Assignment #5 Residual Network

F64126147 胡瑀真

注意事項 1: 書面報告可參考網路資料，但須理解與整理，自己理解與整理後撰寫報告，使用整段網路內容所獲的評分不高

注意事項 2: 勿分享報告給課程同學，相似內容的報告所獲評分不高

1. 說明是否參考或使用網路上程式，或全部程式自行撰寫?

程式中，Data augmentation、Early stopping 和 Dropout 皆來源於上次作業之成果，查詢網路資料後，發現 Learning rate schedule 和 Early stopping 語法相似，故參考 Early stopping 語法，修改成需要的 Learning rate schedule。

2. 看懂 residual network 樣板程式，解析其網路結構(程式區塊 A1 與 A2)

程式區塊 A1

程式可被分為兩大部分——函數 identify_block、函數 convolutional_block。兩函數差異主要在 convolutional_block 有設定 X_shortcut 的卷積層濾波器數量等，使 Shortcut 輸出與主路徑一致。

```
1 def identity_block(X, filters):
2     # Retrieve Filters
3     F1, F2, F3 = filters
4     # Save the input value
5     X_shortcut = X
6
7     # First component of main path
8     X = Conv2D(filters=F1, kernel_size=(1, 1), strides=(1, 1), padding='same')(X)
9     X = BatchNormalization()(X)
10    X = Activation('relu')(X)
11
12    # Second component of main path
13    X = Conv2D(filters=F2, kernel_size=(3, 3), strides=(1, 1), padding='same')(X)
14    X = BatchNormalization()(X)
15    X = Activation('relu')(X)
16
17    # Third component of main path
18    X = Conv2D(filters=F3, kernel_size=(1, 1), strides=(1, 1), padding='same')(X)
19    X = BatchNormalization()(X)
20
21    # Final step: Add shortcut value to main path, and pass it through a RELU activation
22    X = Add()([X, X_shortcut])
23    X = Activation('relu')(X)
24
25    return X
```

(1) 定義一個名為 identify_block 的函數，接收 X 和 filters。

(3) 從 filters 的內容拆解出 F1、F2 和 F3，以設定後續三個卷積層的濾波器數量。

(5) 將 X 存入變數 X_shortcut 中，儲存原本的 feature map。

(8) 建立第一層卷積，其中濾波器數量是 F1，濾波器尺寸是 1×1，滑動時步幅 1 像素（用 stride()把 pooling 併入卷積中），並在 feature map 邊緣補 0，令 feature map 不會愈來愈小。

(9) 加上 Batch Normalization 穩定模型訓練的學習、想像力，並加快收斂速度。

(10) 加上激勵函數 ReLU。

(13~15) 建立第二層卷積，類似(8)~(11)，但其中濾波器數量改為 F2、尺寸改為 3×3。

(18~19) 建立第三層卷積，類似(8)~(9)，但其中濾波器數量改為 F3、尺寸改為 1×1。

(22~23) 將 X 和 X_shortcut 相加，再使用激勵函數 ReLU。

(25) 執行完函數內容後，輸出 X。

```
27 def convolutional_block(X, filters):
28     # Retrieve Filters
29     F1, F2, F3 = filters
30     # Save the input value
31     X_shortcut = X
32
33     ##### MAIN PATH #####
34     # First component of main path
35     X = Conv2D(filters=F1, kernel_size=(1, 1), strides=(2, 2), padding='same')(X)
36     X = BatchNormalization()(X)
37     X = Activation('relu')(X)
38
39     # Second component of main path
40     X = Conv2D(filters=F2, kernel_size=(3, 3), strides=(1, 1), padding='same')(X)
41     X = BatchNormalization()(X)
42     X = Activation('relu')(X)
43
44     # Third component of main path
45     X = Conv2D(filters=F3, kernel_size=(1, 1), strides=(1, 1), padding='same')(X)
46     X = BatchNormalization()(X)
47
48     ##### SHORTCUT PATH ####
49     X_shortcut = Conv2D(filters=F3, kernel_size=(1, 1), strides=(2, 2), padding='same')(X_shortcut)
50     X_shortcut = BatchNormalization()(X_shortcut)
51
52     # Final step: Add shortcut value to main path, and pass it through a RELU activation
53     X = Add()([X, X_shortcut])
54     X = Activation('relu')(X)
55     return X
```

