預訓練模型與轉移學習

吳庭育

tyw@mail.npust.edu.tw

Keras內建的預訓練模型 (Pre-trained Models)

預訓練模型的種類

- Keras應用程式(Applications)是一些Keras內建已經成功完成訓練的深度學習模型，除了模型結構，還包含預訓練的權重，所以也稱為「預訓練模型」(Pre-trained Models)。
- 目前Keras內建可用的預訓練模型如下：
  - Xception
  - VGG16
  - VGG19
  - ResNet, ResNetV2, ResNeXt
  - InceptionV3
  - InceptionResNetV2
  - MobileNet
  - MobileNetV2
  - DenseNet
  - NASNet

ImageNet 競賽的冠軍

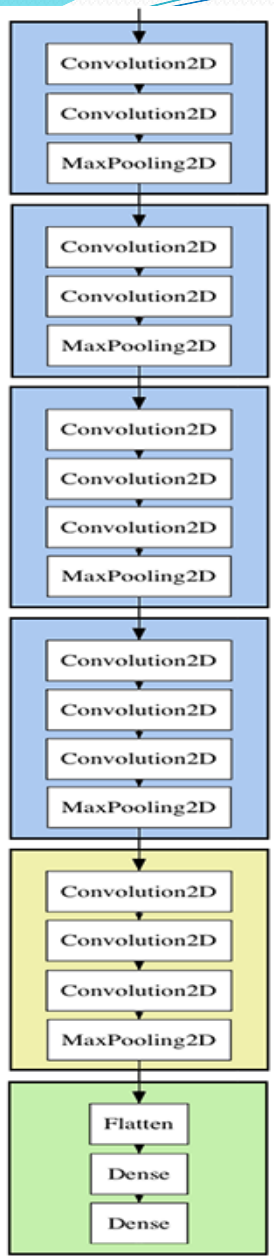
- ImageNet 每年舉辦的競賽(ILSVRC)這幾年產生了不少的CNN冠軍，歷屆比賽的模型演進非常精彩：
  - 2012年冠軍 AlexNet 錯誤率比前一年減少超過10%，且首度引用 Dropout 層。
  - 2014年亞軍 VGGNet 承襲 AlexNet 思路，建立更多層的模型，達到16及19 個隱藏層。
  - 2014年圖像分類冠軍 GoogNet & Inception 同時使用多種不同大小的 Kernel，讓系統決定最佳的 Kernel。Inception 引入 Batch Normalization 等觀念，參見Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift。
  - 2015年冠軍 ResNets 發現 20 層以上的模型前面幾層會發生優化退化(degradation)的狀況，因而提出以『殘差』(Residual)解決問題，參見Deep Residual Learning for Image Recognition。

模型結構資訊

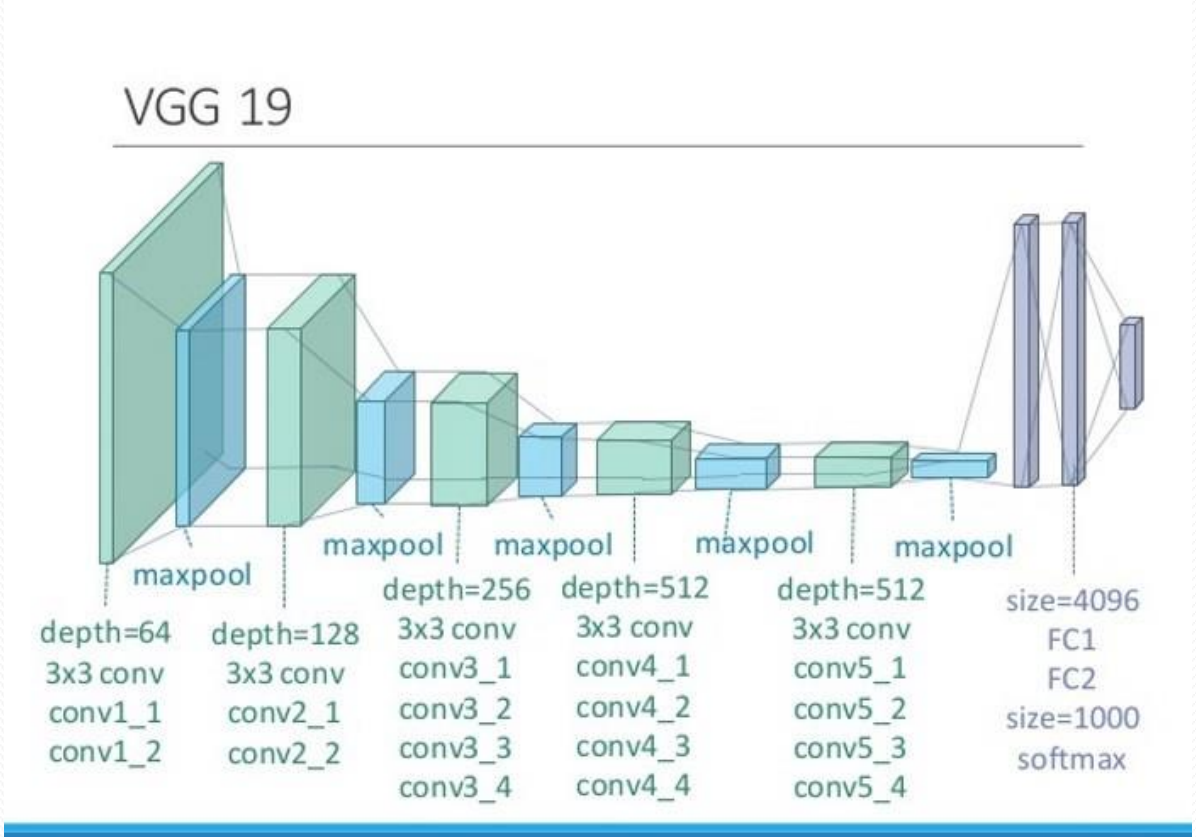
Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88

