INTRODUCTION

本書的目標族群與編排

本書的目標族群與編排

本書是解說以 TensorFlow 打造深度學習模型的入門書籍，一開始先介紹 TensorFlow 的使用，之後再逐步介紹利用高階 API 的 Keras 建構實用的深度學習模型。

目標讀者

以想接觸深度學習的工程師為對象。由於盡可能不使用公式解說，所以只需具備高中數學的程度就能閱讀本書。

編排

基本篇從建構環境開始，並且介紹深度學習、TensorFlow、Keras 的基礎。應用篇則挑戰以 Keras 建構實用的影像處理深度學習模型。

本書將一邊帶著大家了解 TensorFlow 與 Keras 的功能，一邊帶著大家學習可於職場應用的深度學習模型。分

第 1 部分	**基本篇**	
	第 1 章	機械學習函式庫 TensorFlow 與 Keras
	第 2 章	建構開發環境
	第 3 章	透過簡單的範例學習 TensorFlow
	第 4 章	神經網路與 Keras
	第 5 章	利用 Keras 建構 CNN
	第 6 章	應用預訓練模型
	第 7 章	常用的 Keras 功能
第 2 部分	**應用篇**	
	第 8 章	使用 CAE 消除雜訊
	第 9 章	自動上色
	第10章	超高解析度成像
	第11章	轉換畫風
	第12章	影像生成

1.2 深度學習的應用範圍

深度學習雖是目前的當紅炸子雞，但是到底要如何於現實面應用呢？在介紹 TensorFlow 之前，先一起來看看深度學習已於哪些領域應用吧！

1.2.1 影像處理

影像分類

於 2012 年在 ILSVRC 嶄露頭角的是深度學習的影像分類（圖 1.2）。所謂影像分類就是推測影像裡的東西為何的任務，也就是判斷「照片裡的數字是 0 ～ 9 的哪一個數字」、「這張照片是男性還是女性」這類任務。

ILSVRC 以「flamingo（佛朗明哥）」或「gondola（纜車）」這 1000 個等級進行分類，藉此在精確度一較高下。

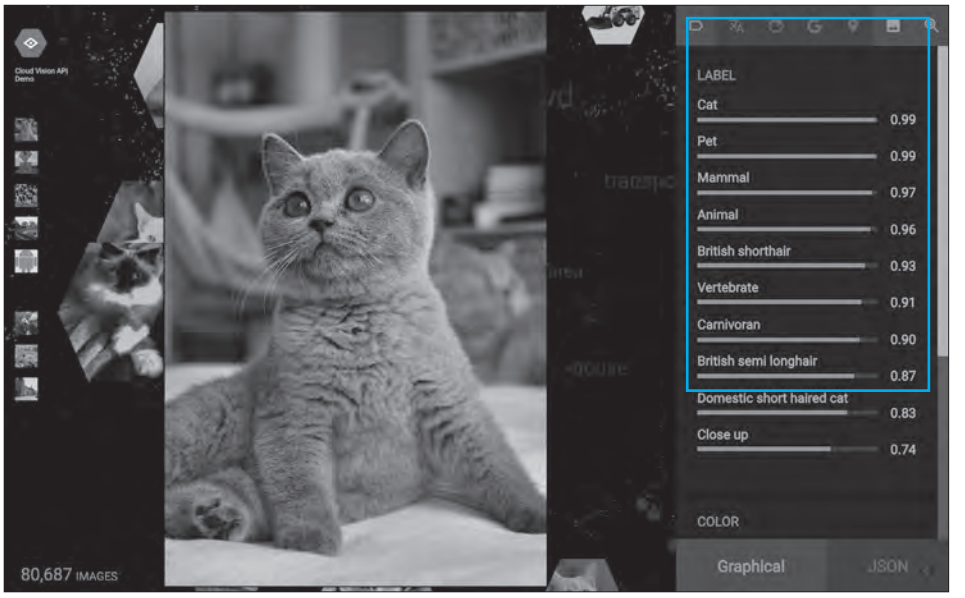

圖 1.2 影像分類的範例。貓咪照片貼了「Cat」或「Pet」這類標籤

出處 Google Cloud Platform Japan Blog
URL https://cloudplatform-jp.googleblog.com/2016/05/cloud-vision-api.html

