前言

在影像上進行機器學習正在徹底改變醫療保健、製造、零售和許多其他領域的現狀與未來。許多以前的難題，現在可以透過訓練機器學習（machine learning, ML）模型來識別影像中的物件而解決。本書的目的是，提供對支援這個快速發展領域的機器學習架構的直覺解釋，並提供實用的程式碼來使用這些機器學習模型，以解決涉及分類、測量、偵測、分割、表達、產生、計數，還有其他的更多問題。

影像分類是深度學習的「hello world」。因此，本書還對深度學習進行了端到端的實用介紹，它可以作為其他深度學習領域的墊腳石，例如自然語言處理。

您將學習如何為電腦視覺任務設計 ML 架構，並使用那些以 TensorFlow 和 Keras 所編寫之流行的、且經過良好測試的預建構模型來進行模型的訓練。您還將學習到如何提高準確性和可解釋性的技術。最後，本書將教您如何設計、實作和調整端到端 ML 生產線（pipeline）以完成影像理解任務。

目標讀者

本書主要目標讀者是想要對影像進行機器學習的軟體開發人員，它適用於將會使用 TensorFlow 和 Keras，來解決常見電腦視覺使用案例的開發人員。

本書中討論的方法附有程式碼範例，這些程式碼範例可以從 *https://github.com/GoogleCloudPlatform/practical-ml-vision-book* 中取得。本書的大部分內容都涉及開源的 TensorFlow 和 Keras，無論您是在本地端、Google Cloud 還是其他雲端環境執行程式碼都可以。

有興趣使用 PyTorch 的開發人員，會發現本書中的文字解釋很有用，但可能需要在別的地方尋找實際的程式碼片段。我們也非常樂見各位也能貢獻與書中程式碼範例同等實用的 PyTorch 程式碼；請向我們的 GitHub 儲存庫提出拉取（pull）請求。

如何使用本書

我們建議您按順序閱讀本書。請確保閱讀、理解並執行本書 GitHub 儲存庫（*https://github.com/GoogleCloudPlatform/practical-ml-vision-book*）中所附的筆記本（notebook）——您可以在 Google Colab 或 Google Cloud 的 Vertex Notebook 中執行它們。建議您在閱讀每一段文本後動手嘗試程式碼，以確保您完全理解所介紹的概念和技術並強烈建議在繼續下一章之前完成每一章中的筆記本。

Google Colab 是免費的，足以執行本書中的大部分筆記本； Vertex Notebook 更強大，因此將能幫助您更快的執行筆記本。第 3、4、11 和 12 章中更複雜的模型和更大的資料集，會受益於 Google Cloud TPU 的使用。因為本書中所有程式碼都是使用開源的 API 來編寫的，所以這些程式碼也應該可以在安裝了最新版本 TensorFlow 的任何其他 Jupyter 環境中執行，無論是您的筆記型電腦、Amazon Web Services（AWS）Sagemaker 還是 Azure ML。但是，我們尚未在這些環境中對其進行測試；如果您發現必須進行任何更改，才能使程式碼在其他環境中運作的情形，請提交拉取請求以幫助其他讀者。

書中程式碼是在 Apache 開源授權下提供給您，它主要是用作教學工具，但也可以作為產出模型的起點。

本書架構

本書的其餘部分架構如下：

- 在第 2 章中，將介紹機器學習、如何讀取影像以及如何使用 ML 模型進行訓練，評估和預測。第 2 章介紹的模型是泛用的，因此在影像上執行得並不是特別好，但是本章中介紹的概念，對於本書的其餘部分是不可或缺的。
- 在第 3 章中，將介紹了一些在影像上運作良好的機器學習模型。從遷移學習（transfer learning）和微調（fine-tuning）開始，然後介紹各種卷積（convolutional）模型，隨著越來越深入本章，這些模型的複雜性也會增加。

- 在第 4 章中，探討如何使用電腦視覺，來解決物件偵測和影像分割問題。第 3 章中介紹的任何骨幹架構，都可以在第 4 章中使用。

- 第 5 章到第 9 章深入研究了建立生產（production）電腦視覺機器學習模型的細節。我們將逐步瞭解標準的 ML 生產線，檢視第 5 章中的資料集建立、第 6 章中的前置處理、第 7 章中的訓練、第 8 章中的監控和評估，以及第 9 章中的部署。這些章節中所討論的方法，適用於第 3 章和第 4 章中討論的任何模型架構和使用案例。

- 第 10 章討論了三個新興趨勢。我們將第 5 章到第 9 章中涵蓋的所有步驟，連結到一個端到端、容器化的 ML 生產線中，然後嘗試了一個無程式碼影像分類系統，該系統可用於快速原型設計，並作為更多客製化模型的基準。最後，我們展示了如何將可解釋性（explainability）建構到影像模型預測中。

- 第 11 章和第 12 章展示了如何使用電腦視覺的基本積木，來解決各種問題，包括影像產生、計數、姿勢偵測等，並為這些進階使用案例提供了實作。

本書字體慣例

本書使用以下的字體慣例：

斜體（*Italic*）

指出新字、網址、電子郵件地址、檔名、以及副檔名。

定寬（`Constant width`）

用於程式列表、以及在段落中提及的程式元素，例如變數或函數名稱、資料庫、資料型別、環境變數、敘述、以及關鍵字。

定寬粗體（**`Constant width bold`**）

顯示命令或其他應由使用者輸入的文字。

定寬斜體（*`Constant width italic`*）

顯示應該被使用者所提供或由語境（context）決定的值所取代的文字。

這個元素用來提出一個提示或建議。

第一章

電腦視覺之機器學習

想像一下，您正坐在花園裡，觀察周圍發生的事情。您的身體有兩個系統在工作：您的眼睛充當感測器（sensor）並建立場景的表達法；而您的認知系統正在理解您的眼睛所看到的東西。因此，您可能會看到一隻鳥、一條蟲子和一些動作，並意識到這隻鳥已經沿著小徑走過去並正在吃一條蟲子（見圖 1-1）。

