南華大學科技學院永續綠色科技碩士學位學程

碩士論文

Master Program of Green Technology for Sustainability
College of Science and Technology
Nanhua University

Master Thesis

用於犬名辨識的卷積神經網絡之多次遷移學習

Multiple Transfer Learnings on Convolutional Neural Networks for Dog Identification

王俊文

Jun-Wen Wong

指導教授：賴信志 博士

Advisor: Shin-Chi Lai, Ph.D.

中華民國 110 年 6 月

June 2021

南 華 大 學

永續綠色科技碩士學位學程

碩 士 學 位 論 文

用於犬名辨識的卷積神經網絡之多次遷移學習

Multiple Transfer Learnings on Convolutional Neural Networks for Dog Identification

研究生 ：王俊文

經考試合格特此證明

口試委員：蔡文凱

莱文河

黃冠雄

指導教授：賴信志

系主任(所長)：永續綠色科技碩士學位學程主任林文賜

口試日期：中華民國 110 年 6 月 28 日

摘要

本研究以『機器學習』的分支，『深度學習』這一技術實現以『狗』為範圍的動物生物辨識為目標。透過利用『深度學習』之神經網絡模型應用於辨別確認一張圖像裡，一隻狗的身份名字。本論文著重探討有關通過學習辨別對象，一隻狗的『軟生物特徵』，例如該對象的品種，以及該生物面部特徵的數據，以提高識別該對象的身份名字這一能力的可能性。透過各個數據分別以『對象品種的完整視覺外觀』、『對象品種的面部外觀』、以及『對象身份名字的面部外觀』分門別類收集成三個數據集，再以遷移學習的概念應用學習到各種『卷積神經網絡』裡，以學習對象的品種，以及往後學習該對象的身份名字。最後產生的神經網絡通過 Android 應用程序的前端應用程序界面部署在 Flask 伺服器上。

關鍵詞：深度學習、卷積神經網絡、遷移學習、辨識狗的身份名字、Flask 伺服器、Android 應用程序

ABSTRACT

This study aims to achieve animal biometric identification, specifically dogs through the use of deep learning, a branch of machine learning. Through the application of deep learning neural network models, identify a dog's name from a photograph or an image. This paper focuses on exploring the possibilities of through learning the study subject's (dog) "soft biometrics", such as subject's breed, as well as subject's facial biometrics, to improve subject's identity recognition. This is achieved by preparing datasets in stages, by altering data into "subjects' breed full visual appearance", "subjects' breed facial appearance", and "subjects' identity facial appearance". The datasets are then applied through the concept of transfer learning on different Convolutional Neural Networks to learn the subject's breed and onto the subject's identity. The final neural network is deployed on a flask server, with the front-end application interface of an android app.

Keywords: deep learning, convolutional neural network, transfer learning, dog identification, flask server, android app

目錄

圖目錄

表目錄

第一章 簡介

1.1 前言

動物個體辨識是指透過特定的標記方式以實現動物個體的辨識和追踪。不同的動物識別方法，例如烙印 [1]，已經使用了數千年。同樣的技術如今也被套用在寵物上，當中最常見的就是透過 RFID 等的微晶片植入 [2] 偵測一隻動物身上的標籤並進行辨識。可這種做法與先前所提的烙印一樣，是作為證明所有權的一種方式，而不是作為動物識別的一種方法。而且微晶片植入最大的問題是其中後續的影響，比如肉體精神上的影響，以及病毒感染的可能性。在這情況下，可以透過 AI 演算法實現動物的個體辨識，俗稱臉部辨識這一技術，則相對的能完全避免先前所述的問題。

1.2 研究目的

動物電子健康的概念是推動創建寵物電子病歷的主要動力之一。對於每條記錄，都會拍攝寵物的照片，並可用於使用圖像處理技術驗證/識別每隻寵物的身份。例如，在偏遠地區發現一隻狗的人可以使用多媒體設備拍攝它的照片，並將其發送到區域獸醫數據庫進行識別。

狗頭或臉的照片可用於辨識一隻狗，這類似於使用人臉識別人。這是一種細粒度分類，是指對具有相似視覺特徵並屬於同一基本級別類的對象進行分類[3]。寵物的主人或獸醫也可以為狗拍攝各種照片並存儲這些圖像數據以供進一步使用，例如協助尋找丟失的狗。Google Play 商店中也有很多應用程序可以找到丟失的狗，例如“PetsFinder”。只需要一份報告，包括丟失或找到寵物的位置和照片；上傳到雲數據庫，寵物的主人便可以在附近的所有寵物列表中搜索丟失的寵物。

多媒體圖像處理和識別不僅可以用於辨識一隻狗，還可以識別品種、身高和其他軟生物特徵。在這項研究中應用了這種方法，特別是對品種進行分類如何幫助提高外觀（面部）識別狗的能力。透過應用現代機器學習，如深度神經網絡，以及轉移學習的技術[4]，該技術使用為一項任務（品種識別）開發的方法來處理另一項任務（狗臉識別）。

第二章 背景知識與相關研究

人工智慧/人工智能（Artificial Intelligence, AI [5]），是指由人製造出來的機器所表現出來的智慧。通常是指電腦模擬/模擬人類思維過程以模仿人類能力或行為的能力。

由上述定義可知，人工智慧這個題目應該早在有計算機出來的時候，應該就有這個名詞的出現了，所以人工智慧這個名詞出現的非常早(早於 1956 年[5])。

但因為早期計算機/電腦的效能和限制，因此只能用來解決一些簡單的問題，無法實際用在解決現實生活的問題，所以理論雖然一直有在發展，但這個主題一直被限制住，沒有蓬勃發展起來。

2.1 機器學習與深度學習

AI 是一個很大的集合，機器學習只是其中的集合，而深度學習也只是機器學習的其中一個子集合，如圖 1 所示。

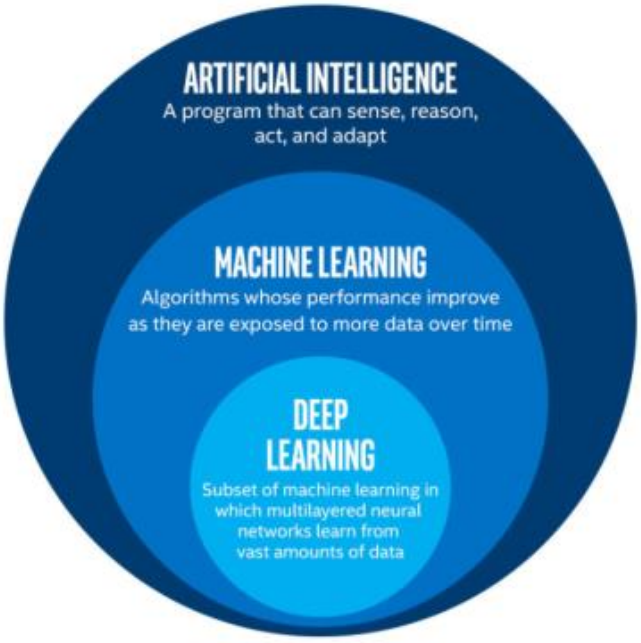

圖 1 人工智慧、機器學習以及深度學習區塊圖[6]

機器學習（Machine Learning， ML）是人工智慧的一種應用，它為 AI 系統提供了從環境中自動學習的能力，並應用這種學習來做出更好的決策。機器學習使用多種算法進行迭代學習、描述和改進數據，以預測更精確的結果。這類演算法使用統計技術來發掘資料的特徵模式，並加以對這些模式執行操作。

深度學習（Deep Learning，DL）是機器學習的一類演算法。深度學習模型可以完全獨立於人類做出自己的預測。其設計靈感來自於人腦的生物神經網絡，透過如神經網絡的多層架構從原始輸入中逐步提取更高級別的特徵。而過去的機器學習模型在許多情況下仍然需要人工干預才能達到最佳結果。以在圖像處理中為例，較低層可以識別邊緣，而較高層可以識別與人類相關的概念，如數字、字母、臉部等。

2.2 卷積神經網絡

卷積神經網絡（Convolutional Neural Network，ConvNet/CNN）是一種深度學習演算法。CNN 接收輸入圖像，為圖像中的各個要素/對象分配其重要性（可學習的權重，weights 和偏差，bias），並能夠區分彼此。與其他分類算法相比，ConvNet 中所需的預處理要低得多。在原始方法中，過濾器是手工設計的，但經過足夠的訓練，CNN 能學習這些過濾器/特徵。

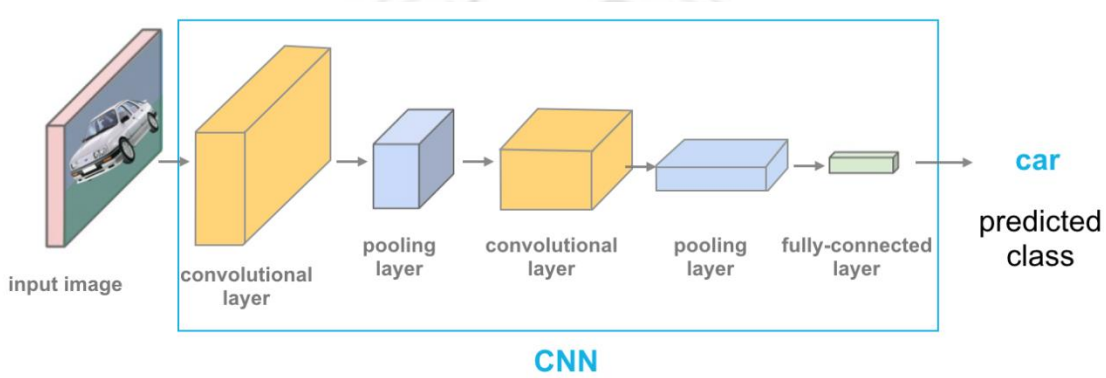

圖 2 CNN 基本架構示意圖[7]

如圖 2 所示，除了第一個卷積層（負責接收輸入圖像）之外，每一層卷積層都接收前一個卷積層的輸出並生成一個輸出特徵圖，並將其傳遞給下一個卷積層。一個架構裡所擁有的 L 層，就會有 L 個直接連接，每層與其後續層之間有一個連接。

2.2.1 DenseNet

DenseNet 架構是把先前標準的 CNN 架構修改成如圖 3 所示。

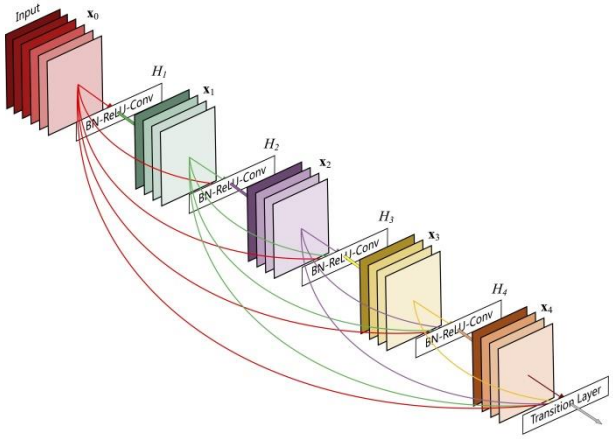

圖 3 DenseNet 之 Dense Block 架構示意圖[8]

