深度學習

HW6

學號 : B103012002

姓名 : 林凡皓

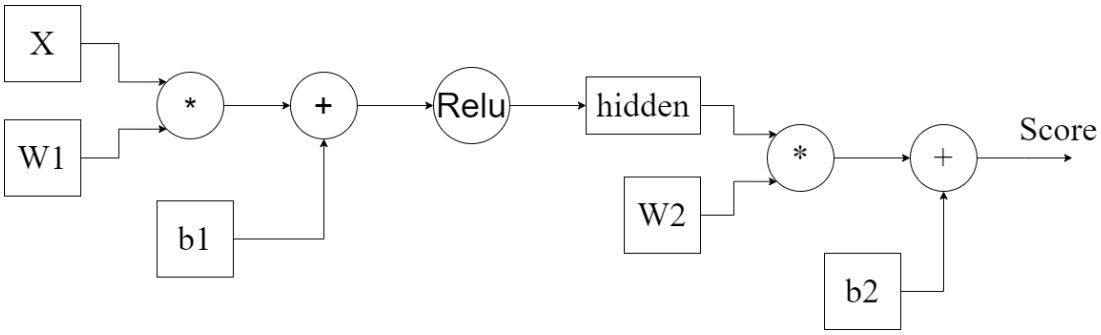
為了方便討論，因此將上次作業內容保留，並在後面加上這次作業內容。本次作業內容從 四、testing our NN on real dataset: CIFAR-10 開始。

一、 Forward pass

- nn_forward_pass :

1. 解題思路 :

   本題主要是計算神經網路對於每個類別計算出來的分數，整個神經網路的 forward path 如下圖



   根據上圖，先利用 torch.clamp( )來計算經過 relu 後的 hidden。接著透過 torch.mm( )來計算最後的 score。

2. 執行結果 :

```
Your scores:
tensor([[ 9.7003e-08, -1.1143e-07, -3.9961e-08],
        [-7.4297e-08,  1.1502e-07,  1.5685e-07],
        [-2.5860e-07,  2.2765e-07,  3.2453e-07],
        [-4.7257e-07,  9.0935e-07,  4.0368e-07],
        [-1.8395e-07,  7.9303e-08,  6.0360e-07]], device='cuda:0')
torch.float32

correct scores:
tensor([[ 9.7003e-08, -1.1143e-07, -3.9961e-08],
        [-7.4297e-08,  1.1502e-07,  1.5685e-07],
        [-2.5860e-07,  2.2765e-07,  3.2453e-07],
        [-4.7257e-07,  9.0935e-07,  4.0368e-07],
        [-1.8395e-07,  7.9303e-08,  6.0360e-07]], device='cuda:0')

Difference between your scores and correct scores: 2.24e-11
```

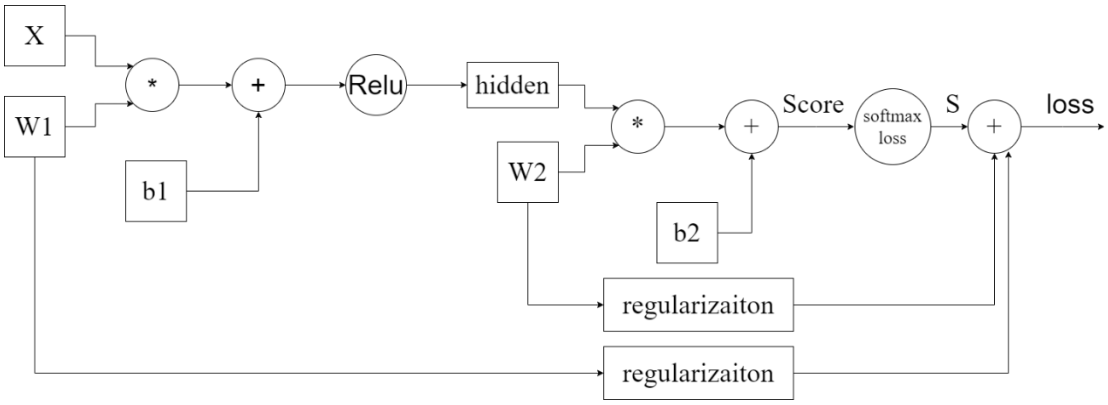
3. 額外討論 :

   經過 relu 後，小於 0 的數字會被歸零，因此使用 torch.clamp( )可以很好的去達到這樣的效果。torch.clamp( )的主要功能為限制 tensor 的範圍，使用的語法為 torch.clamp(input, min, max)，其中 min 為 tensor 的最小值，max 為 tensor 的最大值。

- nn_forward_backward :

1. 解題思路 :

本題主要是利用 forward propagation 的方式來計算 loss。延伸 nn_forward_path 時的神經網路架構，如下圖



根據 softmax loss 的定義，如下圖，來完成一個 function。

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$

在這題，由於網路架構有兩層，因此會有兩個權重值 tensor 分別為 W1 和 W2，因此 regularization term 會有兩項分別為 R(W1)和 R(W2)。

2. 執行結果 :

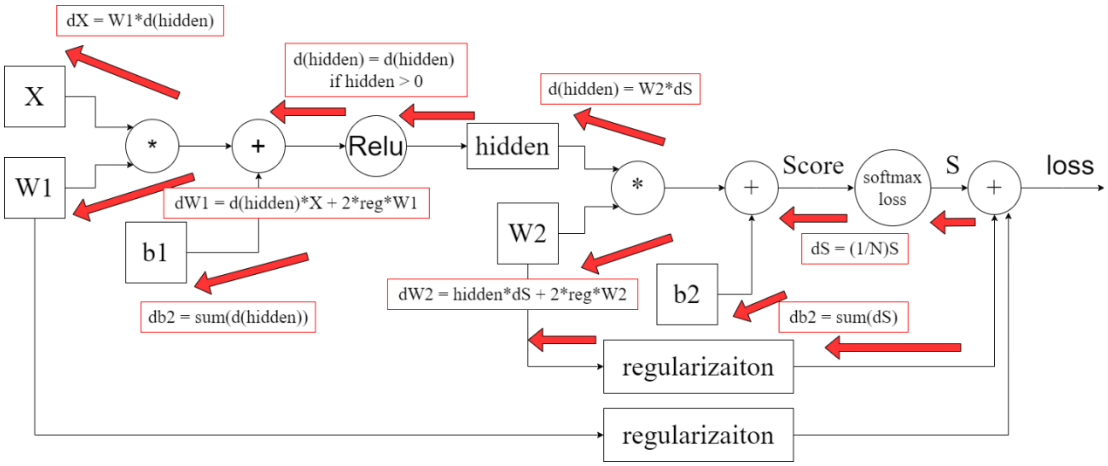
```
Your loss:  1.0986121892929077
Correct loss:  1.0986121892929077
Difference: 0.0000e+00
```

二、 Backward pass

- nn_forward_backward :

1. 解題思路 :

使用 computation graph 來實作 back propagation，computational graph 與計算之 gradient 如下



利用此計算結果加上矩陣相乘的規則(前面矩陣的 column 數目要等於後面矩陣的 row 數目)來判斷哪個矩陣要放前面，或是誰需要做轉置，來完成 back propagation。

2. 執行結果 ：

```
W2 max relative error: 1.261262e-06
b2 max relative error: 3.122771e-09
b1 max relative error: 8.239303e-06
W1 max relative error: 1.441750e-06
```

