第四章

善用 ML Kit 的電腦視覺 Android App

第 3 章介紹了 ML Kit，並說明如何把人臉偵測（face detection）模型運用到行動 App。不過 ML Kit 的功能遠不只如此——它也可以針對一些常見的視覺應用場景，快速構建出原型設計、套用各種自定義的模型，或是直接利用一些現成的解決方案，實作出像是條碼偵測（barcode detection）之類的應用。本章將針對一些電腦視覺的應用場景，探索 ML Kit 裡的一些模型，包括圖片標記與分類（image labeling & classification）以及靜態圖片與動態影片中的物體偵測（object detection）。本章的內容採用 Android 平台，使用 Kotlin 來做為程式語言。第 6 章則會改用 iOS 平台，使用 Swift 開發出同樣的功能。

圖片標記與分類

圖片分類（image classification）可說是機器學習界眾所周知的一個概念，也是各種電腦視覺應用的重要基石。從最簡單的意義上來說，如果你向電腦展示圖片，電腦就告訴你圖片裡有什麼東西，這其實就是在進行圖片分類。舉例來說，如果你向電腦展示圖 4-1 這樣的一張圖片，電腦或許就會把圖片標記為 cat（貓）。

不只如此，ML Kit 的圖片標記（image labeling）功能還會更進一步，列出它在圖片中所看到的東西列表，並給出相應的機率值，因此它在圖 4-1 中看到的不只是一隻貓，或許它還會說自己看到了貓、花、草、雛菊等等。

接著我們就來探索如何建立　個超級簡單的 Android App，對這張圖片進行標記！這裡會使用到 Android Studio 與 Kotlin。如果你還沒準備好這些東西，可以先到 *https://developer.android.com/studio/* 進行下載。

圖 4-1　一張貓的圖片

第 1 步：建立 App、設定 ML Kit

如果你還沒閱讀過第 3 章，或是還不熟悉如何製作 Android App，建議先回頭仔細閱讀第 3 章的內容！App 建立好之後，必須先按照第 3 章的說明，修改　下 build.gradle 檔案。在這個例子中，我們會把之前所添加的 face-detection（人臉偵測）函式庫，替換成 image-labeling（圖片標記）函式庫（參見如下的最後　個項目）：

```
dependencies {
    implementation "org.jetbrains.kotlin:kotlin stdlib:$kotlin version"
    implementation 'androidx.core:core ktx:1.2.0'
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso core:3.2.0'
    implementation 'com.google.mlkit:image labeling:17.0.1'
}
```

完成這個操作之後，由於我們修改過 Gradle 檔案，因此 Android Studio 可能會要求進行同步（sync）。這樣就可以把新的 ML Kit 依賴項目包含進來，重新觸發新的　次 build 組建程序。

第 2 步：建立使用者介面

我們會幫這個 App 建立　個超級簡單的使用者介面，直接對圖片進行標記。在 Android Studio 的 res/layout 目錄下，可以看到　個名為 *activity_main.xml* 的檔案。如果你還不太熟悉，請回頭參閱第 3 章的說明。

這裡會修改　下使用者介面，在線性版面（LinearLayout）放入　個 ImageView、　個 Button 和　個 TextView（如下所示）：

```
<?xml version="1.0" encoding="utf 8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout width="match parent"
    android:layout height="match parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout width="match parent"
        android:layout height="wrap content"
        android:orientation="vertical"
        app:layout constraintStart toStartOf="parent"
        app:layout constraintTop toTopOf="parent">

        <ImageView
            android:id="@+id/imageToLabel"
            android:layout width="match parent"
            android:layout height="match parent"
            android:layout gravity="center"
            android:adjustViewBounds="true"
        />
        <Button
            android:id="@+id/btnTest"
            android:layout width="wrap content"
            android:layout height="wrap content"
            android:text="Label Image"
            android:layout gravity="center"/>
        <TextView
            android:id="@+id/txtOutput"
            android:layout width="match parent"
```

