Train YOLOv2 to detect customized objects

Outline

1. 安裝Darkflow
2. 使用prebuild YOLO模型
3. 訓練新的YOLO模型
   - 準備訓練資料 (圖片 與xml檔): 利用LabelImg 工具框選物件及標注label
   - 修改YOLO模型參數,如class數量, filter數目
   - 開始訓練,相關命令列參數: model , load , gpu,…
   - 測試模型, 相關命令列參數: model , load
   - 儲存模型(weight .pb & metafile .meta), 相關命令列參數: --savepb
   - 載入儲存的模型(.pb & .meta)
- 利用python 應用測試訓練的YOLO模型

Darkflow安裝

將terminal移至你所要存的資料夾裡，輸入
git clone https://github.com/thtrieu/darkflow

```
(yolo-test) D:\nn>git clone https://github.com/thtrieu/darkflow
Cloning into 'darkflow'...
remote: Enumerating objects: 2706, done.
remote: Total 2706 (delta 0), reused 0 (delta 0), pack-reused 2706
Receiving objects: 100% (2706/2706), 18.74 MiB | 481.00 KiB/s, done.
Resolving deltas: 100% (1781/1781), done.
Checking connectivity... done.
```

Darkflow安裝

將terminal移至darkflow資料夾裡，輸入
pip install -e .

```
(yolo-test) D:\nn\darkflow>pip install -e .
Obtaining file:///D:/nn/darkflow
Installing collected packages: darkflow
  Running setup.py develop for darkflow
Successfully installed darkflow
```

執行時可能需要的套件:
pip install opencv-contrib-python
pip install Cython

使用prebuild YOLO模型

將事先訓練好的weight(官方訓練的)放在darkflow下的bin目錄裡

並在terminal中輸入:

python flow --model cfg/yolo.cfg --load bin/yolov2.weights --imgdir sample_img/

```
(base) C:\Users\f\Desktop\y\darkflow>python flow --model cfg/yolo.cfg --load bin
/yolov2.weights --imgdir sample_img/

C:\Users\f\Desktop\y\darkflow\darkflow\dark\darknet.py:54: UserWarning: ./cfg/yo
lov2.cfg not found, use cfg/yolo.cfg instead
  cfg_path, FLAGS.model))
Parsing cfg/yolo.cfg
Loading bin/yolov2.weights ...
Successfully identified 203934260 bytes
Finished in 0.043805837631225586s
Model has a coco model name, loading coco labels.

Building net ...
Source | Train? | Layer description                | Output size
-------+--------+----------------------------------+---------------
       |        | input                            | (?, 608, 608, 3)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 608, 608, 32)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 304, 304, 32)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 304, 304, 64)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 152, 152, 64)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 152, 152, 128)
```

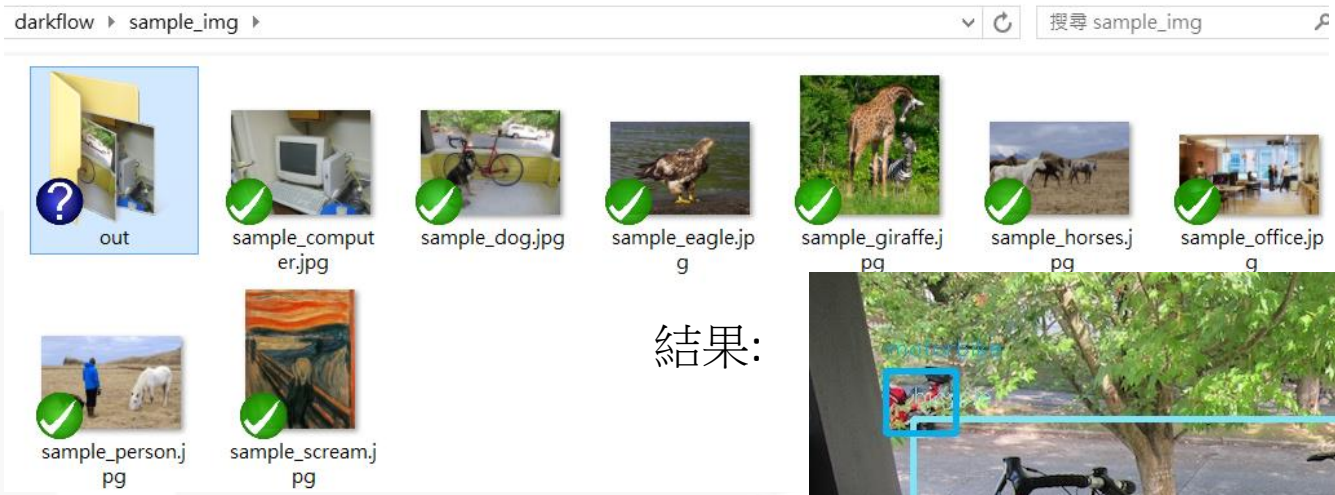
Weight載點:

https://pjreddie.com/media/files/yolov2.weights

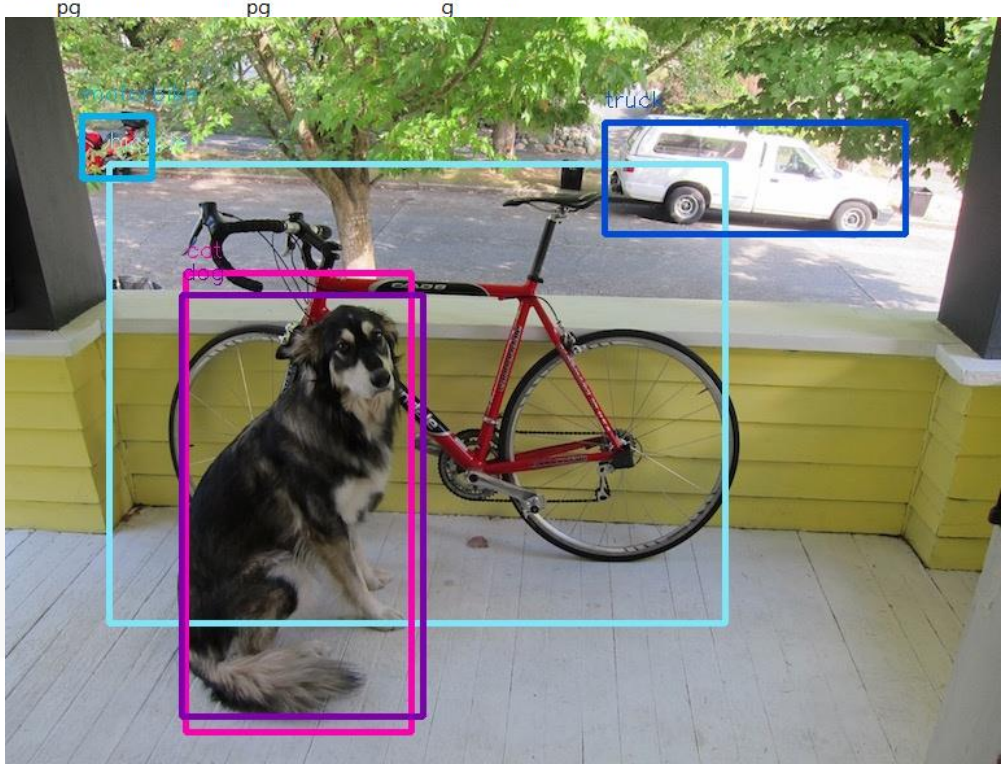
https://drive.google.com/drive/folders/0B1tW_VtY7onidEwyQ2FtQVplWEU