三、 Train the network

- nn_train :

1. 解題思路 ：

   Trianing 的主要在做的事就是更新權重值。更新權重的方式是 gradient descent，也就是往負的 gradient 方向去走。因此，寫法就是 w = w – learning_rate * dw。

2. 執行結果 ：

```
Final training loss:  0.5211756229400635
```

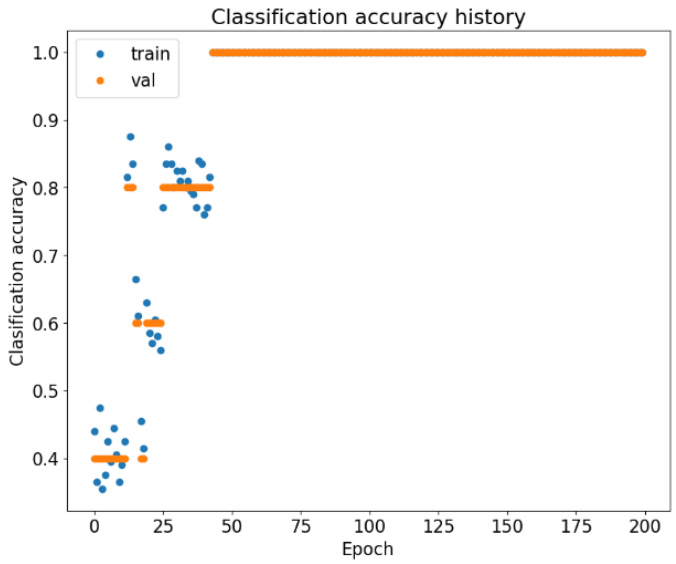
由上圖可以看出，因為 gradient descent 的影響，training loss 會隨著訓練的次數增加而下降，直到最後趨近於飽和，代表說模型的極限已到。

- nn_predict :

1. 解題思路 :

   nn_predict 是用來當作 nn_train 的 pred_func 參數。將我們選擇好的參數、training data 送進我們選擇的計算分數的 function 中，接著在計算的分數中找到最大值的 index 即為預測結果。

2. 執行結果 :



由上圖可以看出準確度隨訓練次數增加而上升，最後趨近飽和，

趨勢有點類似於 loss。

3. 額外討論 ：

為甚麼 loss function 可以用來計算分數?

關於這個問題，先看到 nn_predict 傳入的 loss_func。這裡的 loss_func 使用的是先前定義好的 nn_forward_backward。在 nn_forward_backward 中可以看到有一段 code 是使用 nn_forward_pass 來計算分數，並在 y = None 時，nn_forward_backward 會回傳 score 而不是 loss 和 grad。接著看到定義 nn_pridect 的地方。在定義 nn_predict 的時候，我們並沒有傳入 y，也就是說 y 是 None，因此 nn_forward_backward，就會變成是一個計算分數的函數。

四、 Testing our NN on a real dataset: CIFAR-10

- Train a network :

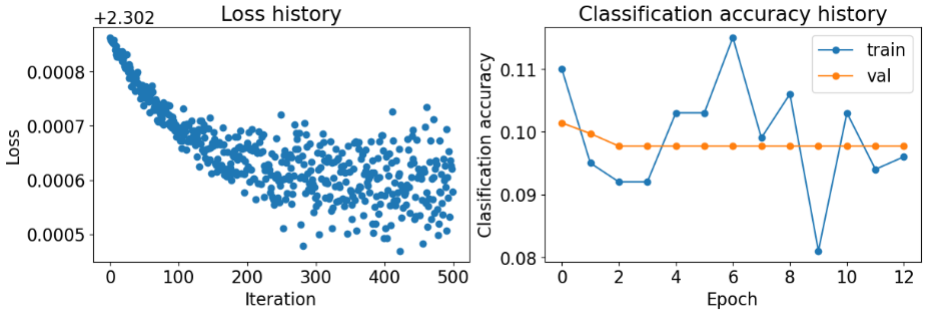
根據之前定義的 TwoLayerNet 來對 CIFAR-10 做訓練。這裡建立一個有一層 hidden layer (size = 36)的 NN，接著對它做訓練，訓練的超參數如下 ：num_iters = 500、batch_size = 1000、learning rate = 1e-2、learning_rate_decay = 0.95、reg = 0.25。訓練結果如下

```
iteration 0 / 500: loss 2.302862
iteration 100 / 500: loss 2.302695
iteration 200 / 500: loss 2.302668
iteration 300 / 500: loss 2.302551
iteration 400 / 500: loss 2.302571
Validation accuracy: 9.77%
```

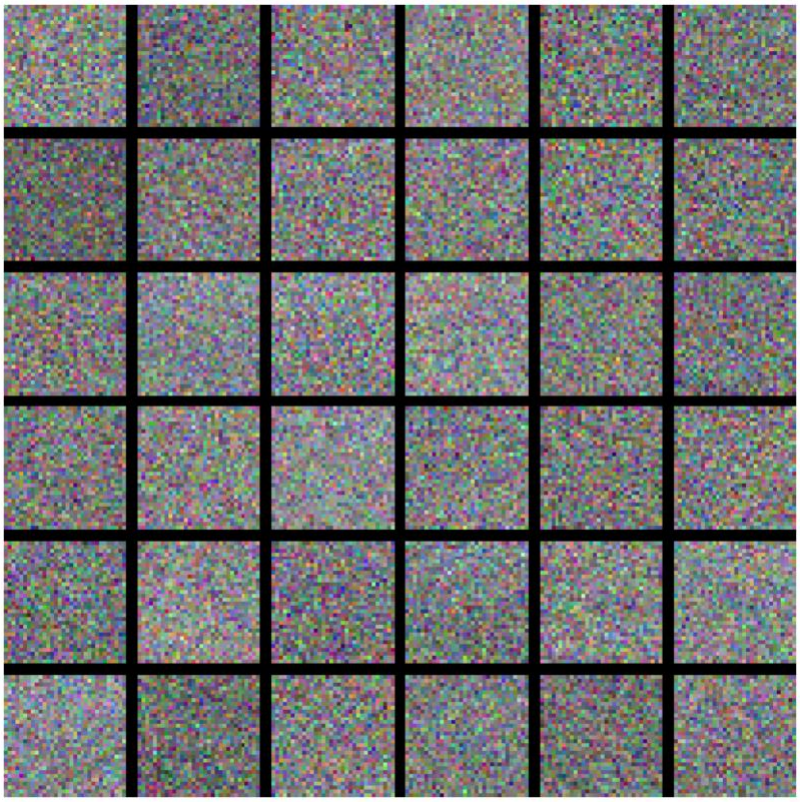
由結果可以看出，訓練基本上無效。接著下來討論原因。

- Debug and training :