第十一章

使用自定義模型的 iOS App

第 9 章運用 TensorFlow Lite Model Maker、Cloud AutoML Vision Edge 以及 TensorFlow 搭配轉移學習的做法，在各式各樣的應用場景下建立了自定義模型。本章就來看看如何把這些模型整合到 iOS App 中。我們將會聚焦於兩大應用場景：圖片辨識與文字分類。如果你已經讀過第 10 章才來到這裡，就會發現我們的討論相當類似，因為模型都不是直接放入 App 就能正常運作。在 Android 中，運用 TensorFlow Lite Model Maker 所建立的模型都會附帶詮釋資料（metadata），而 task 函式庫則可以讓整合工作容易許多。但如果使用的是 iOS，就沒有同等級的支援了；把資料送入模型並解析結果的工作，就要由你自己來進行　些非常低階的工作，處理像是「把內部資料型別轉換成模型可理解的相應張量」這類的工作。讀完本章後，你就能稍微瞭解這些工作相關的　些基礎知識，不過你所遇到的應用場景，還是有可能大不相同，具體來說還是取決於你手中的資料！不過，如果使用的是 ML Kit 可支援的自定義模型類型，則可算是例外的情況；因此，我們也會探索　下如何在 iOS 運用 ML Kit API 處理自定義模型。

把模型裝進 iOS 中

訓練完模型並轉換成 TensorFlow Lite 的 TFLite 格式之後，就可以得到　個二進位 blob 檔案，我們可以把它當成　個 assets 資源，直接添加到 App 之中。App 可以把它載入到 TensorFlow Lite 的直譯器（interpreter），而我們則必須在二進位的層次上，針對輸入與輸出張量寫出相應的程式碼。舉例來說，如果模型接受的是　個浮點數，我們就可以把這個浮點數放進　個 4 Byte 的 `Data` 型別中。為了方便起見，我會在 Swift 裡進行　些擴展（extension），詳情可參見本書的程式碼。而運用模型的做法，大致上就如圖 11-1 所示。

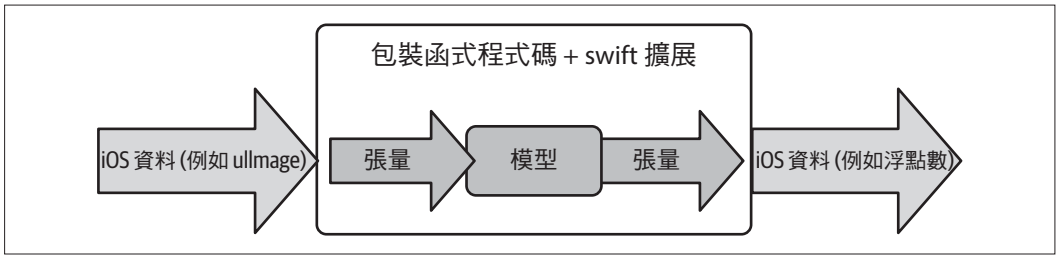

圖 11-1　在 iOS App 中使用模型

舉例來說，如果考慮我們在第 8 章所使用的那個簡單模型，它已透過學習知道數字間的關係為 y = 2x − 1，只要送入　個浮點數，它就會推測出相應的結果。舉例來說，只要把 10 這個值送進去，它就會送回 18.98 或某個很接近的值。雖然所送進去的值是　個浮點數，但實際上我們必須把浮點數的四個 Byte 載入到　個記憶體緩衝區，再把相應的 Pointer 指針送入模型中。舉例來說，假設我們的輸入放在 data 這個變數中，就必須使用下面這樣的程式碼，把它轉換成　個名為 buffer 的 UnsafeMutableBufferPointer：

```
let buffer: UnsafeMutableBufferPointer< Float > =
            UnsafeMutableBufferPointer(start: &data, count: 1)
```

這樣就會建立　個指針，指向 data 在記憶體內所存放的位置，而且由於我們把它的型別指定為 <Float>，並把 count 設定為 1，因此這個 buffer 就是指向 data 所在位址開始的 4 個 Byte。現在你應該明白我之前說「要在記憶體內針對 Byte 進行　些低階操作」這句話是什麼意思了吧！

接下來我們會把這個 buffer 轉換成 Data 型別，再把它複製到 interpreter 直譯器中，以做為第　個輸入張量，做法如下：

```
try interpreter.copy(Data(buffer: buffer), toInputAt: 0)
```

接著只要調用 interpreter 直譯器的 invoke 方法，就可以進行推測：

```
try interpreter.invoke()
```

如果想取得推測的結果，則必須查看輸出張量：

```
let outputTensor = try interpreter.output(at: 0)
```

我們知道 outputTensor 裡包含了　個 Float32 的結果，因此必須把 outputTensor 裡的資料（data）轉換成　個 Float32：

```
let results: [Float32] = [Float32](unsafeData: outputTensor.data) ?? []
```

