Importing the Libraries

In [1]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import keras
import time
```

```
2023-07-13 10:47:45.767460: I tensorflow/core/platform/cpu_feature_guard.cc:182] Thi
s TensorFlow binary is optimized to use available CPU instructions in performance-cr
itical operations.
To enable the following instructions: AVX2 AVX512F FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

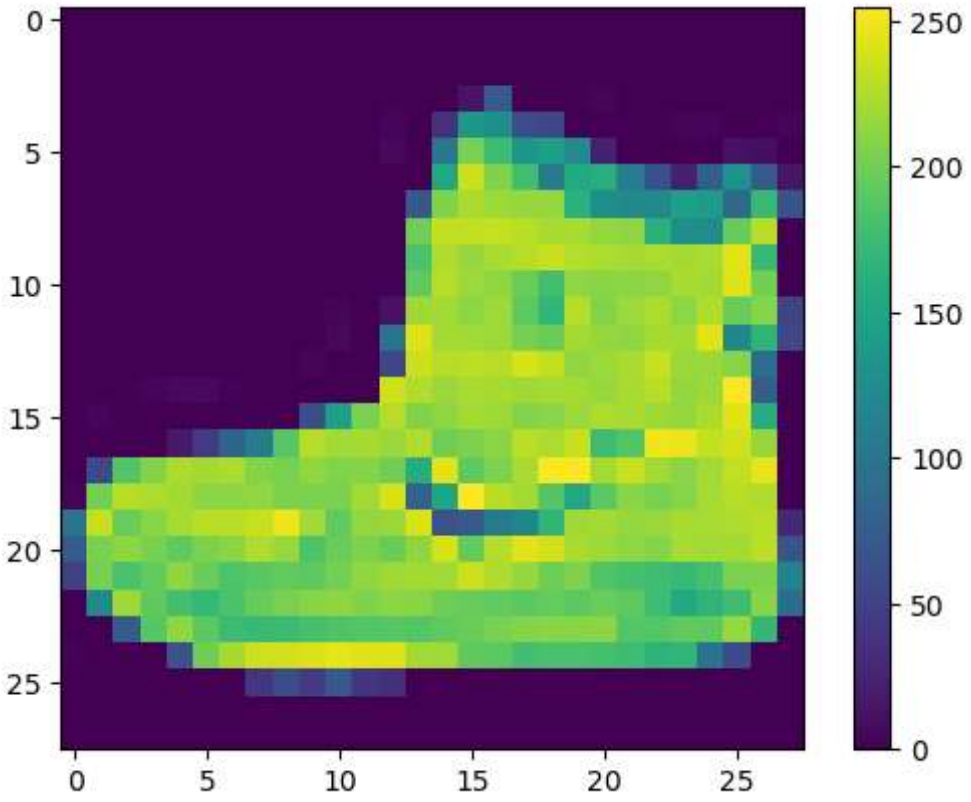
Import the Fashion MNIST dataset

In [2]:
```
fashion_mnist = tf.keras.datasets.fashion_mnist

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

In [3]:
```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Processing the data

In [4]:
```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```



In [5]:
```
train_images = train_images / 255.0

test_images = test_images / 255.0
```

In [6]:
```
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()
```

Build the CNN Model

In [7]:
```
#Create the CNN model
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense

model = Sequential()
```

```
model.add(Conv2D(32, 3, padding='same', activation='relu', kernel_initializer='he_no
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, 3, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))
```

