# 基本分類：對數字圖像進行分類(Basic classification: Classify images of digital number)

In [30]:

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

In [2]:

```
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__)
```

```
2.3.1
```

## 導入MNIST數據集(Import the MNIST dataset)

In [3]:

```
#fashion_mnist = keras.datasets.fashion_mnist

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

mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

In [4]:

```
#class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
#               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
class_names = ['0', '1', '2', '3', '4',
               '5', '6', '7', '8', '9']
```

探索數據(Explore the data)

在訓練模型之前，讓我們探索數據集的格式。下圖顯示了訓練集中有60,000張圖像，每張圖像表示為28 x 28像素：

Let's explore the format of the dataset before training the model. The following shows there are 60,000 images in the training set, with each image represented as 28 x 28 pixels:

In [5]:

```
train_images.shape
```

Out[5]:

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

同樣，訓練集中有60,000個標籤：

Likewise, there are 60,000 labels in the training set:

In [6]:

```
len(train_labels)
```

Out[6]:

```
60000
```

每個標籤都是0到9之間的整數：

Each label is an integer between 0 and 9:

In [7]:

```
train_labels
```

Out[7]:

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

測試集中有10,000張圖像。同樣，每個圖像都表示為28 x 28像素：

There are 10,000 images in the test set. Again, each image is represented as 28 x 28 pixels:

In [8]:

```
test_images.shape
```

Out[8]:

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

測試集包含10,000個圖像標籤：

And the test set contains 10,000 images labels:

In [9]:

```
len(test_labels)
```

Out[9]:

```
10000
```

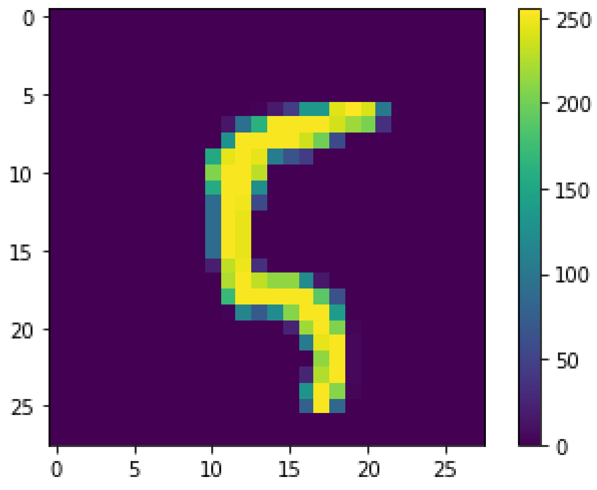
預處理數據(Preprocess the data)

在訓練網絡之前，必須對數據進行預處理。如果您檢查在訓練集中的第一個形象，你會看到像素值落在0到255的範圍內：

The data must be preprocessed before training the network. If you inspect the first image in the training set, you will see that the pixel values fall in the range of 0 to 255:

In [10]:

```
plt.figure()
plt.imshow(train_images[100])
plt.colorbar()
plt.grid(False)
plt.show()
```



將這些值縮放到0到1的範圍，然後再將其輸入神經網絡模型。要做到這一點，由255分的數值是非常重要的是，訓練集和測試集以同樣的方式進行預處理：

Scale these values to a range of 0 to 1 before feeding them to the neural network model. To do so, divide the values by 255. It's important that the *training set* and the *testing set* be preprocessed in the same way:

In [11]:

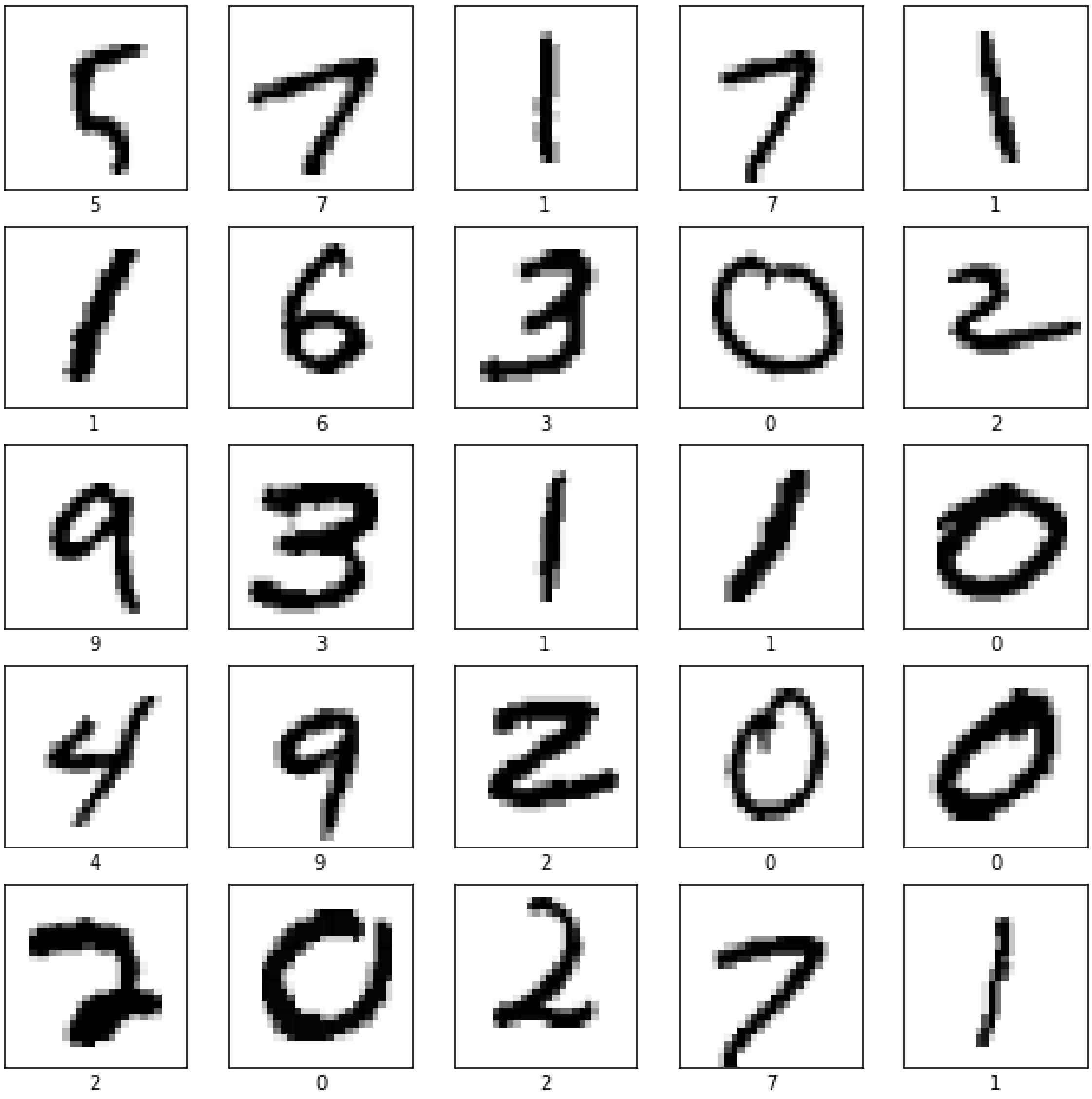
```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

為了驗證數據的格式正確，並準備好構建和訓練網絡，讓我們顯示訓練集中的前25張圖像，並在每個圖像下方顯示班級名稱。

To verify that the data is in the correct format and that you're ready to build and train the network, let's display the first 25 images from the *training set* and display the class name below each image.

In [12]:

```
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+100], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i+100]])
plt.show()
```



構建模型(Build the model)

建立神經網絡需要配置模型的各層，然後編譯模型。

Building the neural network requires configuring the layers of the model, then compiling the model.

設置圖層(Set up the layers)

神經網絡的基本組成部分是層。圖層從輸入到其中的數據中提取表示。希望這些表示對於當前的問題有意義。

深度學習的大部分內容是將簡單的層鏈接在一起。大多數層（例如 `tf.keras.layers.Dense` ）具有在訓練過程中學習的參數。

The basic building block of a neural network is the *layer*. Layers extract representations from the data fed into them. Hopefully, these representations are meaningful for the problem at hand.

