明志科技大學考試命題用紙

109 學年度第 2 學期 期中考試 碩士生 高等類神經網路 課程試題 共 1 頁第 1 頁

注意:□ 准使用計算器。□ 准參考筆記或書本。□另附答案紙每人 0 張 考試時間: 80 分鐘

命題教師___陳思翰_____(簽章)_110_年_3_月_29_日 班級:碩管一甲 座號: M09218001 姓名__周彥廷__
班級:碩管一甲 座號: M09218010 姓名__葉庭佑__

請以影像分類的概念分析工安相關的影像集。

1.請介紹你所下載的影像集。
(1)資料集名稱與來源為何
(2)描述影像集的基礎訊息

(1)資料集來源使用
Kaggle 資料庫
A.Hardhat and Safety Vest Image for Object Detection
https://www.kaggle.com/johnsyin97/hardhat-and-safety-vest-image-for-object-detection

B. fire smoke and neutral
https://www.kaggle.com/slirq123/fire-smoke-and-neutral

C.MASK DeepBelief.ai 深度學習社團
https://drive.google.com/file/d/1DzjHz-rf-ZZtEMzdjix6AJe9YSbX0UTA/view?fbclid=IwAR0GomXuS4PlGTe8--VdSGBCWM2pTEtF4qMs_OOpxD0omMqsT-OxCjFmk-Q

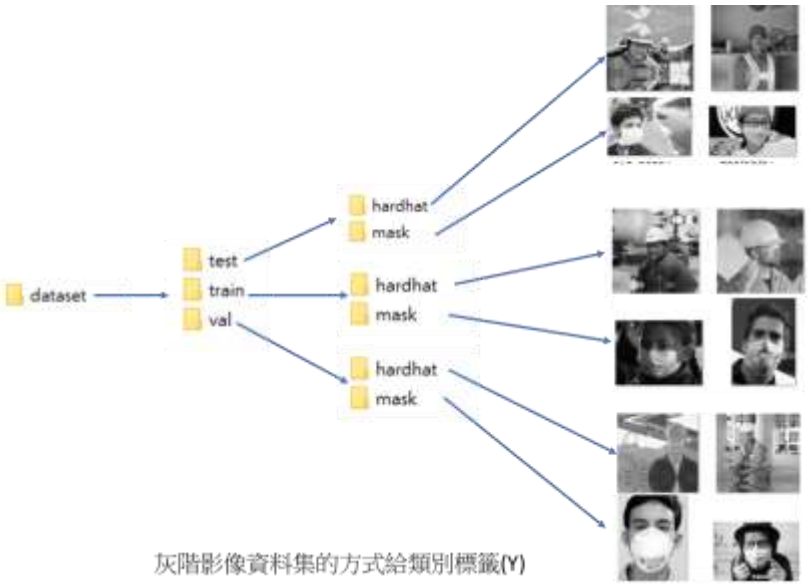
(2)描述影像集的基礎訊息
A.安全帽：內容含有在不同工地情境下，工人穿戴安全帽影像資料集(5000 張)。
目的：收集工人穿戴安全帽特徵影像(500 張，尺寸 416x416)
B. 火焰(Fire)和煙(Smoke)的影像(共 2000 張，尺寸 300x200)
目的：收集火焰與煙霧特徵影像(1000 張)
C.口罩：內容為人臉配戴口罩影像資料集(1547 張)。
目的：收集資料人臉配戴口罩特徵影像(500 張，尺寸 300x200)。

本次專題針對工廠人員常見配戴設備(安全帽、口罩)與可能危機發生(火與煙)進行資料收集，透過類神經網路學習分類上述特徵，進而用於作業現場實現工安影像辨識。
資料集(2000 張)分為 4 類：1.安全帽 2.口罩 3 火焰 4.煙霧。

2. 建模前請執行前處理？
(1)請做 50%-10%-40%的資料分割
(2)是否需做影像前處理
(3)是否需做影像擴增

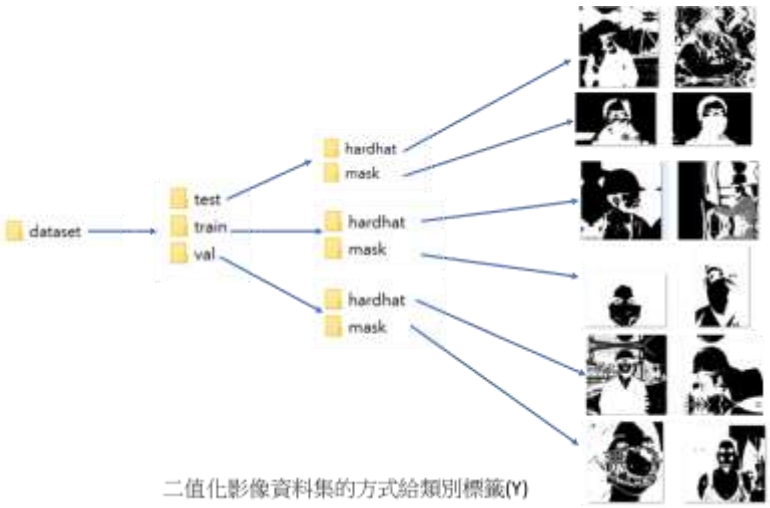
1.依照題意比例切分資料集 1.train 資料(1000 張) 2.val 資料(200 張) 3.test 資料(800 張)
2.設計一簡易訓練實驗，於初步測試後採用影像轉為(1 灰階、2.二值化)使用 LeNet-5 訓練為例，檢視結果在此案例中是否需要作影像前處理。