```
2023-07-13 10:48:12.213274: I tensorflow/core/common_runtime/process_util.cc:146] Cr
eating new thread pool with default inter op setting:
```

Compile the model

In [8]:
```
# complie the model
model.compile(optimizer='adam', loss= 'sparse_categorical_crossentropy', metrics=['a
```

In [9]:
```
#model.compile(optimizer='adam',
 #              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  #             metrics=['accuracy'])
```

Training the Model and Feed the model

In [10]:
```
model.fit(train_images, train_labels, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 156s 83ms/step - loss: 0.4622 - accurac
y: 0.8326
Epoch 2/10
1875/1875 [==============================] - 172s 92ms/step - loss: 0.3226 - accurac
y: 0.8816
Epoch 3/10
1875/1875 [==============================] - 162s 87ms/step - loss: 0.2770 - accurac
y: 0.8973
Epoch 4/10
1875/1875 [==============================] - 156s 83ms/step - loss: 0.2510 - accurac
y: 0.9074
Epoch 5/10
1875/1875 [==============================] - 163s 87ms/step - loss: 0.2336 - accurac
y: 0.9145
Epoch 6/10
1875/1875 [==============================] - 155s 83ms/step - loss: 0.2169 - accurac
y: 0.9194
Epoch 7/10
1875/1875 [==============================] - 161s 86ms/step - loss: 0.2051 - accurac
y: 0.9240
Epoch 8/10
1875/1875 [==============================] - 162s 87ms/step - loss: 0.1940 - accurac
y: 0.9277
Epoch 9/10
1875/1875 [==============================] - 158s 85ms/step - loss: 0.1871 - accurac
y: 0.9298
Epoch 10/10
1875/1875 [==============================] - 157s 84ms/step - loss: 0.1812 - accurac
y: 0.9317
```

Out[10]:
```
<keras.callbacks.History at 0x7f6b95026c10>
```

Evaluate Accuracy

In [11]:
```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

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

```
313/313 - 2s - loss: 0.2130 - accuracy: 0.9263 - 2s/epoch - 6ms/step

Test accuracy: 0.9262999892234802
```

In [14]:
```
y_pred = model.predict(test_images)
#y_pred.round(2)
```

```
313/313 [==============================] - 1s 4ms/step
```

In [13]:
```
plt.figure(figsize=(16, 16))

j = 1
for _ in range(25):
    i = np.random.randint(0, 1000)  # Randomly select an index
    plt.subplot(5, 5, j)
    j += 1
    plt.imshow(test_images[i].reshape(28, 28), cmap='Greys')
    plt.title('Actual = {} / {} \nPredicted = {} / {}'.format(class_names[test_label
    plt.axis('off')

plt.tight_layout()
plt.show()
```

Make predictions

In [15]:
```
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
```

In [16]:
```
predictions = probability_model.predict(test_images) #here time taken by the trained
```

```
313/313 [==============================] - 3s 9ms/step
```

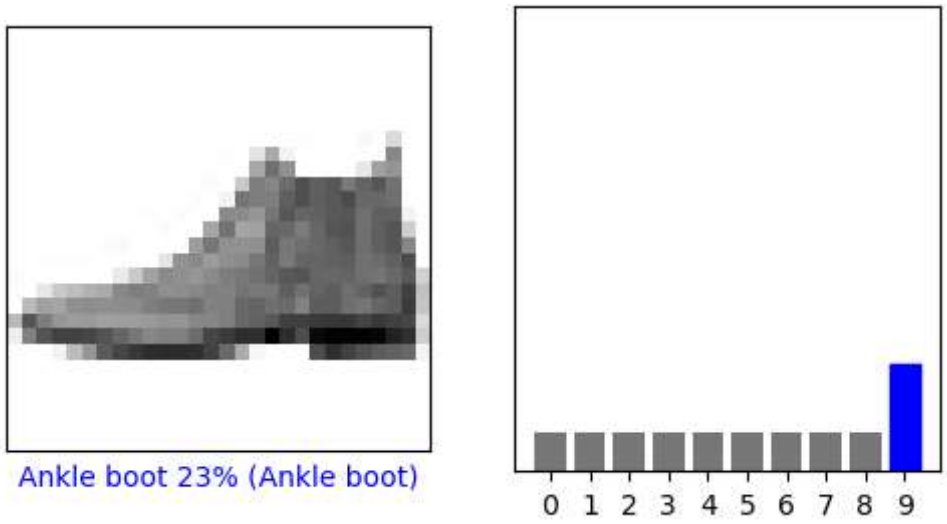
Graph this to look at the full set of 10 class predictions

In [17]:
```
def plot_image(i, predictions_array, true_label, img):
  true_label, img = 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):
  true_label = 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')
```

