Final Project

組別：第 8 組

組員：劉昱旻、張峻瑋、葉俠愷

一、 專題目標

透過訓練模型以實現一動物圖片分類器。目標是完成多個模型，比較不同模型在卡通、彩繪、素描等風格下之圖片分類準確率。

二、 圖片前處理

(一) 找圖片的方法

在訓練資料的圖片來源中，我們每個動物分類總共找了 4000 張左右的圖片。其中有 1000 張圖片取材自第四個作業助教所提供的動物圖片；1000 張卡通圖片全部自己找；1000 張素描圖片有 800 張是從助教所提供的原始圖片轉的，有 200 張是自己找的；1000 張繪畫圖片中，400 張水彩風格與 400 張油畫風格的圖片是轉自助教的原始圖片，其餘 200 張則是自己找的。

在找圖片方面，先後用了三個不同的方法。第一個是到免費圖片素材網站，透過 API 與網站聯繫，下載所需要的圖片資料。然而這個方式有一點慢，而且往往沒有辦法找到足夠的資料，於是之後我們採用另一種方式：使字 bing_image_downloader 套件。雖然每一個關鍵字大約只能找到 100 張圖片，但可透過頻繁更換關鍵字，即可找到足量的圖片，過程並不會太耗時間。第三個是使用 Google 搜尋圖片如搜尋 cartoon elephant，然後使用 chrome 的 download all images (https://chrome.google.com/webstore/detail/download-all-images/ifipmflagepipjokmbdecpmjbibjnakm)插件將圖片全部下載下來並手動濾除.svg 檔就完成了。

(二) 圖片風格轉換

我們透過 cv2.xphoto.oilPainting(image, 7, 1)函式將原始圖片（如圖 1）轉成油畫風格（圖 2）。水彩部分，我們使用

```
    oil_painting_image = cv2.xphoto.oilPainting(image, 5, 8)
    filtered_image = cv2.bilateralFilter(oil_painting_image, 10, 250,
    250)
```

這兩個函式，將圖片轉成水彩風格（如圖 3）。

圖 1：蝴蝶原圖　　圖 2：油畫風格蝴蝶　　圖 3：水彩風格蝴蝶

在素描部分，完整函式如下：

```
def sketch(image):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)
    edges = cv2.Canny(blurred_image, 30, 100)
    _,    inverted_edges    =    cv2.threshold(edges,    128,    255,
cv2.THRESH_BINARY_INV)
    return inverted_edges
```

首先先將圖片轉成灰階，接著使用高斯濾波使圖片邊緣平滑，然後透過 Canny()函式留下邊緣，並透過閾值檢測只留下黑色與白色，清除灰色，以達到素描的效果，成果如圖 4。

圖 4：素描風格蝴蝶

(三) 資料增強

在完整建立 model 的過程中，從網路上找到充足的圖片資源是非常困難的，既然無法再找到更多的資源，我們只能從已經找到的資源裡下手，我們對這些圖片進行處理，用來增加訓練資料集的數據，彌補訓練資料量不足的問題，同時提升模型訓練能力及增加泛化能力，以下為我們實作中有用到的一些轉換。

1. Resize

將 PIL 影像進行影像縮放到固定大小，而實作中，我們是將大小不一的各個影響，統一調整至 256 x 256 的大小。

2. Randomcrop

以圖片(PIL Image)中隨機裁減一塊圖像出來，因為我們要辨識的目標可能只在圖片中的一小角，透過這個方式我們能透過隨機的方式更容易將目標找到，而在實作中，我們擷取的大小為 224 x 224。

3. ColorJitter

隨機調整圖片的亮度(brightness)、對比(contrast)、飽和度(saturation)和色調(hue)，實作中我們只有調整亮度。

4. RandomHorizontalFlip、RandomVerticalFlip

圖片(PIL Image)會在給定的機率下隨機進行水平或是垂直翻轉，實作中翻轉的機率各為 0.5。

5. RandomGrayscale

圖片(PIL Image)會在給定的機率下隨機進行灰階的轉換，圖片轉成灰階會與素描的版本較為相像，一方面灰階的轉換是為了增加素描的樣本量，另一方面，轉換為灰階是為了增加卡通、畫、素描間的關聯性，讓建立的 model 能根據這個關聯性更加容易判斷，在實作中轉成灰階的機率為 0.4。

6. RandomRotation

將圖片隨機旋轉，設定為 45 度，表示能在-45 度及 45 度隨機取一個值。

7. Randomaffine

圖片(PIL Image or torch tensor)保持中心不變的圖像的隨機仿射變換，角度前面有設定過了，這邊就沒有再做，這邊做的為水平及垂直平移，設定的為 0.2。

8. Normalize

將影像(torch tensor)的每個 channel(R,G,B)依據平均數和標準差分別進行影像的正規化(Z-score)，實際上，透過 normalize 後的圖片人眼較難辨識，但對於電腦來說能更有規則性，會更好辨認，所以當訓練及測試圖片送入 model 前，我們也會做 normalize，這後面會提到。

以上幾種轉換，雖然人眼能輕鬆分辨出這幾種轉換後的圖片都來自同一張，但對於電腦及機器來說，這是張完全不同的資料，如圖 5 參考，也是因為這個原因，我們才能以這種方式來增加圖片資料。

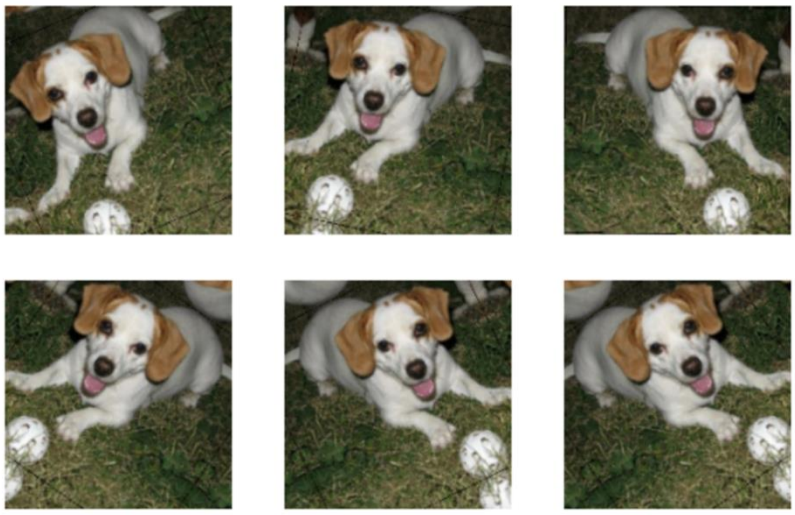

圖 5：不同轉換下的圖片

三、 訓練

(一) 流程

1. 設定參數

batch size

Epoch = 15 → Epoch = 30

1 個 Epoch 指當所有資料都被用來訓練 CNN 模型一次，我們設定的值一開始為 15，但發現 model 都準確率都仍在上升，為了找到收斂的飽和點，所以改成 30 個 Epoch，而對於不同的 model，Epoch 也是代表著紀錄點，例如 VGG 系列，因為參數量非常大，所以需要大量的 GPU 計算，又因為我們使用的是 colab 免費的資源，常常使用到 GPU 用量不足，從而導致訓練過程的中斷，所以我們會以 Epoch 數作為中斷的紀錄，在每個 Epoch 結束後都會儲存模型，能在訓練中斷的情況下，幫助我們使用別的帳號或是在有免費資源的下次時間能繼續訓練。

