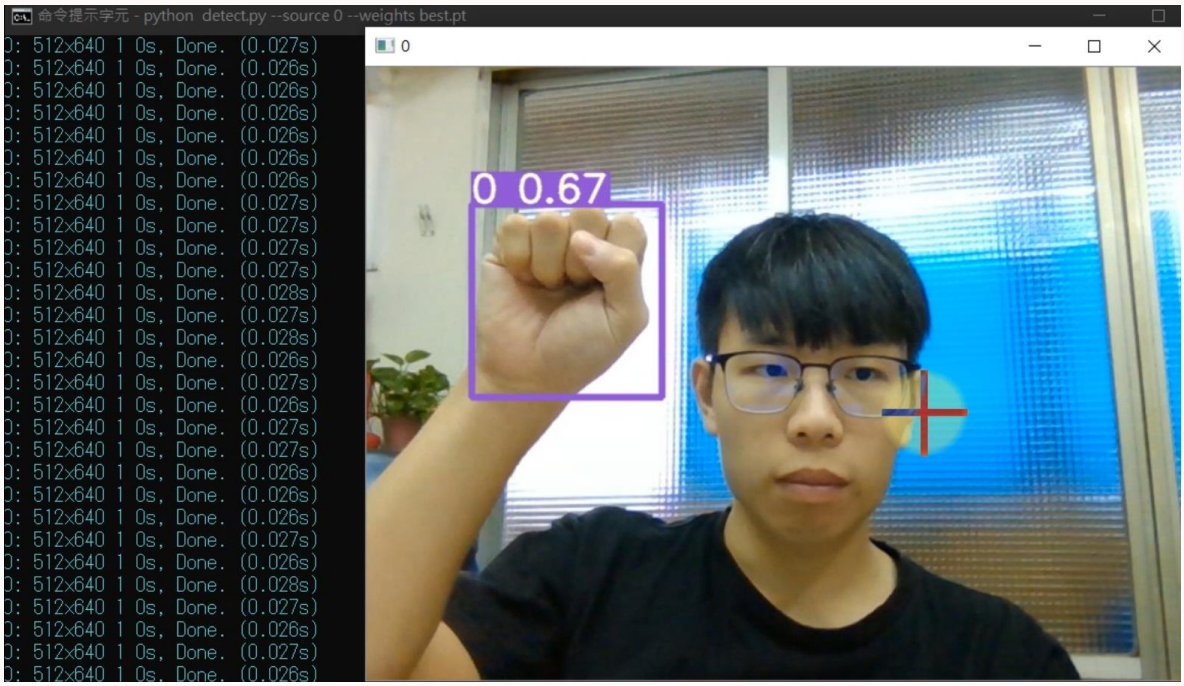
CAVEDU 教育團隊部落格

[技術教學文-格式]

標題	Jetson Nano 實作 YOLOv5 的轉移學習



分類		**標籤**	YOLOv5 Jetson Nano Transfer Learning
撰寫/攝影			
前情提要			
時間		**材料表**	
成本			

難度	中		
前言			
在上一篇 YOLOv5 的文章中我們已經介紹並且應用 YOLOv5 在圖像、影像甚至是即時影像串流中，但是這都是利用他們提供的權重也就是每次辨識出來的項目都是他們給的，假設你想要辨識出特定的物品該怎麼辦？這時候就要來做轉移學習了！			
內文			