Most of deep learning consists of chaining together simple layers. Most layers, such as `tf.keras.layers.Dense` , have parameters that are learned during training.

In [13]:

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10)
])
```

該網絡的第一層 `tf.keras.layers.Flatten` 將圖像格式從二維數組（28 x 28像素）轉換為一維數組（28 * 28 = 784像素）。可以將這一層看作是堆疊圖像中的像素行並將它們排成一行。該層沒有學習參數。它只是重新格式化數據。

像素展平後，網絡由 `tf.keras.layers.Dense` 兩層序列組成。這些是緊密連接或完全連接的神經層。第一 `Dense` 層有128個節點（或神經元）。第二個（和最後）層是一個10節點*SOFTMAX*層返回的10概率得分的陣列總和為1。每個節點包含一個分數，指示當前圖像屬於10類中的一個的概率。

The first layer in this network, `tf.keras.layers.Flatten` , transforms the format of the images from a two-dimensional array (of 28 by 28 pixels) to a one-dimensional array (of 28 * 28 = 784 pixels). Think of this layer as unstacking rows of pixels in the image and lining them up. This layer has no parameters to learn; it only reformats the data.

After the pixels are flattened, the network consists of a sequence of two `tf.keras.layers.Dense` layers. These are densely connected, or fully connected, neural layers. The first `Dense` layer has 128 nodes (or neurons). The second (and last) layer is a 10-node *softmax* layer that returns an array of 10 probability scores that sum to 1. Each node contains a score that indicates the probability that the current image belongs to one of the 10 classes.

編譯模型(Compile the model)

在準備訓練模型之前，需要進行一些其他設置。這些是在模型的編譯步驟中添加的：

- 損失函數 —這措施的模型如何準確的訓練中。您希望最小化此功能，以在正確的方向上“引導”模型。
- 優化器 —這是基於模型看到的數據及其損失函數來更新模型的方式。
- 度量 —用來監控訓練和測試步驟。以下示例使用precision，即正確分類的圖像比例。

Before the model is ready for training, it needs a few more settings. These are added during the model's *compile* step:

- *Loss function* —This measures how accurate the model is during training. You want to minimize this function to "steer" the model in the right direction.
- *Optimizer* —This is how the model is updated based on the data it sees and its loss function.
- *Metrics* —Used to monitor the training and testing steps. The following example uses *accuracy*, the fraction of the images that are correctly classified.

In [14]:

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

訓練模型(Train the model)

訓練神經網絡模型需要執行以下步驟：

1. 將訓練數據輸入模型。在這個例子中，訓練數據是在 `train_images` 和 `train_labels` 陣列。
2. 該模型學會副圖像和標籤。
3. 您要求模型對測試集進行預測（在此示例中為test_images數組）。
4. 驗證預測是否與 `test_labels` 陣列中的標籤匹配。

Training the neural network model requires the following steps:

1. Feed the training data to the model. In this example, the training data is in the `train_images` and `train_labels` arrays.
2. The model learns to associate images and labels.
3. You ask the model to make predictions about a test set—in this example, the `test_images` array.
4. Verify that the predictions match the labels from the `test_labels` array.

餵模型(Feed the model)

要開始訓練，請調用該 `model.fit` 方法，之所以這樣稱呼是因為該方法使模型“適合”訓練數據

To start training, call the `model.fit` method—so called because it "fits" the model to the training data:

In [15]:

```
model.fit(train_images, train_labels, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2587 - ac
curacy: 0.9255
Epoch 2/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1117 - ac
curacy: 0.9672
Epoch 3/10
1875/1875 [==============================] - 2s 989us/step - loss: 0.0755 -
accuracy: 0.9769
Epoch 4/10
1875/1875 [==============================] - 2s 994us/step - loss: 0.0568 -
accuracy: 0.9826
Epoch 5/10
1875/1875 [==============================] - 2s 989us/step - loss: 0.0438 -
accuracy: 0.9864
Epoch 6/10
1875/1875 [==============================] - 2s 974us/step - loss: 0.0351 -
accuracy: 0.9891
Epoch 7/10
1875/1875 [==============================] - 2s 984us/step - loss: 0.0276 -
accuracy: 0.9913
Epoch 8/10
1875/1875 [==============================] - 2s 980us/step - loss: 0.0226 -
accuracy: 0.9932
Epoch 9/10
1875/1875 [==============================] - 2s 985us/step - loss: 0.0190 -
accuracy: 0.9941
Epoch 10/10
1875/1875 [==============================] - 2s 981us/step - loss: 0.0163 -
accuracy: 0.9949
```

Out[15]:

