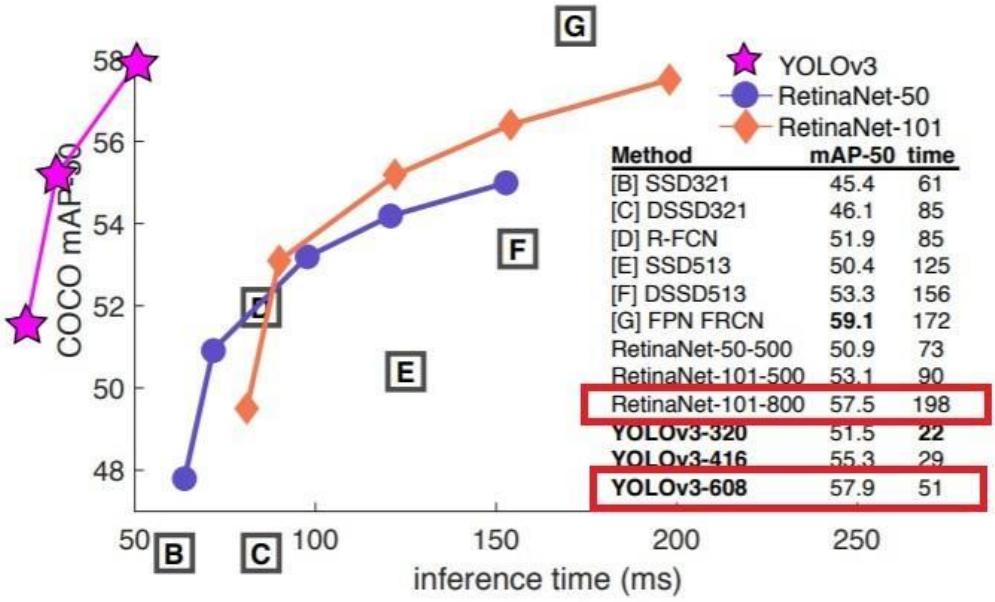
一、專題摘要

1. 專題主題：
辨識浣熊 (raccoon) 與袋鼠 (kangaroo) 的模型訓練與檢測
2. 基本目標：
使用訓練好的模型作為檢測袋鼠與浣熊的應用：
(1) 以下載的圖片當成訓練資料集 (training data) & 驗證資料集 (validation data) 來訓練模型，並透過loss、val_loss指標判斷模型是否收斂。
(2) 以新圖片當成測試資料集 (test data) 來檢測物體，並透過confidence_score指標判斷模型的好壞。
(3) 把影片中的袋鼠與浣熊的位置用預測框標示出來，並標示信心度 & 類別。
(4) 利用資料擴增 (Data augmentation) 方式，增加資料集的數量，觀察模型訓練後的結果。
本次實作採用 Keras-YOLO-V3的深度學習架構，網路backbone是採用Darknet-53，程式碼則是參考https://github.com/qqwweee/keras-yolo3，但因為此實作是要訓練辨識浣熊與袋鼠的模型，所以配合新的資料集把原來程式碼作了修改，才能完成實作程式碼。
研讀論文的實驗數據，論文中比較了YOLO-V3、SSD、RetinaNet等one stage架構的模型檢測速度 & 精確度，結論都是YOLO-V3技勝一籌，所以本實作採用YOLO-V3模型應用在深度學習上，作為解決物體檢測問題，具有很好的優勢。
擷取論文比較數據如下：
It achieves 57.9 AP50 in 51 ms on a Titan X, compared to 57.5 AP50 in 198 ms by RetinaNet, similar performance but 3.8× faster. (如圖)



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

二、實作方法介紹 (介紹使用的程式碼、模組，並附上實作過程與結果的截圖，需圖文並茂。)

1. 使用的程式碼介紹與模組介紹

本實作是使用 colab 平台進行深度學習與物體檢測，程序步驟說明如下：

(1). 先下載keras-yolo3程式碼、yolo_weights.h5 模型權重，這部分可參考 Day41 的程式碼，這裡不多做說明。

(2). 再下載 raccoon_dataset 資料集、kangaroo 資料集 (指令如下:)

```
!gitclonehttps://github.com/experiencor/raccoon_dataset.git
!gitclonehttps://github.com/experiencor/kangaroo.git
```