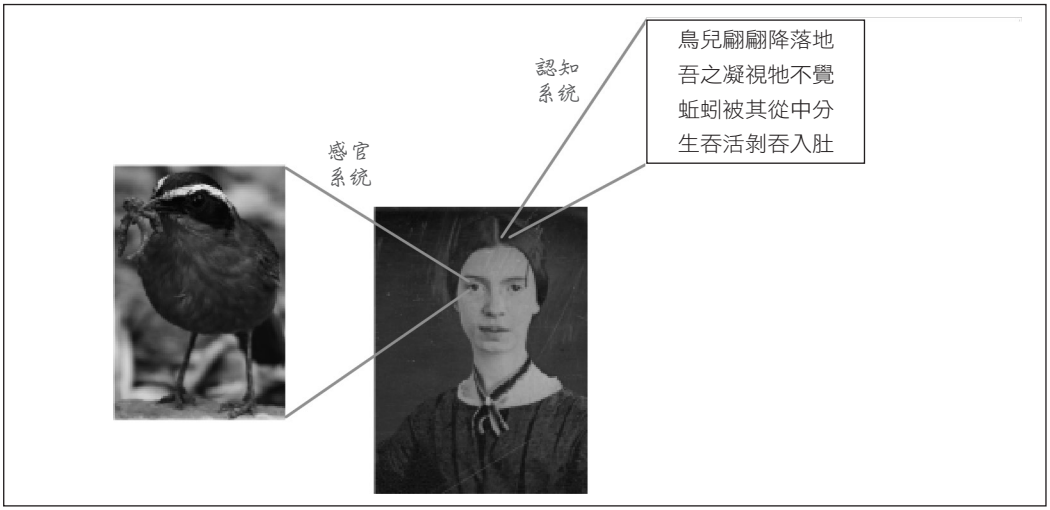


圖 1-1　人類視覺涉及我們的感官和認知系統。

電腦視覺試圖透過提供影像形成（模仿人類**感官**（*sensory*）系統）和機器感知（模仿人類**認知**（*cognitive*）系統）的方法來模仿人類視覺能力。對人體感官系統的模仿側重於硬體以及感測器（如相機）的設計和置放；模仿人類認知系統的現代方法，包括用於從影像中萃取資訊的機器學習 （ML） 方法。我們在本書中介紹的正是這些方法。

2 例如，當我們看到雛菊的照片時，我們的人類認知系統能夠將其識別為雛菊（見圖 1-2）。我們在本書中所建構的用於影像分類的機器學習模型，將從雛菊的照片開始模仿人類的這種能力。

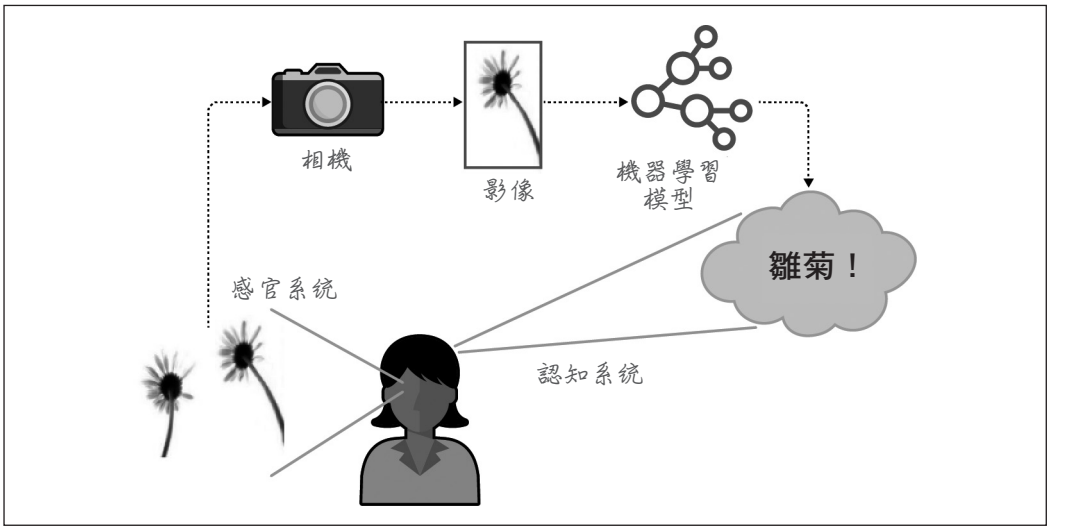


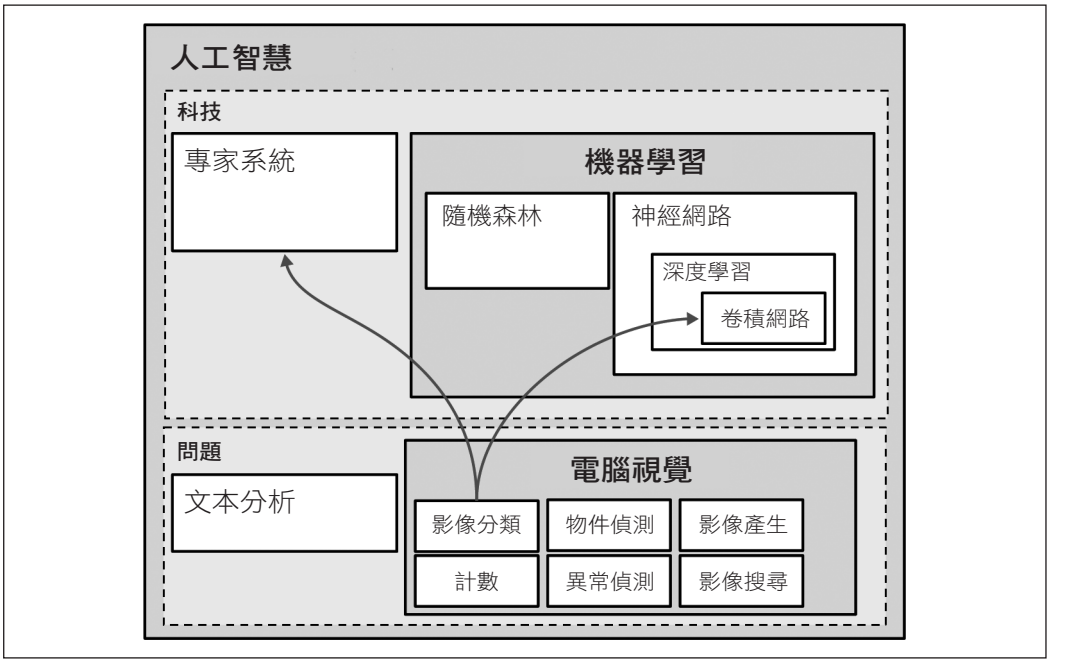
圖 1-2　影像分類機器學習模型模仿人類的認知系統。

機器學習

如果你在 2010 年代初期讀了一本關於電腦視覺的書，那麼用於從照片中萃取資訊的方法並不會涉及機器學習；相反的，您將學習去除雜訊、邊緣尋找、紋理偵測和形態（基於形狀）操作。隨著人工智慧的進步（更具體地說，機器學習的進步），這種情況發生了變化。