在 DenseNet 架構裡，每一層都與其他每一層相連，因此得名 Densely Connected Convolutional Network。對於架構裡所含有的 L 層，就會有 L(L+1)/2 個直接連接。對於每一層，將前面所有層的特徵圖用作輸入，並將其自己的特徵圖用作後續各層的輸入。

DenseNet 是專門為改善高級神經網絡中梯度消失導致的精度下降而開發的。梯度消失是指由於輸入層和輸出層之間的路徑較長，特徵資訊在到達目的地之前就消失了。DenseNet 除了減輕梯度消失的問題，也加強了特徵傳播，鼓勵特徵重用，並大大減少了參數的數量。

而本實驗採用的版本為 DenseNet201 架構圖如表 1 所示，其中還包含了 DenseNet-121，DenseNet-169 以及 DenseNet-264 的版本。

各個版本的差別為 Dense Block 3 和 Dense Block 4 的層數。而本實驗通過 Tensorflow 僅能實現最複雜的版本為 DenseNet 201。

表 1 DenseNet201 用於 ImageNet 辨識之架構

Layers	Output Size	DenseNet-201
Convolution	112 × 112	7 × 7 conv, stride 2
Pooling	56 × 56	3 × 3 max pool, stride 2
Dense Block (1)	56 × 56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv
	28 × 28	2 × 2 average pool, stride 2
Dense Block (2)	28 × 28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv
	14 ×14	2 × 2 average pool, stride 2
Dense Block (3)	14 × 14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Transition Layer (3)	14 × 14	1 × 1 conv
	7 × 7	2 × 2 average pool, stride 2
Dense Block (4)	7 × 7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$
Classification Layer	1 × 1	7 × 7 global average pool
		1000D fully-connected, softmax

資料來源：Densely Connected Convolutional Networks[8]

2.2.2 EfficientNet

EfficientNet 是一種卷積神經網絡架構和縮放方法。它使用複合係數統一縮放深度/寬度/分辨率的所有維度。 與任意縮放這些因素的傳統做法不同，EfficientNet 縮放方法使用一組固定縮放係數統一縮放網絡寬度、深度和分辨率。

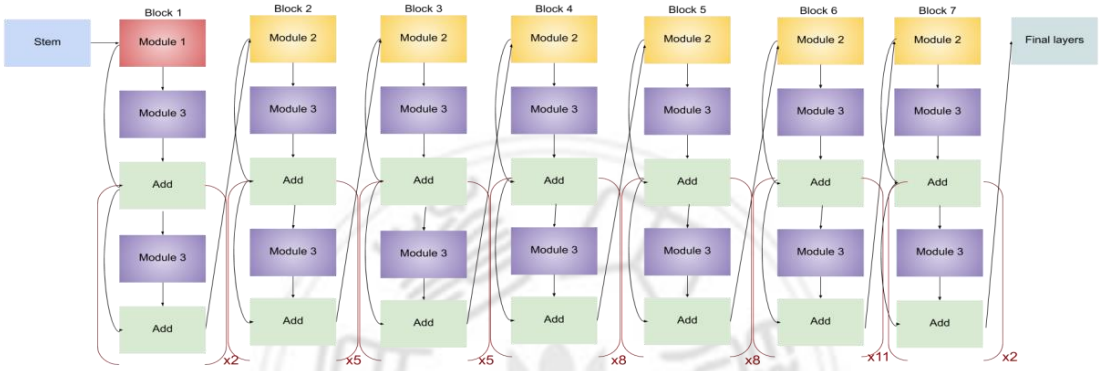

圖 4 EfficientNet B7 架構（相乘數表示括號內的模組重複的次數）[10]

圖 4 為 EfficientNet B7 版本的基本架構，而其中所應用的各模組內容如圖 5，6，7 所示。EfficientNet 系列的架構是透過把圖 5 和圖 7 的模組拼湊而成。其中各個版本架構的差異如表 2 所示。

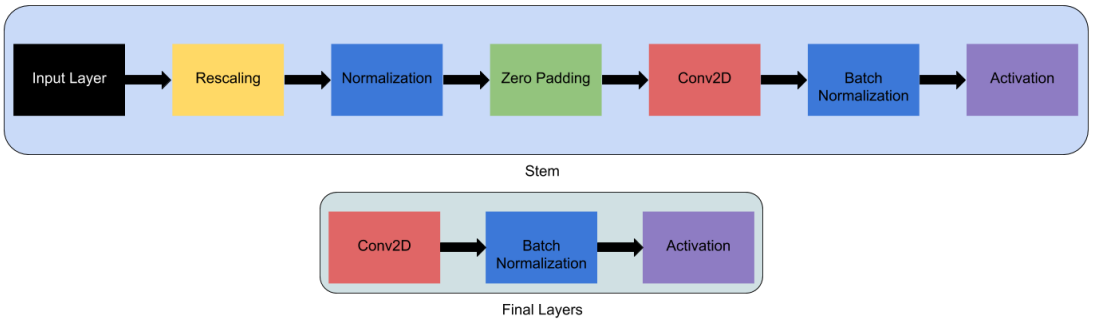

圖 5 EfficientNet 系列（B0-B7）共同之主幹和總結層[10]

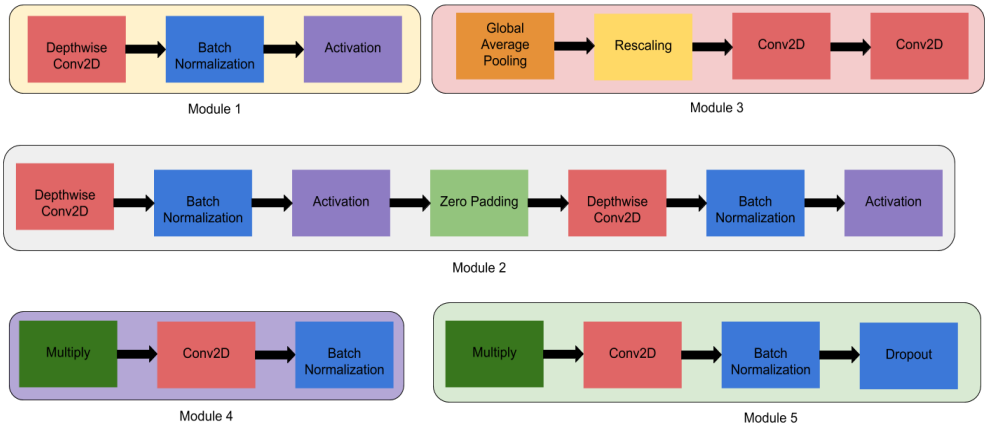

圖 6 EfficientNet 系列（B0-B7）通用的 5 個模組 [10]

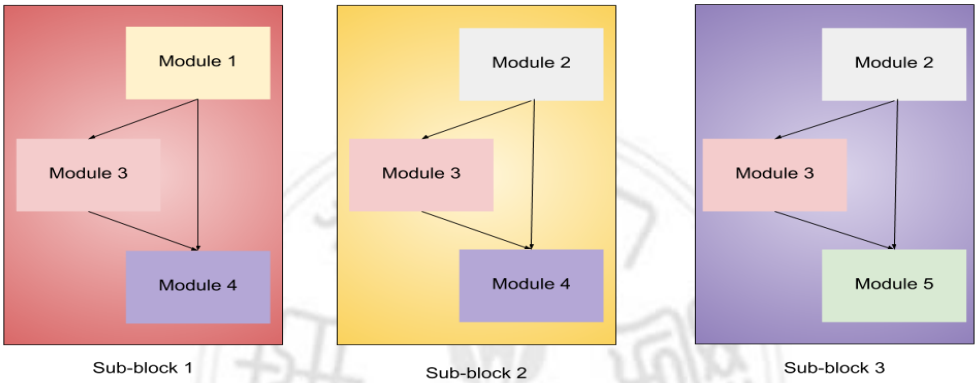

圖 7 EfficientNet 系列（B0-B7）由圖 6 的 5 個模組組合而成的子塊[10]

表 2 ImageNet 上的 EfficientNet 性能結果比對

Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPS	Ratio-to-EfficientNet
EfficientNet-B0	**76.3%**	**93.2%**	**5.3M**	**1x**	**0.39B**	**1x**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	**78.8%**	**94.4%**	**7.8M**	**1x**	**0.70B**	**1x**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	**79.8%**	**94.9%**	**9.2M**	**1x**	**1.0B**	**1x**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	**81.1%**	**95.5%**	**12M**	**1x**	**1.8B**	**1x**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**82.6%**	**96.3%**	**19M**	**1x**	**4.2B**	**1x**
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.3%**	**96.7%**	**30M**	**1x**	**9.9B**	**1x**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.0%**	**96.9%**	**43M**	**1x**	**19B**	**1x**
EfficientNet-B7	**84.4%**	**97.1%**	**66M**	**1x**	**37B**	**1x**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

資料來源：EfficientNet: Rethinking Model Scaling for CNNs[9]

2.2.3 ResNet V2

ResNet，也稱為殘差神經網絡（Residual Network） 是建立在大腦皮層錐體細胞已知的結構上的一種人工神經網絡。殘差神經網絡通過利用跳過連接或跳過某些層的快捷方式來實現這一點。

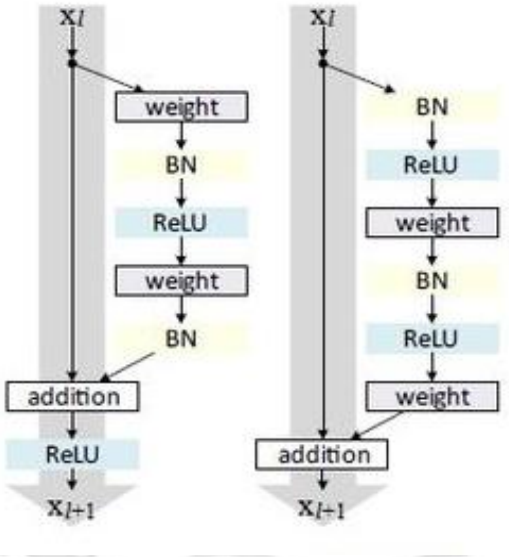

圖 8 ResNet 和 ResNetV2 的殘差單位（Residual Unit）[11]

圖 8 為 ResNet 核心的殘差單位基本架構的演變。ResNet V2 重設計了一種殘差網絡基本單元（單元），如圖 8 右半所示，將激活功能（先 BN 再 ReLU）移到值層之前，形成一種“預激活方式（pre-activation）”的結果，而不是“後激活”方式，並且預激活的單元中的還原值訓練層的都是一的信號，如圖 8 左半所示。這更新同時也增加了 ResNet 的泛化性。

2.2.4 Inception V3

Inception v3 是來自 Inception 系列的捲積神經網絡架構。Inception v3 主要側重於通過修改之前的 Inception 架構來消耗更少的計算能力。Inception v3 網絡的架構是逐步構建的，並把所述的概念都合併到最終架構中。

分解卷積的目的是在不降低網絡效率的情況下減少連接/參數的數量。如圖 9 所示，通過使用 1 層 5 × 5 濾波器，參數數量為 5 × 5 = 25。 通過使用 2 層 3 × 3 濾波器，參數數量為 3 × 3 + 3 × 3 = 18。進而達到減少 28% 的參數數量。

