# 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**](https://drive.google.com/file/d/1ruZbwQpqmyh1VfMgaAaDVPoa49bqdHZi/view?usp=sharing)

=====================================================================

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：

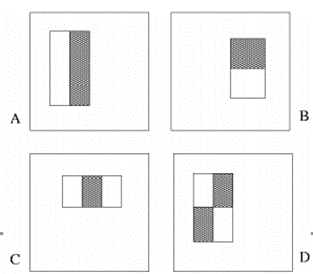
最近疫情頻傳，為了防止疫情擴散，減少人與人之間的接觸是一大重點，但有些場所又不適合禁止人員進入，因此人流管控的重要性就被突顯出來，而在這種情況下，採用人工計算顯然不是明智的方法，所以我們決定做一個可以進行人流管控的系統框架，讓其他使用者可以透過這個框架簡單的做出符合他們要求的人流管控系統。

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：

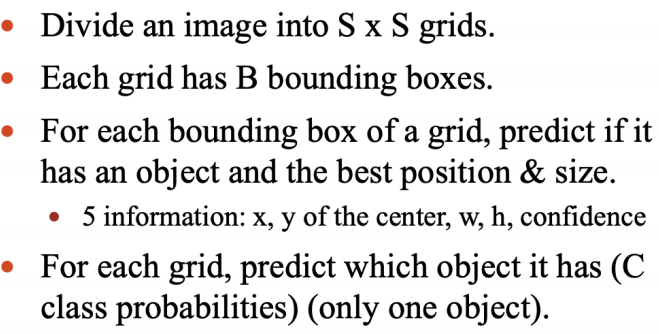
two-rectangle feature type (horizontal/vertical)