```
<tensorflow.python.keras.callbacks.History at 0x25715ec7a88>
```

由於模型火車，顯示損失和精度指標。該模型在訓練數據上達到約0.91（或91%）的精度。

As the model trains, the loss and accuracy metrics are displayed. This model reaches an accuracy of about 0.91 (or 91%) on the training data.

評估準確性(Evaluate accuracy)

接下來，比較模型在測試數據集上的表現：

Next, compare how the model performs on the test dataset:

In [16]:

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

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

```
313/313 - 0s - loss: 0.0762 - accuracy: 0.9799

Test accuracy: 0.9799000024795532
```

事實證明，測試數據集的準確性略低於訓練數據集的準確性。訓練準確性和測試準確性之間的差距代表過度擬合。過度擬合是當一個機器學習模型進行比對在訓練數據上新的，以前看不到的投入差。過度擬合的模型“存儲”訓練數據，而測試數據的準確性較低。有關更多信息，請參見以下內容：

- 證明過度擬合 (https://www.tensorflow.org/tutorials/keras/overfit_and_underfit#demonstrate_overfitting)
- 防止過度擬合的策略 (https://www.tensorflow.org/tutorials/keras/overfit_and_underfit#strategies_to_prevent_overfitting)

It turns out that the accuracy on the test dataset is a little less than the accuracy on the training dataset. This gap between training accuracy and test accuracy represents *overfitting*. Overfitting is when a machine learning model performs worse on new, previously unseen inputs than on the training data. An overfitted model "memorizes" the training data—with less accuracy on testing data. For more information, see the following:

- Demonstrate overfitting (https://www.tensorflow.org/tutorials/keras/overfit_and_underfit#demonstrate_overfitting)
- Strategies to prevent overfitting (https://www.tensorflow.org/tutorials/keras/overfit_and_underfit#strategies_to_prevent_overfitting)

作出預測(Make predictions)

通過訓練模型，您可以使用它來預測某些圖像。模型的線性輸出logits。附加一個softmax層，以將logits (https://developers.google.com/machine-learning/glossary#logits)轉換為更容易解釋的概率。

With the model trained, you can use it to make predictions about some images. The model's linear outputs, logits (https://developers.google.com/machine-learning/glossary#logits). Attach a softmax layer to convert the logits to probabilities, which are easier to interpret.

In [17]:

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

In [18]:

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

在這裡，模型已經預測了測試集中每個圖像的標籤。讓我們看一下第一個預測：

Here, the model has predicted the label for each image in the testing set. Let's take a look at the first prediction:

In [19]:

```
predictions[0]
```

Out[19]:

```
array([2.3416415e-11, 3.4424972e-12, 1.5704960e-09, 2.1787529e-05,
       8.4637209e-15, 3.2559857e-08, 8.0066430e-19, 9.9997807e-01,
       3.6333745e-08, 9.6642481e-08], dtype=float32)
```

預測是10個數字組成的數組。它們代表模型對圖像對應於10種不同服裝中的每一種的“信心”。您可以看到哪個標籤的置信度最高：

A prediction is an array of 10 numbers. They represent the model's "confidence" that the image corresponds to each of the 10 different articles of clothing. You can see which label has the highest confidence value:

In [20]:

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

Out[20]:

```
7
```

因此，模型最有信心該圖像是踝靴或class_names[9]。檢查測試標籤表明此分類是正確的：

So, the model is most confident that this image is an ankle boot, or `class_names[9]` . Examining the test label shows that this classification is correct:

In [21]:

```
test_labels[0]
```

Out[21]:

```
7
```

以圖形方式查看完整的10個類預測。

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

In [22]:

```
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')
```

驗證預測(Verify predictions)

通過訓練模型，您可以使用它來預測某些圖像。

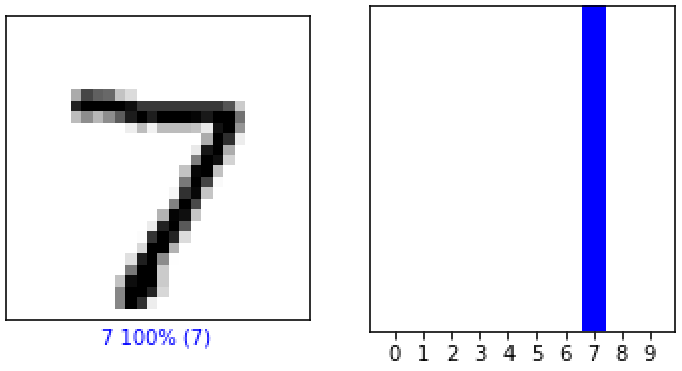
With the model trained, you can use it to make predictions about some images.

讓我們看一下第0張圖像，預測和預測數組。正確的預測標籤為藍色，錯誤的預測標籤為紅色。該數字給出了預測標籤的百分比（滿分為100）。

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

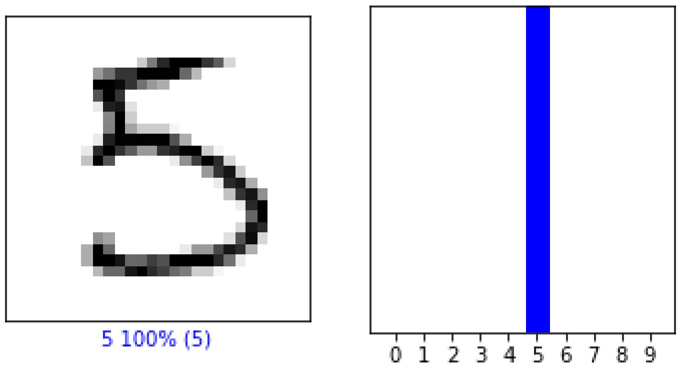
In [23]:

```
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 [24]:

```
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()
```