- Top-1表示只預測一次且正確的機率。
- Top-5表示預測五次只要一次猜對就算正確的機率。
- Size：記憶體的最高佔據量。
- Parameters：參數的數量，愈多就須計算愈久。
- Depth：filters的數目

VGG16/VGG19 模型結構



- VGG 是英國牛津大學 Visual Geometry Group 的縮寫，主要貢獻是使用更多的隱藏層，大量的圖片訓練，提高準確率至90%。VGG16/VGG19分別為16層(13個卷積層及3個全連接層)與19層(16個卷積層及3個全連接層)，結構圖如下。



VGG16使用方法

- 所有Applications執行都只要一行指令，就可以把模型及權重載入程式中，例如載入VGG16的指令如下：
  model = VGG16(weights='imagenet', include_top=True)
  相關參數說明:
- include_top：是否包含頂部(Top) 3層『完全連階層』(fully-connected layers)。
  - include_top = False：只利用VGG16萃取特徵，後面的分類處理，都要自己設計。反之，就是全盤接受VGG16，只是要改變輸入而已。
  - 注意!! 頂部(Top)指的是位於結構圖最後面，因為它是一個『後進先出法』的概念，一層一層堆疊上去，最上面一層就是頂部(Top)。
- weights：使用的權重，分兩種
  - imagenet：即使用ImageNet的預先訓練的資料，約100萬張圖片，判斷1000類別的日常事物，例如動物、交通工具...等，我們通常選這一項。
  - None：隨機起始值，我沒試過，請有興趣的讀者自行測試。

使用預訓練模型進行圖片分類預測

MobileNet物件

```
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import load_img
from keras.applications.mobilenet import MobileNet
from keras.applications.mobilenet import preprocess_input
from keras.applications.mobilenet import decode_predictions

# 建立 MobileNet 模型
model = MobileNet(weights="imagenet", include_top=True)
# 載入測試圖片
img = load_img("koala.png", target_size=(224, 224))
x = img_to_array(img)       # 轉換成 Numpy陣列
print("x.shape: ", x.shape)
# Reshape (1, 224, 224, 3)
img = x.reshape((1, x.shape[0], x.shape[1], x.shape[2]))
# 資料預處理
img = preprocess_input(img)
print("img.shape: ", img.shape)
# 使用模型進行預測
Y_pred = model.predict(img)
# 解碼預測結果
label = decode_predictions(Y_pred)
result = label[0][0]    # 取得最可能的結果
print("%s (%.2f%%)" % (result[1], result[2]*100))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_1_0_224_tf.h5
17227776/17225924 [==============================] - 0s 0us/step
17235968/17225924 [==============================] - 0s 0us/step
x.shape:  (224, 224, 3)
img.shape:  (1, 224, 224, 3)
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
40960/35363 [==================================] - 0s 0us/step
49152/35363 [==========================================] - 0s 0us/step
koala (100.00%)
```

使用MobileNet預訓練模型

- 建立 MobileNet 模型：

  model = MobileNet(weights="imagenet", include_top=True)

  - weights參數：模型使用的權重，參數值imagenet 是使用ImageNet的預訓練權重，其訓練資料集有100萬張圖片，分類成1000種類別，參數值None，就只使用模型結構，需要自行訓練權重。
  - Include_top參數：是否包含模型頂部的完全連結層，這是指平坦層厚的分類神經層，參數值True包含分類神經層，參數值False則否，模型只有特徵萃取的神經層，我們可以自新增所需的分類神經層，稱為轉移學習(Transfer Learning)

- 在載入圖檔koala.png和調整成(224, 224)尺寸後，呼叫img_to_array()函式轉換成 NumPy陣列，我們需要將圖片的NumPy陣列轉換成4D張量(1, 244, 244, 3)，和處理成模型所需的輸入資料格式：

  img = x.reshape((1, x.shape[0], x.shape[1], x.shape[2]))img = preprocess_input(img)

- 最我們可以使用模型進行模型分類預測和解碼預測結果：

```
Y_pred = model.predict(img)
label = decode_predictions(Y_pred)
result = label[0][0]
print("%s (%.2f%%)" % (result[1], result[2]*100))
```

```
koala (100.00%)
```

RasNet50

```
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import load_img
from keras.applications.resnet import ResNet50
from keras.applications.resnet import preprocess_input
from keras.applications.resnet import decode_predictions

# 建立 RasNet50 模型
model = ResNet50(weights="imagenet", include_top=True)
# 載入測試圖片
img = load_img("koala.png", target_size=(224, 224))
x = img_to_array(img)        # 轉換成 Numpy陣列
print("x.shape: ", x.shape)
# Reshape (1, 224, 224, 3)
img = x.reshape((1, x.shape[0], x.shape[1], x.shape[2]))
# 資料預處理
img = preprocess_input(img)
print("img.shape: ", img.shape)
# 使用模型進行預測
Y_pred = model.predict(img)
# 解碼預測結果
label = decode_predictions(Y_pred)
result = label[0][0]    # 取得最可能的結果
print("%s (%.2f%%)" % (result[1], result[2]*100))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5
102973440/102967424 [==============================] - 1s 0us/step
102981632/102967424 [==============================] - 1s 0us/step
x.shape:  (224, 224, 3)
img.shape:  (1, 224, 224, 3)
koala (99.93%)
```