(3). 因本實作是要檢測 Raccoon、Kangaroo 這二類物體，所以必須把 annotation 轉換到訓練時需要的資料形態，而不能直接引用原來VOC的轉換方式，可參考這個程式碼來做修改 (https://github.com/qqwweee/keras-yolo3/blob/master/voc_annotation.py)，改寫後的部分程式碼說明如下：

** 注意要轉換的資料格式為：圖片儲存路徑 / 標註座標 / 類別代碼

舉例: ./kangaroo/images/00153.jpg 1,442,527,789,1

這裡的類別設定 0 代表是 Raccoon，1 代表是 Kangaroo

標註檔:放在 ./annotation_xml 目錄

```
!cp raccoon_dataset/annotations/*.xml ./annotation_xml
!cp kangaroo/annots/*.xml ./annotation_xml
```

標註後的.xml轉換成訓練資料格式

```
def convert_annotation(image_id, list_file):
    in_file = open('annotation_xml/%s.xml'%(image_id))
    tree=ET.parse(in_file)
    root = tree.getroot()
    for obj in root.iter('object'):
        difficult = obj.find('difficult').text
        cls = obj.find('name').text
        if cls not in classes or int(difficult)==1:
            continue
        cls_id = classes.index(cls)  # class index
```

訓練、驗證檔放在 : ./train、./val

```
train_txt = open('train/train.txt', 'w')
print("save train index at train/train.txt")
for train_id in train_index :
    train_txt.write('%s' %(train_id))
    train_txt.write('\n')
train_txt.close()

val_txt = open('val/val.txt', 'w')
print("save val index at val/val.txt")
for val_id in val_index :
    val_txt.write('%s' %(val_id))
    val_txt.write('\n')
val_txt.close()
```

```
執行轉換資料格式結果:
放在 train_labels.txt、val_labels.txt
sets=['train', 'val']
for image_set in sets:
    image_ids = open('%s/%s.txt'%(image_set, image_set)).read().strip().split()
    annotation_path = '%s_labels.txt'%(image_set)
    list_file = open(annotation_path, 'w')
    print("save annotation at %s" % annotation_path)
    for image_id in image_ids:
      if 'raccoon' in image_id :
        list_file.write('./raccoon_dataset/images/%s.jpg' %(image_id))
      else :
        list_file.write('./kangaroo/images/%s.jpg' %(image_id))
      convert_annotation(image_id, list_file)
      list_file.write('\n')
    list_file.close()
```

(4). 為了比較資料集的大小對物體檢測結果的影響，這裡增加了資料集擴增的實作 :

- 參考Day-16 ImageDataGenerator() 模組來產生擴增資料集。

- 並使用 LabelImg 工具，在擴增資料圖上畫出 Ground Truth 實際框，產出 .xml 標註檔，用來增加資料集數量，程式碼如下 :

```
img_gen = ImageDataGenerator(rotation_range=20,
                             width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2, zoom_range=0.2,
                             horizontal_flip=True, dtype=np.float32)

train_set = img_gen.flow_from_directory(from_path, batch_size=16, shuffle=False, save_to_dir=augment_path,
                save_prefix='augment', save_format='jpg', target_size=(416, 416), class_mode='categorical')
```

** 資料集擴增前的 jpg & .xml 檔案 : raccoon 數量為 200，kangaroo數量為 164，共有 364 個資料集。

```
raccoon jpg 檔數量: 200
kangaroo jpg 檔數量: 164
所有 jpg 檔數量: 364
save train index at ./train/train.txt
save val index at val/val.txt
save class at class.txt
save annotation at train_labels.txt
save annotation at val_labels.txt
```

** 資料集擴增後的 jpg & .xml 檔案 (增加一倍) : raccoon 數量擴增為 400, kangaroo數量為 328, 共有 728 個資料集。

```
raccoon jpg 檔數量: 400
kangaroo jpg 檔數量: 328
所有 jpg 檔數量: 728
save train index at train/train.txt
save val index at val/val.txt
save class at class.txt
save annotation at train_labels.txt
save annotation at val_labels.txt
```

