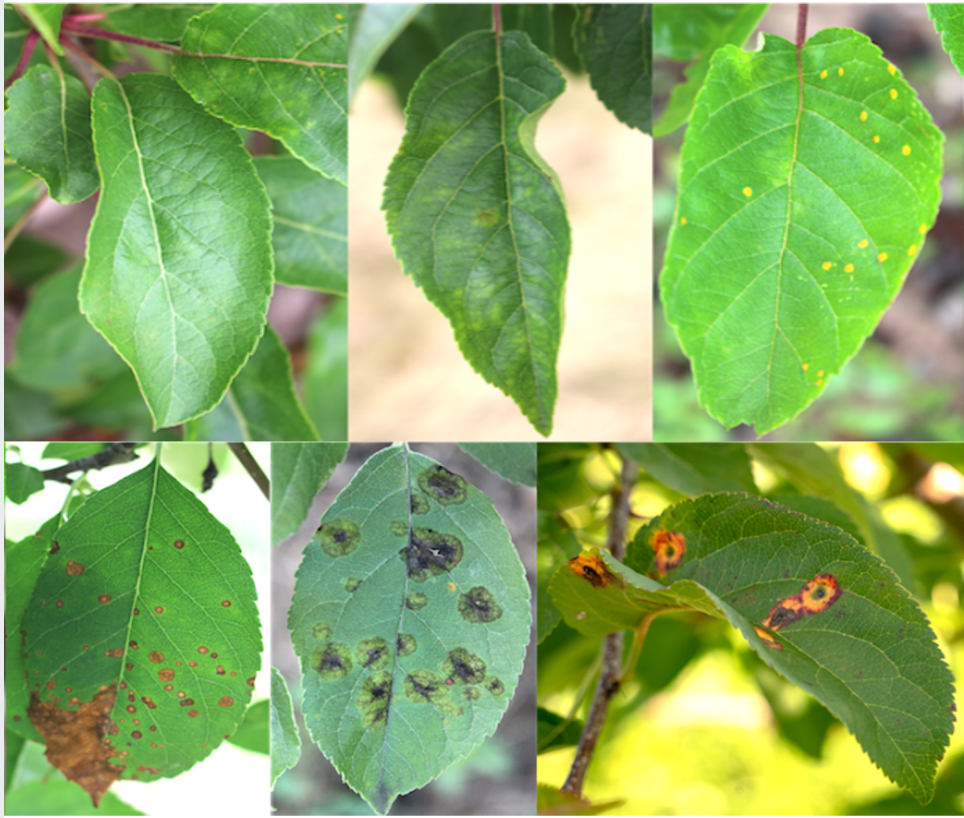
# Plant Pathology Challenge

類神經網路導論期末報告

# 問題定義

傳統上使用人工的方式，消耗人力及費時。再者，對於疾病的錯誤判斷可能導致抗藥性的病原體產生，使得疾病難以被根治。此為CVPR2020的FGVC7研討會挑戰。



# 輸入資料格式

## 輸入

特徵(feature):