InceptionV3

```
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import load_img
from keras.applications.inception_v3 import InceptionV3
from keras.applications.inception_v3 import preprocess_input
from keras.applications.inception_v3 import decode_predictions

# 建立 InceptionV3 模型
model = InceptionV3(weights="imagenet", include_top=True)
# 載入測試圖片
img = load_img("koala.png", target_size=(299, 299))
x = img_to_array(img)        # 轉換成 Numpy陣列
print("x.shape: ", x.shape)
# Reshape (1, 299, 299, 3)
img = x.reshape((1, x.shape[0], x.shape[1], x.shape[2]))
# 資料預處理
img = preprocess_input(img)
print("img.shape: ", img.shape)
# 使用模型進行預測
Y_pred = model.predict(img)
# 解碼預測結果
label = decode_predictions(Y_pred)
result = label[0][0]    # 取得最可能的結果
print("%s (%.2f%%)" % (result[1], result[2]*100))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels.h5
96116736/96112376 [==============================] - 0s 0us/step
96124928/96112376 [==============================] - 0s 0us/step
x.shape: (299, 299, 3)
img.shape: (1, 299, 299, 3)
koala (91.51%)
```

VGG16

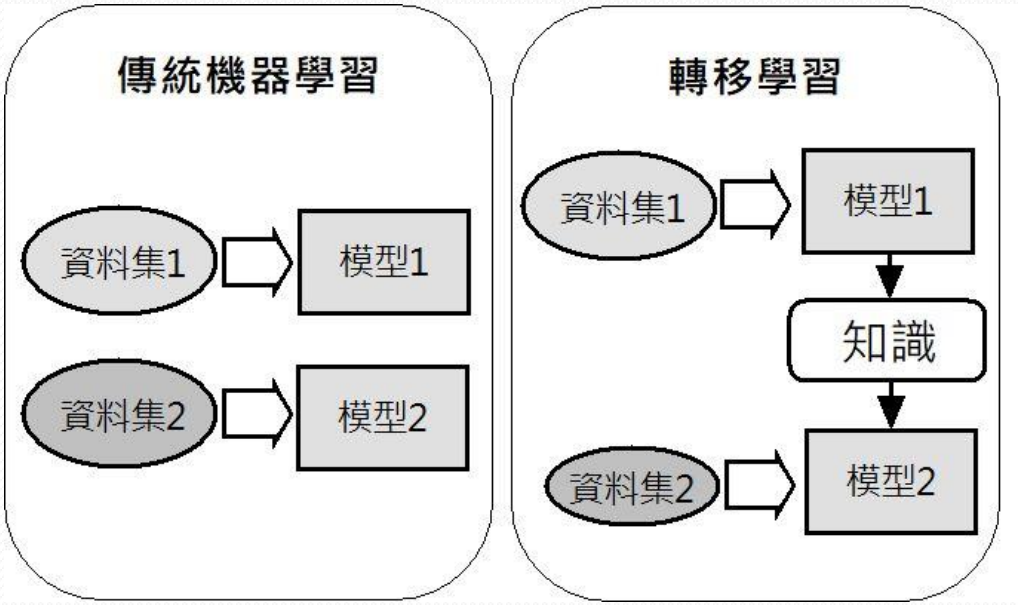
```
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import load_img
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions

# 建立 VGG16 模型
model = VGG16(weights="imagenet", include_top=True)
# 載入測試圖片
img = load_img("koala.png", target_size=(224, 224))
x = img_to_array(img)        # 轉換成 Numpy陣列
print("x.shape: ", x.shape)
# Reshape (1, 224, 224, 3)
img = x.reshape((1, x.shape[0], x.shape[1], x.shape[2]))
# 資料預處理
img = preprocess_input(img)
print("img.shape: ", img.shape)
# 使用模型進行預測
Y_pred = model.predict(img)
# 解碼預測結果
label = decode_predictions(Y_pred)
result = label[0][0]    # 取得最可能的結果
print("%s (%.2f%%)" % (result[1], result[2]*100))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467904/553467096 [==============================] - 5s 0us/step
553476096/553467096 [==============================] - 5s 0us/step
x.shape: (224, 224, 3)
img.shape: (1, 224, 224, 3)
koala (100.00%)
```

轉移學習
(Transfer Learning)

認識轉移學習

- 在人類的學習過程中，我們常常會從之前任務學習到的知識直接套用在目前的任務上，這就是「轉移學習」(Transfer Learning)。轉移學習是一種機器學習技術，可以將原來針對指定任務所建立的已訓練模型，直接更改任務來訓練出解決其他相關任務的模型：