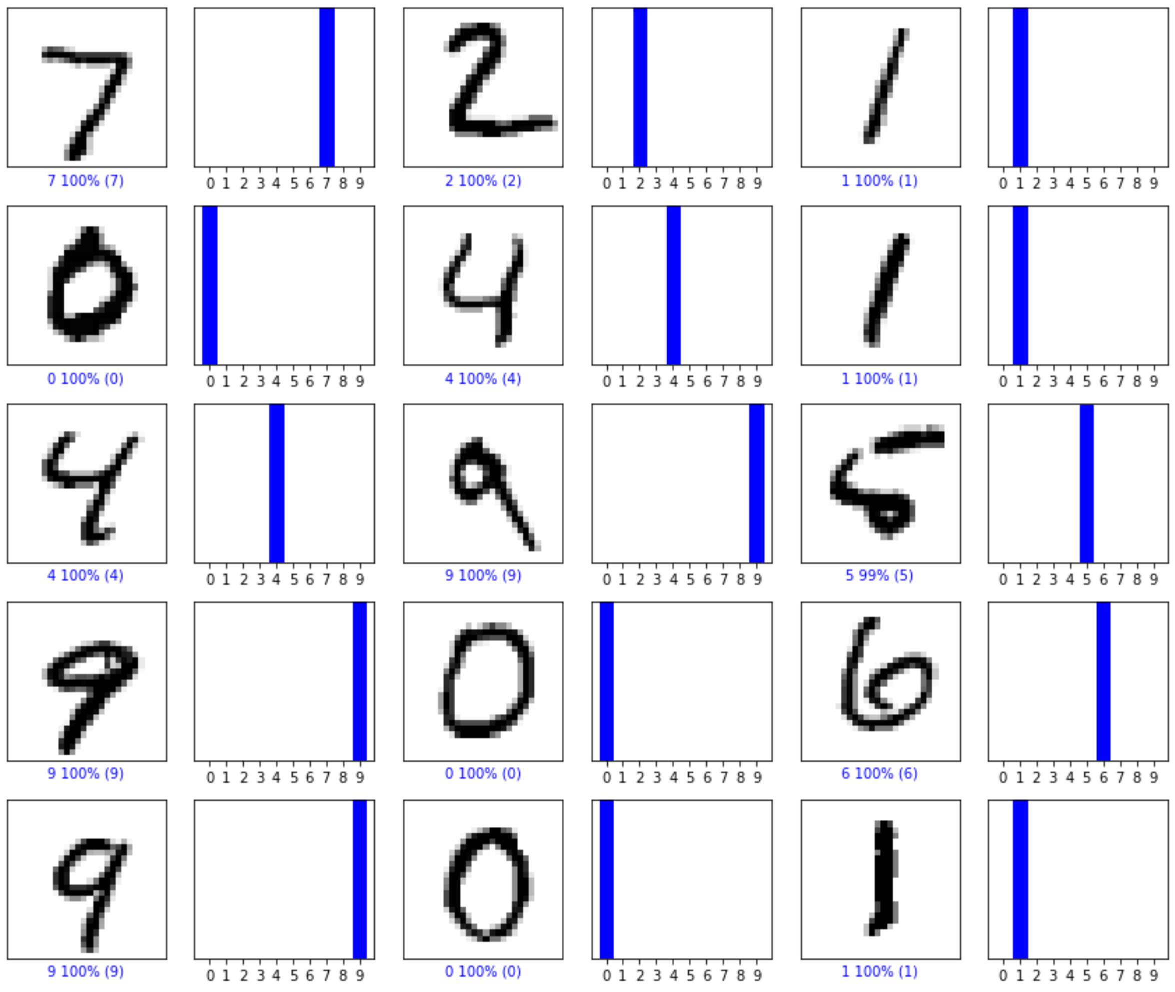


讓我們繪製一些帶有預測的圖像。請注意，即使非常自信，該模型也可能是錯誤的。

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

In [25]:

```
# 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()
```



使用訓練有素的模型(Use the trained model)

最後，使用經過訓練的模型對單個圖像進行預測。

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

In [26]:

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

print(img.shape)
```

```
(28, 28)
```

`tf.keras` 對模型進行了優化，可以一次對一批或一組示例進行預測。因此，即使您使用的是單個圖像，也需要將其添加到列表中：

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

In [27]:

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

print(img.shape)
```

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

現在，為該圖像預測正確的標籤：

Now predict the correct label for this image:

In [28]:

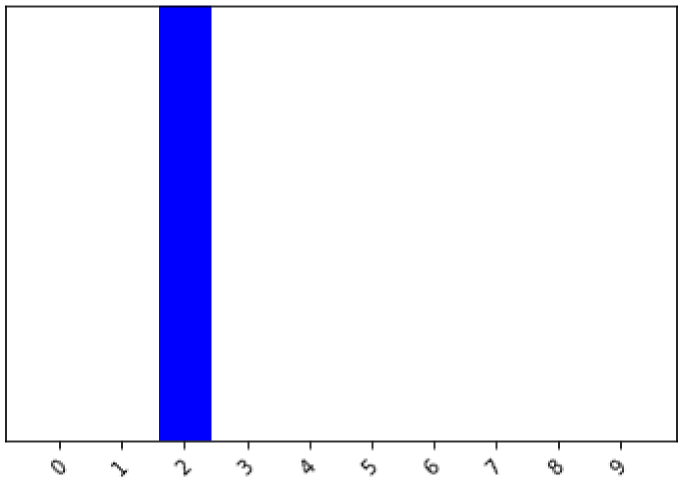
```
predictions_single = probability_model.predict(img)

print(predictions_single)
```

```
[[1.67042019e-10 1.99320812e-06 9.99997854e-01 1.03338003e-07
  1.13792161e-19 2.21969199e-08 1.08163985e-10 1.78718511e-20
  1.71234884e-08 1.55477620e-15]]
```

In [29]:

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



`keras.Model.predic` t返回列表列表-數據批次中每個圖像的一個列表。批量獲取我們（僅）圖像的預測：

`keras.Model.predict` returns a list of lists—one list for each image in the batch of data. Grab the predictions for our (only) image in the batch:

In [30]:

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

Out[30]:

```
2
```

該模型將按預期預測標籤。

And the model predicts a label as expected.