Yolov7環境建置與應用

標題

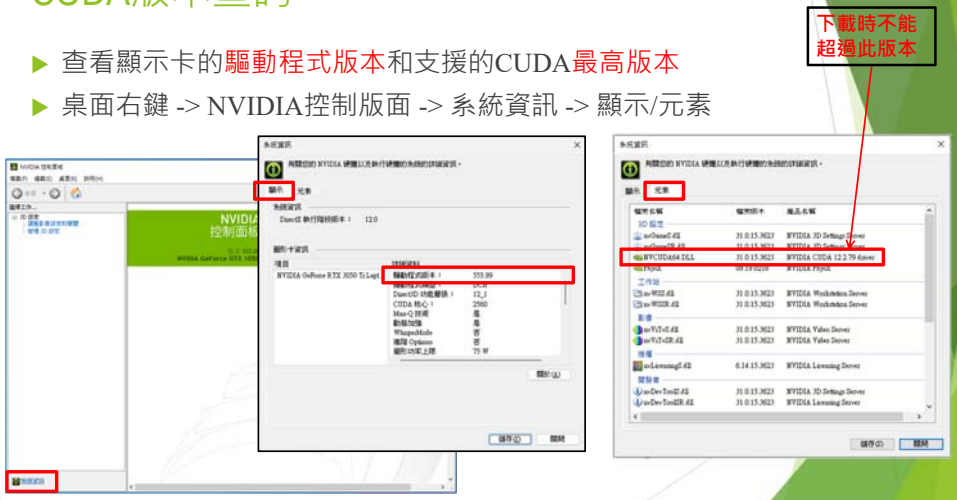
- Yolov7環境建置
- 訓練與辨識
- FFmpeg環境建置
- Yolov7與FFmpeg整合

Anaconda

- 進入Anaconda網頁下載安裝程式
- 執行程式並完成安裝
- 開啟Anaconda Navigator確認安裝是否順利完成

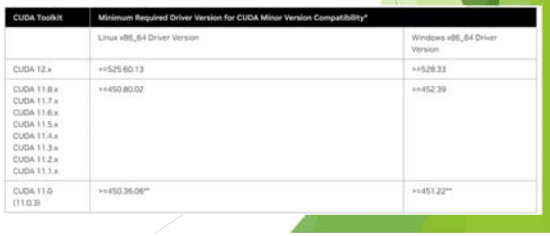
CUDA版本查詢

- 查看顯示卡的驅動程式版本和支援的CUDA最高版本
- 桌面右鍵 -> NVIDIA控制版面 -> 系統資訊 -> 顯示/元素



CUDA

- 根據前面的步驟並根據官方說明手冊找到符合自己版本的CUDA編號
- 若顯示卡驅動程式與CUDA版本不符，則需要更新顯示卡的驅動程式
- 在CUDA下載網頁上找到符合自己版本的程式進行下載
- 使用建議安裝步驟安裝即可



CUDA Toolkit	Minimum Required Driver Version for CUDA Minor Version Compatibility*	
	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 12.x	>=525.60.13	>=528.33
CUDA 11.8.x CUDA 11.7.x CUDA 11.6.x CUDA 11.5.x CUDA 11.4.x CUDA 11.3.x CUDA 11.2.x CUDA 11.1.x	>=450.80.02	>=452.39
CUDA 11.0 (11.0.3)	>=450.36.06**	>=451.22**

cuDNN

- 根據前面下載的CUDA版本下載匹配的cuDNN
- 安裝說明
- cuDNN: https://developer.nvidia.com/rdp/cudnn-archive
- 解壓縮後把cuDNN的三個資料夾放到CUDA資料夾

(CUDA的預設安裝路徑為：

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA)

Yolov7

- 在任意位置創建一個空的資料夾
- 開啟Anaconda Prompt，並透過指令進入該資料夾內(cd)
- 下載Yolov7：

git clone https://github.com/WongKinYiu/yolov7

- 創建一個虛擬環境(-name後可以改為自己想取的名字)：

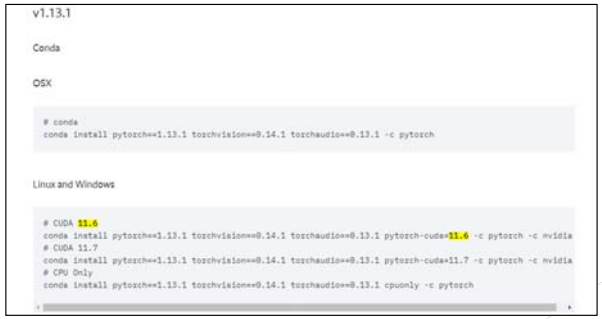
conda create -name yolov7 python=3.9 && activate yolov7