(27) 定義一個名為 convolutional_block 的函數，接收 X 和 filters。

(29~46) 類似(3~19)，但第一層卷積的濾波器步幅為 2 像素。

(49) 設定 X_shortcut 的卷積層，濾波器數量為 F3、尺寸為 1×1，和 X 相同。

(50) 加上 Batch Normalization。

(53~55) 同(22~25)。

程式區塊 A2

```
1 def ResidualNetwork(input_shape=(32, 32, 3), classes=10):
2     # Define the input as a tensor with shape input_shape
3     X_input = Input(input_shape)
4
5     # Stage 1
6     X = Conv2D(64, (3, 3), padding='same')(X_input)
7     X = BatchNormalization()(X)
8     X = Activation('relu')(X)
9
10    # Stage 2
11    X = identity_block(X, [16, 16, 64])
12    X = identity_block(X, [16, 16, 64])
13    X = identity_block(X, [16, 16, 64])
14    X = identity_block(X, [16, 16, 64])
15    X = identity_block(X, [16, 16, 64])
16
17    # Stage 3
18    X = convolutional_block(X, filters=[32, 32, 128])
19    X = identity_block(X, [32, 32, 128])
20    X = identity_block(X, [32, 32, 128])
21    X = identity_block(X, [32, 32, 128])
22
23    # Stage 4
24    X = convolutional_block(X, filters=[64, 64, 256])
25    X = identity_block(X, [64, 64, 256])
26
27    # AVGPOOL
28    X = AveragePooling2D(pool_size=(2, 2), padding='same')(X)
29
30    # Output layer
31    X = Flatten()(X)
32    X = Dense(classes, activation='softmax')(X)
33
34    # Create model
35    ResNet_model = Model(inputs=X_input, outputs=X)
36    return ResNet_model
```

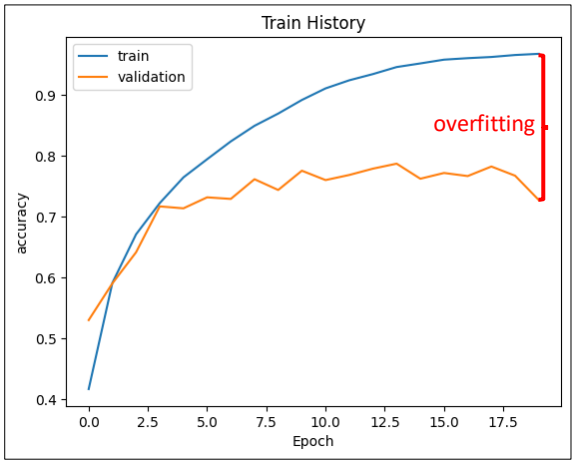
(1) 定義一個名為 ResidualNetwork 的函數，預設輸入 32×32 彩色影像、分類 10 類別。
(3) 將輸入影像形狀存入 X_input 中。
(6) 建立模型的第一層卷積，其中濾波器數量是 64，濾波器尺寸是 3×3，並在 feature map 邊緣補 0，令 feature map 不會愈來愈小。
(7) 加入 Batch Normalization 以穩定模型訓練過程、減緩過渡擬合。
(8) 加上激勵函數 ReLU。
(11~15) 使用在 A1 程式區塊建置的 identify_block 函數，其中 X 為 feature map，濾波器數量 F1、F2 和 F3 分別是 16、16 和 64，透過堆疊 5 次學習 5 次殘差，習得更深的特徵。
(18) 使用在 A1 程式區塊建置的 convolutional_block 函數，其中讓 Shortcut 進行一次卷積。
(19~21) 類似(11~15)，但濾波器數量改為 32、32 和 128，堆疊 3 次。
(24) 類似(18)，但濾波器數量 F1、F2 和 F3 分別是 64、64 和 256。
(25) 類似(11)，，但濾波器數量改為 64、64 和 256。
(28) 用 AveragePooling2D()語法，每 2×2 的區塊便計算、保留其平均值，使該區塊減小為 1×1，達到縮減影像尺寸、保留重要資訊並減少資訊量。
(31) 用 Flatten()將多維 feature map 展平成 1 維向量，以便與後續分類功能的 Fully-Connected layer 連接。
(32) 使用激勵函數 softmax 做機率形式的分類。
(35) 將剛剛建立的模型，指定其輸入層為 X_input，輸出層為 X，再將整個模型存為 ResNet_model，最後在執行完函數後回傳。