1822張 RGB照片

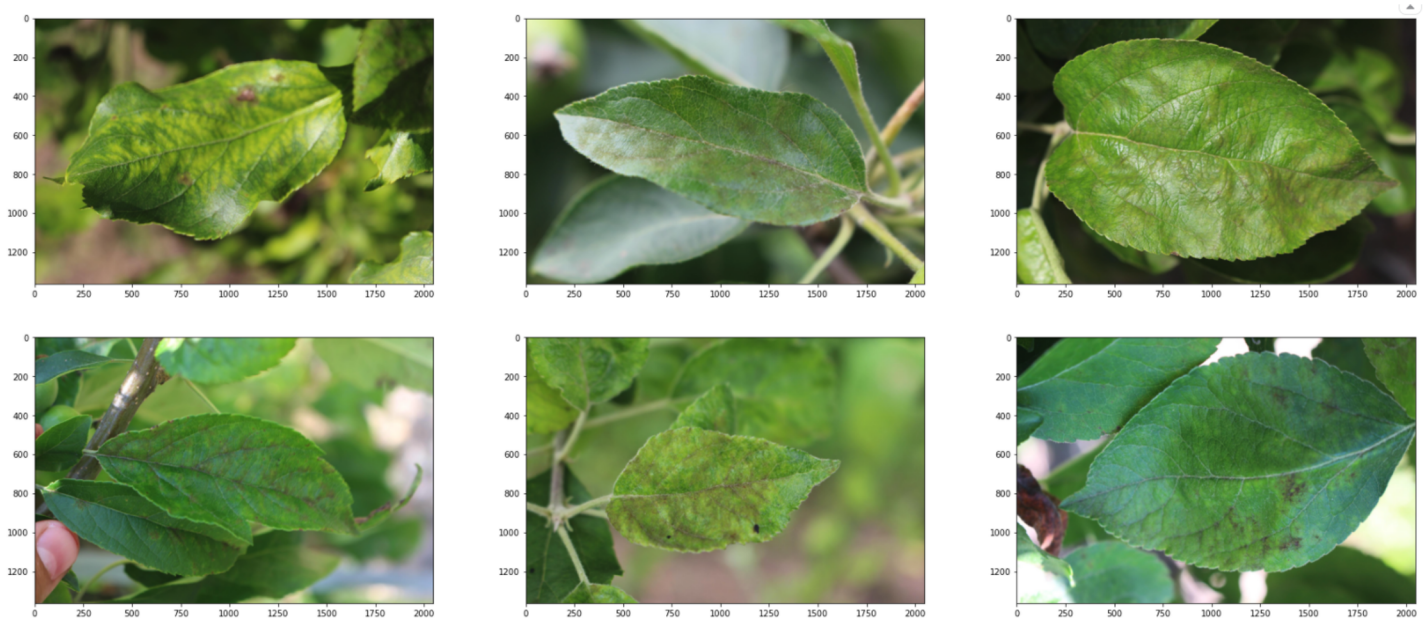
答案

healthy、multiple_diseases、rust、scab

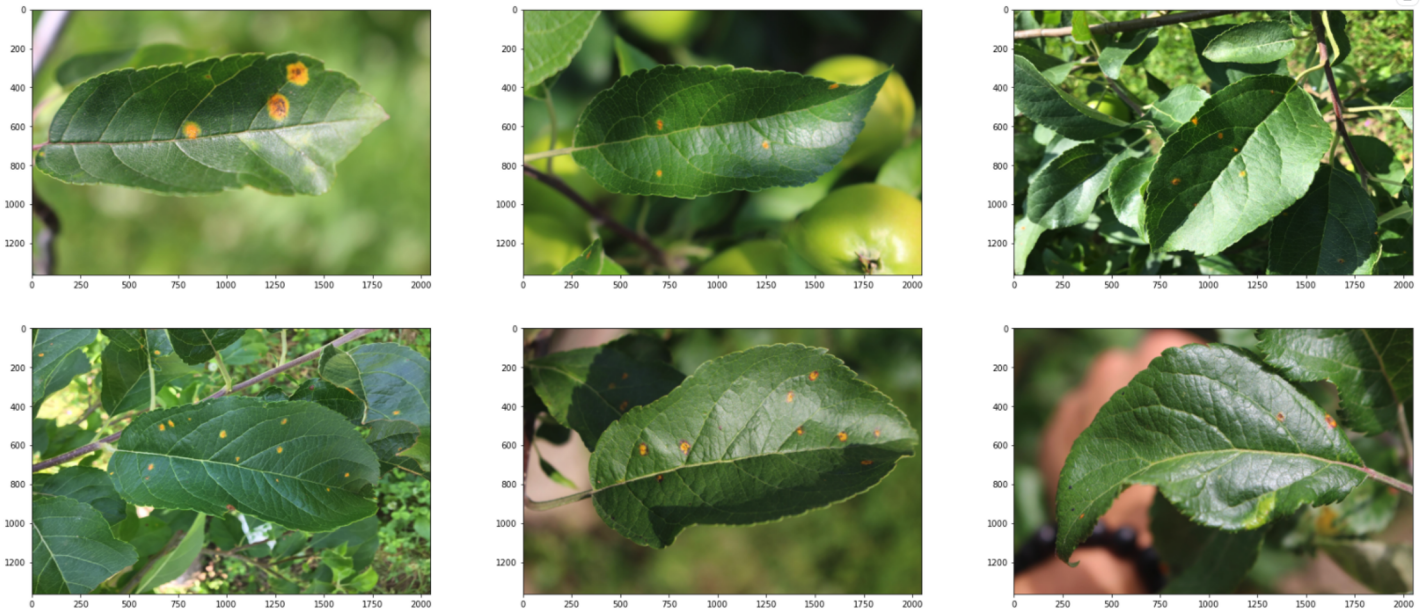
## 輸出

給定test資料集，所檢測的結果。

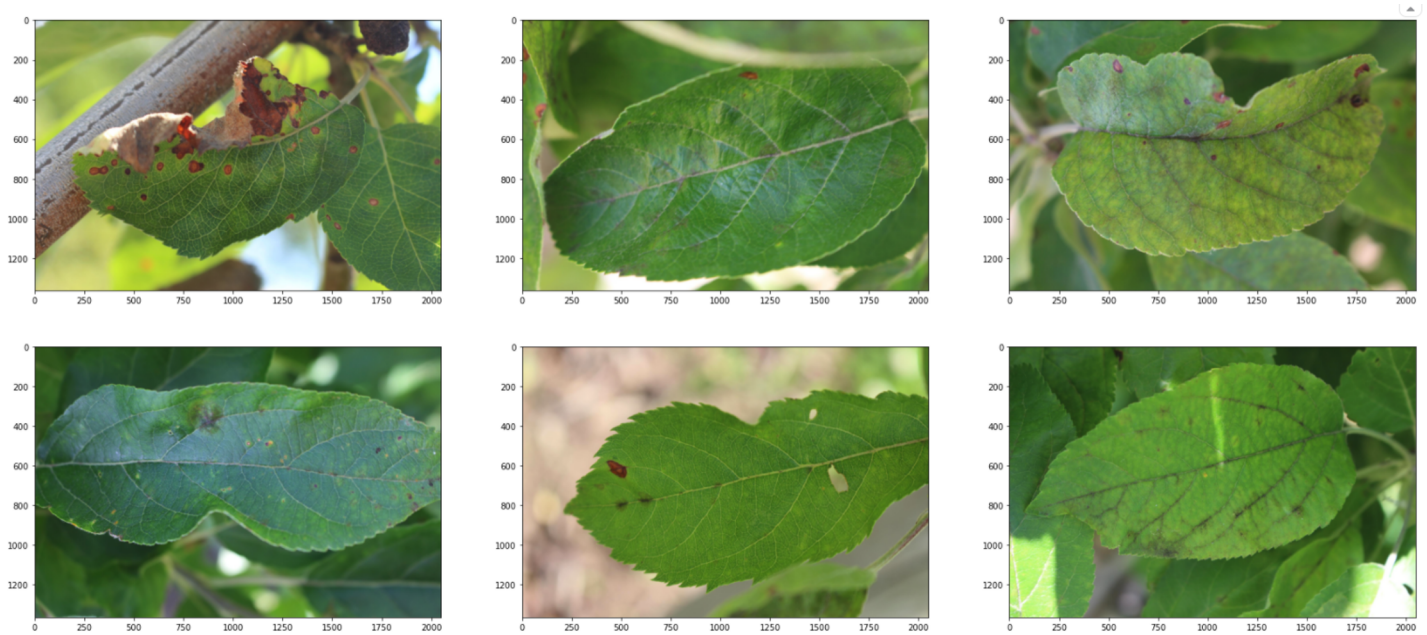
# 輸入答案 - scab



# 輸入答案 - rust

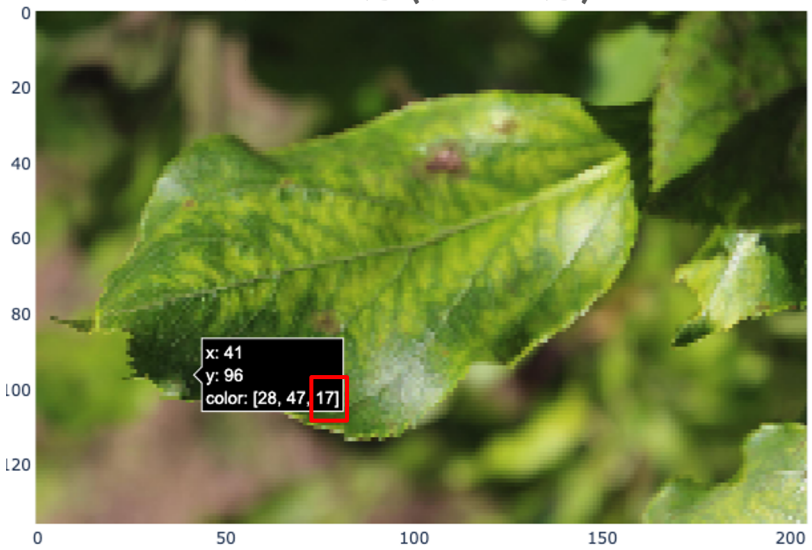


# 輸入答案 - multiple diseases

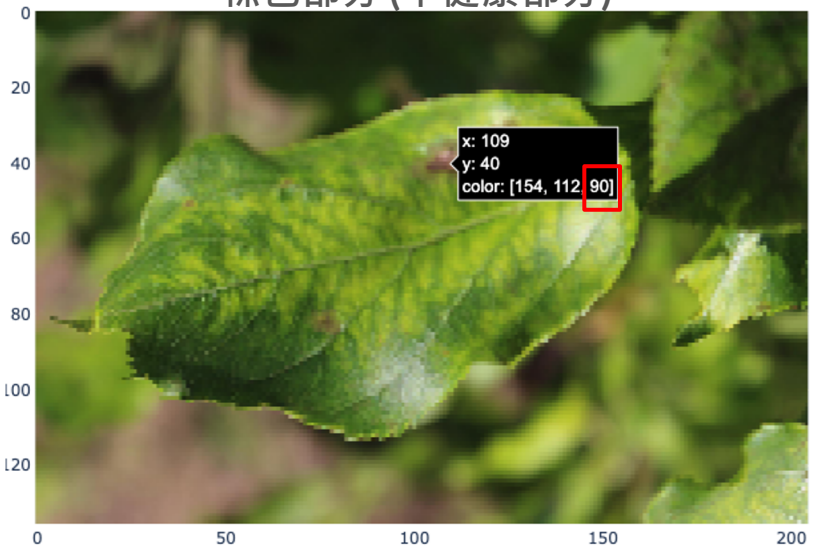


# 輸入資料視覺化

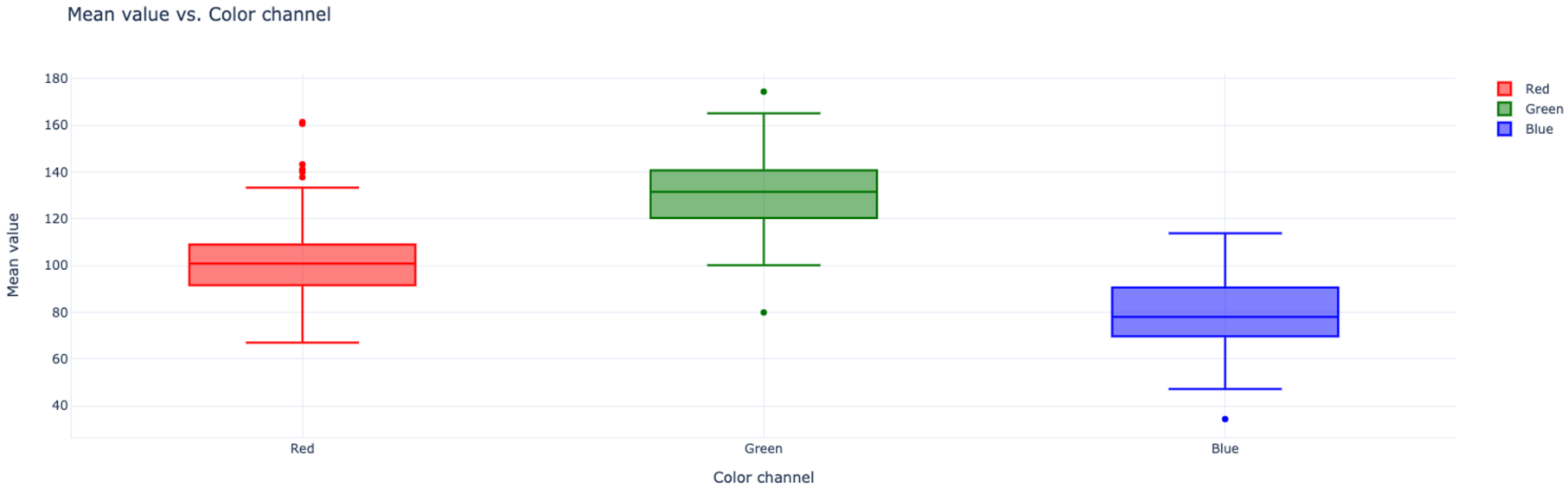
綠色部分 (健康部分)



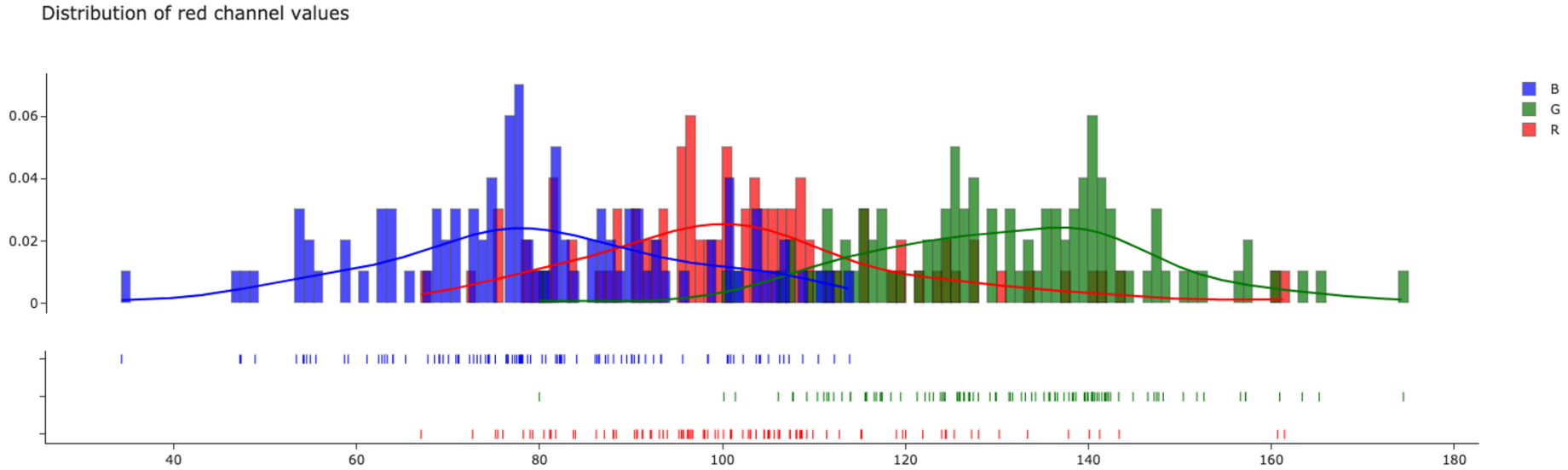

棕色部分 (不健康部分)



# 輸入資料統計

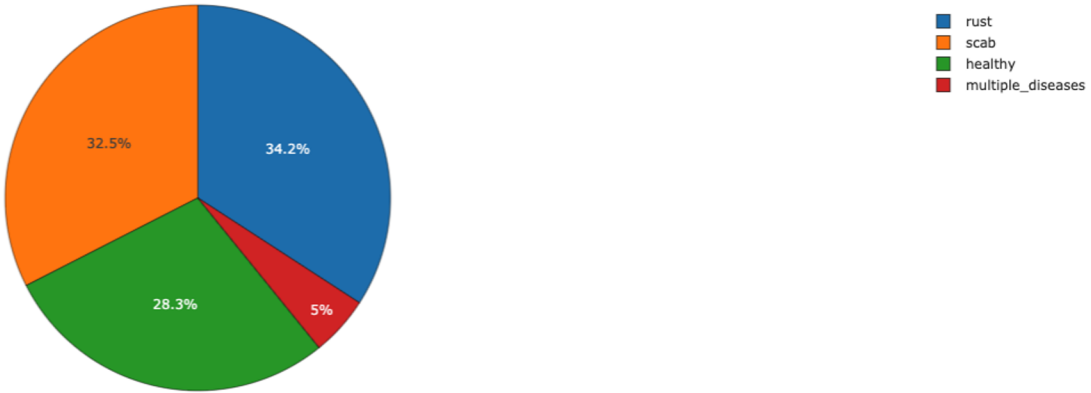

Mean value vs. Color channel
180
160
140
120
100
80
60
40
Mean value
Red
Green
Blue
Color channel
Red
Green
Blue

# 輸入資料分佈


Distribution of red channel values
0.06
0.04
0.02
0
B
G
R
40
60
80
100
120
140
160
180