為了方便觀察，將訓練過程視覺化，視覺化結果如下



將權重也視覺化，結果如下



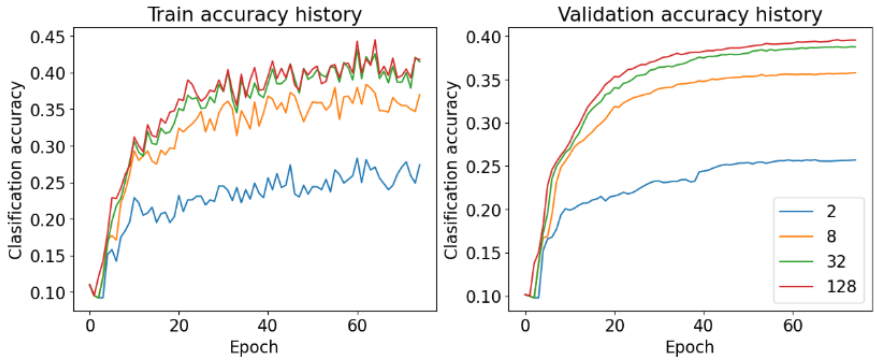
由訓練過程可以看到，validation accuracy 和 training accuracy 基本上沒有提升，代表說現在模型處於 underfitting 的情況，因此我們嘗試一些方法，像是增加 hidden layer size、增加訓練次數、調整 learning rate 等。

- What's wrong

  開始嘗試各種分析來找到目前模型的問題所在。

  1. Hidden layer size :

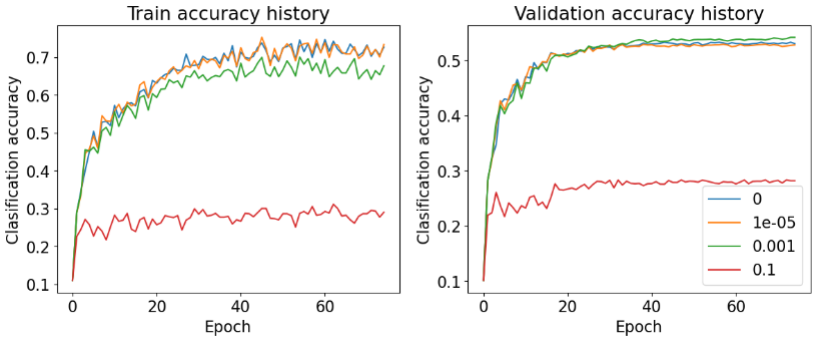
     首先嘗試不同大小的 hidden layer。當 hidden layer 太小，模型會沒辦法很好的擬和 training data，造成 underfitting，因此嘗試不同的 hidden layer size 來觀察 hidden layer size 對於訓練的影響。結果如下



由結果可以看出，當 hidden layer size 提升時，模型能夠有更好的 validation accuracy，這主要是因為更大的 hidden 會有更多的參數可以做調整來更好的去擬合訓練資料。因此先前的模型訓練會 underfit 的原因可能為 hidden layer size 太小的關係。

2. Regularization strength :

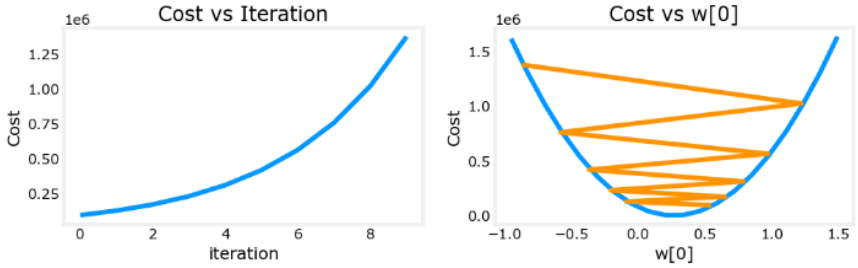
接著探討 regularization strength 對訓練的影響。當 regularization strength 太大時，也會造成模型 underfitting，因此也有必要觀察不同 regularization strength 對於訓練的影響。結果如下圖



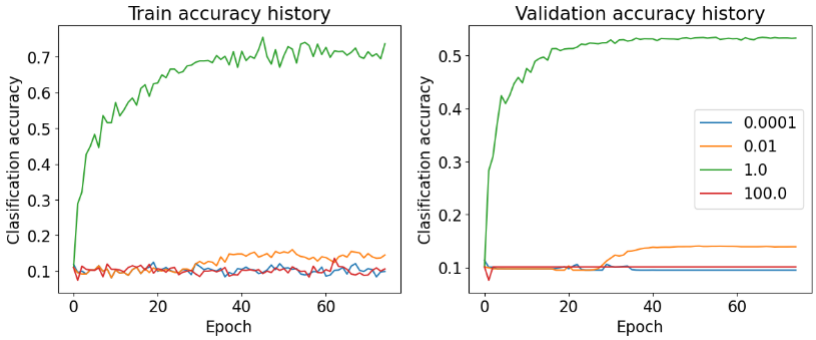
由結果可以看到，regularization strength 要小於 0.001 時，模型 underfitting 的情況才比較不會那麼嚴重。因此剛剛訓練沒有進展也有一部分原因是 regularization strength 太小。

3. Learning rate :

Learning rate 對於訓練來說也有很大的影響，learning rate 太大會成 gradient descent 的過程來回彈跳，如下圖



由上圖可以看出，learning rate 太大不但會造成 loss 無法順利下降，甚至會造成 loss 有不降反升的情形。Learning rate 太小的話會造成 loss 下降過程很慢，或是訓練掉到 local minimum 等現象。嘗試改變 learning rate，結果如下



由結果可以看出，最佳的 learning rate 為 1。

4. 接著使用剛剛測試出來的參數再次訓練。使用參數如下：num_iter = 3000、hidden layer size = 128、regularization strength = 0、learning rate = 1。結果如下

```
iteration 0 / 3000: loss 2.302585
iteration 100 / 3000: loss 1.745387
iteration 200 / 3000: loss 1.492672
iteration 300 / 3000: loss 1.617622
iteration 400 / 3000: loss 1.313138
iteration 500 / 3000: loss 1.320080
iteration 600 / 3000: loss 1.236202
iteration 700 / 3000: loss 1.225645
iteration 800 / 3000: loss 1.138468
iteration 900 / 3000: loss 1.127046
iteration 1000 / 3000: loss 1.081245
iteration 1100 / 3000: loss 1.041731
iteration 1200 / 3000: loss 0.994569
iteration 1300 / 3000: loss 1.020123
iteration 1400 / 3000: loss 1.023700
iteration 1500 / 3000: loss 0.966685
iteration 1600 / 3000: loss 0.913892
iteration 1700 / 3000: loss 0.967342
iteration 1800 / 3000: loss 0.855454
iteration 1900 / 3000: loss 0.886901
iteration 2000 / 3000: loss 0.906134
iteration 2100 / 3000: loss 0.892426
iteration 2200 / 3000: loss 0.850733
iteration 2300 / 3000: loss 0.884139
iteration 2400 / 3000: loss 0.828732
iteration 2500 / 3000: loss 0.866267
iteration 2600 / 3000: loss 0.866648
iteration 2700 / 3000: loss 0.891294
iteration 2800 / 3000: loss 0.834132
iteration 2900 / 3000: loss 0.830277
Validation accuracy: 53.18%
```