Project Magenta 嘗試將深度學習應用於藝術創作，也應用該研究成果完成誰都能透過深度學習產生音樂的研究。

舉例來說，使用 Project Magenta 的研究成果之一的 Performance RNN（圖 1.18）就能根據單純的 MIDI 訊號仿照職業音樂家演奏音樂，而且還能呈現抖音或強弱音這類技巧。

Google DeepMind 開發的 WaveNet 也於 2016 年備受注目（圖 1.19）。之前的音樂生成技術都是輸出 MIDI 的樂譜，但 WaveNet 則是直接輸出波型，所以能產生非常高品質的音樂。

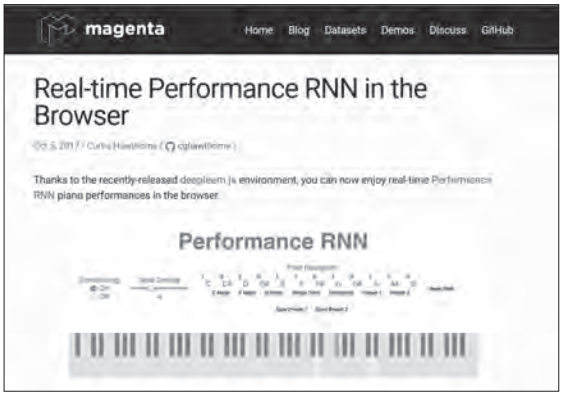

圖 1.18 利用網頁瀏覽器執行的 Performance RNN 的展示程式

出處 「Real-time Performance RNN in the Browser」

URL https://magenta.TensorFlow.org/performance-rnn-browser

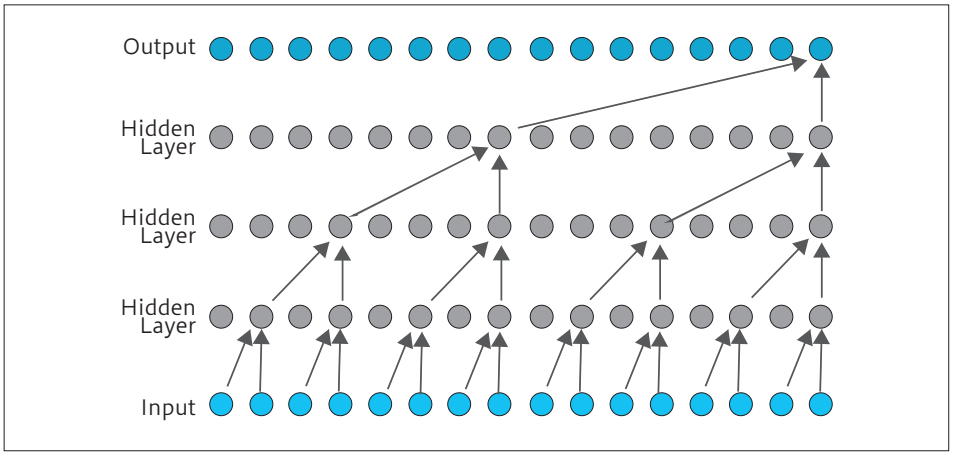

圖 1.19 WaveNet 產生音樂的示意圖

出處 根據「WaveNet: A Generative Model for Raw Audio」的影像作圖

URL https://deepmind.com/blog/wavenet-generative-model-raw-audio/

1.3.3 分散式處理

第三項特徵是分散式處理。從 TensorFlow 的白皮書大力介紹分散式處理的機制就可以知道，TensorFlow 最強的強項就是分散式處理。

深度學習的計算量非常龐大，所以常會遇到需要使用分散式處理的情況。通常要使用分散式學習需要非常進階的技術，但是若使用 TensorFlow，就能稍微調降撰寫分散式處理的難度。

1.3.4 TensorBoard 的視覺化呈現

TensorBoard 的視覺化呈現方式是 TensorFlow 的重要特徵之一。

深度學習常被形容為「黑盒子」，非常多的問題，「但很難了解知道黑盒子裡頭到底發生了什麼事」也是問題。

TensorFlow 隨附的「TensorBoard」（圖 1.23）是一種「幫助了解現在發生了什麼事」的工具，可具體呈現學習時的損失函數的執行過程與中間層的狀況，也可呈現特徵值的嵌入情況，幫助我們替程式除錯以及了解建立的模型。