three-rectangle feature type

four-rectangle feature type

**Figure 1. Haar-like feature**

而YOLO簡單的概念為：



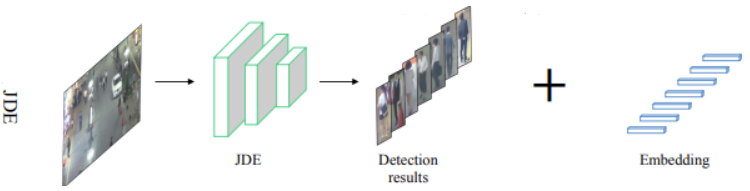
**Figure 2. YOLO\_intro**

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。

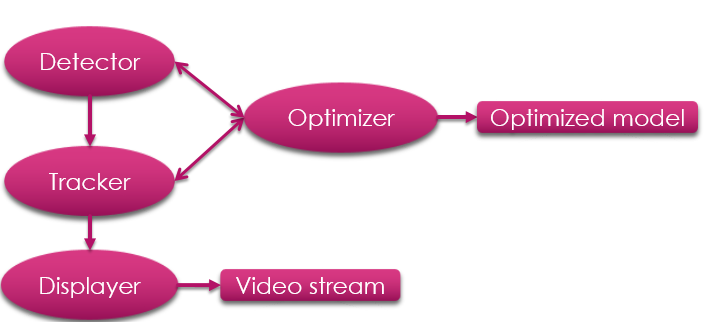
以下是我們的framework架構圖說明

Detector：輸出bounding boxes給Tracker

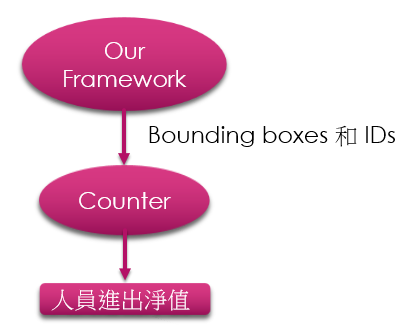
Tracker：Assign ID 給每個bounding box

Displayer：呈現MOT的結果

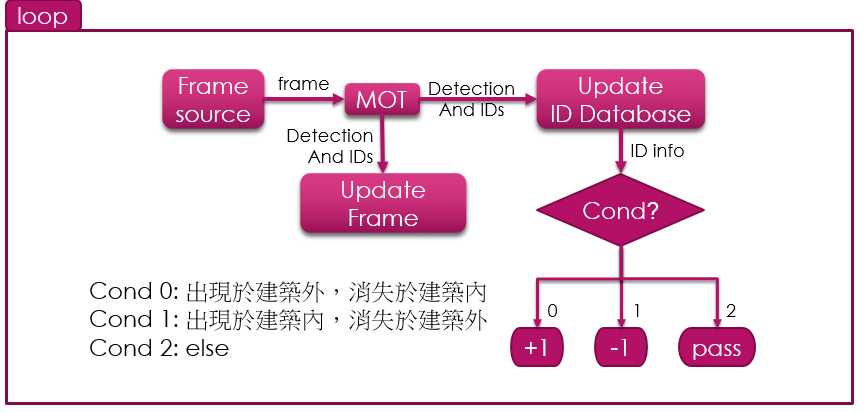
Optimizer：最佳化Detector和Tracker model



**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。

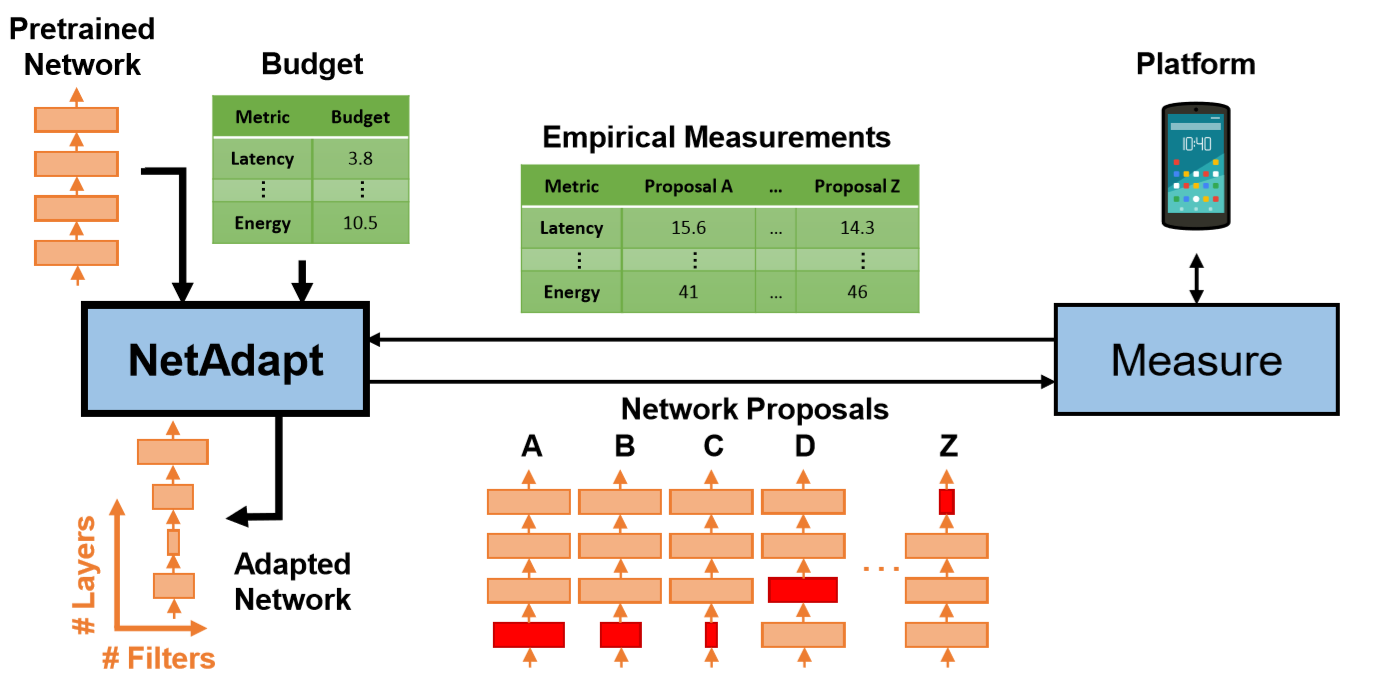
　　在實驗中我們分別測試兩種Detector：用Haar-like feature與用YOLOv5，兩者在人體偵測的表現差異如下：