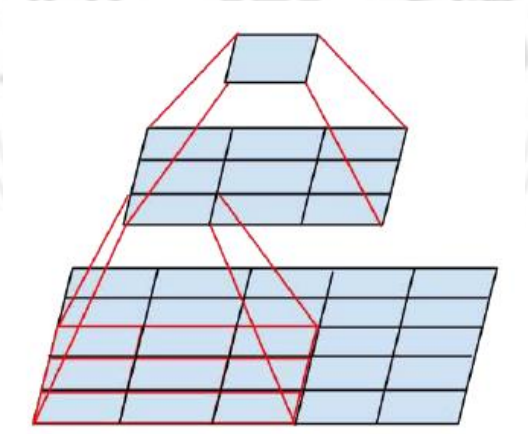

圖 9 取代 5 × 5 卷積的迷你網絡。[12]

輔助分類器是在訓練過程中插入層與層之間的一個小的 CNN，產生的損失被添加到主網絡的損失中。在 GoogLeNet 中，輔助分類器用於更深的網絡，而在 Inception v3 中，輔助分類器充當正則化器。

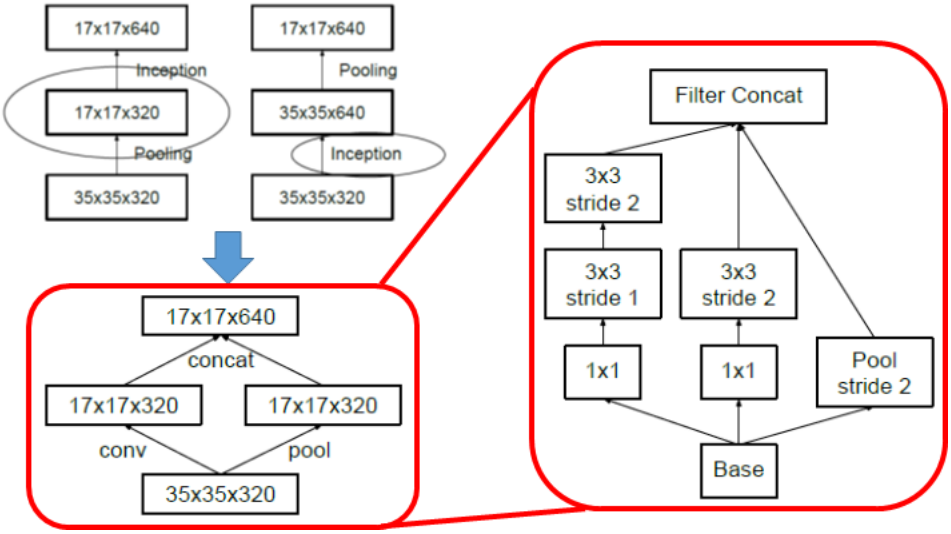

圖 10（左上）常規縮小規模（左下）高效網格縮減（右）高效網格縮減的詳細架構[12]

如圖 10 所示，通過有效的網格尺寸縮減，320 個特徵圖由 conv 完成，步幅為 2。320 個特徵圖通過最大池化獲得。 並且將這 2 組特徵圖拼接成 640 個特徵圖，再進入下一層的 Inception 模塊。 通過這種有效的網格尺寸減小，實現了更便宜但仍然有效的網絡。

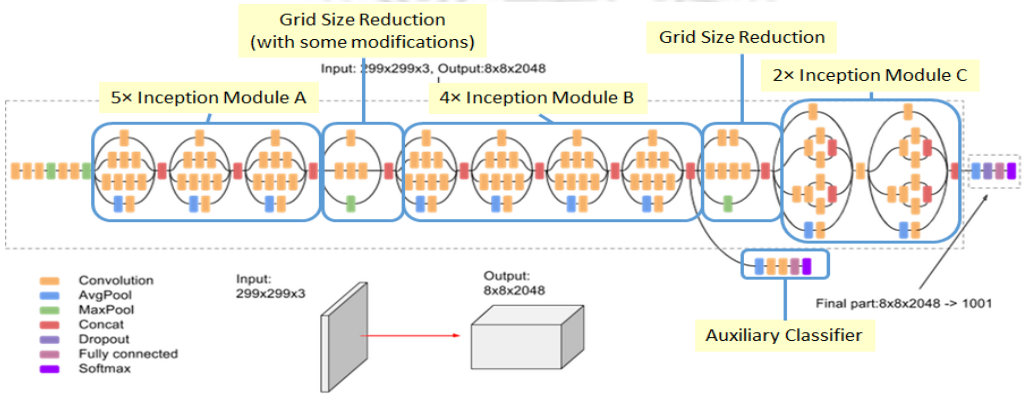

圖 11 Inception-v3 架構圖（Conv 之後使用 Batch Norm 和 ReLU）[13]

2.2.5 InceptionResNet V2

Inception ResNet 是受到 ResNet 性能的啟發而提出的混合型 CNN。Inception ResNet 有兩個版本，v1 和 v2。它是基於 Inception 和 Residual 兩種架構混合的產物。在 Inception-Resnet 裡，多個大小的捲積濾波器與殘差連接相結合。殘差單位的使用不僅避免了由深層結構引起的退化問題，而且減少了訓練時間。

這種結構繼承了之前版本的 Inception-v3，通過將 Inception 模塊轉換為 Residual Inception 塊，增加了更多的 Inception 模塊，並在 Stem 模塊之後增加了一種新型的 Inception 模塊（Inception-A）進行了改進。

其架構圖如圖 12 所示，而各個模組如圖 13 所示。Stem 為主幹，基本上是一般的 CNN 架構，模組 Inception-A 為通用性 Inception 架構，而其中核心的 InceptionResNet A 至 B 的為混合設計應用的架構，Reduction A 和 B 替代了一般的 Pooling 架構。

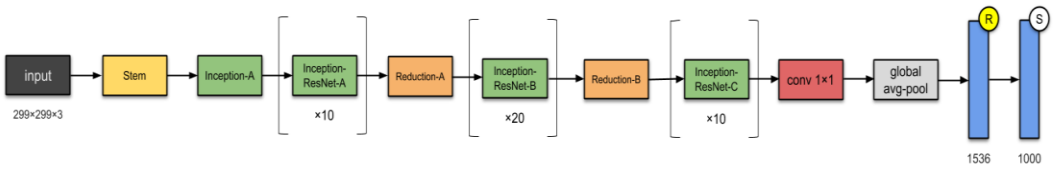

圖 12 Inception-Resnet 系列完整的網絡架構 [15]

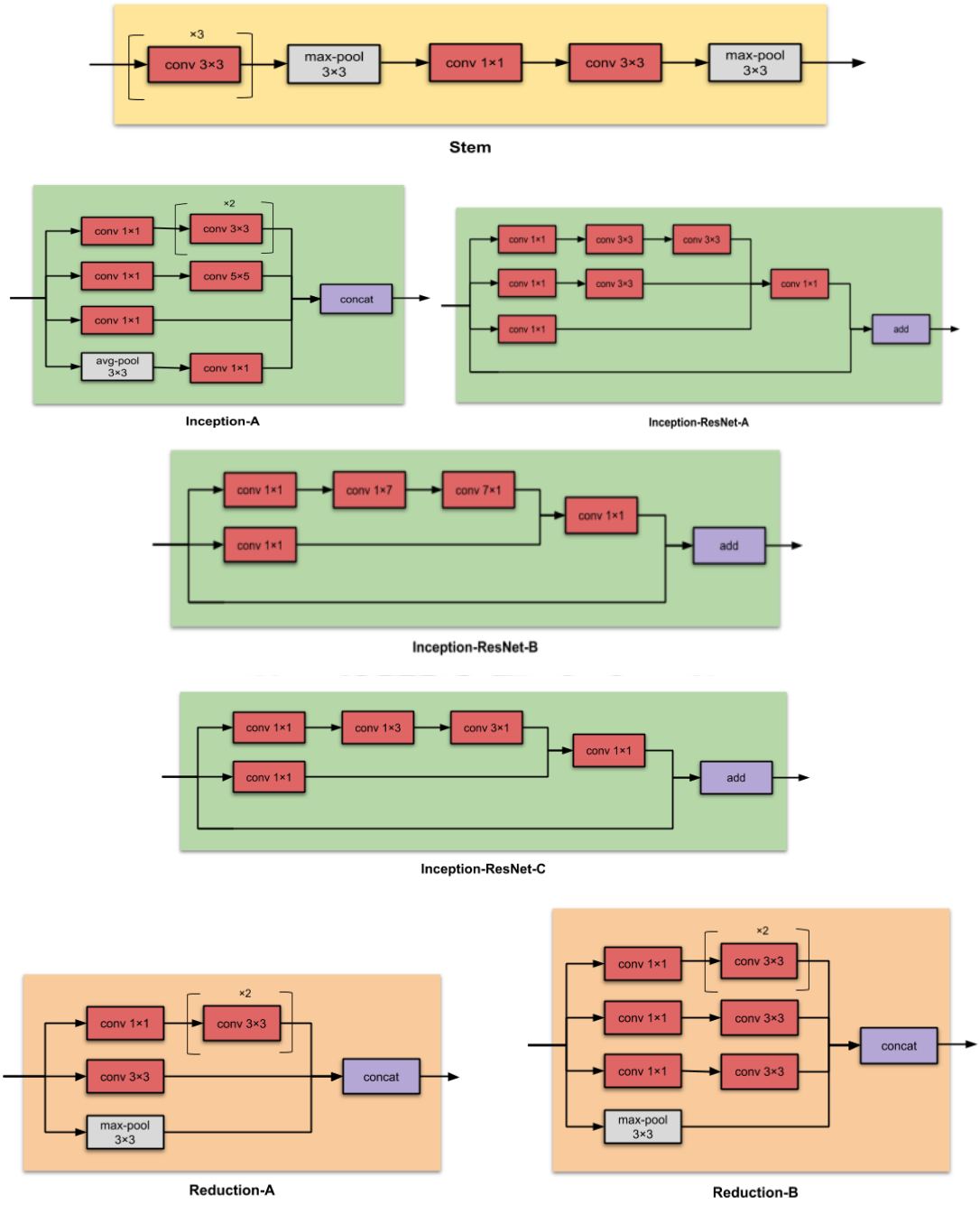

圖 13 Inception-Resnet 子模組的網絡架構 [15]

2.2.6 VGG19

VGG19 是 VGG 模型的變體，簡而言之由 19 層（16 個卷積層、3 個全連接層、5 個 MaxPool 層和 1 個 SoftMax 層）組成。VGG 還有其他變體，如 VGG11、VGG16 等。 VGG19 有 196 億次 FLOP。其架構圖如表 3 所示。

表 3 VGGNet 系列的架構（省略 ReLU 激活函數）

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

資料來源：Very Deep Convolutional Networks for Large Scale Image Recognition [16]

2.3 遷移學習和微調整

遷移學習（Transfer Learning）[4] 是指獲取在一個問題上學習的特徵，並將它們用於一個新的類似問題。 例如，已經學會識別對象 A 的模型的特徵可能有助於啟動旨在識別對象 B 的模型。遷移學習通常用於數據集的數據太少而無法從頭開始訓練全尺寸模型。