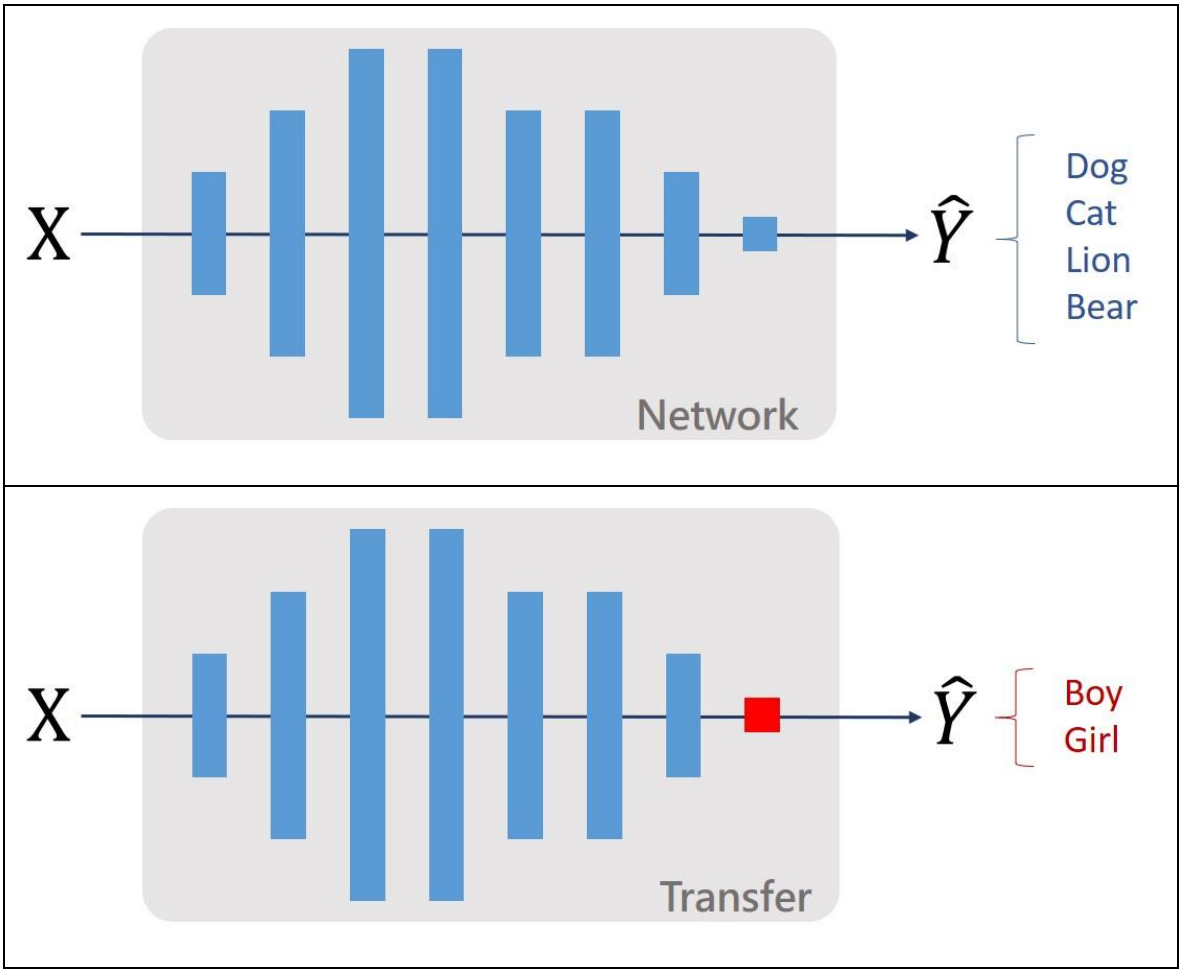
轉移學習（Transfer Learning）

人類會利用過去的經驗來幫助自己吸收新的知識技術，就像是大學念微積分的時候都會想到高中教的解決方法，而神經網路中的轉移學習（遷移式學習） 也是類似的概念。大家都知道神經網路的訓練是相當耗費時間的，一開始初始化權重，經過無數次的訓練才能有結果，那如果能夠將別人訓練好的權重拿來當初始化的權重，是否能加速訓練呢？大部分的情況下答案都是肯定的。

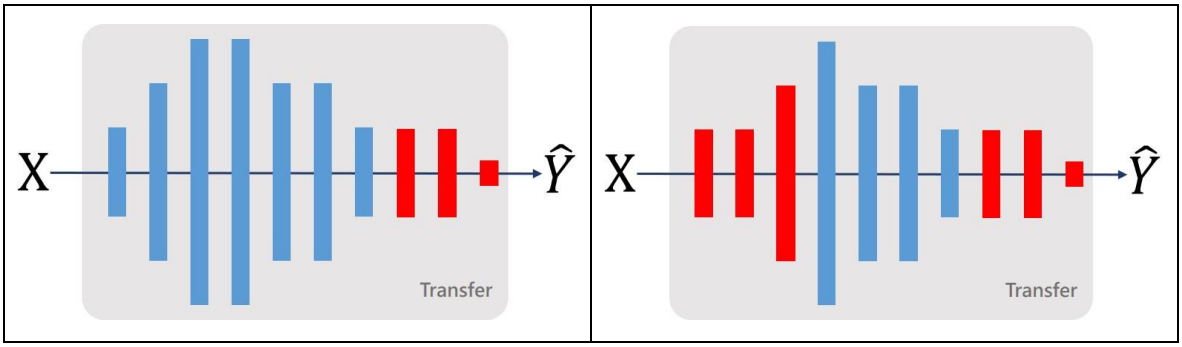
轉移學習簡單原理

以影像辨識來當例子，CNN 中的 kernel 就像是一個特徵萃取濾波器，可能是擅長萃取直線、圓形或者顏色等等的，一般訓練的方法都是隨機找幾個濾波器測試，好的就留下來不好的就再換成別的，這樣一步一步的摸索出來；而轉移學習則是別人已經告訴你可以用這些濾波器，我用過效果不錯！你在自己評估是否好用或是否要修改！所以一定是比一般訓練還要來的快、狠、準！

從下圖可以看到，假設你有一個神經網路用來分辨動物（狗、貓、獅子、熊），這時候你想利用他來辨識男孩、女孩，只需要將最後一層輸出層給換掉，給予隨機的初始化數值，讓神經網路只「微調」這一層即可，藉由先前神經網路辨識動物的方法來辨識男孩女孩。



當然，當你面臨的問題更複雜，你可以不只微調一層！在一般常看到的解釋中，對於語音辨識問題會微調後幾層；影像辨識問題則會微調前幾層。

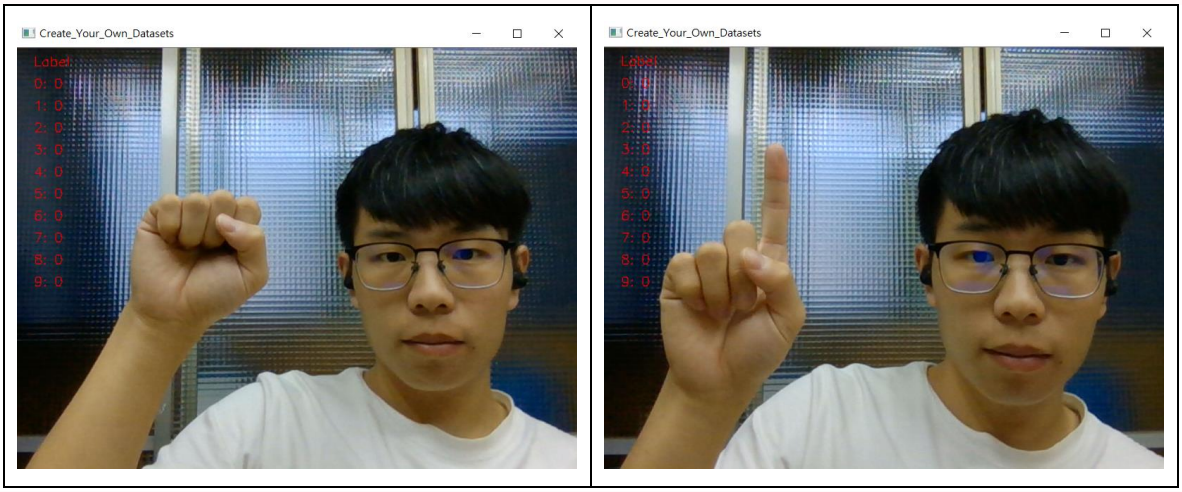


主要的意義在於神經網路前幾層大多用於特徵萃取，對於語音來說前幾層能保留對於聲音特徵的發音方式，後幾層才會聽到聲音的內容；而影像辨識如果要辨識的物品差異太大，就不能使用相同的特徵萃取方式，所以才會選擇微調前幾層。