1.將 RGB 影像轉為灰階 3 通道



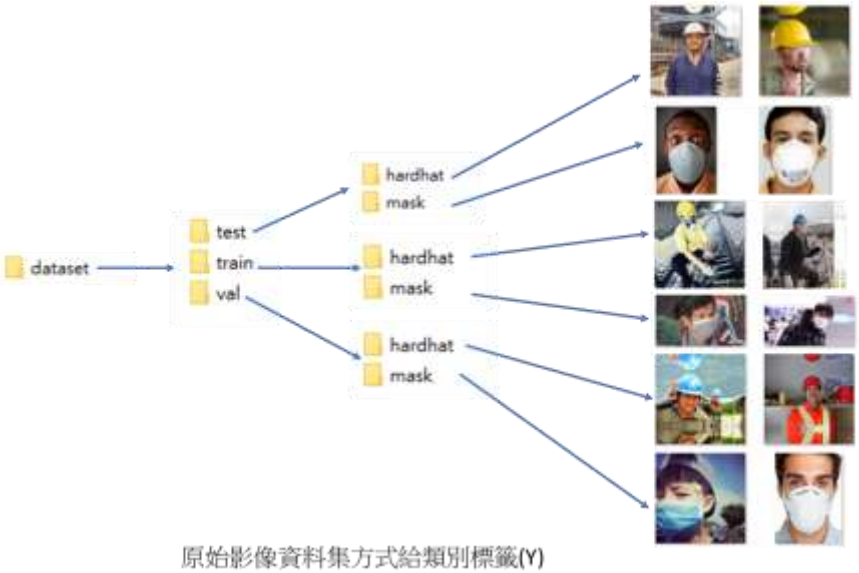
灰階影像資料集的方式給類別標籤(Y)

2.將 RGB 影像作二值化(閾值設 127)



二值化影像資料集的方式給類別標籤(Y)

3.不做影像前處理



原始影像資料集方式給類別標籤(Y)

因考量計算硬體效能有限，此處使用 A.安全帽(取 100 張)，B.口罩(取 100 張)，作為資料集，並根據題目要求進行 50%-10%-40%的資料分割，使用 LeNet-5 進行訓練，訓練結果如表一顯示，於此資料集分類問題中，將影像轉為灰階或二值化，在正確率上並無得到較好的表現，因此後續 CNN 建模資料來源會採用原始影像資料集訓練。

程式碼：

```
#利用 opencv 進行轉灰階三通道
import cv2
import os
def read_path(file_pathname):
    #使用 for 迴圈讀取目錄下所有影像
    for filename in os.listdir(file_pathname):
    print(filename)
    img = cv2.imread(file_pathname+'/'+filename)
    img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    image_np=cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)

    #ret, binary = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV)(閾值設 127)

    cv2.imwrite('C:/Users/MCUT/Desktop/gray'+"/"+filename,image_np)
    #cv2.imwrite('C:/Users/MCUT/Desktop/B_M'+"/"+filename,binary) #寫出二值化影像

read_path("C:/Users/MCUT/Desktop/hadhat1") #設定讀取路徑
```

表一、三種情況之訓練、驗證與測試資料正確率整理

Metrics	**1.轉灰階影像**	**2.影像二值化**	**3.不作影像前處理**
Training accuracy	0.86	0.87	0.91
Validation accuracy	0.75	0.75	0.85
Testing accuracy	0.79	0.79	0.81

(3)是否需做影像擴增

需要進行影像擴增(Image augmentation)，彌補資料不足，並能增強電腦在訓練時學習到不同情境之相同事物與避免過擬合的結果產生。

將進行(旋轉、寬度方向偏移、水平翻轉、高度方向偏移、調整亮度與縮放)。

```
from keras.preprocessing.image import ImageDataGenerator
ImageDataGenerator(rescale=1./255,
rotation_range=,   #旋轉
width_shift_range=,   #寬度方向偏移
horizontal_flip=True, #水平翻轉
height_shift_range=, #高度方向偏移
brightness_range =[0.9,1.1], #調整亮度
zoom_range = [0.9, 1.1]) #縮放
```