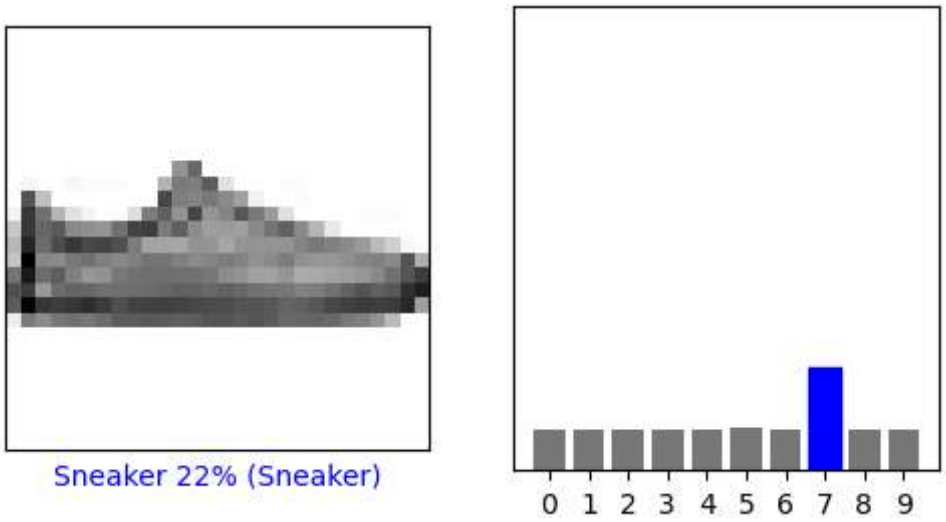
In [18]:

```
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()
```



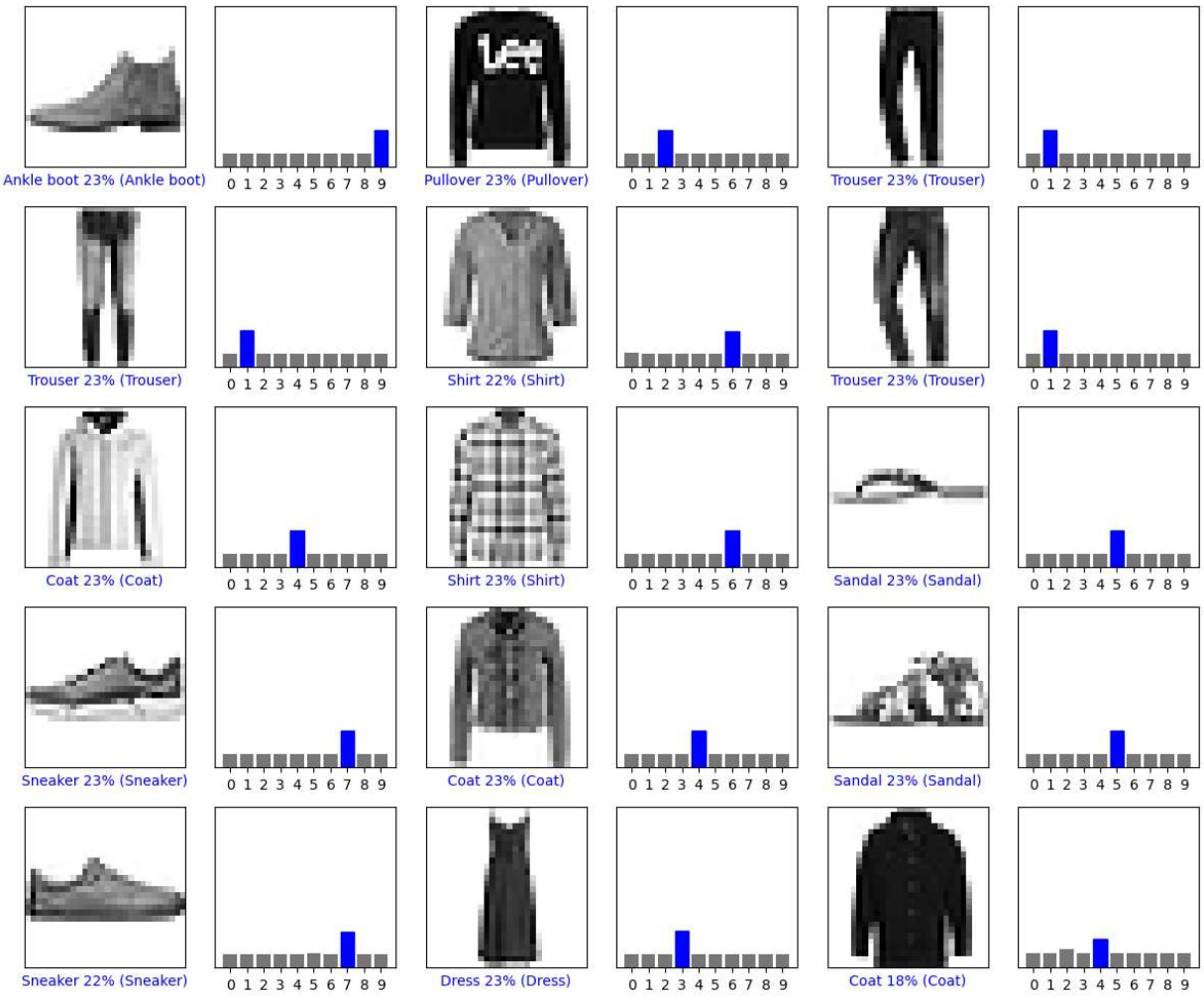
In [19]:

```
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()
```