在深度學習的背景下，遷移學習最常見的流程為：1. 從先前訓練的模型中獲取層。2. 凍結它們，以避免在未來的訓練回合中破壞它們包含的任何信息。3. 在凍結層的頂部添加一些新的、可訓練的層。他們將學習將舊特徵轉化為對新數據集的預測。4. 在數據集上訓練新層。

最後一個可選步驟是微調整（Fine-Tuning）。解凍先前的整個模型（或其中的一部分），並以非常低的學習率在新數據上重新訓練它。 通過逐步使先前訓練的特徵適應新數據，這可能會實現更精準的結果。

2.4 系統環境架設

2.4.1 Anaconda

Anaconda [17] 是 Python 和 R 的 open-source 發行版。它用於數據科學、機器學習、深度學習等。伴隨著 300 多個數據科學庫的可用性，對於任何程序員來說，在 anaconda 上工作變得相當理想 用於數據科學。Anaconda 有助於簡化包管理和部署。Anaconda 附帶了各種各樣的工具，可以使用各種機器學習和 AI 算法輕鬆地從各種來源收集數據。

Anaconda 包含 pip 和 conda 兩種軟體包管理系統，使用中經常會發生由於各個軟體包版本的不相容性而造成的軟體包混亂和錯誤。pip（遞歸“pip 安裝包”）是一個 Python 包安裝程序。它會下載並安裝您要使用的軟件包，但也僅此而已。它不支援同時安裝多個版本的軟體包，會基於編譯器的版本不相容而造成的編譯錯誤。Conda 是一個包管理器的同時，也是一個環境管理器。它隔離了不同的 python 和軟體包版本，因此它們不會相互交互。但是，Conda 最為詬病的部分在於它的安裝軟體來源於 Anaconda 自身的軟體包管理服務，Anaconda Cloud。其中不乏有軟體包的最新或特定版本對於某平台的不匹配而造成的無法安裝。

2.4.2 Tensorflow 和 Keras

TensorFlow [18] 是由 Google 開發並於 2015 年發布的端到端開源深度學習框架。它以文檔和訓練支持、可擴展的生產和部署選項、多個抽象級別以及對不同平台（例如 Android）的支持。TensorFlow 是一個用於神經網絡的符號數學庫，最適合跨一系列任務的數據流編程。所使用的版本為：2.3.0。

Keras [19] 是一種用 Python 編寫的高效高級神經網絡應用程序編程接口(API)。這個 open-source 神經網絡庫旨在提供深度神經網絡的快速實驗，它可以在 CNTK、TensorFlow 和 Theano 之上運行。Keras 專注於模塊化、用戶友好和可擴展。它不處理低級計算；相反，它將它們交給另一個稱為後端的庫。Keras 於 2017 年年中被採用並集成到 TensorFlow 中。用戶可以通過 tf.keras 模組運行，也可以單獨獨立運行 Keras 庫。所使用的版本為：2.4.0。截至目前，Keras 已與 TensorFlow 完全集成。Keras 團隊不再更新或維護 Keras 的獨立版本。往後所提到的 Keras 將只限制於集成在 TensorFlow 中的 API，而不是單獨的獨立庫。

各個 API 的框架比對如表 4 所示。

表 4 各深度學習框架比對 [20]

	Keras	Pytorch	Tensorflow
API Level	High	Low	High and Low
Architecture	Simple, concise, readable	Complex, less readable	Not easy to use
Datasets	Smaller datasets	Large datasets, high performance	Large datasets, high performance
Debugging	Simple network, so debugging is not often needed	Good debugging capabilities	Difficult to conduct debugging
Does it have trained models?	Yes	Yes	Yes
Popularity	Most popular	Third most popular	Second most popular
Speed	Slow, low performance	Fast, high performance	Fast high performance
Written in	Python	Lua	C++, CUDA, Python

資料來源：simplilearn [20]

2.4.3 GPU 設定

包括 Keras 在內的 TensorFlow 代碼將無需明確的代碼配置透明地在單一 GPU 上運行。TensorFlow GPU 支持目前可用於具有 CUDA 卡的 Ubuntu 和 Windows 系統。唯一的硬件要求是擁有具有 CUDA 計算能力的 NVIDIA GPU 卡。TensorFlow 網站[21] 上有當前支持的版本。軟體需求為：Tensorflow，Nvidia 驅動，CUDA Toolkit[23]，以及 cuDNN[24]。本實驗所使用的 GPU 為 Nvidia 的 GeForce RTX 2060 Super。

使用 GPU 時，就算有設定好 seed 和環境變數 TF_DETERMINISTIC_OPS，也無法把 Model 的訓練控制在 100% 的重現（Deterministic）。當在多個並行的多執行緒上運行運算時，通常是無法確定哪個多執行緒將會先結束。多執行緒何時對自己的數據進行操作並不重要，因此例如，將激活函數應用應該是確定性的。但是當這些多執行緒程需要同步時，例如計算總和時，結果可能取決於求和的順序，進而取決於哪個多執行緒先結束的順序。

第三章 實驗方法與系統設計

3.1 多重遷移學習

學習方法以階段性進行。基於最終目標為辨識一隻狗的名字，先進行犬類的學習，再學習犬臉部特徵，最後再學習其名字。以最小的資源消耗量達為次要目的，達成更精確的辨識率。

3.2 資料集

資料集的收集方式大多是網絡公開類型。以『多重遷移學習』的想法為基礎，先收集的資料集是公開資料集裡含有最多犬類圖像的資料集，Stanford Dogs Dataset [25]。再者是含有臉部特徵點的犬類圖像資料集，Columbia Dogs Dataset [26]。以及僅有的一隻狗的臉部圖像對應其個體名字的資料集，Flickr Dogs Dataset [27]。最後則是透過南華大學『狗狗 gogo 志工社』所收集的在其名義下的校內流浪犬的圖像資料。

3.2.1 Stanford Dogs Dataset

總數 20580 張，120 犬類的圖像資料集並且是目前公開資料集裡含有最多犬類圖像資料的資料集。這資料集的內容含：以犬類分類而成的圖像資料，以及四個僅包含狗全身照的座標。在訓練前，

這資料集需要做的前置處理作業為：透過坐標點抓取僅需的圖像裡狗的全身照，再裁邊保留，如圖 14 所示。

圖 14 Australian Terrier 樣本照裁剪前後對比圖

3.2.2 Columbia Dogs Dataset

總數 8351 張，133 犬類的圖像資料集。資料集內容含：以犬類分類而成的照片，狗臉部的 8 項特徵點的坐標、以及依測試及訓練分開用的照片名稱。基於已經透過 seed 的方式設置資料的訓練和測試用的分類，因此忽略資料集所提供的資料集分類。這資料集在訓練前的前置處理作業為：透過資料集所提供的八個特徵點為參考，裁剪保留圖像裡狗的臉部資料，如圖 15 所示。

圖 15 Australian Terrier 樣本照裁剪示意圖

3.2.3 Flickr Dogs Dataset

總數 374 張，42 只狗的個體名對應其臉部的圖像資料集。含有最容易辨識的『哈士奇』，以及最難辨識的『哈巴狗』的圖像資料。其中的樣本照片如圖 16 以及圖 17 所示。

圖 16 哈巴狗 Bandit

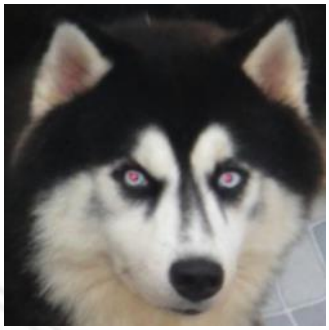

圖 17 哈士奇 Apolo

3.2.4 Nanhua Dogs Dataset

通過南華大學『狗狗 gogo 志工社』所收集的圖像資料集。總數為 103 張照片，對應校內 12 只流浪犬的個體名。在訓練前的前置作處理業為：把個體名改成英文版本，以及裁剪成保留臉部圖像，如圖 18 所示。

圖 18 小白（Whitney）樣本照裁前後對比圖

3.3 Tensorflow 訓練程式說明

通過『遷移學習』的方式把現有的卷積神經網絡（CNN）重新訓練。以 imagenet 訓練結果的權重為基礎宣告各個 model。

訓練參數為：Seed = 0 （含 python，numpy，tensorflow），batch size = 20，epochs = 100。Seed 為隨機種子，用於初始化偽隨機數生成器的數字（或向量）。batch size 為在更新模型之前處理的樣本數，epochs 為通過訓練集的完整通過次數。（資料集將所設定的 batch size 拆分成批次，而每一個 epoch 則會牽扯到相應的批次，完整訓練數為epoch $\times \frac{\text{訓練集總資料數}}{batch\_size}$ ）。

所應用 Tensorflow API 的圖像預處理函式為：

```
datagen = ImageDataGenerator(
    shear_range = 0.1,
    zoom_range = 0.1,
    brightness_range = [0.9,1.1],
    horizontal_flip = True,
    validation_split = 0.3,
    preprocessing_function = preprocess_input
)
train_generator = datagen.flow_from_directory(
    file_dir,
```

```
    target_size = (224, 224),
    batch_size = batch_size,
    class_mode = 'categorical',
    shuffle = True,
    seed = seed,
    subset = "training"
)
test_generator = datagen.flow_from_directory(
    file_dir,
    target_size = (224, 224),
    batch_size = batch_size,
    class_mode = 'categorical',
    shuffle = False,
    seed = seed,
    subset = "validation"
)
```

ImageDataGenerator 是以實時數據資料增強於生成批次的圖像資料函式。shear_range 為推移調整範圍，zoom_range 為變焦調整範圍，brightness 為亮度調正範圍，horizontal_flip 為鏡向翻轉。validation_split （這邊用於 Test split）是指從資料集獨立出來的用於測試驗證比例（0.3 則為 7 成的訓練集，3 成的測試集）。

Preprocessing_function=preprocess_input 則是指定為各個 CNN 的預處理函式。

train_generator 和 test_generator 則是用於指定訓練用和測試用的子資料集。透過先前所定義的 imageDataGenerator 所繼承的函式 flow_from_directory 把圖像資料從檔案路徑呼叫。file_dir 為檔案的路徑，target_size 為圖像調整大小，class_mode 為數據類型（category 為類別型數據），shuffle 為是否數據混排，subset 用於抓取先前所定義的 imageDataGenerator 的子資料集（訓練用的“training”或測試用的 “validation”）。

所應用 Tensorflow API 的 callbacks 為：

```
checkpoint = ModelCheckpoint(
        checkpoint_dir,
        monitor = 'val_loss',
        save_best_only = True,
        save_weights_only = False,
        save_freq='epoch'
)
```

ModelCheckpoint 是用於以設定的頻率保存模型或權重。checkpoint_dir 是檔案儲存路徑。monitor = ‘val_loss’ 是指函式監控的值數（val_loss 為 validation loss，驗證資料集的損失；也可用

val_accuracy，驗證資料集的準確率）。save_best_only 是指依照所監控的值數，僅儲存最好的 model。save_weights_only 為是否僅存權重。save_freq 為儲存頻率，這邊的應用則是以每個 epoch 結束為頻率監控檢測並儲存。

```
reducelr = ReduceLROnPlateau(
        monitor = 'val_loss',
        factor = np.sqrt(0.1),
        patience = 5,
        min_lr = 0.0000001
)
```