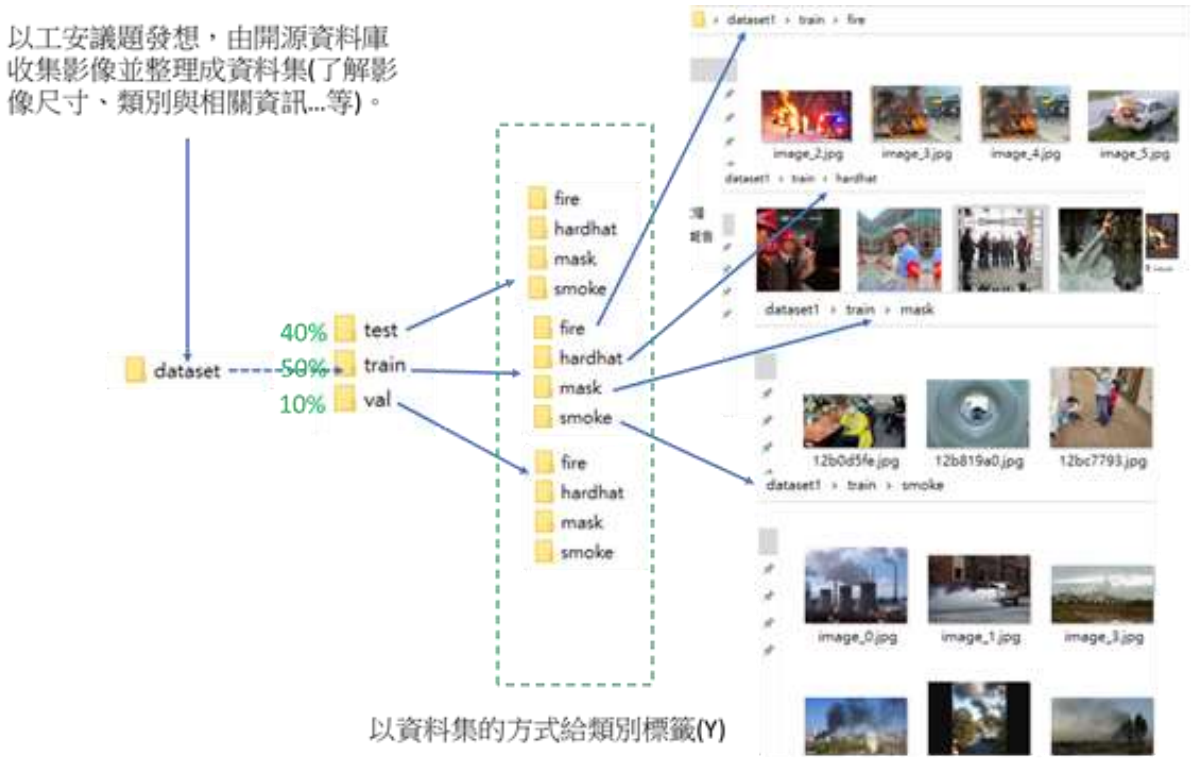
3.建模與解釋。

(1)選擇 3~5 種 CNN 模型以訓練影像集進行建模

(2)請以驗證正確率或其他指標做一些超參數微調實驗

(3)以測試影像集計算混淆矩陣與錯誤率

(4)試針對最佳的 CNN 模型，以 Grad-CAM 做 CNN 視覺化的解釋



程式碼：

```
#設定分析影像大小、檔案路徑
image_size =450   #設定圖片 size (視運算效能與情況作調整)
path_train = 'C:/Users/MCUT/Desktop/mid_dataset/train/'
path_validation = 'C:/Users/MCUT/Desktop/mid_dataset/val/'
path_test = 'C:/Users/MCUT/Desktop/mid_dataset/test/'
#設定訓練、驗證資料的 Python 產生器，並將圖片進行數據擴增
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255, # 壓縮數值範圍
  rotation_range=3, # 旋轉
  width_shift_range=0.1, #寬度方向偏移
  horizontal_flip=True, #水平翻轉
  height_shift_range=0.1, #高度方向偏移
  brightness_range =[0.9,1.1], #調整亮度
  zoom_range = [0.9, 1.1]) #縮放
validation_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(path_train, # 訓練目錄
  target_size=(image_size, image_size), # 調整所有影像大小
  batch_size=40,   #調整批量大小
  class_mode='categorical') # 類別模式為多類別
validation_generator = validation_datagen.flow_from_directory(path_validation,
  target_size=(image_size, image_size), batch_size=40)   #調整批量大小
```

#1.LeNet-5

```
from keras import layers
from keras import models
from keras import optimizers
#import numpy as np
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', kernel_initializer = 'lecun_uniform',
input_shape=(image_size, image_size, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(train_generator.num_classes, activation='softmax'))
model.summary() #顯示模型結構與參數量
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 448, 448, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 224, 224, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 222, 222, 64)      18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 111, 111, 64)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 109, 109, 128)     73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 54, 54, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 52, 52, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 26, 26, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 86528)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 86528)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               44302848
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 2052
=================================================================
Total params: 44,545,732
Trainable params: 44,545,732
Non-trainable params: 0
```

```
from keras.models import Sequential
from keras.layers import Dense,Flatten,Dropout
from keras.layers.convolutional import Conv2D,MaxPooling2D
from keras.utils.np_utils import to_categorical
from keras import optimizers
import numpy as np

#2.AlexNet
model = Sequential()
model.add(Conv2D(96,(11,11),strides=(4,4),input_shape=(image_size,
image_size,3),padding='valid',activation='relu',kernel_initializer='uniform'))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))
model.add(Conv2D(256,(5,5),strides=(1,1),padding='same',activation='relu',kernel_initializer='uniform'))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))
model.add(Conv2D(384,(3,3),strides=(1,1),padding='same',activation='relu',kernel_initializer='uniform'))
model.add(Conv2D(384,(3,3),strides=(1,1),padding='same',activation='relu',kernel_initializer='uniform'))
model.add(Conv2D(256,(3,3),strides=(1,1),padding='same',activation='relu',kernel_initializer='uniform'))
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))
model.add(Flatten())
model.add(Dense(4096,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(train_generator.num_classes,activation='softmax'))
model.summary() #顯示模型結構與參數量
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 110, 110, 96)	34944
max_pooling2d_1 (MaxPooling2	(None, 54, 54, 96)	0
conv2d_2 (Conv2D)	(None, 54, 54, 256)	614656
max_pooling2d_2 (MaxPooling2	(None, 26, 26, 256)	0
conv2d_3 (Conv2D)	(None, 26, 26, 384)	885120
conv2d_4 (Conv2D)	(None, 26, 26, 384)	1327488
conv2d_5 (Conv2D)	(None, 26, 26, 256)	884992
max_pooling2d_3 (MaxPooling2	(None, 12, 12, 256)	0
flatten_1 (Flatten)	(None, 36864)	0
dense_1 (Dense)	(None, 4096)	150999040
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 4096)	16781312
dropout_2 (Dropout)	(None, 4096)	0
dense_3 (Dense)	(None, 4)	16388