In [20]:

```
# 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()
```



Save the Model

In [21]:
```
model.save('fashion_mnist_cnn_model.h5') # Save model
```

Use the trained model

In [22]:
```
# Load model
fashion_mnist_cnn_model = keras.models.load_model('fashion_mnist_cnn_model.h5')
```

Time taken by the Trained model for Predictions

In [37]:
```
##----------------------------Time Taking for predicting-------------

start=time.time()
predictions = fashion_mnist_cnn_model.predict(test_images)
stop=time.time()
print("time taken for inferencing :",stop-start)
```

```
313/313 [==============================] - 2s 5ms/step
time taken for inferencing : 2.64981746673584
```

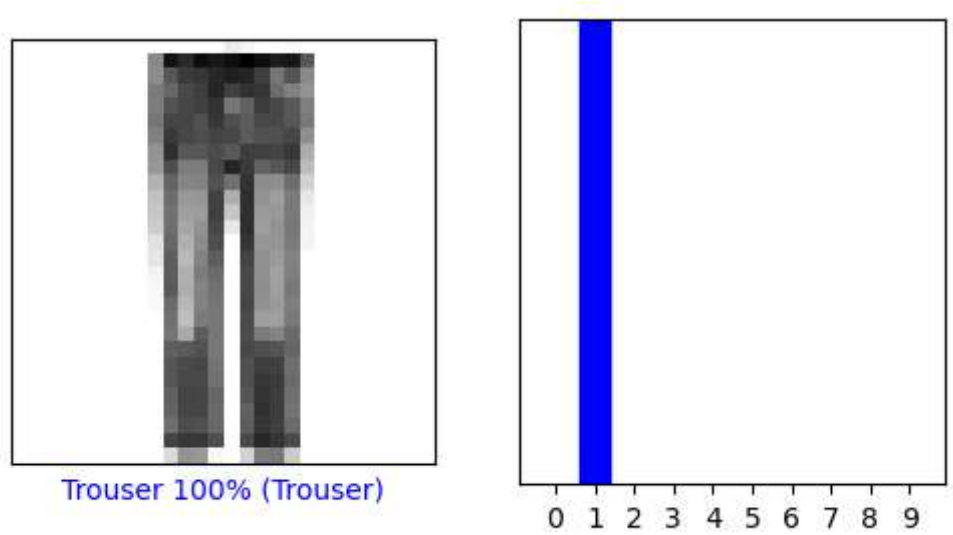
In [24]:
```
def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  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):
  true_label = 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

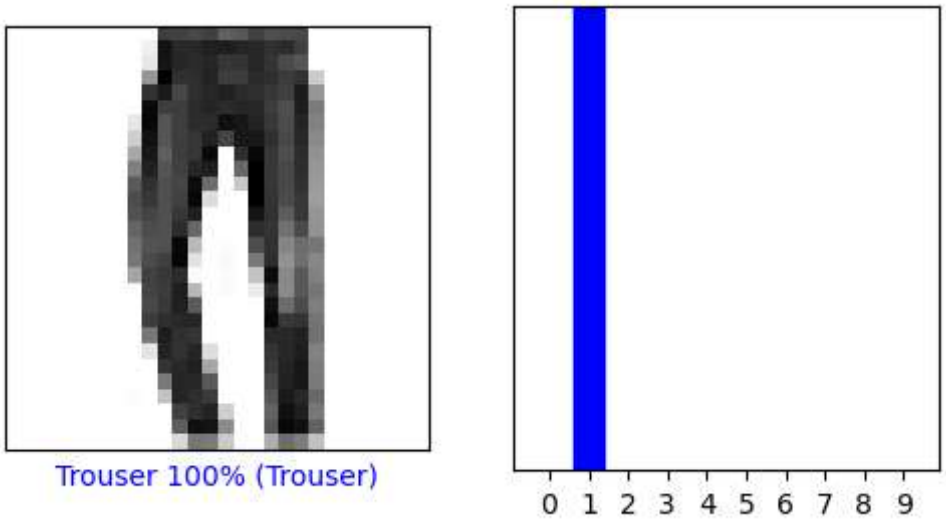
In [25]:

```
i = 15
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()
```



In [26]:

```
i = 5
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()
```



Confusion Matrix

In [27]:

```
import seaborn as sns
import sklearn
from sklearn.metrics import confusion_matrix

plt.figure(figsize=(16,9))
y_pred_labels = [ np.argmax(label) for label in y_pred ]
cm = confusion_matrix(test_labels, y_pred_labels)

# show cm
sns.heatmap(cm, annot=True, fmt='d',xticklabels=class_names, yticklabels=class_names
```

Out[27]: <AxesSubplot:>

USING INTEL OPENVINO OPTIMIZATION TO OPTIMIZE THE TRAINED MODEL

In [38]:

```
import tensorflow as tf
model = tf.keras.models.load_model('fashion_mnist_cnn_model.h5')
tf.saved_model.save(model,'model')
```

```
2023-07-13 11:25:46.939627: I tensorflow/core/common_runtime/executor.cc:1197] [/dev
ice:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and
you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholde
r tensor 'inputs' with dtype float and shape [?,7,7,64]
         [[{{node inputs}}]]
2023-07-13 11:25:46.953641: I tensorflow/core/common_runtime/executor.cc:1197] [/dev
ice:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and
you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholde
r tensor 'inputs' with dtype float and shape [?,128]
         [[{{node inputs}}]]
2023-07-13 11:25:47.108899: I tensorflow/core/common_runtime/executor.cc:1197] [/dev
ice:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and
you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholde
r tensor 'inputs' with dtype float and shape [?,7,7,64]
         [[{{node inputs}}]]
2023-07-13 11:25:47.141276: I tensorflow/core/common_runtime/executor.cc:1197] [/dev
ice:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and
you can ignore this message): INVALID_ARGUMENT: You must feed a value for placeholde
r tensor 'inputs' with dtype float and shape [?,128]
         [[{{node inputs}}]]
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_com
piled_convolution_op, _update_step_xla while saving (showing 3 of 3). These function
s will not be directly callable after loading.
INFO:tensorflow:Assets written to: model/assets
INFO:tensorflow:Assets written to: model/assets
```

CODE TO CONVERT TENSORFLOW KERAS TRAINED MODEL TO INTEL OPENVINO(IR) FORMAT

In [40]:

```
from openvino.inference_engine import IECore, IENetwork
from openvino.runtime import Core
import time
model_xml="saved_model.xml"
model_bin="saved_model.bin"
ie=Core()
model = ie.read_model(model=model_xml,weights=model_bin)
compiled_model = ie.compile_model(model=model, device_name="CPU")
input_layer = compiled_model.input(0)
output_layer = compiled_model.output(0)
infer_request = compiled_model.create_infer_request()
input_tensor = test_images
input_tensor=np.expand_dims(input_tensor,3)
start=time.time()
results = compiled_model.infer_new_request({0: input_tensor})
stop=time.time()
print("Time for inferencing is :",stop-start)
values = next(iter(results.values()))
pred=values
```

