

CAVEDU 教育團隊部落格

[技術教學文-格式]

標題	Jetson Nano 實作 YOLOv5 並嘗試進行台灣即時路況辨識		
<div><div>LABA076-01漢中街122號4樓頂</div><div>yolov5s.pt inference: 0.1616s fps: 5</div><div><div>car 0.44</div><div>car 0.70</div><div>person 0.53</div><div>car 0.69</div><div>car 0.84</div><div>person 0.63</div><div>person 0.57</div><div>person 0.64</div><div>car 0.67</div></div><div>20/10/14 17:03:00</div></div>			
分類		標籤	YOLOv5 Jetson Nano
撰寫/攝影			
前情提要			
時間		材料表	
成本			
難度			

前言

在 YOLOv4 推出沒多久後 YOLOv5 就默默推出了，雖然還沒有被官方正名但是經過了幾個月，該有的地雷應該都被前人踩過了。所以這陣子就開始嘗試 YOLOv5 的實作，除了利用 Jetson Nano 進行範例程式的實作之外，還下載了即時的路況監視器畫面並更改一下範例程式，讓它可以運行即時影像辨識。

內文

前景提要

當時 YOLOv4 出了沒多久 YOLOv5 舊悄然推出了，但是你可以發現 v5 其實不是 v4 的團隊 (Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao) 所做更不是 YOLO 之父 (Joseph Redmon) 所做，而是一間名為 Ultralytics LLC 的公司所開發 (之前它一直都有發布將 YOLO 轉成 PyTorch 的 Github)，在 YOLOv5 發布之前也沒發布論文來佐證 YOLOv5，所以其實很多人對它的存在感到懷疑也鬧出了很大的風波。不過它是基於 PyTorch 所實現，整個架構跟 YOLOv3、v4 的 DarkNet 環境截然不同，對於再修改跟開發上比較簡單，接著就跟我一起利用 Jetson Nano 來完成 YOLOv5 的實作吧！

如果你想了解更多可以看看 Ultralytics LLC 出面說明 YOLOv4 與 YOLOv5 差異 - <https://blog.roboflow.com/yolov4-versus-yolov5/>，而 Ultralytics 也有推出基於 YOLO 的 APP (僅限 IOS)，現在也更新到 YOLOv5 了。



訓練環境

先前都直接用原生的系統來裝套件，我其實不那麼推薦因為有時候不同的專案會需要不同的版本，為了將其獨立開來建議是使用虛擬環境來安裝比較合適。今天會稍微介紹一下虛擬環境的部分，我在 Windows 上常會使用 Anaconda 而 Jetson Nano 上因為 Anaconda 不支援 aarch64 (Arm64) 的核心所以要另外編譯，非常之麻煩！所以我直接使用 Virtualenv (另一個輕便的虛擬環境套件)。

安裝 virtualenv 以及 virtualenvwrapper：

```
$ sudo pip3 install virtualenv virtualenvwrapper
```

修改環境變數：

```
$ nano ~/.bashrc  
export WORKON_HOME=$HOME/.virtualenvs  
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
```

```
export /usr/local/bin/virtualenvwrapper.sh
```

建置虛擬環境：

```
$ mkvirtualenv yolov5
```

開啟虛擬環境：

```
$ workon yolov5
```

可以看到前面會多一個括弧 (env_name) 就是你目前的環境名稱：

```
chun@chun-jetsonNano:~/Chun$ workon yolov5
(yolov5) chun@chun-jetsonNano:~/Chun$
```

接下來先安裝 git，因為要下載 YOLOv5：

```
$ sudo apt-get install git-all -y
```

下載 YOLOv5 的 Github：

```
$ git clone https://github.com/ultralytics/yolov5.git
```

接著安裝所需套件，可以先打開 requirements.txt 來看看所需套件，因為不管是

Raspberry Pi 還是 Jetson 系列，安裝 PyTorch 或 OpenCV 都有特定的方式或來源，所

以我在嘗試別人的 Github 時都會分別開來安裝。

```
GNU nano 2.9.3 requirements.txt Modified
# pip install -r requirements.txt


# base -----
Cython
matplotlib>=3.2.2
numpy>=1.18.5
opencv-python>=4.1.2
pillow
PyYAML>=5.3
scipy>=1.4.1
tensorboard>=2.2
torch>=1.6.0
torchvision>=0.7.0
tqdm>=4.41.0

# coco -----
# pycocotools>=2.0

# export -----
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^\ Replace   ^U Uncut Text ^T To Spell  ^_ Go To Line
```

像是這邊可以看到 Cython、Numpy 都在安裝 PyTorch 的時候會一起安裝，而 OpenCV

因為原生就有所以用 Link 的方式就可以了，所以我們先來處理比較特別的 OpenCV 跟 PyTorch。



dusty_nv Moderator26 Mar '19

Below are pre-built PyTorch pip wheel installers for Python on Jetson Nano, Jetson TX1/TX2, and Jetson Xavier NX/AGX with JetPack 4.2 and newer.

You can now download the `l4t-pytorch` and `l4t-m1` containers from **NGC** for JetPack 4.4

Download one of the PyTorch binaries from below for your version of JetPack, and see the installation instructions to run on your Jetson. These pip wheels are built for ARM aarch64 architecture, so run these commands on your Jetson (not on a host PC).