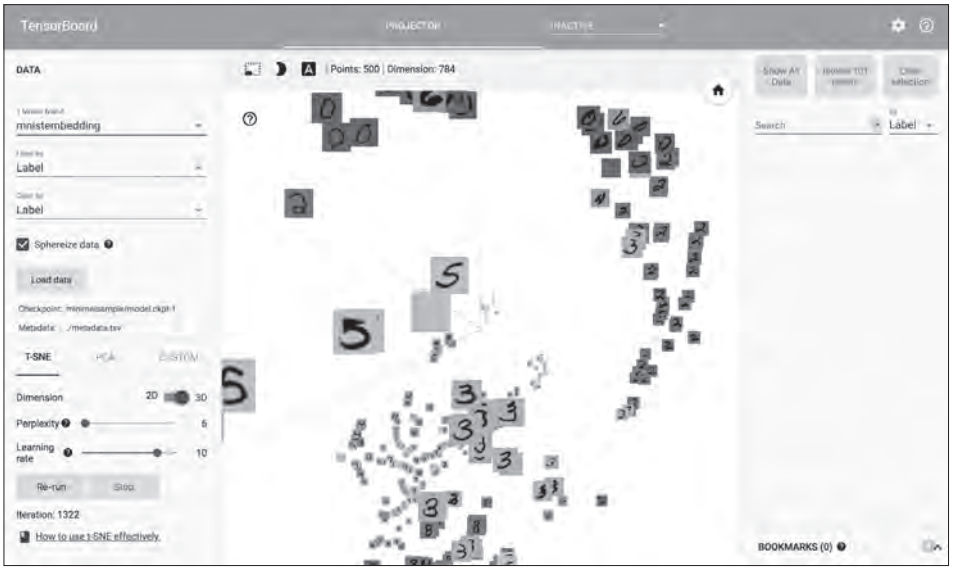

圖 1.23 TensorBoard 呈現的 Embedding Visualization

1.3.5 各種層級的 API 與生態圈

從可控制細節的低階 API 到高階 API 全面涵蓋這點，也是 TensorFlow 的特徵之一。

TensorFlow 在 2015 年發表時，版本只到 0.6，只能使用非常低階的 TensorFlow Core API，但在本書執筆時（2018 年 3 月），Core API 已提供 Layers、Keras、Estimator 這類高階 API。

Estimator 內建的 pre-made Estimator 具有基本的網路構造，只要設定參數就能開始學習，所以不需要太了解演算法就能使用，也讓 TensorFlow 的使用門檻大幅下降。若是 pre-made Estimator 不支援的網路，則可利用 Layer 或 Keras 像組樂高積木般建立模型。本書也將利用 Keras 建立各種模型。

此外，2017 年 3 月，Google Cloud Machine Learning Engine（ML Engine）也公開了（圖 1.24）。

ML Engine 可在未建立基礎環境下，以 CPU 或 GPU 執行分散式學習，也能直接將建立的模型當成 API 使用。

圖 1.24 ML Engine 的畫面

TensorFlow 的社群非常龐大這點也是特徵之一。GitHub 的星星數已超過 90,000 個，遠遠勝過其他的深度學習函式庫。最新的研究論文也都常會以 TensorFlow 撰寫，有助於程式的閱讀、理解與試用。

2.2 Python 的環境建構

要在 TensorFlow 或 Keras 使用 Python 建構模型，就必須建構 Python 的開發環境。本節將介紹 Python 開發環境之一的 Anaconda。

2.2.1 何謂 Anaconda

TensorFlow 為各種語言內建了建構圖表、執行圖表的 API，其中當然也支援 C++、Java 或 Go 這類語言，但是以 Python 的 API 完成度最高，所以通常會從 Python 使用，這也是為什麼在安裝 TensorFlow 之前，要先建立 Python 的環境。這次使用的是與 Windows 相容度極高的 Anaconda 這個 Python 發行版。

Anaconda 使用的「虛擬環境」可自由切換 Python 的版本或函式庫（圖 2.1）。Anaconda 可指定每個虛擬環境的 Python 版本，卻不一定要與下載 Anaconda 時指定的版本一致。在 Windows 環境下，TensorFlow 支援 Python 3.5 與 3.6，所以只要指定其中一個版本與建立虛擬環境，接著再於虛擬環境安裝 TensorFlow 即可。

平常習慣在 macOS 或 Linux 利用 Python 開發的讀者或許可使用 pyenv 這類版本管理工具建立 TensorFlow 的環境，但請恕本書不對 pyenv 多作說明。