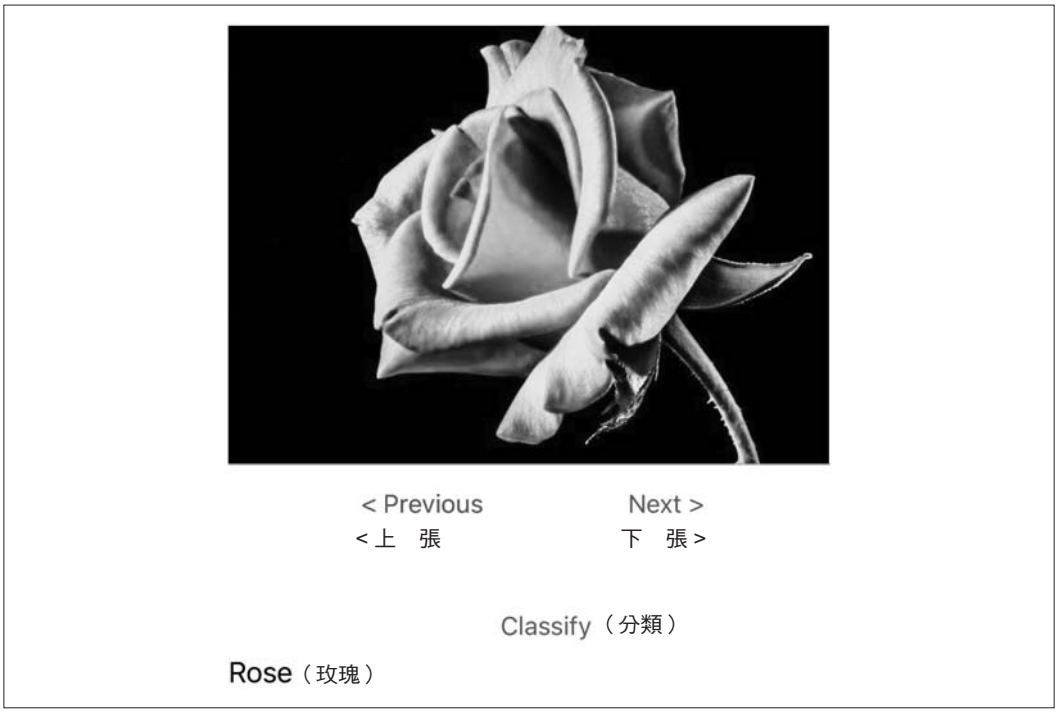

圖 11-2　採用自定義圖片分類模型的一個 App

這個 App 預先載入了好幾張不同類型花朵的圖片，只要按下「Previous」（上　張）與「Next」（下　張）的按鈕，就可切換到不同的圖片。如果按下「Classify」（分類）按鈕，它就會告訴你模型根據這張圖片所推測出的花朵類型。如果想讓這個 App 從你的照片或相機讀取圖片，程式修改起來應該也很簡單，但為了讓 App 盡量保持簡單，我們在這裡只預先載入幾張花朵的圖片。如果你想設計　下這個 App 的 Layout 版面，可直接開啟 *Main.storyboard*，然後把 Storyboard 設計成如圖 11-3 所示。

```
        print("Error loading model!")
        return
    }
```

現在已經把直譯器載入到記憶體，可以準備進行後續的工作了。接下來要做的就是提供　張圖片，讓模型可以對它進行解譯！

第 5 步：把圖片轉換成輸入張量

這個步驟非常複雜，因此在深入研究程式碼之前，我們先用比較直觀的方式，探索　下相關的概念。回頭看　下圖 11-1，你就會注意到 iOS 可以把圖片儲存為 UIImage，不過這樣的形式與模型可辨識的張量有很大的不同。首先我們來瞭解　下，圖片通常是如何保存在記憶體中的。

