

And the model predicts a label as expected.

Adding Noise

```
tr= train_images
```

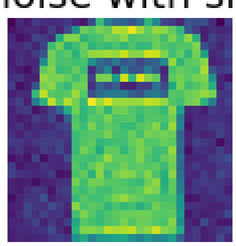
```
import numpy as np
import skimage
im = skimage.img_as_float(train_images[1])
plt.figure(figsize=(15,12))
sigmas = [0.01, 0.05, 0.15, 0.25]
for i in range(4):
    noisy = np.random.normal(im, sigmas[i]**2)
    plt.subplot(2,2,i+1)
    plt.imshow(noisy)
    plt.axis('off')
    plt.title('Gaussian noise with sigma=' + str(sigmas[i]), size=20)
    plt.tight_layout()
    plt.show()
```

Gaussian noise with sigma=0.01

Gaussian noise with sigma=0.05

Gaussian noise with sigma=0.15

Gaussian noise with sigma=0.25



```
def adding_noise(image,sigmas):
    im = skimage.img_as_float(image)
    noisy = np.random.normal(im, sigmas**2)
```

```
    noisy = np.random.normal(im, sigmas**2)
    return noisy


sigmas = [0.01, 0.05, 0.15, 0.25]
print(tr.shape)
z1 = np.copy(train_labels)
z2 = np.copy(train_labels)
z3 = np.copy(train_labels)
z4 = np.copy(train_labels)
tl = np.concatenate((train_labels,z1,z2,z3,z4), axis=0)
for i in range(4):
    new_data=adding_noise(train_images,sigmas[i])
    #print(new_data.shape)
    tr = np.concatenate((tr, new_data), axis=0)
print(len(tr))
```

```
(60000, 28, 28)
300000
```

```
print(len(tl))
```

```
300000
```

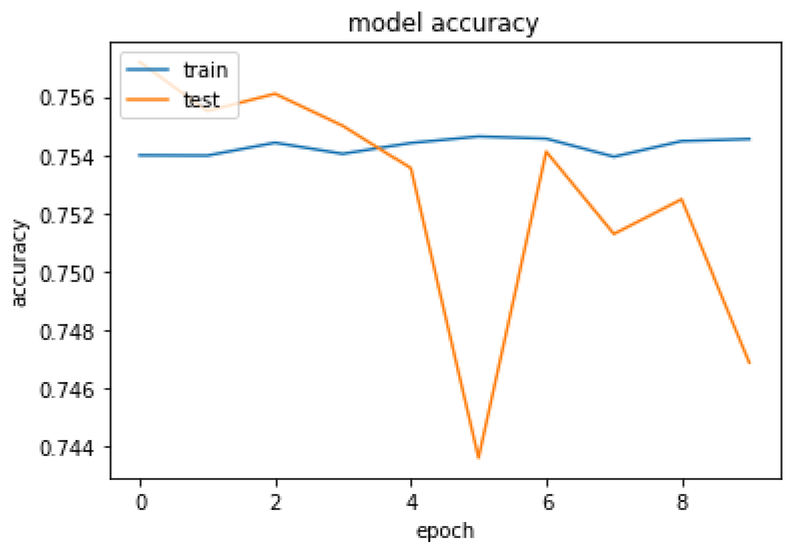
```
m2 =model.fit(tr, tl, validation_split=0.33, epochs=10, verbose=2)
```