人工智慧（artificial intelligence, AI）探索了電腦可以模仿人類能力的方法。**機器學習**（*machine learning*）是 AI 的一個子領域，它透過向電腦展示大量資料並指導它們從中學習來教電腦做到這一點。**專家系統**（*expert system*）是人工智慧的另一個子領域 —— 專家系統透過以程式設計方式，讓電腦遵循人類邏輯來教導電腦模仿人類的能

力。在 2010 年代之前，影像分類等電腦視覺任務，通常是透過建構能夠實作出專家所制定的邏輯的訂製影像過濾器來完成的。如今，影像分類是透過卷積網路（convolutional network）達成，這是一種深度學習形式（見圖 1-3）。



3 圖 1-3　電腦視覺是人工智慧的一個子領域，試圖模仿人類的視覺系統；雖然它過去依賴於專家系統方法，但現在則是透過機器學習完成的。

以圖 1-2 中的雛菊影像為例。機器學習方法透過向電腦顯示大量影像及其標籤（*label*）（或正確答案）來教導電腦去識別影像中的花朵類型。因此，我們會向電腦顯示大量雛菊影像、大量鬱金香影像等等。基於這樣一個**已標記訓練資料集**（*labeled training dataset*），電腦會學習如何對它以前沒有遇到過的影像進行分類。第 2 章和第 3 章討論了這件事是如何發生的。

另一方面，在專家系統方法中，我們會先採訪人類植物學家，瞭解他們如何對花卉進行分類。如果植物學家解釋說 *bellis perennis*（雛菊的學名）是由圍繞黃色中心的白色細長花瓣和綠色圓形葉子組成的，我們會嘗試設計影像處理過濾器來比對這些標準。例如，我們會尋找影像中白色、黃色和綠色的盛行率（prevalence）。然後我們會設計邊緣過濾器來識別葉子的邊界，並設計匹配的形態（morphological）過濾器來查看它們是否與預期的圓形相匹配。我們可能會在 HSV（色調、飽和度、值）空間中平滑化影像，

以決定與花瓣顏色相比之下的花朵中心的顏色。根據這些標準，我們可能會為影像計算一個分數，以評估它是雛菊的可能性；同樣的，我們會為玫瑰、鬱金香、向日葵等設計和應用不同的規則集合。為了對新影像進行分類，我們會選擇該影像得最高分的那個類別。

4 上面的描述說明了建立影像分類模型所需的大量訂製工作。這就是為何影像分類在過去適用性有限原因。

隨著 AlexNet 論文（*https://dl.acm.org/doi/10.1145/3065386*）的發表，這一切在 2012 年發生了變化。該論文作者 ——Alex Krizhevsky、Ilya Sutskever 和 Geoffrey E. Hinton —— 將卷積網路（在第 3 章中會介紹）應用於 ImageNet 大規模視覺識別挑戰賽（ImageNet Large-Scale Visual Recognition Challenge, ILSVRC）中使用的基準資料集，並且能夠大幅超越任何現有的影像分類方法。他們達成了 15.3% 的前 5 名[1] 錯誤率，而亞軍的錯誤率則超過了 26%。像這樣的比賽中典型的改進大約是 0.1%，所以 AlexNet 展示的改進是大多數人預期的一百倍之多！這是非常引人注目的表現。

神經網路（neural networks）在 1970 年代就已經存在（*https://oreil.ly/IRHqY*），而自那時起大約 20 年後卷積神經網路（convolutional neural network, CNN）就出現了 ——Yann LeCun 在 1989 年就提出了這個想法（*https: //oreil.ly/EqY3a*）。那麼，AlexNet 有什麼新鮮事呢？就是這四件事：

圖形處理器（*graphics processing unit, GPU*）

卷積神經網路是一個好主意，但它們在計算上非常昂貴。AlexNet 的作者在稱為 GPU 的特殊用途晶片所提供的圖形渲染（graphics rendering）程式庫上，實作了一個卷積網路。當時，GPU 主要用於高端視覺化和遊戲上。該論文將卷積分組以適配跨兩個 GPU 的模型。GPU 使卷積網路的訓練成為可能（我們將在第 7 章討論跨 GPU 的分散模型訓練）。

整流線性單元（*rectified linear unit, ReLU*）的激發

AlexNet 的建立者在他們的神經網路中使用了一種稱為 ReLU 的非飽和激發函數（non-saturating activation function）。我們將在第 2 章中更詳細的討論神經網路和激發函數；現在，知道使用分段（piecewise）線性非飽和激發函數能使他們的模型收斂得更快就足夠了。

1 前 5 名準確度（*top-5 accuracy*）意味著如果模型在結果的前 5 項中傳回影像正確標籤的話，我們就認為模型是正確的。

5 正則化（*regularization*）

ReLU 的問題 —— 以及它們直到 2012 年才被大量使用的原因 —— 是因為它們不會飽和，神經網路的權重在數值上會變得不穩定。 AlexNet 的作者使用了一種正則化技術來防止權重變得太大。我們也會在第 2 章討論正則化。

深度

由於能夠更快的進行訓練，他們能夠訓練具有更多神經網路層的複雜模型。我們會說層數越多的模型較深；深度的重要性將在第 3 章中討論。

值得肯定的是，正是神經網路深度的增加（前三個想法的結合所促成的）造成 AlexNet 的世界領先地位。CNN 可以使用 GPU 來加速，這一點已在 2006 年得到證實（*https://oreil.ly/9p3Ba*）。ReLU 激發函數本身並不新鮮，正則化更是一種眾所周知的統計技術。最終而言，該模型的卓越性能要歸功於作者的洞察力，也就是他們可以將所有這些結合起來以訓練比以前更深的卷積神經網路。