```
Total params: 171,543,940
Trainable params: 171,543,940
Non-trainable params: 0
```

#使用 Keras Applications 導入 Pre-trained model 以及權重檔 'imagenet'

#3.VGG 19

```
from keras.applications.vgg16 import VGG16 #VGG16
from keras.applications.vgg19 import VGG19
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D, Dropout, Flatten
from keras import optimizers
from keras.optimizers import SGD
backbone = VGG19(weights='imagenet', include_top=False, input_shape=(image_size, image_size, 3))
x = backbone.output
x = GlobalAveragePooling2D()(x) # 額外接全域平均池化(Global Average Pooling，GAP)層
x = Dense(256, activation='relu')(x) # 接 FC 層，選擇 ReLU 活化函數
x = Dropout(0.5)(x) # 使用 Dropout 層
predictions = Dense(train_generator.num_classes, activation='softmax')(x) # 接 Softmax 層
model = Model(inputs=backbone.input, outputs=predictions)
model.summary() #顯示模型結構與參數量
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 450, 450, 3)	0
block1_conv1 (Conv2D)	(None, 450, 450, 64)	1792
block1_conv2 (Conv2D)	(None, 450, 450, 64)	36928
block1_pool (MaxPooling2D)	(None, 225, 225, 64)	0
block2_conv1 (Conv2D)	(None, 225, 225, 128)	73856
block2_conv2 (Conv2D)	(None, 225, 225, 128)	147584
block2_pool (MaxPooling2D)	(None, 112, 112, 128)	0
block3_conv1 (Conv2D)	(None, 112, 112, 256)	295168
block3_conv2 (Conv2D)	(None, 112, 112, 256)	590080
block3_conv3 (Conv2D)	(None, 112, 112, 256)	590080
block3_conv4 (Conv2D)	(None, 112, 112, 256)	590080
block3_pool (MaxPooling2D)	(None, 56, 56, 256)	0
block4_conv1 (Conv2D)	(None, 56, 56, 512)	1180160
block4_conv2 (Conv2D)	(None, 56, 56, 512)	2359808
block4_conv3 (Conv2D)	(None, 56, 56, 512)	2359808
block4_conv4 (Conv2D)	(None, 56, 56, 512)	2359808
block4_pool (MaxPooling2D)	(None, 28, 28, 512)	0
block5_conv1 (Conv2D)	(None, 28, 28, 512)	2359808
block5_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block5_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block5_conv4 (Conv2D)	(None, 28, 28, 512)	2359808
block5_pool (MaxPooling2D)	(None, 14, 14, 512)	0
global_average_pooling2d_1 (	(None, 512)	0
dense_3 (Dense)	(None, 256)	131328
dropout_2 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 4)	1028

```
Total params: 20,156,740
Trainable params: 20,156,740
Non-trainable params: 0
```

#4. Inception v3

```
from keras.layers import Dense, GlobalAveragePooling2D
from keras.applications import InceptionV3
conv_base = InceptionV3(weights='imagenet',
                        include_top=False,
                        input_shape=(image_size, image_size, 3))
conv_base.summary()

from keras import models
from keras import layers
from keras import optimizers
model= models.Sequential()
model.add(conv_base)
#model.add(layers.Flatten())
model.add(GlobalAveragePooling2D())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(train_generator.num_classes, activation='softmax'))
model.summary() #顯示模型結構與參數量
```

Layer (type)	Output Shape	Param #
inception_v3 (Model)	(None, 12, 12, 2048)	21802784
global_average_pooling2d_10	(None, 2048)	0
dense_8 (Dense)	(None, 2048)	4196352
dense_9 (Dense)	(None, 4)	8196

Total params: 26,007,332
Trainable params: 25,972,900
Non-trainable params: 34,432

#5. Xception

```
from keras.layers import Dense, GlobalAveragePooling2D, Dropout
from keras.applications import Xception
conv_base = Xception(weights='imagenet',
                     include_top=False,
                     input_shape=(image_size, image_size, 3))
conv_base.summary()

from keras import models
from keras import layers
from keras import optimizers
model= models.Sequential()
model.add(conv_base)
model.add(GlobalAveragePooling2D()) # GAP
model.add(layers.Dense(1024, activation='relu'))
model.add(Dropout(0.3)) # Dropout 層
model.add(layers.Dense(train_generator.num_classes, activation='softmax'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
xception (Model)             (None, 14, 14, 2048)      20861480
_________________________________________________________________
global_average_pooling2d_1 ( (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              2098176
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 4100
=================================================================
Total params: 22,963,756
Trainable params: 22,909,228
Non-trainable params: 54,528
```

