電腦視覺深度學習實作與應用
期末專題報告

電機所 卓冠廷

Contents

資料讀取與前處理

模型架構探討

模型訓練過程

模型比較與分析

分不開的菜餚

資料讀取

```
from  google.colab  import  drive
import  os
drive.mount('/content/drive')
os.chdir('/tmp')
#存資料到temp
!unzip  /content/drive/MyDrive/電腦視覺/Final_Project/train.zip
!unzip  /content/drive/MyDrive/電腦視覺/Final_Project/test.zip
```

解壓到自身雲端易造成部分資料遺失

資料前處理

各類別的資料平衡

糖醋雞丁	606	糖醋雞丁	690
福山萵苣	104	福山萵苣	690
有機小松菜	513	有機小松菜	690
葡萄	559	葡萄	690
白菜滷	476	白菜滷	690
木瓜	648	木瓜	690
香蕉	640	香蕉	690
大陸妹	573	大陸妹	690
麻油雞	613	麻油雞	690
蒜泥白肉	518	蒜泥白肉	690
白米飯	503	白米飯	690
滷雞腿	672	滷雞腿	690
什錦炒麵	628	什錦炒麵	690
咖哩雞	609	咖哩雞	690
橘子	617	橘子	690
空心菜	656	空心菜	690
馬鈴薯燉肉	553	馬鈴薯燉肉	690
油菜	586	油菜	690
麥克雞塊	524	麥克雞塊	690
紅蘿蔔炒蛋	620	紅蘿蔔炒蛋	690
客家小炒	611	客家小炒	690
瓜仔肉	643	瓜仔肉	690
青江菜	479	青江菜	690
芥藍菜	333	芥藍菜	690
小番茄	681	小番茄	690
棗子	690	棗子	690
麻婆豆腐	569	麻婆豆腐	690
柳丁	572	柳丁	690
義大利麵	659	義大利麵	690
沙茶肉片	511	沙茶肉片	690

資料前處理

Check_img()
→隨機取某種類一張照片
→確認是否為.jpg

ImageDataGenerator
→Data augmentation

.flow()
→生成單張新圖片

.next()
→生成下一張新圖片

```
datagen = ImageDataGenerator(
        rotation_range=90,
        width_shift_range=0.1,
        height_shift_range=0.1,
        shear_range=0.1,
        zoom_range=0.1,
        horizontal_flip=True,
        fill_mode='nearest')
for i in range(len(class_list)):
    imagelist= os.listdir('/tmp/train/'+class_list[i])
    img_len=len(imagelist)
    if len(imagelist)==0 or len(imagelist)==max_len:
        continue
    else:
        for j in range(max_len-len(imagelist)):
            rand=check_img(imagelist,img_len)  #隨機取某種類中的一張照片
            path='/tmp/train/'+str(class_list[i])+'/'+str(imagelist[rand])
            tem_img=load_img(path)
            tem_img=img_to_array(tem_img)
            tem_img = tem_img.reshape((1,) + tem_img.shape)
            save_path='/tmp/train/'+str(class_list[i])
            #  產生新照片
            gener=datagen.flow(tem_img,
                    batch_size=64,
                    shuffle=False,
                    save_to_dir=save_path,
                    save_prefix='trans_'+str(j),
                    save_format='jpg')
            gener.next()
```

資料前處理

訓練測試資料設置

```
from tensorflow import keras
from tensorflow.keras.preprocessing import image_dataset_from_directory
import tensorflow.compat.v2 as tf

IMG_SIZE = 180
NUM_CLASSES = 50
batch_size = 64
base_dir='/tmp/train'
train_ds = image_dataset_from_directory(
        directory=base_dir,
        validation_split=0.2,
        subset='training',
        seed=10,
        batch_size=batch_size,
        image_size=(IMG_SIZE,IMG_SIZE))
val_ds = image_dataset_from_directory(
        directory=base_dir,
        validation_split=0.2,
        seed=10,
        subset='validation',
        batch_size=batch_size,
        image_size=(IMG_SIZE,IMG_SIZE))
class_names = train_ds.class_names
```

```
Found 34498 files belonging to 50 classes.
Using 27599 files for training.
Found 34498 files belonging to 50 classes.
Using 6899 files for validation.
```

模型架構探討

Xception 架構(改良自InceptionV3)

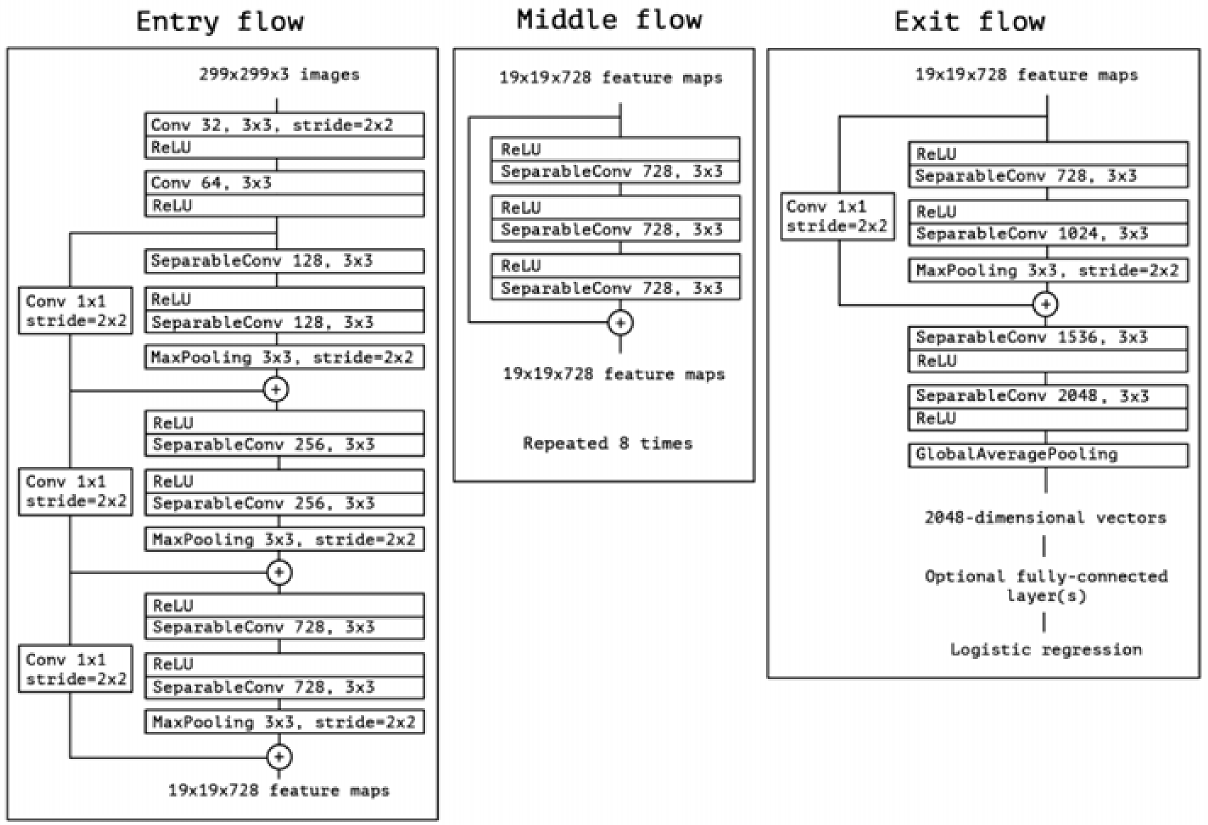
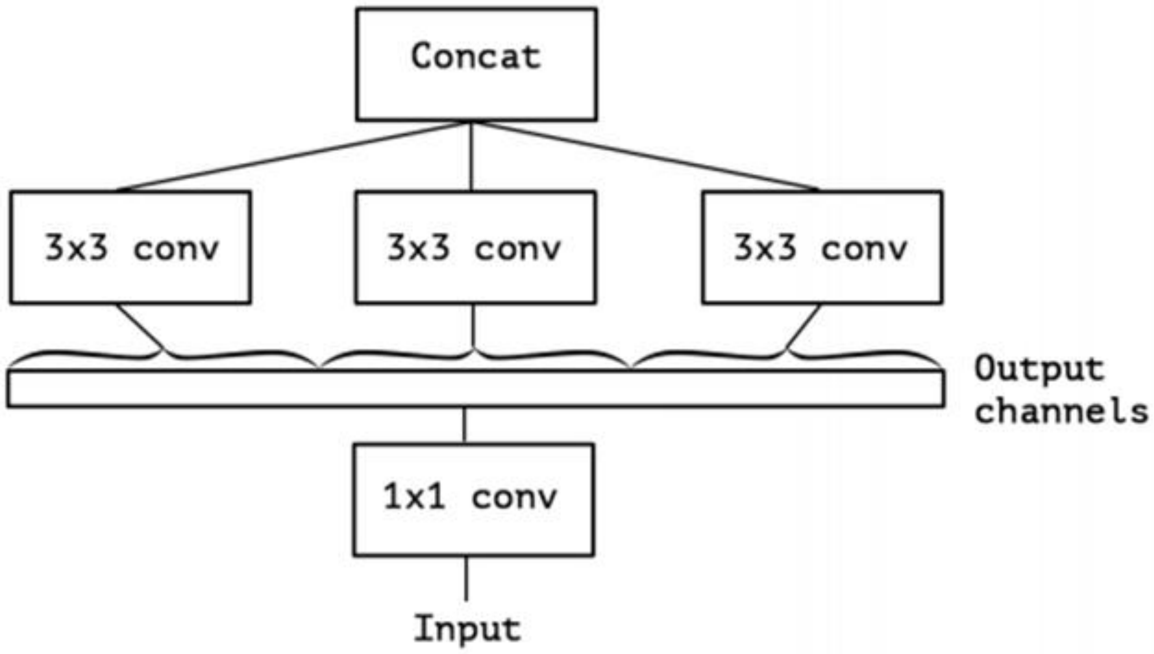


Figure 5. The Xception architecture: the data first goes through the entry flow, then through the middle flow which is repeated eight times, and finally through the exit flow. Note that all Convolution and SeparableConvolution layers are followed by batch normalization [7] (not included in the diagram).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

模型架構探討

Inception module(InceptionV3)