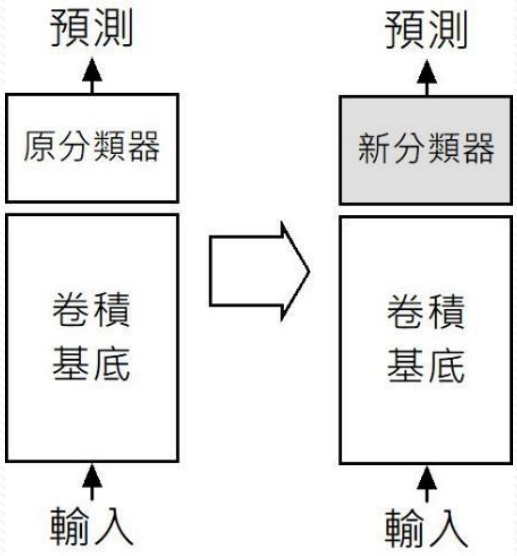


認識轉移學習

- 對於深度學習的卷積神經網路(CNN)來說，神經網路可以分成兩部分，如下所示：
  - 卷積基底(Convolutional Base)：使用多組卷積和池化層開始的神經層，可以執行特徵萃取。
  - 分類器(Classifier)：在卷積基底後是使用Dense全連接層建立的分類器。

認識轉移學習

- 轉移學習之所以有用，這是因為愈前面的卷積層，其學到的特徵是一種愈泛化的特徵，這些特徵並不會因為不同的訓練資料集而訓練出不同的結果，所以學到的特徵，可以轉移至其他相關的圖片分類問題。
- 卷積神經網路保留原來特徵萃取部分的卷積基底，我們只需更改位在頂部的分類器，就可以使用轉移學習來解決其他相關的圖片分類問題。



MNIST手寫辨識的轉移學習

```python
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout
from tensorflow.keras.utils import to_categorical

# 指定亂數種子
seed = 7
np.random.seed(seed)
# 載入資料集
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
# 建立2個資料集，一個數字小於 5，一個數字大於等於 5
X_train_lt5 = X_train[Y_train < 5]
Y_train_lt5 = Y_train[Y_train < 5]
X_test_lt5 = X_test[Y_test < 5]
Y_test_lt5 = Y_test[Y_test < 5]

X_train_gte5 = X_train[Y_train >= 5]
Y_train_gte5 = Y_train[Y_train >= 5] - 5
X_test_gte5 = X_test[Y_test >= 5]
Y_test_gte5 = Y_test[Y_test >= 5] - 5
# 將圖片轉換成 4D 張量
X_train_lt5 = X_train_lt5.reshape(
              (X_train_lt5.shape[0], 28, 28, 1)).astype("float32")
X_test_lt5 = X_test_lt5.reshape(
              (X_test_lt5.shape[0], 28, 28, 1)).astype("float32")
X_train_gte5 = X_train_gte5.reshape(
              (X_train_gte5.shape[0], 28, 28, 1)).astype("float32")
X_test_gte5 = X_test_gte5.reshape(
              (X_test_gte5.shape[0], 28, 28, 1)).astype("float32")
# 因為是固定範圍，所以執行正規化，從 0-255 至 0-1
X_train_lt5 = X_train_lt5 / 255
X_test_lt5 = X_test_lt5 / 255
X_train_gte5 = X_train_gte5 / 255
X_test_gte5 = X_test_gte5 / 255
# One-hot編碼
Y_train_lt5 = to_categorical(Y_train_lt5, 5)
Y_test_lt5 = to_categorical(Y_test_lt5, 5)
Y_train_gte5 = to_categorical(Y_train_gte5, 5)
Y_test_gte5 = to_categorical(Y_test_gte5, 5)
```

```python
# 定義模型
model = Sequential()
model.add(Conv2D(8, kernel_size=(3, 3),
                 input_shape=(28, 28, 1), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(8, kernel_size=(3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.25))
model.add(Dense(5, activation="softmax"))
model.summary()      # 顯示模型摘要資訊
# 編譯模型
model.compile(loss="categorical_crossentropy", optimizer="adam",
              metrics=["accuracy"])
# 訓練模型
history = model.fit(X_train_lt5, Y_train_lt5, validation_split=0.2,
                    epochs=5, batch_size=128, verbose=2)
# 評估模型
loss, accuracy = model.evaluate(X_test_lt5, Y_test_lt5)
print("測試資料集的準確度 = {:.2f}".format(accuracy))
# 顯示各神經層
print(len(model.layers))
for i in range(len(model.layers)):
    print(i, model.layers[i])
# 凍結上層模型
for i in range(4):
    model.layers[i].trainable = False
# 編譯模型
model.compile(loss="categorical_crossentropy", optimizer="adam",
              metrics=["accuracy"])
# 訓練模型
history = model.fit(X_train_gte5, Y_train_gte5, validation_split=0.2,
                    epochs=5, batch_size=128, verbose=2)
# 評估模型
loss, accuracy = model.evaluate(X_test_gte5, Y_test_gte5)
print("測試資料集的準確度 = {:.2f}".format(accuracy))
```

MNIST手寫辨識的轉移學習

- MNIST手寫辨識的轉移學習是使用Keras內建的MNIST資料集，在建立CNN模型後，使用前5個手寫數字圖片(0~4)學習到的手寫數字圖片的特徵，轉移到後5個手寫數字圖片(5~9)的分類辨識。

```
12 #  建立2個資料集，一個數字小於  5，一個數字大於等於  5
13 X_train_lt5  =  X_train[Y_train  <  5]
14 Y_train_lt5  =  Y_train[Y_train  <  5]
15 X_test_lt5  =  X_test[Y_test  <  5]
16 Y_test_lt5  =  Y_test[Y_test  <  5]
17
18 X_train_gte5  =  X_train[Y_train  >=  5]
19 Y_train_gte5  =  Y_train[Y_train  >=  5]  -  5
20 X_test_gte5  =  X_test[Y_test  >=  5]
21 Y_test_gte5  =  Y_test[Y_test  >=  5]  -  5
```