# 輸入資料類別比例

Pie chart of targets



# 類神經模型結構 - InceptionResNetV2

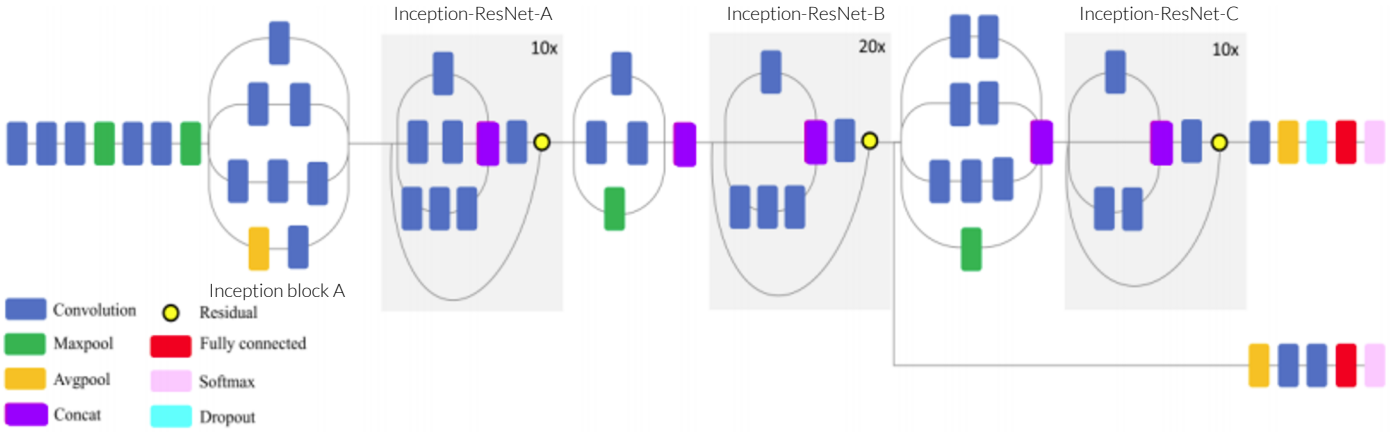
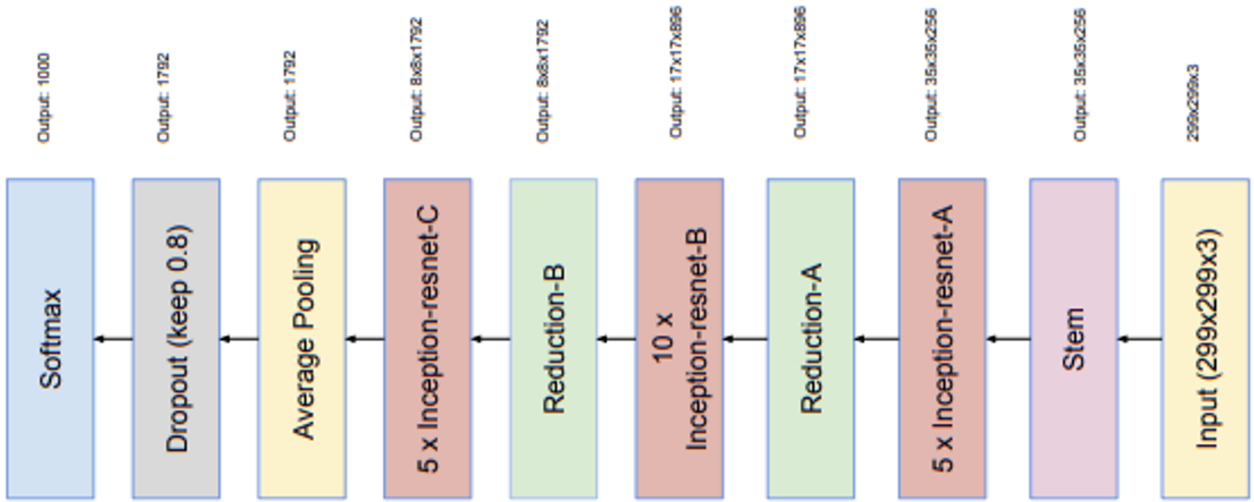



**Figure 5.** Schematic diagram of InceptionResNetV2 model (compressed view).

[2]

# 類神經模型結構 - InceptionResNetV2




[3]

# 類神經模型結構 - InceptionResNetV2

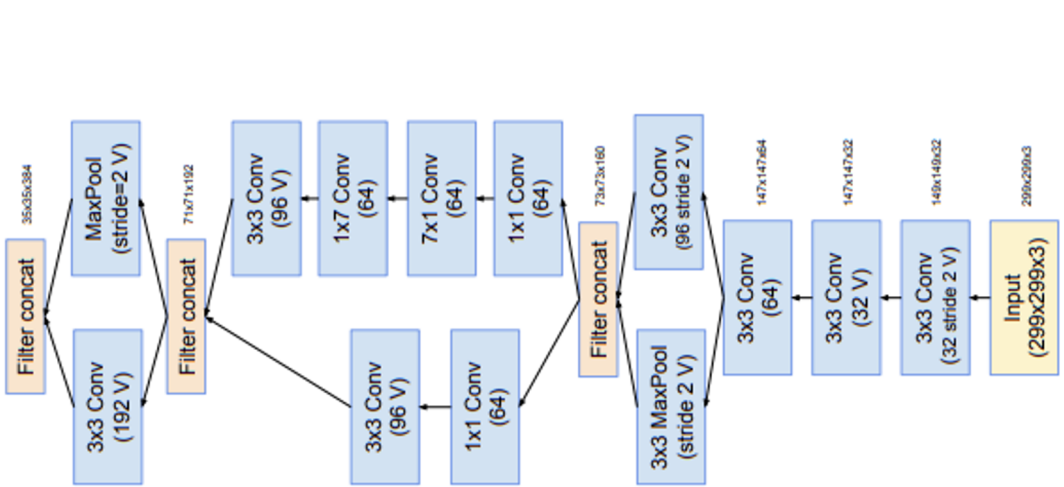



Figure 3. The schema for stem of the pure Inception-v4 and Inception-ResNet-v2 networks. This is the input part of those networks. Cf. Figures 9 and 15

[3]

# 類神經模型結構 - InceptionResNetV2

Inception block A

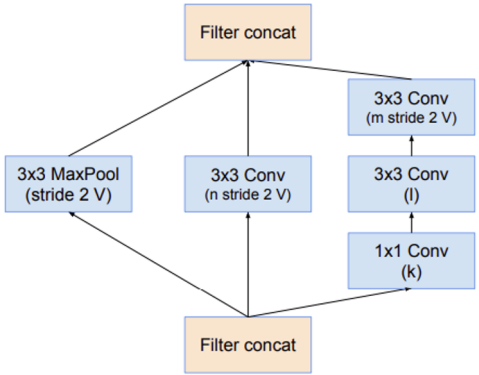



Figure 7. The schema for $35 \times 35$ to $17 \times 17$ reduction module. Different variants of this blocks (with various number of filters) are used in Figure 9, and 15 in each of the new Inception(-v4, -ResNet-v1, -ResNet-v2) variants presented in this paper. The $k$, $l$, $m$, $n$ numbers represent filter bank sizes which can be looked up in Table 1.

Inception block B

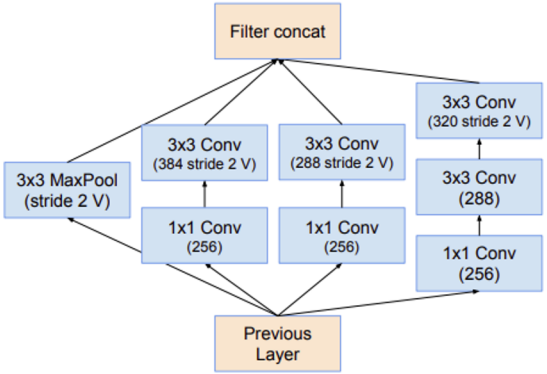



Figure 18. The schema for $17 \times 17$ to $8 \times 8$ grid-reduction module. Reduction-B module used by the wider Inception-ResNet-v1 network in Figure 15.

[3]

# 類神經模型結構 - InceptionResNetV2

Inception-ResNet-A

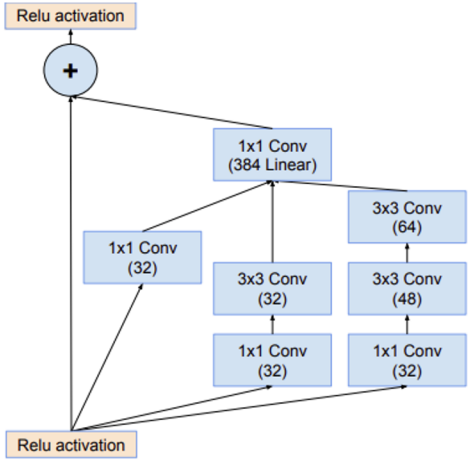



Figure 16. The schema for 35 × 35 grid (Inception-ResNet-A) module of the Inception-ResNet-v2 network.

Inception-ResNet-B

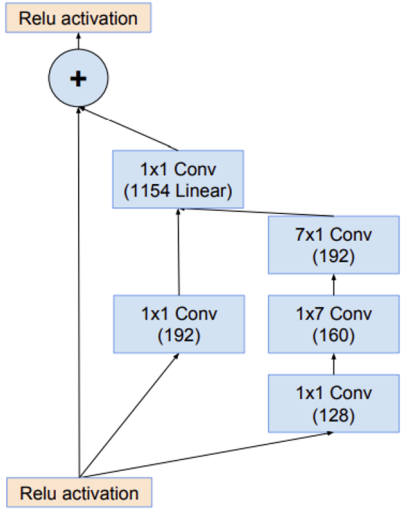



Figure 17. The schema for 17 × 17 grid (Inception-ResNet-B) module of the Inception-ResNet-v2 network.

Inception-ResNet-C

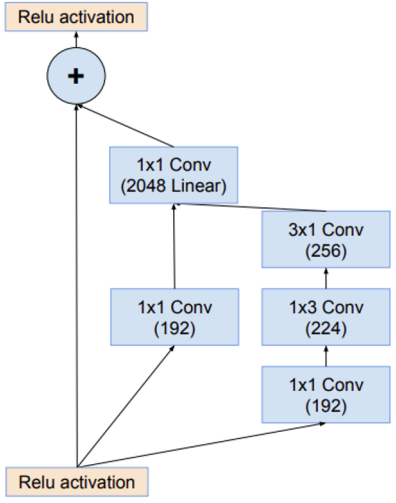



Figure 19. The schema for 8×8 grid (Inception-ResNet-C) module of the Inception-ResNet-v2 network.

[3]

# 相關參數 - InceptionResNetV2

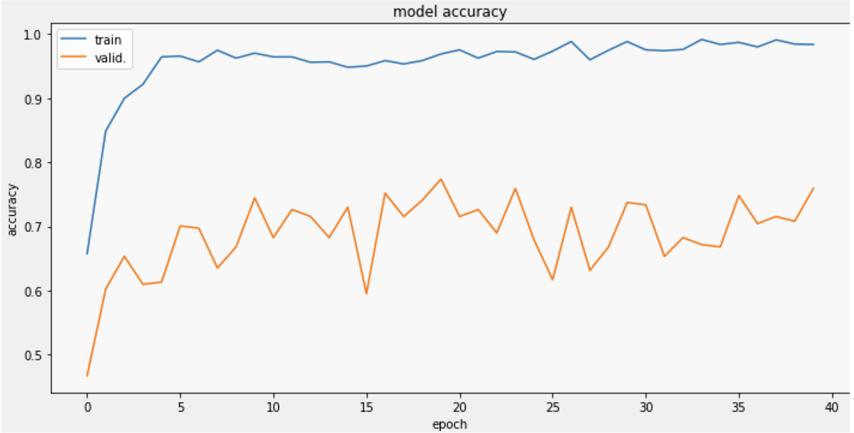
| Layer (type) | Output Shape | Param # |
|---|---|---|
| inception_resnet_v2 (Model) | (None, 23, 23, 1536) | 54336736 |
| global_max_pooling2d (Global | (None, 1536) | 0 |
| dense (Dense) | (None, 4) | 6148 |

Total params: 54,342,884
Trainable params: 54,282,340
Non-trainable params: 60,544

output: 4個類別

Param: 6148 = (1536+1)*4

# 結果 - InceptionResNetV2

# 類神經模型結構 - DenseNet

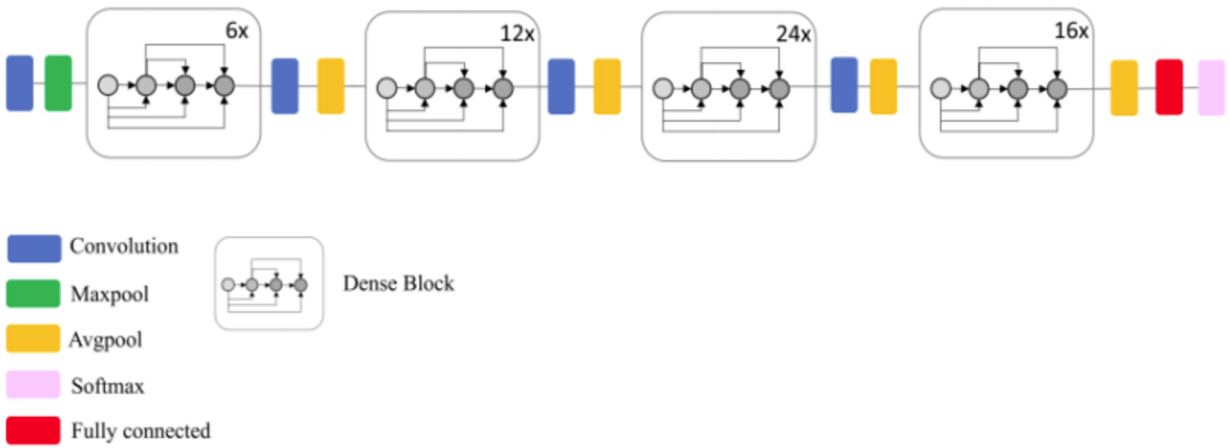



**Figure 6.** Schematic diagram of DenseNet model (compressed view).

[2]