PyTorch pip wheels

- ▶ PyTorch v1.6.0
- ▶ PyTorch v1.5.0
- ▶ PyTorch v1.4.0
- ▶ PyTorch v1.3.0
- ▶ PyTorch v1.2.0
- ▶ PyTorch v1.1.0
- ▶ PyTorch v1.0.0

Instructions

- ▶ Installation
- ▶ Verification
- ▶ Build from Source
- ▶ Note on Upgrading pip

這是官方提供的教學 [PyTorch for Jetson - version 1.6.0 now available](#)，首先在 YOLOv5 提出的安裝套件可以看到建議是 1.6 以上，目前只有 JetPack4.4 才能支援 PyTorch 1.6 哦！所以要特別注意一下自己的 JetPack 版本：

```
$ sudo apt-cache show nvidia-jetpack
```

```
(yolov5) chun@chun-jetsonNano:~/Chun$ sudo apt-cache show nvidia-jetpack
Package: nvidia-jetpack
Version: 4.4-b186
Architecture: arm64
Maintainer: NVIDIA Corporation
Installed-Size: 194
Depends: nvidia-cuda (= 4.4-b186), nvidia-opencv (= 4.4-b186), nvidia-cudnn8 (= 4.4-b186), nvidia-tensorrt (= 4.4-b186), nvidia-visionworks (= 4.4-b186), nvidia-container (= 4.4-b186), nvidia-vpi (= 4.4-b186), nvidia-l4t-jetson-multimedia-api (> 32.4-0), nvidia-l4t-jetson-multimedia-api (< 32.5-0)
Homepage: http://developer.nvidia.com/jetson
Priority: standard
Section: metapackages
Filename: pool/main/n/nvidia-jetpack/nvidia-jetpack_4.4-b186_arm64.deb
Size: 29346
SHA256: 64f791ddc010f5769b838e7bc28225bb1cc836888c2b7c1989efc396b6b8a7e0
SHA1: 12532999c9fa4688cad2d8506174f77ec696e98a
MD5sum: 4412e36f9dba41a27013b8193c918870
Description: NVIDIA Jetpack Meta Package
Description-md5: ad1462289bdbc54909ae109d1d32c0a8
```

安裝 PyTorch :

```
$ wget https://nvidia.box.com/shared/static/9eptse6jyly1ggt9axbja2yrmj6pbarc.whl -O torch-1.6.0-cp36-cp36m-linux_aarch64.whl
$ sudo apt-get install python3-pip libopenblas-base libopenmpi-dev
$ pip3 install Cython
$ pip3 install numpy torch-1.6.0-cp36-cp36m-linux_aarch64.whl
```

安裝 torchvision :

```
$ sudo apt-get install libjpeg-dev zlib1g-dev
$ git clone --branch v0.7.0 https://github.com/pytorch/vision torchvision
$ cd torchvision
$ export BUILD_VERSION=0.7.0
$ sudo python setup.py install
```

這部分大概 10 分鐘內能搞定，可以透過導入函式庫查看版本來確認是否安裝成功：

```
(yolov5) chun@chun-jetsonNano:~/Chun$ python
Python 3.6.9 (default, Jul 17 2020, 12:50:27)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> import torchvision
>>> print(torch.__version__, torchvision.__version__)
1.6.0 0.7.0
>>> █
```

接著要找到原生的 OpenCV 位置：

```
$ sudo find / -name cv2
```

```
(yolov5) chun@chun-jetsonNano:~/Chun$ sudo find / -name cv2
find: '/run/user/1000/gvfs': Permission denied
/usr/lib/python2.7/dist-packages/cv2
/usr/lib/python3.6/dist-packages/cv2
```

尋找 .so 檔案，大家應該都一樣會在 /usr/lib/python3.6/dist-packages/cv2/python-

3.6/ 裡面：

```
(yolov5) chun@chun-jetsonNano:~/Chun$ ls /usr/lib/python3.6/dist-packages/cv2
config-3.6.py  config.py  __init__.py  load_config_py2.py  load_config_py3.py  python-3.6
(yolov5) chun@chun-jetsonNano:~/Chun$ ls /usr/lib/python3.6/dist-packages/cv2/python-3.6/
cv2.cpython-36m-aarch64-linux-gnu.so
```

接著就要建立連結，使用 `ln` 指令：

```
$ ln -s /usr/lib/python3.6/dist-packages/cv2/python-3.6/cv2.cpython-36m-aarch64-linux-gnu.so ~/.virtualenvs/{your env}/cv2.cpython-36m-aarch64-linux-gnu.so
```

```
(yolov5) chun@chun-jetsonNano:~/.virtualenvs/yolov5$ ls
bin  include  lib  share
(yolov5) chun@chun-jetsonNano:~/.virtualenvs/yolov5$ ln -s /usr/lib/python3.6/dist-packages/cv2/python-3.6/cv2.cpython-36m-aarch64-linux-gnu.so ~/.virtualenvs/yolov5/cv2.cpython-36m-aarch64-linux-gnu.so
(yolov5) chun@chun-jetsonNano:~/.virtualenvs/yolov5$ ls
bin  cv2.cpython-36m-aarch64-linux-gnu.so  include  lib  share
```

確認是否安裝成功：

```
(yolov5) chun@chun-jetsonNano:~/.virtualenvs/yolov5$ python
Python 3.6.9 (default, Jul 17 2020, 12:50:27)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'4.1.1'
>>>
```

剩下還沒安裝的套件整理一下會變成下面

```
$ pip3 install matplotlib pillow pyyaml tensorboard tqdm scipy
```

其中 `scipy` 可能原本就有了，由於它安裝要好一陣子所以我建議如果有先執行看看，不行再安裝。

執行 YOLOv5 的範例程式

使用 Github 範例程式 `detect.py`，範例程式主要會用到的引數

<code>--source</code>	圖片、影片目錄或是 0 開啟攝影機
<code>--iou_thred</code>	信心指數的閾值，雖然低一些可框出越多東西但準確度就不敢保證
<code>--weights</code>	權重，有 s、m、l、x 之分