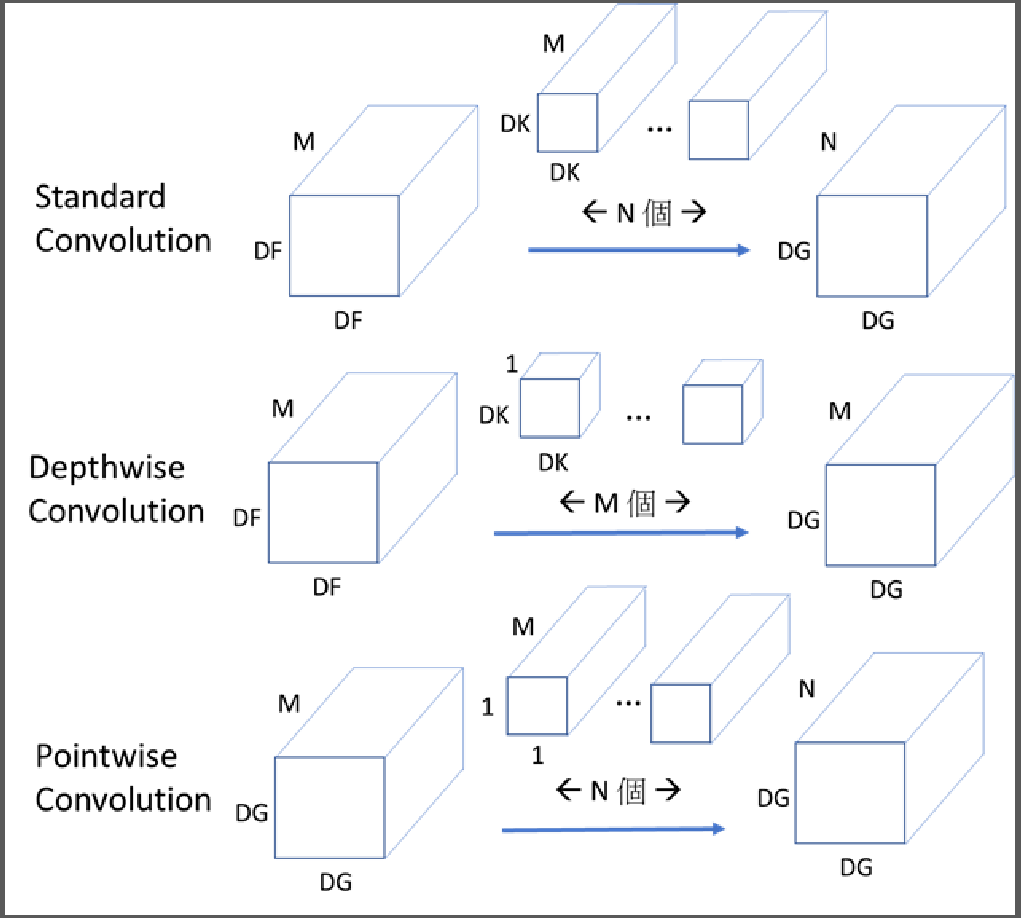
Figure 3. A strictly equivalent reformulation of the simplified Inception module.



先使用 1x1 卷積層，再將輸出分割給三個 3x3 卷積層做輸入
(每個 3x3 卷積層都各自為 1/3 的維度)

模型架構探討

Depthwise separable convolution

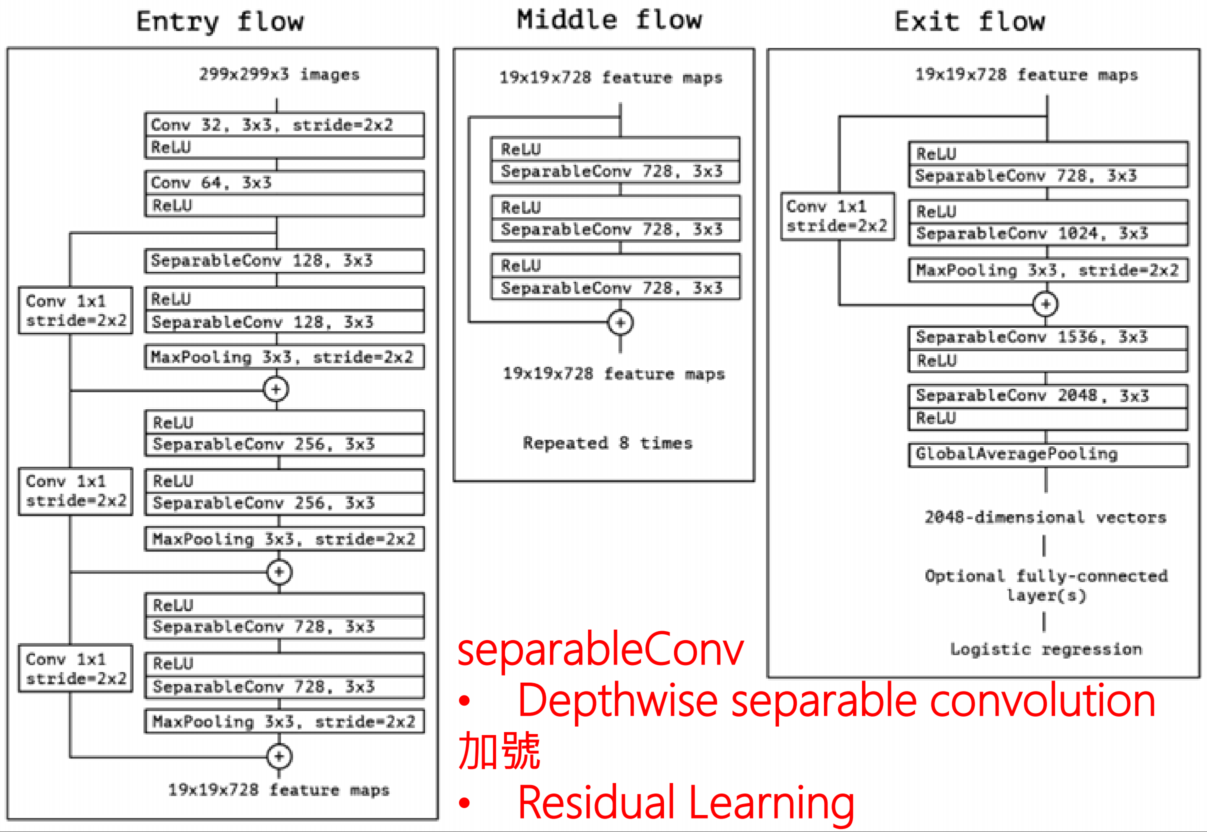


Xception 是先進行 1x1 卷積運算，再對通道卷積
卷積之間會通過ReLU

模型架構探討

Xception 架構(改良自InceptionV3)

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).

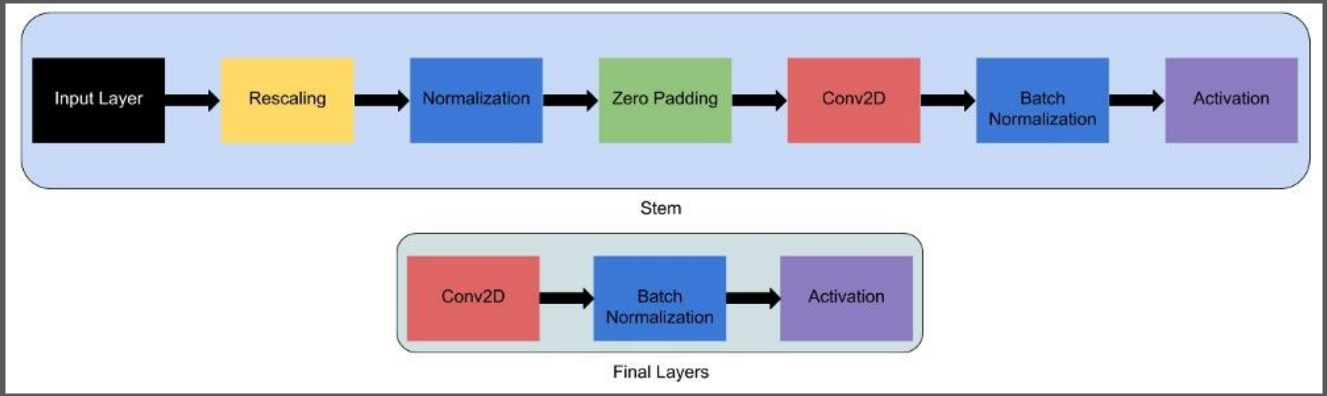


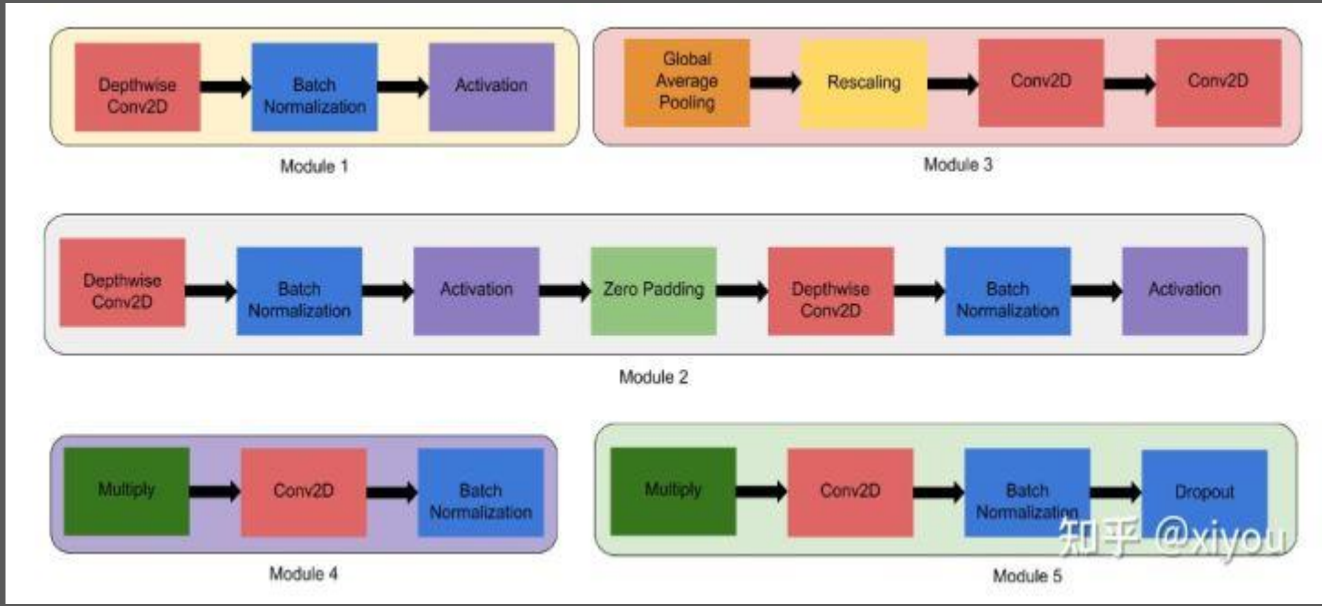
separableConv
- Depthwise separable convolution