圖 2.1 Anaconda 與虛擬環境的關係

4.2 利用 Keras 建立前饋神經網路

前一節介紹了作為神經網路雛型的感知器以及 S 神經元，這一節則要介紹以 Keras 建立前饋神經網路的方法。

4.2.1 前饋神經網路

前一節介紹了神經網路的基本架構，也就是感知器與 S 神經元，而這一節則要介紹包含這兩個部分的前饋神經網路。

前饋神經網路就是讓神經元層層排列，然後只有相鄰的層彼此合併的神經網路。輸入的資料只會順向傳遞，不會逆向回到前一層（圖 4.8）。第一層稱為輸入層，最後一層稱為輸出層，在這兩層之間的層稱為隱藏層。

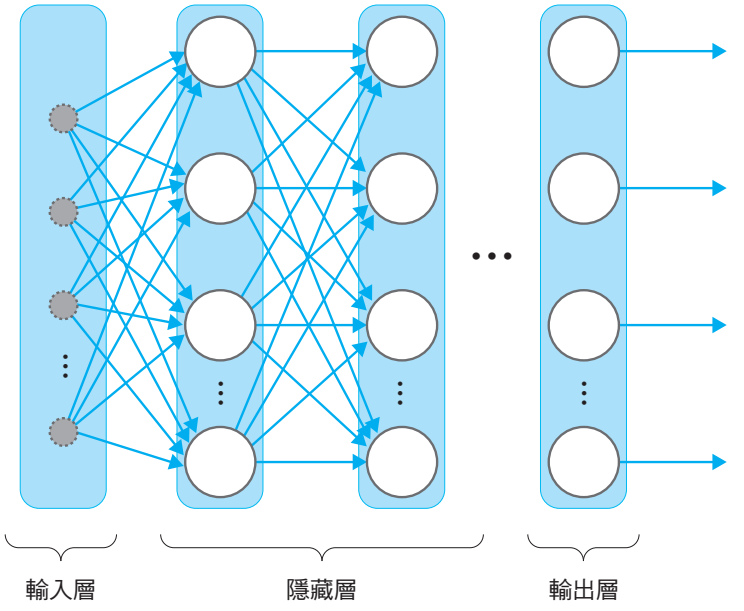

圖 4.8 前饋神經網路

8.1 CAE 的實用性

接下來介紹能於各種課題應用，又屬於基本架構的 CAE。

8.1.1 CAE 的應用範例與實用性

CAE（Convolutional Autoencoder）是利用卷積神經網路建置的基本架構。

CAE 就是利用 CNN 壓縮輸入的影像（編碼），再根據壓縮的資料重新建構（解碼）輸入影像的模型（圖 8.1）。

由於是編碼與解碼的輸出架構，所以也屬於 **Encoder-Decoder（編碼器、解碼器）**的一種。

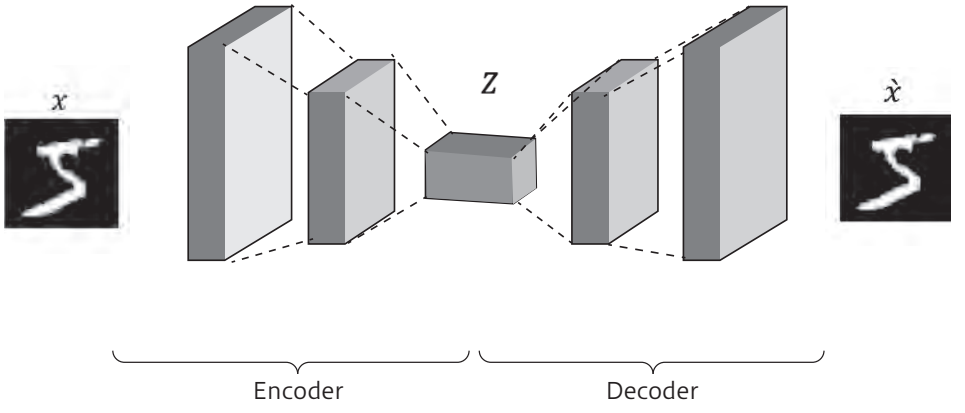

圖 8.1 CAE 的概要圖

CAE 的實際應用包含根據行車記錄器的影像辨識「人體」、「道路」、「路標」這類範圍的「語意分割」（Semantic Segmentaiton）（圖 8.2）以及根據沒有標籤的資料集鎖定異常影像的「異常偵測」。