我們可以下載訓練好的 **weights**，可以利用作者的 `download_weight.sh` 來下載，他需要用到 `util` 資料夾的程式所以我有移動到上一層目錄，嫌麻煩的當然也可以直接到他的 [GoogleDrive](#) 下載。

```
$ cd weights/  
$ mv download_weights.sh ..  
$ cd ..  
$ bash download_weights.sh
```

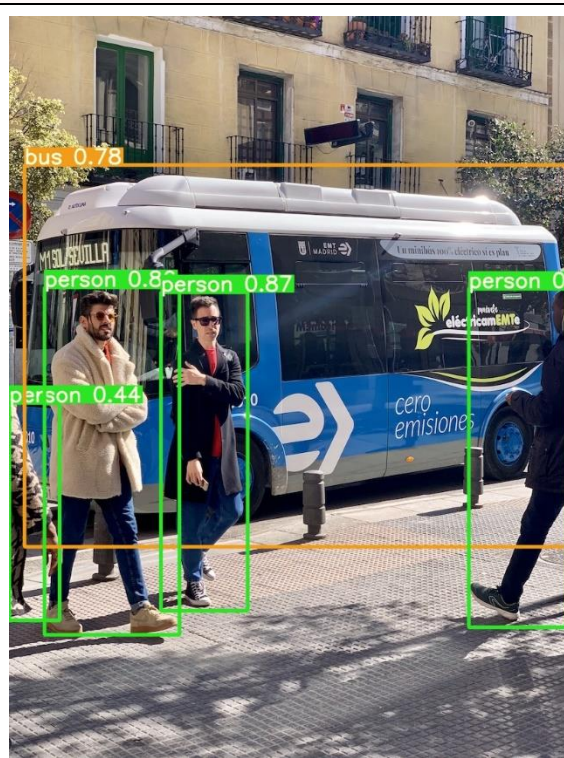
接著可以執行 `detec.py`，我們先使用官網提供的範例圖來測試：

```
$ python detect.py --source inference/images/ --weights weights/yolov5s.pt
```

運行結果如下：

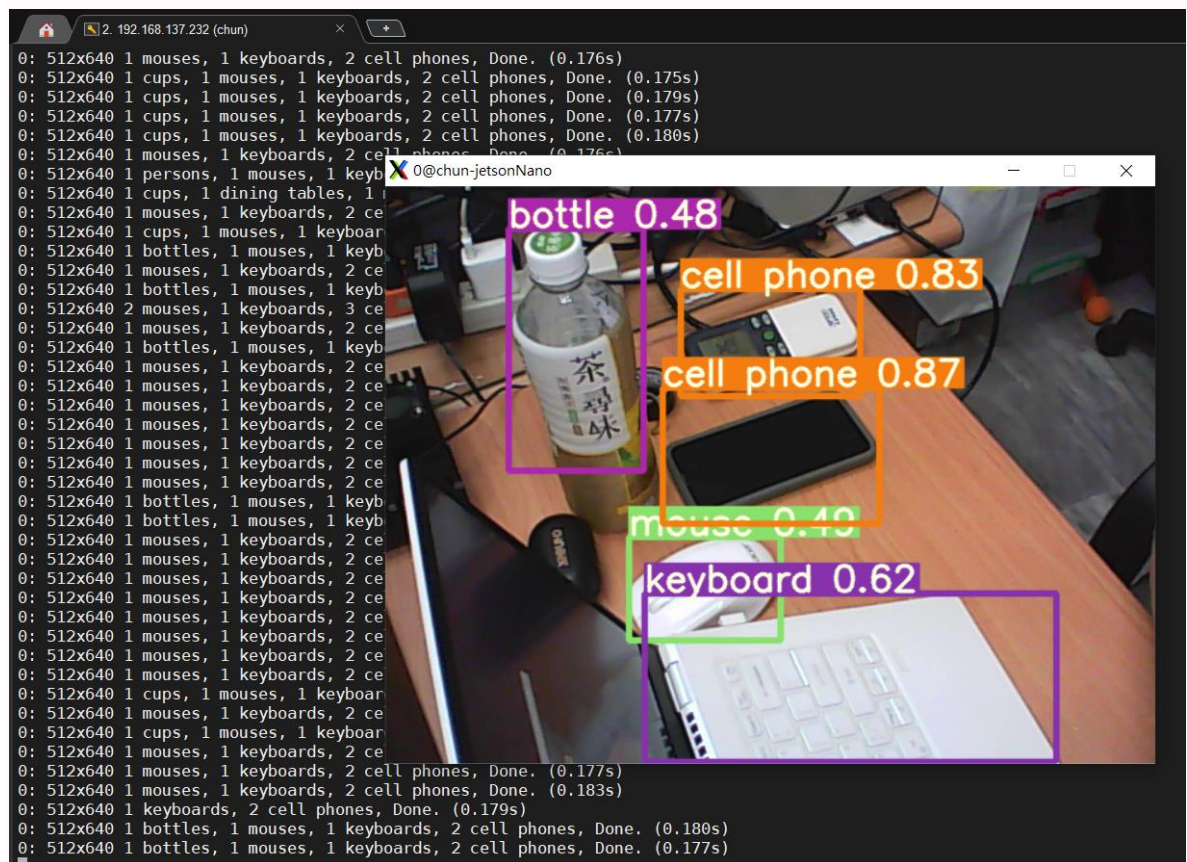
```
(yolov5) chun@chun-jetsonNano:~/Chun/yolov5$ python detect.py --source inference/images/ --weights weights/yolov5s.pt  
Namespace(agnostic_nms=False, augment=False, classes=None, conf_thres=0.4, device='', img_size=640, iou_thres=0.5, output='inference/output', save_txt=False, source='inference/images/', update=False, view_img=False, weights=['weights/yolov5s.pt'])  
Using CUDA device0 _CudaDeviceProperties(name='NVIDIA Tegra X1', total_memory=3962MB)  
Fusing layers...  
Model Summary: 191 layers, 7.46816e+06 parameters, 0 gradients  
image 1/2 /home/chun/Chun/yolov5/inference/images/bus.jpg: 640x512 4 persons, 1 buss, Done. (0.686s)  
image 2/2 /home/chun/Chun/yolov5/inference/images/zidane.jpg: 384x640 2 persons, 2 ties, Done. (0.165s)  
Results saved to inference/output  
Done. (18.201s)
```