# 類神經模型結構 - DenseNet

[4]

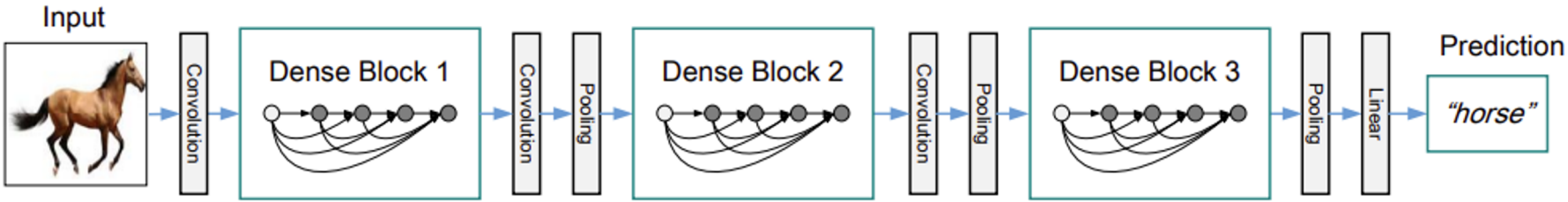



**Figure 2:** A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

# 類神經模型結構 - DenseNet

我們用這

[4]

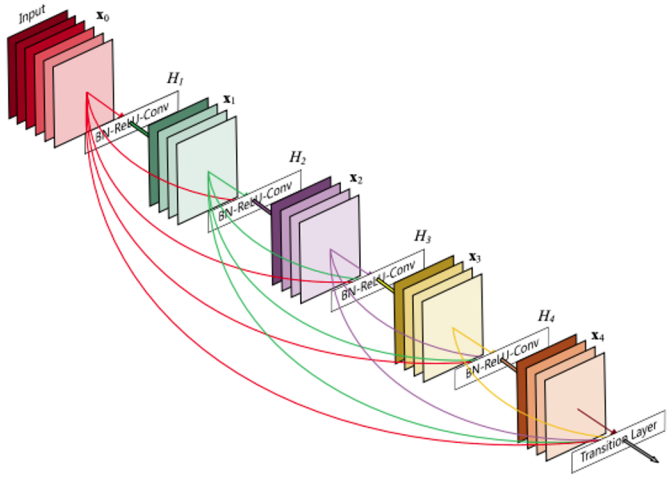



**Figure 1:** A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

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

# 相關參數 - DenseNet

| Layer (type) | Output Shape | Param # |
|---|---|---|
| densenet121 (Model) | (None, 16, 16, 1024) | 7037504 |
| global_average_pooling2d (Gl | (None, 1024) | 0 |
| dense (Dense) | (None, 4) | 4100 |

Total params: 7,041,604
Trainable params: 6,957,956
Non-trainable params: 83,648

output: 4個類別

Param: 4100 = (1024+1)*4

# 相關參數 - DenseNet - modified

| Layer (type) | Output Shape | Param # |
|---|---|---|
| densenet121 (Model) | (None, 16, 16, 1024) | 7037504 |
| global_average_pooling2d (Gl | (None, 1024) | 0 |
| dense (Dense) | (None, 1024) | 1049600 |
| dense_1 (Dense) | (None, 1024) | 1049600 |
| dense_2 (Dense) | (None, 4) | 4100 |

Total params: 9,140,804
Trainable params: 9,057,156
Non-trainable params: 83,648

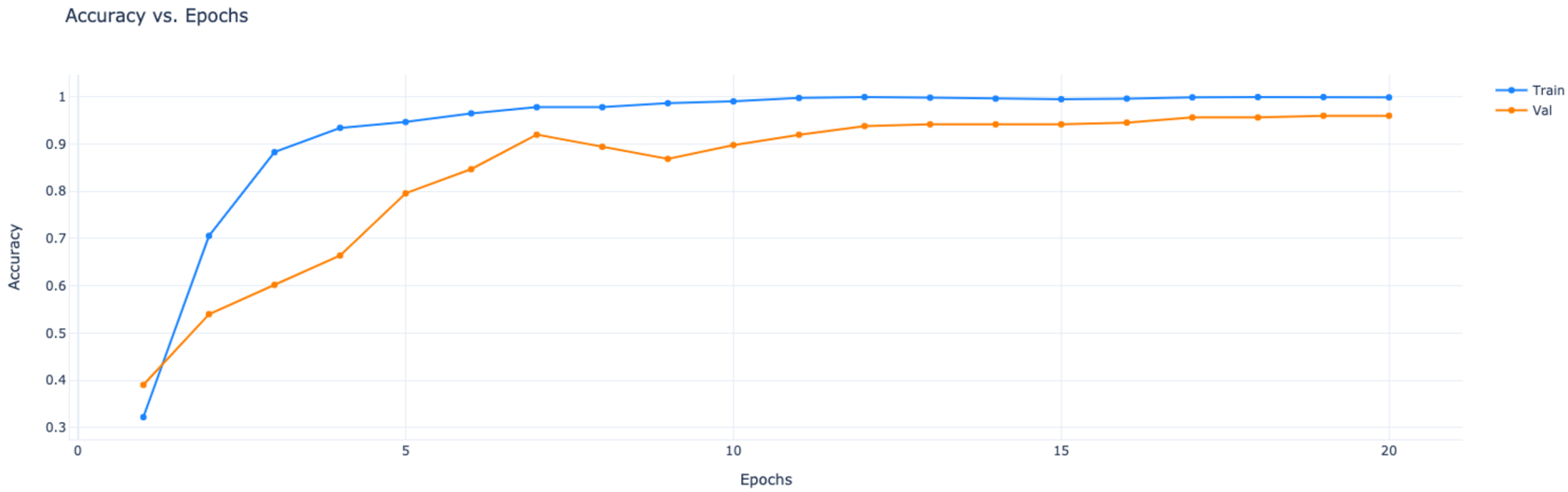
output: 4個類別

Param:

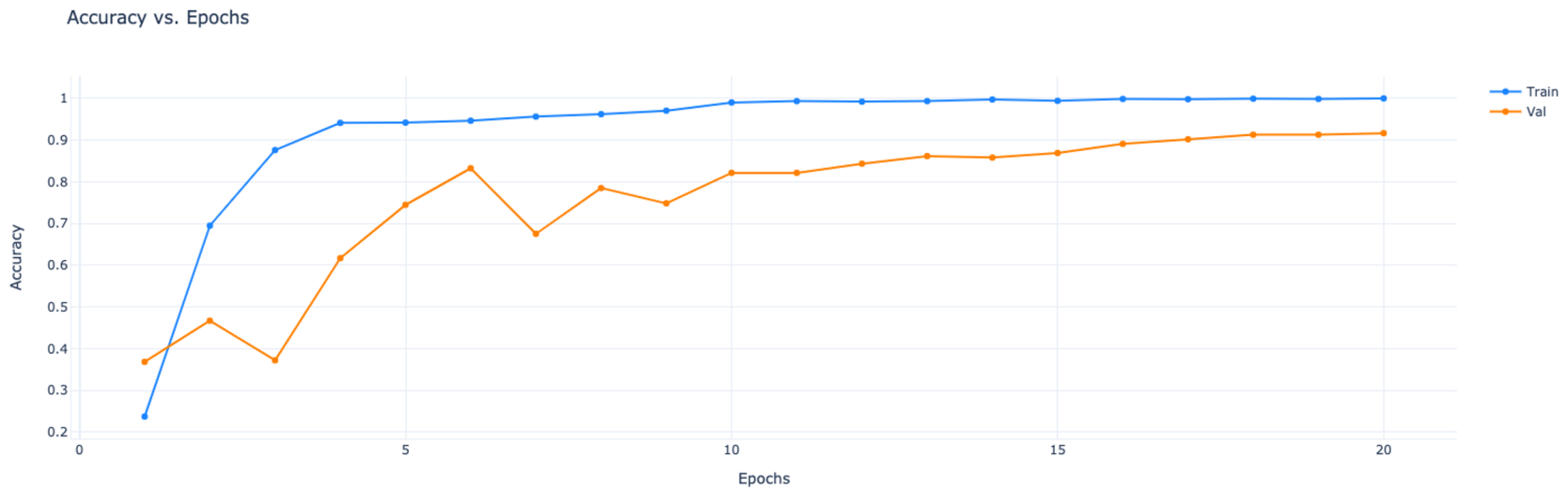
1049600 = (1024+1)*1024

4100 = (1024+1)*4

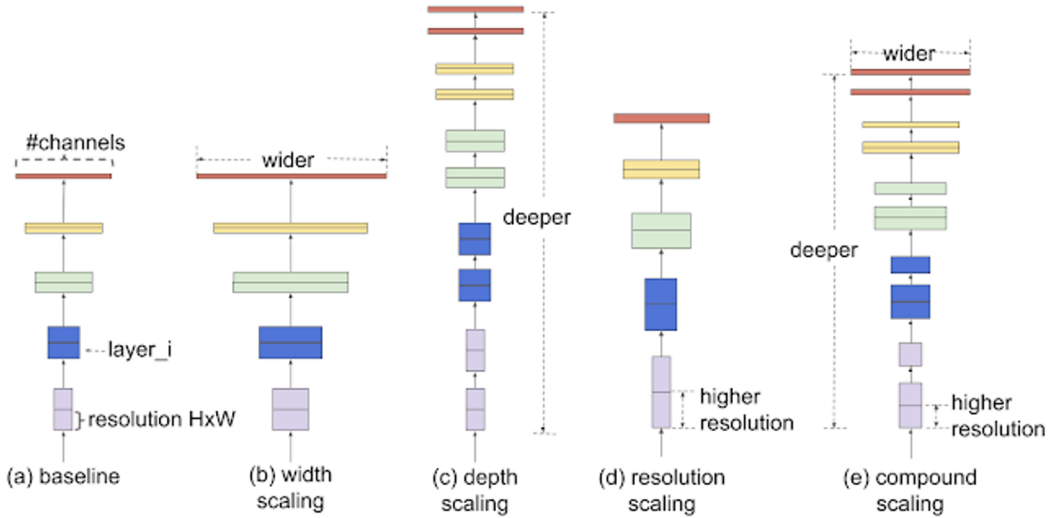
# 結果 - DenseNet


Accuracy vs. Epochs
Accuracy
Epochs
Train
Val
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
5
10
15
20

# 結果 - DenseNet - modified



# 類神經模型結構 - EfficientNet

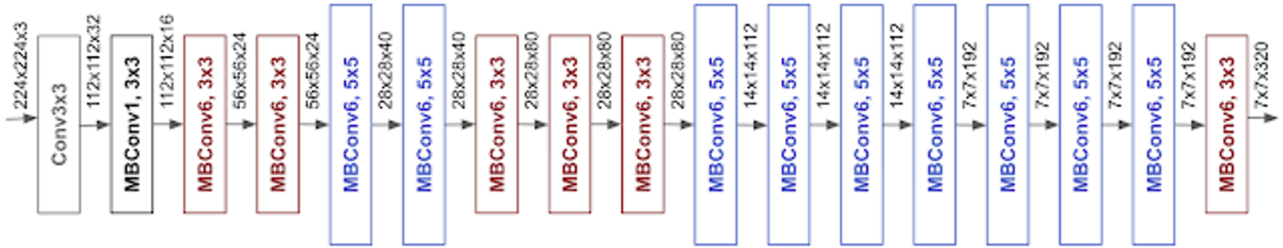



Comparison of different scaling methods. Unlike conventional scaling methods (b)-(d) that arbitrary scale a single dimension of the network, our compound scaling method uniformly scales up all dimensions in a principled way.