在轉移學習的領域相當的有趣也很複雜，這邊我們帶到一點基礎即可，更深入的資訊可以在網路上找到很多～接下來我們就要針對 YOLOv5 進行轉移學習了。

Transfer Learning on YOLOv5

我想讓 YOLO 學會辨識我手指比的數字，先以辨識 0、1 為主。範例如下



建置環境

在上一篇 YOLOv5 中已經介紹如何使用 virtualenv 來建置虛擬環境，所以這邊簡單帶個流程，想要知道更詳細的可以去查看上一篇：

- 安裝 Virtualenv、Virtualenvwrapper、修改環境變數、開啟新環境
- 安裝 git tool、下載 YOLOv5 Github 專案
- 安裝套件 (PyTorch 額外裝、OpenCV 用 Link、Scipy 最後裝)

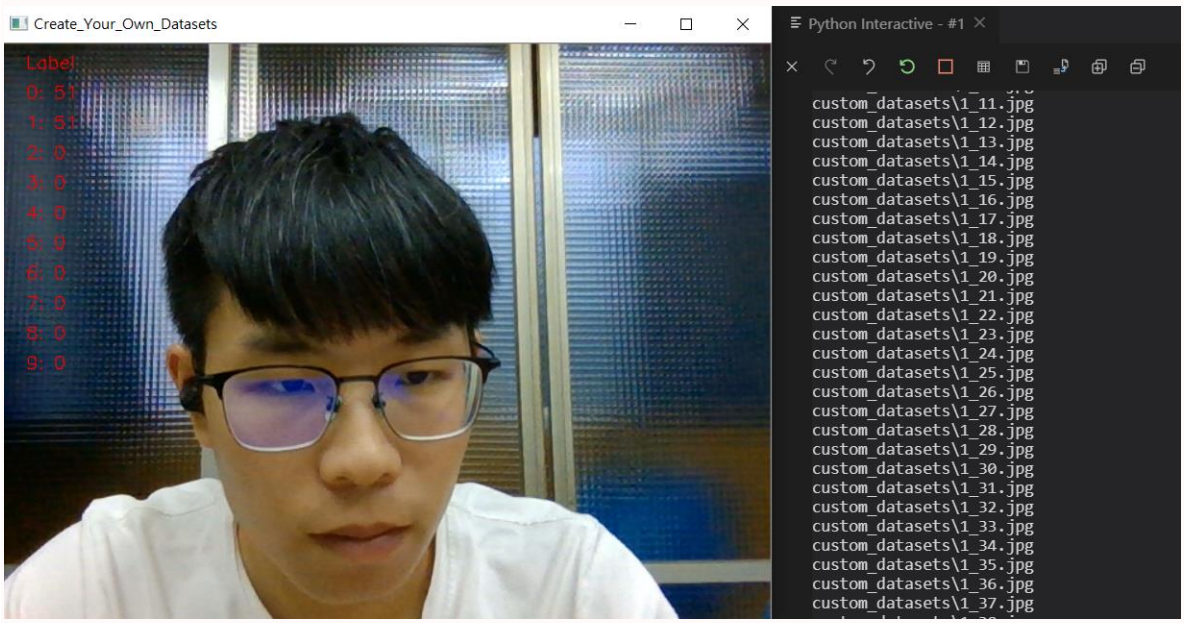
準備數據

如何準備數據呢？可以透過筆電的攝影機來拍照，簡單的拍照程式就是 OpenCV 開啟相機讀取按鍵，儲存照片，這邊就不多作介紹了，我額外寫了拍照數量，這樣我才知道數據是否平均。

```
import cv2
import os
import numpy as np

############## save data ##############
def save(trg_path, idx, frame):
    global label
    save_path = os.path.join(trg_path, rf'{idx}_{label[idx]}.jpg')
    print(save_path)
    cv2.imwrite(save_path, frame)
    label[idx] = label[idx]+1

############## set target path ##############
trg_path = 'custom_datasets'
if os.path.exists(trg_path)==False: os.mkdir(trg_path)

############## get camera & set size ##############
#w_size, h_size = 512, 512
cap = cv2.VideoCapture(0)
#cap.set(cv2.CAP_PROP_FRAME_WIDTH, w_size)
#cap.set(cv2.CAP_PROP_FRAME_HEIGHT, h_size)

############## set parameters ##############
label = np.zeros([10], dtype=int)
print(f'label len : {len(label)}')



############## open camera % save data ##############
```

```
while(True):


    ret, frame = cap.read()
    overlay = frame.copy()
    ############## show info ##############
    text =   '{}{}{}{}{}'.format(f'Label\n0: {label[0]}\n1: {label[1]}\n',
        f'2: {label[2]}\n3: {label[3]}\n' ,
        f'4: {label[4]}\n5: {label[5]}\n' ,
        f'6: {label[6]}\n7: {label[7]}\n' ,
        f'8: {label[8]}\n9: {label[9]}\n' )
    for i, txt in enumerate(text.split('\n')):
        cv2.putText(overlay, txt,(20,25*i+20), cv2.FONT_HERSHEY_SIMPLEX, .5, (0,0,255), 1 )


    cv2.imshow('Create_Your_Own_Datasets', overlay)


    key = cv2.waitKey(1)


    if key== ord('q'): break


    for i in range(10):
        if key == ord(f'{i}'):
            save(trg_path, i, frame)


cap.release()
cv2.destroyAllWindows()
```