圖片中的每個像素，都是用 32 個 bit（也就是 4 個 Byte）來表示。這些位元組裡保存的是紅、綠、藍與 alpha 的強度。參見圖 11-5。

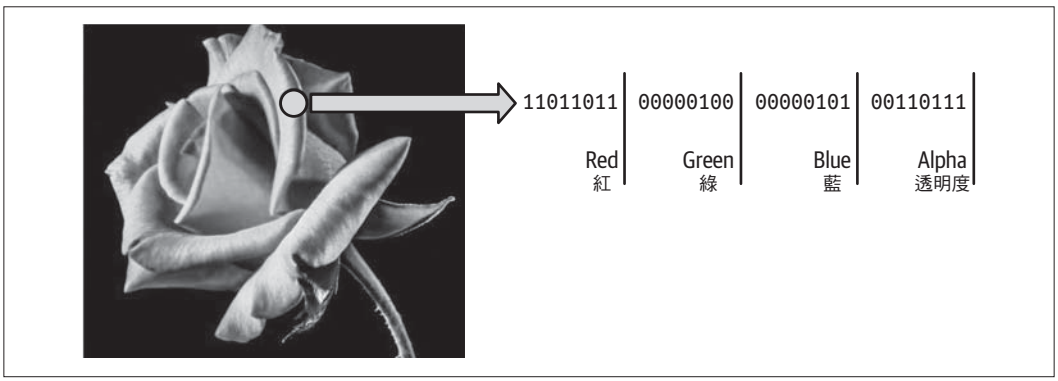

圖 11-5　圖片保存在記憶體內的方式

舉例來說，如果你的圖片是 1000 × 1000 像素，那麼保存這　張圖片的記憶體，就會是　百萬組的 4 個 Byte。整塊記憶體裡的第　組 4 個 Byte，就是最左上角的那個像素，然後下　個像素就是下　組的 4 個 Byte，其餘依此類推。

當我們（透過 Python 使用 TensorFlow）訓練模型來辨識圖片時，我們其實是利用　個可代表此圖片的張量來訓練模型。這樣的張量通常只包含紅、綠、藍三個通道，不包含 alpha。此外，張量的紅、綠、藍通道裡保存的並不是原始的 Byte 內容，而是歸　化過的 Byte 內容。舉例來說，圖 11-5 裡的那個像素，紅色通道為 11011011，也就是 219。歸　化有很多種不同的做法，不過我們選擇的是最簡單的做法，也就是直接除以 255，

第十二章

用 Firebase 來協助 App 產品化

本書到目前為止已探索過如何建立機器學習模型，也研究過如何使用各種技術，把這些模型整合到 Android 或 iOS App 中。我們可以使用 TensorFlow Lite 搭配 些低階操作，直接使用模型，並且處理好資料進出模型的 些轉換程序。針對許多常見的應用場景，我們也可以利用 ML Kit 裡 些高階 API 的優勢，透過非同步程式設計方式輕鬆打造出響應式（responsive）應用。不過，所有例子全都只是製作出 些很簡單的 App，只能在單 Activity 或 View 中進行推測。

如果你想讓 App 成為真正的產品，當然必須走得更遠，而 Firebase 設計的目的，就是想成為 種跨平台解決方案，協助我們建構、發展自己的 App 並從中獲利。

雖然針對 Firebase 全面性的討論已超出本書的範圍，但 Firebase 的免費方案（free tier，也叫 Spark 方案）有個重要又好用的功能可供我們使用：自定義模型架設服務（custom model hosting）。

為何要使用 Firebase 來架設自定義模型？

正如你在本書中所看到的，建立 個 ML 機器學習模型來為使用者解決問題並不困難。像 TensorFlow 或 TensorFlow Lite Model Maker 這類的工具，可以根據你的資料快速訓練模型，運用起來也相對簡單。比較困難的其實是如何建立「正確」的模型；要能夠做到這 點，我們就必須不斷與使用者共同進行測試並更新你的模型，以驗證相應的表現，而且不只要從速度或準確度的角度來看，還要觀察使用者對於你 App 的使用，在這些改

模型訓練完畢之後，我們就擁有了另　個 TFLite 模型，這個模型是以 MobileNet 做為其基礎，就放在 *mm_flowers2* 目錄中。下載之後記得要和第　個模型分開放。下　節我們就會把這兩個模型全都上傳到 Firebase。

使用 Firebase 模型架設服務

Firebase 模型架設服務（Firebase Model Hosting）可以讓我們在 Google 的基礎架構下，架設自己的模型服務。我們可以讓 App 下載並使用這些架設起來的模型，因此，只要使用者有連線能力，我們就可以針對哪些使用者使用哪　個模型，還有下載模型的方式，進行　些管理的工作。本節會探索相應的做法，不過我們必須先建立　個專案。