可以看到經過一些參數的調整，模型準確度從原本的 9.77 %提升到 53.18 %。

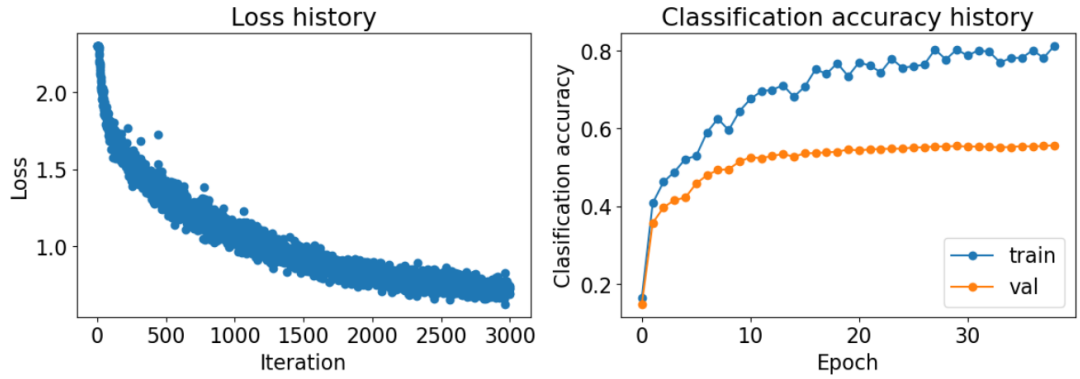
- Tune your hyper parameters :

  接著使用 iteration 的方式來尋找最佳的參數組合。

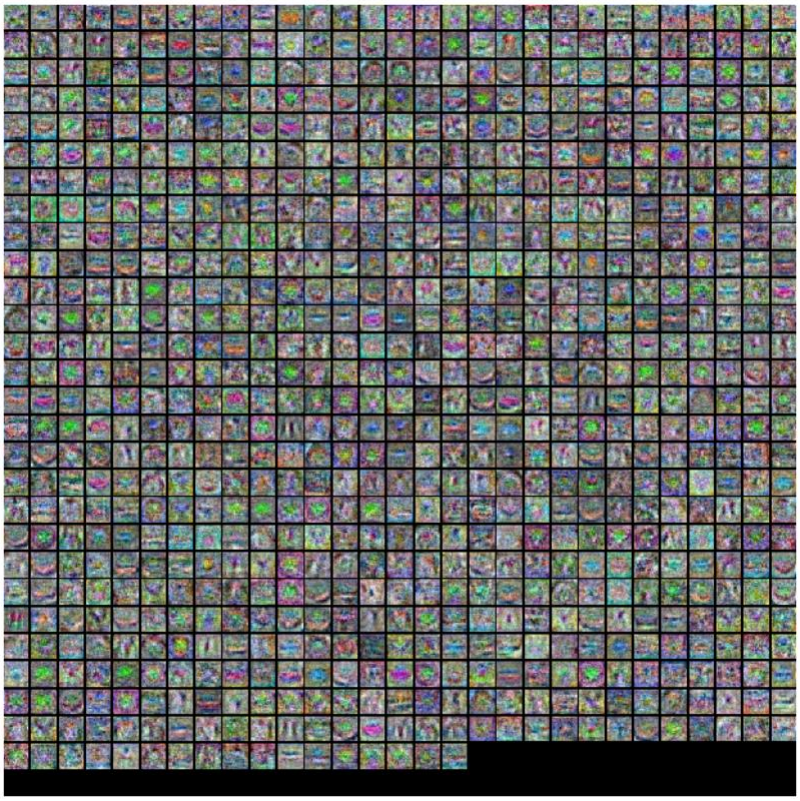
  這裡會使用到 find_best_net 這個 function。這個 function 的實現主要就是從 get_param_set 中取得 learning rate、hidden_sizes、regularization strengths、learning rate decays，並利用 for loop 分別去迭代這些參數，並在訓練完成後判斷目前的模型的 validation accuracy 是否有高於先前最佳的模型，如果有就更新最佳模型以及最佳的 validation accuracy。

  執行結果如下

```
lr:0.8 hs:800 reg:1e-07 lrd:0.88
lr:0.8 hs:800 reg:1e-07 lrd:0.9
lr:0.8 hs:800 reg:1e-07 lrd:0.91
lr:0.8 hs:800 reg:1e-07 lrd:0.92
lr:0.8 hs:850 reg:1e-07 lrd:0.9
lr:0.8 hs:900 reg:1e-07 lrd:0.92
0.5570999979972839
```



當 learning rate = 0.8、hidden layer size = 900、regularization strength = 1e-7、learning rate decay = 0.92 時，得到最佳的 validation accuracy = 55.71 %。接著將此模型的權重視覺化，結果如下



最後將此模型對測試資料做測試，結果如下

```
Test accuracy: 55.12%
```

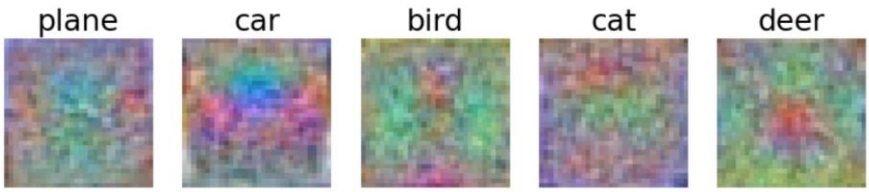
最終 testing accuracy 為 55.12 %。

五、 額外討論與嘗試

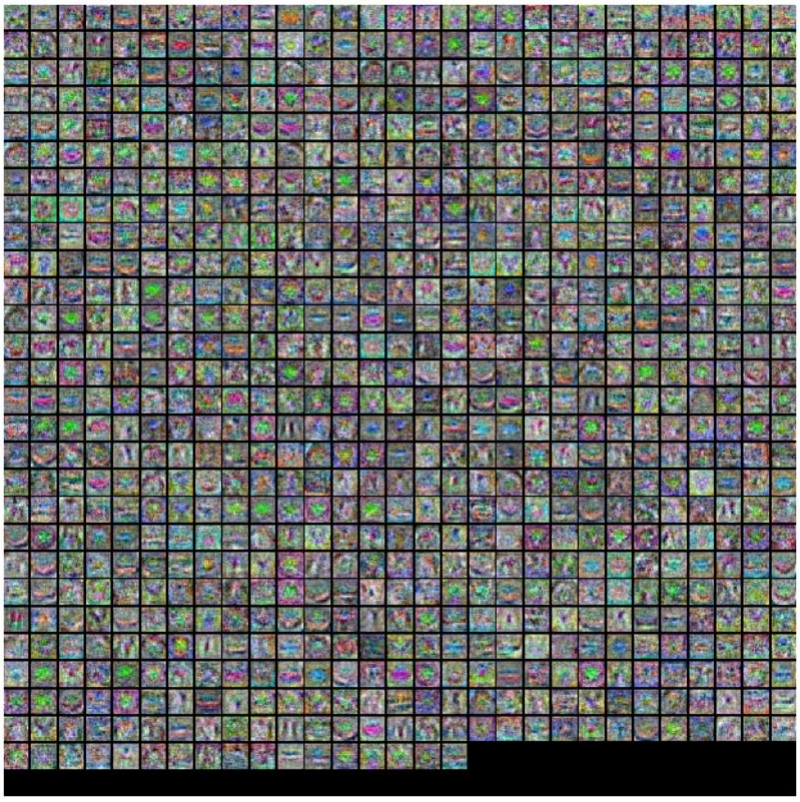
- NN v.s. SVM :