Batch = 32

將所有資料分成多堆分批放到模型裡訓練，而使用 batch 最大的目的是找出一小部分資料的 loss 就去更新模型參數，我們選擇的是 32。

1. batch size 會影響到運算的速度，但也不一定，batch size 越小時，參數更新的時間會越短，但相反的 batch size 越小時，需要在一個 epoch 被更新的次數較會越多，相反時間就拉長。

2. 通常 batch size 較小時，效果會有比較好的準確率

```
Learning rate = 0.0001
```

這個參數負責控制模型的學習，learning rate 最主要會對梯度下降產生影響，當 learning rate 太小時，對模型的權重更新都非常小，這也讓整個訓練都變得緩慢，但當 learning 過大時，會導致無法收斂。

2. 圖片轉換及打包

首先，我們會使用 torch 的內建函式 ImageFolder 來將放在資料夾裡的圖片打包，而 ImageFolder 的好處是會根據資料夾的名稱，自動給每張圖片設定一個標籤，而不需要另外寫函式來設定，打包完後進行分類，以 8:2 的比例將資料夾裡的圖片分成訓練資料以及驗證資料，所以訓練資料總共約為 32000 張，而驗證資料則約為 8000 張，另外，我們會給每個總類加上 400 到 500 張的加強訓練資料，也就是經過上面所提的一系列轉換而成的，另外，每個類別也會有 100 到 200 張的驗證資料。

在丟入 model 訓練之前，圖片包裡的所有圖片也會像增強資料一樣使用 torchvision 裡的函式進行轉換，只不過，在這邊轉換的目的是希望能更加清楚的找到同種類間圖片的連結性和能將不同種類圖片區分的依據，而非像是增強資料一樣是為了補足不足的資料量，所以轉換並不需要太多，在我們的設計裡只有幾個部分，首先，將大小不一的圖片統一縮放至 224 x 224，另外就是如同資料增強所提，將圖片經過 normalize，讓 model 的訓練能更佳的找到分類的依據。

3. 訓練過程(訓練前處理+訓練+驗證)

首先先提到訓練前的一些步驟，首先是我們所使用的 loss function，cross-entropy(交叉熵)，這個損失函數通常用於分類問題，而非講義裡最常出現的 MSE(Mean-Square-Error)，如同 MSE 希望能最小化均方差，cross-entropy 則希望最小化「交叉熵」。第二，建立 model，我們使用的方法並非直接建造架構，而是直接使用 torchvision 內部所建立的一些 model，且 pretrained 設定成 False，等同使用這些 model 在沒有權重的前提下從頭訓練，因為我們選擇的都是深度及架構都非常複雜的 model，所以訓練的過程相當花時間，同時對於硬體資源的要求十分高。

接著，我們就可以根據上面所設定的參數來投入訓練，進行 30 個 Epoch 的訓練，首先為 model 的訓練模式，model.train()，在每次的訓練過程裡，依照設定的 batch size 來將圖片資料一包一包送入訓練，這時，根據訓練中的 model 對於每一批 batch 所預測的結果以及實際上的比較，loss 以及準確率(accuracy)會被計算出來，同時模型參數也會根據 learning rate 的設定而有不同的速度來被更新，當所有訓練圖片都被送完時，model 的訓練模式也就結束了，接下來的是 model 的評估模式，model.eval()，我們使用在一開始準備好的驗證資練來去檢查 model 的表現，一樣圖片會依照設定的 batch size 一包一包送入，將預測的結果與實際情形作比較，算出 loss 以及 accuracy，但這時參數的權重不會被更新，所有驗證圖片跑完，我們能從 loss 是否減少以及驗證資料準確率是否增加，來評估 model 經過這個 epoch 的訓練後，表現是否有所進步，經過我們設定的 epoch 的訓練後，訓練過程就結束了。

(二) 各模型之分析與介紹

以下為我們使用的七個 mode 介紹及表現

1. VGG16

(1) model 介紹

VGG16 是牛津大學在 2014 年提出的捲積類神經網路，在當時具有優異的圖形辨識表現。16 代表總共有 13 層捲積層與 3 層全連接層。

(2) 實作結果

VGG16 訓練過程相對耗時，所以我分為 0-15，15-25，25-30 把 30 個 epoch train 完。

圖 6 為 0-15 個 epoch 的 accuracy 與 loss 對於 epoch 的折線圖。可以看到 accuracy 大致對數型上升，loss 大致指數型下降。但 valid accuracy 相比於 train accuracy 在第 12 個 epoch 大致收斂，Valid accuracy 卻在大約第 6 個 epoch 早早的收斂，到第 15 個 epoch，valid 大概比 train 小了 15%，valid loss 也在第 6 個 epoch 不再下降了。

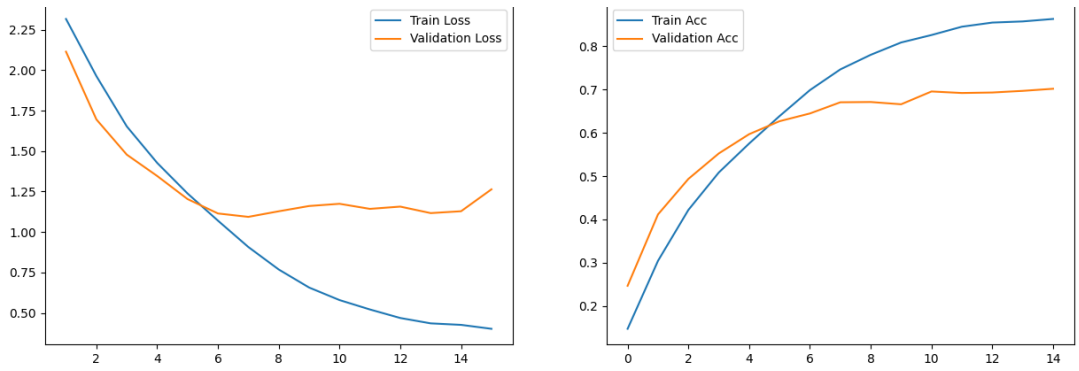

圖 6：VGG16 前 15 個 epoch 的 loss(左)，前 15 個 epoch 的 accuracy(右)

圖 7 為 15-25 個 epoch 的 accuracy 與 loss 對於 epoch 的折線圖。可以看到 train accuracy 雖然幅度不大卻穩定成長，train loss 也是雖然幅度不大卻穩定下降。但 valid accuracy 和 valid loss 卻是在震盪。Valid loss 更是有上升的趨勢。Valid 與 train 已經脫節。