- 首先訓練前5個數字，接著在訓練後5個數字，所以我們需要建立2組訓練和測試資料集，第1組是數字小於5，第二組是數字大於等於5

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
Model: "sequential"
```

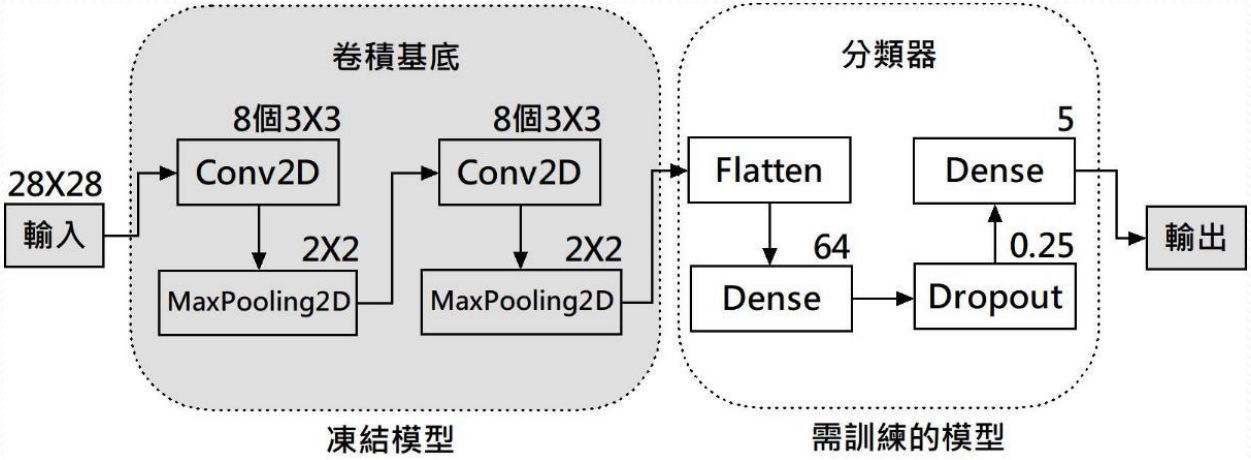
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 8)	80
max_pooling2d (MaxPooling2D)	(None, 13, 13, 8)	0
conv2d_1 (Conv2D)	(None, 11, 11, 8)	584
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 8)	0
flatten (Flatten)	(None, 200)	0
dense (Dense)	(None, 64)	12864
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 5)	325

```
Total params: 13,853
Trainable params: 13,853
Non-trainable params: 0
```

```
Epoch 1/5
192/192 - 13s - loss: 0.4226 - accuracy: 0.8670 - val_loss: 0.1041 - val_accuracy: 0.9681 - 13s/epoch - 68ms/step
Epoch 2/5
192/192 - 1s - loss: 0.1235 - accuracy: 0.9619 - val_loss: 0.0714 - val_accuracy: 0.9763 - 646ms/epoch - 3ms/step
Epoch 3/5
192/192 - 1s - loss: 0.0899 - accuracy: 0.9740 - val_loss: 0.0559 - val_accuracy: 0.9819 - 655ms/epoch - 3ms/step
Epoch 4/5
192/192 - 1s - loss: 0.0740 - accuracy: 0.9777 - val_loss: 0.0422 - val_accuracy: 0.9866 - 665ms/epoch - 3ms/step
Epoch 5/5
192/192 - 1s - loss: 0.0586 - accuracy: 0.9825 - val_loss: 0.0380 - val_accuracy: 0.9891 - 701ms/epoch - 4ms/step
161/161 [==============================] - 0s 3ms/step - loss: 0.0289 - accuracy: 0.9907
測試資料集的準確度 = 0.99
8
0 <keras.layers.convolutional.Conv2D object at 0x7fd7f0607c90>
1 <keras.layers.pooling.MaxPooling2D object at 0x7fd7f09b8450>
2 <keras.layers.convolutional.Conv2D object at 0x7fd7e0620310>
3 <keras.layers.pooling.MaxPooling2D object at 0x7fd7e05f9090>
4 <keras.layers.core.flatten.Flatten object at 0x7fd7e05a3910>
5 <keras.layers.core.dense.Dense object at 0x7fd7e05a3a10>
6 <keras.layers.core.dropout.Dropout object at 0x7fd7e05b6f10>
7 <keras.layers.core.dense.Dense object at 0x7fd7e05b6790>
Epoch 1/5
184/184 - 1s - loss: 0.6927 - accuracy: 0.7741 - val_loss: 0.2636 - val_accuracy: 0.9231 - 1s/epoch - 6ms/step
Epoch 2/5
184/184 - 1s - loss: 0.2973 - accuracy: 0.9042 - val_loss: 0.1630 - val_accuracy: 0.9517 - 570ms/epoch - 3ms/step
Epoch 3/5
184/184 - 1s - loss: 0.2084 - accuracy: 0.9338 - val_loss: 0.1209 - val_accuracy: 0.9645 - 615ms/epoch - 3ms/step
Epoch 4/5
184/184 - 1s - loss: 0.1730 - accuracy: 0.9438 - val_loss: 0.1078 - val_accuracy: 0.9692 - 563ms/epoch - 3ms/step
Epoch 5/5
184/184 - 1s - loss: 0.1445 - accuracy: 0.9524 - val_loss: 0.0930 - val_accuracy: 0.9731 - 570ms/epoch - 3ms/step
152/152 [==============================] - 0s 3ms/step - loss: 0.0856 - accuracy: 0.9737
測試資料集的準確度 = 0.97
```

MNIST手寫辨識的轉移學習

- 因為是使用相同模型結構來訓練後5個手寫數字圖片，在作法上，我們準備凍結卷積基底的2組卷積和池化層，只訓練分類器的2個Dense全連接層來訓練後5個手寫數字圖片：