- 輸入以下指令並等待完成Yolov7的建置：

pip install -r requirements.txt

Pytorch(若成功建置Yolov7環境則跳過)

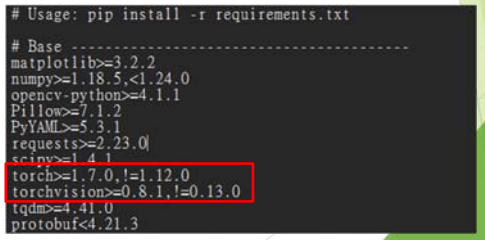
- 若完成前面的安裝步驟發現Pytorch的版本不符，則需要到Pytorch的下載網頁上搜尋符合自己CUDA版本的Pytorch，找到後將指令貼到Anaconda Prompt上進行安裝



Pytorch

- 在Yolov7的資料夾內找到requirements.txt並開啟
- 因為我們已經根據自己電腦的需求安裝了Pytorch了，因此在torch與torchvision兩行前加上「 # 」，防止安裝時安裝了不正確的Pytorch版本，完成後記得儲存
- 再次回到Anaconda Prompt輸入以下指令並等待完成Yolov7的建置：

pip install -r requirements.txt



測試

- 在Yolov7的Github網頁上下載一個預訓練好的權重檔進行測試(在Testing的地方可以找到)，將下載好的權重檔放在Yolov7的資料夾內
- 辨識指令：

python detect.py --weight yolov7.pt --source inference/images/bus.jpg

- 完成後可以在 runs/detect 資料夾內找到辨識好的圖片

Yolov7訓練

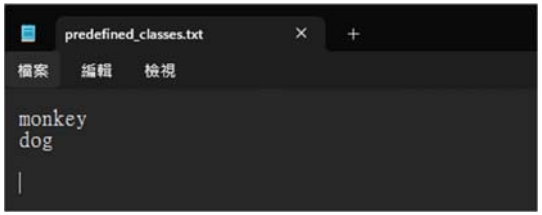
- 在開始訓練之前我們必須準備以下的程式與資料：
- LableImg
- 訓練時設定檔
- 大量的訓練圖片，並會將圖片區分為訓練集、驗證集、測試集

LabelImg下載

- Windows v1.8.0
- 此軟體用於建立預辨識的標籤與標籤位置

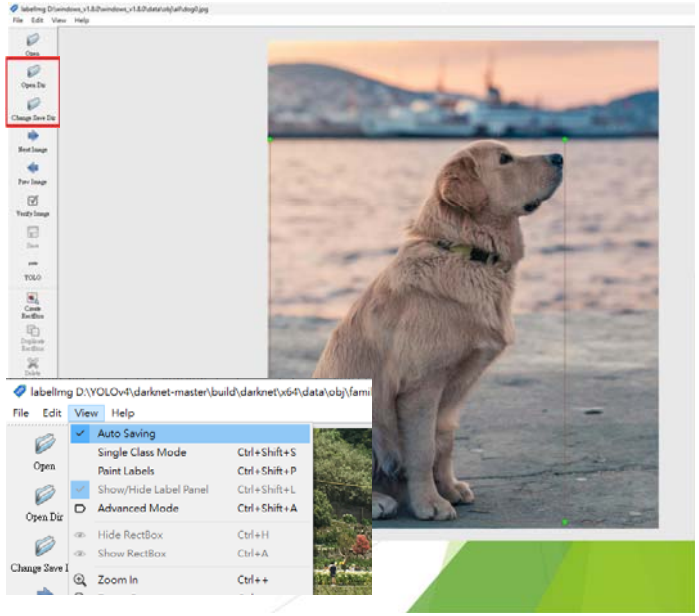
LabelImg

- 打開下載的windows_v1.8.0資料夾
- 先點選data資料夾
- 打開predefined_classes.txt
- 裡面是預定義的標籤集，先將裡面的內容為要標籤的物件種類



LabelImg

- 開啟LabelImag.exe
- 使用Open Dir開啟預訓練的圖片位置
- 在Change Save Dir執行與Open Dir一樣的動作
- 按照右圖啟動自動儲存功能

