第 30 組：人流計算

組內成員：

0716059 巫奇翰、0716301 劉育源、0716310 翁瑞澤

[完整簡介影片]

==========

大綱

- Github repo link
- 動機與介紹
- 相關研究
- 方法
- 實驗
- 結論
- Reference

==========

Github repo link：

https://github.com/liuyy3364/Introduntion-to-Artificial-Intelligence-Final-Project

動機與介紹：

動機：

最近疫情頻傳，為了防止疫情擴散，減少人與人之間的接觸是一大重點，但有些場所又不適合禁止人員進入，因此人流管控的重要性就被突顯出來，而在這種情況下，採用人工計算顯然不是明智的方法，所以我們決定做一個可以進行人流管控的系統框架，讓其他使用者可以透過這個框架簡單的做出符合他們要求的人流管控系統。

介紹：

我們的框架是利用結合 Object Detection 與 Multiple Object Tracking (MOT)功能，做出可以偵測並追蹤人員的 Detection-based Tracking 框架，並提供簡單的出入計算與 web server，讓使用者可以方便根據其需求更改系統。

相關研究：

要達成我們的目標，有兩個重點，一是要可以偵測出人來，二是可以持續追蹤被偵測出來的人，這兩點分別可以找到相關研究：Object Detection 與 Multiple Object Tracking (MOT)。

Object Detection：這裡探討兩種方式：
Haar-like feature 與 YOLOv5。
Haar-like feature 的部分是四種長方形的特徵：
two-rectangle feature type (horizontal/vertical)
three-rectangle feature type
four-rectangle feature type

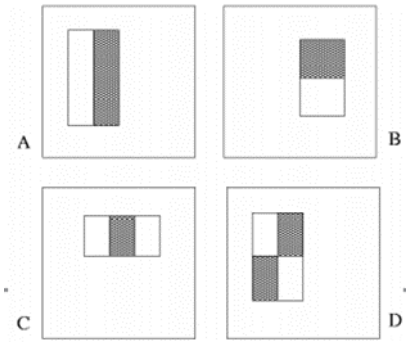

Figure 1. Haar-like feature

而 YOLO 簡單的概念為：

- Divide an image into S x S grids.
- Each grid has B bounding boxes.
- For each bounding box of a grid, predict if it has an object and the best position & size.
  - 5 information: x, y of the center, w, h, confidence
- For each grid, predict which object it has (C class probabilities) (only one object).

Figure 2. YOLO_intro

Multiple Object Tracking (MOT)：這裡介紹三種方式：

SORT、DeepSORT、JDE。

SORT 由 Kalman Filter, Hungarian Algorithm 組成，Kalman Filter 可以大概預測物體下個 frame 的位置，而 Hungarian Algorithm 則是一個任務分配的組合最佳化演算法。

而 DeepSORT 就是 SORT 再加入了外觀的訊息來匹配前後 frame 的 object。

最後 JDE 就是將 object detect 和 appearance embedding 融合在同一個網路裡一起訓練，直接算出 detection location, class, embedding feature。