**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。

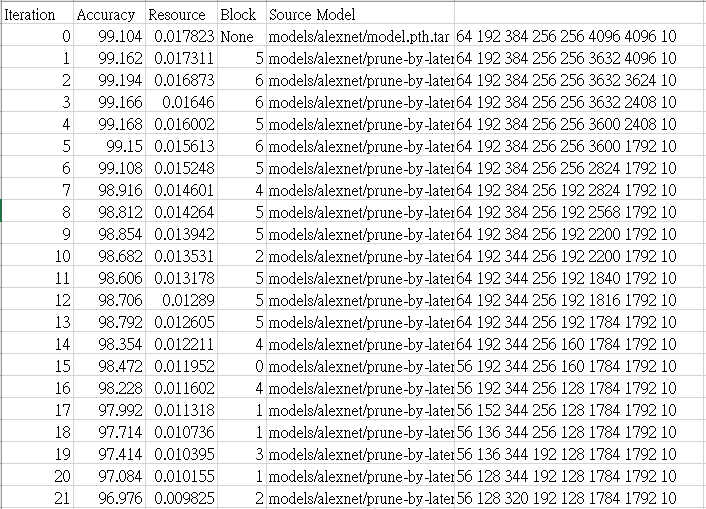
NetAdapt:

****

**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。





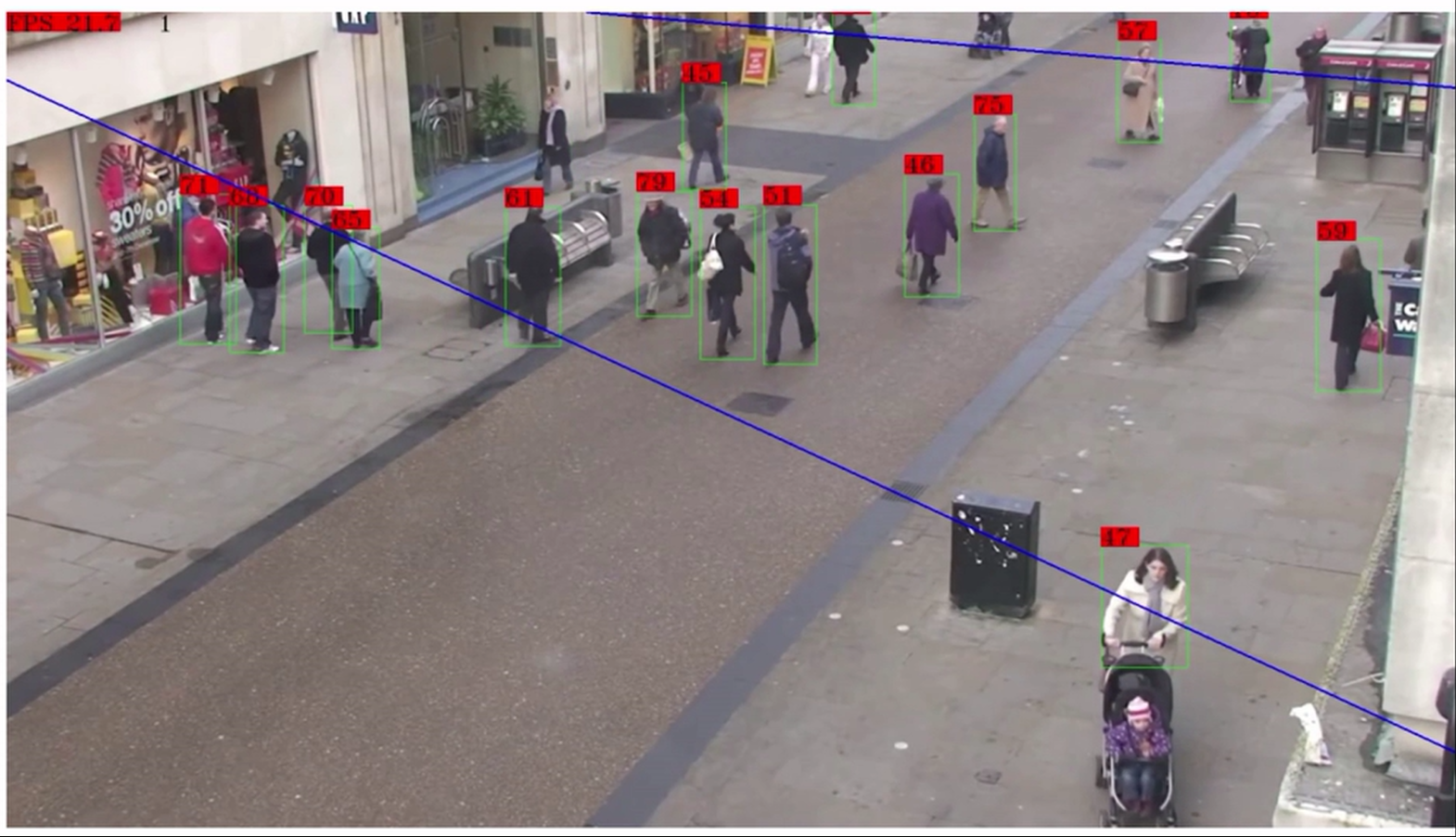
**Figure 9. Output Model Log of NetAdapt**