** 實際訓練時, 是把資料集分成訓練集資料 (佔80%)、驗證集資料 (佔20%) 來進行訓練。

(5). 訓練模型並儲存權重檔案 : 這裡的程式碼重點, 主要是把剛才已做好前處理的轉換資料 (如下), 拿去建立 yolo 模型再開始訓練模型, 其他的訓練程式碼可參照 Day41 的程式碼部分來進行 :

```
轉換好的資料格式檔案餵給模型做訓練
annotation_path_train = 'train_labels.txt' # 轉換好格式的 train 標註檔案
annotation_path_val = 'val_labels.txt' # 轉換好格式的 val 標註檔案
classes_path = 'class.txt'
log_dir = 'logs/000/' # 訓練好的模型儲存的路徑


model = create_model(input_shape, anchors, num_classes,
    freeze_body=2, weights_path='model_data/yolo_weights.h5')
```

(6). 物體檢測 : 最後使用 YOLO(model_path, classes_path) 模組來進行物體檢測, 可以在此模組輸入不同參數 (score, iou) 來觀察檢測結果的變化。程式碼 (如下) :

```
# 使用預訓練的權重來偵測物體
from yolo import YOLO
classes_path = 'class.txt'
log_dir = 'logs/000/' # 訓練好的模型儲存的路徑
yolo_model = YOLO(model_path=log_dir + 'trained_weights_final.h5', classes_path=classes_path)

image_list1, image_list2 = [], []
for img in range(6) :
  image_list1.append('test/kangaroo_000' + str(img) +'.jpg')
for img in range(6) :
  image_list2.append('test/raccoon_000' + str(img) +'.jpg')
image_list2.extend(image_list1)

from PIL import Image
import matplotlib.pyplot as plt
for idx, im in enumerate(image_list2) :
    r_image = yolo_model.detect_image(Image.open(im))
    display(r_image)
```

(7). 影片檢測 : 使用 detect_video() 模組來檢測 .mp4 影片檔，把剛才建立的 yolo_model，餵給 detect_video() 模組進行影片檢測並存檔。

** 要對 detect_video() 進行部分程式碼修改 (如下)，才能正常運作 :

** 提醒程式碼要加入 cv2.VideoWriter_fourcc(*'MP4V') 編碼方式使用 MP4 格式，最後統計出 [總共有多少 frames] 與 [每個 frame 檢測時間]，並計算出 [影片的平均 fps]。程式碼 (如下) :

偵測影片模組:

```
def detect_video(yolo, video_path, output_path=""):
    vid = cv2.VideoCapture(video_path)
    if not vid.isOpened():
        raise IOError("Couldn't open webcam or video")
    video_FourCC    = int(vid.get(cv2.CAP_PROP_FOURCC))
    video_FourCC = cv2.VideoWriter_fourcc(*'MP4V')

    video_fps = vid.get(cv2.CAP_PROP_FPS)
    video_size = (int(vid.get(cv2.CAP_PROP_FRAME_WIDTH)),
                  int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT)))
    isOutput = True if output_path != "" else False
    if isOutput:
        out = cv2.VideoWriter(output_path, video_FourCC, video_fps, video_size)
```

總計多少 frames，每個 frame 檢測時間

```
video_cnt = 0
video_playtime = []
while True:
    return_value, frame = vid.read()
    video_cnt += 1
    if return_value == True :
        image = Image.fromarray(frame)
        start_time = time.time()
        image = yolo.detect_image(image)
        end_time = time.time()
        time_img = end_time - start_time
        video_playtime.append(round(time_img, 3))
        result = np.asarray(image)
        cv2.putText(result, text=fps, org=(3, 15), fontFace=cv2.FONT_HERSHEY_SIMPLEX,
                    fontScale=0.50, color=(255, 0, 0), thickness=2)
        if isOutput:
            out.write(result)
     else :
        break
vid.release()  # release vid resource
out.release()  # release out resource
return video_playtime, video_cnt
```