加號
- Residual Learning

模型架構探討

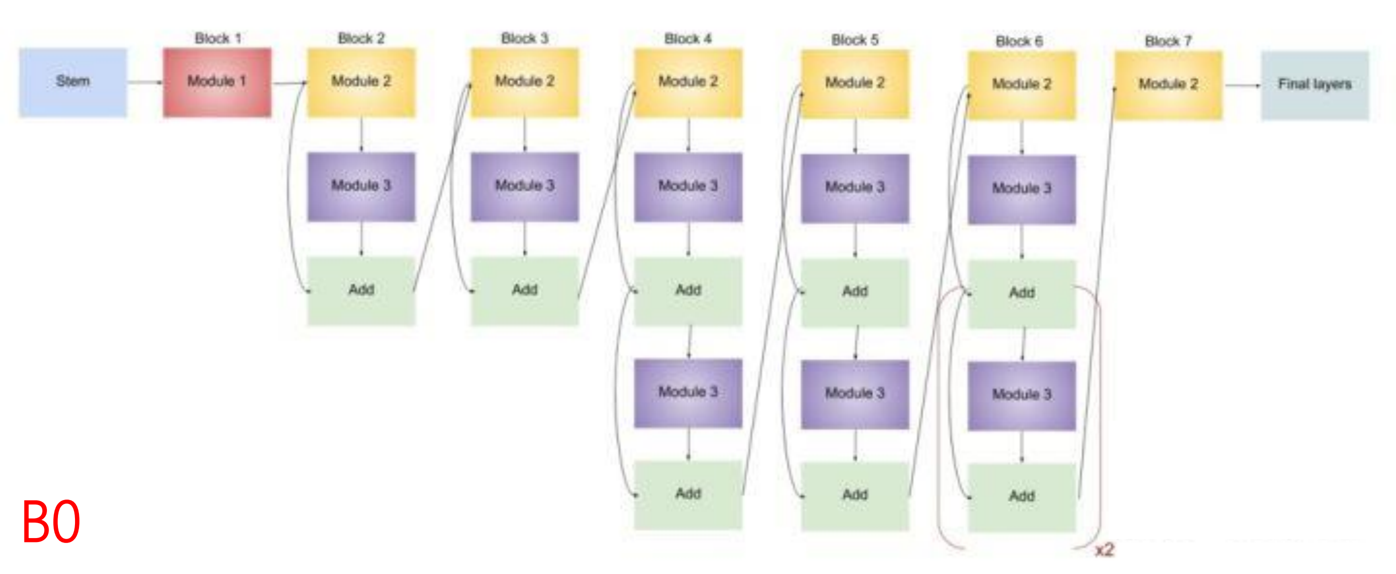
EfficientNet B0-B7架構(237-813層)