這一章雖然只介紹最基本的 CAE，但只要在**第 10 章**介紹的超高解析度成像與**第 12 章**介紹的影像生成花點心思，就能產生高解析度的影像。

MNIST 的內容為 0 至 255 整數的矩陣，但本章則是先標準化這個矩陣，讓數值收斂在 0 ～ 1 的範圍。此外，這次使用的是 binary crossentropy 損失函數，所以將 `activation` 函數指定為 sigmoid，可讓模型的輸出值介於 0 ～ 1 的範圍。

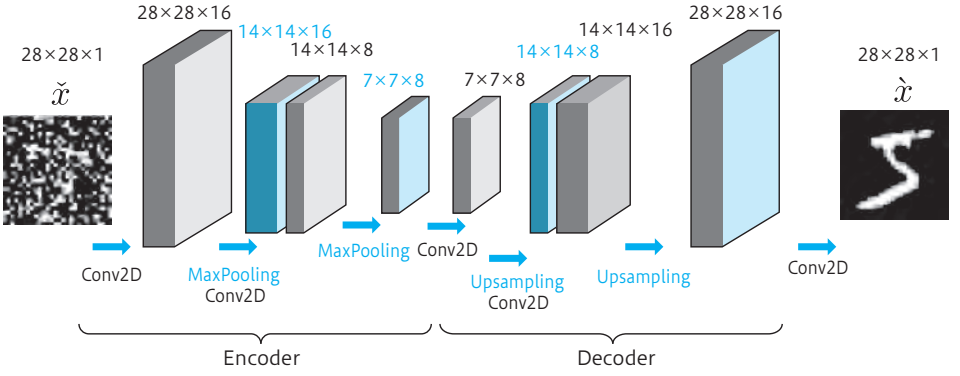

圖 8.7　消除雜訊的 CAE 神經網路圖

8.3.5　確認模型的概要

利用 `summary` 方法確認建置的模型，可發現輸入值與輸出值的大小一樣，也可看出 `max_pooling2d_1`、`max_pooling2d_2` 的部分的確將影像縮減為一半（列表 8.6 ❶），`conv2d_2` 的確縮減了色版，影像的確被壓縮了（列表 8.6 ❷）。從中也可以發現 Encoder 是以 2 層卷積層與池化層組成（列表 8.6 ❸）。

接著還可以發現這次在 `conv2d_4` 的部分將色版數放大為 2 倍（列表 8.6 ❹），也於 `up_sampling2d_1`、`up_sampling2d_2`（列表 8.6 ❺）將隱藏層的輸出結果放大為 2 倍，讓輸出結果與輸入影像的大小相同之後，再於 `conv2d_5` 統整為 1 個色版。到此為止，都是 Decoder 的內容（列表 8.6 ❻）。

列表 8.6　確認模型的概要

In

```
autoencoder.summary()
```

Out

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 16)        160
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 8)         1160
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 8)           0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 8)           584
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 14, 14, 8)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 16)        1168
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 28, 28, 16)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 1)         145
=================================================================
Total params: 3,217
Trainable params: 3,217
Non-trainable params: 0
_________________________________________________________________
```


8.3.6 利用高斯雜訊資料進行學習與預測

接著要將高斯雜訊資料當成輸入影像，並將原始影像當成正確解答影像，讓模型進行學習（列表 8.7）。

列表 8.7 利用高斯雜訊資料學習

In