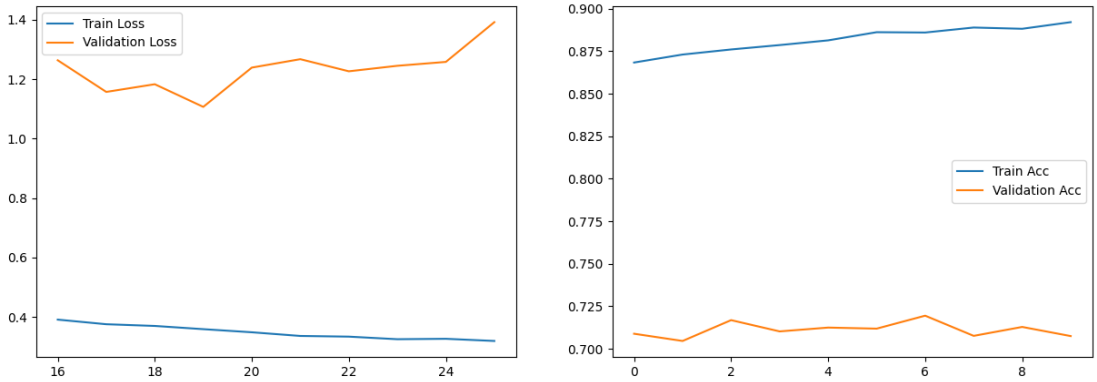

圖 7：VGG16 15-25 個 epoch 的 loss(左)，15-25 個 epoch 的 accuracy(右)

圖 8 為 26-30 個 epoch 的 accuracy 與 loss 對於 epoch 的折線圖。可以看到 train accuracy 與 valid accuracy 已經不太成長，歷經 5 個 epoch，兩者卻只上升不到 0.05%。train loss 也是不太變。Valid loss 則在特定範圍內震盪。

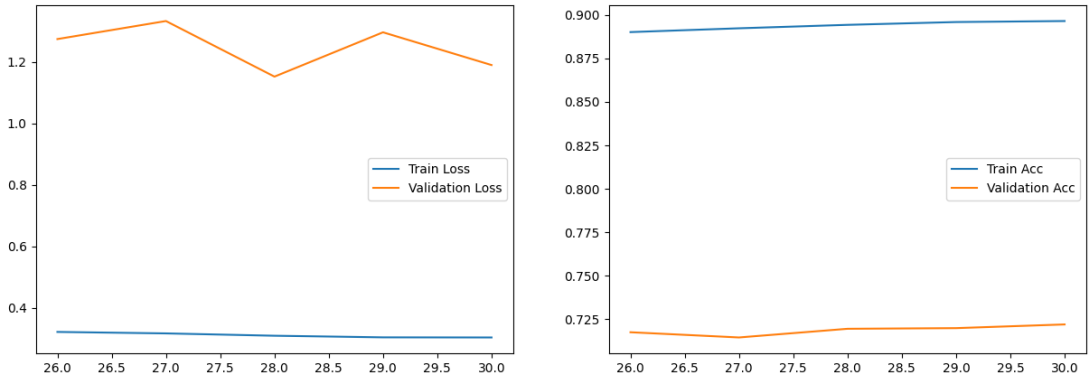

圖 8：VGG16 26-30 個 epoch 的 loss(左)，26-30 個 epoch 的 accuracy(右)

總結來說 VGG16 在前 15 個 epoch， accuracy 有一定的上升幅度。但後 15 epoch train accuracy 上升幅度不大，valid accuracy 有震盪的現象產生。

2. VGG19

(1) model 介紹

VGG 系列的特點是使用多個較小的卷積核(3 x 3)的卷積層來取代具有較大卷積核(5 x 5)的卷積層，這樣的設計能在同樣大小的圖片區域裡，提高網路的深度，從而能進行更多的非線性映射，另一方面則能降低參數量，又因為架構的簡單，能透過不斷疊加的方式來增加 model 的深度，而 VGG19 的設計則是比 VGG16 多了 3 層卷積層，相反的，VGG 系列的缺點也很明顯，因為模型太過肥大，導致模型的效率非常差，需要非常多的運算單元才能完成 model 參數的計算及更新。

(2) 實作結果

實作過程中，因為 model 的運算太過吃硬體資源了，所以我們將 VGG19 的訓練分成三次來進行完成，以下為三次的結果。

model 名稱	best_train_loss	best_train_acc	final_train_loss	final_train_acc	best_valid_loss	best_valid_acc	final_valid_loss	final_valid_acc
vgg19_LIU_10	0.690061	0.771	0.690061	0.771	1.178287	0.643	1.19222	0.643
vgg19_LIU_20	0.375312	0.872	0.375312	0.872	1.176845	0.678	1.287063	0.678
vgg19_LIU_30	0.325684	0.888	0.325684	0.888	1.499077	0.691	1.567718	0.68

表 1：VGG19 實作結果

Epoch(到) 10

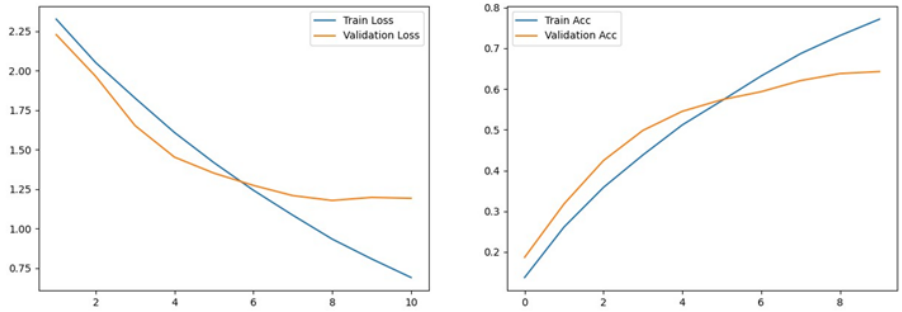

圖 9：VGG19 前 10 個 epoch 的 loss(左)，前 10 個 epoch 的 accuracy(右)

1. Train loss 以及 Train acc 分別以線性的方式減少及增加。

2. Valid loss 以及 Valid acc 分別減少及增加，但皆在 Epoch = 5 時，變化率開始下降，趨於飽和。

Epoch(到) 20

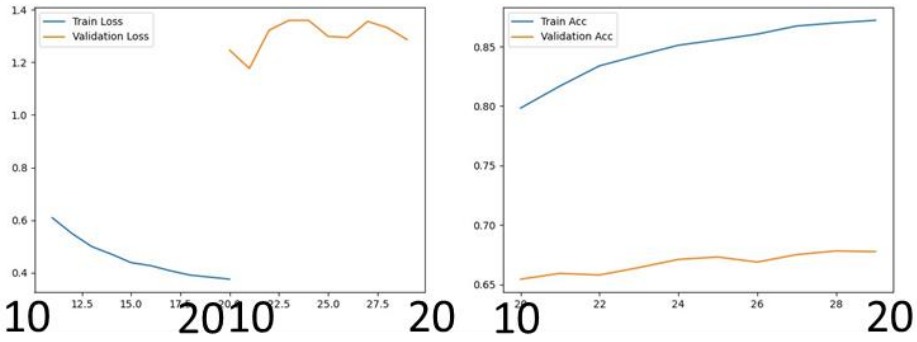

圖 10：VGG19 11-20 個 epoch 的 loss(左)，11-20 個 epoch 的 accuracy(右)

