對服裝圖像進行分類(Basic classification: Classify images of clothing)

本指南訓練了一個神經網絡模型來對衣服的圖像進行分類，例如運動鞋和襯衫。如果您不了解所有細節，也可以；這是完整的TensorFlow程序的快速概述，詳細內容隨您進行。

本指南使用tf.keras (https://www.tensorflow.org/guide/keras)（高級API）在TensorFlow中構建和訓練模型。

This guide trains a neural network model to classify images of clothing, like sneakers and shirts. It's okay if you don't understand all the details; this is a fast-paced overview of a complete TensorFlow program with the details explained as you go.

This guide uses tf.keras (https://www.tensorflow.org/guide/keras), a high-level API to build and train models in TensorFlow.

In [1]:

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

In [3]:

```
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__)
```

```
2.0.0
```

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

本指南使用Fashion MNIST (https://github.com/zalandoresearch/fashion-mnist)數據集，其中包含10個類別的70,000張灰度圖像。圖像顯示了低分辨率（28 x 28像素）的單個衣物，如下所示：

Fashion MNIST sprite

Figure 1. Fashion-MNIST samples (https://github.com/zalandoresearch/fashion-mnist) (by Zalando, MIT License).

Fashion MNIST旨在替代經典MNIST (http://yann.lecun.com/exdb/mnist/)數據集-通常用作計算機視覺的機器學習程序的“ Hello，World”。MNIST數據集包含手寫數字（0、1、2等）的圖像，格式與您將在此處使用的衣服的格式相同。

本指南將Fashion MNIST用於多種用途，因為它比常規MNIST更具挑戰性。這兩個數據集相對較小，用於驗證的算法按預期工作。它們是測試和調試代碼的良好起點。

在這裡，6圖像被用來訓練網絡和10,000張，以評估網絡如何準確地學會了圖像分類。您可以直接從TensorFlow訪問時裝MNIST。直接從TensorFlow導入和加載Fashion MNIST數據：

This guide uses the Fashion MNIST (https://github.com/zalandoresearch/fashion-mnist) dataset which contains 70,000 grayscale images in 10 categories. The images show individual articles of clothing at low resolution (28 by 28 pixels), as seen here:

Fashion MNIST sprite

Figure 1. Fashion-MNIST samples (https://github.com/zalandoresearch/fashion-mnist) (by Zalando, MIT License).

Fashion MNIST is intended as a drop-in replacement for the classic MNIST (http://yann.lecun.com/exdb/mnist/) dataset—often used as the "Hello, World" of machine learning programs for computer vision. The MNIST dataset contains images of handwritten digits (0, 1, 2, etc.) in a format identical to that of the articles of clothing you'll use here.

This guide uses Fashion MNIST for variety, and because it's a slightly more challenging problem than regular MNIST. Both datasets are relatively small and are used to verify that an algorithm works as expected. They're good starting points to test and debug code.

Here, 60,000 images are used to train the network and 10,000 images to evaluate how accurately the network learned to classify images. You can access the Fashion MNIST directly from TensorFlow. Import and load the Fashion MNIST data directly from TensorFlow:

In [4]:

```
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()
```

加載數據集將返回四個NumPy數組：

- 在 `train_images` 和 `train_labels` 陣列的*訓練集*-The數據模型用來學習。
- 針對測試集，和 `test_images` ，對模型進行了測試 `test_labels` 。

圖像是28x28 NumPy數組，像素值範圍是0到255。標籤是整數數組，範圍是0到9。這些對應於圖像表示的衣服類別：

標籤	類
0	T卹/上衣
1	褲子
2	拉過來
3	連衣裙

4	塗層
5	涼鞋
6	襯衫
7	運動鞋
8	袋
9	腳踝靴

每個圖像都映射到一個標籤。由於*類名稱*不包括數據集，將它們存儲在這裡以後使用繪製圖像時：

Loading the dataset returns four NumPy arrays:

- The `train_images` and `train_labels` arrays are the *training set*—the data the model uses to learn.
- The model is tested against the *test set*, the `test_images` , and `test_labels` arrays.

The images are 28x28 NumPy arrays, with pixel values ranging from 0 to 255. The *labels* are an array of integers, ranging from 0 to 9. These correspond to the *class* of clothing the image represents:

Label	Class
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Each image is mapped to a single label. Since the *class names* are not included with the dataset, store them here to use later when plotting the images:

In [5]:

```
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 [6]:

```
train_images.shape
```

Out[6]:

```
(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 [7]:

```
len(train_labels)
```

Out[7]:

```
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 [8]:

```
train_labels
```

Out[8]:

```
array([9, 0, 0, ..., 3, 0, 5], 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 [9]:

```
test_images.shape
```

Out[9]:

```
(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 [10]:

```
len(test_labels)
```

Out[10]:

```
10000
```

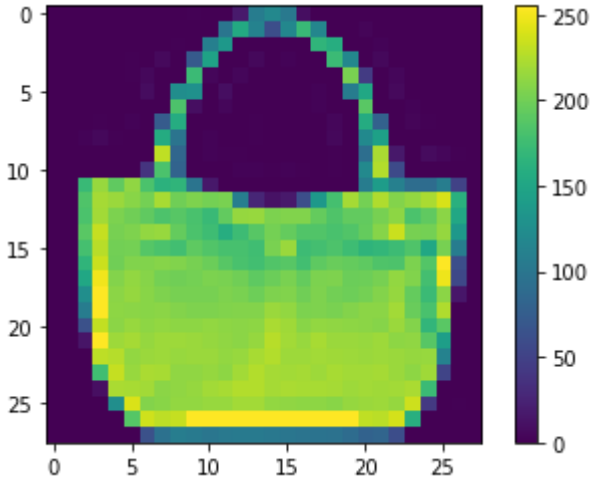
預處理數據(Preprocess the data)

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 [11]:

```
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 [12]:

```
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 [13]:

```
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 [14]:

```
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 [15]:

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