  由這次作業與前幾次作業相比可以發現，這於同樣的資料及來說，一個只有一層 hidden layer 的 NN 的準確度可以來到 54.19 %，而 SVM 的準確度大約是 40 %左右。之所以一個非常簡單的神經網路的準確度可以高於 SVM 主要是因為它們的權重與所產生的模板的差異。

  下圖為 SVM 所產生的模板



對於一個類別來說，SVM 只會產生一個模板，這會使 SVM 無法應對太多種情況。下圖為 NN 的模板



由此圖可以發現到，對於一個類別，NN 會產生出多個對應的模板，這不只使 NN 可以應對更多種情況，也能使 NN 有更佳的判斷準確度。

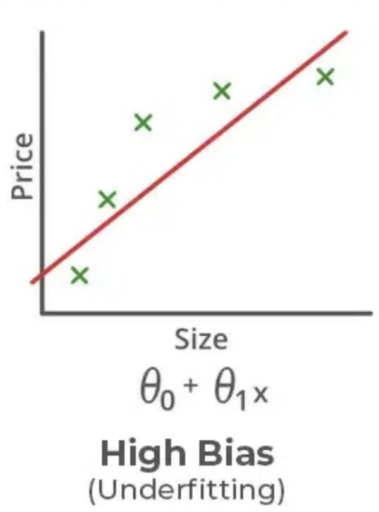
如果去疊加更多層的 NN，甚至是使用 convolutional layer，神經網路

會產生出更多的模板，甚至是可以更好的抓取關鍵特徵，這又會再讓神經網路的準確度上升。但是就如同教授上課所講，要訓練一個好的神經網路會需要大量的資料，如果資料不足，神經網路也沒有辦有效的訓練權重值，導致神經網路的功能無法完全被發揮之外，還要花費一堆時間訓練模型。因此，雖然說 SVM 或是 KNN 之類的架構的準確度無法與神經網路做比較，但是在一些情況下，SVM 與 KNN 也會是不錯的選擇。

- Overfitting and underfitting :

  Overfitting 和 underfitting 是訓練模型時很重要的兩個指標，在這次訓練過程中，也有發生 underfitting 和 overfitting 的現象，因此拿出來做討論。

  1. Underfitting : 也有人將其稱為 high bias。Underfitting 代表說目前的模型，即便是對訓練資料來說，也無法擬合的太好，如下圖所示



  underfitting 的特色為 training accuracy 和 validation accuracy 都會很低。

  對於這個現象，增加訓練資料是沒有用的，畢竟連目前的資料都無法有效擬合，更何況是再加入其他資料。對於 underfitting，解決方法如下

  (1) 增加其他特徵 : 如果可以增加其他特徵讓模型去抓取，模型會有更好的機會去根據這些特徵找出資料中的規律，並更加有效的去擬合資料。

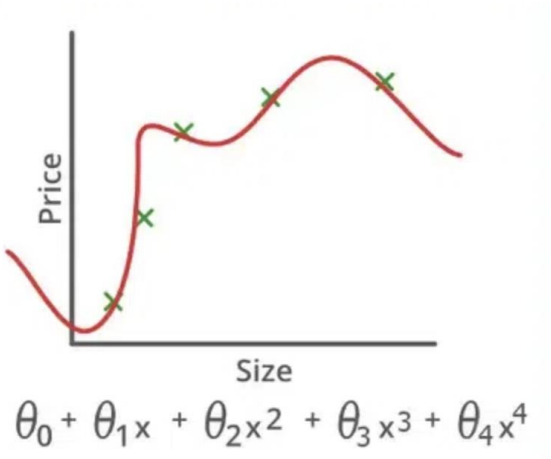
  (2) 讓模型更加複雜 : 不論是在 linear classifier 中加入一些高次多項式的項式，或是堆疊更深的神經網路，都會很有效的解決掉

underfitting 的問題，但是成本就是訓練與預測的時間與計算量都會增加。

(3) 增加訓練次數。

(4) 降低 regularization strength。

2. Overfitting： 也有人稱為 high variance。Overfitting 代表說模型已經過度擬合訓練資料了，換句話說，模型已經成為專門為該訓練資料所設計的模型了，對於沒看過的資料有不好的泛化能力。此現象如下圖所示



Overfitting 的特色為 training accuracy 很高，但是 validation accuracy 沒那麼高，通常 testing accuracy 和 validation accuracy 會差不少。對於 overfitting，解決方法如下

(1) 增加訓練資料 ： 假如訓練資瞭可以包含所有可能性，那就沒有 overfitting 的說法了，畢竟沒有情況是模型沒見過的，因此增加訓練資料可以幫助解決 overfitting 的問題。

(2) 在訓練模型時使用 early stopping 的機制，讓模型在過度擬合時及時停止訓練。

(3) 加大 regularization strength。

(4) 對於神經網路而言，可以使用 dropout 或是避免使用全連接層來避免 overfitting 的情況。

- Transfer learning

1. Transfer learning 方法與分類

底下的 source 定義為已經存在的知識，target 定義為欲訓練的領域。

(1) Based on instance or said sample： 通過權重的分配，分別作用

到 source 和 target。例如，在 source 中有一個樣本和 target 中的樣本很像，那我們可以加大刺樣本對應到的權重。

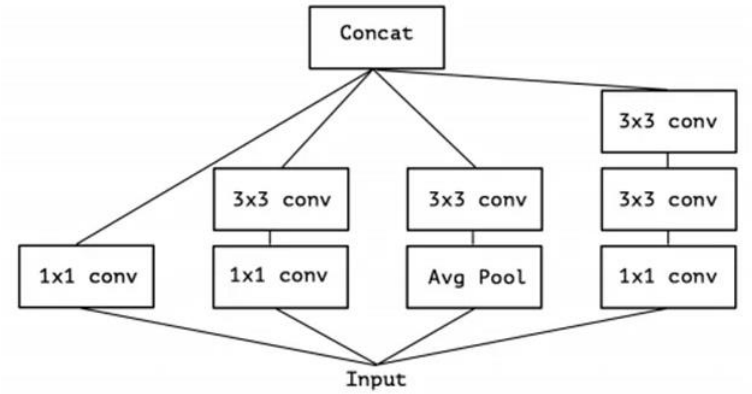
(2) Based on features：將 source 和 target 中的特徵變換同一個空間。在 source 和 target 上的 feature 差異很大，我們可以透過將他們的特徵變換到同一個空間，如此一來我們就可以更加方便研究 source 和 target 的相關內容和性質。

(3) Based on model：將 source 的參數與 target 共享。將 pre-trained 的 model 修改一些 layer 的參數。這次嘗試做的也是 based on model 的 transfer learning。