```
62 #  顯示各神經層
63 print(len(model.layers))
64 for  i  in  range(len(model.layers)):
65         print(i,  model.layers[i])
66 #  凍結上層模型
67 for  i  in  range(4):
68         model.layers[i].trainable  =  False
```

- 為了確認需凍結的神經層索引範圍，請先使用for迴圈顯示模型個神經層的資訊

```
8
0 <keras.layers.convolutional.Conv2D object at 0x7fd7f0607c90>
1 <keras.layers.pooling.MaxPooling2D object at 0x7fd7f09b8450>
2 <keras.layers.convolutional.Conv2D object at 0x7fd7e0620310>
3 <keras.layers.pooling.MaxPooling2D object at 0x7fd7e05f9090>
4 <keras.layers.core.flatten.Flatten object at 0x7fd7e05a3910>
5 <keras.layers.core.dense.Dense object at 0x7fd7e05a3a10>
6 <keras.layers.core.dropout.Dropout object at 0x7fd7e05b6f10>
7 <keras.layers.core.dense.Dense object at 0x7fd7e05b6790>
```

- 上述執行結果可以看到模型共有8層，我們需要凍結模型的0-3層
- 需要再次編譯模型，然後才能訓練第2個轉移學習的分類模型

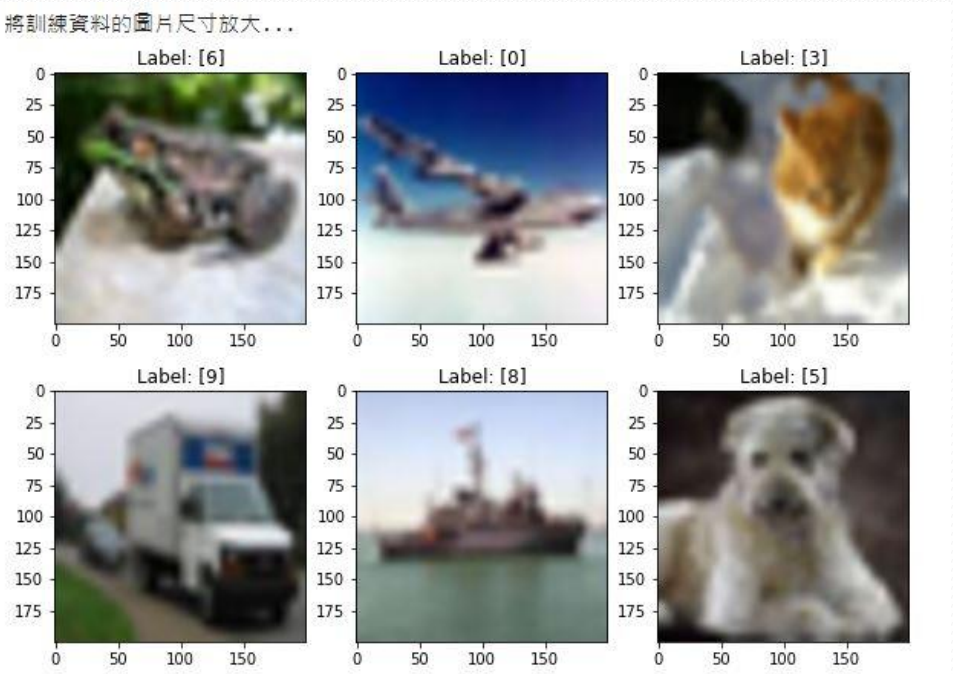
```
69 #  編譯模型
70 model.compile(loss="categorical_crossentropy",  optimizer="adam",
71                                 metrics=["accuracy"])
72 #  訓練模型
73 history  =  model.fit(X_train_gte5,  Y_train_gte5,  validation_split=0.2,
74                                             epochs=5,  batch_size=128,  verbose=2)
75 #  評估模型
76 loss,  accuracy  =  model.evaluate(X_test_gte5,  Y_test_gte5)
77 print("測試資料集的準確度  =  {:.2f}".format(accuracy))
```

調整cifar10資料集的圖片尺寸

```python
import numpy as np
from PIL import Image
from keras.datasets import cifar10
import matplotlib.pyplot as plt

# 指定亂數種子
seed = 10
np.random.seed(seed)
# 載入資料集
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
# 打亂 2 個 Numpy 陣列
def randomize(a, b):
        permutation = list(np.random.permutation(a.shape[0]))
        shuffled_a = a[permutation]
        shuffled_b = b[permutation]

        return shuffled_a, shuffled_b

X_train, Y_train = randomize(X_train, Y_train)
# 取出部分的訓練資料
X_train = X_train[:500]
Y_train = Y_train[:500]
# 將訓練資料的圖片尺寸放大
print("將訓練資料的圖片尺寸放大...")
X_train_new = np.array(
    [np.asarray(Image.fromarray(X_train[i]).resize(
                    (200, 200))) for i in range(0, len(X_train))])
# 繪出6張圖片
fig = plt.figure(figsize=(10,7))
sub_plot= 230
for i in range(0, 6):
        ax = plt.subplot(sub_plot+i+1)
        ax.imshow(X_train_new[i], cmap="binary")
        ax.set_title("Label: " + str(Y_train[i]))