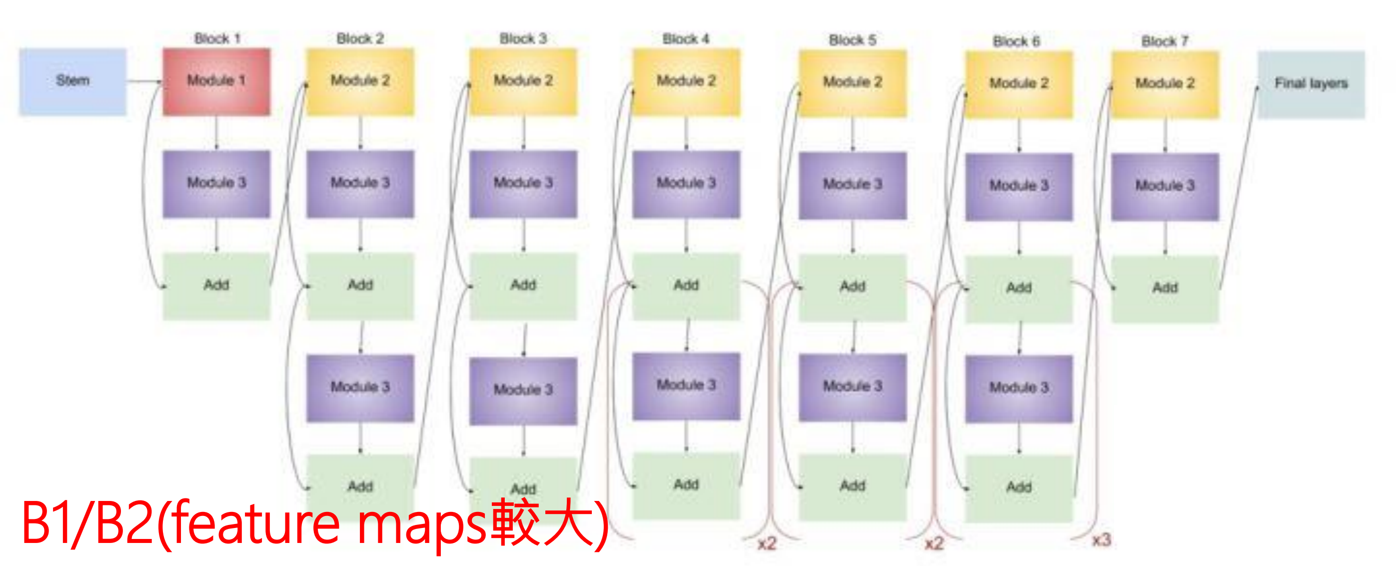




模型架構探討

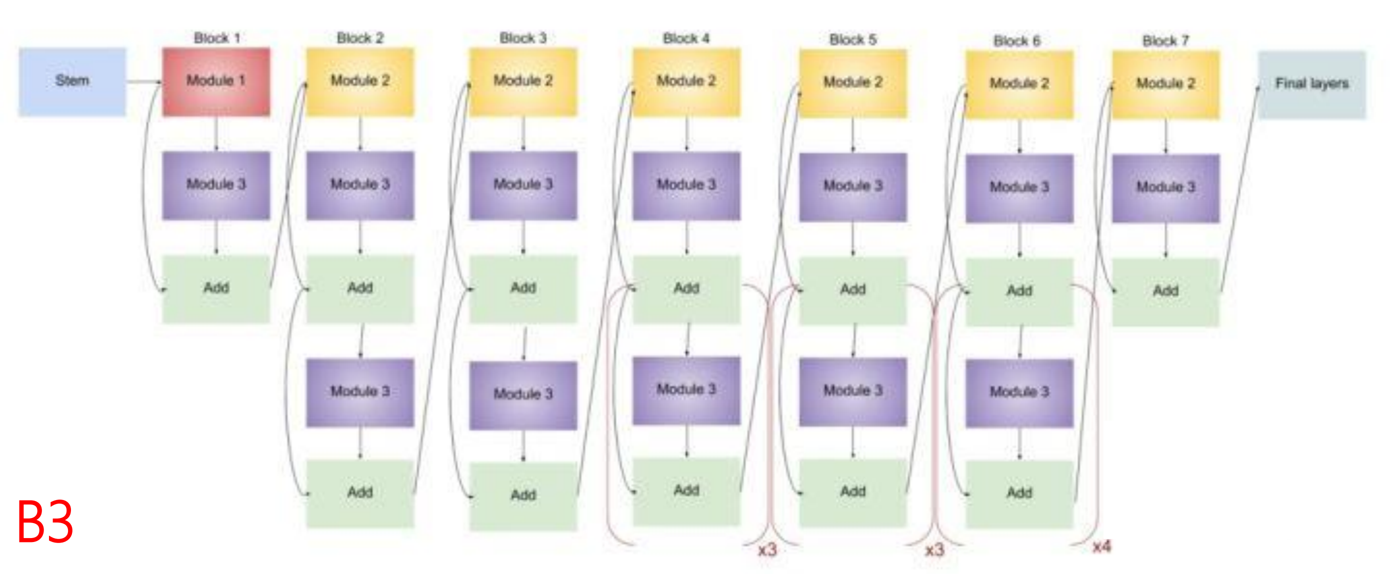
EfficientNetB0/B1/B2架構

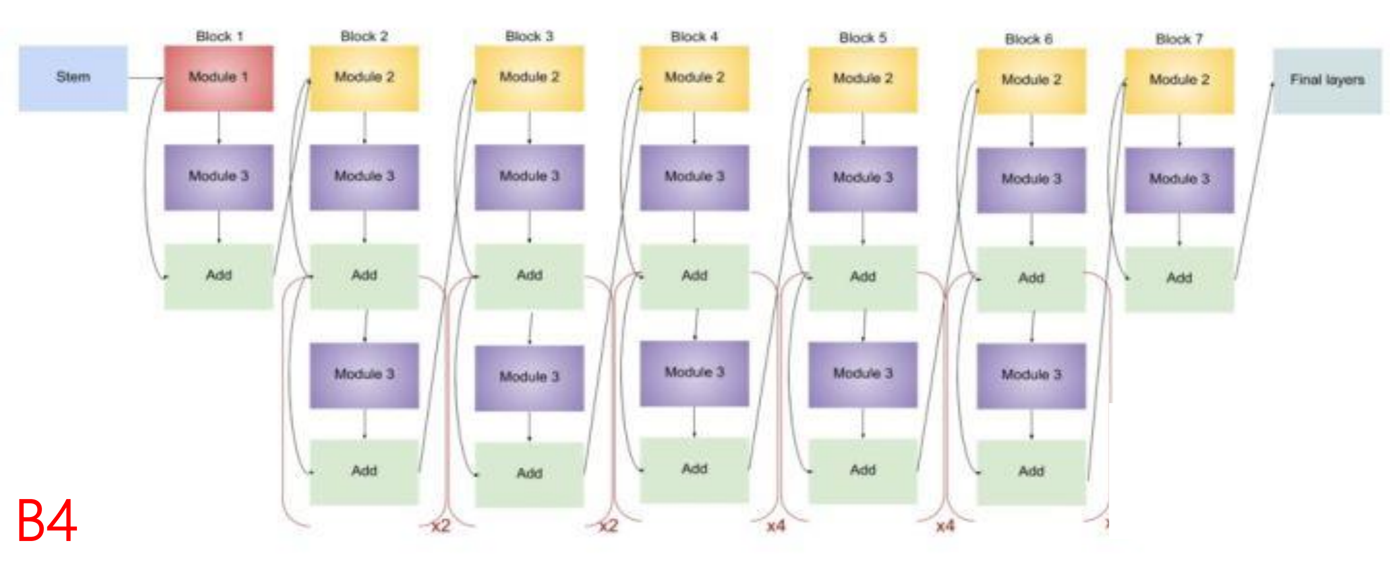




模型架構探討

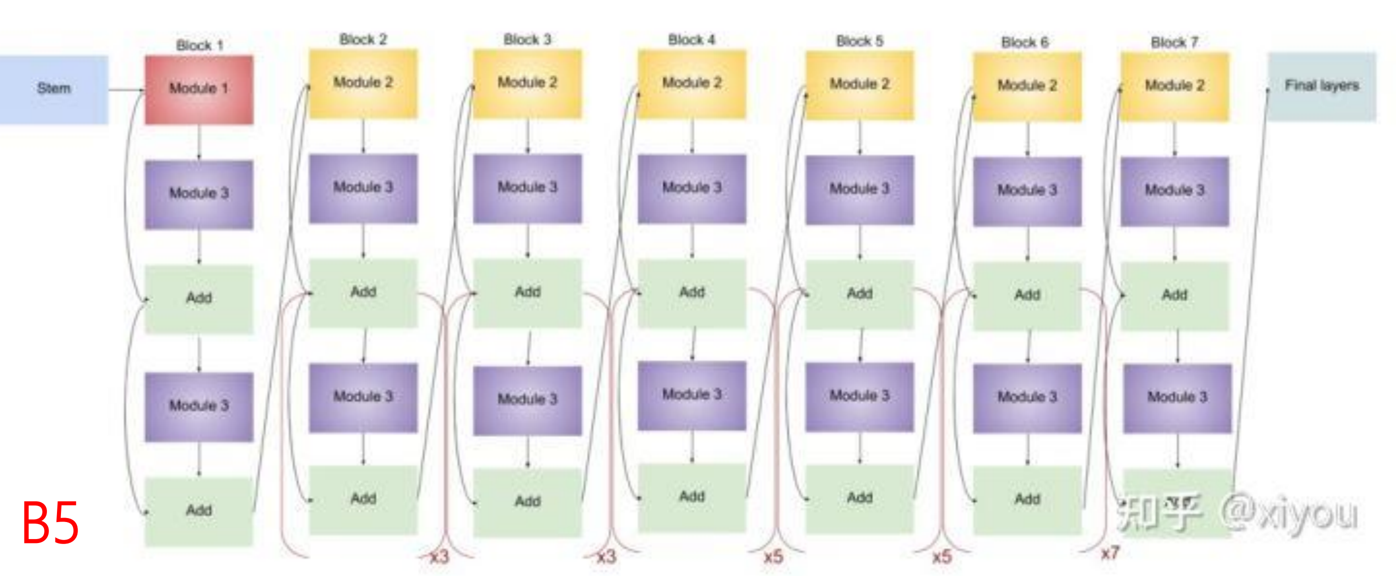
EfficientNetB3/B4架構

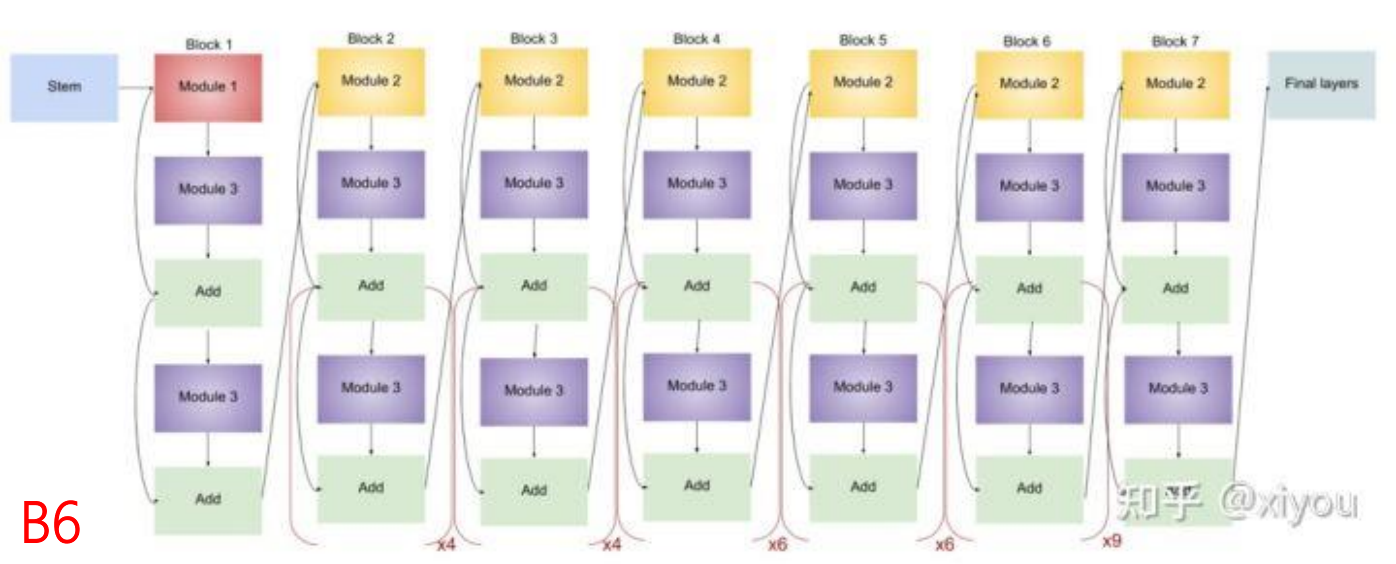




模型架構探討

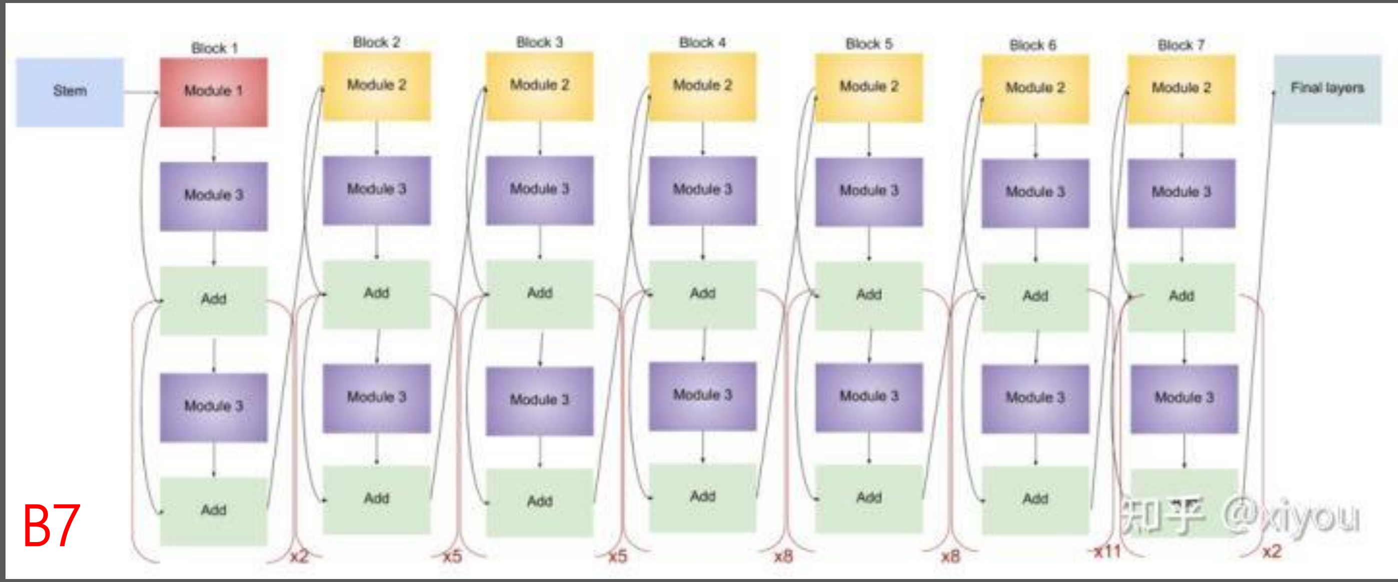
EfficientNetB5/B6架構





模型架構探討

EfficientNetB7架構



核心理念-複合縮放(Compound Scaling)

同時調整網路深度、寬度、解析度這三種的縮放方法

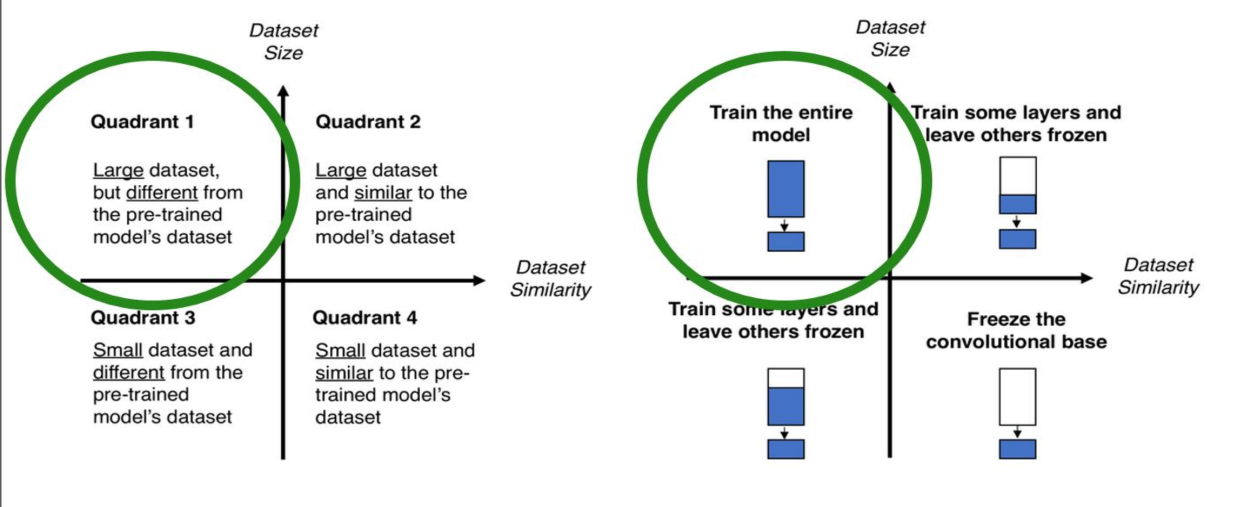
$$
\begin{aligned}
\text{depth: } & d = \alpha^{\phi} \\
\text{width: } & w = \beta^{\phi} \\
\text{resolution: } & r = \gamma^{\phi} \\
\text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\
& \alpha \geq 1, \beta \geq 1, \gamma \geq 1
\end{aligned}
$$

模型訓練過程

Data augmentation

```python
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers

data_augmentation = Sequential(
    [
        preprocessing.RandomFlip("horizontal"),
        preprocessing.RandomTranslation(height_factor=0.1, width_factor=0.1),
        preprocessing.RandomRotation(0.1),
        preprocessing.RandomZoom(0.1),
        preprocessing.RandomContrast(0.1),
    ],
    name="img_augmentation",
)
```

Transfer Learning



模型訓練過程

Freeze base model

Xception

```
Epoch 1/3
432/432 [==============================] - 62s 135ms/step - loss: 4.4032 - accuracy: 0.0331 - top5 accuracy: 0.1426 - val_loss: 4.0126 - val_accuracy: 0.0487 - val_top5 accuracy: 0.1809
Epoch 2/3
432/432 [==============================] - 59s 134ms/step - loss: 4.1677 - accuracy: 0.0457 - top5 accuracy: 0.1869 - val_loss: 3.9703 - val_accuracy: 0.0568 - val_top5 accuracy: 0.2155
Epoch 3/3
432/432 [==============================] - 58s 133ms/step - loss: 4.0614 - accuracy: 0.0536 - top5 accuracy: 0.2122 - val_loss: 3.9094 - val_accuracy: 0.0628 - val_top5 accuracy: 0.2384
```