計算影片的平均 fps ：

```
# 1.偵測 Kangaroo.mp4
video_playtime, video_cnt = detect_video(yolo_model, video_path="Kangaroo.mp4",
                                         output_path="Kangaroo_new_1.mp4")
print('Kangaroo.mp4 total frames: ', video_cnt)  # 總共有多少 frames

# Kangaroo 平均 fps :
avg_fps = 1/np.mean(video_playtime)
print("Kangaroo.mp4 avg fps: %.3f" % avg_fps)
```

三、成果展示 (介紹成果的特點為何，並撰寫心得。)

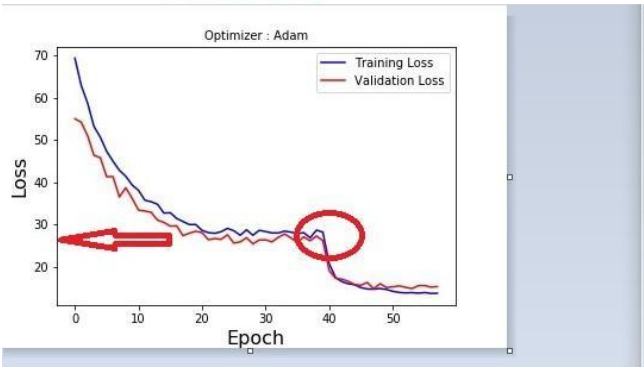
1. 訓練模型時的 loss 損失比較 :

訓練模型時，可以觀察一下有資料擴增的 loss 權重差異，從 loss 記錄資料來看，可以看出有資料擴增的 loss 下降趨勢比較快一些，這會讓後面要做的物體檢測精確度提高一些，可以從實際的測試圖片檢測結果得到驗證 (如下)

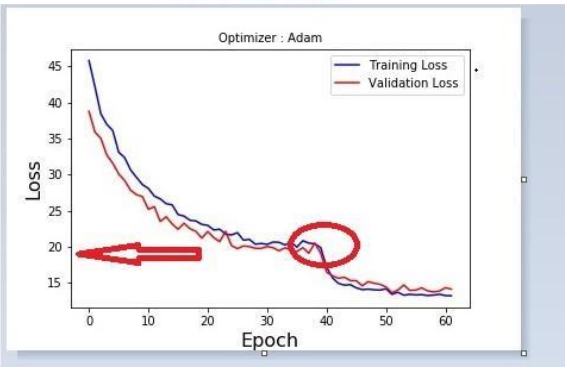
。

** 在 epoch=40 時，圖像擴增前的 loss 權重只下降到 25 左右，擴增後的 loss 權重已下降到 20 左右。

圖像擴增前：



圖像擴增後：



** 訓練停止的 loss 值：圖像擴增後的 loss 也是比圖像擴增前的 loss 下降的多一些 (如下)。

圖像擴增前：

```
Epoch 65/100
18/18 [==============================] - 28s 2s/step - loss: 13.8613 - val_loss: 15.6171
Epoch 66/100
18/18 [==============================] - 28s 2s/step - loss: 14.0042 - val_loss: 15.6609
Epoch 67/100
18/18 [==============================] - 28s 2s/step - loss: 13.8151 - val_loss: 15.2986

Epoch 00067: ReduceLROnPlateau reducing learning rate to 9.999999974752428e-08.
Epoch 68/100
18/18 [==============================] - 28s 2s/step - loss: 13.8430 - val_loss: 15.4145
Epoch 00068: early stopping
```

圖像擴增後：

```
Epoch 69/100
36/36 [==============================] - 54s 2s/step - loss: 13.3028 - val_loss: 13.7366
Epoch 70/100
36/36 [==============================] - 54s 2s/step - loss: 13.4133 - val_loss: 13.8623
Epoch 71/100
36/36 [==============================] - 55s 2s/step - loss: 13.2328 - val_loss: 14.3186

Epoch 00071: ReduceLROnPlateau reducing learning rate to 1.0000000116860975e-08.
Epoch 72/100
36/36 [==============================] - 55s 2s/step - loss: 13.2116 - val_loss: 14.1286
Epoch 00072: early stopping
```