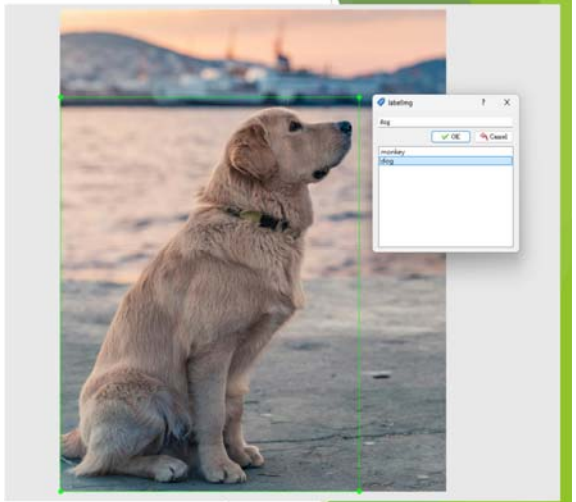


LabelImg

- 在左圖按一下PascalVOC改成右圖YOLO以符合YOLO的標籤格式

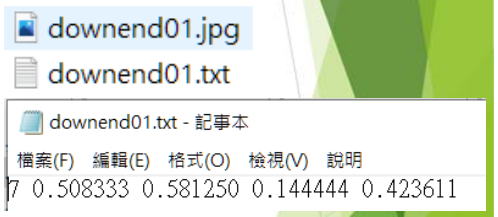
LabelImg

- 按下Create RectBox
- 框選預辨識的位置，會跳出預定義的Label清單，也就是剛剛我們輸入至predefined_classes.txt的標籤



LabelImg

- 將所有圖片都框選並標註標籤
- 這時每一張圖片會產生1個txt，裡面會儲存圖片標籤的訊息
- 以右圖為例，7為第八個標籤(0到7)，後面為框選方框的四個角位置
- 一張照片可以有多個標籤



訓練設定檔準備

- 在data資料夾內創建一個.yaml檔案(可從已有的檔案中複製)，開啟後修改以下內容：

(1) 訓練資料路徑(train, valid, test)

(2) 類別數量

(3) 類別名稱

```
train: ./data/monkey_train.txt
val: ./ data /monkey_valid.txt
test: ./coco/ monkey_test.txt

# number of classes
nc: 1

# class names
names: [ 'monkey' ]
```

- 在cfg/training資料夾內同樣創建一個.yaml檔案(可從已有的檔案中複製)，開啟後修改Class數量

```
# parameters
nc: 1  # number of classes
```

訓練圖片資料集

- 訓練資料集分成三個部分，分別為訓練集(training set)、驗證集(validation set)和測試集(test set)
- 常使用的分配比率為

  70% train, 15% val, 15% test

  80% train, 10% val, 10% test

  60% train, 20% val, 20% test
- 沒有一個絕對的比率，需要透過測試得到精度較高的比率

開始訓練

- 透過以下指令開始進行訓練：

python train.py --device 0 --batch-size 8 --data data/mydata.yaml --img 640 --cfg cfg/training/yolov7_custom.yaml --weights yolov7.pt

綠色字須根據自己檔案位置進行修改，指令的內容可開啟train.py來查看訓練的指令

指令內主要會增加幾個項目：

device：使用第幾個編號的GPU來訓練，0表示使用一個GPU進行訓練

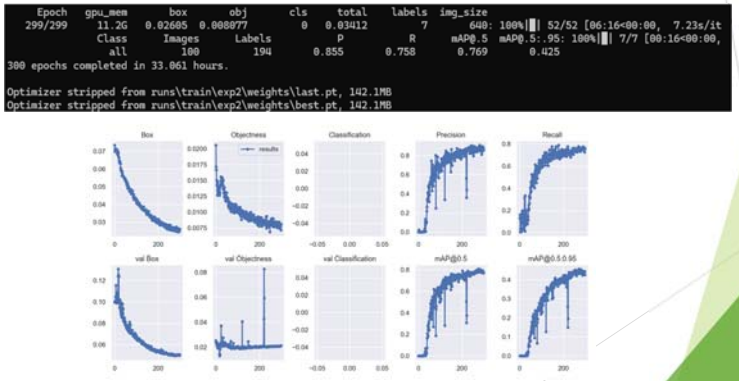
batch：每次batch學習採用多少的樣本資料

img：將圖片壓縮成指定的大小後進行訓練

也可以輸入指令python train.py help來新增其他設定來調整訓練

訓練完成

- 訓練完成後我們就可以在runs/train資料夾內找到我們訓練好的權重檔與訓練時的一些數據



影像辨識

- 完成訓練後就可以透過以下指令來測試訓練結果：

python detect.py --weights best.pt --conf 0.25 --img-size 640 --source monkey_test.jpg --view-img