第 1 步：建立 Firebase 專案

如果要使用 Firebase，就必須在 Firebase 控制台建立　個 Firebase 專案。只要直接前往 *http://firebase.google.com*，就可以開始使用。你可以先嘗試　下其中的示範，或是觀看　段關於 Firebase 的影片。準備好之後，請點擊「Get started」（開始使用）。參見圖 12-1。

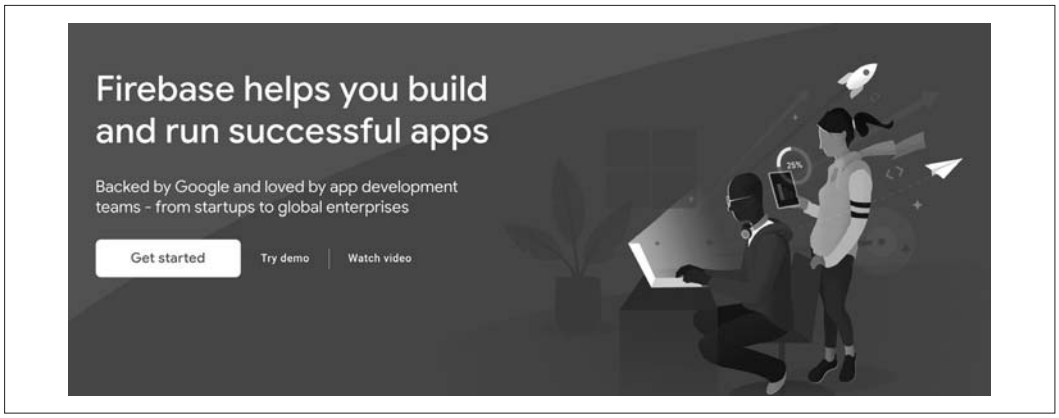

圖 12-1　開始使用 Firebase

點擊按鈕之後，就會進入控制台頁面，這裡會列出現有的專案。如果你是第　次使用，就只會看到「Add project」（添加專案）的按鈕，如圖 12-2 所示。

行動 App 存取雲端模型

我們整本書都在介紹如何建立模型，並把模型轉換成 TensorFlow Lite 格式，以便把模型運用到你的行動 App 中。在第 1 章討論了　些理由（如減少延遲、強化隱私），說明直接在行動裝置中使用模型確實是很好的　種做法。不過，有時你或許並不想直接把模型部署到行動裝置中——也許模型對行動裝置來說太過龐大或太複雜，或許你想經常更新模型，或者你不想承擔被人逆向工程的風險，留給他人隨意濫用你智慧財產權的機會。

在那樣的情況下，我們就會把模型部署在伺服器中，並運用伺服器執行推測，再以某種伺服器的形式管理客戶端的請求；調用模型進行推測之後，再把推測的結果當成回應送回來。圖 14-1 顯示的就是這種做法的高階概念圖。

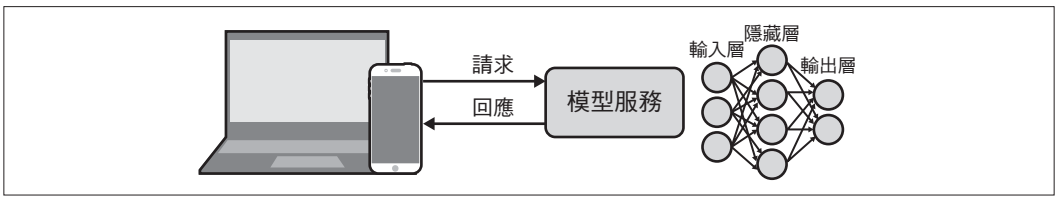

圖 14-1　模型伺服器架構的高階概念圖

這種架構的另　個好處，就是可以針對模型漂移（model drift）的情況進行管理。如果我們把模型部署在裝置中，萬　大家不去更新 App（或是無法更新 App），就無法取得最新的模型，到最後免不了會有多種模型同時存在的情況。另外有時候我們反而希望，多種不同版本的模型可以同時存在；例如使用的硬體設備比較高階的人，也許可以採用比較大又比較準確的模型版本，而其他人則可以採用比較小但準確度稍低的模型版本。原本這類的管理工作，或許相當困難！但如果把模型架設在伺服器中，就不必擔心這個問題了，因為我們完全可以控制，讓哪　種硬體平台執行哪　個模型。讓模型在伺服端