使用prebuild YOLO模型

sample_img資料夾中便會多一個out的資料夾，裡面有著所有照片經過yolo檢測的結果

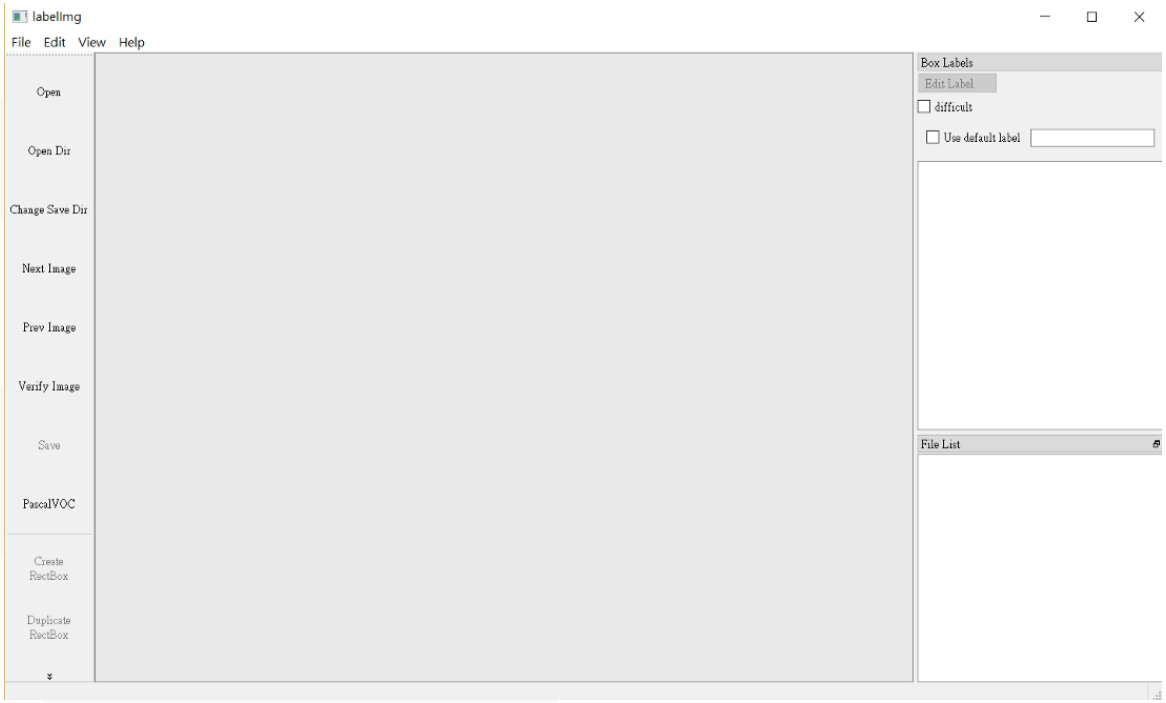


結果:



Labelimg安裝與執行

在terminal中分別輸入:
1.git clone https://github.com/tzutalin/labelImg
2.cd lableimg
3.python lableimg.py



執行時可能需要的套件:
conda install pyqt=5
pip install resources
pip install requests
pip install staty
pip install lxml