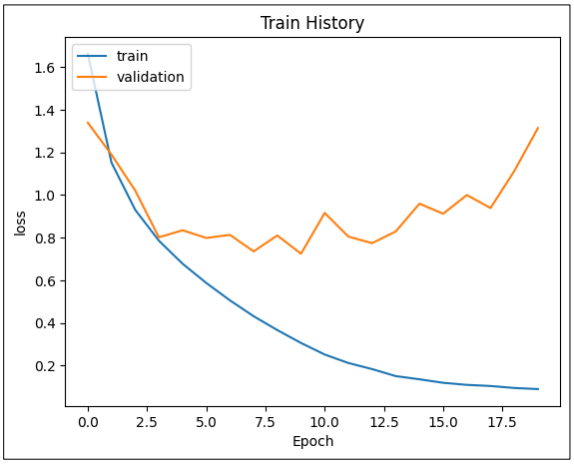
3. 使用 residual network 提高分類準確度過程

3.1 提高分類準確度的過程

(從課程給定的 template model 到所獲得最佳模型的過程，須報告至少三個分類準確度提高的過程，包含最佳模型，不包含 template model)

template model 訓練結果





Accuracy of testing data = 72.3%

Data augmentation

在前次作業中，我使用 Data augmentation 令測試資料的準確度上升近 8%，相較其他方式讓準確度高許多。因此，針對此次作業中，訓練和驗證資料準確度差異大的過擬合現象，我決定首先運用 Data augmentation，期望藉增加資料多樣性，減輕過渡擬合現象，並增加模型的泛化、推演能力。

```
 7 img_gen = ImageDataGenerator(rotation_range=15, width_shift_range=0.1, height_shift_range=0.1,
 8                              horizontal_flip=True, zoom_range=0.1, validation_split=0.2)
 9
10 batch_size = 32
11 train_generator = img_gen.flow(x_train_norm, y_TrainOneHot, batch_size=batch_size, subset='training')
12 val_generator = img_gen.flow(x_train_norm, y_TrainOneHot, batch_size=batch_size, subset='validation')
13
14 steps = train_generator.n // train_generator.batch_size
15 validation_steps = val_generator.n // val_generator.batch_size
16
17 train_history = ResNet.fit(train_generator, steps_per_epoch=steps, validation_data=val_generator,
18                            validation_steps=validation_steps, epochs=25, shuffle=True)
```

改變原本的 ResNet.fit()，先從 tensorflow.keras.preprocessing.image 匯入 ImageDataGenerator，隨機變換原本影像的角度、縮放等，再用改變過的影像做模型訓練，使影像有更多變化性。

