專題報告

B08901179
陳奕瑒

這學期做的是 YOLO 的研究，由於一開始我對 YOLOv4、YOLOv5 沒什麼研究、概念，想說 version 4 的版本應該比較簡單吧，就決定做 YOLOv4 了。最前面幾週，我是參考 https://medium.com/ching-i/yolo-c49f70241aa7 上面的 tutorial 進行操作。在 training/testing data 產生的過程，作者是用 wider face 資料庫，轉換 label 的程式是用作者寫的 code，不過在跑訓練的時候遇到幾個 error，由於我對 train.py、轉換的 code 都不太熟悉，所以一開始覺得蠻挫折的！（改進：以後我應該直接參考原作者的 tutorial 就好，其他人寫的教學通常都沒在更新，所以出 bug/error 的時候比較難解決）。後來，啟翰貼給我原作者的教學，在慢慢操作後，終於完成第一次的訓練，如圖一所示，我用的 class 只有 1 個，training data 有 1500 張，validation 有 300 張，使用的 model 是 YOLOv4.cfg。

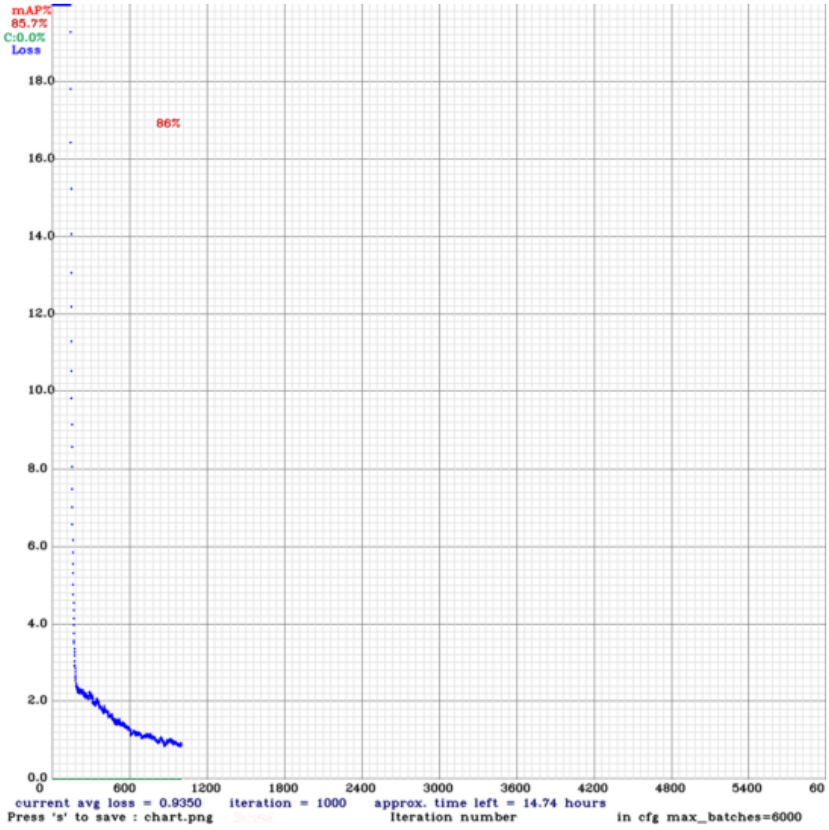

Fig. 1. YOLOv4.cfg 的 training result

可以看到 iteration = 1000，mAP 就可以達到 86%，後來嘗試用不同模型去做比較，得到的結果如圖二、圖三所示。

	YOLOv4-tiny	YOLOv4
Iteration	1200	1000
mAP	82.0%	85.7%
avg loss	0.4464	0.935

Fig. 2. 不同 model 在 mAP、avg loss 的比較結果

Original	YOLOv4-tiny	YOLOv4
		

Fig. 3. 不同 model 預測的結果

在報告這邊的時候，教授有問我說為什麼 YOLOv4 的 mAP 比 YOLOv4-tiny 來得大，avg loss 卻也同樣 YOLOv4 比較大。理想上 mAP 的計算方式應該會跟 avg loss 有關，所以不應該得到這樣的結果才對。那時候，因為我對 mAP 的計算方式不太了解，所以沒能回答出來。所以，下一個禮拜，我去閱讀了 Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression 這篇論文。mean average precision (mAP) = $\frac{1}{|classes|}\sum_{c\in classes}\frac{TP(c)}{TP(c)+FP(c)}$；若 IoU > Threshold，則會被認定為 True Positive (TP)，反之，則為 False Positive (FP)。所以，我們的目標是要讓 IoU 越快超過 Threshold 越好，至於如何讓 IoU 變更好，就和 loss 的計算方式有緊密的關係。好的 loss 計算方式，可以讓模型 predict 的 bounding box 越快收斂到 ground truth 的 bounding box。論文裡面提到了不同 loss 的計算方式如圖三、圖四所示。