深度對於人們對神經網路重新興起的興趣非常重要，以至於整個領域都被稱為深度學習（*deep learning*）。

深度學習使用案例

深度學習是機器學習的一個分支，它使用多層神經網路。深度學習優於先前存在的電腦視覺方法，現在已成功應用於許多其他形式的非結構化資料：視訊、音訊、自然語言文本等。

深度學習使我們能夠從影像中萃取資訊，而無需建立訂製的影像處理過濾器，或用程式設計出人類的邏輯。在使用深度學習進行影像分類時，我們需要成百上千甚至數百萬張影像（越多越好），而且我們知道它們正確的標籤（如「鬱金香」或「雛菊」）。這些標記的影像，可用於訓練影像分類深度學習模型。

只要您能將任務制定為自資料中學習的話，就可以使用電腦視覺機器學習方法來解決問題。例如，考慮光學字元辨識（optical character recognition, OCR）的問題 —— 獲取掃描影像並從中萃取文本。最早的 OCR 方法涉及教導電腦對單一字母的外觀進行樣式匹配（pattern matching）。由於一些原因，這被證明是種具有挑戰性的方法。例如：

- 有多種字體存在，因此一個字母可以用多種方式書寫。
- 字母有不同的大小，因此樣式匹配必須是尺度不變（scale-invariant）的。
- 裝訂的書籍無法平放，因此掃描的字母會變形。
- 僅辨識單一字母是不夠的；我們需要萃取整個文本。構成單字、行或段落的規則很複雜（參見圖 1-4）。

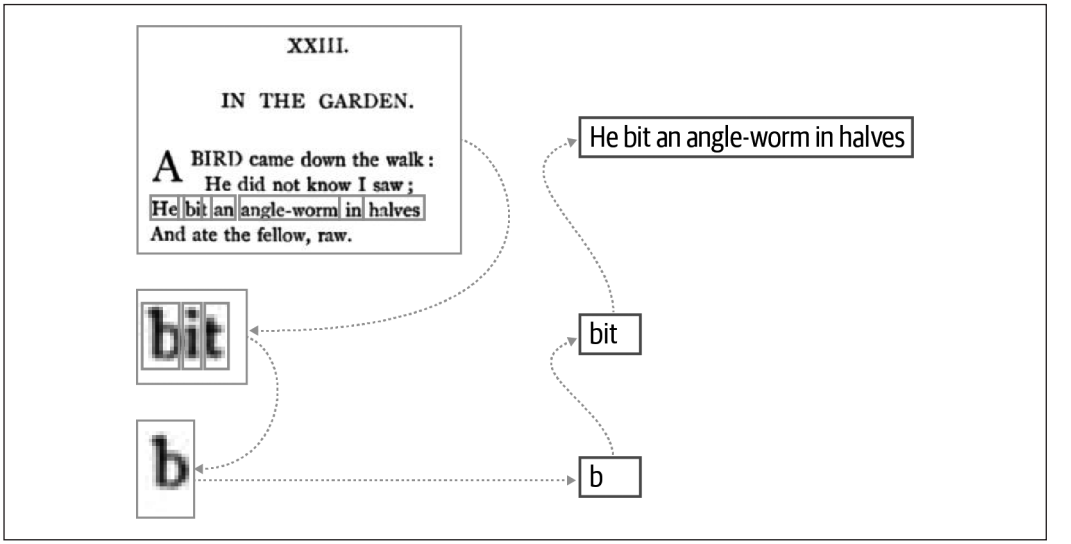


圖 1-4　基於規則的光學字元辨識需要識別線條，將它們分解成單字，然後識別每個單字的組成字母。

另一方面，透過使用深度學習，OCR 可以很容易地制定為影像分類系統。有許多書籍已經數位化，可以透過向模型展示書籍的掃描影像，並使用數位化文本作為標籤來訓練模型。

電腦視覺方法為各種真實世界的問題提供了解決方案。除 OCR 外，電腦視覺方法已成功應用於醫學診斷（使用 X 光和 MRI 等影像）、自動化零售運算（例如讀取二維條碼、識別空的貨架、檢查蔬菜品質等）、監控（透過衛星影像監控作物產量、監控野生動物攝影機、入侵者偵測等）、指紋辨識，以及汽車安全（在安全距離內跟車、透過路標辨識速限變化、自動停車、自動駕駛等）。

7 電腦視覺已應用在許多行業中。在政府中，它已被用於監控衛星影像、建設智慧城市以及海關和安全檢查；在醫療保健領域，它已被用於識別眼部疾病，並從乳房 X 光照片中發現癌症的早期跡象；在農業中，它已被用於發現故障的灌溉幫浦、評估作物產量和識別葉子的疾病；在製造業中，它可用於工廠間的品質控制和視覺檢查，在保險領域，它已被用於在事故發生後自動評估車輛的損壞情況。

總結

電腦視覺幫助電腦理解照片等數位影像的內容。從 2012 年的一篇開創性論文開始，電腦視覺的深度學習方法已經取得了巨大的成功。如今，我們發現電腦視覺在眾多行業中得到了成功應用。

我們將在第 2 章中透過建立我們的第一個機器學習模型來開始我們的旅程。

第二章

視覺機器學習模型

在本章中，您將學習如何表達影像並訓練基本的機器學習模型來對影像進行分類。您會發現線性和完全連接神經網路在影像上的效能很差。但是，在此過程中，您將學習到如何使用 Keras API 來實作 ML 原語（primitive）以及訓練 ML 模型。

本章的程式碼位於本書 GitHub 儲存庫（*https://github.com/GoogleCloudPlatform/practical-ml-vision-book*）的 *02_ml_models* 資料夾中。我們提供適用於該處的程式碼範例和筆記本檔名。