1. Train loss 以及 Train acc 仍分別減少及增加。

2. Valid loss 趨於飽和，上下跳動、Valid acc 仍有些微的成長。

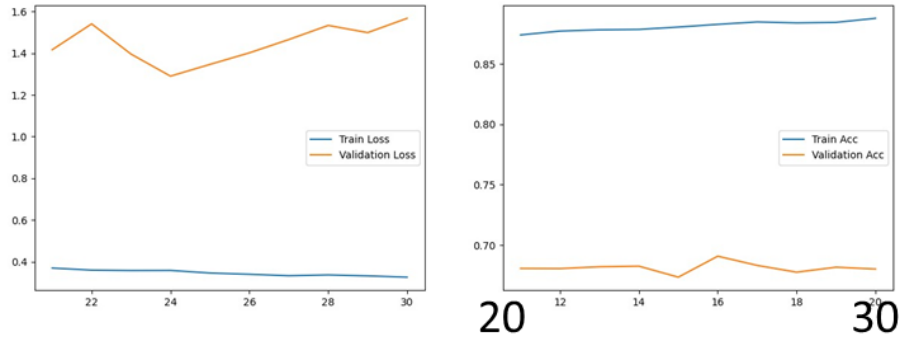

圖 11：VGG19 21-30 個 epoch 的 loss(左)，21-30 個 epoch 的 accuracy(右)

1. Train loss 以及 Train acc 仍分別減少及增加，程度非常小。

2. Valid loss、Valid acc 皆趨於飽和，上下跳動、但沒有明顯成長。

3. ResNet34

(1) 實作結果

best_train_loss	best_train_acc	final_train_loss	final_train_acc
0.297	0.901	0.297	0.901
best_valid_loss	best_valid_acc	final_valid_loss	final_valid_acc
0.986026	0.748	1.130765	0.742.

表 2：ResNet34 實作結果

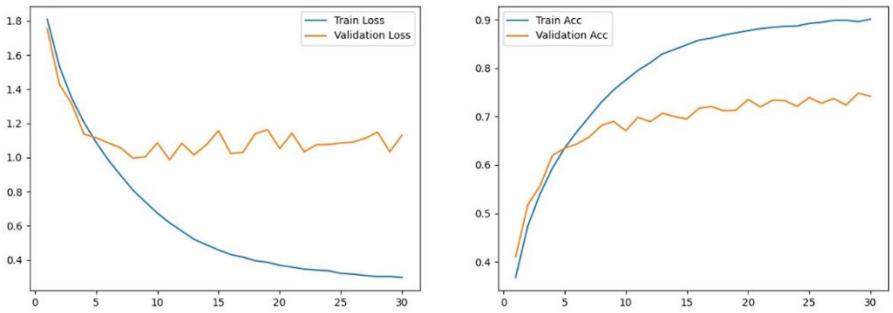

圖 12：ResNet34 1-30 個 epoch 的 loss(左)，1-30 個 epoch 的 accuracy(右)

1. Train loss 以及 Train acc 仍分別減少及增加，程度由大而小

2. Valid loss 在 Epoch = 10 後趨於飽和，上下跳動、但沒有明顯成長，Valid acc 在 Epoch = 10 後跳動，但仍有上升的趨勢

3. 較不吃硬體資源，所以能一次跑完。

4. ResNet50

(1) model 介紹

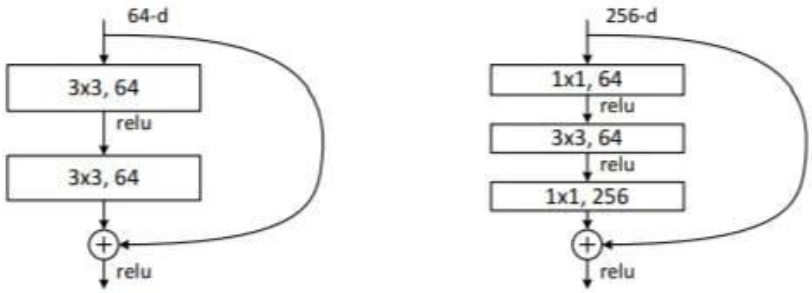

Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

圖 13：ResNet50 之 bottleneck 結示意圖

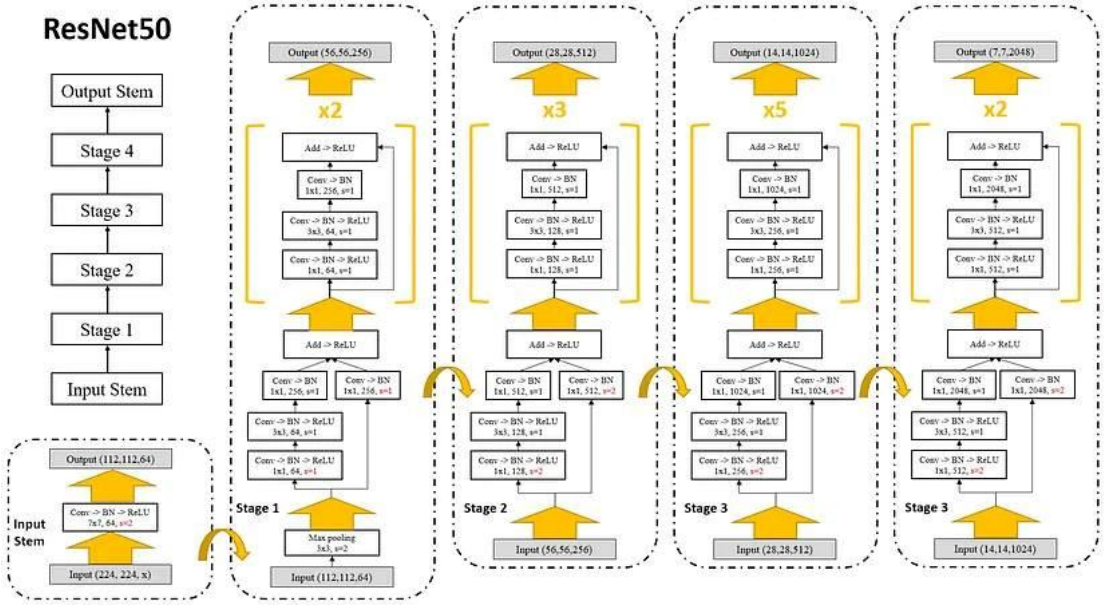

圖 14：ResNet50 內部架構

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

圖 15：ResNet50 內部捲積

ResNet50 與 ResNet34 最大的不同在於，前者使用了 bottleneck block 的設計，如圖 13。如此一來即便層數變多，運算量也不至於過度膨脹。而其他的差異主要是內部的層數，可參閱圖 14 與圖 15。