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](https://drive.google.com/drive/folders/1dYwdrwy5BwoVeThIv9_OnUlmwFYN_3qw?usp=sharing)

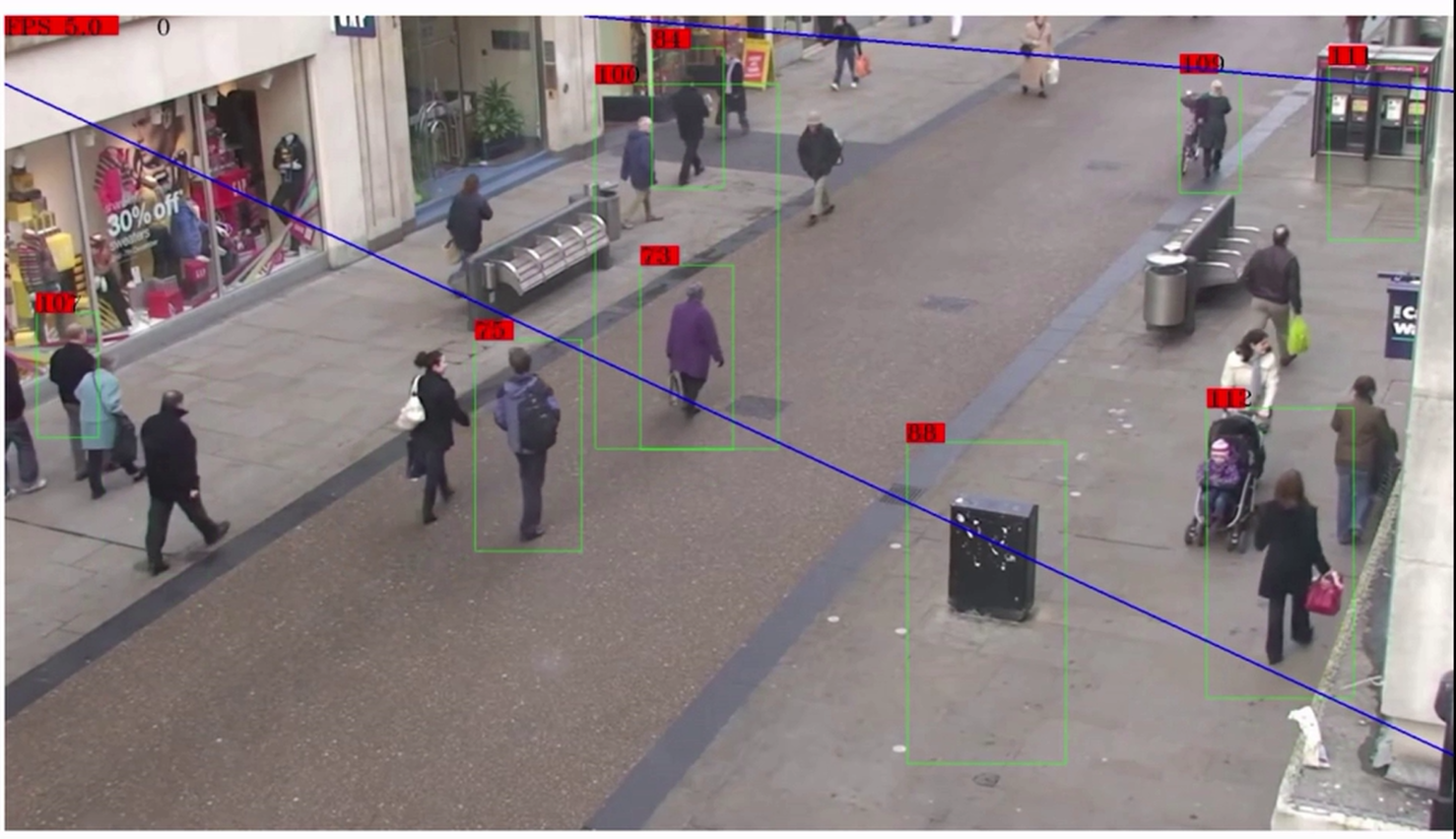
實驗一：



**Figure 10.** [**v1 yolov5**](https://drive.google.com/file/d/1jo7S7ztgSa6lxH9sdkituwqyij-FB_YQ/view?usp=sharing)

　　