準備訓練資料 (圖片 與xml檔): 利用LabelImg 工具框選物件及標注label

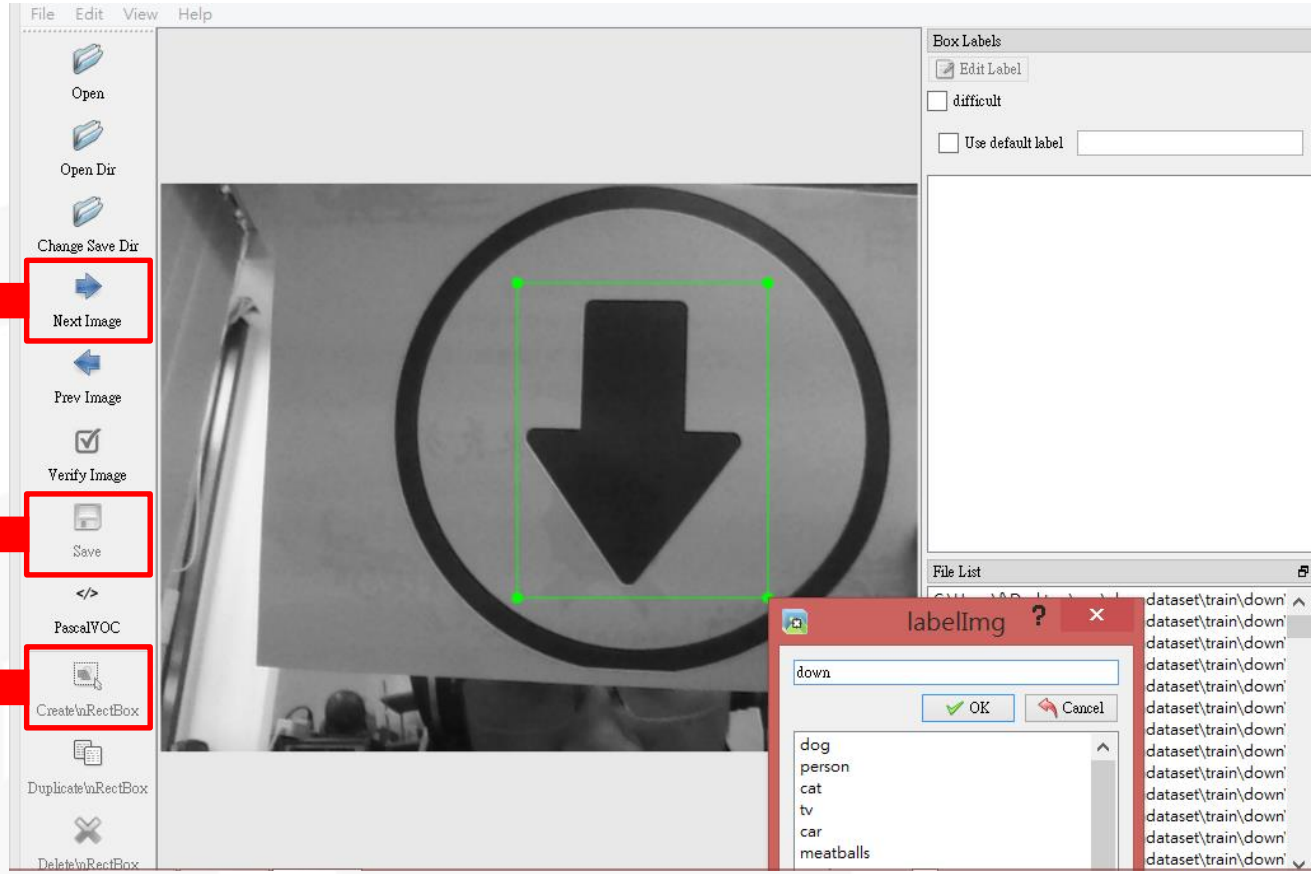
LabelImg使用(Step 1)

LabelImg使用(Step 2)

框選並標籤下一張照片

將框選物件並標籤的照片資料存檔

框選物件並標籤



人工進行物件框選與標籤的注意事項

在作label框選時，請考慮是否有足夠的紋理特徵
Ex:柑橘框選

修改YOLO模型參數

設定Filter與class數量

修改darkflow/cfg資料夾中cfg檔案

在darkflow/cfg/資料夾中有不同的yolo cfg檔，選擇適合自己的，複製出一份xxx.cfg，並修改複製檔案中的2個部份

```
110 [convolutional]
111 size=1
112 stride=1
113 pad=1
114 filters=425
115 activation=linear
116
117 [region]
118 anchors = 0.738768,0.874946,  2.42204,2.65704,  4.30971,7.04493,  10.246,4.59428,
     12.6868,11.8741
119 bias_match=1
120 classes=80
121 coords=4
122 num=5
123 softmax=1
124 jitter=.2
125 rescore=1
```