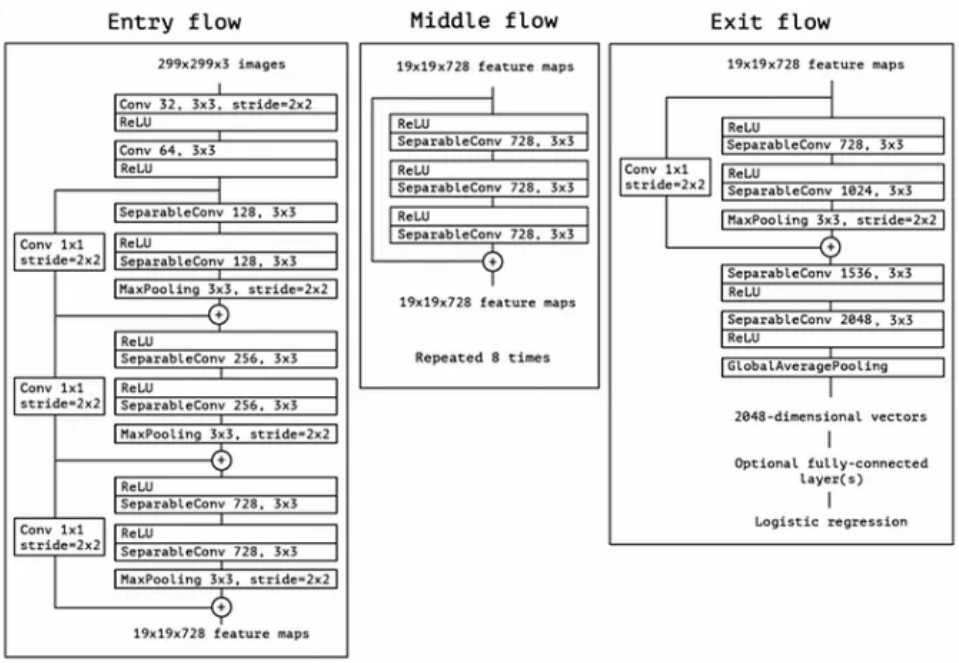
2. 嘗試

這次選擇使用兩種模型，分別為 Xception 與 ResNet50，來做 transfer learning。先對兩個模型作介紹。

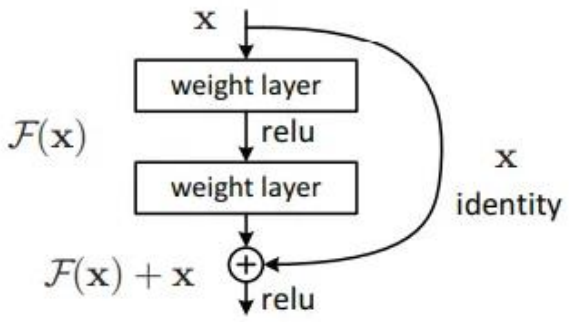
(1) Xception：此模型是根據 InceptionV3 做改進，採用 depthwise separable convolution 來代替原本的 Inception module，並加入了 ResNet 的 Residual Learning 方法。InceptionV3 的主要概念是讓模型自己去找最好的特徵提取方式，架構如下



所謂的 depthwise separable convolution 可以分成兩個步驟，第一步為 depthwise convolution，即對各個維度通道各自做卷積運算。第二步是 pointwise convoltion，即對不同維度通道進行融合。Xception 架構如下



(2) ResNet50 : ResNet 在 VGG 的基礎上，新增了 skip connection。此動作不但部會新增任何參數，還能夠解決梯度消失的問題。ResNet 中最關鍵的就是 residual learning。Residual learning 的是意圖如下圖



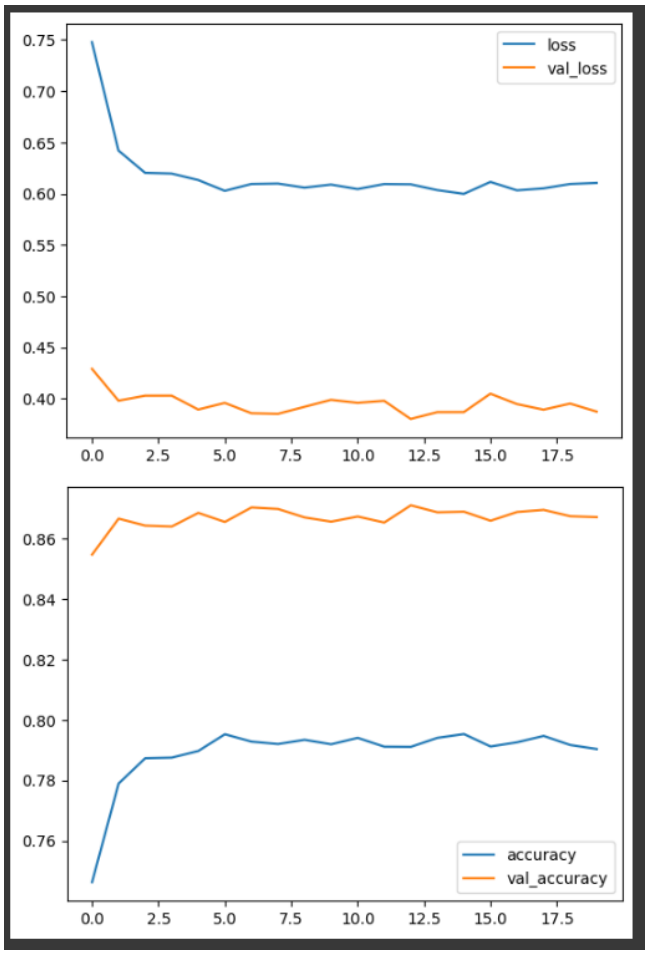
上圖的概念就是 residual mapping。在 ResNet 中，將目標 H(x)換成 F(x) = H(x) – x，即 residual。優化 residual 會比直接學習恆等 mapping (x → H(x))來的容易，有了 residual mapping 之後，模型只需要想辦法讓每個 residual block 學習出來的 residual 變成 0 (等價於恆等映射 H(x) = x)，然後移項就可以得到 H(x) = F(x) + x，即我們原本要的 mapping 函數。

接著展示 transfer lerning 後的結果，首先是 Xception，網路架構如下圖，

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 32, 32, 3)]       0

 lambda (Lambda)             (None, 224, 224, 3)       0

 sequential (Sequential)     (None, 224, 224, 3)       0

 tf.math.truediv (TFOpLambd  (None, 224, 224, 3)       0
 a)

 tf.math.subtract (TFOpLamb  (None, 224, 224, 3)       0
 da)

 xception (Functional)       (None, 7, 7, 2048)        20861480

 global_average_pooling2d (  (None, 2048)              0
 GlobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 10)                20490

=================================================================
Total params: 20881970 (79.66 MB)
Trainable params: 20490 (80.04 KB)
Non-trainable params: 20861480 (79.58 MB)
_________________________________________________________________
```

訓練過程如下