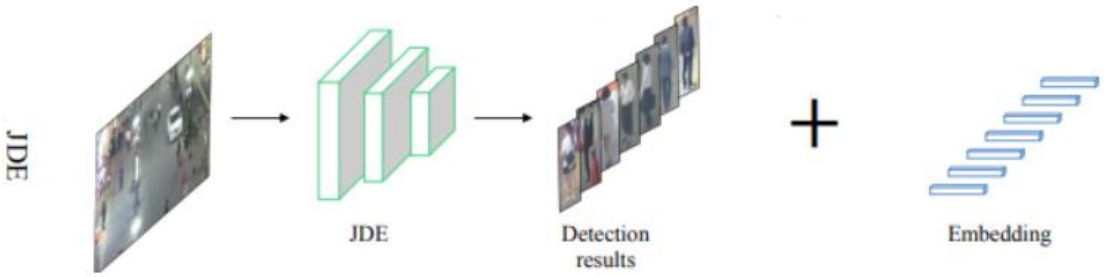

Figure 3. JDE intro

方法：

在主要框架中我們實作了四個功能，分別是 Detect、Track、Display、Optimize。

以下是我們的 framework 架構圖說明

Detector：輸出 bounding boxes 給 Tracker

Tracker：Assign ID 給每個 bounding box

Displayer：呈現 MOT 的結果

Optimizer：最佳化 Detector 和 Tracker model

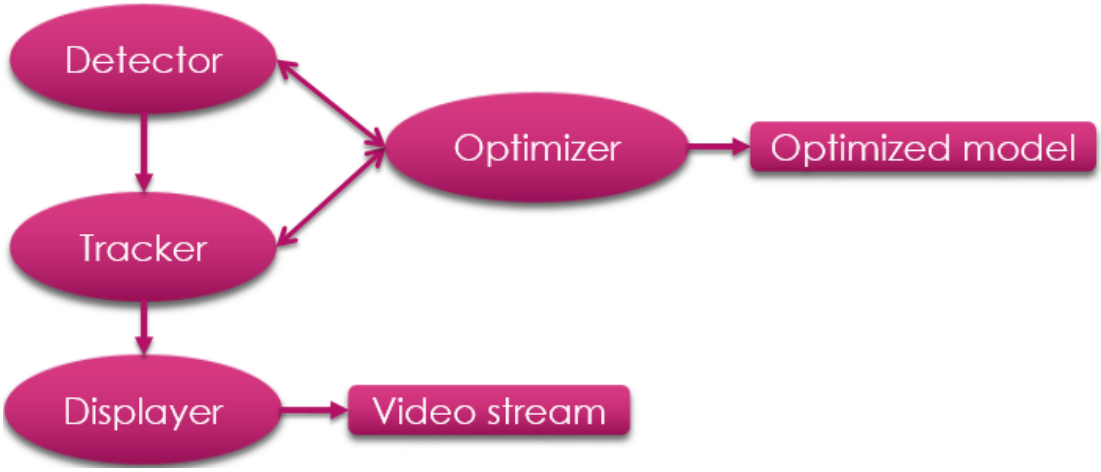

Figure 4. Architecture Graph of Our Framework

做完 framework 後，我們利用它做一個人流計算器。

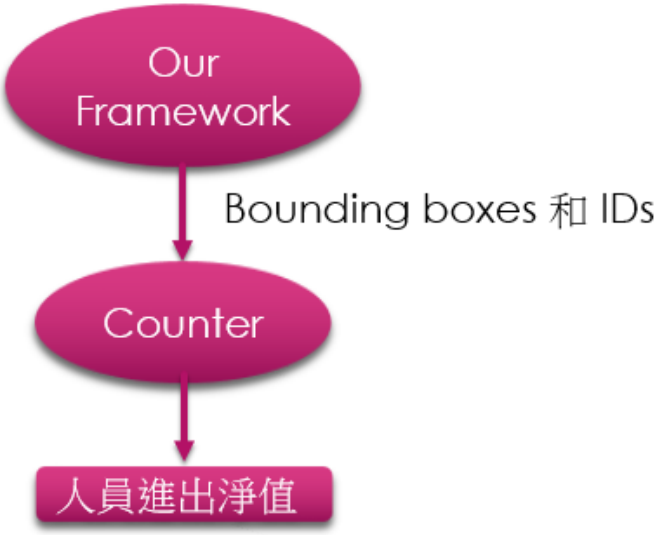

Figure 4. Architecture Graph of Our Person Counter

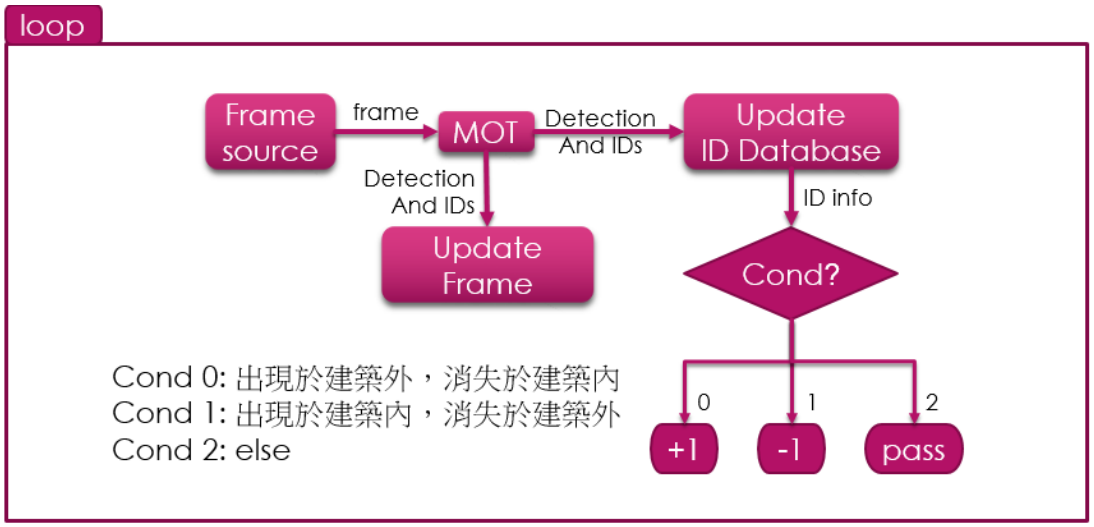

Figure 5. Flow Chart of counter

我們的 framework 位於 Figure 6. 的 MOT 處，在迴圈中，一開始由 Frame Source 收到新的 frame 後輸出 Detection 的 bounding boxes 和 Tracking 的 IDs，再用這些輸出更新 ID database，在看資料庫中有沒有符合圖中三種情況，若是 Cond 0 或 Cond 1，則人員進出淨值變動，並將此 ID 移出 ID database 中。

實驗部件：

Detect 部份我們採用了上述 Haar-like feature 與 YOLOv5 兩種方法，Track 的部分使用了 SORT，Display 的部分採用了 OpenCV 和 Flask，Optimize 的部分採用了 Netadapt，而 Dataset 採用 Oxford Town Centre。

在實驗中我們分別測試兩種 Detector：用 Haar-like feature 與用 YOLOv5，兩者在人體偵測的表現差異如下：

Figure 6. Detector Comparision

可以看出 YOLOv5 的表現比 Haar cascade Classifier 好很多，雖然 YOLOv5 把左上的幾個假人也框選出來，但他們的 Confidence 普遍偏低，能輕易地篩選掉。

NetAdapt:

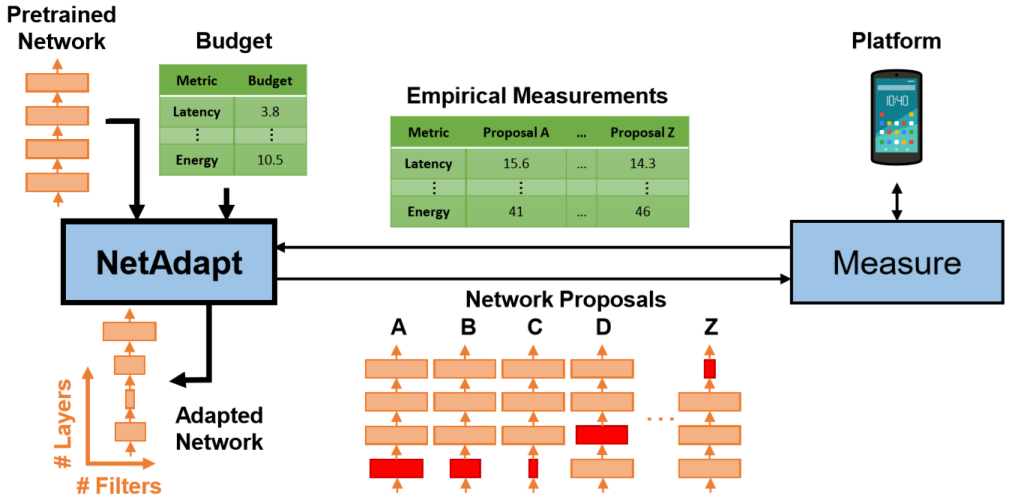

Figure 8. Architecture Graph of NetAdapt

他是一個 pytorch model 的 auto-pruner framework，輸入為一個 pytorch pre-trained model 與其架構，以及關於資源的預算及種類，若種類是 Latency 則還需要一個 latency lookup table，此 LUT 是由直接量測各種形狀 layer inference 於目標裝置得到，用於加速提出 network proposal 時資源的計算，其輸出為一個所需資源在輸入預算內的 model。

NetAdapt 內部的演算法是透過多次 iteration，每次 iteration 都會提出每個 layer prune 到符合應下降的所需資源比例的 proposal，在 finetune 每個 proposal 後，選出其中 Accuracy 最高的作為下一次 iteration 的輸入 model，最後停止於所需資源小於資源預算時。

Iteration	Accuracy	Resource	Block	Source Model	
0	99.104	0.017823	None	models/alexnet/model.pth.tar	64 192 384 256 256 4096 4096 10
1	99.162	0.017311	5	models/alexnet/prune-by-later	64 192 384 256 256 3632 4096 10
2	99.194	0.016873	6	models/alexnet/prune-by-later	64 192 384 256 256 3632 3624 10
3	99.166	0.01646	6	models/alexnet/prune-by-later	64 192 384 256 256 3632 2408 10
20	97.084	0.010155	1	models/alexnet/prune-by-later	56 128 344 192 128 1784 1792 10
21	96.976	0.009825	2	models/alexnet/prune-by-later	56 128 320 192 128 1784 1792 10

Figure 9. Output Model Log of NetAdapt

它的缺點是非常慢，不好 demo，上圖之前 prune Alexnet 時的輸出，可以看到每次 iteration 只有一個 layer (see Block column)有變動，經過越多 iteration，所需的 resource 越小，同時 accuracy 卻沒下降很多。

實驗結果：

Videos: 連結

實驗一：

Figure 10. v1 yolov5

上圖為第一個測試影片的示意截圖，我們在第一個影片中設定從右上區域走進、左下區域走出為進入(+1)；從左下區域走進、右上區域走出為離開 (-1)，看完影片會發現左上角出現的結果大多數時間都在 0 跟 1 之間跳動，這是因為原始影片的人流趨勢算是很平均的，大多數情況下有人進入就會有人離開。