將filter數量改成5*(幾種label+5)
Ex: 我有4種label
5*(4+5)=45
就要將425改成45

改成label數量
Ex: 我有4種label
就要將80改成4

修改YOLO模型參數

設定Labels

修改darkflow資料夾中cfg跟labels.txt檔案

打開labels.txt，把裡面的標籤改成你要的Label名稱

開始訓練

訓練Model

在darkflow資料夾中輸入:
python flow --model [model.cfg] --train [--load [weights]] --dataset [image path] --annotation [annotation path] [--gpu 1.0]

```
Finished in 8.190147638320923s

Enter training ...

cfg/tiny-yolo_baby_4.cfg parsing test/training_baby/Robert_Candy/annotations
Parsing for ['Candy', 'Robert']
[==================>]100%  test_3 (91).xml
Statistics:
Candy: 211
Robert: 168
Dataset size: 250
Dataset of 250 instance(s)
Training statistics:
        Learning rate : 1e-05
        Batch size    : 4
        Epoch number  : 1000
        Backup every  : 2000
step 1 - loss 106.58182525634766 - moving ave loss 106.58182525634767
step 2 - loss 106.62338256835938 - moving ave loss 106.58598098754884
```

[model.cfg]:前面修改的**cfg**檔名
[image path]:儲存照片的資料夾名
[annotation path]:儲存照片標籤完的資料的資料夾名
[--gpu 1.0]:是否使用**gpu**訓練
[--load [weights]]:是否使用**pretrained weights** 來進行訓練

開始訓練

訓練Model

完成前一步驟後，model會開始訓練，等他跑完，model也就訓練完成了!

```
step 7480 - loss 0.5787131190299988 - moving ave loss 0.6972996513014321
step 7481 - loss 0.7103055715560913 - moving ave loss 0.6986002433268981
step 7482 - loss 0.694850504398346 - moving ave loss 0.6982252694340428
step 7483 - loss 0.6602804660797119 - moving ave loss 0.6944307890986098
Finish 69 epoch(es)
step 7484 - loss 0.8060882091522217 - moving ave loss 0.7055965311039709
step 7485 - loss 0.7279226779937744 - moving ave loss 0.7078291457929513
step 7486 - loss 0.7303983569145203 - moving ave loss 0.7100860669051082
step 7487 - loss 0.6814758777618408 - moving ave loss 0.7072250479907816
step 7488 - loss 0.6795366406440735 - moving ave loss 0.7044562072561108
step 7489 - loss 0.5975558757781982 - moving ave loss 0.6937661741083195
step 7490 - loss 0.5565756559371948 - moving ave loss 0.6800471222912071
Finish 70 epoch(es)
step 7491 - loss 0.7097848057746887 - moving ave loss 0.6830208906395553
step 7492 - loss 0.5260033011436462 - moving ave loss 0.6673191316899644
step 7493 - loss 0.8163402676582336 - moving ave loss 0.6822212452867913
step 7494 - loss 0.6774263381958008 - moving ave loss 0.6817417545776923
step 7495 - loss 0.8473349809646606 - moving ave loss 0.6983010772163891
step 7496 - loss 0.5025731921195984 - moving ave loss 0.6787282887067101
step 7497 - loss 0.7954717874526978 - moving ave loss 0.6904026385813089
Finish 71 epoch(es)
step 7498 - loss 0.65382319688797 - moving ave loss 0.686744694411975
step 7499 - loss 0.5413708686828613 - moving ave loss 0.6722073118390637
step 7500 - loss 0.7029291391372681 - moving ave loss 0.6752794945688841
Checkpoint at step 7500
step 7501 - loss 0.6322606801986694 - moving ave loss 0.6709776131318627
step 7502 - loss 0.6782676577568054 - moving ave loss 0.671706617594357
step 7503 - loss 0.6801575422286987 - moving ave loss 0.6725517100577912
step 7504 - loss 0.7598941326141357 - moving ave loss 0.6812859523134257
Finish 72 epoch(es)
step 7505 - loss 0.5269620418548584 - moving ave loss 0.665853561267569
step 7506 - loss 0.5228486657142639 - moving ave loss 0.6515530717122385
```

訓練結果

Yolo training default的epoch數量是1000，然後data每經過2000個data會把目前的參數存成ckpt檔在ckpt的資料夾

the .meta file is where the weights are stored.
.meta, .index and .data are files related to TensorFlow.