拍完之後資料都會存放在 data 的資料夾中 (雖然照片是顯示 custom_datasets)：

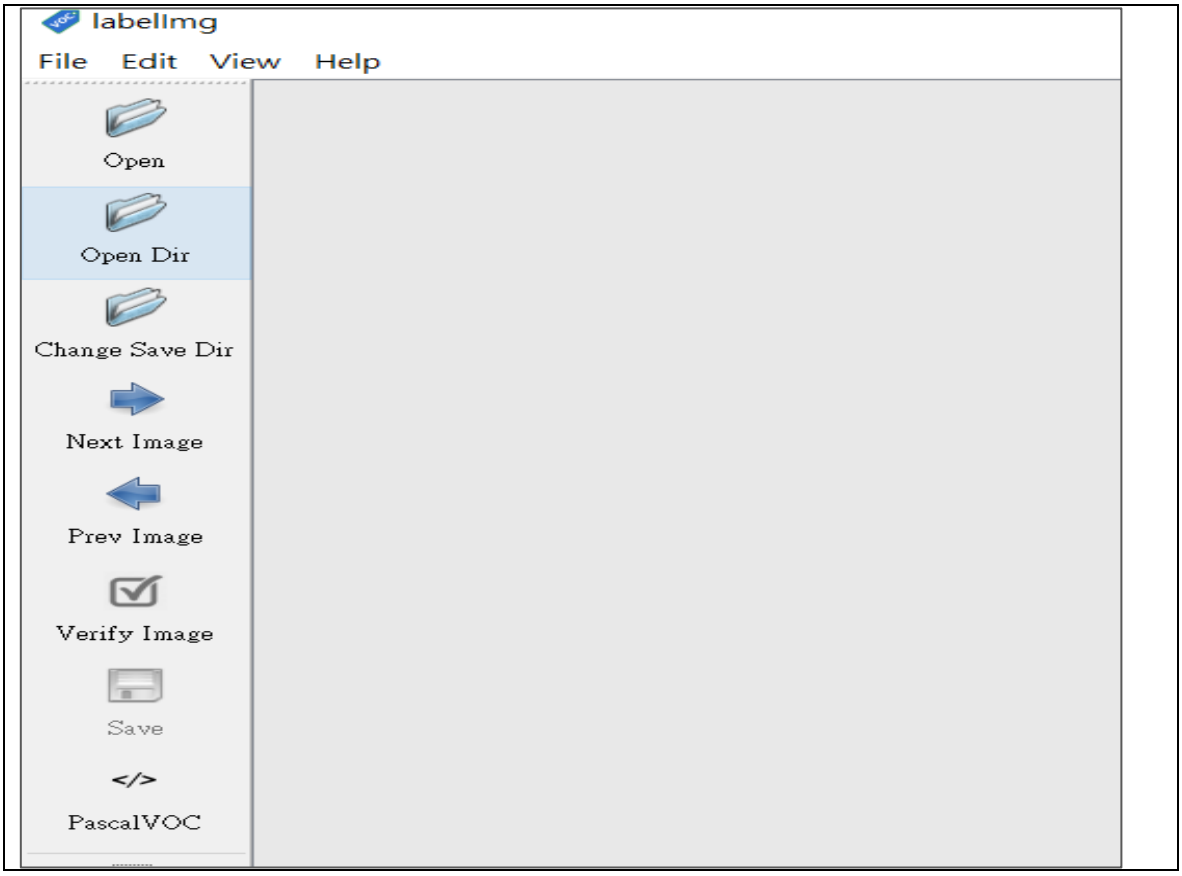


接下來就是最煩人標 Label 了，我們可以透過許多工具來完成，這邊我們使用的是

LabelImg，一個非常簡易的工具，直接有支援 YOLO 格式。

點擊 Open Dir 選擇欲開啟的數據集、Change Save Dir 則是標完的數據放哪裡、

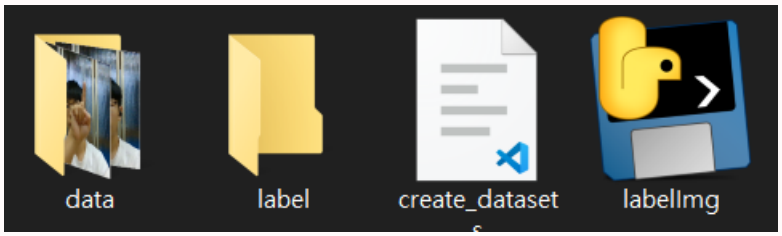
PascalVOC 是儲存的格式再點一下會變成 YOLO。



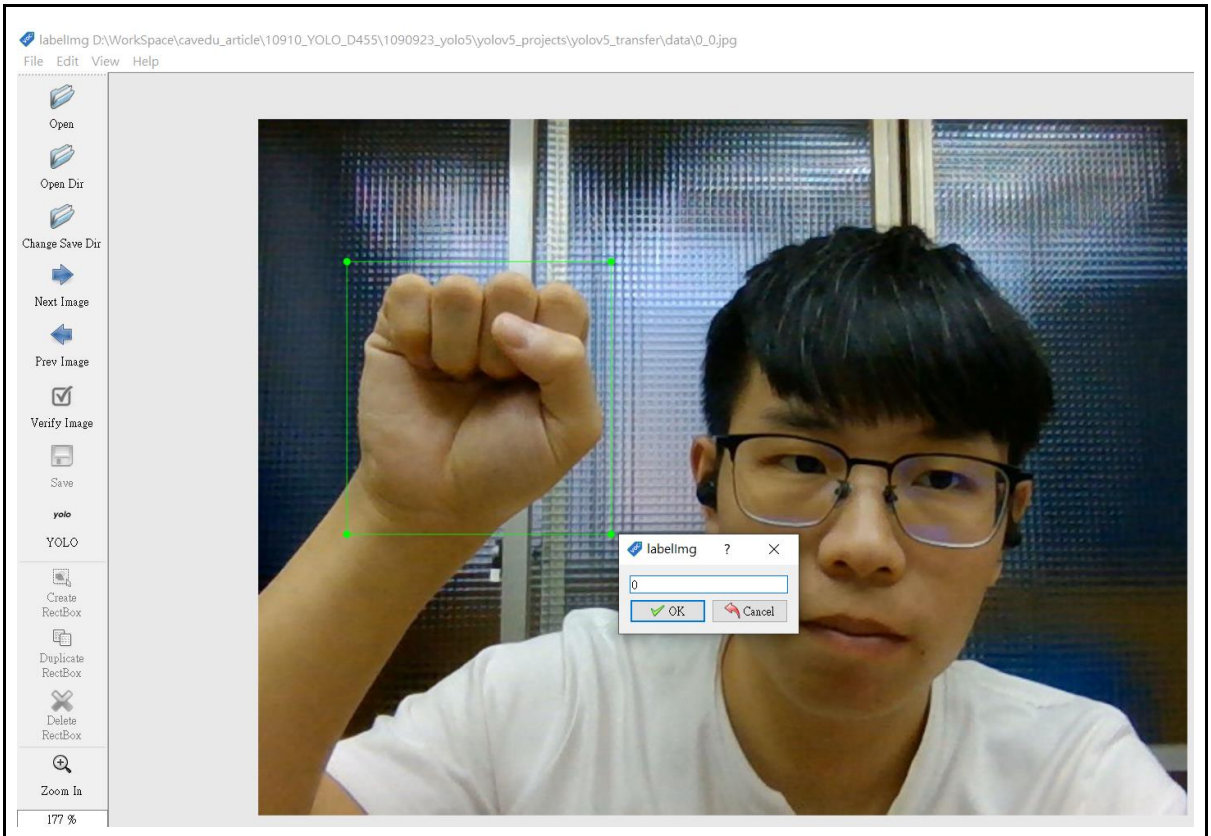
開啟照片的所在資料夾之後會讓你選擇 Annotation 的資料夾，也就是標籤的文檔。