#6.ResNet 34 (50)

```
from keras.applications import ResNet50
from keras.layers import Dense, GlobalAveragePooling2D, Dropout, Flatten
from keras import optimizers
from keras.optimizers import SGD

conv_base = ResNet50(weights='imagenet',
                     include_top=False,
                     input_shape=(image_size, image_size, 3))
conv_base.summary()
from keras import models
from keras import layers
from keras import optimizers

model= models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(1024, activation='relu'))
model.add(Dropout(0.5)) # 接 Dropout 層
model.add(layers.Dense(1024, activation='relu'))
model.add(Dropout(0.5)) # 接 Dropout 層
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dense(train_generator.num_classes, activation='softmax'))
model.summary()
```

Layer (type)	Output Shape	Param #
resnet50 (Model)	(None, 2, 2, 2048)	23587712
flatten_1 (Flatten)	(None, 8192)	0
dense_1 (Dense)	(None, 1024)	8389632
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 1024)	1049600
dropout_2 (Dropout)	(None, 1024)	0
dense_3 (Dense)	(None, 1024)	1049600
dense_4 (Dense)	(None, 1024)	1049600
dense_5 (Dense)	(None, 4)	4100

Total params: 35,130,244
Trainable params: 35,077,124
Non-trainable params: 53,120

```
# 模式編譯
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(lr=1e-5),
metrics=['acc']) #設定損失函數、優化器與學習率
# 模式訓練
history = model.fit_generator(
  train_generator,
  steps_per_epoch=30,   # 設定每完成一組 epoch 需要幾步
  epochs=20,   # 設定總訓練次數
  validation_data=validation_generator,
  validation_steps=30) # 設定 validation 步數
# 儲存 model
model.save('C:/Users/MCUT/Desktop/model.h5') #存成 mode .h5 檔
```

(1) 建立 CNN 模型對資料集進行訓練，分別是 LeNet5、AlexNet、VGG19 (16)、ResNet34 (50)、Inception v3 與 Xception，並將結果正確率整理如表二所示，由於考量電腦計算能力有限，在不發生 OOM (Out of Memory)情形，steps_per_epoch 設定 30 到 50, validation_steps 設定 20 到 50，epoch 設定 20 到 30，即可達到收斂趨勢，VGG19 與 VGG16 在此資料集訓練表現大致相同，在此列上 VGG19，另 ResNet50 混淆矩陣如圖一所示，經數次實驗調整後仍無法有效使降低 loss 更新權重，因此使用 ResNet34，

表二、記錄五種 CNN 模型訓練(train)、驗證(valid)與測試(test)正確率整理如下：

Metrics	LeNet 5	AlexNet	VGG 19	Inception v3	Xception	ResNet 34	DesNet121
Training accuracy	0.68	0.76	0.92	0.99	0.92(0.94)	0.82	0.96
Validation accuracy	0.61	0.71	0.82	0.91	0.82(0.83)	0.69	0.87
Testing accuracy	0.66	0.76	0.86	0.96	0.88(0.87)	0.74	0.94
Model epochs	20	20	20	20	20(30)	20	20

圖一、CNN 模型混淆矩陣數據整理如下：

Found 800 images belonging to 4 classes.

	precision	recall	f1-score	support
fire	0.84	0.72	0.77	200
hardhat	0.51	0.89	0.65	200
mask	0.63	0.51	0.57	200
smoke	0.88	0.52	0.65	200
accuracy			0.66	800
macro avg	0.71	0.66	0.66	800
weighted avg	0.71	0.66	0.66	800

```
[[144  41  12   3]
 [  6 178  11   5]
 [ 12  80 102   6]
 [ 10  51  36 103]]
```

cm_LeNet

Found 800 images belonging to 4 classes.

	precision	recall	f1-score	support
fire	0.76	0.90	0.82	200
hardhat	0.80	0.65	0.71	200
mask	0.77	0.68	0.72	200
smoke	0.71	0.81	0.76	200
accuracy			0.76	800
macro avg	0.76	0.76	0.75	800
weighted avg	0.76	0.76	0.75	800

```
[[179   3   9   9]
 [ 25 129  16  30]
 [ 13  24 135  28]
 [ 17   5  16 162]]
```

cm_AlexNet

	precision	recall	f1-score	support
fire	0.85	0.92	0.88	200
hardhat	0.95	0.81	0.88	200
mask	0.82	0.84	0.83	200
smoke	0.83	0.87	0.85	200
accuracy			0.86	800
macro avg	0.87	0.86	0.86	800
weighted avg	0.87	0.86	0.86	800

```
[[184   0   2  14]
 [  8 163  22   7]
 [ 10   8 168  14]
 [ 14   0  12 174]]
```

cm_VGG19

Found 800 images belonging to 4 classes.

	precision	recall	f1-score	support
fire	0.94	0.97	0.96	200
hardhat	0.95	0.99	0.97	200
mask	0.99	0.94	0.96	200
smoke	0.96	0.95	0.96	200
accuracy			0.96	800
macro avg	0.96	0.96	0.96	800
weighted avg	0.96	0.96	0.96	800

```
[[194   0   1   5]
 [  0 198   0   2]
 [  3  10 187   0]
 [  9   0   1 190]]
```

cm_Inception v3

Found 800 images belonging to 4 classes.