對兩張圖片進行推論，耗費時間為 18 秒：



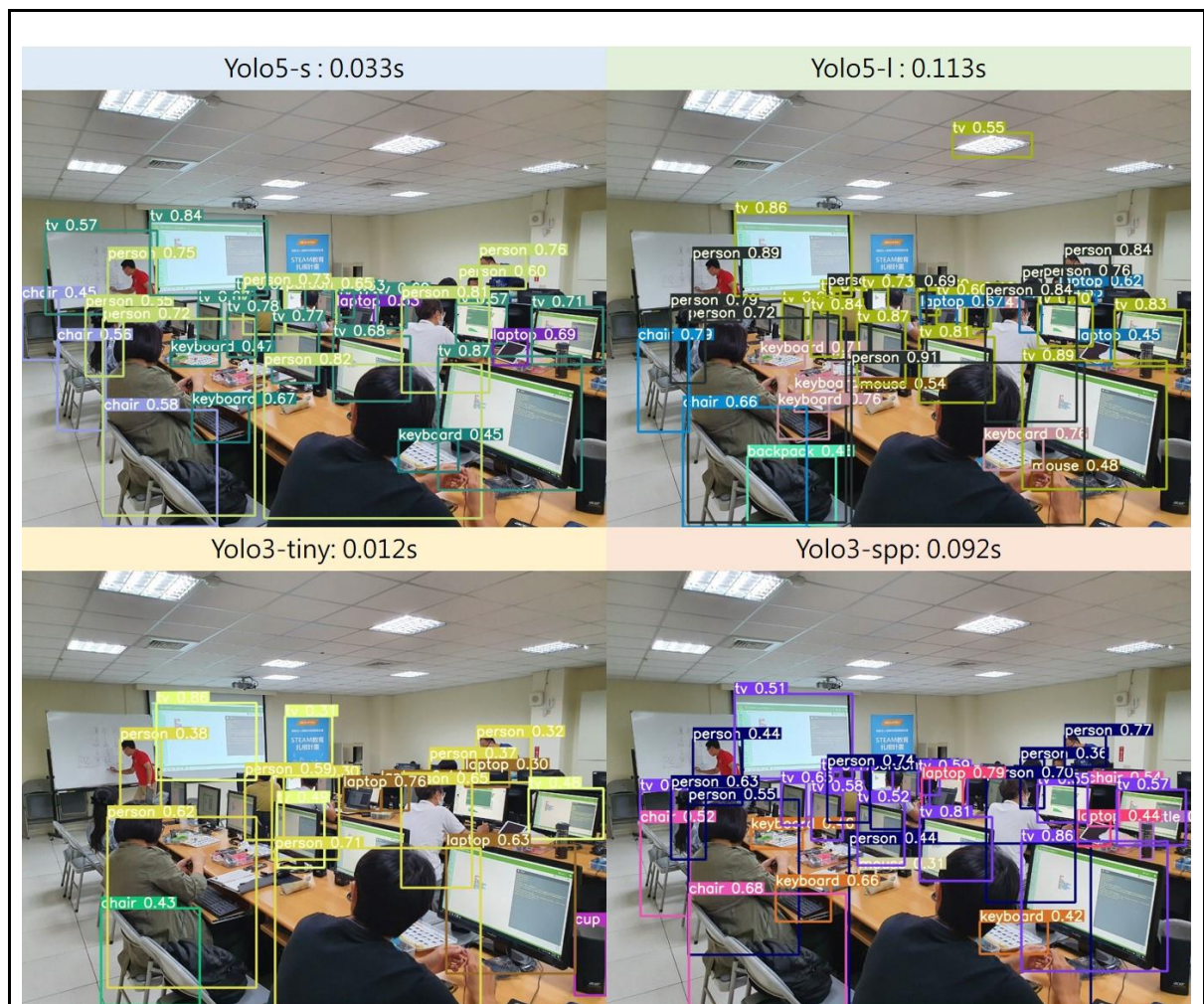


還可以接上相機做即時影像偵測：



比較各模型差異

這邊我們拿 YOLOv3 跟 YOLOv5 來做比較，可以注意到 v3-tiny 雖然秒數最少但運行結果不盡理想；然後 v5-s 目前看起來最好，秒數少但框出來的物品多準確度也蠻高的；v5-l 、v3-spp 準確度高但是也會框到一些錯誤的物品。