EfficientNetB0

```
Epoch 1/3
432/432 [==============================] - 46s 91ms/step - loss: 4.5455 - accuracy: 0.0460 - top5 accuracy: 0.1835 - val_loss: 3.5081 - val_accuracy: 0.1232 - val_top5 accuracy: 0.3590
Epoch 2/3
432/432 [==============================] - 39s 90ms/step - loss: 3.8115 - accuracy: 0.1086 - top5 accuracy: 0.3341 - val_loss: 2.9446 - val_accuracy: 0.2286 - val_top5 accuracy: 0.5405
Epoch 3/3
432/432 [==============================] - 39s 88ms/step - loss: 3.3534 - accuracy: 0.1672 - top5 accuracy: 0.4518 - val_loss: 2.5583 - val_accuracy: 0.3085 - val_top5 accuracy: 0.6446
```

Without freeze base model 勝

Xception

```
Epoch 1/20
432/432 [==============================] - 420s 921ms/step - loss: 2.8748 - accuracy: 0.2742 - top5 accuracy: 0.5705 - val_loss: 1.5493 - val_accuracy: 0.5173 - val_top5 accuracy: 0.8706
Epoch 2/20
432/432 [==============================] - 400s 924ms/step - loss: 1.4556 - accuracy: 0.5502 - top5 accuracy: 0.8836 - val_loss: 1.1022 - val_accuracy: 0.6418 - val_top5 accuracy: 0.9335
Epoch 3/20
432/432 [==============================] - 398s 919ms/step - loss: 1.1296 - accuracy: 0.6384 - top5 accuracy: 0.9298 - val_loss: 0.9304 - val_accuracy: 0.6929 - val_top5 accuracy: 0.9491
```

EfficientNetB0

```
Epoch 1/20
432/432 [==============================] - 212s 436ms/step - loss: 3.8214 - accuracy: 0.1353 - top5 accuracy: 0.3501 - val_loss: 2.5092 - val_accuracy: 0.3238 - val_top5 accuracy: 0.6471
Epoch 2/20
432/432 [==============================] - 186s 428ms/step - loss: 2.2718 - accuracy: 0.3719 - top5 accuracy: 0.7165 - val_loss: 1.8211 - val_accuracy: 0.4663 - val_top5 accuracy: 0.8066
Epoch 3/20
432/432 [==============================] - 186s 429ms/step - loss: 1.6516 - accuracy: 0.5057 - top5 accuracy: 0.8459 - val_loss: 1.4554 - val_accuracy: 0.5521 - val_top5 accuracy: 0.8746
```

模型訓練過程

Optimizer

選用Adamax
→相較Adam可以節省運算資源
Optimizer比較

Optimizer	特點
SGD	• 有機會跳出目前局部收斂進而進到另一個局部收斂而得到最小值，而得到全局最小值 • 需自行設定learning rate，較難選擇到合適的learning rate • 會造成loss function有嚴重的震蕩 • 需要較長時間收斂至最小值
Momentum	• 能夠在相關方向加速SGD，抑制SGD的嚴重震蕩，進而加快收斂 • 需自行設定learning rate與β，有可能會使參數的移動方向偏移梯度下分的方向，進而導至沒有那麼快速的收斂
AdaGrad	• 能夠自動調整learning rate，進而調整收斂 • 適合處理稀疏梯度 • 依然需要人工設置一個全局的learning rate • 後期，分母梯度平方的累加會越來越大，會使梯度趨近於0，使得訓練結束
Adam	• 結合了AdaGrad與Momentum的優點 • 適用於大數據集和高維空間的資料 • 目前最常使用的一個Optimizer

模型訓練過程

Xception

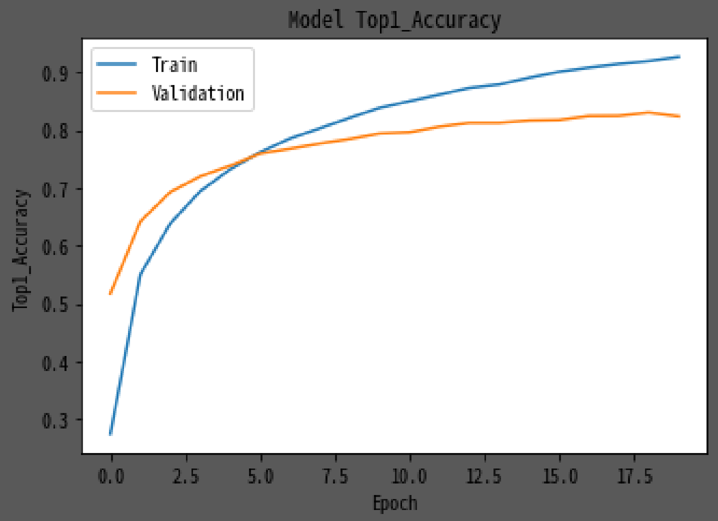
- Optimizer：Adamax
- Image size：180
- Learning rate：1e-4
- GlobalAveragePooling2D
- BatchNormalization

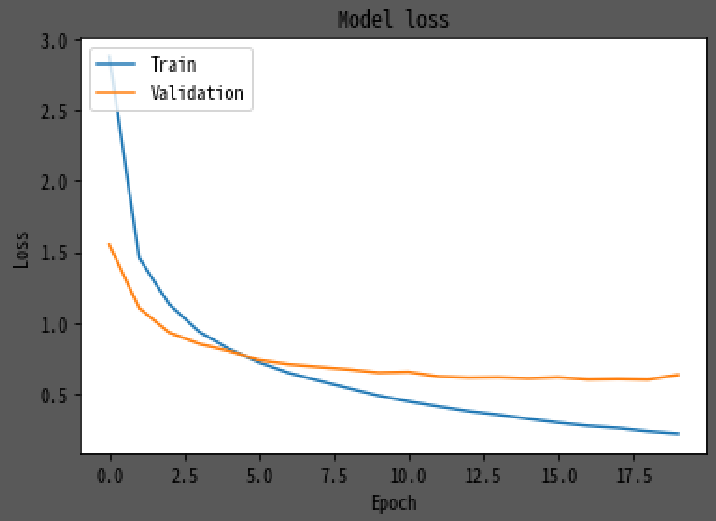
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 180, 180, 3)]     0

 img_augmentation (Sequentia  (None, 180, 180, 3)      0
 l)

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

 dropout (Dropout)           (None, 6, 6, 2048)        0

 avg_pool (GlobalAveragePool  (None, 2048)             0
 ing2D)

 batch_normalization_4 (Batc  (None, 2048)             8192
 hNormalization)

 dropout_1 (Dropout)         (None, 2048)              0

 pred (Dense)                (None, 50)                102450

=================================================================
Total params: 20,972,122
Trainable params: 20,913,498
Non-trainable params: 58,624
_________________________________________________________________
```

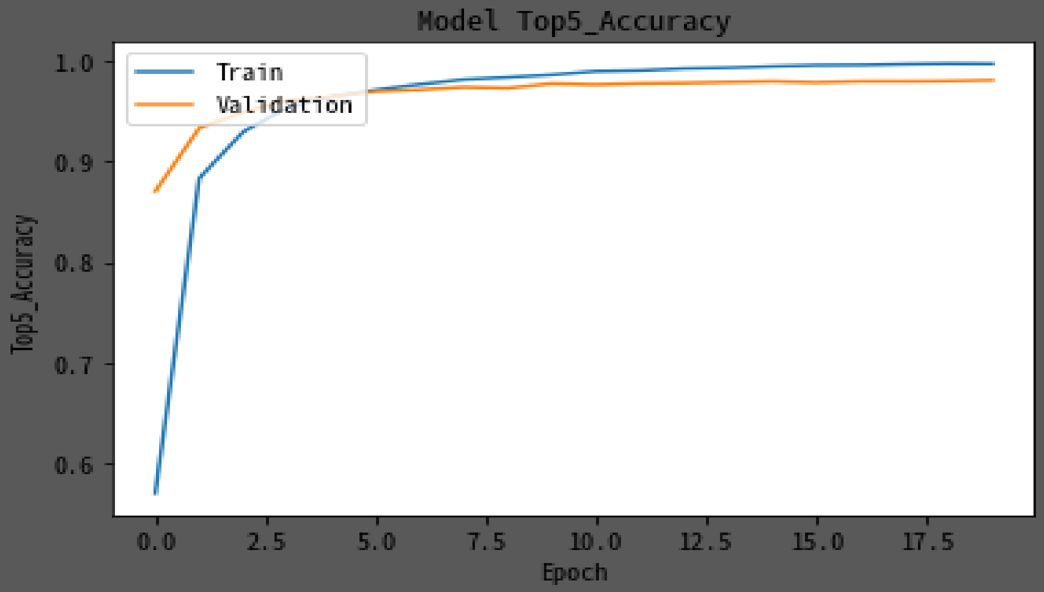
```python
from tensorflow.keras.applications.xception import Xception
from tensorflow.keras.models import Model
from tensorflow.keras import optimizers

inputs = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = data_augmentation(inputs)
Xception = Xception(include_top=False, weights='imagenet')
x=Xception(x)
x = layers.Dropout(0.5)(x)
x = layers.GlobalAveragePooling2D(name="avg_pool")(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.3)(x)
outputs = layers.Dense(NUM_CLASSES, activation="softmax", name="pred")(x)
# # freeze the weight
# Xception.trainable = False
model = Model(inputs,outputs)
model.compile(loss='sparse_categorical_crossentropy',
         optimizer=optimizers.Adamax(learning_rate=1e-4),
         metrics=['accuracy',tf.keras.metrics.SparseTopKCategoricalAccuracy(k=5, name="top5 accuracy")]
         )
model.summary()
```

模型訓練過程

Xception







模型訓練過程

EfficientNetB0

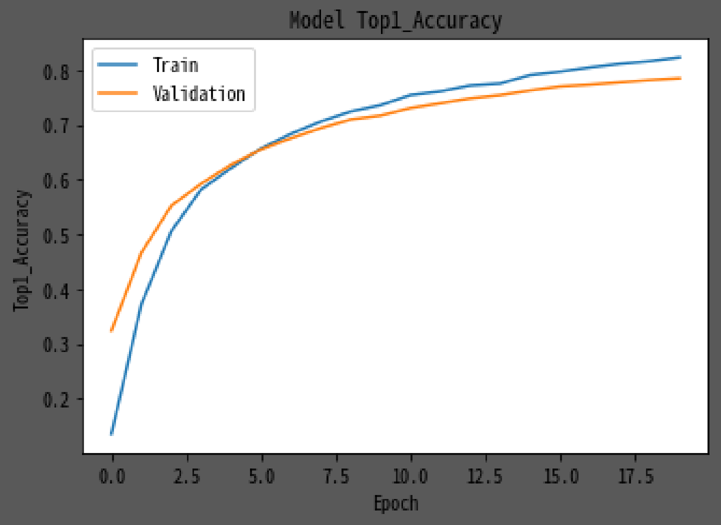
- Optimizer：Adamax
- Image size：180
- Learning rate：1e-4
- GlobalAveragePooling2D
- BatchNormalization

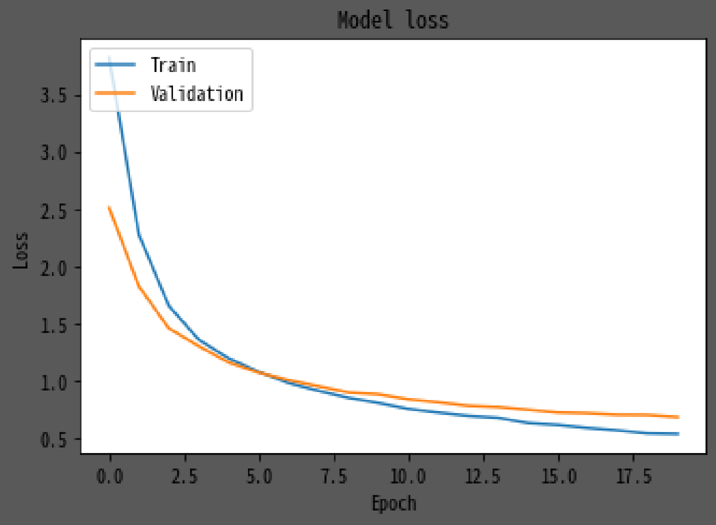
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 180, 180, 3)]     0

 img_augmentation (Sequentia  (None, 180, 180, 3)      0
 l)

 efficientnetb0 (Functional)  (None, None, None, 1280)  4049571

 dropout (Dropout)           (None, 5, 5, 1280)        0

 avg_pool (GlobalAveragePool  (None, 1280)             0
 ing2D)

 batch_normalization (BatchN  (None, 1280)             5120
 ormalization)

 dropout_1 (Dropout)         (None, 1280)              0

 pred (Dense)                (None, 50)                64050

=================================================================
Total params: 4,118,741
Trainable params: 4,074,158
Non-trainable params: 44,583
_________________________________________________________________
```