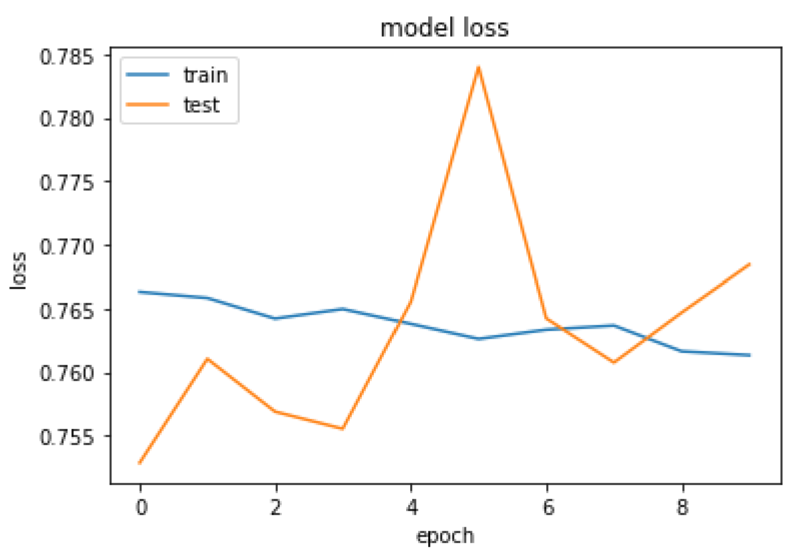
```
Train on 200999 samples, validate on 99001 samples
Epoch 1/10
200999/200999 - 21s - loss: 0.7663 - accuracy: 0.7540 - val_loss: 0.7529 - val_accuracy:
Epoch 2/10
200999/200999 - 21s - loss: 0.7658 - accuracy: 0.7540 - val_loss: 0.7610 - val_accuracy:
Epoch 3/10
200999/200999 - 21s - loss: 0.7642 - accuracy: 0.7544 - val_loss: 0.7569 - val_accuracy:
Epoch 4/10
200999/200999 - 21s - loss: 0.7650 - accuracy: 0.7540 - val_loss: 0.7555 - val_accuracy:
Epoch 5/10
200999/200999 - 21s - loss: 0.7638 - accuracy: 0.7544 - val_loss: 0.7655 - val_accuracy:
Epoch 6/10
200999/200999 - 21s - loss: 0.7626 - accuracy: 0.7546 - val_loss: 0.7840 - val_accuracy:
Epoch 7/10
200999/200999 - 22s - loss: 0.7633 - accuracy: 0.7546 - val_loss: 0.7642 - val_accuracy:
Epoch 8/10
200999/200999 - 21s - loss: 0.7637 - accuracy: 0.7539 - val_loss: 0.7608 - val_accuracy:
Epoch 9/10
200999/200999 - 22s - loss: 0.7616 - accuracy: 0.7545 - val_loss: 0.7647 - val_accuracy:
Epoch 10/10
200999/200999 - 21s - loss: 0.7613 - accuracy: 0.7545 - val_loss: 0.7684 - val_accuracy:
```

```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print('\nTest accuracy:', test_acc)
```

```
10000/10000 - 0s - loss: 0.8156 - accuracy: 0.7378

Test accuracy: 0.7378
```

```
plt.plot(m2.history['accuracy'])
plt.plot(m2.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(m2.history['loss'])
plt.plot(m2.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





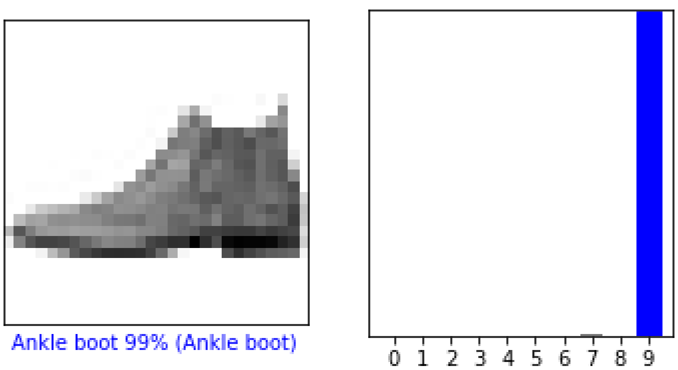
```
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
```

```
predictions = probability_model.predict(test_images)
```

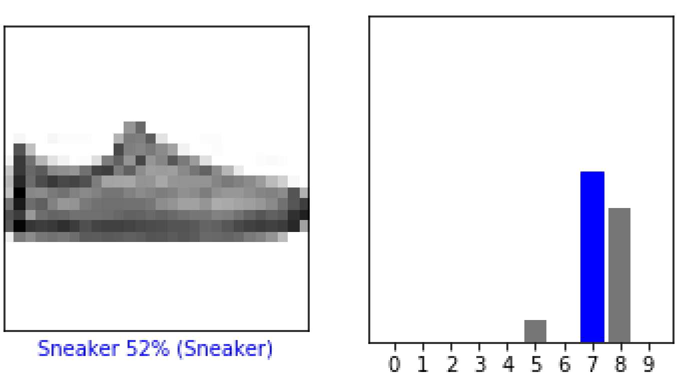
```
predictions[10]
```

```
array([0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1], dtype=float32)
```