PascalVOC 的話是選擇 .xml 而 YOLO 則是.txt，最後因為還要存放標籤所以也新增了

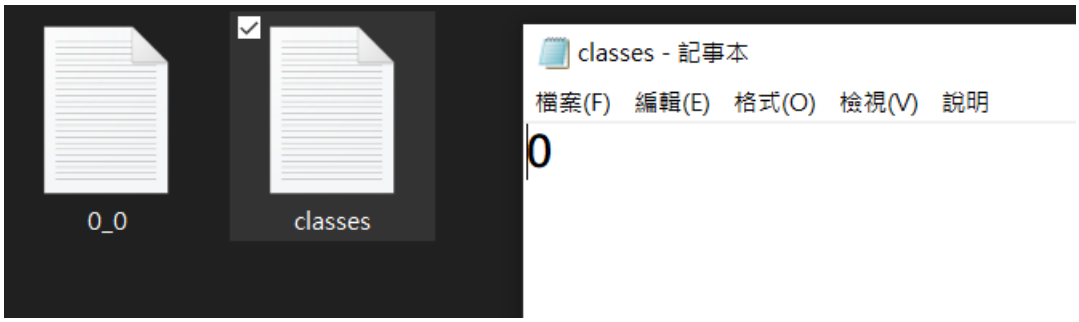
一個 label 的資料夾。



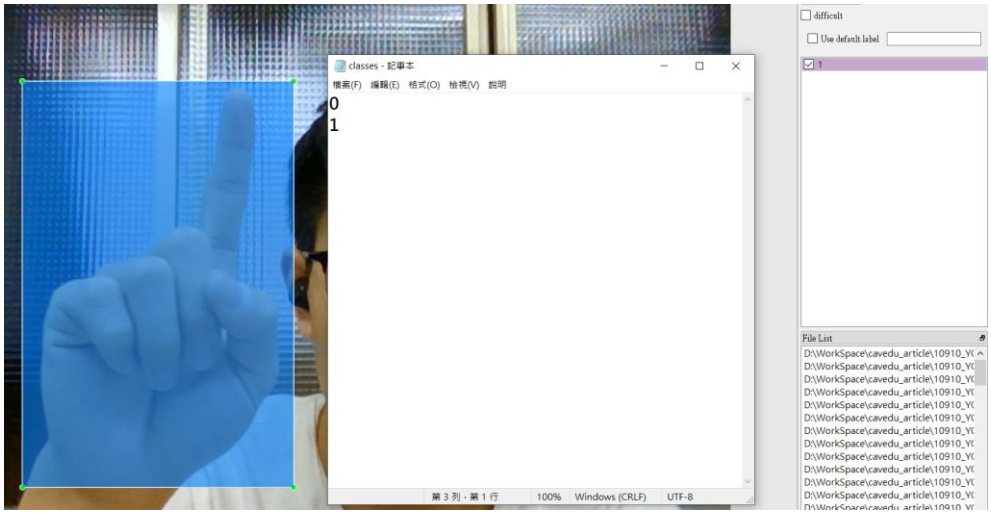
都準備好就可以開始標 Label 了，按下快捷鍵 w 可以繪製，接著就可以儲存：