拿 YOLOv5 來應用在即時路況影像

對於 Jetson 系列的開發版相當多人會拿來做自駕車，而 YOLO 所訓練的 coco_2017

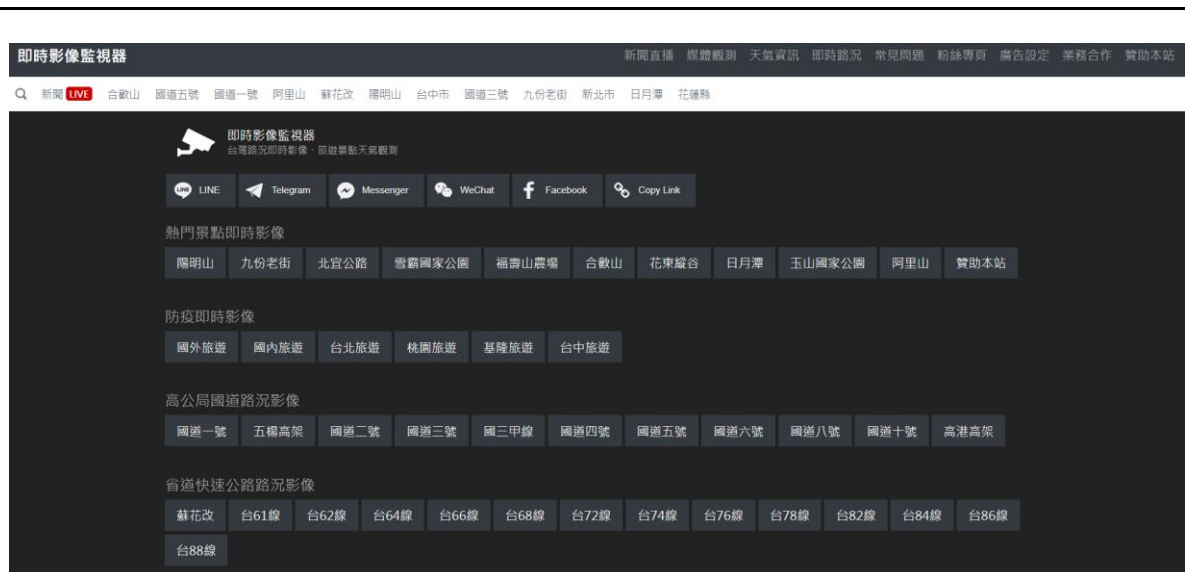
數據集也能用於偵測人、車，所以我們先直接拿 pre-trained model 來實際運行看

看，第一步是要取得即時路況影像，這樣類型的影像直接用手機路就太過時了，

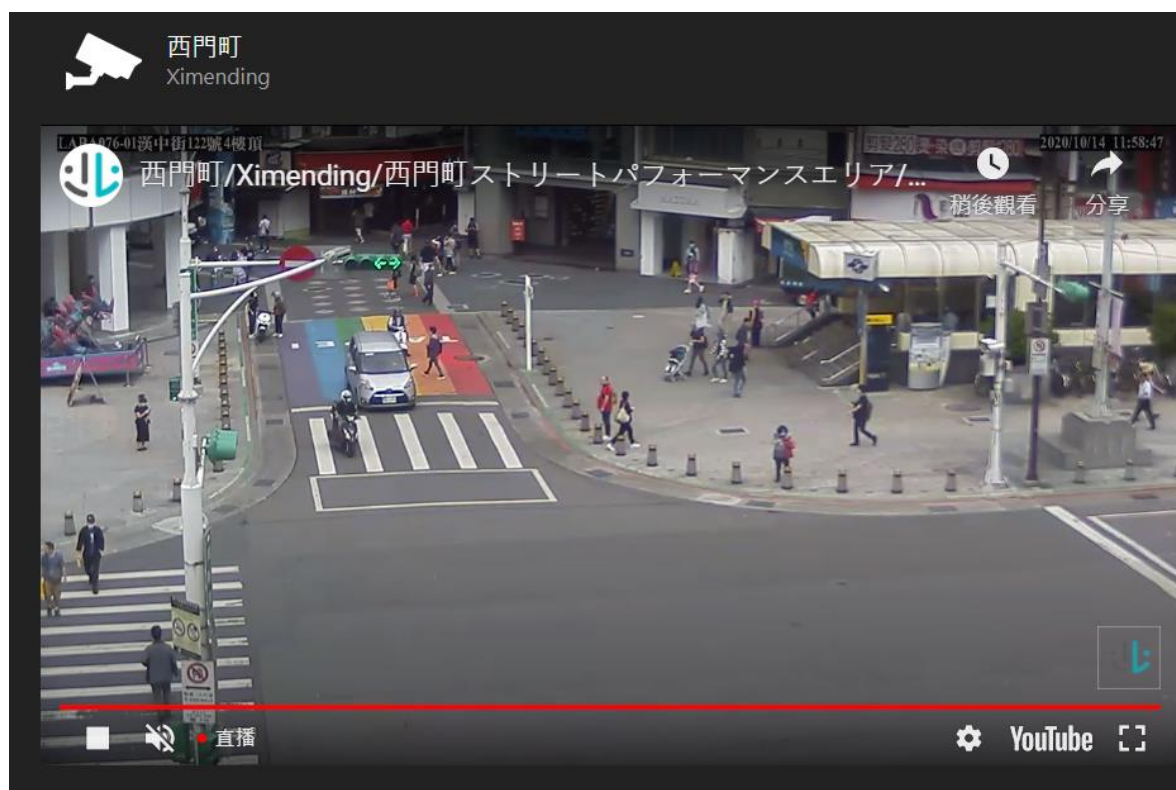
所以我們來玩點不一樣的，我們可以到下列這個網站獲取「[即時影像監視器](https://tw.live/)」

<https://tw.live/>

這個網站有各式各樣的台灣路況可以查看，而這些都是即時影像。



仔細看了一下西門町的路口監視器畫面比較清晰也人多，所以最後我選擇西門町的即時影像，接下來就要考慮一個大問題了~我該如何將直播影片給下載下來！



其實你可以發現它從 Youtube 直播影片連動過來的，所以我寫了這支程式用來擷取 Youtube 影像直播，主要利用 pafy 跟 vlc 來下載 mp4 影片，並且利用 moviepy 來

剪輯預設秒數，首先先安裝相關套件：

```
$ pip install pafy youtube-dl python-vlc moviepy
```

因為 `moviepy` 跟影片有關要安裝相關的編碼格式，Windows 本身就有了但是 Linux 需要額外安裝，我透過 `pip` 安裝在虛擬環境中是行不通的，所以只好安裝在本身的环境：

```
$ sudo apt install ffmpeg
```