	precision	recall	f1-score	support
fire	0.90	0.84	0.87	200
hardhat	0.99	0.78	0.87	200
mask	0.91	0.96	0.94	200
smoke	0.77	0.94	0.85	200
accuracy			0.88	800
macro avg	0.89	0.88	0.88	800
weighted avg	0.89	0.88	0.88	800

```
[[169   0   1  30]
 [  7 155  16  22]
 [  2   1 193   4]
 [  9   0   2 189]]
```

cm_Xception epoch = 20

Found 800 images belonging to 4 classes.

	precision	recall	f1-score	support
fire	0.88	0.82	0.85	200
hardhat	1.00	0.74	0.85	200
mask	0.91	0.97	0.94	200
smoke	0.75	0.94	0.84	200
accuracy			0.87	800
macro avg	0.89	0.87	0.87	800
weighted avg	0.89	0.87	0.87	800

```
[[165   0   1  34]
 [  9 149  16  26]
 [  3   0 195   2]
 [ 10   0   2 188]]
```

cm_Xception epoch = 30

Found 800 images belonging to 4 classes.

	precision	recall	f1-score	support
fire	0.86	0.76	0.81	200
hardhat	0.69	0.74	0.71	200
mask	0.66	0.70	0.68	200
smoke	0.77	0.76	0.76	200
accuracy			0.74	800
macro avg	0.74	0.74	0.74	800
weighted avg	0.74	0.74	0.74	800

```
[[151  20  20   9]
 [ 10 148  23  19]
 [  3  39 140  18]
 [ 11   8  30 151]]
```

cm_ResNet34

Found 800 images belonging to 4 classes.

	precision	recall	f1-score	support
fire	0.00	0.00	0.00	200
hardhat	0.00	0.00	0.00	200
mask	0.00	0.00	0.00	200
smoke	0.25	1.00	0.40	200
accuracy			0.25	800
macro avg	0.06	0.25	0.10	800
weighted avg	0.06	0.25	0.10	800

```
[[  0   0   0 200]
 [  0   0   0 200]
 [  0   0   0 200]
 [  0   0   0 200]]
```

cm_ResNet50

(2)根據題意，選擇驗證正確率與測試正確率最高演算法(Inception v3)作為調整依據，並進行超參數微調實驗，探討模型效能表現是否能更好。

表三 紀錄更改超參數數值與對應訓練(train)、驗證(valid)與測試(test)正確率整理如下：

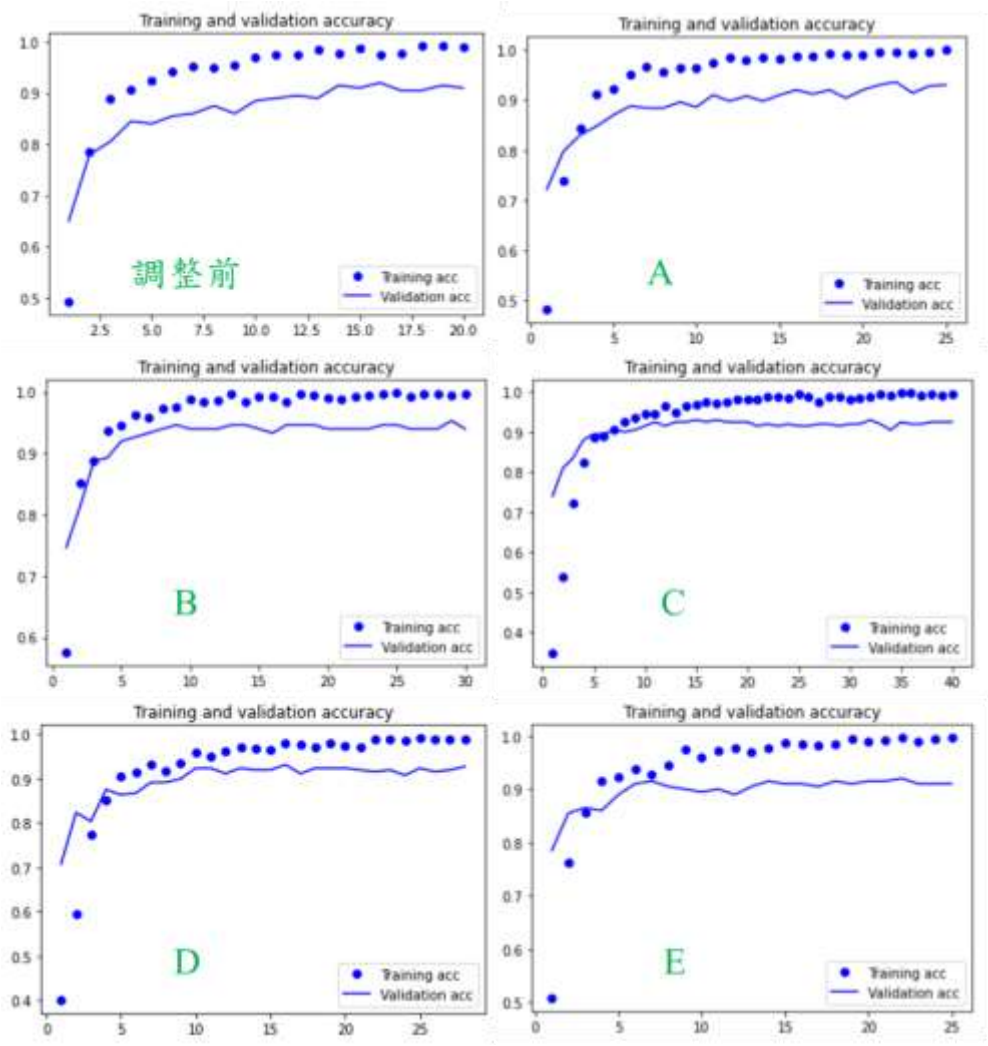
	image_size	影像擴增 batch_size	steps_per_epoch	validation_steps	epochs
A	350	20	35	20	25
B	350	15	50	10	30
C	350	15	60	15	40
D	420	10	100	20	28
E	400	10	80	80	25