ReduceLROnPlateau 是用於當指標停止改進時降低學習率。監控指標為統一的 'val_loss'，factor 為降低學習率的因子（new_lr = lr × factor），patience 為所設定的無改善的 epoch 基準數往後降低學習率，min_lr 為學習率的下限值。

```
earlystop = EarlyStopping(
        monitor = 'val_loss',
        min_delta = 0.0001,
        patience = 20,
)
```

EarlyStopping 是用於當指標停止改進時停止訓練。監控指標為統一的 'val_loss'。min_delta 為能視為改進的最小變化，小於指定值則不會被視於改善。patience 為沒有改善的 epoch 數，往後則停止訓練。

所應用的 CNN model 為： DenseNet 201， EfficientNet B7（基於記憶體容量關係，batch size 重新設定為 5）， InceptionResNet V2， Inception V3，ResNet152 V2 以及 VGG19。以 imageNet，訓練結果為基礎，再以 include_top = False 把最後的輸出辨識層排除掉。程式碼範例為：

```
from tensorflow.keras.applications.efficientnet import
        EfficientNetB7, #CNN 名字
        preprocess_input
base_model = EfficientNetB7( #以各 CNN 名字作修正
        include_top = False,
        weights = 'imagenet',
        input_shape = (224, 224, 3)
)
```

先是把先前宣告的 CNN 為基礎的最後一層（統稱為 output）鏈接，在其基礎上加一層 global average pooling，以及兩次 dropout，batch normalization 和 dense 層，並重新訓練。最後一層 dense 直連輸出辨識層。程式碼為：

```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(rate = 0.2, seed = seed)(x)
x = BatchNormalization()(x)
x = Dense(
        1280,
        activation = 'relu',
        kernel_initializer = glorot_uniform(seed=seed),
        bias_initializer = 'zeros')(x)
x = Dropout(rate = 0.2, seed = seed)(x)
x = BatchNormalization()(x)
predictions = Dense(
        classes,
        activation = 'softmax',
        kernel_initializer = random_uniform(seed = seed),
        bias_initializer = 'zeros')(x)
model = Model(inputs = base_model.input, outputs = predictions)
```

global average pooling 用於替換傳統 CNN 中的完全連接層。dropout 層在訓練期間的每一個訓練步中以所設定的頻率（0.2）將輸入隨機設為 0。batch_normalization 以所設定的 batch 為基準，把前一層的結果標準化。最後連接完整鏈接的 dense 層，以標準格式 softmax 為激活函式。透過 kernel_initializer 以先前宣告好的隨機種子 seed，對應上隨機函式 glorot uniform 和 random uniform 宣告初始權重。最後以 bias_initializer = 'zeros' 統一宣告偏差值為 0。

遷移學習則砍掉重新訓練最後一層。基於最後一層 dense 直連輸出辨識層，因此程式碼為把 model 最後兩層彈出，並以同樣架構重新宣告。程式碼為：

```
model = load_model(old_model_dir)
x = model.layers[-2].output #popping output & prediction dense layer
predictions = Dense(
        classes,
        activation = 'softmax',
        kernel_initializer = random_uniform(seed = seed),
        bias_initializer = 'zeros')(x)
model = Model(inputs = model.input, outputs = predictions)
```

最後以標準的 Adam 為優化函式配上先前設定的學習，和 categorical crossentropy 為 loss 函式編譯神經網絡。當然再加上把神經網絡先前的所有層重新設定為可訓練。程式碼為：

```
optimizer = Adam(lr=0.0001)
loss = "categorical_crossentropy"
for layer in model.layers:
    layer.trainable = True
model.compile(optimizer = optimizer,
    loss = loss,
    metrics = ["accuracy"])
```

3.4 系統架構

訓練完成的神經網絡 model 透過 Flask 伺服器公開於學校區域網絡，再以 Android App 固態 IP 地址抓取用於辨識。

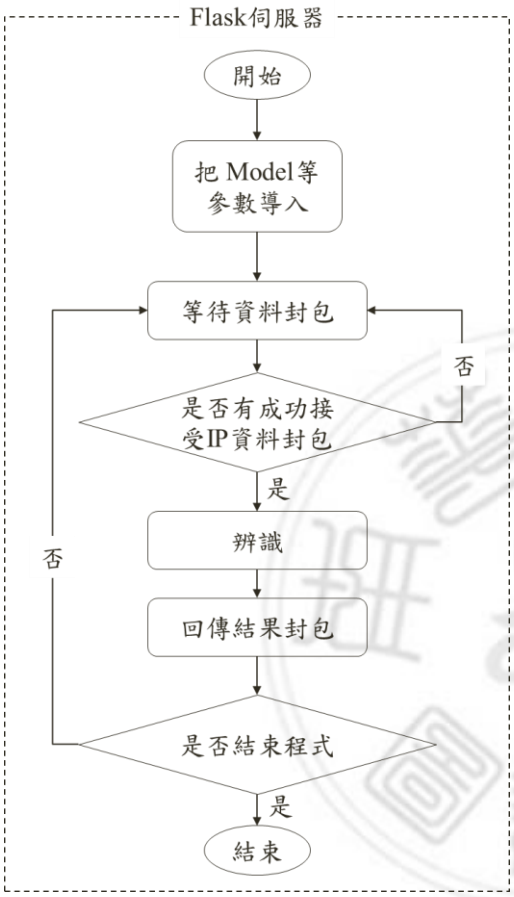

圖 19 Flask 伺服器運作流程圖

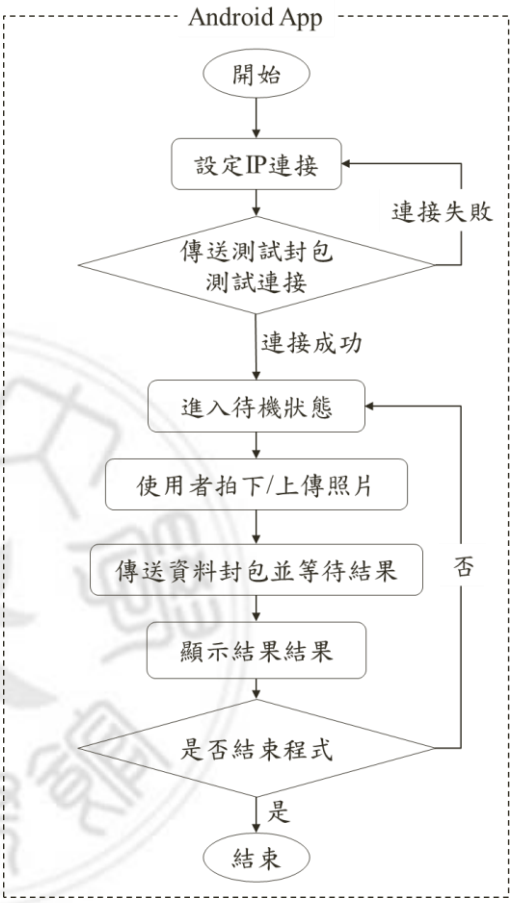

圖 20 Android App 運作流程圖

3.4.1 Flask 伺服器

伺服器以固態 IP 地址架設，各個 model 以不同的網域子地址的方式等待呼叫。伺服器處理方式為：從 Android 系統單向接受 flask 傳輸圖像一張，並儲存於伺服器內。透過標準的 ImageNet 權重偵測圖像是否是一隻狗，是則進入辨識功能並把前 5 項辨識結果以 Json 格式回傳，否則回傳錯誤 Json 訊息。

這種系統的架構設計是基於幾點原因的考慮。一、本系統能以不同的 IP 子地址的方式，分開處理不同 Model 的辨識。二、雲端處理系統目前多為綁定 Github 處理，並限制一次處理的最高記憶體載量為 500MB，容量大的 Model 不適用。

目前這系統有一個隱患，越多的 model 架構會佔用越多的記憶體。雖然可以通過在需要的時候呼叫宣告所需的 model 和參數，但也會造成嚴重的延遲。

3.4.2 Android App

App 的處理方式為通過手機相機抓取對象照片，並回傳到 Flask 伺服器進行辨識，最後再把結果回傳顯示於 App 上。使用者也可以選擇以上傳照片的方式把照片傳到 Flask 伺服器進行辨識的功能。

其操作流程為，先設定網絡 IP 連接伺服器，這邊包含選擇辨識功能（辨識名字或品種以及各個 Model 的選項），這是基於伺服器仍未定點架設於學校的區域網絡而是架設在私人筆電裡，其操作介面如圖 21 所示。

其次，使用者可以透過攝影或上傳照片兩種方式選擇讓本系統抓取應回傳給伺服器的狗狗的照片，其中的介面如圖 22 所示。

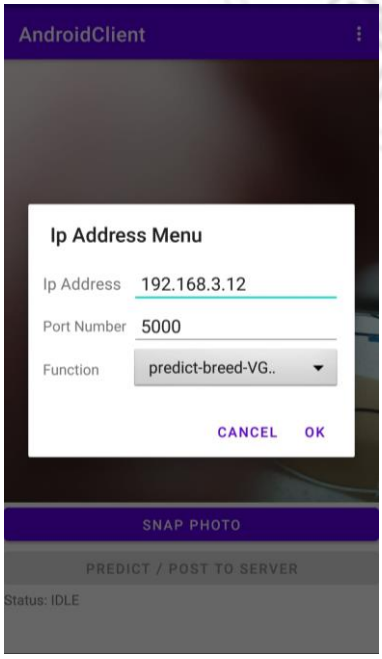

圖 21 Android App Ip 設定和功能選擇介面

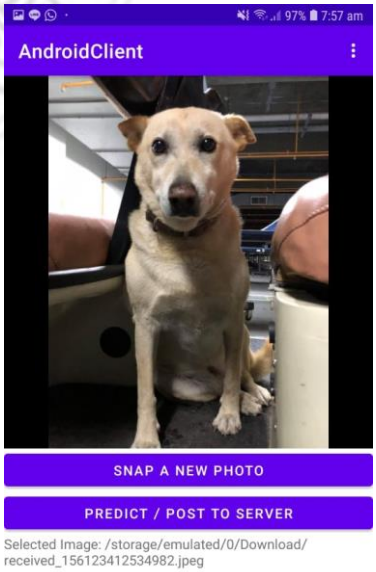

圖 22 拍攝或上傳照片介面

最後使用者點選『PREDICT / POST TO SERVER』按鍵把照片上傳到伺服器等待辨識結果的回傳，並把結果透過邊界式的餅形圖的方式以及細節數據的方式顯示出來，如圖 23 以及圖 24 的所示。邊界式的餅形圖將會把最高的結果置放於上中部，並會以順時鐘的方式依序顯示前 5 項辨識結果，其中留空的部分為辨識成其他的結果。而辨識結果完整的細節數據則顯示於 App 下半的狀態欄。