(2) 實作結果

best_train_loss	best_train_acc	final_train_loss	final_train_acc
0.375489	0.875	0.416874	0.861
best_valid_loss	best_valid_acc	final_valid_loss	final_valid_acc
1.076554	0.709	1.20994	0.694

表 3：ResNet50 實作結果

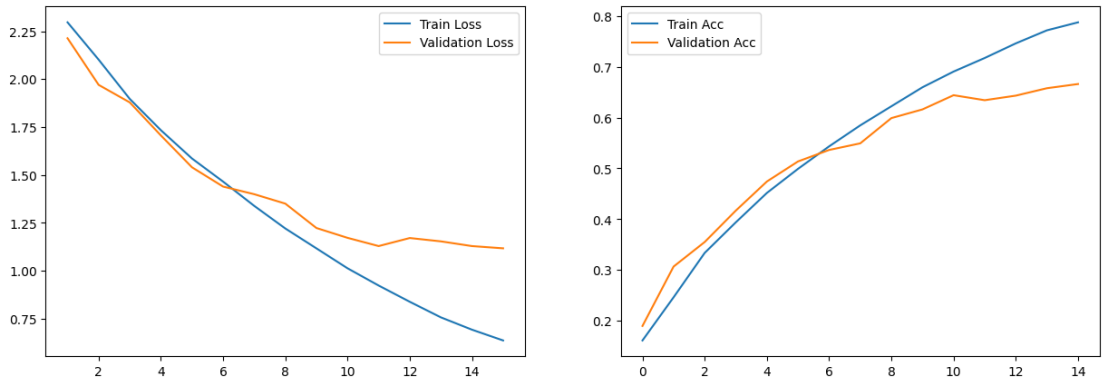

圖 16：ResNet50 1-15 個 epoch 的 loss(左)，1-15 個 epoch 的 accuracy(右)

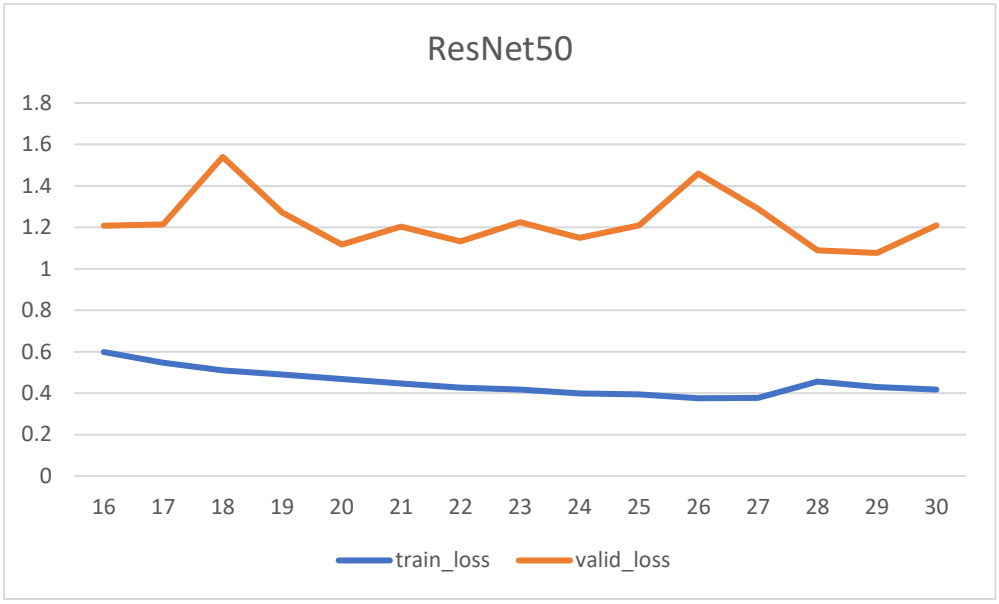

圖 17(a)： ResNet50 16-30 個 epoch 的 loss

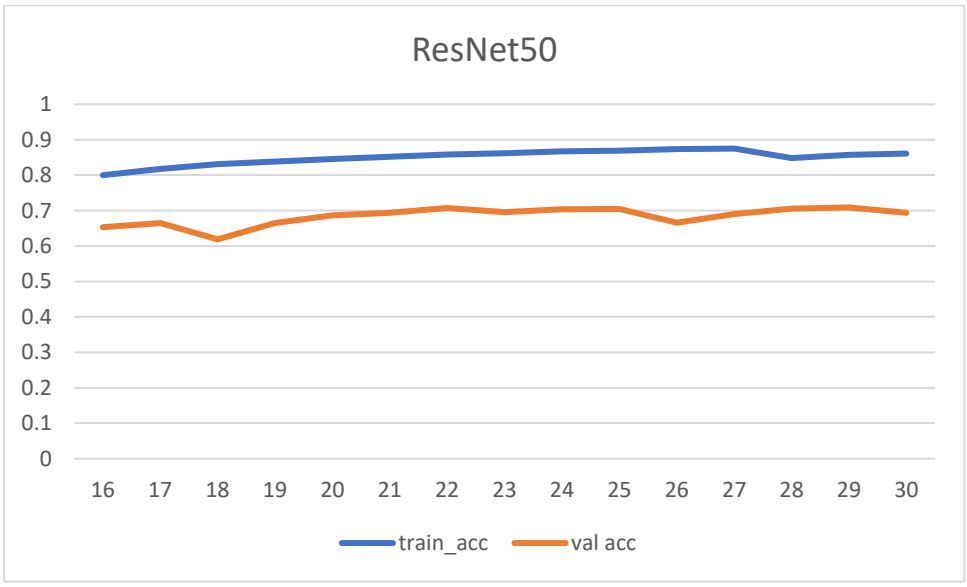

圖 17(b)：ResNet50 16-30 個 epoch 的 accuracy

從圖 16、17 可見，在前 15 個 epoch 訓練損失與驗證損失都不斷在減少，而在後 15 個 epoch 則趨於平緩。從圖 16、17 可見，前 15 個 epoch 準確率不斷在增加，而後 15 個 epoch 則趨於收斂，訓練準確度可達 8 成 6，而驗證準確率可逼近 7 成。

5. ResNeXt50

(1) model 介紹

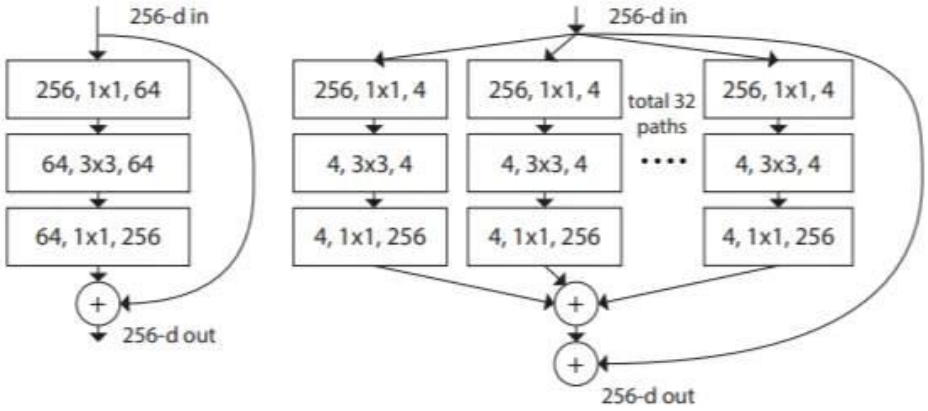

Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

圖 18：ResNeXt 的 cardinality

ResNeXt 與 ResNet 系列最大的不同，在於引入 Split-transfer-merge 的概念，也就是資料輸入進來後，分成許多條平行的路進行運算，如圖 18。此方法在 Inception 模型亦有使用，但 Inception 過於複雜，參數調整不易，是遜於 ResNeXt 的地方。