```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```

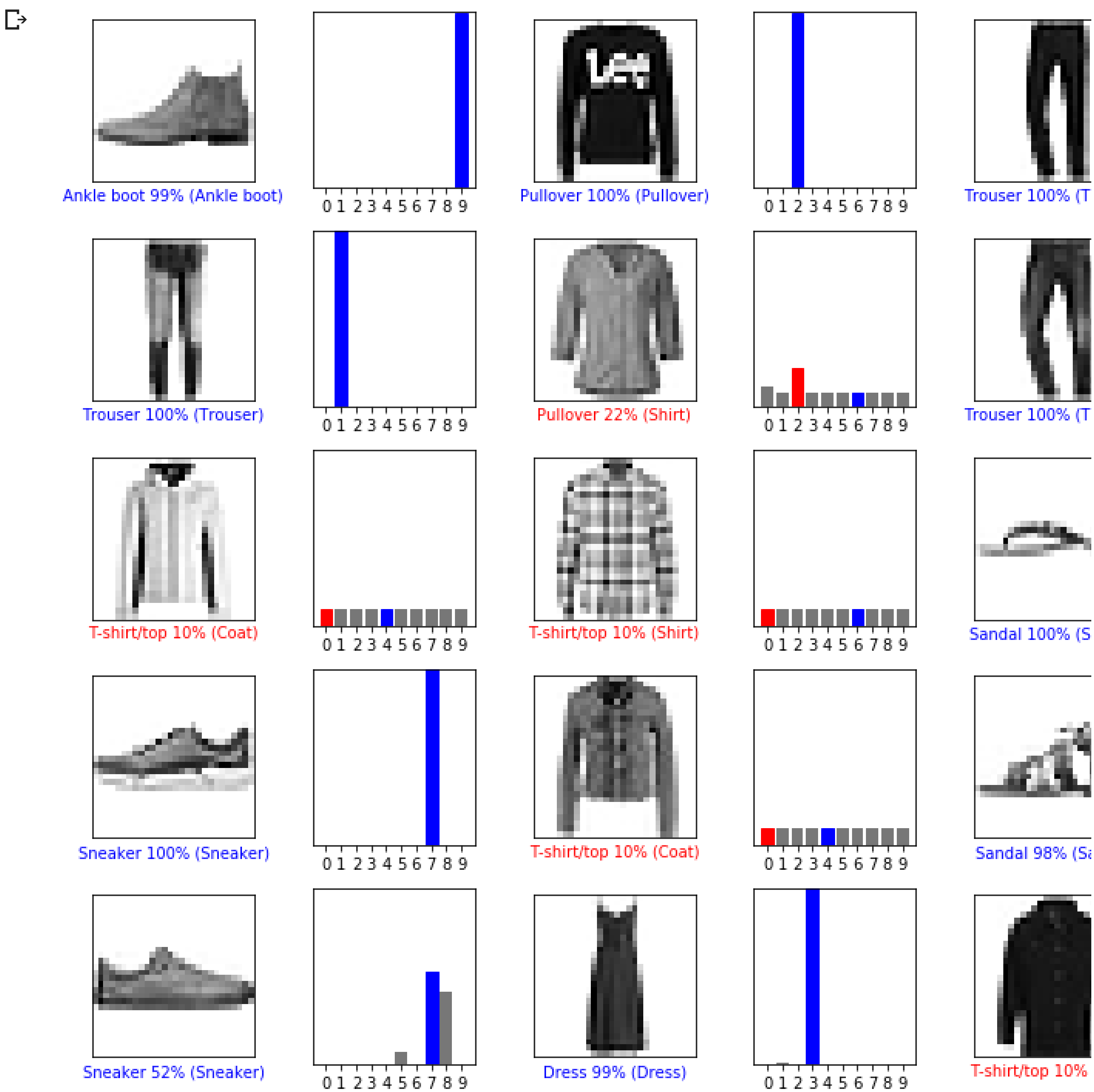



```
i = 12
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```




```
# Plot the first X test images, their predicted labels, and the true labels.
# Color correct predictions in blue and incorrect predictions in red
```

```
# Color correct predictions in blue and incorrect predictions in red.
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```



Summerize the conclusion

After adding the regularizer model accuracy declined compared to without regularizer model but whe
samples the model seems performing almost same on training data but much worse on validation se

The two graph shows the accuracy and loss over the epochs.