實驗二：

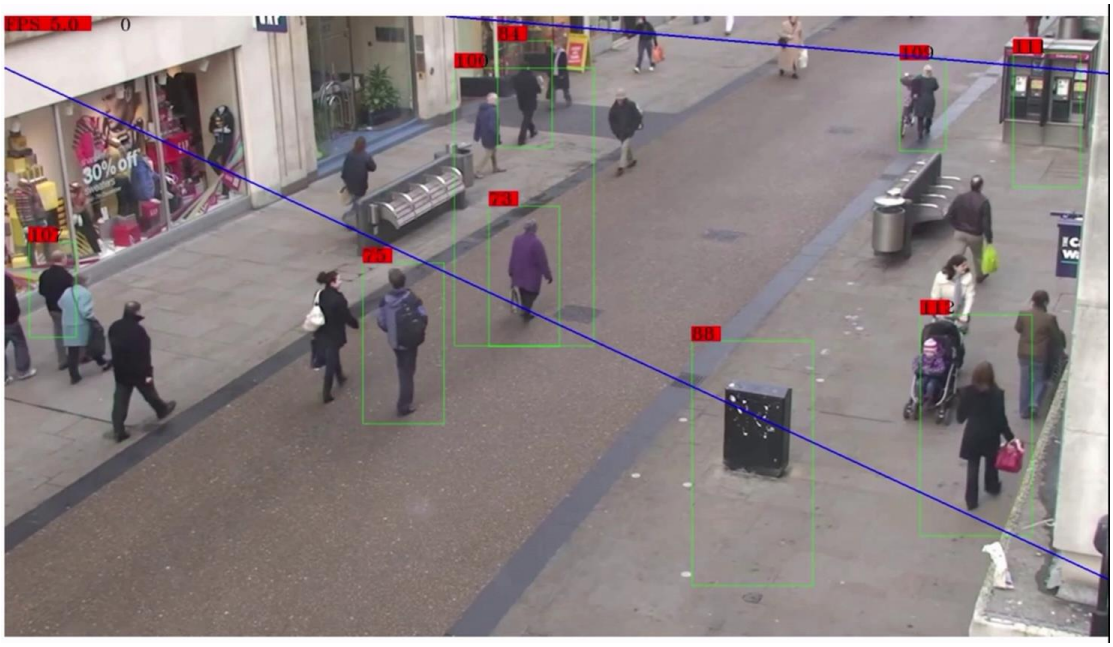

Figure 11. v2 haar

上圖為第二個測試影片的示意截圖，看完影片會發現左上角出現的結果大多數時間都不會動，原因是我們可以看到用 Haar feature 做的 Detector 框出來的東西要嘛不對要嘛沒框到，框對的只占少部分，這導致了 Tracker 不能好好運作，所以 counter 也就沒辦法正確執行。