(2) 實作結果

best_train_loss	best_train_acc	final_train_loss	final_train_acc
0.348442	0.884	0.348442	0.884
best_valid_loss	best_valid_acc	final_valid_loss	final_valid_acc
1.050411	0.715	1.2529	0.698

表 4：ResNeXt 實作結果

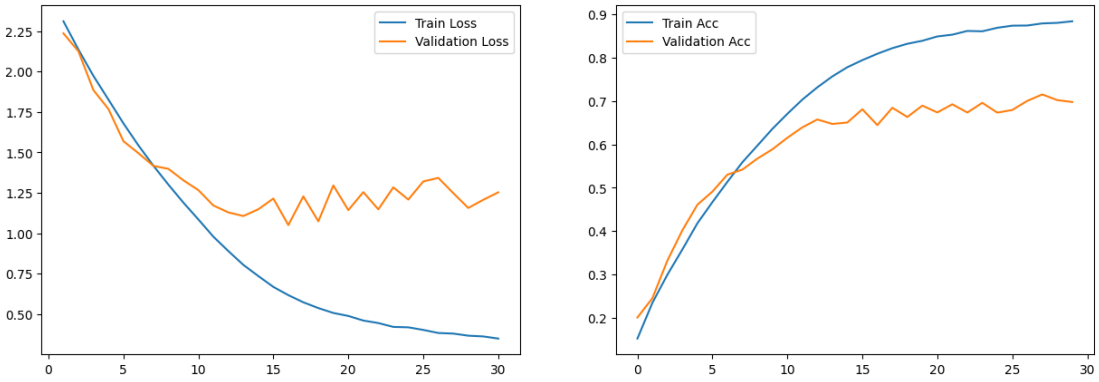

圖 19：ResNeXt 1-30 個 epoch 的 loss(左)，1-30 個 epoch 的 accuracy(右)

從圖 19(左)可見，前 15 個 epoch 的訓練損失與驗證損失皆大幅下降，而後 15 個 epoch 訓練損失下降程度趨緩，而驗證損失則在 1.25 左右上下起幅。從圖 19(右)可見，前 15 個 epoch 訓練準確率與驗證準確率皆大幅提升，而後 15 個 epoch 的訓練準確率上昇速度趨緩，驗證準確率則在 6 成多左右振盪。

6. Xception

(1) model 介紹

Figure 5. The Xception architecture: the data first goes through the entry flow, then through the middle flow which is repeated eight times, and finally through the exit flow. Note that all Convolution and SeparableConvolution layers are followed by batch normalization [7] (not included in the diagram). All SeparableConvolution layers use a depth multiplier of 1 (no depth expansion).

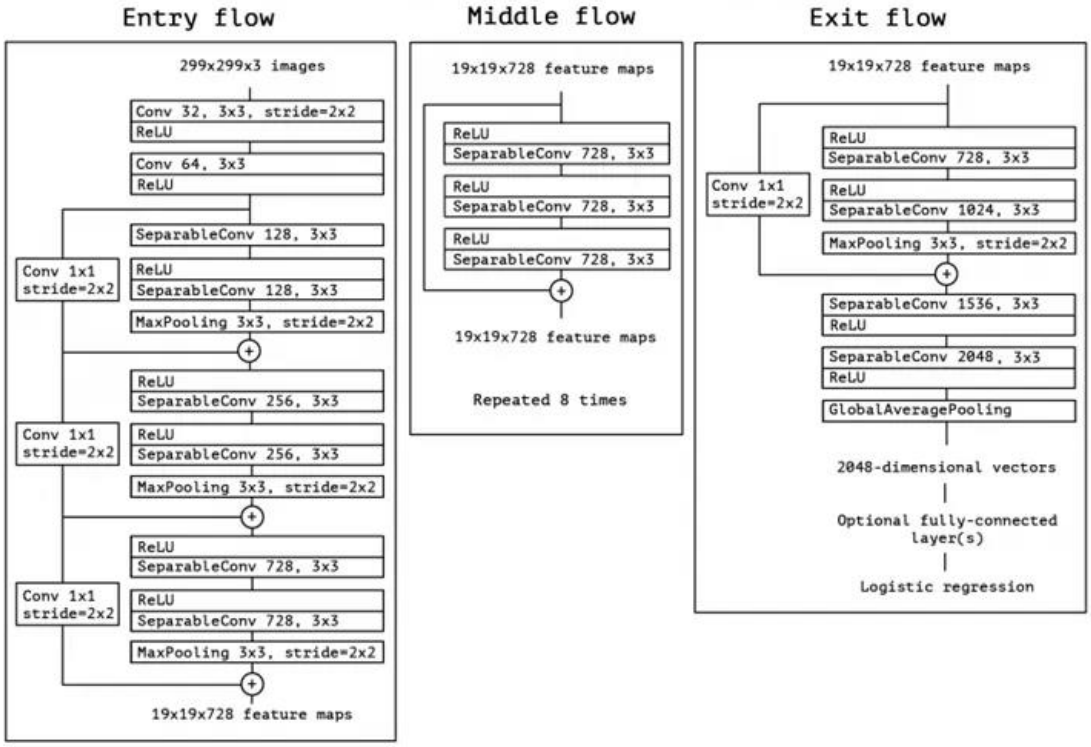

圖 20：Xception 架構

圖 20 為 Xception 的詳細架構。Xception 與 Inception_v3 最大的不同之處，在於採用 depthwise separable convolution。其原理是將原本的捲積拆分成 2 部分，先把卷積弄成 1x1 的卷積，再處理它們。如此一來在不增加網絡複雜的的情況下即可提高效果。

(2) 實作結果

best_train_loss	best_train_acc	final_train_loss	final_train_acc
0.257619	0.914	0.257619	0.914
best_valid_loss	best_valid_acc	final_valid_loss	final_valid_acc
0.868902	0.763	1.122948	0.749