用於機器感知的資料集

就本書的目標而言，如果針對一個實際問題並建構各種機器學習模型來解決它的話，將會很有幫助。假設我們已經收集並標記了近四千張花朵照片的資料集。*5-flowers* 資料集中有五種類型的花朵（參見圖 2-1），資料集中的每張影像都已經用它所呈現的花朵類型進行了標記。

圖 2-1　5-flowers 資料集中的照片有五種花：雛菊、蒲公英、玫瑰、向日葵和鬱金香。

假設我們想要建立一個電腦程式，當提供影像時，它會告訴我們影像中的花是什麼類型。我們要求機器學習模型去學習感知影像中的內容，因此您可能會看到這種任務被稱為**機器感知**（*machine perception*）。具體來說，感知的類型類比於人類的視覺，所以這個課題被稱為**電腦視覺**（*computer vision*），在此案例中我們將透過影像分類來解決它。

5-Flowers 資料集

5-flowers 資料集是由 Google 建立的，並以創用 CC（Creative Commons）授權置於公共領域（public domain）。它以 TensorFlow 資料集（*https://oreil.ly/tqwFi*）進行發布，並以 JPEG 檔案的形式，在公共的 Google Cloud Storage 儲存桶（`gs://cloud-ml-data/`）中提供使用。這使得此資料集既逼真（它由現成的相機所收集的 JPEG 照片組成）又易於存取。因此，我們將在本書中持續使用它作為範例。

一個範例，但不是模板

5-flowers 資料集是一個很好的學習資料集，但您不應將其用作建立訓練資料集的模板（template）。從作為模板的角度來看，有幾個因素使 5-flowers 資料集不符合標準：

數量
: 要從頭開始訓練 ML 模型，您通常需要收集數百萬張影像。有一些替代方法可以處理較少的影像，但您應該嘗試收集最大的實用（且合乎倫理！）的資料集。

資料格式
: 將影像儲存為單獨的 JPEG 檔案是非常沒有效率的，因為您的大部分模型訓練時間，都將花費在等待資料的讀取上。最好是使用 TensorFlow Record 格式。

內容
: 資料集本身由**找到的**資料所組成 —— 並不是只包含針對該分類任務而收集的影像。如果您的問題領域允許的話，您應該更有目的性的收集資料。稍後會詳細說明。

標籤
: 影像標籤本身就是一個主題。本資料集是手動標記的。對於較大的資料集來說，這件事可能會變得不切實際。

我們將在整本書中討論這些因素並提供有關如何設計、收集、組織、儲存和標記資料的最佳實務。

在圖 2-2 中，您可以看到幾張鬱金香照片。請注意，它們的範圍涵蓋了從特寫照片到鬱金香田的照片，這些全部都是人類可以毫無問題的標記為鬱金香的照片，但如果只能使用簡單的規則，對我們來說會是一個困難的問題 —— 例如，如果我們說鬱金香是一朵細長的花，那只有第一張和第四張影像合格。

圖 2-2　這五張鬱金香照片在焦段、花色，以及和圖框中的內容方面差異很大。

標準化影像收集

我們選擇要解決一個難題，也就是花朵影像都是在真實世界情況下收集的。但是，在實務中，您通常可以透過將影像的收集方式，進行標準化來簡化機器感知問題。例如，您可以指定必須在受控條件、使用泛光打法（flat lighting）和相同變焦下收集影像。這在製造業中很常見 —— 可以精確指定工廠的條件。以這樣的方式設計掃描器也很常見，也就是物體只能以一個方向來放置。作為機器學習從業者，您應該尋找使機器感知問題更容易的方法。這不是作弊 —— 這是為成功做好準備的明智之舉。

但是請記住，您的訓練資料集必須能夠反映模型要進行預測時所需的條件。如果您的模型僅針對專業攝影師拍攝的花朵照片進行訓練，那麼對於業餘愛好者拍攝的照片，其照明、變焦和取景選擇可能會有所不同，它的效果可能會很差。

讀取影像資料

為了訓練影像模型，我們需要將影像資料讀入我們的程式。要能讀取 JPEG 或 PNG 等標準格式的影像，並準備好用它來訓練機器學習模型這件事包含四個步驟（完整程式碼可在本書的 GitHub 儲存庫中的 *02a_machine_perception.ipynb* 中找到）：

```
import tensorflow as tf
def read_and_decode(filename, reshape_dims):
    # 1. 讀取檔案。
    img = tf.io.read_file(filename)
    # 2. 將壓縮的字串轉換為 3D uint8 張量。
    img = tf.image.decode_jpeg(img, channels=3)
    # 3. 將 3D uint8 轉換為 [0,1] 範圍間的浮點數。
    img = tf.image.convert_image_dtype(img, tf.float32)
    # 4. 調整影像為想要的大小。
    return tf.image.resize(img, reshape_dims)
```

我們首先從持久性儲存裝置中讀取影像資料到記憶體中成為位元組序列：

```
img = tf.io.read_file(filename)
```

此處的變數 `img` 是一個包含位元組陣列的張量（tensor）（請參閱第 13 頁的〈張量是什麼？〉）我們剖析這些位元組以將它們轉換為像素資料 —— 這也稱為解碼（*decoding*）資料，因為像 JPEG 這樣的影像格式，要求您從查找表（lookup table）來解碼像素值：

```
img = tf.image.decode_jpeg(img, channels=3)
```

在這裡，我們指定我們只需要 JPEG 影像中的三個顏色頻道（紅色、綠色和藍色），然後不要不透明度（opacity），即第四個頻道。您可用的頻道取決於檔案本身。灰階（grayscale）影像可能只有一個頻道。

像素將由 `uint8` 型別且在 [0,255] 範圍內的 RGB 值來組成。因此，在第三步中，我們將它們轉換為浮點數，並將值縮放到範圍 [0,1] 內。這是因為機器學習優化器有經過調整，可以妥善的處理小數值：

```
img = tf.image.convert_image_dtype(img, tf.float32)
```

最後，我們將影像調整為所需的大小。機器學習模型是被建構成處理已知大小的輸入。由於真實世界中的影像可能具有任意的大小，因此您可能需要縮小、裁剪或擴展它們以適配您所需的大小。例如，要將影像調整為 256 像素寬和 128 像素高，我們將指定：

```
tf.image.resize(img,[256, 128])
```