接著就是主要程式的部分：

```
def capture_video(opt):

    f_name = 'org.mp4'      # 下載影片名稱
    o_name = opt.output     # 裁剪影片名稱
    sec = opt.second        # 欲保留秒數
    video = pafy.new(opt.url) # 取得 Youtube 影片

    r_list = video.allstreams # 取得串流來源列表

    print_div()
    for i,j in enumerate(r_list): print( '[ {} ] {} {}'.format(i,j.title,j))
    idx = input("\nChoose the resolution : ")

    if idx:
        ### 選擇串流來源
        trg = r_list[int(idx)]
        print_div('您選擇的解析度是: %s'%(trg))

        ### 下載串流
        vlcInstance = vlc.Instance()
        player = vlcInstance.media_player_new() # 創建一個新的 MediaPlayer 實例
        media = vlcInstance.media_new(trg.url)  # 創建一個新的 Media 實例
        media.get_mrl()
        media.add_option(f"sout=file/ts:{f_name}") # 將媒體直接儲存
        player.set_media(media)                   # 設置 media_player 將使用的媒體
```

```

player.play()                # 播放媒體
time.sleep(1)                # 等待資訊跑完

### 進度條、擷取影片

clock(sec)                   # 播放 sec 秒 同時運行 進度條
cut_video(f_name, sec, o_name) # 裁切影片，因為停 n 秒長度不一定是 n

### 關閉資源

player.stop()                # 關閉撥放器以及釋放媒體

```

其餘副函式，大部分是美觀用，像是用來取得終端機視窗大小以及打印分隔符號

等，在 clock 的部分費了些心思寫了類似 tqdm 的進度條，最後 cut_video 就是剪

取影片，從第 0 秒到第 n 秒：

```

### 取得 terminal 視窗大小
def get_cmd_size():
    import shutil
    col, line = shutil.get_terminal_size()
    return col, line

### 打印分隔用
def print_div(text=None):
    col, line = get_cmd_size()
    col = col - 1
    if text == None:
        print('-'*col)
    else:
        print('-'*col)
        print(text)
        print('-'*col)

### 計時、進度條
def clock(sec, sign = '#'):
    col, line = get_cmd_size()
    col_ = col - 42
    bar_bg = '-'*(col_)

```

```

print_div()

for i in range(sec+1):

    bar_idx = (col_//sec*(i+1))
    bar = "
    for t in range(col_): bar += sign if t <= bar_idx else bar_bg[t]

    percent = int(100/sec*(i))
    end = '\n' if i==sec else '\r'
    print('Download Stream Video [{:02}/{:02}s] [{}] ({:02}%).format(i, sec, bar, percent), end=end)
    time.sleep(1)

### 擷取特定秒數並儲存
def cut_video(name, sec, save_name):
    print_div()
    print('Cutting Video Used Moviepy\n')
    ffmpeg_extract_subclip(name, 0, sec, targetname=save_name)
    print_div(f'save file {save_name}')

```

為了使用更方便，我增加了 argparse 命令列選項，「-u」為 Youtube 連結；「-o」為輸出影片名稱；「-s」為輸出影片秒數：

```

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-u', '--url', help='youtube url')
    parser.add_argument('-o', '--output', type=str, default='sample.mp4', help='save file path\name')
    parser.add_argument('-s', '--second', type=int, default=10, help='video length')
    opt = parser.parse_args()
    capture_video(opt)

```

執行結果：

```

$ python capture_livestream.py -u
'https://www.youtube.com/watch?v=iVpOdRU0r9s&feature=emb_title&ab_channel=%E8%87%BA%E5%8C%97%E
5%B8%82%E6%94%BF%E5%BA%9CTaipeiCityGovernment' -o test.mp4 -s 10

```



```

(yolov5) chun@chun-jetsonNano:~/Chun/yolov5_video$ python capture_livestream.py -u 'https://www.youtube.com/watch?v=iVp0dRU0r9s&feature=emb_title&ab_channel=E8%7B%A%E5%8C%97%E5%B8%82%E6%94%B%E5%B3%A9CTaipeiCityGovernment' -o test.mp4 -s 10
[ 0 ] 西門町/Ximending/西門町ストリートパフォーマンスエリア/監視 normal:mp4@256x144
[ 1 ] 西門町/Ximending/西門町ストリートパフォーマンスエリア/監視 normal:mp4@426x240
[ 2 ] 西門町/Ximending/西門町ストリートパフォーマンスエリア/監視 normal:mp4@640x360
[ 3 ] 西門町/Ximending/西門町ストリートパフォーマンスエリア/監視 normal:mp4@854x480
[ 4 ] 西門町/Ximending/西門町ストリートパフォーマンスエリア/監視 normal:mp4@1280x720
[ 5 ] 西門町/Ximending/西門町ストリートパフォーマンスエリア/監視 normal:mp4@1920x1080

Choose the resolution : 4
您選擇的解析度是: normal:mp4@1280x720

[0000007f6c17d5c0] mpeg4audio decoder: AAC channels: 2 samplerate: 48000
[0000007f500016e0] ts demux error: libdvbpsi error (PSI decoder): TS duplicate (received 0, expected 1) for PID 0
[0000007f500016e0] ts demux error: libdvbpsi error (PSI decoder): TS duplicate (received 0, expected 1) for PID 4095
[0000007f500016e0] ts demux error: libdvbpsi error (PSI decoder): TS duplicate (received 0, expected 1) for PID 0
[0000007f500016e0] ts demux error: libdvbpsi error (PSI decoder): TS duplicate (received 0, expected 1) for PID 4095
[0000007f500016e0] ts demux error: libdvbpsi error (PSI decoder): TS duplicate (received 0, expected 1) for PID 0
[0000007f500016e0] ts demux error: libdvbpsi error (PSI decoder): TS duplicate (received 0, expected 1) for PID 4095
[0000007f500016e0] ts demux error: libdvbpsi error (PSI decoder): TS duplicate (received 0, expected 1) for PID 0
[0000007f500016e0] ts demux error: libdvbpsi error (PSI decoder): TS duplicate (received 0, expected 1) for PID 4095
Download Stream Video [10/10s] [#####] (100%)
Cutting Video Used Moviepy

Moviepy - Running:
>>> "+" ".join(cmd)
Moviepy - Command successful

save file test.mp4

(yolov5) chun@chun-jetsonNano:~/Chun/yolov5_video$ ls
capture_livestream.py detect.py inference_models org.mp4 readme.txt streaming_detected.py test.mp4 utils yolov5l.pt yolov5s.pt
(yolov5) chun@chun-jetsonNano:~/Chun/yolov5_video$