```
Epoch 1/20
1563/1563 [==============================] - 248s 152ms/step - loss: 0.7477 - accuracy: 0.7463 - val_loss: 0.4291 - val_accuracy: 0.8546
Epoch 2/20
1563/1563 [==============================] - 234s 149ms/step - loss: 0.6420 - accuracy: 0.7789 - val_loss: 0.3979 - val_accuracy: 0.8665
Epoch 3/20
1563/1563 [==============================] - 237s 151ms/step - loss: 0.6202 - accuracy: 0.7873 - val_loss: 0.4028 - val_accuracy: 0.8642
Epoch 4/20
1563/1563 [==============================] - 237s 151ms/step - loss: 0.6195 - accuracy: 0.7875 - val_loss: 0.4029 - val_accuracy: 0.8639
Epoch 5/20
1563/1563 [==============================] - 234s 149ms/step - loss: 0.6132 - accuracy: 0.7897 - val_loss: 0.3893 - val_accuracy: 0.8684
Epoch 6/20
1563/1563 [==============================] - 237s 151ms/step - loss: 0.6027 - accuracy: 0.7952 - val_loss: 0.3958 - val_accuracy: 0.8654
Epoch 7/20
1563/1563 [==============================] - 236s 151ms/step - loss: 0.6094 - accuracy: 0.7928 - val_loss: 0.3857 - val_accuracy: 0.8702
Epoch 8/20
1563/1563 [==============================] - 234s 149ms/step - loss: 0.6098 - accuracy: 0.7920 - val_loss: 0.3851 - val_accuracy: 0.8697
Epoch 9/20
1563/1563 [==============================] - 233s 149ms/step - loss: 0.6058 - accuracy: 0.7934 - val_loss: 0.3920 - val_accuracy: 0.8669
Epoch 10/20
1563/1563 [==============================] - 236s 151ms/step - loss: 0.6088 - accuracy: 0.7919 - val_loss: 0.3988 - val_accuracy: 0.8655
Epoch 11/20
1563/1563 [==============================] - 237s 151ms/step - loss: 0.6044 - accuracy: 0.7940 - val_loss: 0.3958 - val_accuracy: 0.8672
Epoch 12/20
1563/1563 [==============================] - 233s 149ms/step - loss: 0.6093 - accuracy: 0.7911 - val_loss: 0.3978 - val_accuracy: 0.8652
Epoch 13/20
1563/1563 [==============================] - 233s 149ms/step - loss: 0.6090 - accuracy: 0.7910 - val_loss: 0.3801 - val_accuracy: 0.8709
Epoch 14/20
1563/1563 [==============================] - 233s 149ms/step - loss: 0.6034 - accuracy: 0.7940 - val_loss: 0.3868 - val_accuracy: 0.8686
Epoch 15/20
1563/1563 [==============================] - 237s 151ms/step - loss: 0.5997 - accuracy: 0.7953 - val_loss: 0.3868 - val_accuracy: 0.8688
Epoch 16/20
1563/1563 [==============================] - 236s 151ms/step - loss: 0.6114 - accuracy: 0.7912 - val_loss: 0.4048 - val_accuracy: 0.8658
Epoch 17/20
1563/1563 [==============================] - 236s 151ms/step - loss: 0.6032 - accuracy: 0.7926 - val_loss: 0.3947 - val_accuracy: 0.8687
Epoch 18/20
1563/1563 [==============================] - 237s 151ms/step - loss: 0.6051 - accuracy: 0.7947 - val_loss: 0.3892 - val_accuracy: 0.8694
Epoch 19/20
1563/1563 [==============================] - 237s 151ms/step - loss: 0.6094 - accuracy: 0.7917 - val_loss: 0.3951 - val_accuracy: 0.8673
Epoch 20/20
1563/1563 [==============================] - 234s 150ms/step - loss: 0.6104 - accuracy: 0.7903 - val_loss: 0.3873 - val_accuracy: 0.8670
```



可以看到，validation accuracy 可以來到 86.7 %，比起這次作業的 NN 高出很多。

接著嘗試做 ResNet50 的 transfer learning。網路架構如下

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 resnet50 (Functional)       (None, 1, 1, 2048)        23587712

 flatten_2 (Flatten)         (None, 2048)              0

 dense_10 (Dense)            (None, 1024)              2098176

 dense_11 (Dense)            (None, 512)               524800

 dense_12 (Dense)            (None, 256)               131328

 dense_13 (Dense)            (None, 128)               32896

 dense_14 (Dense)            (None, 10)                1290

=================================================================
Total params: 26376202 (100.62 MB)
Trainable params: 26323082 (100.41 MB)
Non-trainable params: 53120 (207.50 KB)
```

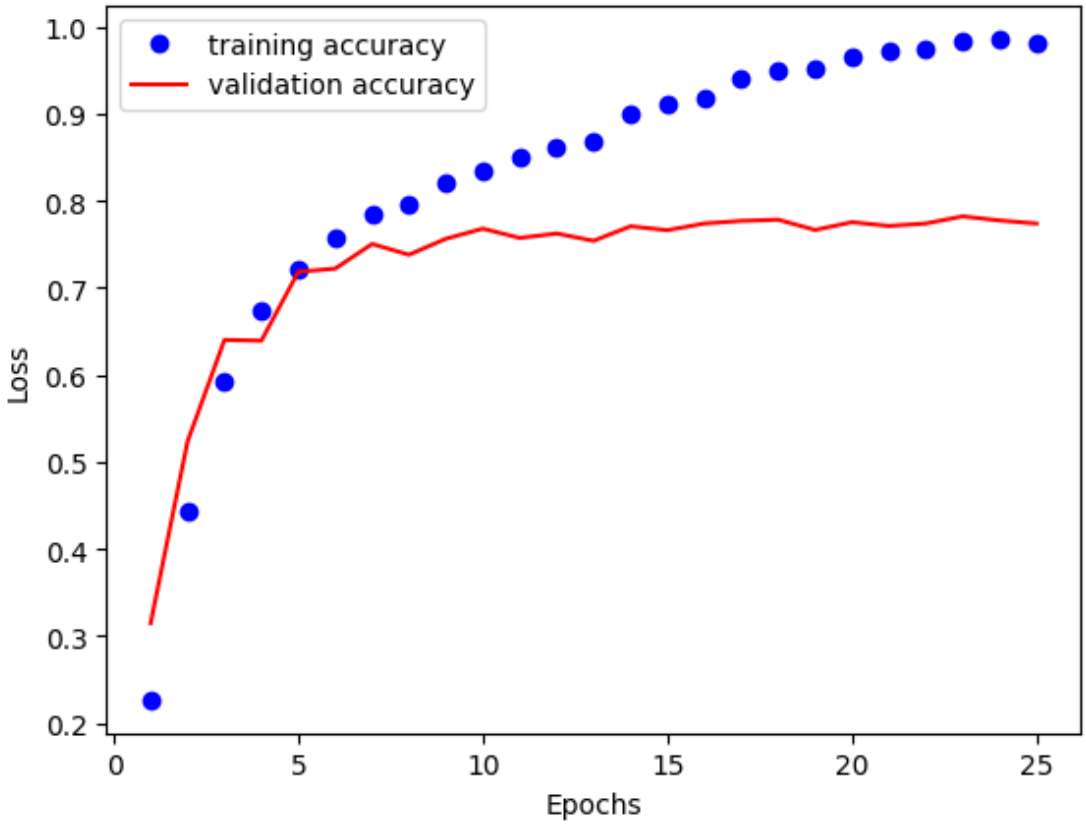
訓練過程如下

```
Epoch 40: val_loss did not improve from 1.12483
196/196 [==============================] - 7s 34ms/step - loss: 0.9390 - accuracy: 0.7483 - val_loss: 1.1477 - val_accuracy: 0.6789 - lr: 3.2768e-04
Epoch 41/100
195/196 [============================>.] - ETA: 0s - loss: 0.9199 - accuracy: 0.7570
Epoch 41: val_loss did not improve from 1.12483
196/196 [==============================] - 6s 33ms/step - loss: 0.9199 - accuracy: 0.7569 - val_loss: 1.1477 - val_accuracy: 0.6808 - lr: 2.6214e-04
Epoch 42/100
195/196 [============================>.] - ETA: 0s - loss: 0.9173 - accuracy: 0.7573
Epoch 42: val_loss did not improve from 1.12483
196/196 [==============================] - 7s 33ms/step - loss: 0.9174 - accuracy: 0.7572 - val_loss: 1.1449 - val_accuracy: 0.6829 - lr: 2.6214e-04
Epoch 43/100
195/196 [============================>.] - ETA: 0s - loss: 0.9131 - accuracy: 0.7603
Epoch 43: ReduceLROnPlateau reducing learning rate to 0.00020971521735191345.

Epoch 43: val_loss did not improve from 1.12483
196/196 [==============================] - 7s 35ms/step - loss: 0.9131 - accuracy: 0.7603 - val_loss: 1.1535 - val_accuracy: 0.6803 - lr: 2.6214e-04
Epoch 44/100
195/196 [============================>.] - ETA: 0s - loss: 0.8957 - accuracy: 0.7685
Epoch 44: val_loss did not improve from 1.12483
196/196 [==============================] - 6s 32ms/step - loss: 0.8959 - accuracy: 0.7683 - val_loss: 1.1538 - val_accuracy: 0.6801 - lr: 2.0972e-04
Epoch 45/100
195/196 [============================>.] - ETA: 0s - loss: 0.8938 - accuracy: 0.7697
Epoch 45: val_loss did not improve from 1.12483
196/196 [==============================] - 6s 33ms/step - loss: 0.8936 - accuracy: 0.7697 - val_loss: 1.1579 - val_accuracy: 0.6849 - lr: 2.0972e-04
Epoch 46/100
195/196 [============================>.] - ETA: 0s - loss: 0.8892 - accuracy: 0.7707Restoring model weights from the end of the best epoch: 31.

Epoch 46: ReduceLROnPlateau reducing learning rate to 0.00016777217388153076.

Epoch 46: val_loss did not improve from 1.12483
196/196 [==============================] - 7s 34ms/step - loss: 0.8893 - accuracy: 0.7707 - val_loss: 1.1623 - val_accuracy: 0.6856 - lr: 2.0972e-04
Epoch 46: early stopping
```