在第 6 章中，我們將看到此方法並不會保留長寬比（aspect ratio），而且我們也會查看其他調整影像大小的選項。

張量是什麼？

在數學中，一維陣列被稱為向量（vector），二維陣列稱為矩陣（matrix）。**張量**（*tensor*）是一個可以有任意維數的陣列（維度數稱為**秩**（*rank*））。一個 12 列（row）18 行（column）的矩陣會被說成是具有 (12, 18) 的形狀（shape）和 2 的秩。因此，張量可以具有任意形狀。

Python 中常見的陣列數學程式庫稱為 `numpy`。您可以使用這個程式庫來建立 *n* 維陣列，但問題是它們不是硬體加速（hardware-accelerated）的。例如，這是一個形狀為 (4) 的一維陣列：

```
x = np.array([2.0, 3.0, 1.0, 0.0])
```

而這是一個 5D 的零陣列（請注意，形狀中有五個數字）：

```
x5d = np.zeros(shape=(4, 3, 7, 8, 3))
```

要使用 TensorFlow 獲得硬體加速，您可以使用下列方式將任一 `numpy` 陣列轉換為張量，也就是 TensorFlow 表達陣列的方式：

```
tx = tf.convert_to_tensor(x, dtype=tf.float32)
```

您可以使用以下方法將張量轉換回 `numpy` 陣列：

```
x = tx.numpy()
```

在數學上，`numpy` 陣列和 TensorFlow 張量是同一回事。然而，有一個重要的實際區別 —— 所有的 `numpy` 算術都是在 CPU 上完成的，而 TensorFlow 程式碼則是在 GPU 上執行（如果有的話）。因此，做這件事：

```
x = x * 0.3
```

一般會比下面這樣更沒效率：

```
tx = tx * 0.3
```

一般來說，使用 TensorFlow 運算的次數越多，程式的效率就越高。如果您對程式碼進行**向量化**（*vectorize*）（以處理批次影像），這樣您就可以執行單一原地（in-place）張量運算，而不是一堆微小的純量（scalar）運算，那麼效率也會更高。

這些步驟不是一成不變的。如果您的輸入資料包含以頻帶交錯（band interleaved）格式提供的衛星遙測影像，或以醫學數位成像與通訊（Digital Imaging and Communications in Medicine, DICOM）格式提供的腦部掃描影像的話，您顯然不會使用 `decode_jpeg()` 來解碼這些影像。同樣的，您可能不會總是需要調整影像的大小。在某些情況下，您可能會選擇將資料裁剪到所需的大小或用零來填充它。在其他情況下，您可能會調整大小、保持長寬比不變、然後填充剩餘的像素。這些前置處理運算將在第 6 章中討論。

視覺化影像資料

永遠要視覺化（visualize）一些影像以確保您已正確的讀取資料 —— 一個常見的錯誤是以旋轉或鏡像影像的方式來讀取資料。將影像視覺化也有助於瞭解機器感知問題的挑戰性。

我們可以使用 Matplotlib 的 `imshow()` 函數來視覺化影像，但為此我們必須首先使用 `numpy()` 函數將影像（TensorFlow 張量）轉換為 `numpy` 陣列。

```
def show_image(filename):
    img = read_and_decode(filename, [IMG_HEIGHT, IMG_WIDTH])
    plt.imshow(img.numpy());
```

請在我們的其中一張雛菊影像的上試用它，我們會得到如圖 2-3 所示的結果。

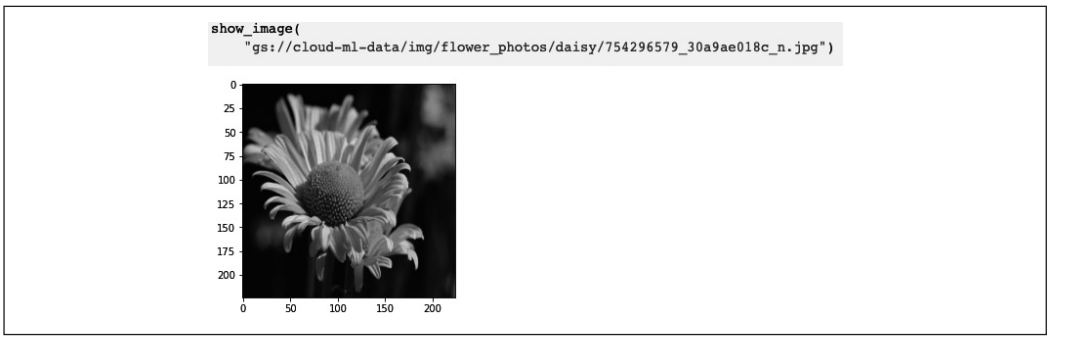


圖 2-3　確保將資料視覺化以確保您正確讀取資料。

請注意圖 2-3 中的檔案名稱包含了花的類型（daisy，雛菊）。這意味著我們可以使用 TensorFlow 的 `glob()` 函數來進行萬用（wildcard）匹配以獲取所有鬱金香影像：

```
tulips = tf.io.gfile.glob(
    "gs://cloud-ml-data/img/flower_photos/tulips/*.jpg")
```

圖 2-2 顯示了執行此程式碼來視覺化包含了五張鬱金香照片的面板的結果。

讀取資料集檔案

我們現在知道如何讀取影像了。但是為了訓練機器模型，我們需要讀取許多影像。我們還必須獲得每張影像的標籤。我們可以透過使用 `glob()` 來進行萬用匹配以獲取所有影像的串列：

```
tf.io.gfile.glob("gs://cloud-ml-data/img/flower_photos/*/*.jpg")
```

然後，在得知我們資料集中的影像有命名慣例的情況下，我們可以取一個檔名並使用字串運算來萃取標籤。例如，我們可以使用以下方法刪除字首：

```
basename = tf.strings.regex_replace(
    filename,
    "gs://cloud-ml-data/img/flower_photos/", "")
```