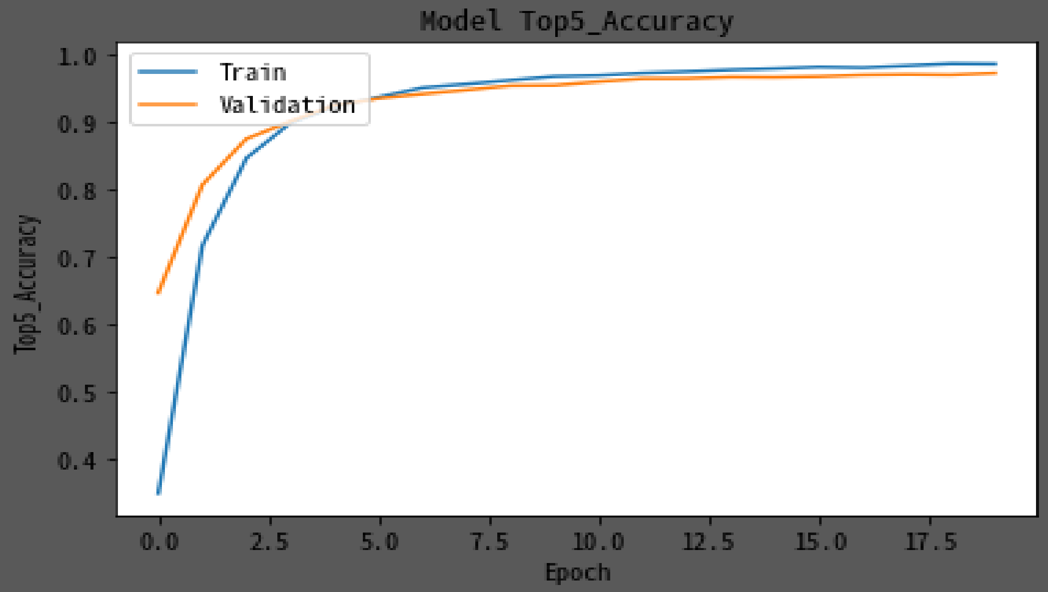
```python
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.models import Model
from tensorflow.keras import optimizers

inputs = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = data_augmentation(inputs)
EfficientNetB0 = EfficientNetB0(include_top=False, weights='imagenet')
x=EfficientNetB0(x)
x = layers.Dropout(0.5)(x)
x = layers.GlobalAveragePooling2D(name="avg_pool")(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.3)(x)
outputs = layers.Dense(NUM_CLASSES, activation="softmax", name="pred")(x)
# # freeze the weight
# EfficientNetB0.trainable = False
model = Model(inputs,outputs)
model.compile(loss='sparse_categorical_crossentropy',
          optimizer=optimizers.Adamax(learning_rate=1e-4),
          metrics=['accuracy',tf.keras.metrics.SparseTopKCategoricalAccuracy(k=5, name="top5 accuracy")]
          )
model.summary()
```

模型訓練過程

EfficientNetB0







模型比較

不同圖片大小精準度

Xception

Image size=180

```
Epoch 17/20
432/432 [==============================] - 398s 919ms/step - loss: 0.2726 - accuracy: 0.9083 - top5 accuracy: 0.9960 - val_loss: 0.5990 - val_accuracy: 0.8245 - val_top5 accuracy: 0.9797
Epoch 18/20
432/432 [==============================] - 397s 917ms/step - loss: 0.2575 - accuracy: 0.9145 - top5 accuracy: 0.9968 - val_loss: 0.6024 - val_accuracy: 0.8248 - val_top5 accuracy: 0.9797
Epoch 19/20
432/432 [==============================] - 398s 918ms/step - loss: 0.2344 - accuracy: 0.9195 - top5 accuracy: 0.9974 - val_loss: 0.5980 - val_accuracy: 0.8306 - val_top5 accuracy: 0.9800
Epoch 20/20
432/432 [==============================] - 397s 916ms/step - loss: 0.2175 - accuracy: 0.9267 - top5 accuracy: 0.9972 - val_loss: 0.6311 - val_accuracy: 0.8239 - val_top5 accuracy: 0.9809
```

Image size=224

```
Epoch 17/20
432/432 [==============================] - 582s 1s/step - loss: 0.1950 - accuracy: 0.9352 - top5 accuracy: 0.9983 - val_loss: 0.5231 - val_accuracy: 0.8487 - val_top5 accuracy: 0.9849
Epoch 18/20
432/432 [==============================] - 583s 1s/step - loss: 0.1835 - accuracy: 0.9389 - top5 accuracy: 0.9987 - val_loss: 0.5148 - val_accuracy: 0.8523 - val_top5 accuracy: 0.9839
```

勝

EfficientNetB0

Image size=180

```
Epoch 17/20
432/432 [==============================] - 186s 428ms/step - loss: 0.5864 - accuracy: 0.8048 - top5 accuracy: 0.9799 - val_loss: 0.7142 - val_accuracy: 0.7733 - val_top5 accuracy: 0.9690
Epoch 18/20
432/432 [==============================] - 186s 429ms/step - loss: 0.5649 - accuracy: 0.8114 - top5 accuracy: 0.9826 - val_loss: 0.7006 - val_accuracy: 0.7775 - val_top5 accuracy: 0.9697
Epoch 19/20
432/432 [==============================] - 185s 428ms/step - loss: 0.5389 - accuracy: 0.8162 - top5 accuracy: 0.9858 - val_loss: 0.6988 - val_accuracy: 0.7816 - val_top5 accuracy: 0.9691
Epoch 20/20
432/432 [==============================] - 185s 427ms/step - loss: 0.5339 - accuracy: 0.8231 - top5 accuracy: 0.9853 - val_loss: 0.6812 - val_accuracy: 0.7849 - val_top5 accuracy: 0.9717
```

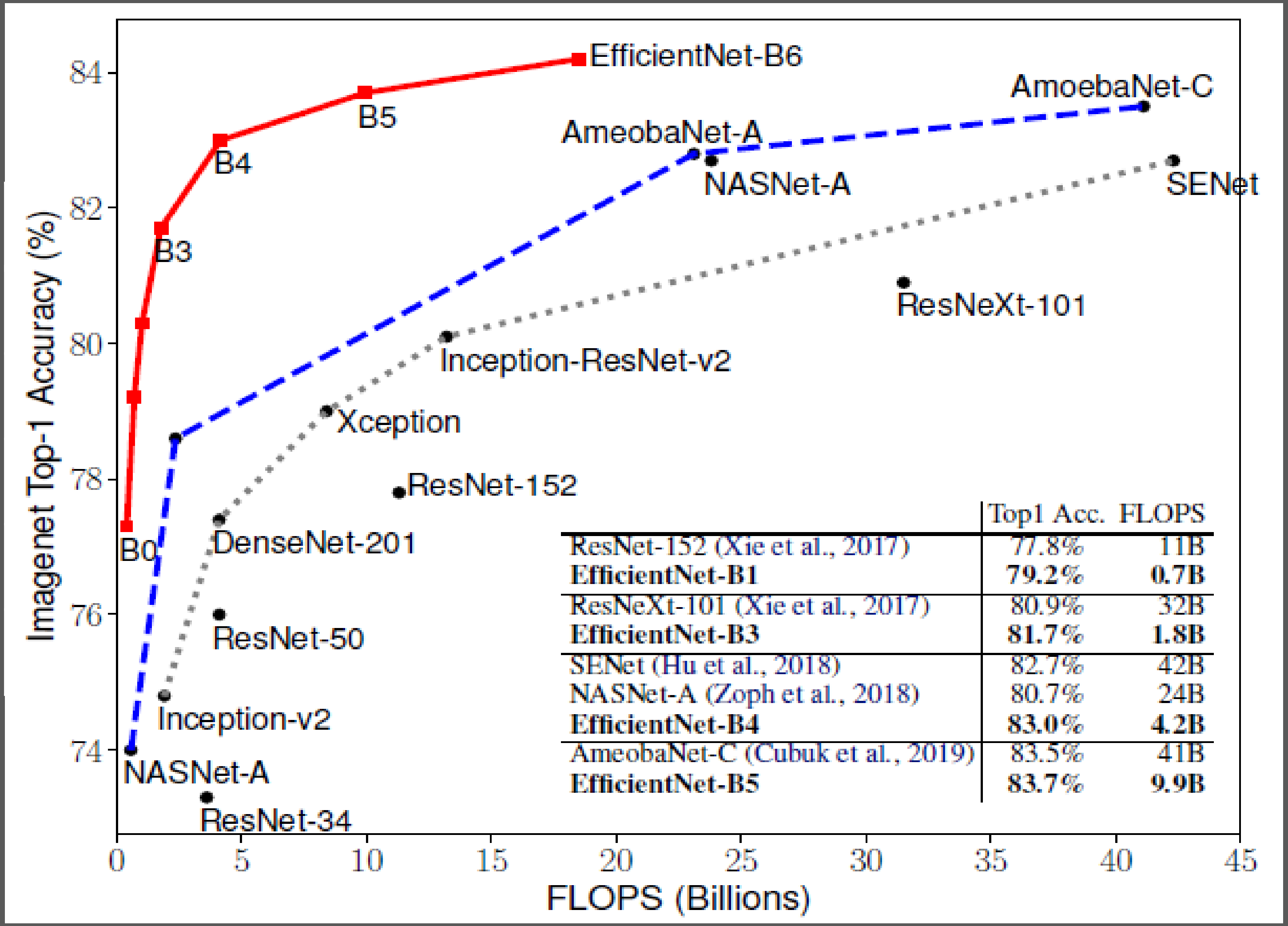
Image size=224

```
Epoch 17/20
432/432 [==============================] - 281s 649ms/step - loss: 0.4612 - accuracy: 0.8458 - top5 accuracy: 0.9884 - val_loss: 0.5958 - val_accuracy: 0.8053 - val_top5 accuracy: 0.9793
Epoch 18/20
432/432 [==============================] - 281s 649ms/step - loss: 0.4432 - accuracy: 0.8531 - top5 accuracy: 0.9891 - val_loss: 0.5806 - val_accuracy: 0.8124 - val_top5 accuracy: 0.9816
Epoch 19/20
432/432 [==============================] - 281s 650ms/step - loss: 0.4181 - accuracy: 0.8594 - top5 accuracy: 0.9906 - val_loss: 0.5931 - val_accuracy: 0.8103 - val_top5 accuracy: 0.9797
Epoch 20/20
432/432 [==============================] - 281s 649ms/step - loss: 0.4059 - accuracy: 0.8648 - top5 accuracy: 0.9920 - val_loss: 0.5960 - val_accuracy: 0.8094 - val_top5 accuracy: 0.9817
```