```
Time for inferencing is : 0.3059720993041992
```

here the time taken after converting the Trained Model to OpenVino Format is 1.26ms

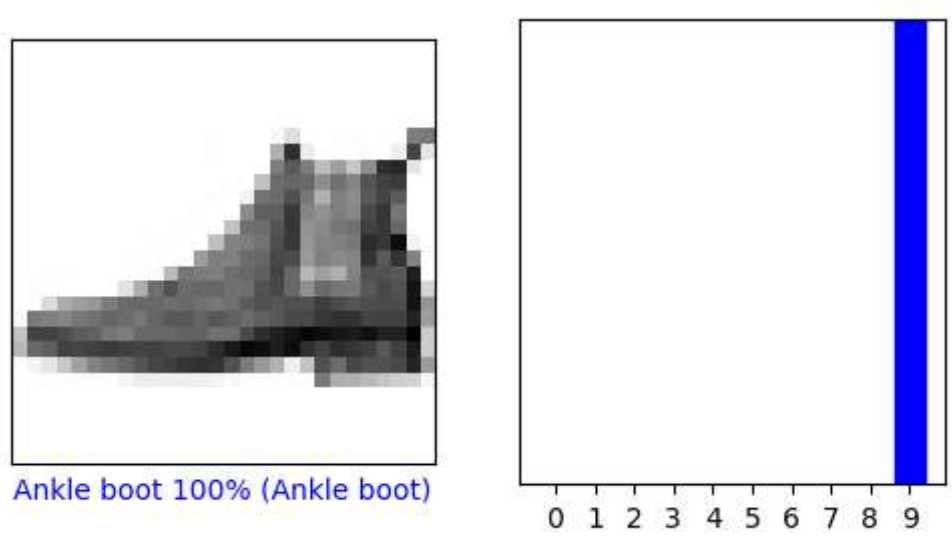
In [41]:

```
def plot_image(i, predictions_array, true_label, img):
  true_label, img = 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):
  true_label = 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')
```

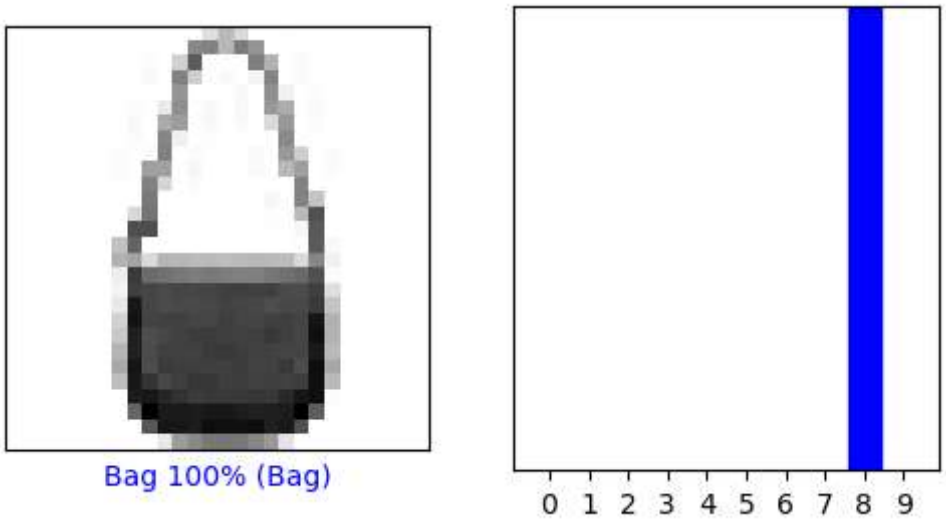
In [44]:

```
i = 2874
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, pred[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, pred[i],  test_labels)
plt.show()
```



In [45]:

```
i = 9997
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()
```



In [ ]:

```
# So in the conclusion,the Trained model has predicted in 4ms but after converting t
#has been Optimized and by uisng Intel devcloud it is Predicting output in 0.36 ms.
#So after applying the Intel Optimization to the Trained model the model optimized.
```