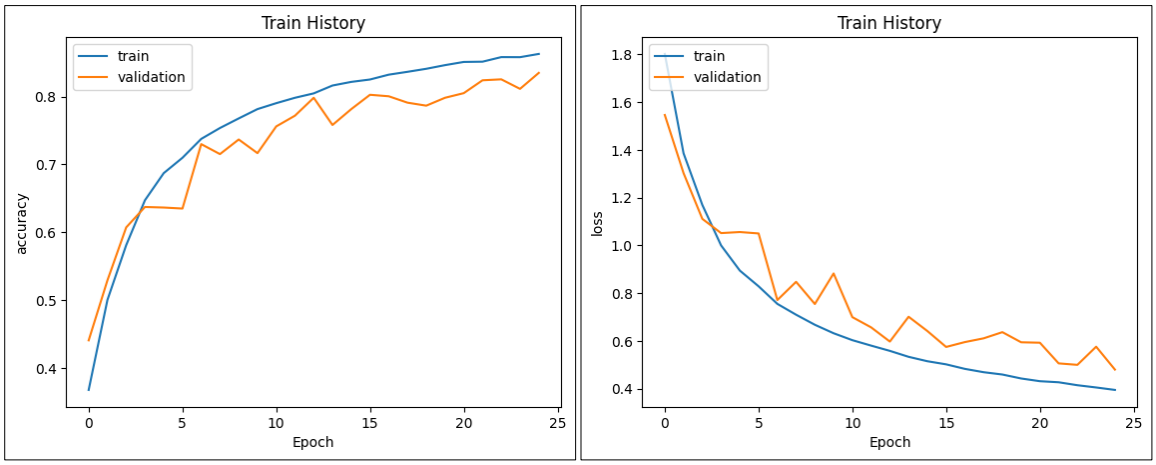
(97) 使用 ImageDataGenerator()語法，將影像隨機旋轉±15 度、隨機水平移動±10%、隨機垂直移動±10%、隨機水平翻轉圖片、隨機放大或縮小影像，最後分割 20%的資料作為驗證資料。

(100) 設定 batch_size 為 32，表示 32 筆資料為一組。

(101~102) 運用 flow()語法，以動態取樣的方式分別設定訓練和驗證資料的 generator，計算後續 fit()語法會需要的 train_generator 和 validation_data。

(104~105) 計算後續 fit()語法會需要的 steps_per_epoch、validation_steps，分別表示 32 筆資料為一組後，每一輪 epoch 要從 generator 中取幾次 batch 才能完整訓練。

(107) 使用 fit()語法訓練模型。



Accuracy of testing data = 82.9%

從 Data Augmentation 後的 train history 可發現，訓練資料的準確度隨訓練而遞增，最後達到約 85%，而驗證資料的準確度雖偶有波折，但整體仍穩定增加，最後超過 80%。此外，由於訓練與驗證的準確度差距大幅縮減，顯示過渡擬合的問題被解決、模型的訓練過程穩定，又測試資料的 accuracy 從 72.3%升至 82.9%，攀升超過 10%，表示 Data Augmentation 的效果非常好。