[5]

# 類神經模型結構 - EfficientNet-B0

[5]




The architecture for our baseline network EfficientNet-B0 is simple and clean, making it easier to scale and generalize.

MBConv [6]

# 類神經模型結構 - EfficientNet-B0[5]

*Table 1.* **EfficientNet-B0 baseline network** – Each row describes a stage $i$ with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

| Stage $i$ | Operator $\hat{\mathcal{F}}_i$ | Resolution $\hat{H}_i \times \hat{W}_i$ | #Channels $\hat{C}_i$ | #Layers $\hat{L}_i$ |
|---|---|---|---|---|
| 1 | Conv3x3 | $224 \times 224$ | 32 | 1 |
| 2 | MBConv1, k3x3 | $112 \times 112$ | 16 | 1 |
| 3 | MBConv6, k3x3 | $112 \times 112$ | 24 | 2 |
| 4 | MBConv6, k5x5 | $56 \times 56$ | 40 | 2 |
| 5 | MBConv6, k3x3 | $28 \times 28$ | 80 | 3 |
| 6 | MBConv6, k5x5 | $14 \times 14$ | 112 | 3 |
| 7 | MBConv6, k5x5 | $14 \times 14$ | 192 | 4 |
| 8 | MBConv6, k3x3 | $7 \times 7$ | 320 | 1 |
| 9 | Conv1x1 & Pooling & FC | $7 \times 7$ | 1280 | 1 |

$$\mathcal{N} = \bigodot_{i=1\ldots s} \mathcal{F}_i^{L_i}\left(X_{\langle H_i, W_i, C_i \rangle}\right) \tag{1}$$

where $\mathcal{F}_i^{L_i}$ denotes layer $F_i$ is repeated $L_i$ times in stage $i$, $\langle H_i, W_i, C_i \rangle$ denotes the shape of input tensor $X$ of layer

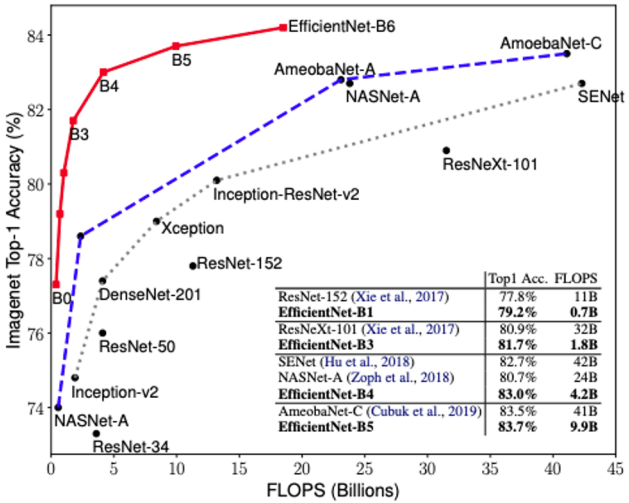
$$\begin{aligned} \max_{d,w,r} \quad & Accuracy\left(\mathcal{N}(d,w,r)\right) \\ s.t. \quad & \mathcal{N}(d,w,r) = \bigodot_{i=1\ldots s} \hat{\mathcal{F}}_i^{d\cdot\hat{L}_i}\left(X_{\langle r\cdot\hat{H}_i, r\cdot\hat{W}_i, w\cdot\hat{C}_i \rangle}\right) \\ & \text{Memory}(\mathcal{N}) \le \text{target\_memory} \\ & \text{FLOPS}(\mathcal{N}) \le \text{target\_flops} \end{aligned} \tag{2}$$

where $w, d, r$ are coefficients for scaling network width, depth, and resolution; $\hat{\mathcal{F}}_i, \hat{L}_i, \hat{H}_i, \hat{W}_i, \hat{C}_i$ are predefined parameters in baseline network (see Table 1 as an example).

$$\begin{aligned} \text{depth: } & d = \alpha^\phi \\ \text{width: } & w = \beta^\phi \\ \text{resolution: } & r = \gamma^\phi \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \ge 1, \beta \ge 1, \gamma \ge 1 \end{aligned} \tag{3}$$

# 類神經模型結構 - EfficientNet

[5]




| | 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** |

*Figure 5.* **FLOPS vs. ImageNet Accuracy –** Similar to Figure 1 except it compares FLOPS rather than model size.