測試模型

在**darkflow**資料夾中輸入:

python flow --model **[model.cfg]** --load **[weights]** --imgdir **[images]** **[--gpu 1.0]** **[--json]**

```
(base) C:\Users\f\Desktop\y\darkflow>python flow --model cfg/yolo.cfg --load bin
/yolov2.weights --imgdir sample_img/
```

[model.cfg]:欲使用的**model**
[weights]:訓練出的**weight**檔，若想使用訓練出來的**ckpt**檔，可輸入**-1**便會自動讀取最新的**ckpt**檔
[images]:照片的資料夾檔名
[--gpu 1.0]:是否使用**gpu**
[--json]:是否將結果存成**json**檔

待上面的步驟跑完後，**[images]**中會多一個**out**的資料夾，結果便會存在那

儲存模型(weight .pb & metafile .meta)

在darkflow資料夾中輸入:

Saving the lastest checkpoint to protobuf file
python flow --model [model.cfg] --load -1 --savepb

Saving graph and weights to protobuf file
python flow --model [model.cfg] --load [yolo.weights] --savepb

上面的指令會產生一個 .pb file與一個 .meta file，若要在其他地方使用此模型便只要複製這2份檔案即可

載入儲存的模型(.pb & .meta)

在darkflow資料夾中輸入:

python flow --pbLoad [model.pb] --metaLoad [yolo.meta] --imgdir [images] [--gpu 1.0]

[model.pb]: model的pb檔
[yolo.meta] : weight 的meta檔
[images]: 執行Yolo偵測的照片的資料夾
[--gpu 1.0]: 是否使用gpu

利用python 應用測試訓練的YOLO模型

Yolo在python中的使用(.cfg & ckpt)

```
1  from darkflow.net.build import TFNet
2  import cv2
3  import numpy as np
4
5
6  options = {"model": "cfg/yolo_arrow.cfg",
7             "threshold": 0,
8             "load": -1,
9             }
10
11 tfnet = TFNet(options)
12 tfnet.load from ckpt()
```

訓練的cfg檔

根據多少的confidence來判斷是否是所欲的偵測物(0~1) ex:0.1代表10% confidence

-1 會讀取最新的weight檔

由於我們的weight檔是ckpt檔

```
imgcv = cv2.imread("images/frame262.jpg")
result = tfnet.return_predict(imgcv)
print(result)
```

Result會回傳偵測到的物體的座標，confidence，label

```
[{'label': 'down', 'topleft': {'y': 204, 'x': 316}, 'confidence': 0.39508244, 'bottomright': {'y': 396, 'x': 455}}]
```

利用python 應用測試訓練的YOLO模型

Yolo在python中的使用(.pb & .meta)

```python
from darkflow.net.build import TFNet
import cv2
import numpy as np


options = {"pbLoad": "cfg/yolo_arrow.pb",
           "threshold": 0,
           "metaLoad": "yolo_arrow.meta",
           }

tfnet = TFNet(options)
imgcv = cv2.imread("images/frame262.jpg")
result = tfnet.return_predict(imgcv)
print(result)
```

Model的.pb檔

根據多少的confidence來判斷是否是所欲的偵測物(0~1)
ex:0.1代表10%
confidence

讀取weight檔

Result會回傳偵測到的物體的座標，confidence，label

```
[{'label': 'down', 'topleft': {'y': 204, 'x': 316}, 'confidence': 0.39508244, 'bottomright': {'y': 396, 'x': 455}}]
```

訓練結果