Go deep + Early stopping

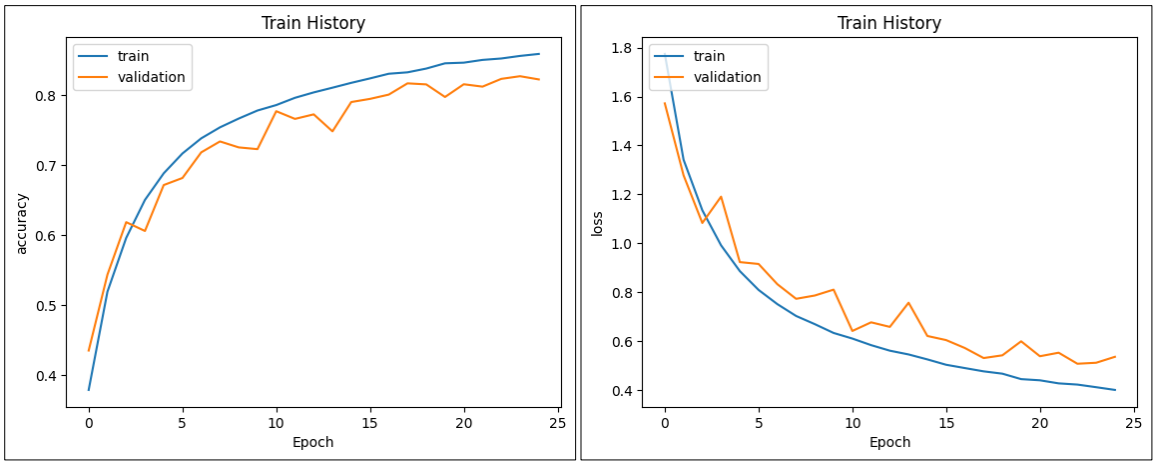
由於 Residual Network 學習殘差的特性，可以有效減少梯度消失的問題，增加深度的動作較傳統的 CNN 模型安全，也不會因深度增加而使準確度下降。因此，透過增加模型的深度搭配 Early stopping，在避免過擬合的情況下，讓模型學習更深特徵，期望進一步增加準確度。

```
def ResidualNetwork(input_shape=(32, 32, 3), classes=10):
    # Define the input as a tensor with shape input_shape
    X_input = Input(input_shape)

    # Stage 1
    X = Conv2D(64, (3, 3), padding='same')(X_input)
    X = BatchNormalization()(X)
    X = Activation('relu')(X)

    # Stage 2
    X = identity_block(X, [16, 16, 64])
    X = identity_block(X, [16, 16, 64])
    X = identity_block(X, [16, 16, 64])
    X = identity_block(X, [16, 16, 64])
    X = identity_block(X, [16, 16, 64])
    X = identity_block(X, [16, 16, 64])
    X = identity_block(X, [16, 16, 64])
```

在函數 ResidualNetwork 中 Stage 2 的部分，增加 2 層 identify_block 函數做堆疊。

```
early_stopping = EarlyStopping(monitor='val_accuracy', patience=10, min_delta=0.0001, restore_best_weights=True)
train_history = ResNet.fit(train_generator, steps_per_epoch=steps, validation_data=val_generator,
                           validation_steps=validation_steps, epochs=25, shuffle=True, callbacks=[early_stopping])
```

使用 EarlyStopping()語法，偵測 val_accuracy 的值，當該數值超過 10 個週期都沒有改善便停止訓練，並判斷最小變化量為 0.0001，且訓練結束時權重會回復到訓練過程中表現最佳的數值。



Accuracy of testing data = 83.4%

觀察 Go deep + Early stopping 後的 train history 可發現，訓練和驗證資料的準確度、損失曲線皆和前次的訓練結果相似，整體準確度、損失函數值分別穩定上升與下降。並且，Early stopping 的機制也沒有被觸發，顯示模型學習更深後，過擬合的問題沒有發生，訓練及驗證資料準確度也穩定增加，測試資料的準確度從 82.9%增至 83.4%，提高 0.5%。