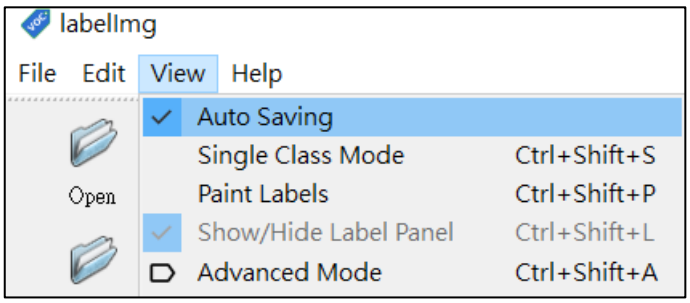
接下來就可以看到 label 的資料夾已經有對應的 txt 檔產生：



當我們標到其他標籤，classes 會自動更新：



這邊可以開啟　Auto Saving 才不用每次都要再按儲存：



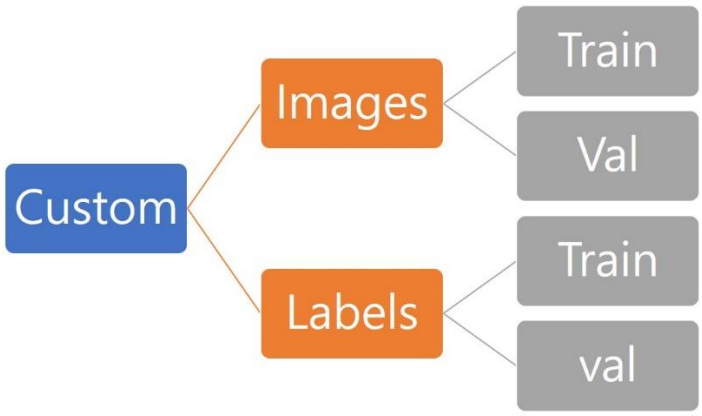
而他當然也可以標多個標籤，這邊我直接擷取教學影片的畫面：

整體而言，如果資料量不大，LabelImg 算是不錯的選擇，市面上有些已經結合影像

處理或物件辨識先幫使用者標出目標，使用者再做二次修改的！不過那些都要付費

就是了！所以教學或舉例用這種簡易版也是不錯的選擇。

接下來我們要將數據資料稍微整理一下，目標如下：



先取得資料以及確認一下資料夾是否存在，不存在就創建：

```
############## Check dir ##############

dataset_dir ='data'
label_dir = 'label'
dataset = os.listdir(dataset_dir)
label = os.listdir(label_dir)

images = []
labels = []

custom_dir = 'custom'
data_root = os.path.join(custom_dir, 'images')
label_root = os.path.join(custom_dir, 'labels')

data_train_dir = os.path.join(data_root, 'train')
data_val_dir = os.path.join(data_root, 'val')
label_train_dir = os.path.join(label_root, 'train')
```

```
label_val_dir = os.path.join(label_root, 'val')


dir_list = [data_train_dir, data_val_dir, label_train_dir, label_val_dir]


for dir in dir_list:
    if os.path.exists(dir)==False:
        print(f"Create dir : {dir}")
        os.makedirs(dir)
```

接著定義一些常用的副函式 (打亂順序、拼接檔名) 以及取得驗證資料 (因為資料較少所以我只取 5 筆)：

```
############## Get val data ##############


#打亂數據
def shuffle(data):
    arr = np.array(data)
    np.random.shuffle(arr)
    return arr.tolist()


#拼接檔名
def get_path(src, name, ftype):
    return f'{src}\{name}.{ftype}'


val_num = 5
pure_name = [ i.split('.')[0] for i in dataset ]
val_data = shuffle(pure_name)[:val_num]


print('Total Data length: ', len(pure_name))
print('Validation Data: ', val_data)


for d in val_data:


    src_data = get_path(dataset_dir, d, 'jpg')
    src_label = get_path(label_dir, d, 'txt')
    trg_data = get_path(data_val_dir, d, 'jpg')
    trg_label = get_path(label_val_dir, d, 'txt')
```

```
        shutil.copy(src_data, trg_data)
        shutil.copy(src_label, trg_label)

        pure_name.remove(d)
```

在取得驗證資料的同時我也將資料列表的驗證資料名稱刪除，所以接下來只要把剩餘的資料移動到 train 中的 images、labels 即可。

```
############## Split data ##############

print("="*50)
print('New Data length: ', len(pure_name))

for d in pure_name:

        src_data = get_path(dataset_dir, d, 'jpg')
        src_label = get_path(label_dir, d, 'txt')
        trg_data = get_path(data_train_dir, d, 'jpg')
        trg_label = get_path(label_train_dir, d, 'txt')

        shutil.copy(src_data, trg_data)
        shutil.copy(src_label, trg_label)

print("Finish！")
```

完成資料整理後就要去定義 yaml 檔，yolo 的訓練都會依靠 yaml 來定義數據集的資訊，而 yaml 檔都會放在 yolov5\data 資料夾當中，我使用的方法是複製 coco.yaml 來修改，檔名先改成 custom.yaml，內容的部分主要修改 nc 為標籤的種類數量、name 為標籤名稱：

```
# train and val data as 1) directory: path/images/, 2) file: path/images.txt, or 3) list: [path1/images/, path2/images/]
train: ../custom/images/train/    # 128 images
val: ../custom/images/train/    # 128 images
```

```
# number of classes
nc: 2


# class names
names: ['0', '1']
```

Colab 進行 Training

全部完成之後就要進行訓練了，我們使用 Colab 來實現，大概跑個 10 個 epochs 就有一些成效出現了！首先，要先上傳到 Google Drive (以下簡稱 GDrive) 上，這邊我們上傳到 GDrive 上之後要注意一下檔案擺放的位置記得是 yolov5 資料夾與 custom 資料夾同層。

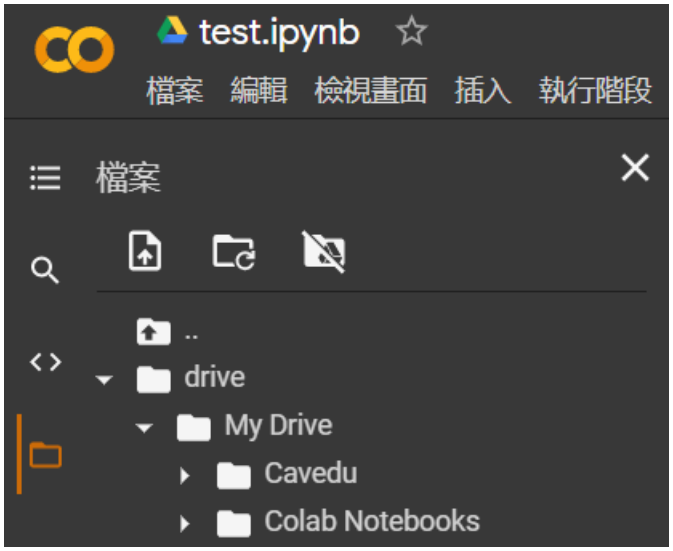
接著開啟一個 Colab 檔，從編輯>筆記本設定中轉換成 GPU 模式，並且稍微查看一下自己拿到哪一個 GPU。

```
import torch
print(torch.cuda.get_device_properties(0))
```

```
import torch
print(torch.cuda.get_device_properties(0))
```

```
_CudaDeviceProperties(name='Tesla K80', major=3, minor=7, total_memory=11441MB, multi_processor_count=13)
```