由上圖可以看出，訓練到最後 validation accuracy 為 68.5 %，而這時候不只是 validation loss 無法有效提升，連 training loss 都無法順利下降。這顯然是不合理的，情況有兩種可能，第一就是發生 underfitting 的問題，但是我認為不是這種可能，因為我有自己設置 callbacks 來防止 overfitting 的情況發生，因此訓練並非是迭代完成而停止，而是因為 loss 無法再順利下降而停止。第二種可能為原本的 ResNet50 的訓練資料與 CIFAR-10 相差很遠，這會造成說這組 ResNet 的參數是無法有效地應用到 CIFAR-10 的資料及上。為了解決這個情況，我讓原本的 ResNet 模型的參數變得可以做訓練，修改後結果如下

```
Epoch 19: val_loss did not improve from 0.76937
391/391 [==============================] - 28s 71ms/step - loss: 0.1636 - accuracy: 0.9519 - val_loss: 0.9921 - val_accuracy: 0.7665 - lr: 6.4000e-04
Epoch 20/100
391/391 [==============================] - ETA: 0s - loss: 0.1231 - accuracy: 0.9658
Epoch 20: val_loss did not improve from 0.76937
391/391 [==============================] - 28s 71ms/step - loss: 0.1231 - accuracy: 0.9658 - val_loss: 1.0259 - val_accuracy: 0.7754 - lr: 5.1200e-04
Epoch 21/100
391/391 [==============================] - ETA: 0s - loss: 0.1015 - accuracy: 0.9729
Epoch 21: val_loss did not improve from 0.76937
391/391 [==============================] - 28s 73ms/step - loss: 0.1015 - accuracy: 0.9729 - val_loss: 1.0638 - val_accuracy: 0.7711 - lr: 5.1200e-04
Epoch 22/100
391/391 [==============================] - ETA: 0s - loss: 0.0992 - accuracy: 0.9736
Epoch 22: ReduceLROnPlateau reducing learning rate to 0.00040960004553198815.

Epoch 22: val_loss did not improve from 0.76937
391/391 [==============================] - 30s 76ms/step - loss: 0.0992 - accuracy: 0.9736 - val_loss: 1.1373 - val_accuracy: 0.7739 - lr: 5.1200e-04
Epoch 23/100
391/391 [==============================] - ETA: 0s - loss: 0.0669 - accuracy: 0.9838
Epoch 23: val_loss did not improve from 0.76937
391/391 [==============================] - 29s 73ms/step - loss: 0.0669 - accuracy: 0.9838 - val_loss: 1.1676 - val_accuracy: 0.7821 - lr: 4.0960e-04
Epoch 24/100
391/391 [==============================] - ETA: 0s - loss: 0.0597 - accuracy: 0.9858
Epoch 24: val_loss did not improve from 0.76937
391/391 [==============================] - 28s 71ms/step - loss: 0.0597 - accuracy: 0.9858 - val_loss: 1.2474 - val_accuracy: 0.7774 - lr: 4.0960e-04
Epoch 25/100
391/391 [==============================] - ETA: 0s - loss: 0.0696 - accuracy: 0.9823Restoring model weights from the end of the best epoch: 10.

Epoch 25: ReduceLROnPlateau reducing learning rate to 0.00032768002711236477.

Epoch 25: val_loss did not improve from 0.76937
391/391 [==============================] - 28s 72ms/step - loss: 0.0696 - accuracy: 0.9823 - val_loss: 1.2465 - val_accuracy: 0.7739 - lr: 4.0960e-04
Epoch 25: early stopping
```

將訓練過程視覺化，結果如下



由上圖可以看到，最終的 validation accuracy 大落在 78 %左右，而這一次的訓練，training accuracy 也能夠順利提升到 100 %左右，代表說模型有辦法去擬合 CIFAR-10 資料集了。

六、 Reference

[1] OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. https://chat.openai.com/

[2] GreeksforGeeks “ML | Underfitting and Overfitting” https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/

[3] 陳明佐 “Transfer Learning 轉移學習 “ https://medium.com/%E6%88%91%E5%B0%B1%E5%95%8F%E4%B8%80%E5%8F%A5-%E6%80%8E%E9%BA%BC%E5%AF%AB/transfer-learning-%E8%BD%89%E7%A7%BB%E5%AD%B8%E7%BF%92-4538e6e2ffe4

[4] TIN-WAI PHOEBE “Transfer Learning with Xception for CIFAR-10” https://www.kaggle.com/code/phoebewongtw/transfer-learning-with-xception-for-cifar-10

[5] 馨伊的閱讀筆記 “Inception 系列 – Xception” https://medium.com/ching-i/inception-%E7%B3%BB%E5%88%97-xception-fd2a4a4e7e82

[5] stack overflow “Transfer learning with Keras, validation accuracy does not improve from outset” https://stackoverflow.com/questions/58676652/transfer-learning-with-keras-validation-accuracy-does-not-improve-from-outset

[6] Chi Ming Lee “值觀理解 ResNet –簡介、觀念及實作(Python Keras)” https://medium.com/@rossleecooloh/%E7%9B%B4%E8%A7%80%E7%90%86%E8%A7%A3resnet-%E7%B0%A1%E4%BB%8B-%E8%A7%80%E5%BF%B5%E5%8F%8A%E5%AF%A6%E4%BD%9C-python-keras-8d1e2e057de2