plt.show()
```

調整cifar10資料集的圖片尺寸

- 雖然cifar10訓練資料集有50,000張圖片，受限於電腦的記憶體容量，Python程式無法將全部圖片都在記憶體中放大成(200, 200)來建立訓練資料集，我們只能取出部分cifar10圖片來建立訓練資料集，在第15-5-2節是取出前5,000張；第15-5-3節取出前7,500張。
- Python程式是在說明如何調整cifar10資料集的圖片尺寸，我們只準備取出部分cifar10資料集的圖片和將圖片在記憶體中放大成(200, 200)，程式是使用和Ch13_5_1.py相同的方式來打亂NumPy陣列後，取出前500張訓練資料集的圖片(以500張為例)：
- 將訓練資料集的圖片尺寸放大成(200, 200)



cifar10資料集
ResNet50預訓練模型的轉移學習

```python
import numpy as np
from keras.datasets import cifar10
from keras.applications.resnet import ResNet50, preprocess_input
from keras.models import Sequential
from keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.utils import to_categorical
from PIL import Image

# 指定亂數種子
seed = 10
np.random.seed(seed)
# 載入資料集
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
# 打亂 2 個 Numpy 陣列
def randomize(a, b):
        permutation = list(np.random.permutation(a.shape[0]))
        shuffled_a = a[permutation]
        shuffled_b = b[permutation]

        return shuffled_a, shuffled_b

X_train, Y_train = randomize(X_train, Y_train)
X_test, Y_test = randomize(X_test, Y_test)
# 取出10%訓練, 10%測試
X_train = X_train[:5000]
Y_train = Y_train[:5000]
X_test = X_test[:1000]
Y_test = Y_test[:1000]
# One-hot編碼
Y_train = to_categorical(Y_train, 10)
Y_test = to_categorical(Y_test, 10)
# 載入 ResNet50 模型
resnet_model = ResNet50(weights="imagenet",
                                    include_top=False,
                                    input_shape=(200, 200, 3
# 調整X_train的圖片尺寸
print("調整X_train的圖片尺寸...")
X_train_new = np.array(
    [np.asarray(Image.fromarray(X_train[i]).resize(
                (200, 200))) for i in range(0, len(X_train))])
X_train_new = X_train_new.astype("float32")
# 訓練資料的資料前處理
train_input = preprocess_input(X_train_new)
# 使用 ResNet50 模型預測訓練資料的特徵資料
print("使用 ResNet50 模型預測訓練資料的特徵資料...")
train_features = resnet_model.predict(train_input)
# 調整X_test的圖片尺寸
print("調整X_test的圖片尺寸...")
X_test_new = np.array(
    [np.asarray(Image.fromarray(X_test[i]).resize(
                (200, 200))) for i in range(0, len(X_test))])
X_test_new = X_test_new.astype("float32")
# 測試資料的資料前處理
test_input = preprocess_input(X_test_new)
# 使用 ResNet50 模型預測測試資料的特徵資料
print("使用 ResNet50 模型預測測試資料的特徵資料...")
test_features = resnet_model.predict(test_input)
# 定義模型
model = Sequential()
model.add(GlobalAveragePooling2D(
            input_shape=train_features.shape[1:]))
model.add(Dropout(0.5))
model.add(Dense(10, activation="softmax"))
# 編譯模型
model.compile(loss="categorical_crossentropy", optimizer="adam",
                        metrics=["accuracy"])
# 訓練模型
history = model.fit(train_features, Y_train,
                                    validation_data=(test_features, Y_test),
                                    epochs=14, batch_size=32, verbose=2)
# 評估模型
print("\nTesting ...")
loss, accuracy = model.evaluate(test_features, Y_test)
print("測試資料集的準確度 = {:.2f}".format(accuracy))
# 顯示圖表來分析模型的訓練過程
import matplotlib.pyplot as plt
# 顯示訓練和驗證損失
loss = history.history["loss"]
epochs = range(1, len(loss)+1)
val_loss = history.history["val_loss"]
plt.plot(epochs, loss, "bo-", label="Training Loss")
plt.plot(epochs, val_loss, "ro--", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
# 顯示訓練和驗證準確度
acc = history.history["accuracy"]
epochs = range(1, len(acc)+1)
val_acc = history.history["val_accuracy"]
plt.plot(epochs, acc, "bo-", label="Training Accuracy")
plt.plot(epochs, val_acc, "ro--", label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

預訓練模型的轉移學習

- Python程式使用Keras預訓練模型建立轉移學習有兩種方法：
  - 第一種方法：使用預訓練模型的卷積基底(include_top=False)呼叫predict()函式將訓練資料集輸出成NumPy陣列的特徵資料後，建立一個全新的分類模型，然後使用卷積基底輸出的特徵資料作為輸入資料來訓練模型，請注意！這種方法不能使用圖片增強。
  - 第二種方法：直接在預訓練模型的卷積基底新增Dense層的分類器，然後使用訓練資料集進行訓練，因為資料會經過整個卷積基底，需要花費更多計算和訓練時間，但是可以使用圖片增強。

ResNet50預訓練模型的轉移學習

- 本節ResNet50預訓練模型的轉移學習是使用第一種方法，Python程式使用前面小節的方法取出前5,000張圖片，即10%訓練和10%測試資料集。
- 接著載入ResNet50模型，include_top參數值是False，input_shape參數是輸入圖片的形狀：

```
resnet_model = ResNet50(weights="imagenet", include_top=False,
 input_shape=(200, 200, 3))
```