Metrics	**未調整**	**A**	**B**	**C**	**D**	**E**
Training accuracy	0.99	1.00	0.99	0.99	0.99	0.99
Validation accuracy	0.91	0.93	0.94	0.92	0.92	0.91
Testing accuracy	0.96	0.97	0.98	0.97	0.98	0.98
Model epochs	20	25	30	40	28	25

ABC 組使用 ReLU + Dropout

DE 組使用使用 Leaky ReLU + Dropout

圖二、訓練資料與驗證資料訓練過程



Inception v3 演算法對此資料集在 epochs 為(20 到 40 間) Training accuracy 與 Testing accuracy 即可有效地收斂，但 Validation accuracy 仍有差距，策略 1：Inception v3 已做(Batch Normalization, BN)，後面接上 ReLU、LeakyReLU 搭配 Dropout 以維持梯度的穩定，策略 2：將 epoch 調高，嘗試增加至 100，但發現 epoch 50 之後 Validation loss 不降反增，此問題也於實習課請教工程師，可能訓練次數不夠多，要大於某次數後才會有顯著效果，由於時間與資源有限，所以僅將實驗成果呈現出來。

(3) 根據表三結果，選擇 B 組超參數所訓練之模型作為測試影像集計算混淆矩陣與錯誤率

```
Found 800 images belonging to 4 classes.
              precision    recall  f1-score   support

        fire       0.96      0.98      0.97       200
     hardhat       0.98      0.99      0.99       200
        mask       0.99      0.96      0.97       200
       smoke       0.98      0.97      0.97       200

    accuracy                           0.98       800
   macro avg       0.98      0.98      0.98       800
weighted avg       0.98      0.98      0.98       800

[[196   0   1   3]
 [  0 199   0   1]
 [  3   5 192   0]
 [  5   0   1 194]]
```

由混淆矩陣得到測試影像集的正確率 0.98，測試影像資料錯誤率 0.02。

(4)簡介 CAM 用途與原理

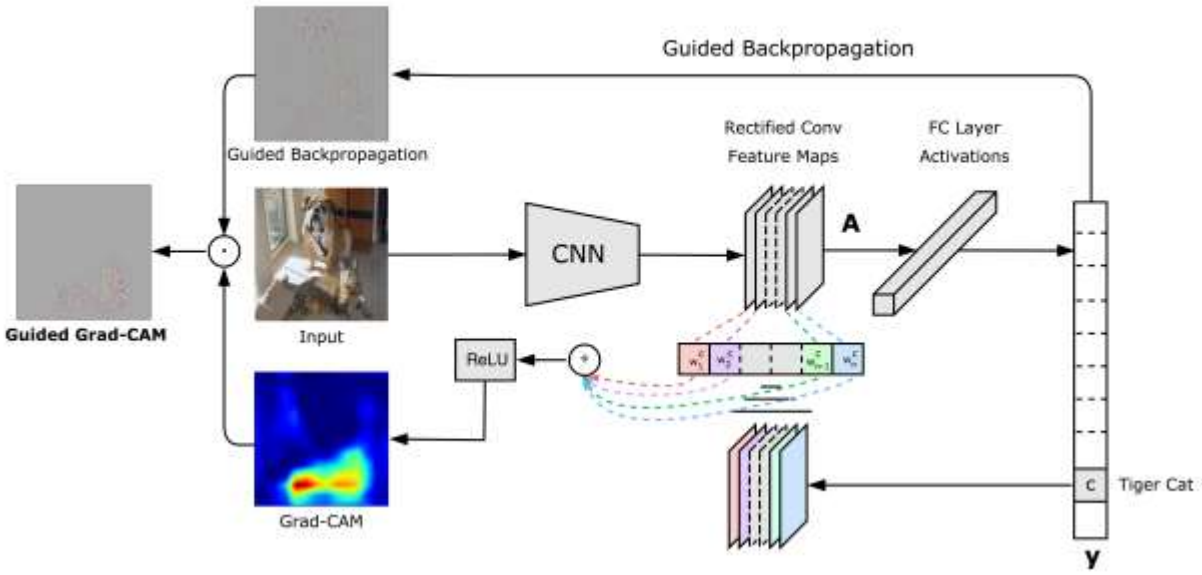
類別活化映射(Class Activation Mapping，CAM)概念，最後一層的卷積層生成每張特徵圖經過 GAP 變成一個像素，此像素包含整個特徵圖訊息，最後將一維的像素陣列乘以權重(w)，經過 softmax 知道「某類別」的值最大，則判定該分類為「某類別」。