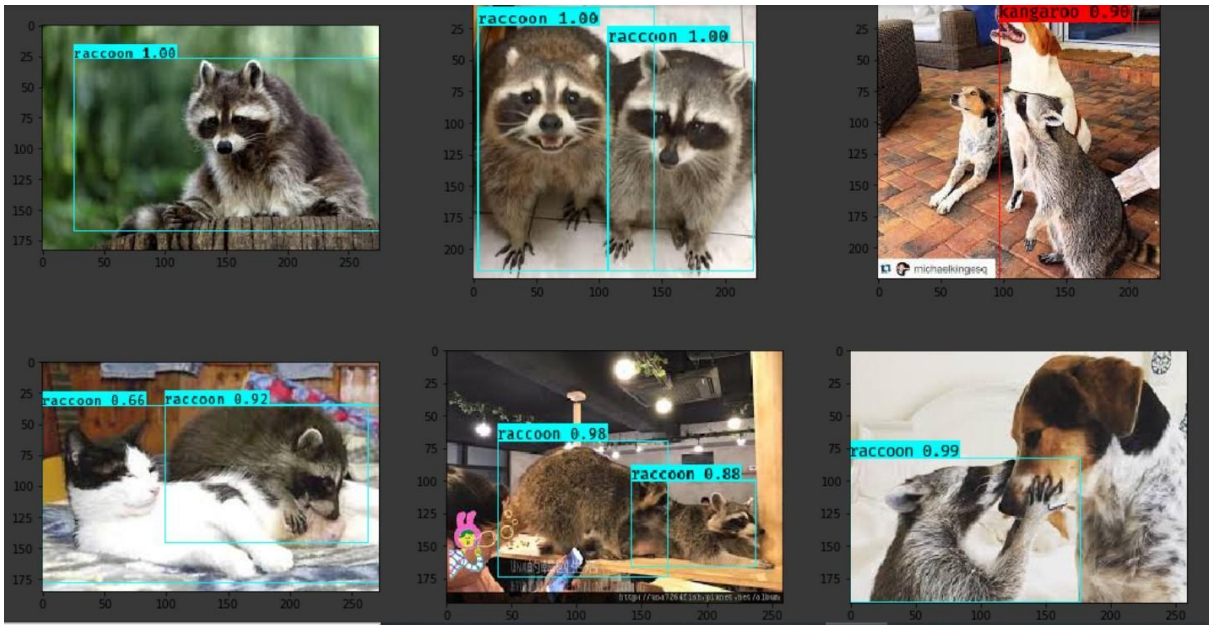
2. 圖片檢測：使用 12 張測試圖片來檢測物體

** 這裡的 12 張圖片是使用圖像擴增後的檢測結果，從檢測結果看起來，準確性已經有提高，但還是有部分檢測錯誤之處 (如下)：

(1). 其中第 9 張圖片把 raccoon 檢測成 kangaroo

(2). 其中第 10 張圖片把 貓 檢測成raccoon

- 後面會針對圖像擴增前後的比對再作說明。



3. 使用Data augmentation 圖像擴增技巧，來提高檢測的精確度：