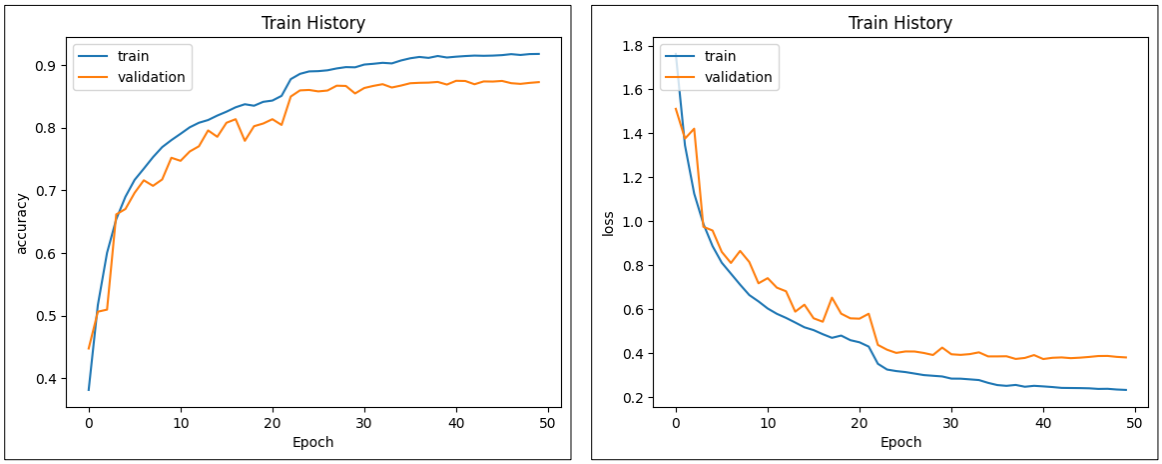
Learning rate schedule

參考作業講義中提示，運用 Learning rate schedule 動態調整學習效率，再增加 epoch 數量，給模型更多時間學習。

```
17 early_stopping = EarlyStopping(monitor='val_loss', patience=10, min_delta=0.0001, restore_best_weights=True)
18 reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.00001, verbose=1)
19 train_history = ResNet.fit(train_generator, steps_per_epoch=steps, validation_data=val_generator,
20                            validation_steps=validation_steps, epochs=50, shuffle=True, callbacks=[early_stopping, reduce_lr])
```

Learning rate schedule 的邏輯和程式與 Early stopping 相似：透過偵測 val_loss 的值，當數值超過 5 個 epoch 都沒有改善時，便降低學習效率為原本的 0.2 倍，並判斷最小變化量為 0.00001。

最後，將 epoch 從 25 增加至 50。



Accuracy of testing data = 87.3%

從 Learning rate schedule 後的 train history 可看出，訓練和驗證資料的準確度及損失函數值，皆穩定增加與減少，最後訓練資料準確度超及 90%，驗證資料準確度將近 90%，兩者相差小，無過擬合問題。

此外，觀察準確度及損失函數值曲線可發現，約 23 epoch 時，曲線起伏減緩許多、形狀較之前的週期更加平穩，但仍穩定增減，顯示學習效率的調降發生。訓練後，測試資料的準確度從 83.4%增至 87.3%，增加近 5%，進步幅度相當大。

Go deep + Dropout

由於 Learning rate schedule、Early stopping，以及 Residual Network 的特性，我持

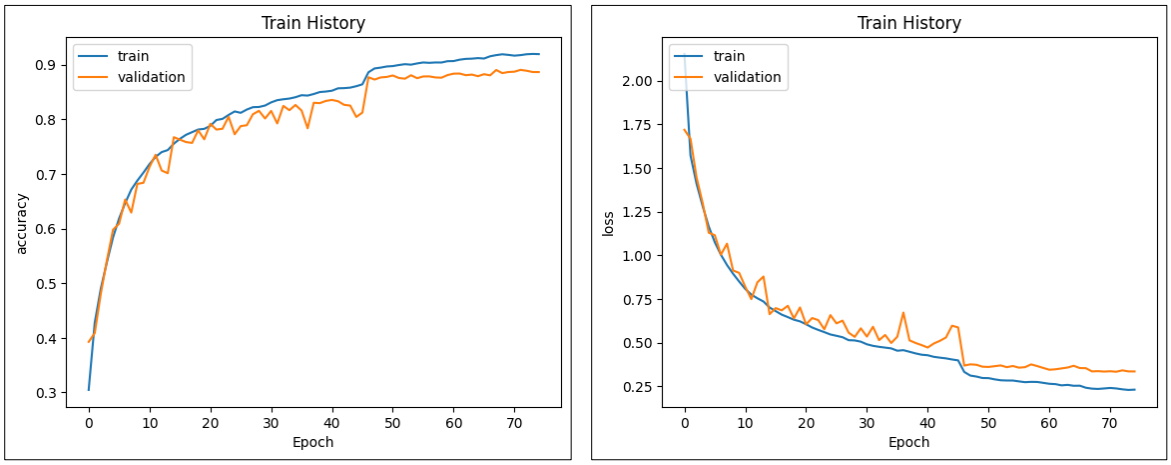
續增加模型深度、epoch 數量，再搭配一層 Dropout，以提升準確率與泛化能力，同時有效抑制過擬合。

```
# Stage 3
X = convolutional_block(X, filters=[32, 32, 128])
X = identity_block(X, [32, 32, 128])
X = identity_block(X, [32, 32, 128])
X = identity_block(X, [32, 32, 128])
X = identity_block(X, [32, 32, 128])
X = identity_block(X, [32, 32, 128])

# Stage 4
X = convolutional_block(X, filters=[64, 64, 256])
X = identity_block(X, [64, 64, 256])
X = identity_block(X, [64, 64, 256])


# AVGPOOL
X = AveragePooling2D(pool_size=(2, 2), padding='same')(X)

# Output layer
X = Flatten()(X)
X = Dropout(0.5)(X)
X = Dense(classes, activation='softmax')(X)
```