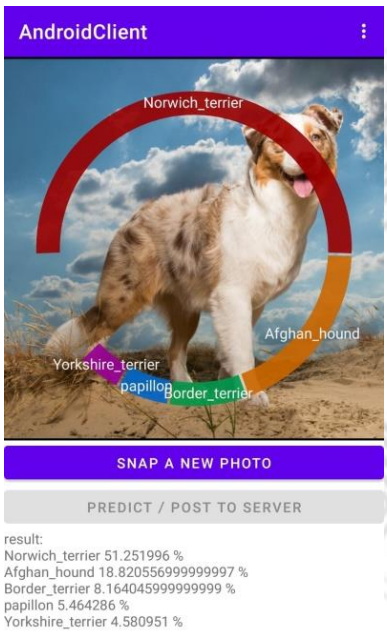

圖 23 辨識狗種的結果介面

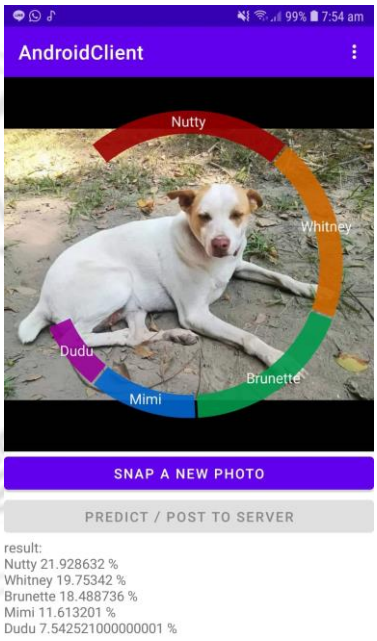

圖 24 辨識狗名的結果介面

第四章 實驗結果與討論

4.1 CNN 訓練成果

訓練成果透過兩種方式收集進行分析。左邊的曲線圖為透過收集 Model 的每一 Epoch 訓練的精準度和損失而成的曲線圖；右邊的矩陣圖為透過 Tensorflow 的 Model Checkpoint 函式抓取的最優 Model 的辨識結果，其中色調為數據的體現方式，而排版的含義分別是縱向的正確結果以及橫向的辨識結果，由左上至右下為最優解。

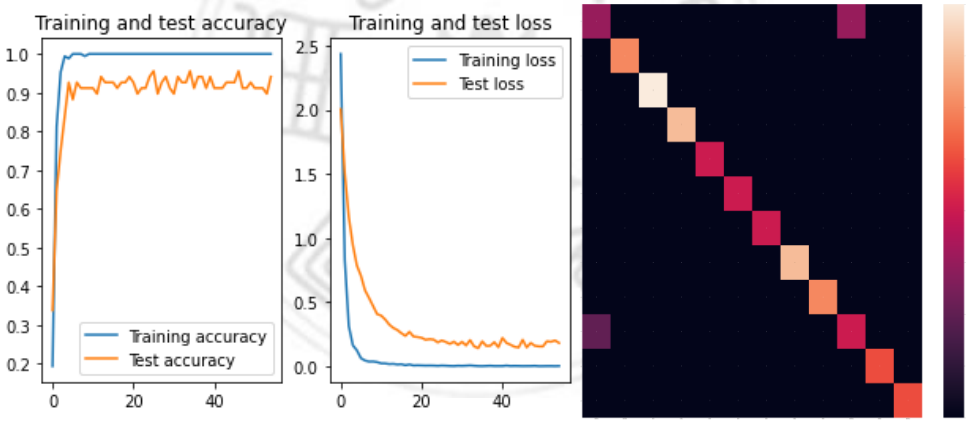

圖 25 DenseNet201 nanhua 訓練結果

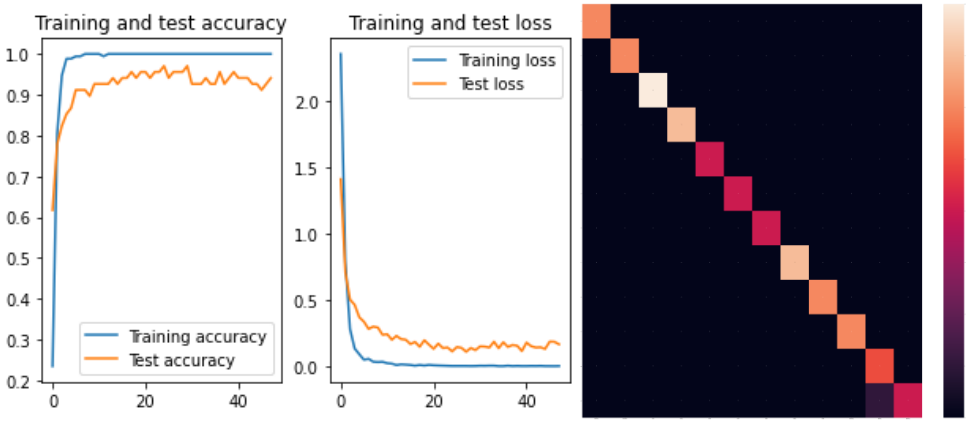

圖 26 DenseNet201 nanhua 最終訓練結果(Stanford-Columbia-Flickr-Nanhua)

圖 25 為 DenseNet201 基於 Nanhua Dogs Dataset 單一資料集的訓練結果：以 epoch 55 提前結束，耗時 7 分鐘 42 秒，辨識率為 93%。圖 26 為本論文提案的漸層資料集做遷移學習的最終成果：以 epoch 48 提前結束，最終資料集（Nanhua）的耗時為 5 分鐘 43 秒，包含前面資料集的總累計耗時為 3 小時 1 分鐘 10 秒。DenseNet201 在本實驗裡的變化雖然不顯著，不過也是有比較些許的優化。

在所訓練的各個 CNN 架構裡，儘管錯誤率是各個 model 裡最高的，VGG19 的變化仍是最為顯著的。如圖 26 所示，VGG19 在單一訓練 Nanhua Dogs Dataset 的成果：100 epoch 完整訓練，總耗時 9 分鐘，辨識率為 53%，而且仍有無法辨識的對象所在。相對的，透過本論文提案的漸層資料集做遷移學習的最終成果則如圖 27 所示。最終的訓練成果為：epoch 61 的訓練提前結束，5 分 31 秒的最後資料集訓練，加上前提的三個資料集訓練，總耗時為 5 小時 41 分鐘 06 秒，辨識率為 71%。

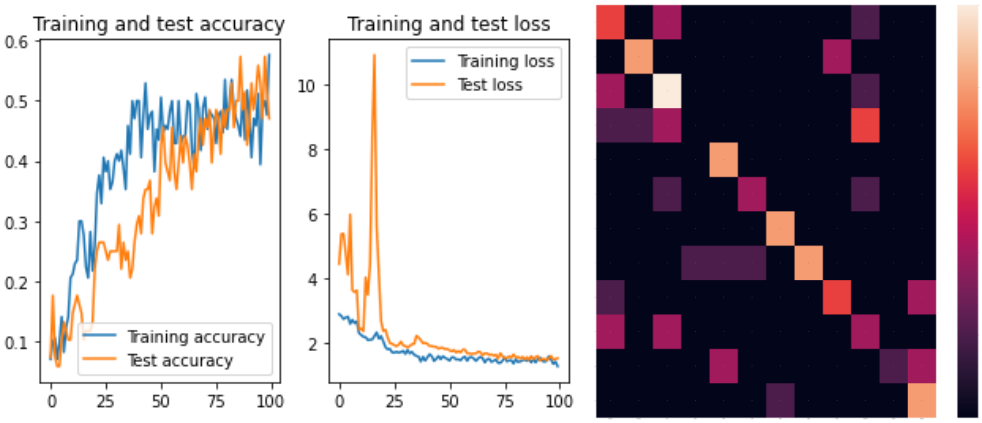

圖 27 VGG19 nanhua 訓練結果

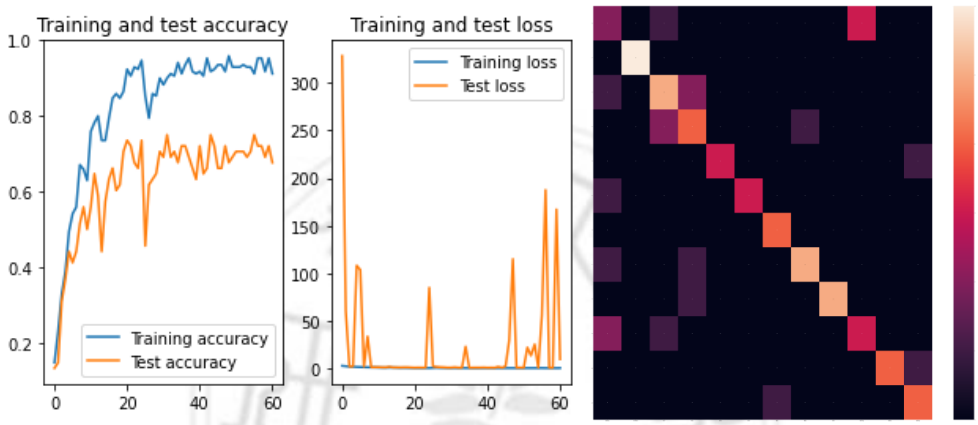

圖 28 VGG19 最終訓練結果 (Stanford-Columbia-Flickr-Nanhua)

然而也不是每一個訓練都能一帆風順，其中不乏有嚴重無法忽視的情況存在。如圖 29 所示，透過 Flickr，再訓練 Nanhua 資料集的成果裡，辨識率為 88%，但是混淆矩陣能明顯看到第一對象（Blackey）裡的辨識幾乎是錯誤的。這種情況在 InceptionV3，InceptionResNetV2，ResNet152V2 的訓練結果裡比較常見，如圖 30 以及圖 31 所示。