讓 Colab 跟自己的 GDrive 連動，因為我連動過了所以畫面不太一樣：



連動之後可以透過 os 套件移動到 yolov5 的位置，並且安裝缺失套件：

```
import os
os.chdir('/content/drive/My Drive/Cavedu/Article/yolov5')
!ls
!pip install -U PyYAML
```

```
coco2017labels.zip  inference     requirements.txt  train.py        yolov5s.pt
data                LICENSE       runs              tutorial.ipynb
detect.py           models        sotabench.py      utils
Dockerfile          __pycache__   test.ipynb        weights
hubconf.py          README.md     test.py           yolov5l.pt
Collecting PyYAML
  Downloading https://files.pythonhosted.org/packages/64/c2/b80047c7ac2478f9501676c988a5411ed5572f35d1beff9cae07d321512c/PyYAML-5.3.1.tar.gz (269kB)
     |████████████████████████████████| 276kB 4.6MB/s
Building wheels for collected packages: PyYAML
  Building wheel for PyYAML (setup.py) ... done
  Created wheel for PyYAML: filename=PyYAML-5.3.1-cp36-cp36m-linux_x86_64.whl size=44619 sha256=31f593b0b21c707aea53246d93fdbf81640fb7fa047c8a25f5c6de
  Stored in directory: /root/.cache/pip/wheels/a7/c1/ea/cf5bd31012e735dc1dfea3131a2d5eae7978b251083d6247bd
Successfully built PyYAML
Installing collected packages: PyYAML
  Found existing installation: PyYAML 3.13
    Uninstalling PyYAML-3.13:
      Successfully uninstalled PyYAML-3.13
Successfully installed PyYAML-5.3.1
```

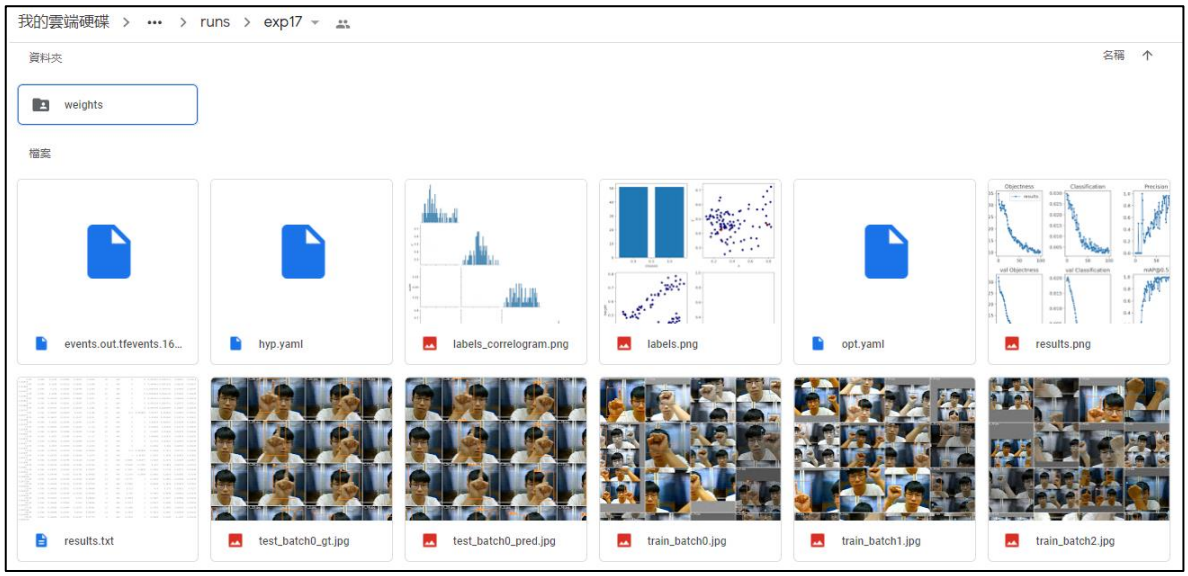
懶惰如我，改寫太麻煩直接使用指令進行訓練即可：

```
!!python train.py --img 640 --batch 16 --epochs 30 --data ./data/custom.yaml --cfg ./models/yolov5s.yaml --weights
'yolov5s.pt'
```