進行推測的另　個優點就是，我們可以輕易針對不同的使用者，測試不同的模型版本。參見圖 14-2。

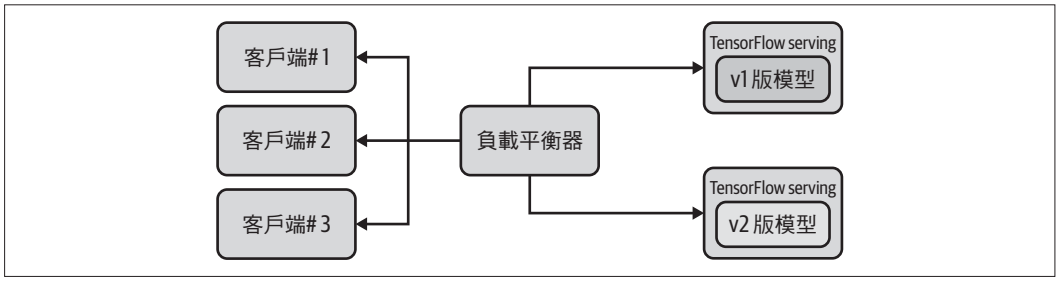

圖 14-2　在伺服端進行推測，即可輕鬆管理不同的模型

我們在這裡可以看到兩種不同版本的模型（分別叫做 v1 版模型、 v2 版模型），並利用負載平衡器部署到不同的客戶端。在這張圖中可以看到，我們採用了所謂的 TensorFlow Serving 來進行管理——接下來我們就會探索如何安裝與使用，並介紹如何訓練與部署個簡單的模型。

安裝 TensorFlow Serving

TensorFlow Serving 可安裝成兩種不同的伺服器架構。第　種就是 `tensorflow-model-server`，它是　個已完全最佳化的伺服器，也就是針對各種不同架構採用各平台專屬的編譯器選項。　般來說，這應該是首選的選項，除非你的伺服器找不到相應的架構。第二種替代方案則是 `tensorflow-model-server-universal`，它是採用最基本的最佳化方式來進行編譯，因此應該可適用於所有的機器，如果遇到 `tensorflow-model-server` 無法正常運作的情況，這就是　個很好的替代選項。TensorFlow Serving 有很多種不同的安裝方式，不但可以使用 Docker，也可以使用 `apt` 直接安裝套件。接著我們就來研究這兩種不同的做法。

用 Docker 進行安裝

Docker 是個很好用的工具，它可以把作業系統以及軟體依賴項目全都封裝到　個簡單好用的 image 映像。如果想快速啟動與執行某個東西，使用 Docker 或許就是最簡單的做法。首先，我們可以用 `docker pull` 來取得 TensorFlow Serving 套件：

```
docker pull tensorflow/serving
```

完成此動作之後，再到 GitHub 以 clone（克隆）的方式取得 TensorFlow Serving 的程式碼：

```
git clone https://github.com/tensorflow/serving
```

這裡頭包含　些範例模型，其中包括　個名為 Half Plus Two 的模型，只要給　個值，就會送回該值的　半加二的結果。如果想使用此模型，就要先設定　個名為 TESTDATA 的變數，然後把它的值設定為範例模型的路徑：

```
TESTDATA="$(pwd)/serving/tensorflow serving/servables/tensorflow/testdata"
```

然後就可以用 Docker image 映像來執行 TensorFlow Serving：

```
docker run  t   rm  p 8501:8501 \
     v "$TESTDATA/saved model half plus two cpu:/models/half plus two" \
     e MODEL NAME=half plus two \
    tensorflow/serving &
```

這樣就會在 8501 這個通訊埠，建立　個伺服器的實體（本章隨後還會更詳細說明這是怎麼做到的），然後在伺服器中執行所指定的模型。如此　來，我們就可以透過 *http://localhost:8501/v1/models/half_plus_two:predict* 來存取模型了。

如果想把資料送入模型執行推測，我們可以把　個包含值的張量，用 POST 的方式送往這個網址。下面就是利用 curl 來執行的　個範例（如果是在你開發程式的電腦中執行，請另外打開　個獨立的終端機程式來執行下面的指令）：

```
curl  d '{"instances": [1.0, 2.0, 5.0]}' \
     X POST http://localhost:8501/v1/models/half plus two:predict
```

圖 14-3 顯示的就是相應的結果。

圖 14-3　執行 TensorFlow Serving 的結果