勝

模型比較

綜合比較



	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**79.2%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.7%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**83.0%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2019)	83.5%	41B
EfficientNet-B5	**83.7%**	**9.9B**

模型比較

模型評估

```
print("Accuracy  score:",  accuracy_score(actual,predict))
print("Report:\n",classification_report(actual,predict,target_names=class_names))
```

EfficientNetB0

Accuracy score: 0.7480730802169568
Report:

	precision	recall	f1-score	support
三杯雞	0.71	0.60	0.65	134
什錦炒麵	0.78	0.78	0.78	158
咖哩雞	0.68	0.75	0.72	153
塔香海茸	0.96	0.77	0.86	137
大陸妹	0.68	0.38	0.48	144
客家小炒	0.92	0.65	0.76	153
小番茄	0.99	0.95	0.97	171
有機小松菜	0.28	0.47	0.35	129
有機青松菜	0.33	0.66	0.44	99
木瓜	0.91	0.97	0.94	162
柳丁	0.78	0.80	0.79	143
棗子	0.92	0.98	0.95	173
橘子	0.84	0.80	0.82	155
沙茶肉片	0.42	0.25	0.31	128
油菜	0.33	0.19	0.24	147
洋蔥炒蛋	0.94	0.61	0.74	124
滷蛋	0.94	0.94	0.94	155
滷雞腿	0.78	0.89	0.83	169
玉米炒蛋	0.88	0.88	0.88	138
瓜仔肉	0.63	0.80	0.70	161
番茄炒蛋	0.85	0.82	0.83	126
白米飯	0.94	0.96	0.95	126
白菜滷	0.80	0.68	0.74	119
福山萵苣	0.20	0.50	0.28	26
空心菜	0.69	0.85	0.76	164
糖醋雞丁	0.68	0.54	0.60	152
紅蘿蔔炒蛋	0.96	0.83	0.89	156
義大利麵	0.66	0.72	0.69	165
芥藍菜	0.71	0.26	0.38	84
菠菜	0.74	0.24	0.36	153
葡萄	0.99	0.94	0.97	140
蒜泥白肉	0.50	0.74	0.60	130
蒸蛋	0.79	0.84	0.81	127
蓮霧	0.97	0.97	0.97	109
螞蟻上樹	0.93	0.69	0.79	124
西瓜	0.97	0.94	0.95	161
豆芽菜	0.91	0.78	0.84	164
關東煮	0.69	0.91	0.79	136
青江菜	0.49	0.60	0.54	120
香蕉	0.96	1.00	0.98	160
香酥魚排	0.73	0.89	0.80	168
馬鈴薯燉肉	0.52	0.82	0.64	139
高麗菜	0.80	0.92	0.85	122
鳳梨	0.96	0.95	0.96	145
鵝白菜	0.74	0.75	0.75	141
鹽酥雞	0.76	0.80	0.78	151
麥克雞塊	0.96	0.87	0.91	131
麻婆豆腐	0.81	0.73	0.77	143
麻油雞	0.70	0.57	0.63	154
黑胡椒豬柳	0.53	0.54	0.54	137
accuracy			0.75	7006
macro avg	0.75	0.74	0.73	7006
weighted avg	0.77	0.75	0.75	7006

Xception

勝

Accuracy score: 0.7751926919783043
Report:

	precision	recall	f1-score	support
三杯雞	0.61	0.78	0.69	134
什錦炒麵	0.76	0.84	0.80	158
咖哩雞	0.76	0.74	0.75	153
塔香海茸	0.97	0.83	0.89	137
大陸妹	0.67	0.62	0.65	144
客家小炒	0.73	0.78	0.75	153
小番茄	0.99	0.96	0.98	171
有機小松菜	0.29	0.49	0.36	129
有機青松菜	0.42	0.54	0.47	99
木瓜	0.88	0.98	0.93	162
柳丁	0.80	0.84	0.82	143
棗子	0.97	0.94	0.95	173
橘子	0.85	0.87	0.86	155
沙茶肉片	0.38	0.42	0.40	128
油菜	0.39	0.24	0.30	147
洋蔥炒蛋	0.94	0.71	0.81	124
滷蛋	0.91	0.95	0.93	155
滷雞腿	0.82	0.86	0.84	169
玉米炒蛋	0.95	0.88	0.91	138
瓜仔肉	0.89	0.76	0.82	161
番茄炒蛋	0.86	0.87	0.86	126
白米飯	0.98	0.94	0.96	126
白菜滷	0.71	0.76	0.73	119
福山萵苣	0.23	0.35	0.27	26
空心菜	0.75	0.72	0.74	164
糖醋雞丁	0.54	0.59	0.56	152
紅蘿蔔炒蛋	0.92	0.85	0.88	156
義大利麵	0.81	0.69	0.75	165
芥藍菜	0.48	0.51	0.49	84
菠菜	0.68	0.48	0.56	153
葡萄	0.98	0.94	0.96	140
蒜泥白肉	0.51	0.66	0.57	130
蒸蛋	0.89	0.89	0.89	127
蓮霧	0.96	0.98	0.97	109
螞蟻上樹	0.86	0.89	0.87	124
西瓜	0.96	0.93	0.94	161
豆芽菜	0.97	0.86	0.91	164
關東煮	0.80	0.86	0.83	136
青江菜	0.63	0.57	0.60	120
香蕉	0.99	0.99	0.99	160
香酥魚排	0.83	0.90	0.86	168
馬鈴薯燉肉	0.62	0.83	0.71	139
高麗菜	0.92	0.80	0.85	122
鳳梨	0.95	0.95	0.95	145
鵝白菜	0.75	0.78	0.77	141
鹽酥雞	0.87	0.69	0.77	151
麥克雞塊	0.89	0.89	0.89	131
麻婆豆腐	0.85	0.73	0.79	143
麻油雞	0.67	0.57	0.62	154
黑胡椒豬柳	0.68	0.61	0.65	137
accuracy			0.78	7006
macro avg	0.77	0.76	0.76	7006
weighted avg	0.79	0.78	0.78	7006

Precision：預測該類別正確數/預測到該類別的總數

Recall：預測該類別正確數/該類別總數

F1-score：(2*Recall* Precision)/(Recall+Precision)

模型結果分析

Xception

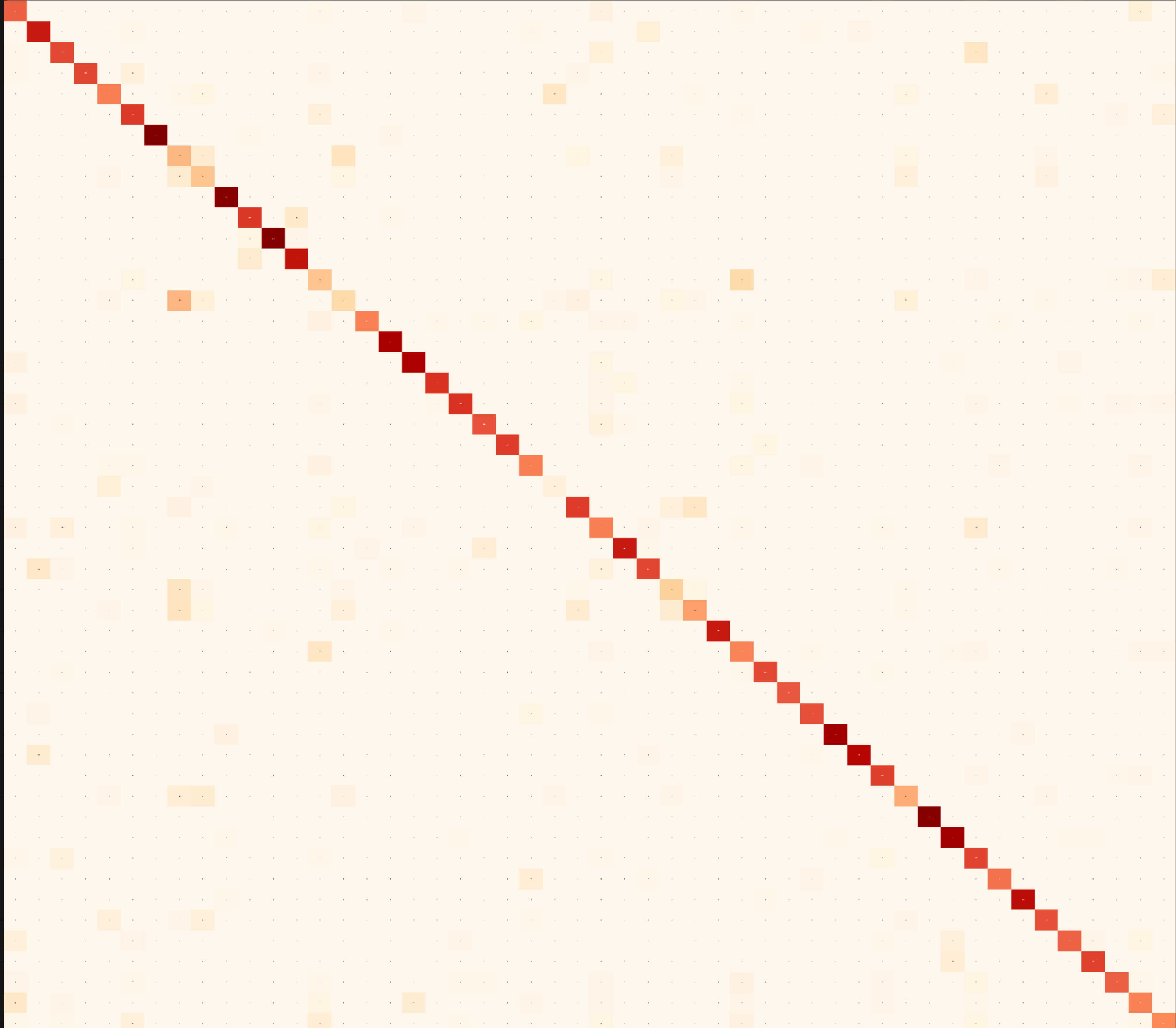
預測並排序精準度

```
predict=[]
ave_acc=[]
actual=[]
for cls in range(len(class_names)):
    imagelist= os.listdir('/tmp/test/'+class_names[cls])
    total_acc=0
    if len(imagelist)==0:
        continue
    else:
        for img in range(len(imagelist)):
            actual.append(cls)
            path='/tmp/test/'+str(class_names[cls])+'/'+str(imagelist[img])
            tem_img=load_img(path,target_size=(IMG_SIZE,IMG_SIZE))
            tem_img = np.expand_dims(tem_img, axis = 0)
            pred = model.predict(tem_img)[0]
            top_inds = pred.argsort()[::-1][:1]
            for top1 in top_inds:
                total_acc=total_acc+pred[top1]
                predict.append(top1)
        ave_acc.append(total_acc/len(imagelist))
```

```
print("Average accuracy: \n")
top_acc=sorted(ave_acc, reverse = True)
top_class=[]
while len(top_class)<49:
    for i in range(len(class_names)):
        for j in range(len(class_names)):
            if top_acc[i]==ave_acc[j]:
                top_class.append(class_names[j])
                break
for k in range(len(class_names)):
    if k < 9:
        print('   精準度Top.0{}:  {:.3f}    {}'.format(k+1,top_acc[k],  top_class[k]))
    else:
        print('   精準度Top.{}:  {:.3f}    {}'.format(k+1,top_acc[k],  top_class[k]))
```