```

接著可以直接執行範例程式來運行看看：

```
$ python detect.py --source test.mp4 --weights yolov5s.pt
```

```

(yolov5) chun@chun-jetsonNano:~/Chun/yolov5_video$ python detect.py --source test.mp4 --weights yolov5s.pt
Namespace(agnostic_nms=False, augment=False, classes=None, conf_thres=0.4, device='', img_size=640, iou_thres=0.5, output='inference/output', save_txt=False,
sources='test.mp4', update=False, view_img=False, view_img=False, weights=['yolov5s.pt'])
Using CUDA device0_CudaDeviceProperties(name='NVIDIA Tegra X1', total_memory=3964MB)

Fusing layers...
Model Summary: 191 layers, 7.46816e+06 parameters, 0 gradients
video 1/1 (1/300) /home/chun/Chun/yolov5_video/test.mp4:

/home/chun/Chun/yolov5_video/test.mp4

(3, 384, 640)

(720, 1280, 3)

<VideoCapture 0x7f1d28d770>
384x640 3 persons, 3 cars, 1 motorcycles, Done. (0.307s)
video 1/1 (2/300) /home/chun/Chun/yolov5_video/test.mp4:

```

結果如下：

```

<VideoCapture 0x7f1d28d770>
384x640 7 persons, 2 cars, Done. (0.165s)
Results saved to inference/output
Done. (83.954s)

```

使用 Jetson Nano 運行平均 0.165 秒一幀，10 秒的影片總共耗費 83 秒完成，這邊

提供運行完的影片給大家參考：



yolo5_sample_vid
eo.mp4

但其實我個人覺得這樣的影片無法告訴讀者到底 Nano 效能是好是壞，所以我修改

了一下範例程式將它變成即時影像辨識的方式，程式中稍微計算了 FPS 大概在 5 左右，一個順暢的影片 FPS 至少要在 30 以上，所以可以看到有些許的卡頓，當然我用遠端也有可能造成更多的 Delay：



yolo5_stream_vid
eo_Trim.mp4

修改的內容相當簡單，就是將原本要讀取照片或影片的部分擷取出來，改成只有照片，並且在一開始讀檔的方式改成用 OpenCV 讀取影像，最終修改後的程式如下：

```
import argparse
import os
import platform
import shutil
import time
from pathlib import Path
import numpy as np

import cv2
import torch
import torch.backends.cudnn as cudnn
from numpy import random

from models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages
from utils.general import (
    check_img_size, non_max_suppression, apply_classifier, scale_coords,
    xyxy2xywh, plot_one_box, strip_optimizer, set_logging)
from utils.torch_utils import select_device, load_classifier, time_synchronized

# Image process
#####

def img_preprocess(img0, img_size=640):
```

```

# Padded resize
img = letterbox(img0, new_shape=img_size)[0]

# Convert
img = img[:, :, ::-1].transpose(2, 0, 1) # BGR to RGB, to 3x416x416
img = np.ascontiguousarray(img)

# cv2.imwrite(path + '.letterbox.jpg', 255 * img.transpose((1, 2, 0))[:, :, ::-1]) # save letterbox image
return img, img0

# Get target size
#####

def letterbox(img, new_shape=(640, 640), color=(114, 114, 114), auto=True, scaleFill=False, scaleup=True):
    # Resize image to a 32-pixel-multiple rectangle https://github.com/ultralytics/yolov3/issues/232
    shape = img.shape[:2] # current shape [height, width]
    if isinstance(new_shape, int):
        new_shape = (new_shape, new_shape)

    # Scale ratio (new / old)
    r = min(new_shape[0] / shape[0], new_shape[1] / shape[1])
    if not scaleup: # only scale down, do not scale up (for better test mAP)
        r = min(r, 1.0)

    # Compute padding
    ratio = r, r # width, height ratios
    new_unpad = int(round(shape[1] * r)), int(round(shape[0] * r))
    dw, dh = new_shape[1] - new_unpad[0], new_shape[0] - new_unpad[1] # wh padding
    if auto: # minimum rectangle
        dw, dh = np.mod(dw, 64), np.mod(dh, 64) # wh padding
    elif scaleFill: # stretch
        dw, dh = 0.0, 0.0
        new_unpad = (new_shape[1], new_shape[0])
        ratio = new_shape[1] / shape[1], new_shape[0] / shape[0] # width, height ratios

    dw /= 2 # divide padding into 2 sides
    dh /= 2