並使用以下方法取得類別：

```
label = tf.strings.split(basename, '/')[0]
```

像往常一樣，請參閱本書的 GitHub 儲存庫以獲取完整程式碼。

然而，出於泛化和可重複性的原因（在第 5 章中會進一步解釋），最好提前保留我們將用於評估的影像。在 5-flowers 資料集中這件事已經完成，用於訓練和評估的影像被列在與影像同一個 Cloud Storage 儲存桶中的兩個檔案中：

```
gs://cloud-ml-data/img/flower_photos/train_set.csv
gs://cloud-ml-data/img/flower_photos/eval_set.csv
```

這些是逗號分隔值（comma-separated value, CSV）檔案，其中的每一列包含一個檔案名稱，後面跟著它的標籤。

讀取 CSV 檔案的一種方法是使用 `TextLineDataset` 來讀取文本行，我們傳入一個函數來處理透過 `map()` 函數讀取的每一行：

```
dataset = (tf.data.TextLineDataset(
    "gs://cloud-ml-data/img/flower_photos/train_set.csv").
    map(parse_csvline))
```

我們正在使用 tf.data API，它可以透過一次僅讀取少量資料元素並在讀取時同時執行轉換來處理大量資料(即使它不能全部放入記憶體中)。這件事是透過使用了稱為 tf.data.Dataset 的抽象化來表達元素的序列來達成的。在我們的生產線中，每個元素都是一個包含了兩個張量的訓練範例。第一個張量是影像，第二個則是標籤。不同類型的 Dataset 會對應到不同的檔案格式。我們正在使用 TextLineDataset，它會讀取文本檔案並假設每一行都是不同的元素。

parse_csvline() 是我們提供的一個函數，用於解析行、萃取影像的檔名、讀取影像、並傳回影像及其標籤：

```
def parse_csvline(csv_row):
    record_defaults = ["path", "flower"]
    filename, label = tf.io.decode_csv(csv_row, record_defaults)
    img = read_and_decode(filename, [IMG_HEIGHT, IMG_WIDTH])
    return img, label
```

傳入 parse_csvline() 函數的 record_defaults，指明了在處理缺少一個或多個值的那些行時，TensorFlow 需要替換的內容。

為了驗證此程式碼是否有效，我們可以印出訓練資料集的前三個影像的每個頻道平均像素值：

```
for img, label in dataset.take(3):
    avg = tf.math.reduce_mean(img, axis=[0, 1])
    print(label, avg)
```

在此程式碼片段中，take() 方法將資料集截斷為包含三個項目。請注意，因為 decode_csv() 傳回了一個元組 (img, label)，這就是我們在迭代資料集時會獲得的。印出整個影像是一個糟糕的主意，因此我們使用 tf.reduce_mean() 來印出影像中的平均像素值。

結果的第一行是（為了可讀性添加了換行字元）：

```
tf.Tensor(b'daisy', shape=(), dtype=string)
tf.Tensor([0.3588961  0.36257887 0.26933077],
          shape=(3,), dtype=float32)
```

請注意，標籤是一個字串張量，平均值是長度為 3 的一維張量。為什麼我們會得到一維張量呢？那是因為我們向 reduce_mean() 傳入了一個 axis 參數：

```
avg = tf.math.reduce_mean(img, axis=[0, 1])
```

如果我們沒有提供軸的話，那麼 TensorFlow 就會計算所有維度的平均值並傳回一個純量值。回想一下，影像的形狀是 `[IMG_HEIGHT, IMG_WIDTH, NUM_CHANNELS]`。因此，透過提供 `axis=[0, 1]`，我們要求 TensorFlow 計算所有行（`axis=0`）和所有列（`axis=1`）的平均值，而不是平均的 RGB 值（參見圖 2-4）。

還有另一個原因使得像這樣印出影像的統計資料是有幫助的。如果您的輸入資料已損壞並且您的影像中存在無法表達的浮點資料（技術上稱為`NaN`（*https://oreil.ly/E0xc2*））的話，那麼平均值本身也將是`NaN`。這是確保您在讀取資料時沒有出錯的便捷方法。

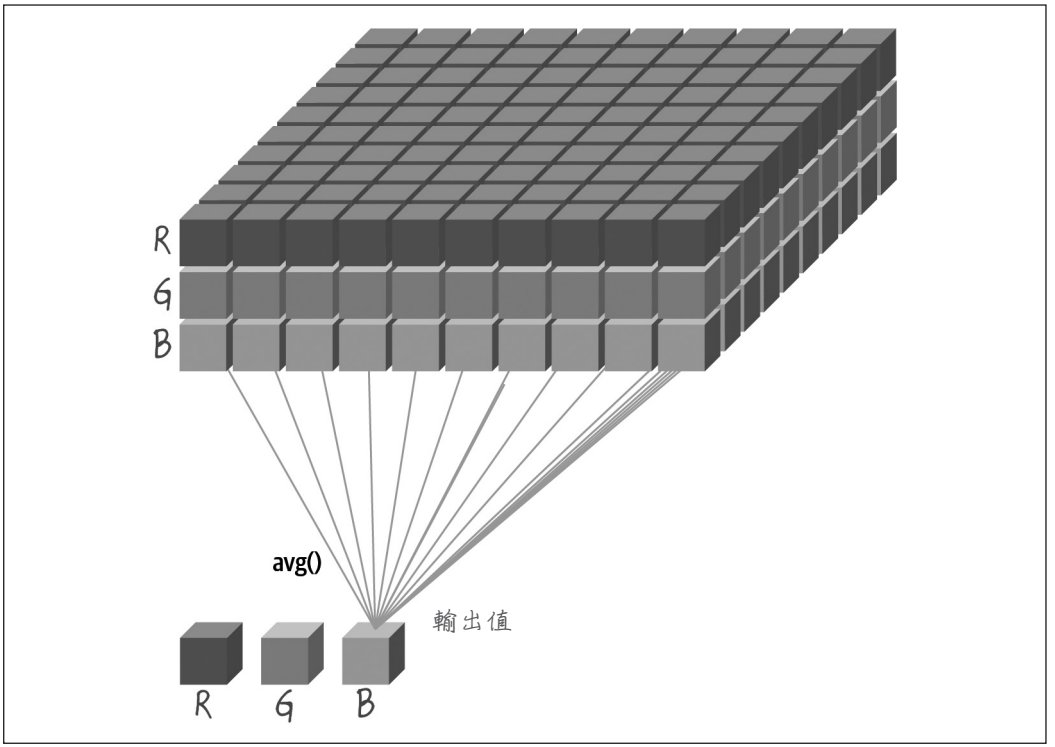


圖 2-4　我們沿影像的列軸和行軸計算 `reduce_mean()`。

使用 Keras 的線性模型

如圖 2-4 所示，`reduce_mean()` 函數對影像中的每個像素值賦予同樣的權重。如果我們對影像中的每個「寬 * 高 * 3 個像素頻道點」應用不同的權重會怎樣呢？