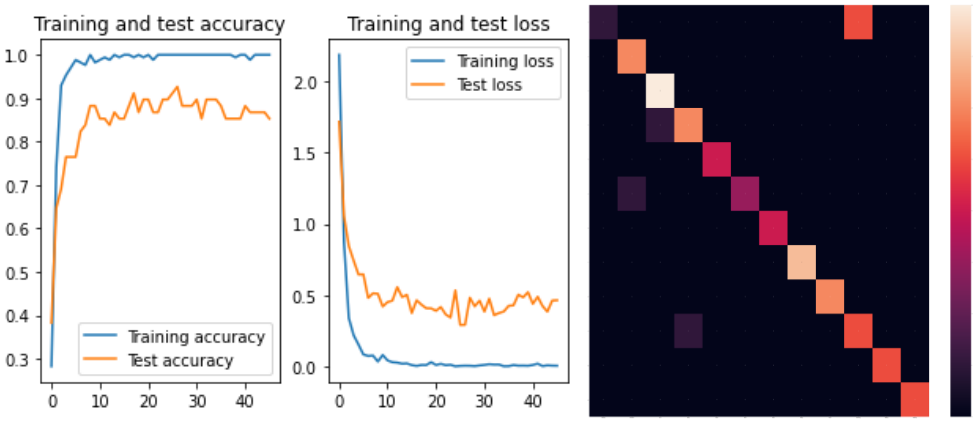

圖 29 InceptionV3 flickr-nanhua 訓練結果

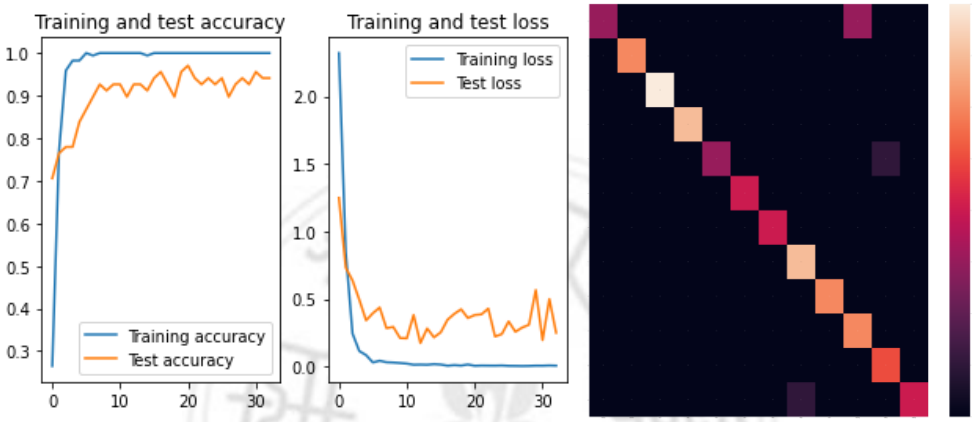

圖 30 InceptionResNetV2 columbia-nanhua 訓練結果

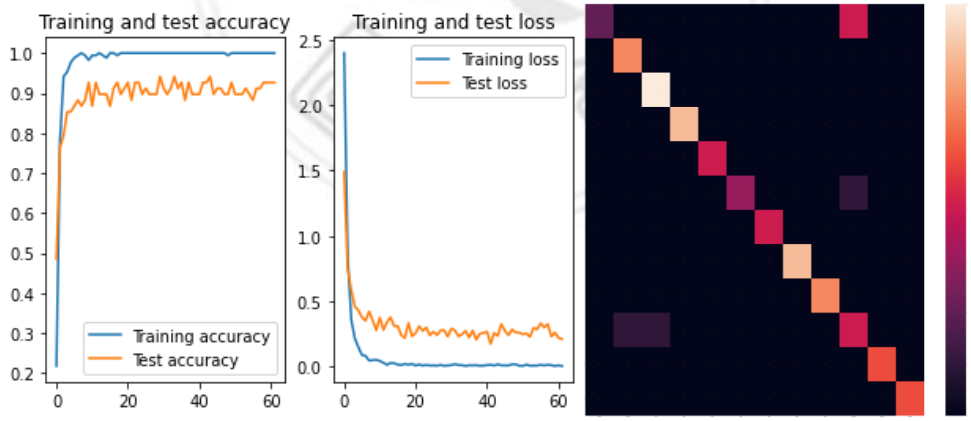

圖 31 ResNetV2 stanford-flickr-nanhua 訓練結果

4.2 CNN 資源應用

在訓練這段期間除了準確率，另一個重點就是如何節省所耗費的時間。Earlystopping 函式不僅能做到這點，還能有效的防止一個 Model 由於過度訓練所造成的 overfitting。

透過 Earlystopping 函式讓訓練提早結束為止的 epoch 記錄如表 5 所示，而其中所耗費的時間記錄如表 6 所示。左四欄為所訓練的資料集，其數號為訓練集的訓練順序。右六格為各個 CNN 架構的個別記錄，full-trained 為完整 100 epoch 訓練，無提早結束。

透過表 5 和 6 的資料加上以下算式 $(總\ epoch\ 數 - 提早停止\ epoch\ 數) \times (耗時 \div 提早停止\ epoch\ 數)$，推斷出均節省耗時如表 7 所示。

表 5 各個訓練在提早結束訓練為止所通過的 epoch 數量

Stanford	Columbia	Flickr	Nanhua	DenseNet201	EfficientNetB7	Inception ResNetV2	InceptionV3	ResNet152V2	VGG19
1				epoch 28	epoch 21	epoch 21	epoch 29	epoch 31	epoch 45
	1			epoch 35	epoch 24	epoch 69	epoch 35	epoch 46	**full-trained**
		1		epoch 76	epoch 71	epoch 71	epoch 65	epoch 76	epoch 75
1	2			epoch 38	epoch 59	epoch 32	epoch 50	epoch 46	epoch 33
1		2		epoch 65	epoch 64	epoch 36	**full-trained**	epoch 46	epoch 91
	1	2		epoch 59	epoch 75	epoch 38	epoch 55	epoch 40	epoch 98
1	2	3		epoch 59	epoch 62	epoch 47	epoch 46	epoch 51	epoch 81
			1	epoch 55	epoch 78	epoch 40	epoch 43	epoch 59	**full-trained**
1			2	epoch 55	epoch 44	epoch 32	epoch 43	epoch 58	epoch 41
	1		2	epoch 64	epoch 54	epoch 33	epoch 53	epoch 42	epoch 63
		1	2	epoch 65	epoch 47	epoch 62	epoch 46	epoch 41	epoch 93
1	2		3	epoch 60	epoch 91	epoch 33	epoch 55	epoch 47	epoch 93
1		2	3	epoch 85	epoch 31	epoch 35	epoch 41	epoch 62	epoch 79
	1	2	3	epoch 71	epoch 52	epoch 26	epoch 55	epoch 80	epoch 86
1	2	3	4	epoch 48	epoch 36	epoch 40	epoch 54	epoch 43	epoch 61
平均提早結束 epoch 數				57.53333	53.93333	41	51.33333	51.2	75.93333
最低提早結束 epoch 數				28	21	21	29	31	33
最高提早結束 epoch 數				85	91	71	full-trained	80	full-trained
加總 epoch 數				863	809	615	770	768	1139
總節省 epoch 數				637	691	885	730	732	361

表 6 各個訓練在提早結束訓練為止的耗時

Stanford	Columbia	Flickr	Nanhua	DenseNet201	EfficientNetB7	Inception ResNetV2	InceptionV3	ResNet152V2	VGG19
1				1 小時 47 分鐘 12 秒	4 小時 35 分鐘 10 秒	1 小時 26 分鐘 30 秒	1 小時 41 分鐘 37 秒	2 小時 37 分鐘 36 秒	4 小時 11 分鐘 56 秒
	1			56 分鐘 10 秒	2 小時 10 分鐘 55 秒	2 小時 20 分鐘 16 秒	48 分鐘 18 秒	1 小時 44 分鐘 12 秒	3 小時 47 分鐘 6 秒
		1		6 分鐘 58 秒	19 分鐘 17 秒	8 分鐘 2 秒	4 分鐘 54 秒	9 分鐘 14 秒	8 分鐘 46 秒
1	2			1 小時 1 分鐘 33 秒	5 小時 27 分鐘 45 秒	1 小時 6 分鐘 11 秒	1 小時 10 分鐘 47 秒	1 小時 44 分鐘 28 秒	1 小時 14 分鐘 17 秒
1		2		7 分鐘 43 秒	20 分鐘 31 秒	6 分鐘 3 秒	8 分鐘 43 秒	7 分鐘 46 秒	10 分鐘 26 秒
	1	2		7 分鐘 3 秒	24 分鐘 27 秒	5 分鐘 53 秒	4 分鐘 58 秒	7 分鐘 20 秒	11 分鐘 26 秒
1	2	3		6 分鐘 42 秒	19 分鐘 36 秒	7 分鐘 51 秒	4 分鐘 18 秒	8 分鐘 45 秒	9 分鐘 22 秒
			1	7 分鐘 42 秒	17 分鐘 11 秒	6 分鐘 45 秒	4 分鐘 32 秒	8 分鐘 14 秒	9 分鐘
1			2	6 分鐘 7 秒	9 分鐘 56 秒	4 分鐘 24 秒	4 分鐘 33 秒	7 分鐘 35 秒	3 分鐘 47 秒
	1		2	7 分鐘 9 秒	12 分鐘 2 秒	4 分鐘 24 秒	4 分鐘 59 秒	5 分鐘 37 秒	5 分鐘 42 秒
		1	2	7 分鐘 25 秒	10 分鐘 33 秒	7 分鐘 25 秒	4 分鐘 22 秒	5 分鐘 21 秒	8 分鐘 29 秒
1	2		3	7 分鐘 4 秒	19 分鐘 4 秒	4 分鐘 38 秒	5 分鐘 13 秒	6 分鐘 16 秒	8 分鐘 23 秒
1		2	3	8 分鐘 56 秒	7 分鐘 18 秒	4 分鐘 29 秒	3 分鐘 57 秒	7 分鐘 28 秒	7 分鐘
	1	2	3	8 分鐘 11 秒	11 分鐘 19 秒	3 分鐘 3 秒	5 分鐘 21 秒	9 分鐘 49 秒	7 分鐘 48 秒
1	2	3	4	5 分鐘 43 秒	8 分鐘 33 秒	5 分鐘 2 秒	4 分鐘 57 秒	5 分鐘 32 秒	5 分鐘 31 秒
總耗時				3 小時 13 分鐘 38 秒	4 小時 24 分鐘 37 秒	2 小時 04 分鐘 56 秒	2 小時 43 分鐘 29 秒	3 小時 39 分鐘 13 秒	2 小時 56 分鐘 59 秒

表 7 各訓練均訓練所節省的耗時

Stanford	Columbia	Flickr	Nanhua	DenseNet201	EfficientNetB7	Inception ResNetV2	InceptionV3	ResNet152V2	VGG19
1				3 小時 42 分鐘	17 小時 14 分鐘 54 秒	5 小時 25 分鐘 13 秒	3 小時 36 分鐘 33 秒	5 小時 50 分鐘 45 秒	5 小時 08 分鐘
	1			1 小時 44 分鐘	6 小時 54 分鐘 12 秒	1 小時 03 分鐘 02 秒	1 小時 29 分鐘 55 秒	2 小時 02 分鐘 24 秒	無
		1		02 分鐘 24 秒	07 分鐘 44 秒	03 分鐘 23 秒	02 分鐘 55 秒	02 分鐘 48 秒	02 分鐘 55 秒
1	2			1 小時 40 分鐘 14 秒	3 小時 47 分鐘 33 秒	2 小時 20 分鐘 32 秒	1 小時 10 分鐘 50 秒	2 小時 02 分鐘 24 秒	2 小時 30 分鐘 45 秒
1		2		04 分鐘 05 秒	11 分鐘 24 秒	01 分鐘 04 秒	無	54 秒	01 分鐘 03 秒
	1	2		04 分鐘 47 秒	50 秒	09 分鐘 18 秒	03 分鐘 45 秒	11 分鐘	14 秒
1	2	3		04 分鐘 47 秒	12 分鐘 02 秒	53 秒	05 分鐘 24 秒	49 秒	02 分鐘 13 秒
			1	06 分鐘	04 分鐘 46 秒	01 分鐘	05 分鐘 42 秒	05 分鐘 28 秒	無
1			2	05 分鐘 15 秒	13 分鐘 04 秒	09 分鐘 04 秒	05 分鐘 42 秒	05 分鐘 36 秒	05 分鐘 54 秒
	1		2	04 分鐘 12 秒	09 分鐘 58 秒	08 分鐘 56 秒	04 分鐘 42 秒	07 分鐘 44 秒	03 分鐘 05 秒
		1	2	04 分鐘 05 秒	11 分鐘 29 秒	04 分鐘 26 秒	05 分鐘 24 秒	07 分鐘 52 秒	35 秒
1	2		3	04 分鐘 40 秒	01 分鐘 57 秒	08 分鐘 56 秒	04 分鐘 30 秒	07 分鐘 04 秒	35 秒
1		2	3	01 分鐘 30 秒	16 分鐘 06 秒	08 分鐘 40 秒	05 分鐘 54 秒	04 分鐘 26 秒	01 分鐘 45 秒
	1	2	3	03 分鐘 23 秒	10 分鐘 24 秒	08 分鐘 38 秒	04 分鐘 30 秒	02 分鐘 20 秒	01 分鐘 10 秒
1	2	3	4	06 分鐘 04 秒	14 分鐘 56 秒	08 分鐘	03 分鐘 50 秒	07 分鐘 36 秒	03 分鐘 15 秒
總節省耗時				7 小時 57 分鐘 26 秒	29 小時 51 分鐘 19 秒	10 小時 01 分鐘 05 秒	7 小時 09 分鐘 36 秒	10 小時 59 分鐘 10 秒	8 小時 01 分鐘 29 秒