表 5：Xception 實作結果

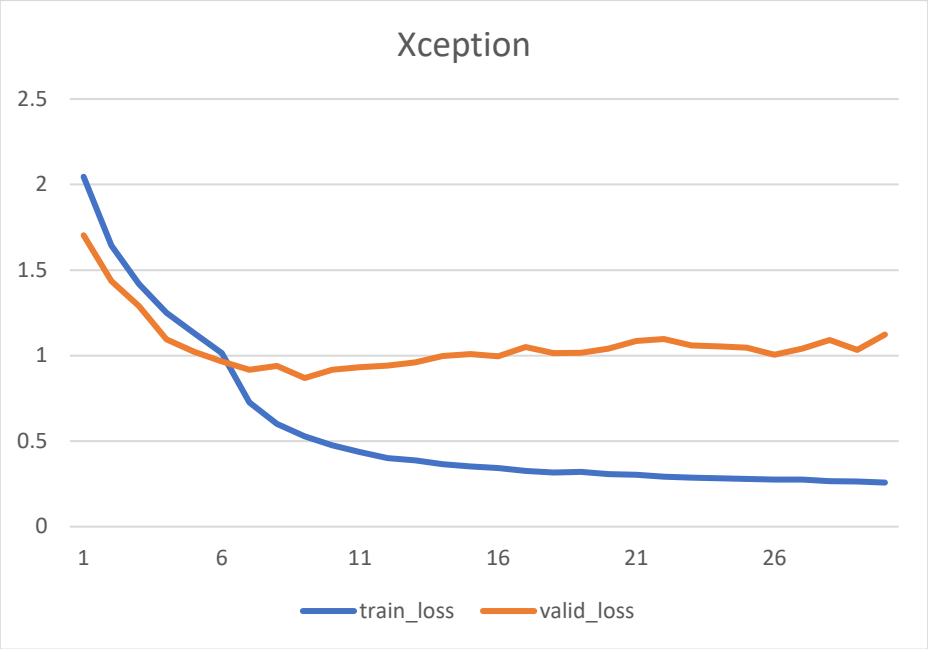

圖 21(a)：Xception 1-30 個 epoch 的 loss

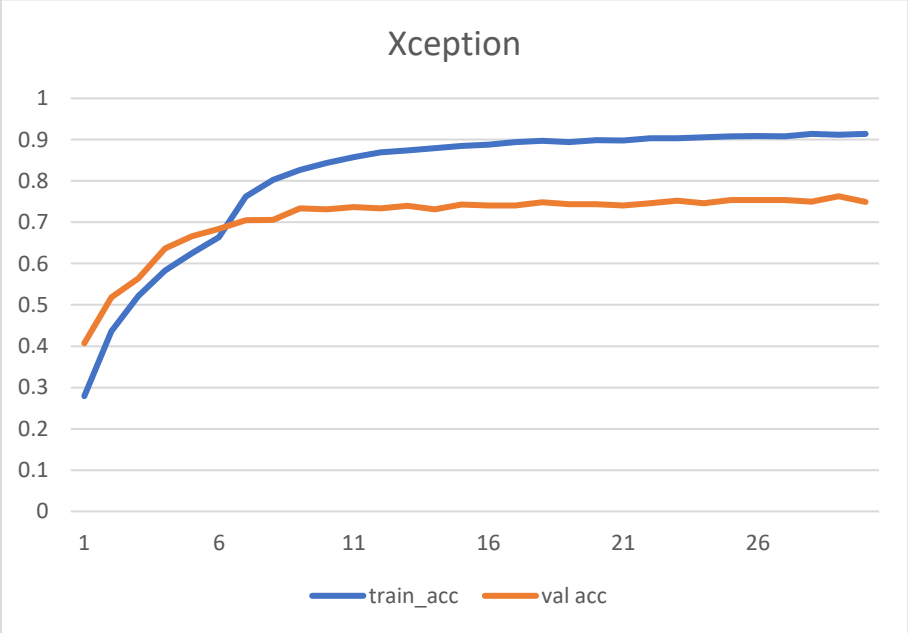

圖 21(b)：Xception 1-30 個 epoch 的 accuracy

從圖 21 可看出，Xception 可以以第 8 個 epoch 做為分界，前 8 個 epoch 的訓練損失與驗證損失都持續下降，訓練準確度與驗證準確度都持續上升；而第 9 到第 30 個 epoch 可觀察到訓練損失、訓練準確度、驗證損失與驗證準確度都已收斂。其中訓練準確度最後可達 9 成。

7. DenseNet161

(1) model 介紹

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv, 3 × 3 conv] × 24	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48	[1 × 1 conv, 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv, 3 × 3 conv] × 16	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

圖 22：DenseNet 架構

圖 22 為 DenseNet 的內部架構。與一般模型最大的不同之處在於，DenseNet 每一級之間都有相連，使得它不會有梯度消失的現象，資訊可貫徹各層。然而這也造成較後面的層數需負擔比較大的計算量。

(2) 實作結果

best_train_loss	best_train_acc	final_train_loss	final_train_acc
0.813833	0.727	0.813833	0.727
best_valid_loss	best_valid_acc	final_valid_loss	final_valid_acc
0.826624	0.738	0.826624	0.738

表 6：DenseNet 實作結果

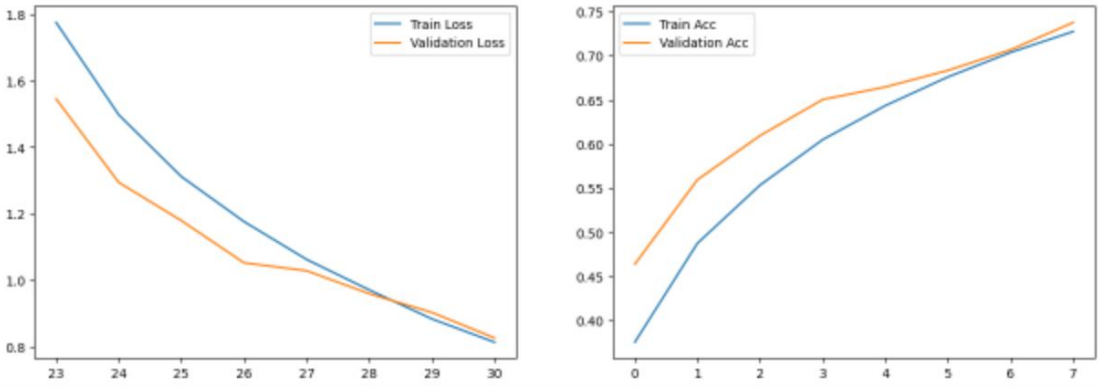

圖 23：DenseNet 23-30 個 epoch 的 loss(左)，23-30 個 epoch 的 accuracy(右)

由於資料遺失，圖 23 只記載第 23 到第 30 個 epoch，圖 23(右)中的 0 到 7 指的是 23 到 30。單看圖 23，可推測訓練損失與驗證損失都還有持續往下的空間，而訓練準確率與驗證準確率也都有上昇的空間。

(三) model 的比較

1. loss

final train loss:

Xception < ResNet34 < VGG16 < VGG19 < ResNeXt50 < ResNet50 < DenseNet161

final valid loss:

Densenet161 < Xception < ResNet34 < VGG16 < ResNet50 < ResNeXt50 < VGG19

2. accuracy

final train acc:

Xception > ResNet34 > VGG16 > VGG19 > ResNeXt50 > ResNet50 > DenseNet161

final valid acc:

Xception > ResNet34 > DenseNet161 > VGG16 > ResNeXt50 > ResNet50 > VGG19

先從 training 角度來看，Xception 以及 resnet34 表現最佳，緊接著在後的是 VGG 系列，而 resnext50、resnet50 及 densenet161 則排在後面，再來從驗證角度來看，densenet161 的表現提升了不少，這代表 densenet161 需要的訓練與其他 model 相比來得少就能有更佳的效果了，而 Xception 以及 resnet34，表現仍十分優秀，VGG19 則跌至最後，這代表 VGG19 需要的訓練量要達到與其他 model 同樣的效果，所需的訓練量相較下來較大，其餘 model 表現維持一樣的趨勢，從 model 間的比較可以發現在實作的情況下，model 的複雜度與準確度沒有一定關係，例如 VGG16 表現比 VGG19 好，ResNet34 表現比 ResNet50 好，我們認為是較複雜的 model 還沒訓練到極限，也就是代表我們沒有讓這幾個 model 能發揮到最好，可能有幾個原因，首先是資料量的不足，這些越複雜的 model，在得獎前都是經過上千萬張圖片的訓練，而我們能給予的圖片量明顯不足，這導致經過少數圖片訓練後，複雜的架構並不會有較好的表現，第二個，訓練程度不足，因為我們時間上的限制，只讓 epoch 為 30 而已，若能搭配更多的圖片以及更多的訓練過程，這些較複雜的 model 才能發揮他們的水準，相較下來，較簡單結構的 model 反而表現較好，如 ResNet34 以及 VGG16，由此可知，在只有較少量的圖片資料的情況時，使用這些較簡單的 model 能有達到較佳的準確率的可能。

四、 測試

(一) 圖片來源

我們每個動物類別各使用了 100 張圖片以進行測試。在這 100 張圖片中，4

種風格分別各佔 25 張圖片。除了原始實景圖片是採用助教所提供的圖片外，其他都是透過第一部分第一小節所提及的爬蟲方法在網路上抓的。這些圖片與訓練圖片資料皆不重複。

(二) Test 與 demo 結果的比較

1. Test 結果

Xception(0.8) > ResNet34(0.791) > DenseNet161(0.763) > ResNeXt50(0.739) > VGG16(0.74) > VGG19(0.716) > ResNet50(0.715)

test 的結果與 valid 的也大致相同。可以發現較新推出的 model 如 Xception、DenseNet、ResNeXt 其表現比 VGG 這種較先推出的 model 表現更好。

2. Demo 結果

model	cartoon	painting	sketch	avg
xception	0.82	0.805	0.9	0.8416
densenet161	0.79	0.77	0.85	0.803
resnet50	0.79	0.725	0.825	0.78
vgg16	0.78	0.7	0.86	0.78
resnet34	0.785	0.685	0.855	0.775
resnext50	0.73	0.705	0.84	0.7583
vgg19	0.725	0.655	0.795	0.725

表 7：各模型之 Demo 結果

由表 7 可以發現 demo 的結果 painting 準確率最低這可能是因為我們找的 painting 的形式與助教給的有點區別。Sketch 的準確率比 cartoon 好，可能是因為 sketch 本身的細節比 cartoon 多，cartoon 的樣式也比 sketch 多元所導致的。

Painting 作為準確率最低的類別同時也是標準差最大的類別，其最高與最低的差距為 sketch 與 cartoon 的 1.5 倍，也點出 painting 的資料收集的問題。

由於訓練資料中，painting 的類別有油畫與水彩 2 種風格，輔以部分網路上抓的圖片，且這些油畫圖片是用程式轉的，使得純油畫的訓練資料佔比不高。相較於卡通圖案全部都是網路上抓卡通圖案，而素描又只是抓圖片輪廓以進行判斷，這可能是 3 種風格中 painting 的準確率最低的可能原因之一。

五、 結論

可以發現 accuracy 大致為指數型上升 valid accuracy 相較於 train accuracy 較

早收斂且到後幾個 epoch 有震盪的現象。valid accuracy 都收斂在 0.7-0.8 之間可能與訓練資料量有關。較新推出的 model 如 Xception 與 DenseNet 的表現的確比較先推出的好。

六、 心得

(一) 劉昱旻

透過這次的期末專題，我對於深度學習以及機器學習的概念有更進一步的了解了，雖然實作中遇到一些困難，包括 colab 資源不夠，網路斷線等等，但很幸運大部分問題都有解決了，希望透過這次的經驗能學會一些解決事情的方法，我也希望這學期學到的技巧在未來不論是在求學或是求職的過程裡都能有所發揮，幫助自己成長茁壯。

(二) 張峻瑋

在這次做業中完整的體驗做出一個圖片分類模型的過程，從最一開始的尋找訓練資料，到去訓練模型與驗證結果。雖然說沒有 Google Colab 我們就沒有辦法完成這項作業，但也因為 Colab 免費版的諸多限制，使得我們必須想辦法在有限的時間（隨時可能會斷線）與空間（有限的 GPU 記憶體）盡量能夠多訓練我們的模型。也很高興透過這次專題，讓我對機器學習的訓練過程有進一步的認識。

(三) 葉俠愷

首先我非常感謝組員給我的支持與幫助。這次的 final project 讓我認識並實作了更多種 CNN 模型算是將課堂所學學以致用。Model 的 training 過程也遇到測資不夠以及免費的 colab 資源不足的問題，但都一一解決掉了。我也在這次 project 中學習到與團隊一起解決問題的經驗。

七、 參考資料

[1] frankie. (2019, January 22). API 到底是什麼？ 用白話文帶你認識. Medium. https://medium.com/codingbar/api-%E5%88%B0%E5%BA%95%E6%98%AF%E4%BB%80%E9%BA%BC-%E7%94%A8%E7%99%BD%E8%A9%B1%E6%96%87%E5%B8%B6%E4%BD%A0%E8%AA%8D%E8%AD%98-95f65a9cfc33

[2] Shiau, J. (2020, January 11). Residual Leaning: 認識 ResNet 與他的冠名後繼者 ResNeXt、ResNeSt. Medium. https://medium.com/ai-blog-tw/deep-learning-residual-leaning-%E8%AA%8D%E8%AD%98resnet%E8%88%87%E4%BB%96%E7%9A%84%E5%86%A0%E5%90%8D%E5%BE%8C%E7%B9%BC%E8%80%85resnext-resnest-6bedf9389ce

[3] 小小将. (2018, May 24). DenseNet：比 ResNet 更优的 CNN 模型. 知乎. https://zhuanlan.zhihu.com/p/37189203

[4] Xie, S., Girshick, R., Doll´ar, P., Tu, Z., & He, K. (2015). Aggregated Residual Transformations for Deep Neural Networks. CVPR 2017. https://doi.org/https://doi.org/10.48550/arXiv.1611.05431

[5] 李謦伊. (2020, November 19). Inception 系列 — Xception. Medium. https://medium.com/ching-i/inception-%E7%B3%BB%E5%88%97-xception-fd2a4a4e7e82

[6] AI 之路. (2017, July 15). Xception 算法详解. CSDN. https://blog.csdn.net/u014380165/article/details/75142710