(1) --weights：指定使用的權重檔

(2) --conf：設定信心程度(confidence)，若辨識出的信心程度低於此設定的數值，則不會記錄該辨識出的結果

(3) --img_size：將辨識的圖片壓縮成指定大小

(4) --source：辨識圖片路徑

(5) --view-img：顯示辨識結果

產生辨識結果

- 在我們使用辨識完圖片後，Yolov7會自動將辨識好的圖片會預設存放在“ runs/detect ”資料夾內，並再自動產生一個" exp "的資料夾存放每次辨識好的圖片。
- 要將辨識出的結果儲存，我們只需要在指令中加上" --save-txt " 和 "--save-conf "即可
- E.g.： python detect.py --weights best.pt --conf 0.25 --save-txt --save-conf --source inference/images/image3.jpg --view-img

FFmpeg安裝

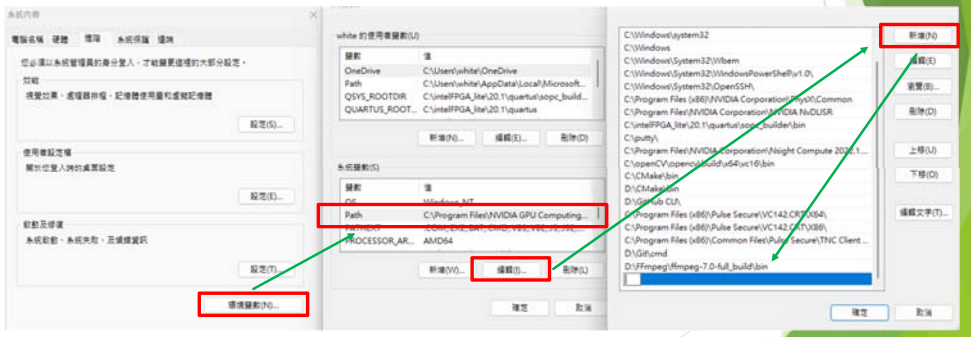
- 進入FFmpeg下載網站並在 release builds 內找到 ffmpeg-release-full.7z

設定環境變數

- 下載完成後解壓縮，並找到 “bin” 資料夾並複製路徑

設定環境變數

- 電腦環境變數內找到 “PATH” 變數，編輯並將剛才複製好的路徑新增進去



Anaconda 環境

- 由於之後執行整體環境時會統一使用Anaconda Prompt開啟，因此我們也要在Anaconda的環境上安裝FFmpeg
- 安裝指令：

conda install ffmpeg

FFmpeg 指令

- 設定完成後就可以開啟 CMD並輸入" ffmpeg "進行測試，若有顯示版本等資料表示安裝及設定完成
- 1. 列出可用設備

ffmpeg -hide_banner -list_devices true -f dshow -i dummy

CMD內會顯示可用的設備，將設備名稱後面有 " (video) " 的設備名稱紀錄或複製下來

- 2. 開啟攝影機

ffplay -f dshow -i video="Your Device Name"

FFmpeg 指令

- 3. 攝影鏡頭擷取

ffmpeg -f dshow -i video= "Your Device Name" capture1.mp4

指令開始前請先移動到(cd)想要的目錄內，擷取的檔案會直接儲存在該路徑下，"capture1.mp4" 為儲存的檔案名稱與副檔名，可依據需求自行更改

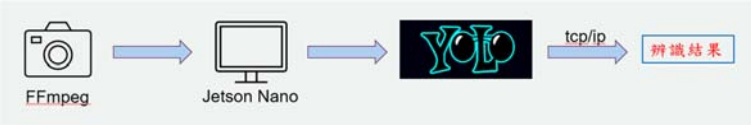
4.錄製與截圖(本次重點)

ffmpeg -f dshow -i video="Your Device Name" -vf fps=10 -update 1 image.png

此指令會開起攝影機後約每0.1秒(10fps)截取一次圖片，並儲存為 "image.png" -update 會將每次截取的圖片覆蓋到 "image.png"檔案中，因此我們就可以使用Yolov7不斷讀取 "image.png"並進行辨識