- 使用 ResNet50 模型預測訓練資料集來輸出train_features特徵資料：

```
X_train_new = np.array( [np.asarray(Image.fromarray
(X_train[i]).resize((200, 200))) for i in range(0, len(X_train))])
X_train_new = X_train_new.astype("float32")
train_input = preprocess_input(X_train_new)
train_features = resnet_model.predict(train_input
```

ResNet50預訓練模型的轉移學習

- 同樣方式可以使用ResNet50模型預測測試資料集來輸出test_features特徵資料：

```
X_test_new = np.array([np.asarray(Image.fromarray(X_test[i]).resize(
        (200, 200))) for i in range(0, len(X_test))])
X_test_new = X_test_new.astype("float32")
test_input = preprocess_input(X_test_new)
test_features = resnet_model.predict(test_input)
```

- 我們可以定義一個分類器的神經網路，如下所示：

```
model = Sequential() model.add(GlobalAveragePooling2D(
input_shape=train_features.shape[1:]))
model.add(Dropout(0.5))
model.add(Dense(10, activation="softmax"))
```

cifar10資料集
MobileNet預訓練模型的轉移學習

```python
import numpy as np
from keras.datasets import cifar10
from keras.applications.mobilenet import MobileNet, preprocess_input
from keras.models import Sequential
from keras.layers import Dense, Dropout, GlobalAveragePooling2D
from tensorflow.keras.utils import to_categorical
from PIL import Image

# 指定亂數種子
seed = 10
np.random.seed(seed)
# 載入資料集
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()
# 打亂 2 個 Numpy 陣列
def randomize(a, b):
        permutation = list(np.random.permutation(a.shape[0]))
        shuffled_a = a[permutation]
        shuffled_b = b[permutation]

        return shuffled_a, shuffled_b

X_train, Y_train = randomize(X_train, Y_train)
X_test, Y_test = randomize(X_test, Y_test)
# 取出15%訓練, 20%測試
X_train = X_train[:7500]
Y_train = Y_train[:7500]
X_test = X_test[:2000]
Y_test = Y_test[:2000]
# One-hot編碼
Y_train = to_categorical(Y_train, 10)
Y_test = to_categorical(Y_test, 10)
# 載入 ResNet50 模型
mobilenet_model = MobileNet(weights="imagenet",
                                                include_top=False,
                                                input_shape=(224, 224, 3))
# 調整X_train的圖片尺寸
print("調整X_train的圖片尺寸...")
X_train_new = np.array(
    [np.asarray(Image.fromarray(X_train[i]).resize(
                (224, 224))) for i in range(0, len(X_train))])
X_train_new = X_train_new.astype("float32")
# 訓練資料的資料前處理
train_input = preprocess_input(X_train_new)
# 調整X_test的圖片尺寸
print("調整X_test的圖片尺寸...")
X_test_new = np.array(
    [np.asarray(Image.fromarray(X_test[i]).resize(
                (224, 224))) for i in range(0, len(X_test))])
X_test_new = X_test_new.astype("float32")
# 測試資料的資料前處理
test_input = preprocess_input(X_test_new)
# 定義模型
model = Sequential()
model.add(mobilenet_model)
model.add(Dropout(0.5))
model.add(GlobalAveragePooling2D())
model.add(Dropout(0.5))
model.add(Dense(10, activation="softmax"))
model.summary()      # 顯示模型摘要資訊
# 凍結上層模型
mobilenet_model.trainable = False
# 編譯模型
model.compile(loss="categorical_crossentropy", optimizer="adam",
                        metrics=["accuracy"])
# 訓練模型
history = model.fit(train_input, Y_train,
                                    validation_data=(test_input, Y_test),
                                    epochs=17, batch_size=32, verbose=2)
# 評估模型
print("\nTesting ...")
loss, accuracy = model.evaluate(test_input, Y_test)
print("測試資料集的準確度 = {:.2f}".format(accuracy))
# 顯示圖表來分析模型的訓練過程
import matplotlib.pyplot as plt
# 顯示訓練和驗證損失
loss = history.history["loss"]
epochs = range(1, len(loss)+1)
val_loss = history.history["val_loss"]
plt.plot(epochs, loss, "bo-", label="Training Loss")
plt.plot(epochs, val_loss, "ro--", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
# 顯示訓練和驗證準確度
acc = history.history["acc"]
epochs = range(1, len(acc)+1)
val_acc = history.history["val_acc"]
plt.plot(epochs, acc, "bo-", label="Training Acc")
plt.plot(epochs, val_acc, "ro--", label="Validation Acc")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

MobileNet預訓練模型的轉移學習

- MobileNet預訓練模型的轉移學習是使用前面小節說明的第二種方法來建立轉移學習，我們準備直接在MobileNet預訓練模型的卷積基底新增全新的Dense層分類器。
- Python程式是使用相同方式來取出和放大訓練和測試資料集的cifar10圖片，分別取出7,500張和2,000張。

MobileNet預訓練模型的轉移學習

- 定義模型的程式碼：

```
model = Sequential()
model.add(mobilenet_model)
model.add(Dropout(0.5))
model.add(GlobalAveragePooling2D())
model.add(Dropout(0.5))
model.add(Dense(10, activation="softmax"))
model.summary()
```

- 然後，我們需要凍結MobileNet預訓練模型的卷積基底，也就是在訓練時不更新這些神經層的權重：

```
mobilenet_model.trainable = False
```

End !