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。

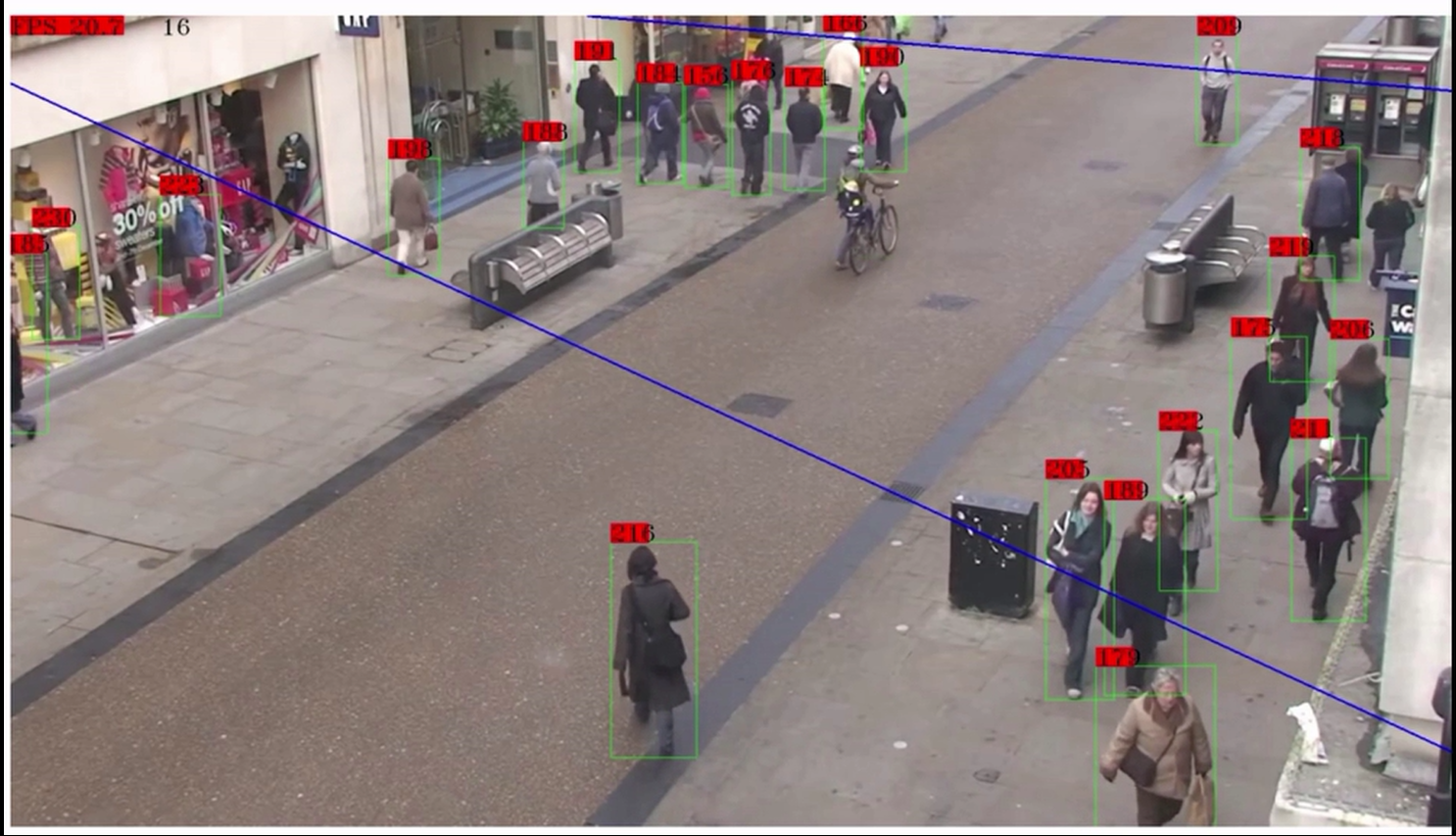
實驗二：

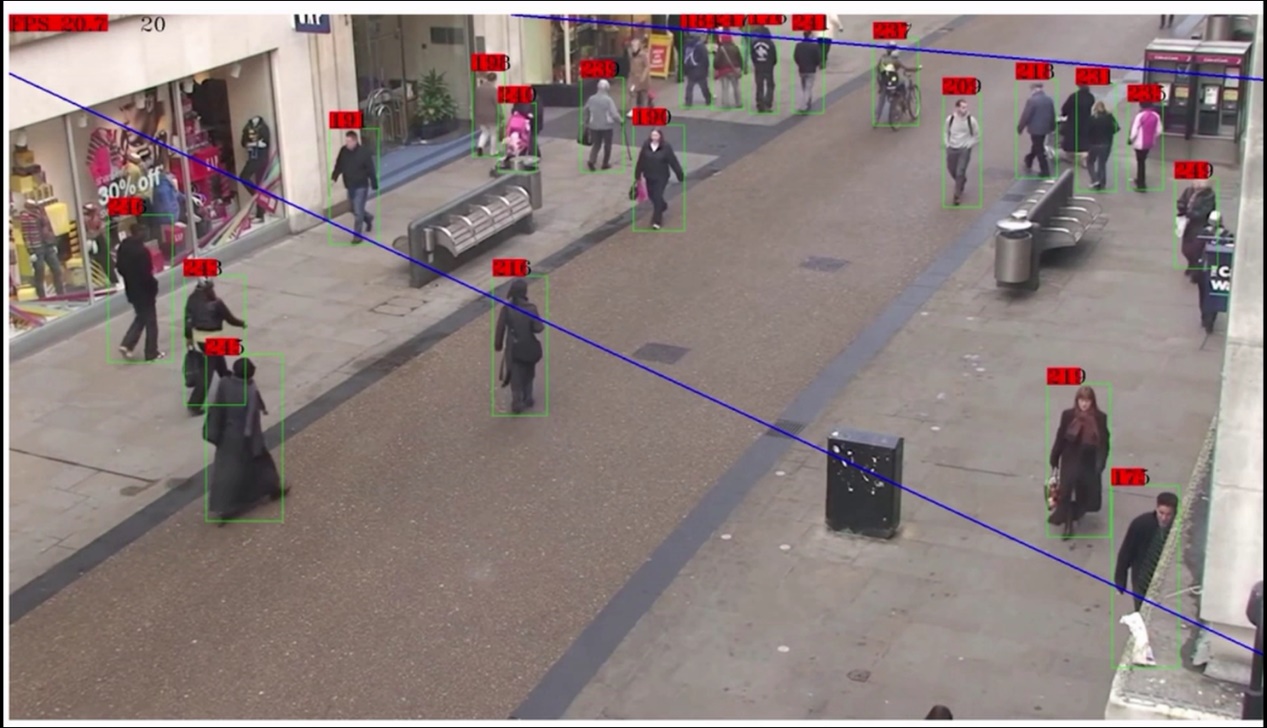


**Figure 11.** [**v2 haar**](https://drive.google.com/file/d/1kuU_BWej5WEyILbPzJ7TZ4wsp8ol31VD/view?usp=sharing)

　　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**](https://drive.google.com/file/d/1XU6IWtmS9EtDVjyI7nBtBGpsw_O3wCyD/view?usp=sharing) **t = 1:07**

**Figure 13.** [**v2 yolov5**](https://drive.google.com/file/d/1XU6IWtmS9EtDVjyI7nBtBGpsw_O3wCyD/view?usp=sharing) **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