程式修改

- 由於我們需要不斷的去讀取攝影機所擷取的圖片，因此我們需要去修改detect.py裡的程式，使其能夠地不斷去做辨識
- 再來因為我們需要使用到辨識完成的結果，因此我們會在程式中加上TCP/IP的網路程式，將辨識的結果透過網路傳送到另一個我們寫好的Server端程式



detect.py

- 建議複製一份detect.py檔案，將其更改名字後透過文字編輯軟體開啟
- 在程式前面引入函示庫的部分加上下圖的程式，將socket的函示庫引入並定義TCP/IP連線中的IP位址與PORT編號和BUFFER大小
- detect.py 範例程式

```
from utils.plots import plot_one_box
from utils.torch_utils import select_device, load_classifier, time_s
import socket

HOST = '127.0.0.1'
PORT = 4100
BUFFERSIZE = 1024

def detect(save_img=False):
    source, weights, view_img, save_txt, imgsz, trace = opt.source,
```

detect.py

- 在70行找到for迴圈的程式，也就是進行辨識的程式，在程式前面加上建立socket與連線的程式
- 在for迴圈前加上" while 1 "將其改成無限迴圈，並將70~157行的程式進行縮排

```
t0 = time.time()

#create socket
sck = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sck.connect((HOST, PORT))
recv_data = sck.recv(BUFFERSIZE)

while 1:
    for path, img, im0s, vid_cap in dataset:
        img = torch.from_numpy(img).to(device)
        img = img.half() if half else img.float()  # uint8 to fp16/32
```

detect.py

- 找到第120行，這裡就是我們在指令中加上--save-conf與--save-txt會執行的程式，也就是儲存辨識結果，我們就可以在這裡透過網路將結果傳送出來，我們在這裡加上以下的程式

```
# Write results
for *xyxy, conf, cls in reversed(det):
    if save_txt:  # Write to file
        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
        line = (cls, *xywh, conf) if opt.save_conf else (cls, *xywh)  # label format
        result_out = 'Class : ' + names[int(cls)] + ' ' + ('%g ' * len(line)).rstrip() % line + '\n'

        if recv_data.decode() != None :
            sck.send(result_out.encode())
            recv_data = sck.recv(1024)
            print('recv: ' + recv_data.decode())
            #print(type(recv_data))
        with open(txt_path + '.txt', 'a') as f:
            f.write(('%g ' * len(line)).rstrip() % line + '\n')

    if save_img or view_img:  # Add bbox to image
        label = f'{names[int(cls)]} {conf:.2f}'
        plot_one_box(xyxy, im0, label=label, color=colors[int(cls)], line_thickness=1)
```

datasets.py

- 因為不斷讀取圖片時若該圖片還尚未完成，Yolov7會將辨識進行中斷，因此我們還需要再修改另一個程式來防止辨識中斷
- 開啟utils/datasets.py，找到第183行後並修改成以下程式(同樣建議複製一份原始檔進行保存)：
- datasets.py

```
    else:
        # Read image
        self.count += 1
        #img0 = cv2.imread(path)  # BGR
        while 1:
            img0 = cv2.imread(path)
            if img0 is None:
                continue
            else:
                break
        assert img0 is not None, 'Image Not Found ' + path
        #print(f'image {self.count}/{self.nf} {path}: ', end='')
```

Sever.py

- 再來我們就可以撰寫一個Server的程式來接收Yolov7辨識出的資料了
- Server範例程式

.bat

- 最後我們就可以撰寫一個.bat檔來同時執行所有的動作了，包含使用ffmpeg開啟鏡頭、開啟Server、使用Yolov7進行辨識，完成後就可以使用Anaconda Prompt開啟此.bat來執行所有流程了(務必使用Anaconda Prompt執行)
- .bat檔內容如下(Yolov7中的路徑與相機名稱需更改成自己的)
- @echo off
- del image.png
- start cmd.exe /k "conda activate && ffmpeg -f dshow -i video="Your device name" -vf fps=10 -update 1 image.png"
- timeout /t 3
- start cmd.exe /k "conda activate && python server.py"
- timeout /t 3
- start cmd.exe /k "conda activate && python detect_loop_tcp.py --weights yolov7.pt --conf 0.25 --save-txt --save-conf --source image.png --view-img"