在函數 ResidualNetwork 中 Stage 3 的部分，增加 2 層 identify_block 函數做堆疊，並在 Stage 4 的部分，增加 1 層 identify_block 函數做堆疊。

由於 Dropout 只針對大且帶有 Dense layer 的 NN 有效果，故在分類輸出層前，增加 Dropout(0.5)。

最後，將 epoch 從 50 增加至 75。



Accuracy of testing data = 89.2%

從 Train history 中可發現，兩準確度和損失函數值的變化與前次模型訓練結果相似，但仍隨著巡練而持續爬升、降低，最後訓練準確度超過 90%，驗證準確度接近 90%。

觀察曲線形狀，可發現約 45 epoch 後，曲線的形狀不再有明顯上下起伏，表示 Learning rate schedule 的觸發。最後，測試資料的準確度從 87.3%增至 89.2%，增加近 2%，並超過我上次作業的準確度 87.9%。

3.2 所完成的最佳模型

(模型結構、所使用的超參數、模型訓練與驗證、測試資料的分類準確度)

模型結構

模型是 Residual Network，具有學習「訓練結果與初始值之差」的特性。模型中，首先 Stage 1，堆疊一層 Conv2D，再搭配一個 Batch Normalization 與激活函數 ReLU。接著，分別堆疊 7 層、2 層和 1 層 identify_block 函數於 Stage 2、3 和 4 中，以增加模型的學習深度，是模型能學到更抽象的特徵。

而後，於展平多維 feature map 成 1 維向量前，使用 AveragePooling2D()，取每 2×2 的區塊平均值，使該區塊減小為 1×1，達到縮減影像尺寸、保留重要資訊並減少資訊量。展平後，於分類層前使用 Dropout(0.5)，以隨機地忽略一些神經元，使模型對神經元的特定權重敏感度下降，提升其泛化能力也不易過擬合。

最後，用 Dense(10)和 softmax 做輸出機率形式的分類結果。

使用的超參數

Optimizer：Adam；Batch Size：32；Epochs：75；Dropout：0.5；Validation Split：0.2；Early Stopping：monitor=val_loss、patience=10、restore_best_weights=True；Learning Rate Schedule：monitor=val_loss、patience=5、factor=0.2。

模型訓練與驗證

模型前後運用 Batch Normalization 和 Dropout 穩定模型訓練過程、減緩過渡擬合。而模型內，堆疊多層 identify_block 函數，並穿插個別 convolutional_block，透過增加模型深度，強化模型學習特徵的能力。

訓練模型前，例用 Data Augmentation 藉由將影像旋轉、平移、水平翻轉和縮放，提高訓練資料多樣性，顯著提升模型資料泛化能力，解決過渡擬合問題並提高分類準確度。

訓練過程中，偵測驗證資料損失函數值，當無明顯提升時，運用 Early Stopping 停止訓練，並保留準確度最佳時的模型權重，或運用 Learning Rate Schedule 將學習效率降低為原本的 0.2 倍。