# 相關參數 - EfficientNet

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
efficientnet-b7 (Model)      (None, 16, 16, 2560)      64097680
_________________________________________________________________
global_average_pooling2d_1 ( (None, 2560)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 10244
=================================================================
Total params: 64,107,924
Trainable params: 63,797,204
Non-trainable params: 310,720
_________________________________________________________________
```

output: 4個類別

Param: 10244 = (2560+1)*4

# 相關參數 - EfficientNet-modified

| Layer (type) | Output Shape | Param # |
|---|---|---|
| efficientnet-b7 (Model) | (None, 16, 16, 2560) | 64097680 |
| global_average_pooling2d_1 ( | (None, 2560) | 0 |
| dense_3 (Dense) | (None, 2560) | 6556160 |
| dense_4 (Dense) | (None, 2560) | 6556160 |
| dense_5 (Dense) | (None, 4) | 10244 |

Total params: 77,220,244
Trainable params: 76,909,524
Non-trainable params: 310,720

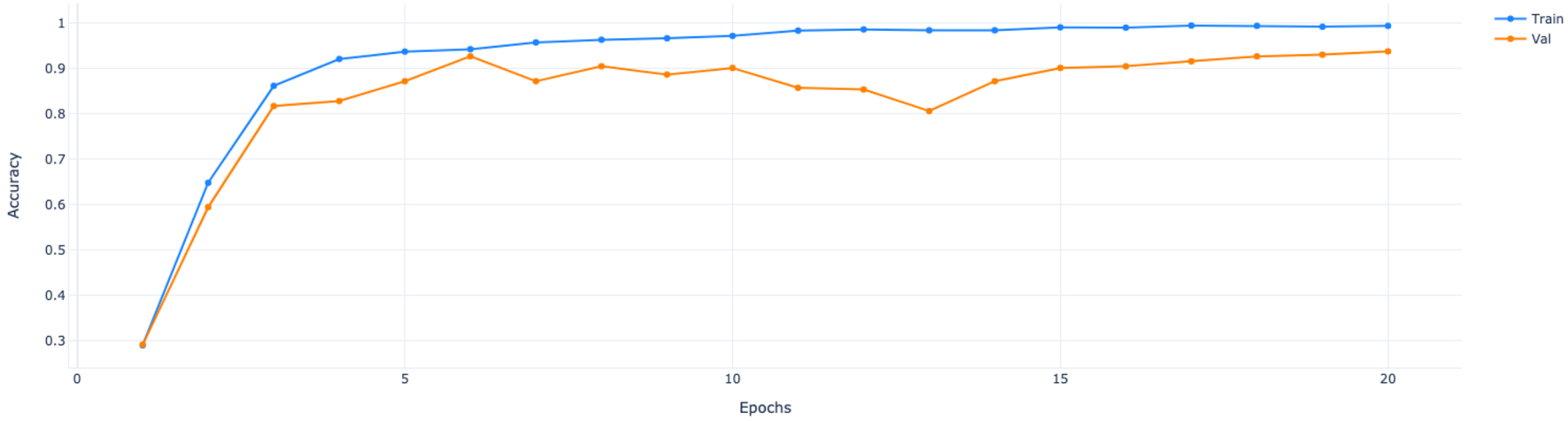
output: 4個類別

Param:

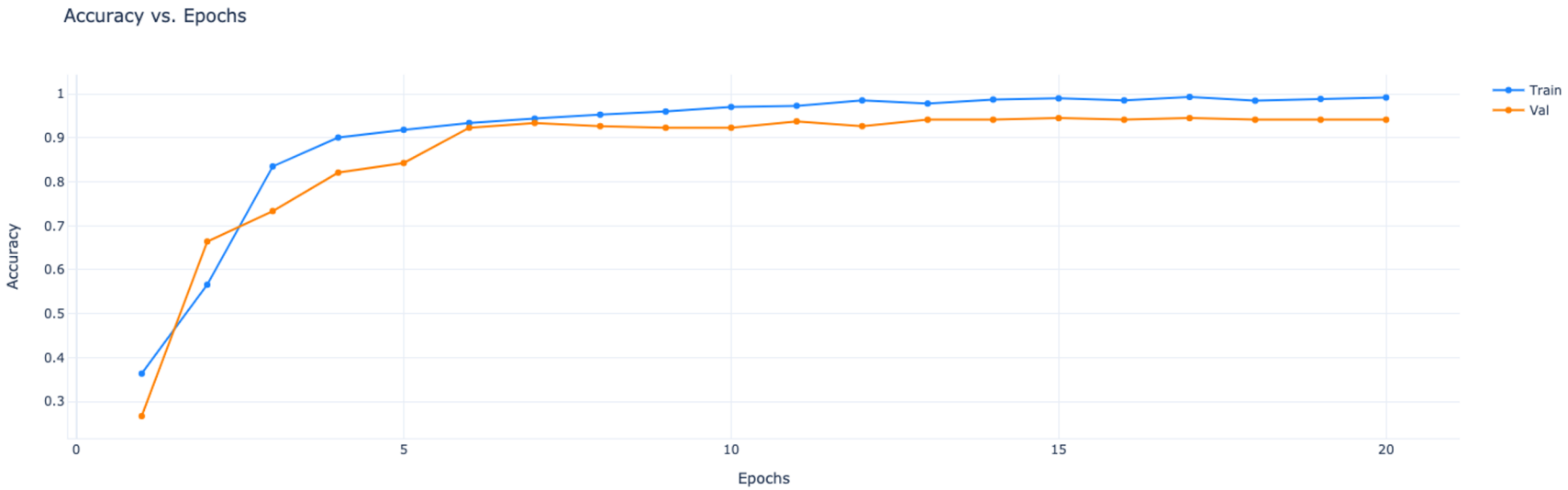
6556160 = (2560+1)*2560

10244 = (2560+1)*4

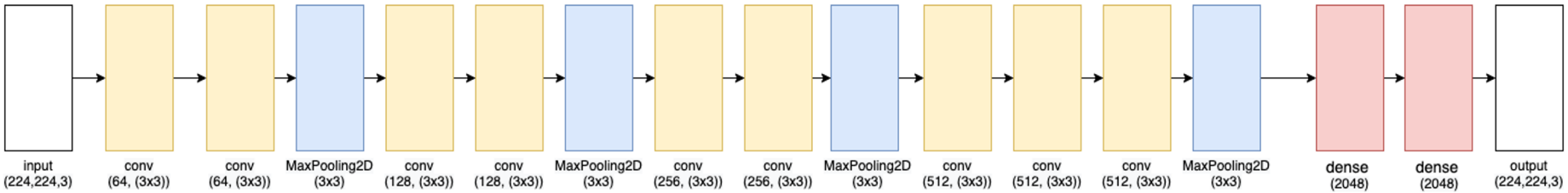
# 結果 - EfficientNet



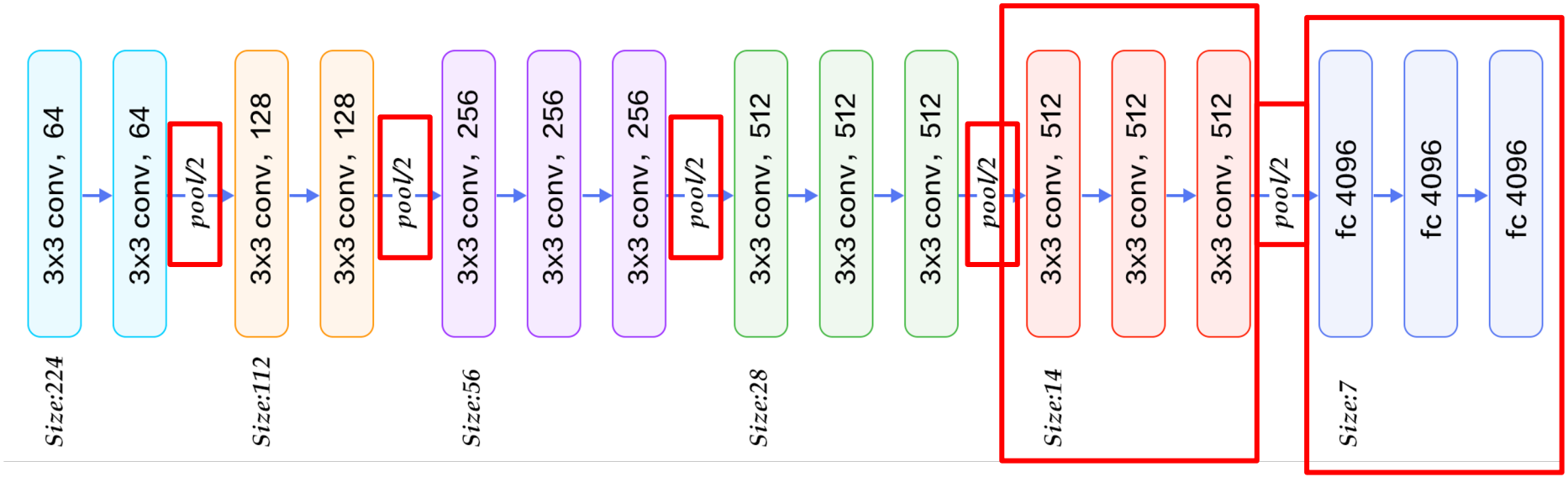
# 結果 - EfficientNet-modified



# 類神經模型結構 - leafNet (our model)



# 類神經模型結構 - leafNet與VGG19不同的地方



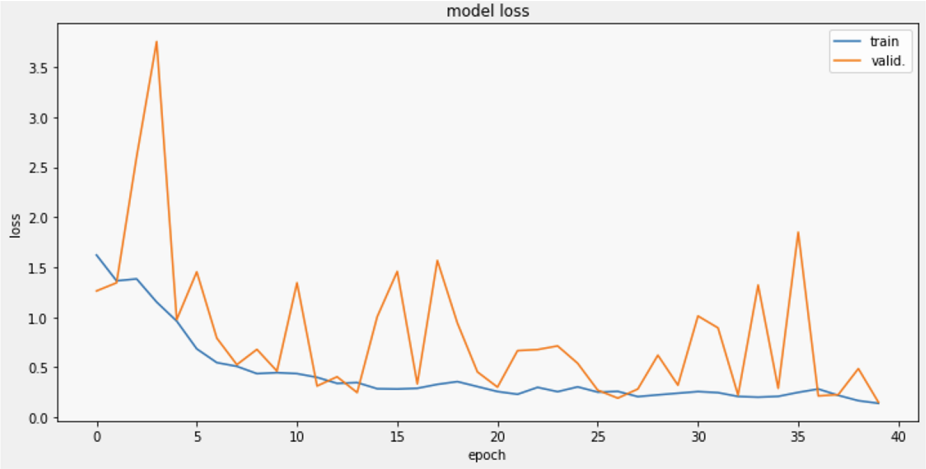
# 相關參數 - leafNet

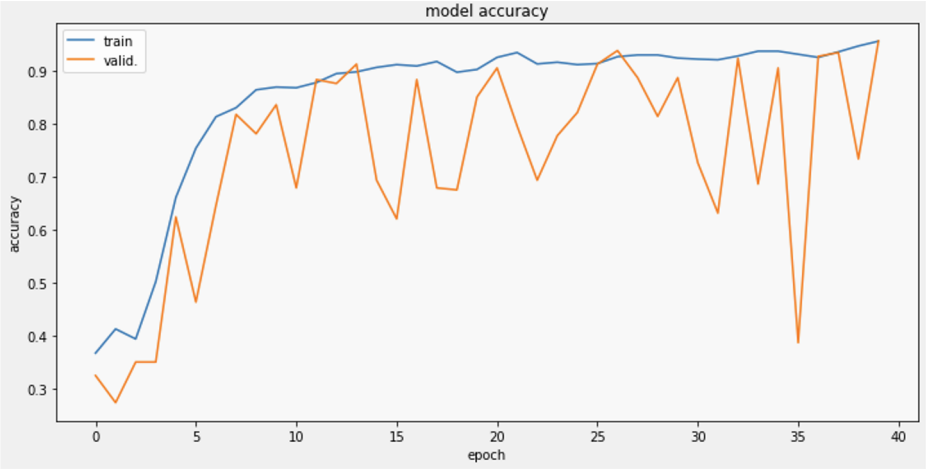
Model: "sequential_1"

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_9 (Conv2D) | (None, 224, 224, 64) | 1792 |
| batch_normalization_11 (Batc | (None, 224, 224, 64) | 256 |
| activation_12 (Activation) | (None, 224, 224, 64) | 0 |
| conv2d_10 (Conv2D) | (None, 224, 224, 64) | 36928 |
| batch_normalization_12 (Batc | (None, 224, 224, 64) | 256 |
| activation_13 (Activation) | (None, 224, 224, 64) | 0 |
| max_pooling2d_4 (MaxPooling2 | (None, 74, 74, 64) | 0 |
| conv2d_11 (Conv2D) | (None, 74, 74, 128) | 73856 |
| batch_normalization_13 (Batc | (None, 74, 74, 128) | 512 |
| activation_14 (Activation) | (None, 74, 74, 128) | 0 |
| conv2d_12 (Conv2D) | (None, 74, 74, 128) | 147584 |
| batch_normalization_14 (Batc | (None, 74, 74, 128) | 512 |
| activation_15 (Activation) | (None, 74, 74, 128) | 0 |
| max_pooling2d_5 (MaxPooling2 | (None, 24, 24, 128) | 0 |
| conv2d_13 (Conv2D) | (None, 24, 24, 256) | 295168 |
| batch_normalization_15 (Batc | (None, 24, 24, 256) | 1024 |
| activation_16 (Activation) | (None, 24, 24, 256) | 0 |
| conv2d_14 (Conv2D) | (None, 24, 24, 256) | 590080 |
| batch_normalization_16 (Batc | (None, 24, 24, 256) | 1024 |
| activation_17 (Activation) | (None, 24, 24, 256) | 0 |
| max_pooling2d_6 (MaxPooling2 | (None, 8, 8, 256) | 0 |
| conv2d_15 (Conv2D) | (None, 8, 8, 512) | 1180160 |
| batch_normalization_17 (Batc | (None, 8, 8, 512) | 2048 |
| activation_18 (Activation) | (None, 8, 8, 512) | 0 |
| conv2d_16 (Conv2D) | (None, 8, 8, 512) | 2359808 |
| batch_normalization_18 (Batc | (None, 8, 8, 512) | 2048 |
| activation_19 (Activation) | (None, 8, 8, 512) | 0 |
| conv2d_17 (Conv2D) | (None, 8, 8, 512) | 2359808 |
| batch_normalization_19 (Batc | (None, 8, 8, 512) | 2048 |
| activation_20 (Activation) | (None, 8, 8, 512) | 0 |
| max_pooling2d_7 (MaxPooling2 | (None, 2, 2, 512) | 0 |
| flatten_1 (Flatten) | (None, 2048) | 0 |
| dense_3 (Dense) | (None, 2048) | 4196352 |
| batch_normalization_20 (Batc | (None, 2048) | 8192 |
| activation_21 (Activation) | (None, 2048) | 0 |
| dense_4 (Dense) | (None, 2048) | 4196352 |
| batch_normalization_21 (Batc | (None, 2048) | 8192 |
| activation_22 (Activation) | (None, 2048) | 0 |
| dense_5 (Dense) | (None, 4) | 8196 |
| activation_23 (Activation) | (None, 4) | 0 |

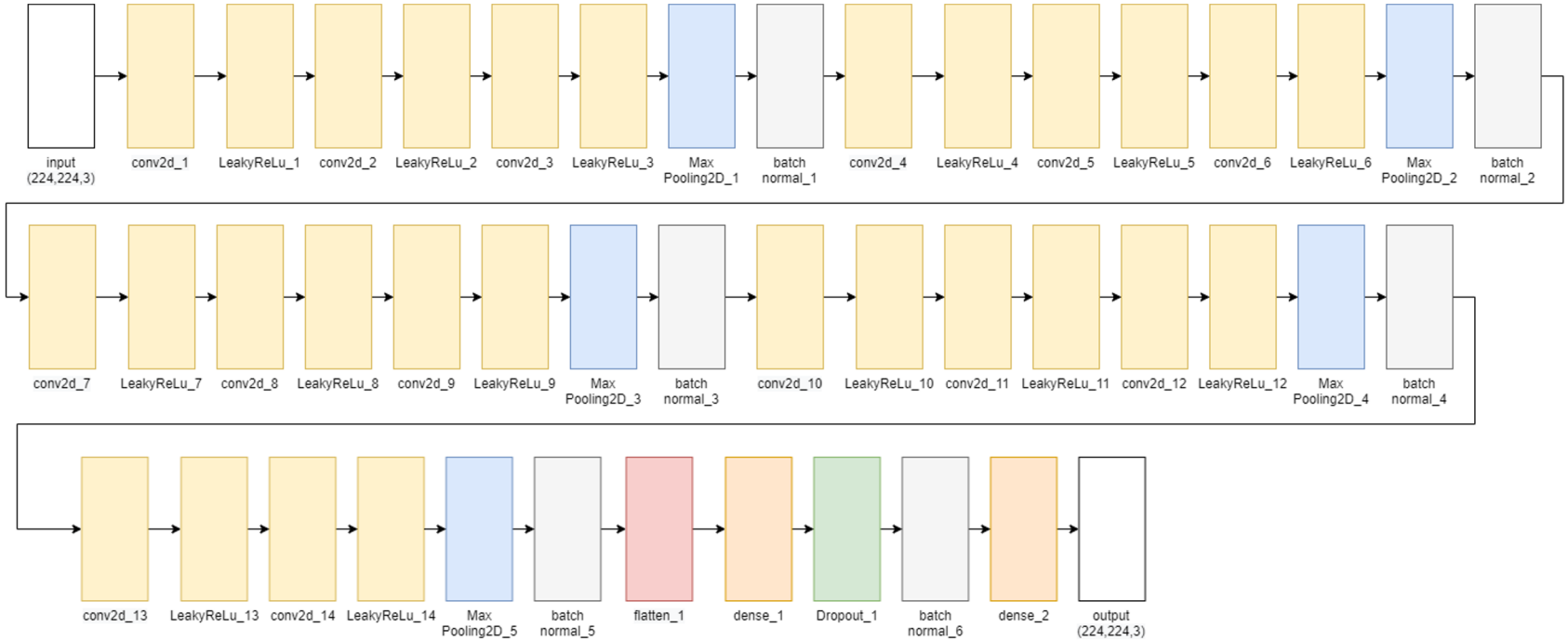
Total params: 15,472,196
Trainable params: 15,459,140
Non-trainable params: 13,056

# 結果 - leafNet


model loss
train
valid.
loss
epoch
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
5
10
15
20
25
30
35
40


model accuracy
train
valid.
accuracy
epoch
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
5
10
15
20
25
30
35
40

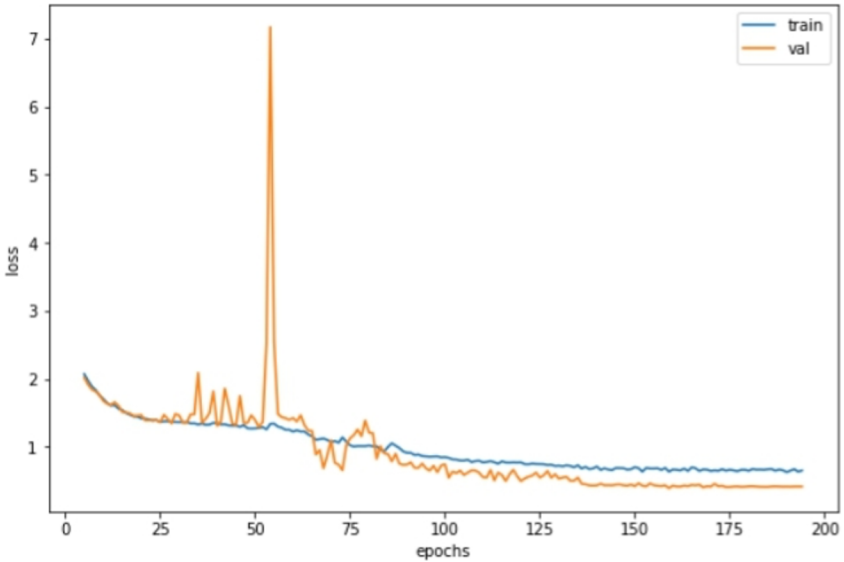
# 相關參數 - model - 2

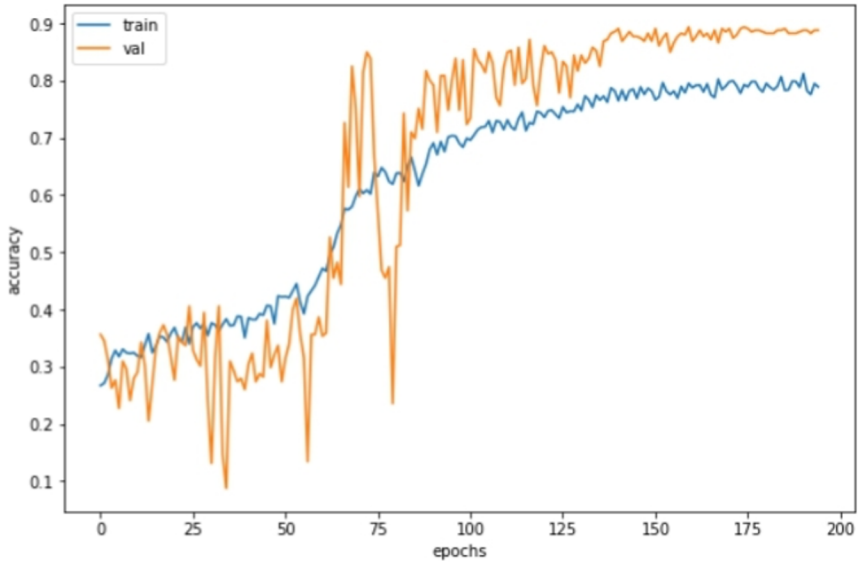

input (224,224,3)
conv2d_1
LeakyReLu_1
conv2d_2
LeakyReLu_2
conv2d_3
LeakyReLu_3
Max Pooling2D_1
batch normal_1
conv2d_4
LeakyReLu_4
conv2d_5
LeakyReLu_5
conv2d_6
LeakyReLu_6
Max Pooling2D_2
batch normal_2
conv2d_7
LeakyReLu_7
conv2d_8
LeakyReLu_8
conv2d_9
LeakyReLu_9
Max Pooling2D_3
batch normal_3
conv2d_10
LeakyReLu_10
conv2d_11
LeakyReLu_11
conv2d_12
LeakyReLu_12
Max Pooling2D_4
batch normal_4
conv2d_13
LeakyReLu_13
conv2d_14
LeakyReLu_14
Max Pooling2D_5
batch normal_5
flatten_1
dense_1
Dropout_1
batch normal_6
dense_2
output (224,224,3)

# 相關參數 - model - 2