經過 GAP 後的像素陣列會乘以權重 w，權重 w 的值越大代表該像素所代表的影像影響越大。知道 w 是指每張特徵圖可以被分為「某類別」的重要程度，把整張特徵圖的像素點皆乘以權重 w 後疊加，便能夠依據每張特徵圖的重要性關注不同的區域。CAM 可以視覺化神經網路的分類關注區域，但是如果要使用 CAM 的先決條件是必須使用 GAP 接在最後一層的卷積層後，因此在過去訓練時不是使用 GAP 的模型都必須修改模型結構，並重新訓練。

資料來源
https://medium.com/%E6%89%8B%E5%AF%AB%E7%AD%86%E8%A8%98/grad-cam-introduction-d0e48eb64adb

簡介 Grad-CAM 用途與成果展示

Grad-CAM 關鍵是能夠透過倒傳遞(Back Propagation)計算在 CAM 中使用的權重 w，求得神經元對於最後一層卷積層的特徵圖偏微分，也就是利用倒傳遞計算梯度，將其可視化繪製熱度圖(Heat map)，運作流程如下圖所示：



Selvaraju, R. R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., & Batra, D. (2016).

```
#實現 Grad-CAM
import os
import cv2
from keras import backend as K
from keras.preprocessing import image
from keras.applications import imagenet_utils
from keras.applications.inception_v3 import InceptionV3, preprocess_input
from matplotlib import pyplot as plt
import numpy as np
# 影像處理
def processing_image(img_path):
    # 讀取影像為 PIL 影像
    img = image.load_img(img_path, target_size=(224, 224)) #尺寸修改
    # 轉換 PIL 影像為 nparray
    x = image.img_to_array(img)
    # 加上一個 batch size，例如轉換 (224, 224, 3) 為（1, 224, 224, 3)
    x = np.expand_dims(x, axis=0)
    # 將 RBG 轉換為 BGR，並解減去各通道平均
    x = preprocess_input(x)
    return x
```

```
#計算卷積層的權重
def gradcam(model, x):
    # 取得影像的分類類別
    x = img = processing_image(img_path)
    preds = model.predict(x)
    pred_class = np.argmax(preds[0])
    print(pred_class)
    # 取得影像分類名稱
    pred_class_name = imagenet_utils.decode_predictions(preds)[0][0][1]
    #pred_class_name = imagenet_utils.decode_predictions(preds)
    # 預測分類的輸出向量
    pred_output = model.output[:, pred_class]
    # 最後一層 convolution layer 輸出的 feature map
    #預設是 ResNet 的最後一層 convolution layer
    last_conv_layer = model.get_layer('mixed10') #這邊更改為 Inception_v3 model 的最後一層名稱
    # 求得分類的神經元對於最後一層 convolution layer 的梯度
    grads = K.gradients(pred_output, last_conv_layer.output)[0]

    # 求得針對每個 feature map 的梯度加總
    pooled_grads = K.sum(grads, axis=(0, 1, 2))

    # K.function() 讓我們可以藉由輸入影像至 `model.input` 得到 `pooled_grads` 與
    # `last_conv_layer[0]` 的輸出值，像似在 Tensorflow 中定義計算圖後使用 feed_dict
    # 的方式。
    iterate = K.function([model.input], [pooled_grads, last_conv_layer.output[0]])

    # 傳入影像矩陣 x，並得到分類對 feature map 的梯度與最後一層 convolution layer 的
    # feature map
    pooled_grads_value, conv_layer_output_value = iterate([x])

    # 將 feature map 乘以權重，等於該 feature map 中的某些區域對於該分類的重要性
    for i in range(pooled_grads_value.shape[0]):
        conv_layer_output_value[:, :, i] *= (pooled_grads_value[i])

    # 計算 feature map 的 channel-wise 加總
    heatmap = np.sum(conv_layer_output_value, axis=-1)
    return heatmap, pred_class_name

#繪製熱力圖
def plot_heatmap(heatmap, img_path, pred_class_name):
    # ReLU
    heatmap = np.maximum(heatmap, 0)
    # 正規化
    heatmap /= np.max(heatmap)
    # 讀取影像
    img = cv2.imread(img_path)
    fig, ax = plt.subplots()
    im = cv2.resize(cv2.cvtColor(cv2.imread(img_path), cv2.COLOR_BGR2RGB), (img.shape[1],
img.shape[0]))
```

```
    # 拉伸 heatmap
    heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
    heatmap = np.uint8(255 * heatmap)
    # 以 0.6 透明度繪製原始影像
    ax.imshow(im, alpha=0.6)
    # 以 0.4 透明度繪製熱力圖
    ax.imshow(heatmap, cmap='jet', alpha=0.4)
    plt.title(pred_class_name)
    plt.show()
```

```
from keras import models
import tensorflow as tf
from keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.models import load_model

#載入自己訓練的模型
model = load_model('C:/Users\MCUT/Desktop/Inceptionv3.h5')
model.summary()

#model = InceptionV3(weights = 'imagenet') #使用 imagenet 權重
#model.summary()
#讀取圖片
img_path = '圖片路徑'
img = processing_image(img_path) #呼叫函數進行影像處理
heatmap, pred_class_name = gradcam(model, img) #呼叫函數計算權重
plot_heatmap(heatmap, img_path, pred_class_name) #呼叫函數將熱力圖呈現
```