測試資料的分類準確度

原始分類準確度為： 72.3%；使用 Data Augmentation 後，分類準確度為 82.9%；接著，加入 Go deep 搭配 Early stopping 後，分類準確度為 83.4%；而後，加入 Learning rate schedule 後，分類準確度為 87.3%；最後，Go deep 並 Dropout，分類準確度達到 89.2%。

根據最後的 train history 可知，兩曲線差距小，模型無過渡擬合問題、梯度消失問題，學習效果良好、分類準確度高。

4. VGG 與 residual network 分類準確度比較

4.1 模型結構與模型超參數

模型結構部分，作業四中，VGG-like 模型依序建立三組卷積層（每組內含三層卷積）、一層 MaxPooling2D、一層 Flatten、兩層 Dense(128)、一層 Dr 一層 opout(0.2)、Dense(10)搭配激勵函數 softmax 輸出分類結果。此外，訓練前，使用 Data Augmentation

增加訓練資料多樣性；訓練時，使用 Early Stopping 使模型在過渡擬合前停止訓練，避免過渡擬合。

此次作業中，residual network 模型先建立一層卷積，再依序堆疊七層 identity block、一層 convolutional block、五層 identity block、一層 convolutional block、兩層 identity block、一層 AveragePooling2D、一層 Flatten、一層 Dropout(0.5)和一層 Dense(10)搭配激勵函數 softmax 輸出分類結果。與 VGG-like 模型相同，運用 Data Augmentation 和 Early Stopping，並在訓練時額外使用 Learning Rate Schedule 提升準確度、泛化能力。

兩模型中，每層卷積層後皆搭配 Batch Normalization 與激活函數 ReLU 以穩定訓練過程和提升非線性特徵。residual network 模型中，每層 identity block、convolutional block 皆包含三層卷積，並在卷積後做殘差的相加學習更深層特徵，而 convolutional block 較 identity block，增加影像降維的功能。

模型超參數部分，兩者差別在於 residual network 有 Learning Rate Schedule 相關的超參數，且 epoch 為 75、Dropout rate 為 0.5；相較之下，VGG 無 Learning Rate Schedule 相關的超參數，其 epoch 和 Dropout rate 較少，分別是 25 和 0.2。

4.2 模型比較結果

比較兩模型結構，可發現 VGG-like 單純使用多層卷積做訓練，而 residual network 透過 identity block 和 convolutional block，達到「即使模型深度增加，模型仍能避免梯度消失且有效學習」，進而使 residual network 的模型深度遠勝 VGG-like。

另外，VGG-like 運用 MaxPooling2D 對特徵圖做保留最大值的降維，而 residual network 使用 convolutional block 中卷積層濾波器的步幅，以及 AveragePooling2D 對特徵圖做運算和保留平均值的降維處理。

最後，residual network 更額外採用 Learning Rate Schedule 穩定模型的收斂。

訓練結果顯示，residual network 的測試資料準確度為 89.2%，而 VGG 為 87.9%，residual network 的比較結果較佳，相差 1.3%。

5. 以分類問題而言，VGG 與 residual network 那個模型較好，為什麼?

從本次與前次作業的成果可發現，residual network 的測試資料準確度較 VGG 高 1.3%，為 89.2%。因此，以分類問題而言，residual network 較好。

觀察 residual network 和 VGG 的差異，可推測 residual network 較好，是因其中的 Shortcut 結構，透過學習殘差而非整體特徵的方式，讓 residual network 的模型深度較深，在學習較複雜特徵和增進泛化能力的同時，不易產生梯度消失或過渡擬合的問題。

因 residual network 可以安全有效地藉增加深度而加強學習能力，雖然目前我的 residual network 準確度僅較 VGG 高 1.3%，但藉由增加 epoch 數量和調整 learning rate schedule，便很有潛力讓準確度進一步提高。

相較之下，VGG 學習整體特徵，如若加深深度，容易使模型太過複雜和貼合訓練資料，而發生過渡擬合、降低泛化能力和準確度。

參考資料

1. 上課講義…..
2. http://www.deeplearning.com