EfficientNetB7 基於其最大參數量的前提省下最多的時間。VGG19 儘管提早停止的效能不明顯，仍是節省了不少時間，可推斷其效率無法與其他的架構匹敵。InceptionV3 儘管有一個無法獲得 earlystopping 函式的優惠，可其他的訓練裡提早停止的 epoch 數相當低，推測是有什麼異常，有待修改參數作重新訓練。InceptionResNetV2 是整體而言最快完成訓練的架構。

表 8 為總訓練結果（stanford-columbia-flickr-nanhua）的各個 CNN 架構資訊比較表。EfficentNetB7 基於其複雜性，總訓練時間是最高的。VGG19 雖然能透過其總參數的最少量間接獲得最快運算速度的優勢，可其辨識準確性有待加強。InceptionV3 在不擴張太多參數值的容量依然與 DenseNet201 有著匹配的辨識準確性。ResNet152V2 以匹配 EfficientNetB7 的參數量，卻有著更少的訓練時間。InceptionResNetV2 在擁有最快訓練結束時間，但基於其參數量以及與 ResNet152V2 及 InceptionV3 一樣的單一個體的錯誤辨識，故排除這三個 Model。

表 8 總結果 model 資訊比較

CNN 架構	檔案容量（四捨值）	總參數	辨識準確性	總訓練時間（含前置訓練時間）
DenseNet201	245 MB	20,809,036	99%	3 小時 01 分鐘 10 秒
EfficientNetB7	791 MB	67,406,499	96%	10 小時 31 分鐘 04 秒
InceptionResNetV2	663 MB	56,330,732	99%	2 小時 45 分鐘 34 秒
InceptionV3	288 MB	24,454,188	97%	3 小時 01 分鐘 39 秒
ResNet152V2	717 MB	60,983,052	87%	4 小時 36 分鐘 21 秒
VGG19	243 MB	20,703,564	71%	5 小時 41 分鐘 06 秒

4.3 CNN 辨識率

表 9 各個 Model 於各個訓練階段時所達成的辨識率

Stanford	Columbia	Flickr	Nanhua	DenseNet201	EfficientNetB7	Inception ResNet V2	InceptionV3	ResNet152 V2	VGG19
1				83%	78%	80%	81%	78%	34%
	1			90%	85%	90%	89%	89%	71%
		1		87%	87%	71%	68%	83%	27%
1	2			91%	90%	90%	90%	89%	66%
1		2		89%	85%	78%	80%	86%	52%
	1	2		89%	91%	84%	81%	86%	42%
1	2	3		94%	89%	88%	81%	84%	70%
			1	93%	90%	91%	93%	88%	53%
1			2	96%	93%	99%	94%	94%	75%
	1		2	96%	97%	94%	99%	93%	75%
		1	2	94%	94%	91%	88%	94%	35%
1	2		3	99%	93%	99%	97%	90%	79%
1		2	3	94%	93%	96%	94%	90%	72%
	1	2	3	94%	94%	93%	93%	93%	71%
1	2	3	4	99%	96%	99%	97%	87%	71%
平均辨識率（整體）				93%	91%	90%	88%	88%	59%
最低辨識率（整體）				83%	78%	71%	68%	78%	27%
最高辨識率（整體）				99%	97%	99%	99%	94%	79%
平均辨識率（目標，Nanhua）				96%	94%	95%	94%	91%	66%
最低辨識率（目標，Nanhua）				93%	90%	91%	88%	87%	35%
最高辨識率（目標，Nanhua）				99%	97%	99%	99%	94%	79%

在表 9 的比較中，VGG19 雖然可以在各個階段看到比較顯著的辨識率變化，但在整體上表現仍不如其他架構。EfficientNetB7 基於其性質，在各階層的辨識率變化為偏低甚至為減少，推斷此 Model 已經進入瓶頸。而以表 9 的方式表示， Flickr 資料集基於其複雜性，能發現其中各個 Model 的單一訓練結果幾乎普遍為最低，而往後訓練的 Nanhua 資料集則相對偏高，故能推斷 Nanhua 資料集並不如 Flickr 資料集的複雜。DenseNet201 在各個階段的變化雖然不顯著，但在辨識率以及仍有優化的基礎將其作為最優 Model 保留應用。

4.4 系統測試結果

本系統基於有必要針對各個 Model 的實地測試而採取分段式 Model 呼叫應用。透過使用者選擇所需的辨識進而呼叫和宣告對應的 Model 進行辨識。這邊會造成一個僵局：過多的 Model 呼叫應用會導致記憶體過載而造成的錯誤，雖然能夠通過程式把各個 Model 在辨識後進行從記憶體去除，但相對的必須在每次需要時重新呼叫各個 Model，這會造成大量的延遲。

另外網絡的狀況也會影響本系統的效率。本系統必須把照片傳輸到伺服器後，才能進行辨識這一點，會因網絡 IP 封包的傳送而造成的延遲。Model 可以以更簡化的方式直接架構於 App 上，但會辨識效能會被打折。

本系統是透過 flask 直接把 Model 以單一 IP 的方式分享到學校的廣域網絡上，因此也有網絡安全的疑慮。

第五章 結論與未來展望

『深度學習』這一技術由於現代電腦的運算能力增強，以及 GPU 加速因此得以普及。而以『狗』為範圍的動物生物辨識除了能幫助以圖像處理的方式尋找失踪的寵物犬以外，還能往後處理結合額外資料辨別一隻狗的『混血』，甚至能偵測一隻狗的外傷和生理情況。以下為後續研究方向：

（1）雲端運算處理優化

目前本系統為以實體電腦作為後端伺服器把 Model 架構於以學校為範圍的廣域網絡上。這種設計法把系統綁定於學校的網絡系統上，使其變得極其被動於自主營運。另外本系統完全忽略了網絡安全的疑慮。這兩項問題都能通過『雲端運算處理優化』的方式解決。只要把 Model 架構於公開的雲端處理中心如 Amazon 和 Google 的雲端處理中心就能得以避免。

（2）Nanhua Dogs 資料集的更新和複雜化

目前所使用的辨識目標的資料集基於單一來源的關係，其中特徵值的重複性較高。再加上考慮到會有流浪犬離開和增加的可能性，有必要再次收集資料並且增加資料集的複雜化。

（3）辨識系統和 CNN Model 架構的更新和優化

目前的系統為以圖像臉部辨識系統為主，有望把 CNN Model 在不損失太多的辨識效能的情況下優化處理速度，以方便往後把臉部圖像辨識系統更新成影像辨識系統。

（4）前端平台的優化和/或移植

目前的系統為在 Android 系統上的主動操作式系統。其中的功能並不多，而且會因網絡和遠端處理的關係造成一定的延遲。可以把前端處理器移植到限定手機使用的網頁版，或直接把前端和後端一起移植到微處理上，如樹莓派。

參考文獻

[1] “維基百科，烙印” [Online] Available: https://en.wikipedia.org/wiki/Livestock_branding

[2] “維基百科，嵌入式微晶片” [Online] Available: https://en.wikipedia.org/wiki/Microchip_implant_(animal)

[3] B. Yao, G. Bradski, and L. Fei-Fei, “A codebook-free and annotationfree approach for fine-grained image categorization,” in IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 3466– 3473

[4] L. Torrey and J. Shavlik, “Transfer learning,” in Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques, 2010, pp. 242–264.

[5] “維基百科，人工智能史” [Online] Available: https://en.wikipedia.org/wiki/History_of_artificial_intelligence

[6] Intel. “How to Get Started as a Developer in AI.” October 2016. [Online] Available: https://software.intel.com/en-us/articles/how-to-get-started-as-a-developer-in-ai.

[7] “C. Camacho, Convoulutional Neural Networks,” 2018 [Online] Available: https://cezannec.github.io/Convolutional_Neural_Networks/

[8] G. Huang, Z. Liu, L. Maaten, and K.Q. Weinberger, “Densely Connected Convolutional Networks”, in CVPR, 2017.

[9] M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.” 2019.

[10] “V. Agarwal, Complete Architectural Details of all EfficientNet Models,” 2020 [Online] Available: https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142

[11] K. He, X. Zhang, S. Ren, and J. Sun, “Identity Mappings in Deep Residual Networks” 2016.

[12] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna , “Rethinking the Inception Architecture for Computer Vision” 2015.

[13] “SH. Tsang, Review: Inception-v3 — 1st Runner Up (Image Classification) in ILSVRC 2015,” 2018 [Online] Available: https://sh-tsang.medium.com/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

[14] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning" 2016

[15] “R. Karim, Illustrated: 10 CNN Architectures,” 2019 [Online] Available: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#643c

[16] K. Simonyan, and A. Zisserman “Very Deep Convolutional Networks for Large-Scale Image Recognition”, 2015

[17] “Anaconda Documentation,” [Online] Available: https://docs.anaconda.com/

[18] “Tensorflow API Documentation,” [Online] Available: https://www.tensorflow.org/api_docs

[19] “About Keras,” [Online] Available: https://keras.io/about/

[20] “Keras vs Tensorflow vs Pytorch: Understanding the Most Popular Deep Learning Frameworks，” May 2021，[Online] Available: https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article

[21] “Tensorflow, GPU Support” [Online] Available: https://www.tensorflow.org/install/gpu#hardware_requirements

[22] “NVIDIA, CUDA GPUs,” [Online], Available: https://developer.nvidia.com/cuda-gpus

[23] “NVIDIA, CUDA Toolkit Archive,” [Online] Available: https://developer.nvidia.com/cuda-toolkit-archive

[24] "NVIDIA cuDNN," [Online] Available: https://developer.nvidia.com/cudnn

[25] A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei, "Novel dataset for fine-grained image categorization," in First Workshop on FineGrained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition, Colorado Springs, CO, June 2011.

[26] J. Liu, A. Kanazawa, D. Jacobs, and P. Belhumeur. Dog breed classification using part localization. In Computer Vision– ECCV 2012, pp. 172–185.

[27] Moreira, T.P., Perez, M.L., Werneck, R. *et al.* Where is my puppy? Retrieving lost dogs by facial features. *Multimed Tools Appl* **76,** 15325–15340 (2017).