模型結果分析

Xception



模型結果分析
Xception

Top10種類精準度

共同特點

- 顏色單一
- 特徵較明顯
- 不易有配料

```
Average accuracy:

  精準度Top.01:  0.995  香蕉
  精準度Top.02:  0.979  蓮霧
  精準度Top.03:  0.975  白米飯
  精準度Top.04:  0.972  滷蛋
  精準度Top.05:  0.970  小番茄
  精準度Top.06:  0.967  葡萄
  精準度Top.07:  0.966  木瓜
  精準度Top.08:  0.965  西瓜
  精準度Top.09:  0.962  棗子
  精準度Top.10:  0.958  鳳梨
```

模型結果分析

Xception

倒數Top10種類精準度

共同特點

- 肉眼難以分辨
- 多為青菜
- 配料太多元

```
精準度Top.41:  0.768  芥藍菜
精準度Top.42:  0.753  糖醋雞丁
精準度Top.43:  0.752  黑胡椒豬柳
精準度Top.44:  0.734  有機青松菜
精準度Top.45:  0.727  菠菜
精準度Top.46:  0.716  青江菜
精準度Top.47:  0.704  沙茶肉片
精準度Top.48:  0.697  福山萵苣
精準度Top.49:  0.681  有機小松菜
精準度Top.50:  0.671  油菜
```

油菜

有機小松菜

福山萵苣

青江菜

模型結果分析

Xception

倒數Top10種類精準度

共同特點

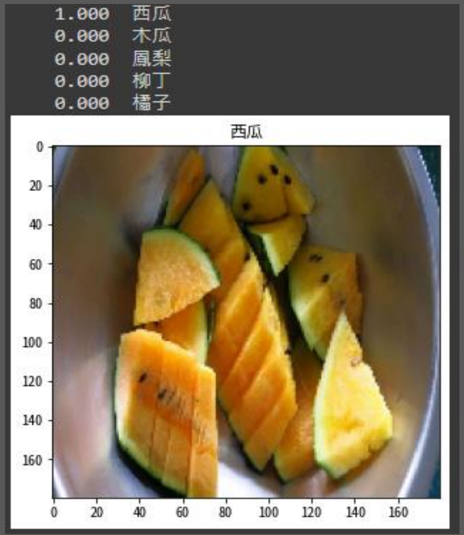
- 肉眼難以分辨
- 多為青菜
- 配料太多元

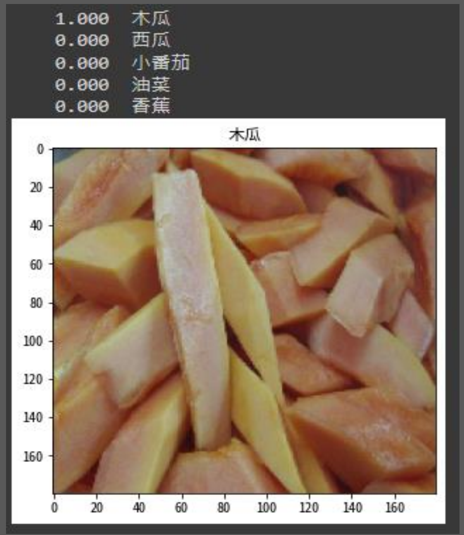
精準度Top.41:	0.768	芥藍菜
精準度Top.42:	0.753	糖醋雞丁
精準度Top.43:	0.752	黑胡椒豬柳
精準度Top.44:	0.734	有機青松菜
精準度Top.45:	0.727	菠菜
精準度Top.46:	0.716	青江菜
精準度Top.47:	0.704	沙茶肉片
精準度Top.48:	0.697	福山萵苣
精準度Top.49:	0.681	有機小松菜
精準度Top.50:	0.671	油菜

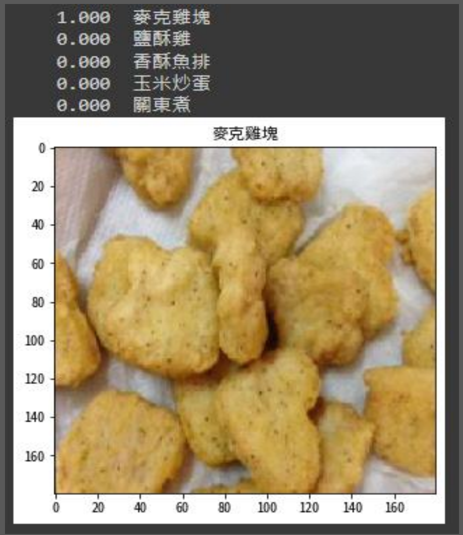
沙茶肉片

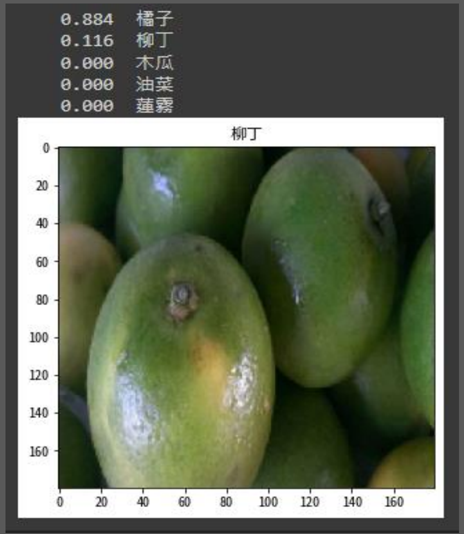
模型結果分析
Xception

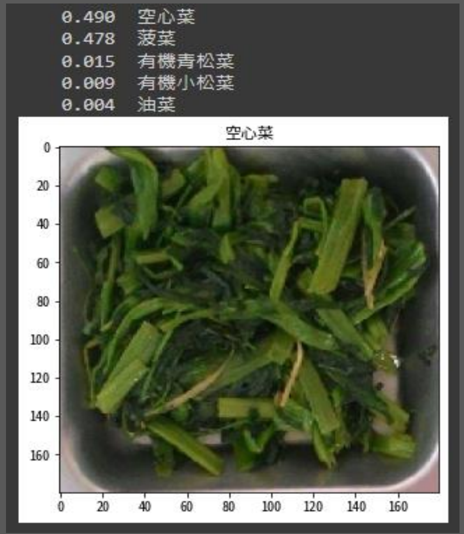
Top5 accuracy











分不開的菜餚

易錯類別

篩選條件

- 預測該錯誤類別數大於正確數
- 該錯誤類別數占總數兩成以上

共同特點

- 顏色相似
- 輪廓特徵相似

Xception

勝

```python
for i in range(len(class_names)):
    max_val=0
    max_cls=''
    sec_val=0
    sec_cls=''
    imagelist= os.listdir('/tmp/test/'+class_names[i])
    for j in range(len(class_names)):
        if i!=j:
            if mat[j][i]>sec_val:
                sec_val=mat[j][i]
                sec_cls=class_names[j]
        if mat[j][i]>max_val:
            max_val=mat[j][i]
            max_cls=class_names[j]
    if class_names[i]!=max_cls:
        print('Actual:   {:10}  Max_predict:   {:10}'.format(class_names[i],max_cls))
    if sec_val/len(imagelist)>0.2 and max_cls!=sec_cls:
        print('Actual:   {:10}  Sec_predict:   {:10}'.format(class_names[i],sec_cls))
```

```
Actual:  有機小松菜      Max_predict:  油菜
Actual:  福山萵苣      Max_predict:  大陸妹
Actual:  蒜泥白肉      Sec_predict:  沙茶肉片
```

EfficientNetB0

```python
for i in range(len(class_names)):
    max_val=0
    max_cls=''
    sec_val=0
    sec_cls=''
    imagelist= os.listdir('/tmp/test/'+class_names[i])
    for j in range(len(class_names)):
        if i!=j:
            if mat[j][i]>sec_val:
                sec_val=mat[j][i]
                sec_cls=class_names[j]
        if mat[j][i]>max_val:
            max_val=mat[j][i]
            max_cls=class_names[j]
    if class_names[i]!=max_cls:
        print('Actual:   {:10}  Max_predict:   {:10}'.format(class_names[i],max_cls))
    if sec_val/len(imagelist)>0.2 and max_cls!=sec_cls:
        print('Actual:   {:10}  Sec_predict:   {:10}'.format(class_names[i],sec_cls))
```

```
Actual:  有機小松菜      Sec_predict:  油菜
Actual:  有機青松菜      Sec_predict:  油菜
Actual:  福山萵苣      Max_predict:  大陸妹
Actual:  空心菜      Sec_predict:  菠菜
Actual:  蒜泥白肉      Sec_predict:  沙茶肉片
```

分不開的菜餚

易錯類別

共同特點

- 顏色相似
- 輪廓特徵相似

有機小松菜

油菜

分不開的菜餚

易錯類別

共同特點

- 顏色相似
- 輪廓特徵相似

福山萵苣

大陸妹

分不開的菜餚

易錯類別

共同特點

- 顏色相似
- 輪廓特徵相似

蒜泥白肉

沙茶肉片

Thanks For Listening