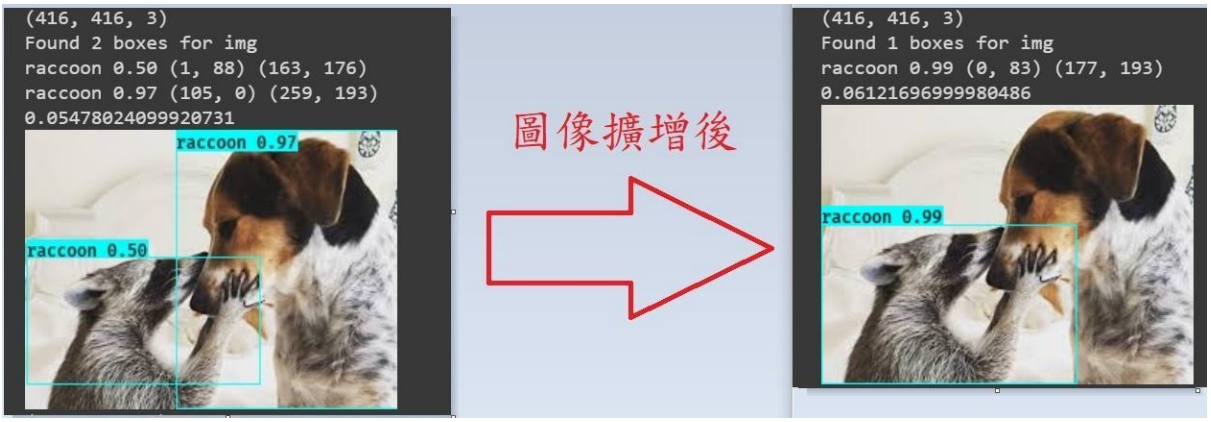
(1). 先用 ImageDataGenerator() 的程式碼產生擴增圖片檔。

(2). 使用 LabelImg 套件工具，在擴增資料圖上畫出 Ground Truth 實際框，產出 .xml 標註檔，增加資料集數量。

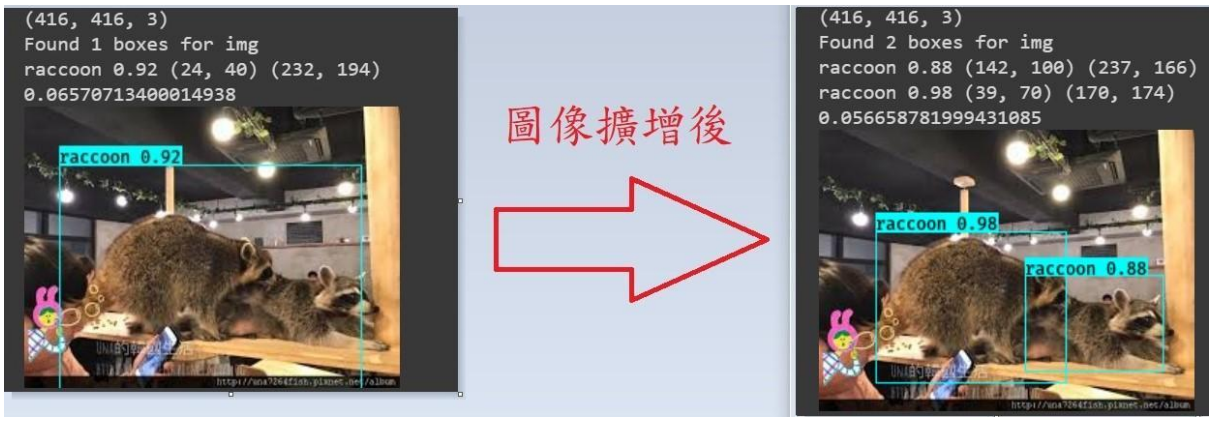
(3). 增加擴增圖片檔後，可以觀察到檢測的測試圖片中，[預測框的信心度 score] 與 [物體檢測的準確度]，都提高了，從這裡可以看出增加資料集數量，對於模型的檢測效果幫助很大。