給定一張新影像，我們可以計算其所有像素值的加權平均值。然後我們可以使用這個值在五種花朵中進行選擇。因此，我們將計算 5 個這樣的加權平均值（這樣我們實際上是在學習 寬 * 高 * 3 * 5 個權重值；見圖 2-5），並根據輸出最大的情況來選擇花朵類型。

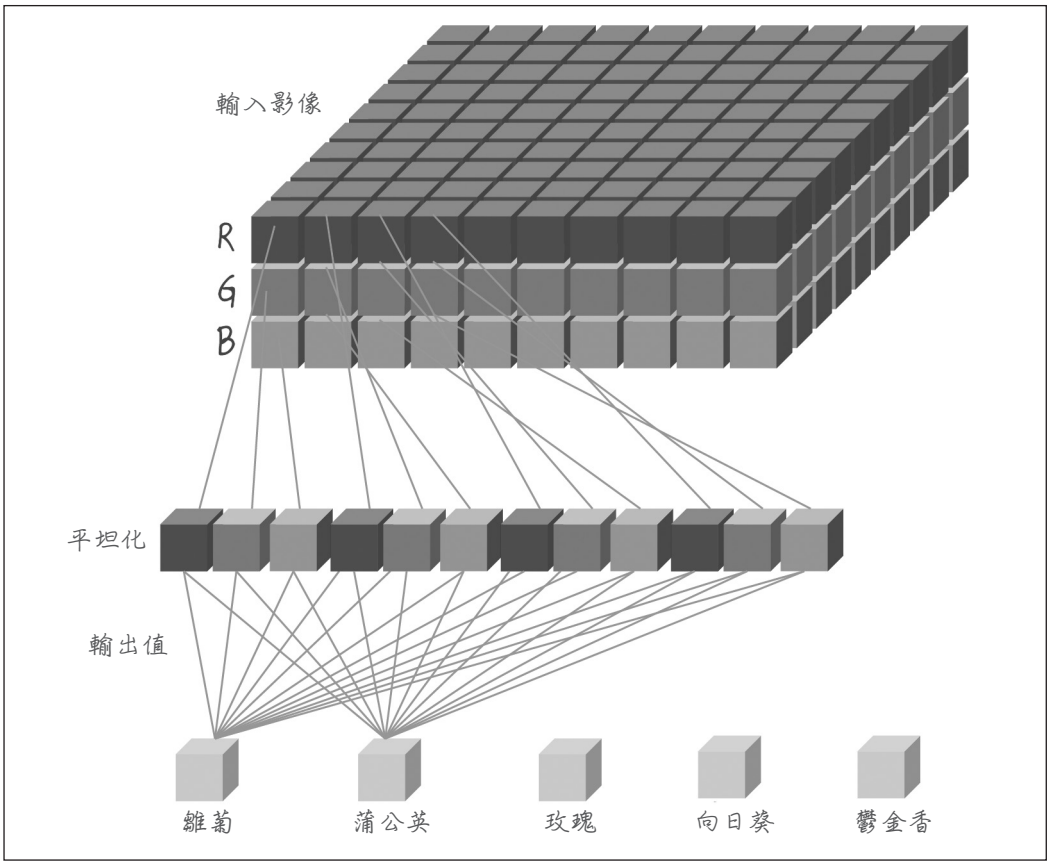


圖 2-5　在線性模型中，有五個輸出，每個類別一個；每個輸出值都是輸入像素值的加權總和。

在實務上，還會添加了一個稱為*偏差*（*bias*）的常數項，因此我們可以將每個輸出值表達為：

$$Y_j = b_j + \sum_{\text{列}} \sum_{\text{行}} \sum_{\text{頻道}} (w_i * x_i)$$

沒有偏差的話，如果所有像素都是黑色，我們將強制輸出為零。

Keras 模型

與使用低階 TensorFlow 函數來編寫上述方程式相比，使用更高階的抽象化（abstraction）會更方便。TensorFlow 1.1 附帶了一個這樣的抽象化，也就是 Estimator API，而且目前仍然支援 Estimator 以達成向下相容性。但是，自從 TensorFlow 2.0 以來，Keras API 一直是 TensorFlow 的一部分，我們建議您使用它。

線性模型可以在 Keras 中表達如下：

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
    tf.keras.layers.Dense(len(CLASS_NAMES))
])
```

循序模型（*sequential model*）由連接的**層**（*layer*）所組成，使得某一層的輸出是下一層的輸入。層是一個 Keras 的組件，它接受張量作為輸入，再對該輸入應用一些 TensorFlow 運算，然後輸出一個張量。

第一層是內隱式的（implicit）輸入層，它要求一個 3D 影像張量。第二層（`Flatten` 層）接受 3D 影像張量作為輸入，並將其重塑（reshape）為具有相同數量的值的 1D 張量。此 `Flatten` 層連接到一個 `Dense` 層，每一類的花朵在此都有一個輸出節點。`Dense` 這個名稱意味著每個輸出都是每個輸入的加權總和，並且沒有共享權重。我們將在本章後面遇到其他常見類型的層。

要使用此處所定義的 Keras 模型，我們需要對訓練資料集呼叫 `model.fit()`，並對要分類的每張影像呼叫 `model.predict()`。為了訓練模型，我們需要告訴 Keras 如何根據訓練資料集來優化權重。要這樣做的方法是**編譯**（*compile*）模型、指定要使用的**優化器**（*optimizer*）、要最小化的**損失**（*loss*）、以及要報告的**量度**（*metric*）。例如：

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

Keras 的 `predict()` 函數將對影像進行模型計算。如果我們先看一下預測程式碼，`compile()` 函數的參數會更有意義，所以讓我們從那裡開始。