```
Using TensorFlow backend.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 222, 222, 32)      896
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 222, 222, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 220, 220, 32)      9248
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 220, 220, 32)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 216, 216, 32)      25632
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 216, 216, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 108, 108, 32)      0
_________________________________________________________________
batch_normalization_1 (Batch (None, 108, 108, 32)      128
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 106, 106, 64)      18496
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 106, 106, 64)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 104, 104, 64)      36928
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 104, 104, 64)      0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 100, 100, 64)      102464
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 100, 100, 64)      0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 50, 50, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 50, 50, 64)        256
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 48, 48, 128)       73856
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 48, 48, 128)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 46, 46, 128)       147584
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 46, 46, 128)       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 42, 42, 128)       409728
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 42, 42, 128)       0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 21, 21, 128)       0
_________________________________________________________________
batch_normalization_3 (Batch (None, 21, 21, 128)       512
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 19, 19, 256)       295168
_________________________________________________________________
leaky_re_lu_10 (LeakyReLU)   (None, 19, 19, 256)       0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 17, 17, 256)       590080
_________________________________________________________________
leaky_re_lu_11 (LeakyReLU)   (None, 17, 17, 256)       0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 13, 13, 256)       1638656
_________________________________________________________________
leaky_re_lu_12 (LeakyReLU)   (None, 13, 13, 256)       0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 256)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 6, 6, 256)         1024
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 4, 4, 512)         1180160
_________________________________________________________________
leaky_re_lu_13 (LeakyReLU)   (None, 4, 4, 512)         0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 2, 2, 512)         2359808
_________________________________________________________________
leaky_re_lu_14 (LeakyReLU)   (None, 2, 2, 512)         0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 1, 1, 512)         0
_________________________________________________________________
batch_normalization_5 (Batch (None, 1, 1, 512)         2048
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                8208
_________________________________________________________________
dropout_1 (Dropout)          (None, 16)                0
_________________________________________________________________
batch_normalization_6 (Batch (None, 16)                64
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 68
=================================================================
Total params: 6,901,012
Trainable params: 6,898,996
Non-trainable params: 2,016
_________________________________________________________________
```

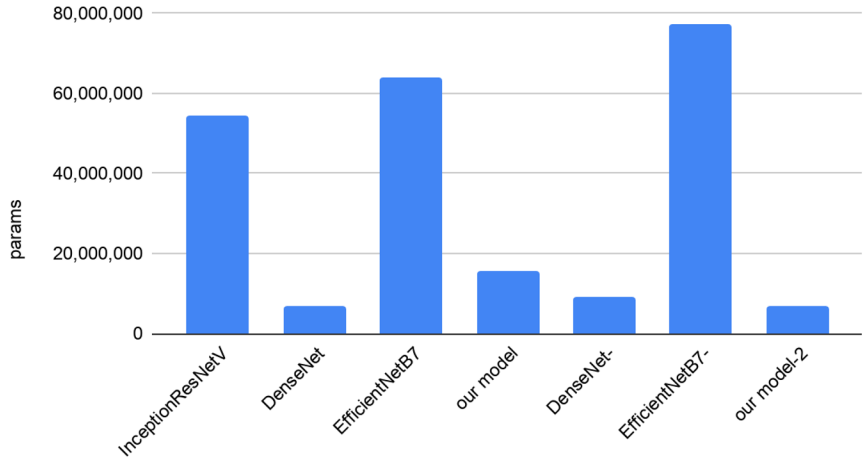
# 結果 - model - 2


train
val
loss
epochs
0
25
50
75
100
125
150
175
200
1
2
3
4
5
6
7

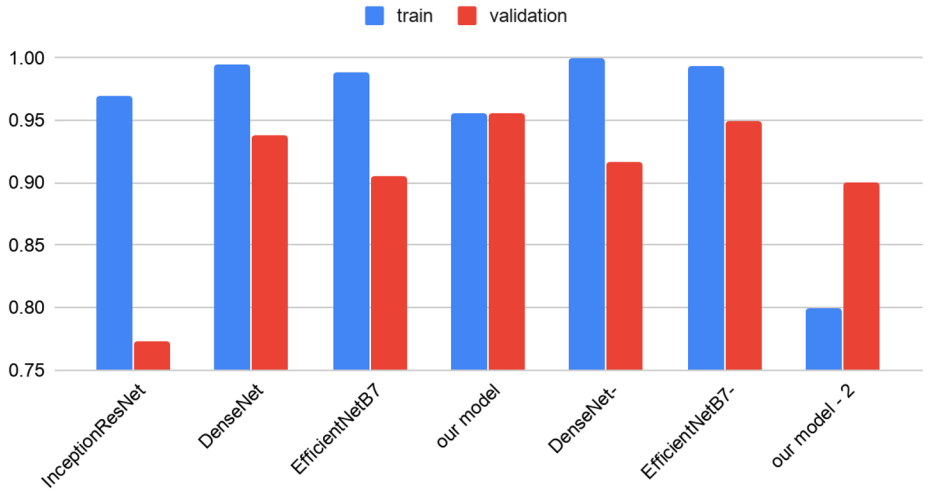

train
val
accuracy
epochs
0
25
50
75
100
125
150
175
200
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

# 綜合比較

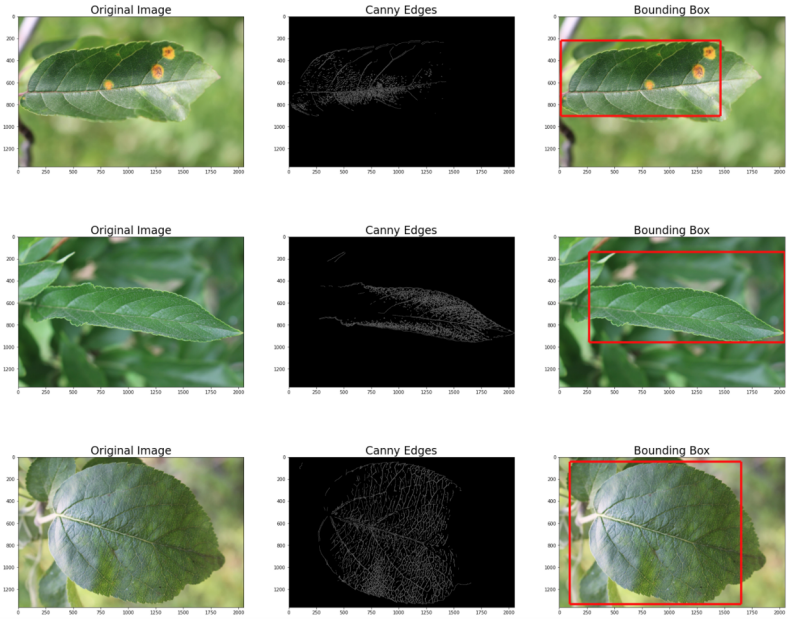
縦軸：params，横軸：




train和validation



# 可以改進的地方




[7]

(a) (b)

*Fig. 12: (a) Image inputted, (b) Image result*

# 實作講解

# Distributed training with TensorFlow

定義 strategy