實驗三：

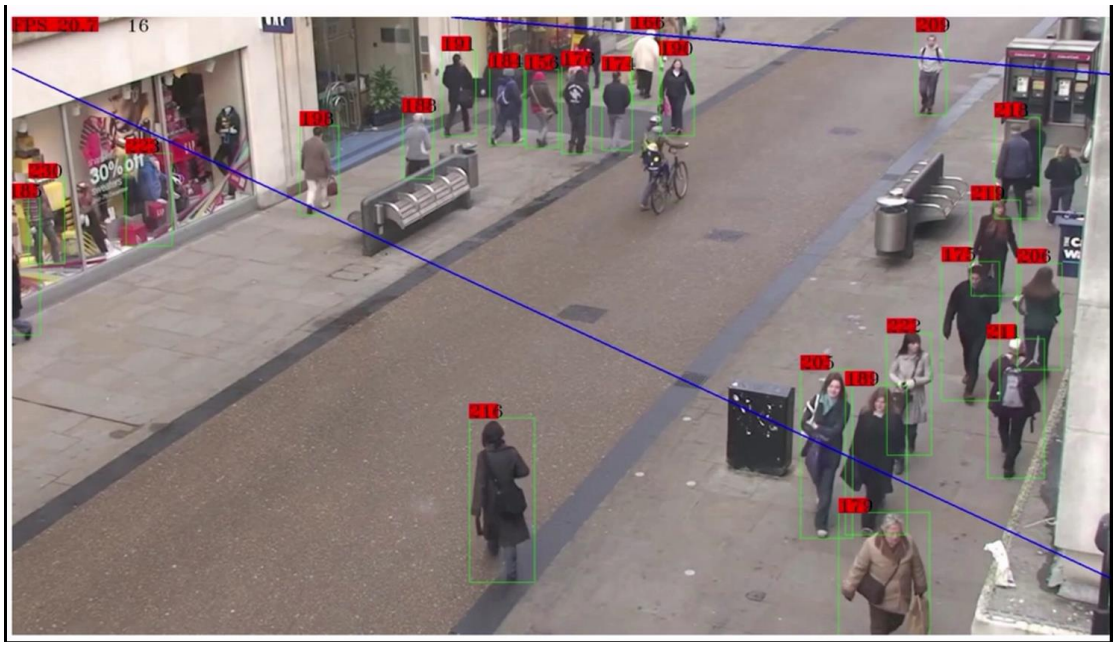

Figure 127. v2 yolov5 t = 1:07

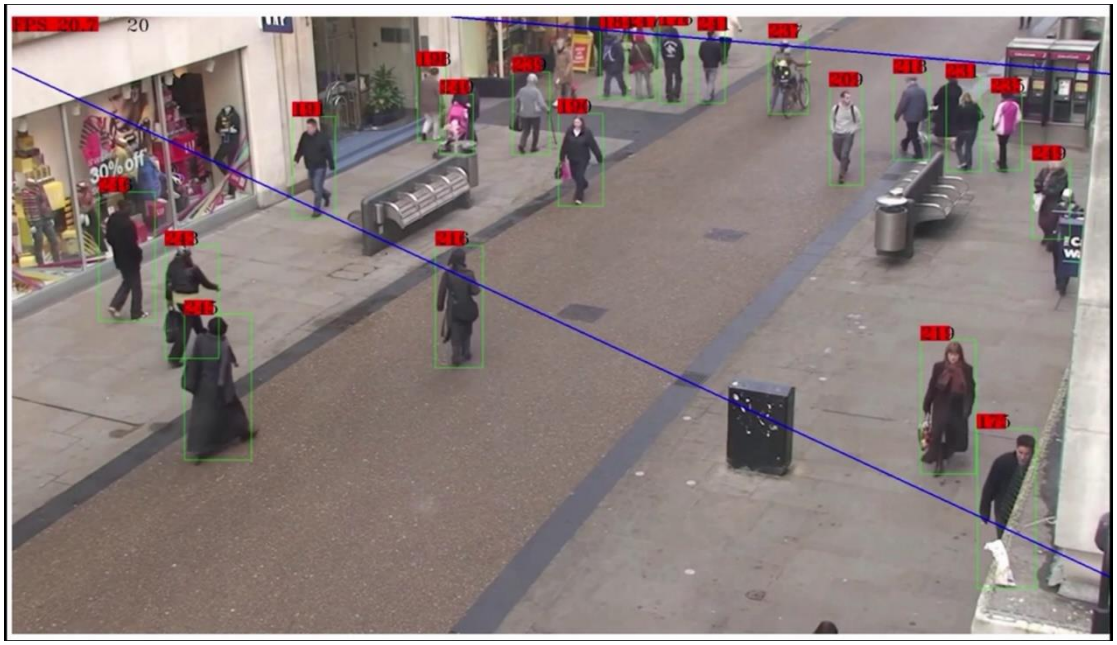

Figure 13. v2 yolov5 t = 1:12

上面兩張為第三個測試影片的示意截圖，相較第一部影片，我們拿掉了離開的計算，只計算進入人數，可以明顯的看到左上角的人數不再是介於 0 跟 1 之間，且比較 Figure12 與 Figure13 可以發現 t = 1:07 時人數為 16，而 t = 1:12 時人數為 20，剛好就是 Figure12 右下角 4 個人進入之後。

結論：

Detecting 跟 Tracking 根據演算法的不同，可以是獨立的，也可以是合作的。Single object tracker 速度比 detecting 快，可是需要處理追蹤物件消失的問題； Detection based tracking 則需要處理 ID switching 的問題，同時受限於龐大的 model 與計算量，兩者各有優缺。不過現在也有研究者嘗試將 single object tracker 使用在 MOT 問題上，希望能夠引起新的研究方向。

References：

Haar-like feature：
https://www.researchgate.net/profile/Michael-Jones-66/publication/3940582_Rapid_Object_Detection_using_a_Boosted_Cascade_of_Simple_Features/links/0f31753b419c639337000000/Rapid-Object-Detection-using-a-Boosted-Cascade-of-Simple-Features.pdf
YOLOv5： https://github.com/ultralytics/yolov5
SORT： https://github.com/abewley/sort
Netadapt：https://github.com/denru01/netadapt

使用到的課內知識：

Haar-like feature: OpenCV Haar Cascade Classifier
CNN, DNN: YOLOv5