- 以下紀錄圖像擴增前後的效果比對：

** 擴增前:狗被檢測為raccoon; 擴增後:檢測正確只有一隻raccoon



** 擴增前:檢測只有一隻raccoon; 擴增後:檢測正確為二隻raccoon



** 擴增前:狗被檢測為kangaroo; 擴增後:檢測正確只有一隻kangaroo

4. 影片檢測：

(1). GPU：使用的硬體是 colab 的 Tesla T4。

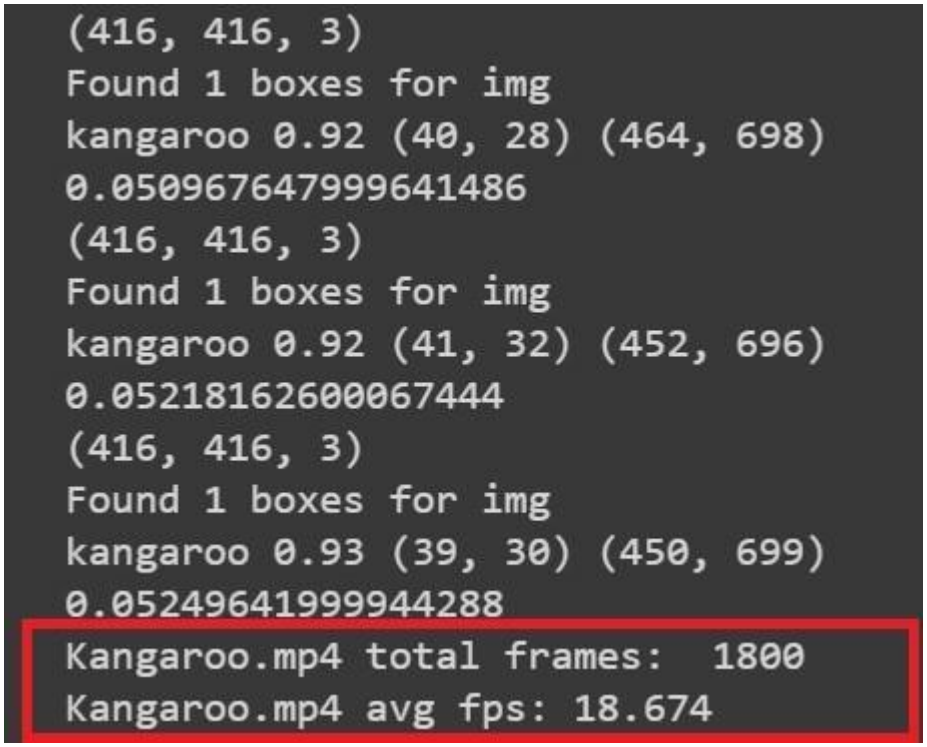
(2). 調整 YOLO() 的 [信心度score threshold] 與 [iou threshold] 的結果比較：

A. 當score_threshold設高一些時，預測框產生的數量會減少，相對會加速 video (.MP4) 檢測速度，即 fps 上升。

B. 這裡的 iou_threshold 指的是 NMS (Non-Maximum Suppression)，預測框的 IOU 重疊率。當iou_threshold設低一些時，預測框保存下來的會減少，相對會加速 video (.MP4) 檢測速度，即 fps 上升。

** 不過這些 threshold 的調整，不能隨意調整，以免提升了檢測速度卻漏失掉了精確度，最後還是要看你實際上的運用，去選擇最適合你的設定值。

** kangaroo 實作統計出來的影片平均速度為 18.6 fps (如下)，速度雖不是很快，還沒達到可以稱為 real time 程度，但影片的檢測流暢度看起來已經算是平順了，只是檢測精確度偶爾會發生錯誤，這是 YOLO3 模型還要再改進的的地方。不過本次實作的資料集只有幾百張，數量上還不足以讓檢測的精確度再提高。(本次實作還沒加入 mAP 的統計數據，後續可再用 mAP 來看精確度的比較數據)



四、結論 (總結本次專題的問題與結果)

1. 資料集的數量要夠多，訓練出來的模型精確度會比較好一些，如果資料集真的不易取得，還是可以利用資料增強 (data augmentation) 的技巧產生擴增資料集，來訓練出精確度較高的模型。
2. 利用Yolo3 多尺度的特性來檢測物體，從實作的結果來看，我認為檢測速度與精確度都已達到一定的水準了。但如果再提高模型的檢測效率，那就必須再對模型瘦身，即是要增加運算能力與降低參數的數量，例如輕量化模型 MobileNet，可延伸用來提高檢測速度 fps。
3. 將來可再運用 ResNet、Inception 的 CNN 網路架構，試著嘗試改良 Yolo3 模型。

** 影片檢測連結：
Yolo3 深度學習袋鼠檢測影片：https://youtu.be/jQS59957OvM
Yolo3 深度學習浣熊檢測影片：https://youtu.be/kFskOJjXZBg