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 | | |
|  | | | |
| **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：**    **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。    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。    這是官方提供的教學[PyTorch for Jetson - version 1.6.0 now available](https://forums.developer.nvidia.com/t/pytorch-for-jetson-version-1-6-0-now-available/72048)，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 |     **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：**    **接著要找到原生的OpenCV位置：**   |  | | --- | | $ sudo find / -name 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：    **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 |     **確認是否安裝成功：**    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   |  |  | | --- | --- | | --source | 圖片、影片目錄或是 0 開啟攝影機 | | --iou\_thred | 信心指數的閥值，雖然低一些可框出越多東西但準確度就不敢保證 | | --weights | 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](https://drive.google.com/drive/folders/1Drs_Aiu7xx6S-ix95f9kNsA6ueKRpN2J)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：**  **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，第一步是要取得到即時路況影像，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環境：   |  | | --- | | $ 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%E5%B8%82%E6%94%BF%E5%BA%9CTaipeiCityGovernment' -o test.mp4 -s 10 |     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 |     結果如下：    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：    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：    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LAbel  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> | | | |