```
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Running on TPU ', tpu.master())
except ValueError:
    tpu = None

if tpu:
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
    print('use tpu')
else:
    if tf.config.list_physical_devices('GPU'):
      strategy = tf.distribute.MirroredStrategy()
      print('use gpu')
    else:  # use default strategy
      print('use cpu')
      strategy = tf.distribute.get_strategy()
```

使用

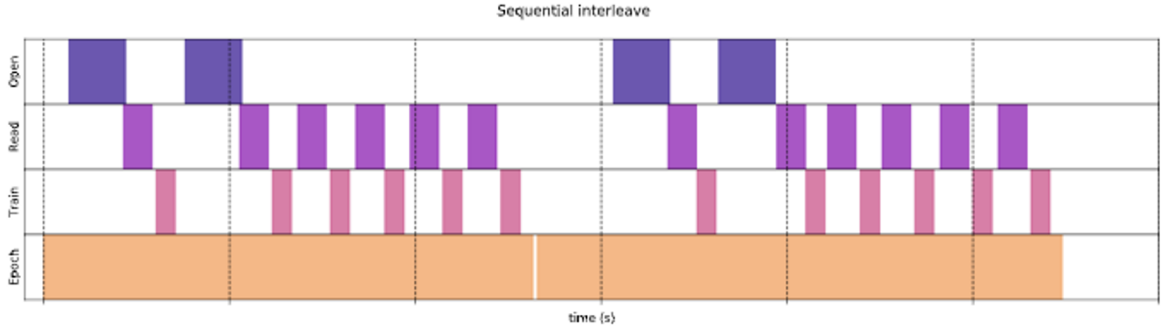
```
In [16]: with strategy.scope():
             model3 = Sequential()

             model3.add(Conv2D(64, (3, 3), input_shap
             model3.add(BatchNormalization(axis=-1))
             model3.add(Activation('relu'))
             model3.add(Conv2D(64, (3, 3),padding='sa
             model3.add(BatchNormalization(axis=-1))
             model3.add(Activation('relu'))
             model3.add(MaxPooling2D(pool_size=(3,3))
```

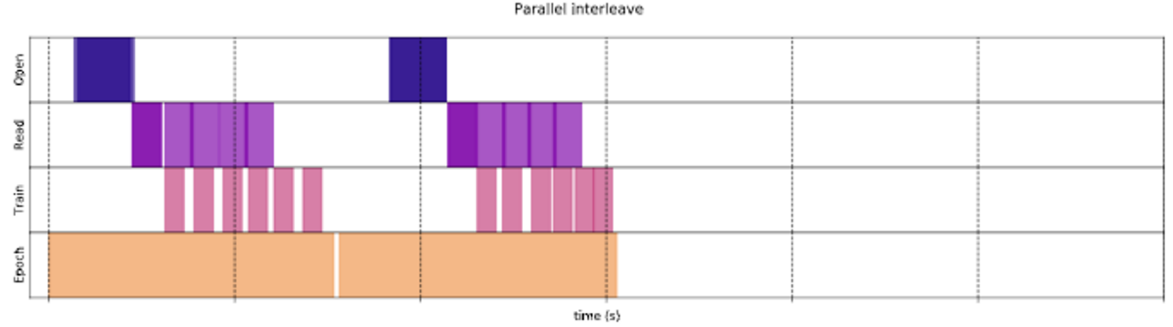
小提示： tf.config.list_physical_devices()

# Build TensorFlow input pipelines

origin




parallelized



# Build TensorFlow input pipelines

定義 tf.data.Dataset

```
train_dataset_1 = (
tf.data.Dataset
    .from_tensor_slices((train_paths, train_labels))
    .map(decode_image, num_parallel_calls=AUTO)
    .cache()
    .map(data_augment, num_parallel_calls=AUTO)
    .repeat()
    .shuffle(512)
    .batch(BATCH_SIZE)
    .prefetch(AUTO)
)
valid_dataset = (
    tf.data.Dataset
    .from_tensor_slices((valid_paths, valid_labels))
    .map(decode_image, num_parallel_calls=AUTO)
    .batch(BATCH_SIZE)
    .cache()
    .prefetch(AUTO)
)

test_dataset = (
    tf.data.Dataset
    .from_tensor_slices(test_paths)
    .map(decode_image, num_parallel_calls=AUTO)
    .map(data_augment, num_parallel_calls=AUTO)
    .batch(BATCH_SIZE)
)
```

使用

```
history3 = model3.fit(
    train_dataset_1,
    epochs=EPOCHS,
    callbacks=[es],
    steps_per_epoch=STEPS_PER_EPOCH,
    validation_data=valid_dataset if SPLIT_VALIDATION else None,
)
```

# save model / load model

save

```
!mkdir -p saved_model
model3.save('saved_model/our-model')
import pandas as pd
pd.DataFrame.from_dict(history3.history).to_csv('history-1.csv',index=False)
```

```
INFO:tensorflow:Assets written to: saved_model/our-model/assets
```

小提示: 查看model存不存在

```
!ls saved_model
!ls saved_model/model-1
```

```
model-1  model-1-1
assets  saved_model.pb  variables
```

load

```
InceptionResNetV2 = tf.keras.models.load_model('saved_model/model-1')
InceptionResNetV2.summary()
```

```
Current values:
NotebookApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_resnet_v2 (Model)  (None, 23, 23, 1536)      54336736
_________________________________________________________________
global_max_pooling2d (Global (None, 1536)              0
_________________________________________________________________
dense (Dense)                (None, 4)                 6148
=================================================================
Total params: 54,342,884
Trainable params: 54,282,340
Non-trainable params: 60,544
```

# 相關連結

1. https://www.kaggle.com/alanhc/fork-of-plant-pathology-2020-eda-models?scriptVersionId=37593370
2. https://www.kaggle.com/alanhc/fork-of-plant-pathology-2020-eda-models?scriptVersionId=37674876
3. http://120.125.83.237:8086/notebooks/demo.ipynb
4. http://120.125.83.237:8086/notebooks/week18%20-%20IncepresnetV2%2BENB7-Copy1.ipynb
5. http://120.125.83.237:8086/notebooks/week18%20-%20IncepresnetV2%2BENB7-Copy2.ipynb
6. https://colab.research.google.com/drive/1WBNOQE1CzBEuRrvgKjlPUACdK-eYPzFk?usp=sharing
7. https://docs.google.com/presentation/d/1S1qe0ZPdjJQNO04-AOHunQ4qWvtONIWou6CSfR1ENGs/edit?usp=sharing
8. https://docs.google.com/document/d/1G7mgdnRqBf98OWn2IThESGO2uYmfNhpiA8Fc0BKG4U0/edit?usp=sharing

# 附錄 - efficientNet計算-以Bo為例

*Table 1.* **EfficientNet-B0 baseline network** – Each row describes a stage $i$ with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

| Stage $i$ | Operator $\hat{\mathcal{F}}_i$ | Resolution $\hat{H}_i \times \hat{W}_i$ | #Channels $\hat{C}_i$ | #Layers $\hat{L}_i$ |
|---|---|---|---|---|
| 1 | Conv3x3 | $224 \times 224$ | 32 | 1 |
| 2 | MBConv1, k3x3 | $112 \times 112$ | 16 | 1 |
| 3 | MBConv6, k3x3 | $112 \times 112$ | 24 | 2 |
| 4 | MBConv6, k5x5 | $56 \times 56$ | 40 | 2 |
| 5 | MBConv6, k3x3 | $28 \times 28$ | 80 | 3 |
| 6 | MBConv6, k5x5 | $14 \times 14$ | 112 | 3 |
| 7 | MBConv6, k5x5 | $14 \times 14$ | 192 | 4 |
| 8 | MBConv6, k3x3 | $7 \times 7$ | 320 | 1 |
| 9 | Conv1x1 & Pooling & FC | $7 \times 7$ | 1280 | 1 |

Starting from the baseline EfficientNet-B0, we apply our compound scaling method to scale it up with two steps:

- STEP 1: we first fix $\phi = 1$, assuming twice more resources available, and do a small grid search of $\alpha, \beta, \gamma$ based on Equation 2 and 3. In particular, we find the best values for EfficientNet-B0 are $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$, under constraint of $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.
- STEP 2: we then fix $\alpha, \beta, \gamma$ as constants and scale up baseline network with different $\phi$ using Equation 3, to obtain EfficientNet-B1 to B7 (Details in Table 2).

Notably, it is possible to achieve even better performance by searching for $\alpha, \beta, \gamma$ directly around a large model, but the search cost becomes prohibitively more expensive on larger models. Our method solves this issue by only doing search once on the small baseline network (step 1), and then use the same scaling coefficients for all other models (step 2).

# 參考


1. Thapa, R., Snavely, N., Belongie, S., & Khan, A. (2020). The Plant Pathology 2020 challenge dataset to classify foliar disease of apples. *arXiv preprint arXiv:2004.11958*.
2. Mahdianpari, M., Salehi, B., Rezaee, M., Mohammadimanesh, F., & Zhang, Y. (2018). Very deep convolutional neural networks for complex land cover mapping using multispectral remote sensing imagery. *Remote Sensing*, *10*(7), 1119.
3. Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017, February). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*.
4. Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700-4708).
5. Tan, M., & Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*.
6. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 2820-2828).
7. Veites-Campos, S. A., Ramírez-Betancour, R., & González-Pérez, M. (2018). Identification of cocoa pods with image processing and artificial neural networks. *International Journal of Advanced Engineering, Management and Science*, *4*(7).

# 參考


8. Mazen, F. M., & Nashat, A. A. (2019). Ripeness classification of bananas using an artificial neural network. *Arabian Journal for Science and Engineering*, *44*(8), 6901-6910.
9. Hassan, N. M. H., & Nashat, A. A. (2019). New effective techniques for automatic detection and classification of external olive fruits defects based on image processing techniques. *Multidimensional Systems and Signal Processing*, *30*(2), 571-589.
10. Basri, H., Syarif, I., Sukaridhoto, S., & Falah, M. F. (2019). INTELLIGENT SYSTEM FOR AUTOMATIC CLASSIFICATION OF FRUIT DEFECT USING FASTER REGION-BASED CONVOLUTIONAL NEURAL NETWORK (FASTER R-CNN). *Jurnal Ilmiah Kursor*, *10*(1).
11. Sahu, D., & Dewangan, C. (2017). Identification and classification of mango fruits using image processing. *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol*, *2*(2), 203-210.
12. Sahu, D., & Potdar, R. M. (2017). Defect identification and maturity detection of mango fruits using image analysis. *American Journal of Artificial Intelligence*, *1*(1), 5-14.
13. Thendral, R., & Suhasini, A. (2017). Automated skin defect identification system for orange fruit grading based on genetic algorithm. *Current Sci*, *112*(8), 1704-1711.
14. Ireri, D., Belal, E., Okinda, C., Makange, N., & Ji, C. (2019). A computer vision system for defect discrimination and grading in tomatoes using machine learning and image processing. *Artificial Intelligence in Agriculture*, *2*, 28-37.