```

```

if shape[:::-1] != new_unpad: # resize
    img = cv2.resize(img, new_unpad, interpolation=cv2.INTER_LINEAR)
top, bottom = int(round(dh - 0.1)), int(round(dh + 0.1))
left, right = int(round(dw - 0.1)), int(round(dw + 0.1))
img = cv2.copyMakeBorder(img, top, bottom, left, right, cv2.BORDER_CONSTANT, value=color) # add
border

return img, ratio, (dw, dh)

def print_div(text):

    print(text, '\n')
    print('='*40, '\n')

# Detect func
#####

def detect(save_img=False):

    print_div('INTIL')

    out, source, weights, view_img, save_txt, imgsz = \
        opt.output, opt.source, opt.weights, opt.view_img, opt.save_txt, opt.img_size

    # Initialize
    print_div('GET DEVICE')
    set_logging()
    device = select_device(opt.device)
    half = device.type != 'cpu' # half precision only supported on CUDA

    # Load model
    print_div('LOAD MODEL')
    model = attempt_load(weights, map_location=device) # load FP32 model
    imgsz = check_img_size(imgsz, s=model.stride.max()) # check img_size
    if half:
        model.half() # to FP16

    # Second-stage classifier
    print_div('LOAD MODEL_CLASSIFIER')
    classify = False
    if classify:
        modelc = load_classifier(name='resnet101', n=2) # initialize

```

```

        modelc.load_state_dict(torch.load('weights/resnet101.pt', map_location=device)['model']) # load weights
        modelc.to(device).eval()

# Get names and colors
print_div('SET LABEL COLOR')
names = model.module.names if hasattr(model, 'module') else model.names
colors = [[random.randint(0, 255) for _ in range(3)] for _ in range(len(names))]

# Run inference
#####

print_div("RUN INFERENCE")

img = torch.zeros((1, 3, imgsz, imgsz), device=device) # init img
_ = model(img.half() if half else img) if device.type != 'cpu' else None # run once

video_path = source
cap = cv2.VideoCapture(video_path)

print_div('Start Play VIDEO')
while cap.isOpened():

    ret, frame = cap.read()
    t0 = time.time()

    if not ret:
        print_div('No Frame')
        break

    fps_t1 = time.time()

    img, img0 = img_preprocess(frame) # img: Resize , img0:Orginal
    img = torch.from_numpy(img).to(device)
    img = img.half() if half else img.float() # uint8 to fp16/32
    img /= 255.0 # 0 - 255 to 0.0 - 1.0
    if img.ndimension() == 3:
        img = img.unsqueeze(0)

# Inference

```

```

t1 = time_synchronized()

pred = model(img, augment=opt.augment)[0]

# Apply NMS : 取得每項預測的數值
pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, classes=opt.classes,
agnostic=opt.agnostic_nms)
t2 = time_synchronized()

# Apply Classifier : 取得該數值的 LLabel
if classify:
    pred = apply_classifier(pred, modelc, img, img0)

# Draw Box
for i, det in enumerate(pred):

    s = '%gx%g ' % img.shape[2:] # print string
    gn = torch.tensor(img0.shape)[[1, 0, 1, 0]] # normalization gain whwh
    if det is not None and len(det):
        # Rescale boxes from img_size to im0 size
        det[:, :4] = scale_coords(img.shape[2:], det[:, :4], img0.shape).round()

        # Print results
        for c in det[:, -1].unique():
            n = (det[:, -1] == c).sum() # detections per class
            s += '%g %ss, ' % (n, names[int(c)]) # add to string

        # Write results
        for *xyxy, conf, cls in reversed(det):
            label = '%s %.2f' % (names[int(cls)], conf)
            plot_one_box(xyxy, img0, label=label, color=colors[int(cls)], line_thickness=3)

# Print Results(inference + NMS)
print_div('%sDone. (%.3fs)' % (s, t2 - t1))

# Draw Image
x, y, w, h = (img0.shape[1]//4), 25, (img0.shape[1]//2), 30
cv2.rectangle(img0, (x, 10),(x+w, y+h), (0,0,0), -1)

```

```

        rescale = 0.5

        re_img0 = (int(img0.shape[1]*rescale),int(img0.shape[0]*rescale))

        cv2.putText(img0, '{} | inference: {:.4f}s | fps: {:.4f}'.format(opt.weights[0], t2-t1, 1/(time.time()-t0)),(x+20,
y+20),cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2)

        cv2.imshow('Stream_Detected', cv2.resize(img0, re_img0) )

        key = cv2.waitKey(1)
        if key == ord('q'): break

    # After break
    cap.release()
    cv2.destroyAllWindows()

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default='yolov5s.pt', help='model.pt path(s)')
    parser.add_argument('--source', type=str, default='inference/images', help='source') # file/folder, 0 for webcam
    parser.add_argument('--output', type=str, default='inference/output', help='output folder') # output folder
    parser.add_argument('--img-size', type=int, default=640, help='inference size (pixels)')
    parser.add_argument('--conf-thres', type=float, default=0.4, help='object confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.5, help='IOU threshold for NMS')
    parser.add_argument('--device', default="", help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='display results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --class 0, or --class 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--update', action='store_true', help='update all models')
    opt = parser.parse_args()
    print(opt)

    with torch.no_grad():
        if opt.update: # update all models (to fix SourceChangeWarning)
            for opt.weights in ['yolov5s.pt', 'yolov5m.pt', 'yolov5l.pt', 'yolov5x.pt']:
                detect()
                strip_optimizer(opt.weights)

```

```
else:
```

```
    detect()
```

結語

YOLOv5 縱使不是正統、最快的 YOLO 但是基於 PyTorch 實做的 YOLO 讓我們修改更加的方便，以往要在 DarkNet 上運行現在只要能裝好 PyTorch 基本就可以了，檔案大小也差非常多，邊緣裝置的負擔也不會太大！所以支持大家可以去體驗看看 YOLOv5 的方便性、輕便性，在下一篇文章中我會教大家如何使用 YOLOv5 來訓練自己的數據！

相關文章

在 Jetson Nano (TX1/TX2)上使用 Anaconda 与 PyTorch 1.1.0

<https://zhuanlan.zhihu.com/p/64868319>

YOLOv5 github

<https://github.com/ultralytics/yolov5>