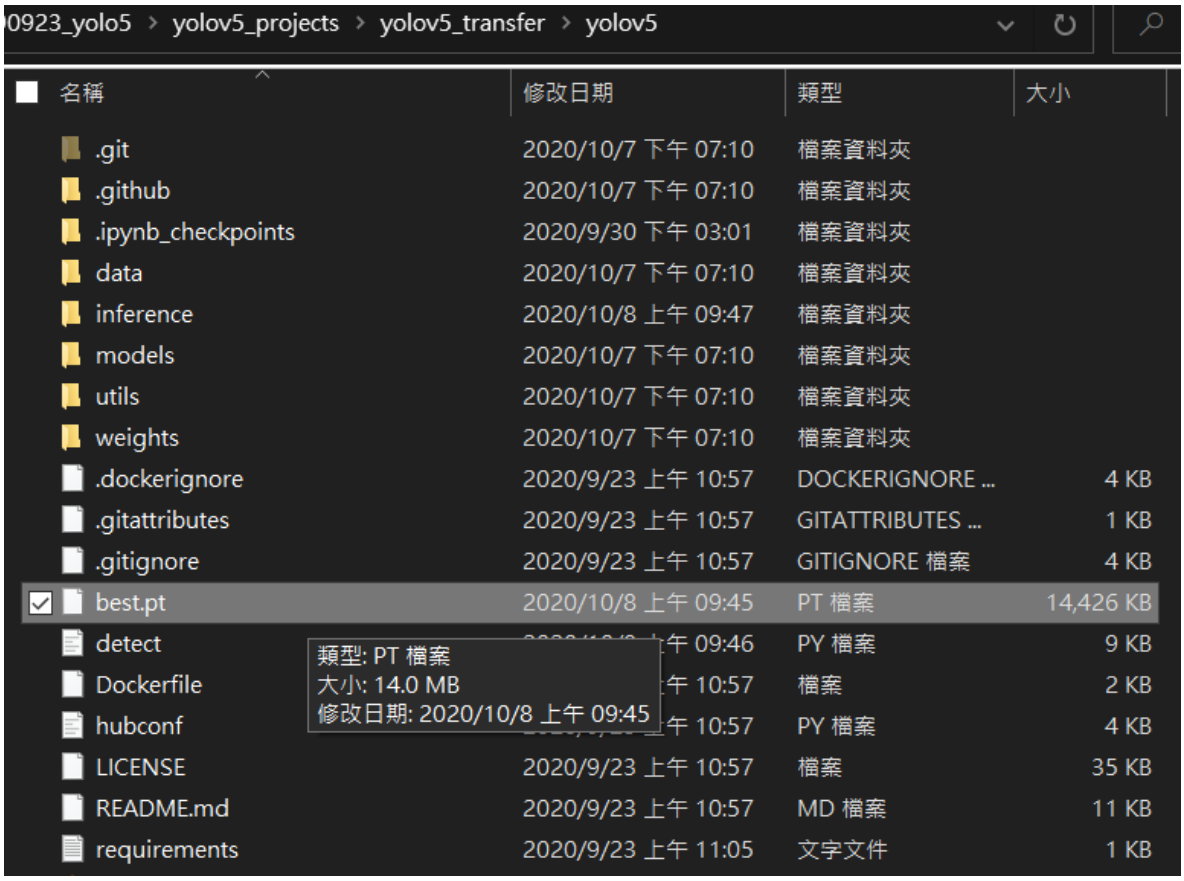
最後在 run 的資料夾中找到 exp 的資料夾，像我訓練過很多次所以數字顯示 17，

總之就是找到最後一個數字就是最後一次訓練的：

點開來可以看到有很多張驗證結果的圖片還有一個存放權重的資料夾：

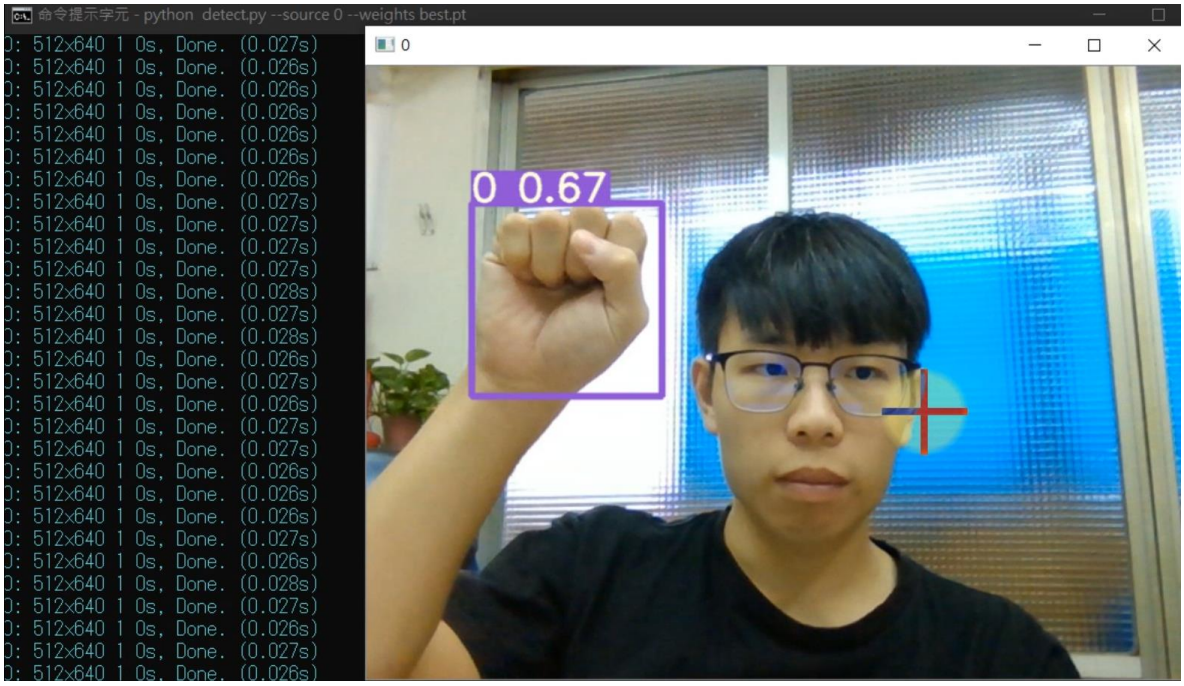


裡面通常會有兩個，一個是 best 一個 last，就如字面上的意思就是最好的跟最後的，當然！看到 best 就直接選 best！我們將其下載到我們電腦端的 YOLOv5 資料夾中。



執行 detect.py 的程式，並且使用 –source 0 ，也就是開啟相機進行即時影像辨識的部分，然後權重則選擇我們剛剛下載下來的權重：

```
python detect.py --source 0 --weights best.pt
```



可以看到已經有不錯的成效了，不過因為訓練數據太少，回合數太低，把這兩項加強之後應該就會改善不穩定的狀況了~當然照片的背景也很重要，換個背景說不定準度又下降了！

影片成果

22_YOLOv5_Transfer.mp4

結語

這篇結束相信你已經學會如何利用強大的 YOLOv5 來實作自己的客製化物件辨識了！

如果有想更了解什麼請再留言告訴我。

相關文章
在 Jetson Nano (TX1/TX2)上使用 Anaconda 与 PyTorch 1.1.0 https://zhuanlan.zhihu.com/p/64868319