```
autoencoder.fit(
                x_train_gauss, # 輸入值：高斯雜訊資料
                x_train,       # 正確解答：原始影像
                epochs=10,     # 學習的時期數
```

9.2 執行自動上色處理的準備

要執行自動上色，必須執行獨立的前置處理與事後處理，所以在此要先介紹執行自動上色處理的各種準備。

9.2.1 神經網路全貌圖

這次要建置的神經網路的全貌圖與處理流程請參考圖 9.2 。

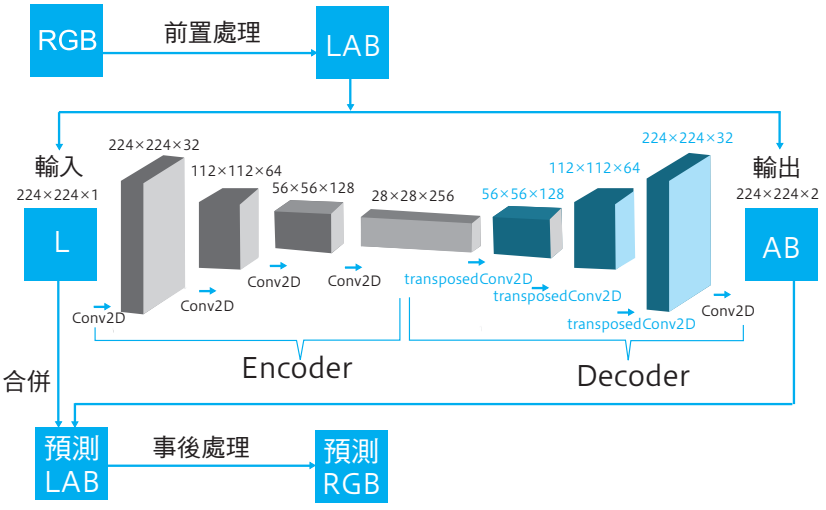

圖 9.2 神經網路全貌圖

到目前為止學過的模型都是處理「RGB」模式的影像，但本章要建置的模型則會在前置處理的部分，將影像從「RGB」模式轉換成「LAB」模式，建立輸入「L」的值之後，輸出「AB」的值的模型。為什麼要先轉換影像的色彩模式呢？其中的細節會於下一節補述。

本章建置的模型與前一章建置的模型一樣，構造都非常簡單，卻有更多細部的調整，所以先在此統一介紹。第一個細部調整的是追加間隔（stride）為 2 的卷積處理，取代最大值池化層。前一章的 Encoder 部分是將最大值池化層放在卷積層之後，但這次間隔為 2 的卷積層則於同一層執行影像縮減處理與卷積處理。

11.1.3 神經網路建置概要

接著讓我們先一窺神經網路的概貌。這次要建置的神經網路主要分成產生影像的神經網路（轉換神經網路）與計算損失的神經網路（損失神經網路）這兩個部分（圖 11.2）。

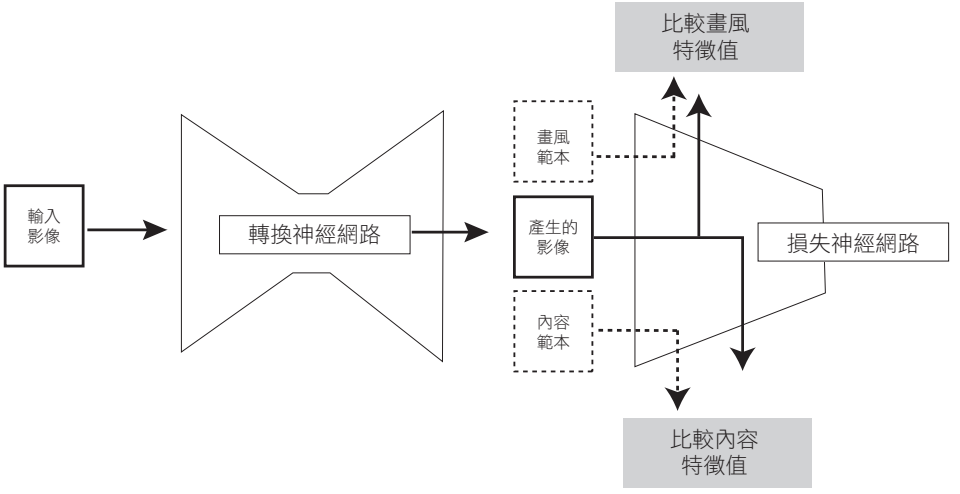

圖 11.2 轉換畫風的神經網路概念圖

圖 11.2 左側的轉換神經網路與前一章之前的 CAE 相同，輸入影像之後，就會產生畫風轉換後的影像。

圖 11.2 右側的損失神經網路則使用預訓練模型 VGG16，之所以採用 VGG16 是因為，目前已知道稍微改變 VGG16 的構造，就能輸出內容與畫風的特徵值，只要將位於左側的轉換神經網路所產生的影像輸入損失神經網路，就能輸出內容與畫風的特徵值。之後只要比較這個特徵值與範本影像的特徵值，再將兩邊的落差定義為損失即可。

為了讓損失縮小，所以只學習轉換神經網路的權重。轉換畫風所需的只有轉換神經網路，等到學習結束後，就不會再用到損失神經網路。