$$IoU = \frac{|B \cap B^{gt}|}{|B \cup B^{gt}|}$$

$$\mathcal{L}_{IoU} = 1 - \frac{|B \cap B^{gt}|}{|B \cup B^{gt}|}$$

$$\mathcal{L}_{GIoU} = 1 - IoU + \frac{|C - B \cup B^{gt}|}{|C|}$$

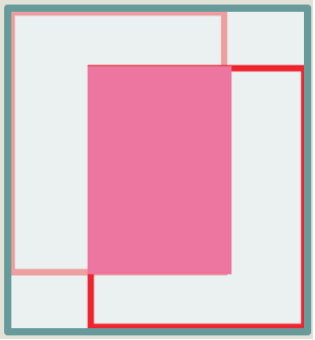

Fig. 3. IoU loss and Generalized IoU loss

$$\mathcal{L}_{DIoU} = 1 - IoU + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2}$$

$$\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2} + \alpha v.$$

Fig. 4. Proposed Distance-IoU loss and Complete IoU loss

$\mathcal{L}_{GIoU}$ 因為只考慮到 overlapped 面積，沒有考慮到 bounding box 的 central point，所以收斂的速度會比較慢；作者提出的$\mathcal{L}_{DIoU}$則修改了原先 loss 的計算方式，多考慮了兩個 bounding box 的中心點的距離，$\mathcal{L}_{CIoU}$更多考慮了 bounding box 的長寬比，測試得到的結果如圖五所示。

Loss / Evaluation	AP		AP75	
	IoU	**GIoU**	**IoU**	**GIoU**
$\mathcal{L}_{IoU}$	46.57	45.82	49.82	48.76
$\mathcal{L}_{GIoU}$	47.73	46.88	52.20	51.05
Relative improv. %	2.49%	2.31%	4.78%	4.70%
$\mathcal{L}_{DIoU}$	48.10	47.38	52.82	51.88
Relative improv. %	3.29%	3.40%	6.02%	6.40%
$\mathcal{L}_{CIoU}$	**49.21**	**48.42**	**54.28**	**52.87**
Relative improv. %	**5.67%**	**5.67%**	**8.95%**	**8.43%**
$\mathcal{L}_{CIoU}(D)$	**49.32**	**48.54**	**54.74**	**53.30**
Relative improv. %	**5.91%**	**5.94%**	**9.88%**	**9.31%**

Fig. 5. Quantitative comparison of YOLOv3 (Redmon and Farhadi 2018) trained using $\mathcal{L}_{IoU}$ (baseline), $\mathcal{L}_{GIoU}$, $\mathcal{L}_{DIoU}$ and $\mathcal{L}_{CIoU}$

可以發現作者提出的 loss 計算方式，在 IoU/GIoU 的表現上面都比$\mathcal{L}_{IoU}$、$\mathcal{L}_{GIoU}$ 來得好。雖然我了解了關於 IoU 和 mAP 之間的關係，不過似乎還是無法解釋老師問我的問題。後來，我仔細去看了原作者計算 loss 的程式，發現說他的 avg loss 是 training 的 loss 而不是 validation 的 loss，所以圖二也就是圖六的問題就可以順利解釋！

	YOLOv4-tiny	YOLOv4
Iteration	1200	1000
mAP	82.0%	85.7%
avg loss	0.4464	0.935

Fig. 6. mAP 和 avg loss 之間的問題

完成 YOLOv4 的訓練後，準備進行 YOLOv4 的 quantize。但是，當我試著整合學長 INT8_quantization_tutorial 和 YOLOv4 的時候，發現 YOLOv4 的編譯方式是 C++而學長的 tutorial 是 pytorch (python)，所以我去找了 YOLOv4 pytorch 版本=> WongKinYiu/pytorch。可是把 WongKinYiu/pytorch 版本的 YOLOv4 丟到遠端跑的時候，出現了許多 error，一個是 GCC 版本要超過 5.0，另一個 error 是 cuda 的問題。由於 WongKinYiu 提供的 YOLOv4 pytorch，要安裝 mish-cuda，但是這個 mish-cuda 在遠端會有一些問題。接下來的三週都在 fix 這幾個 problems，但都沒有太大的突破，也讓我產生放棄 YOLOv4，選擇 YOLOv5 的念頭。所以，我請教啟翰關於 YOLOv5 的用法，隨即進行了訓練，訓練結果如圖七所示。

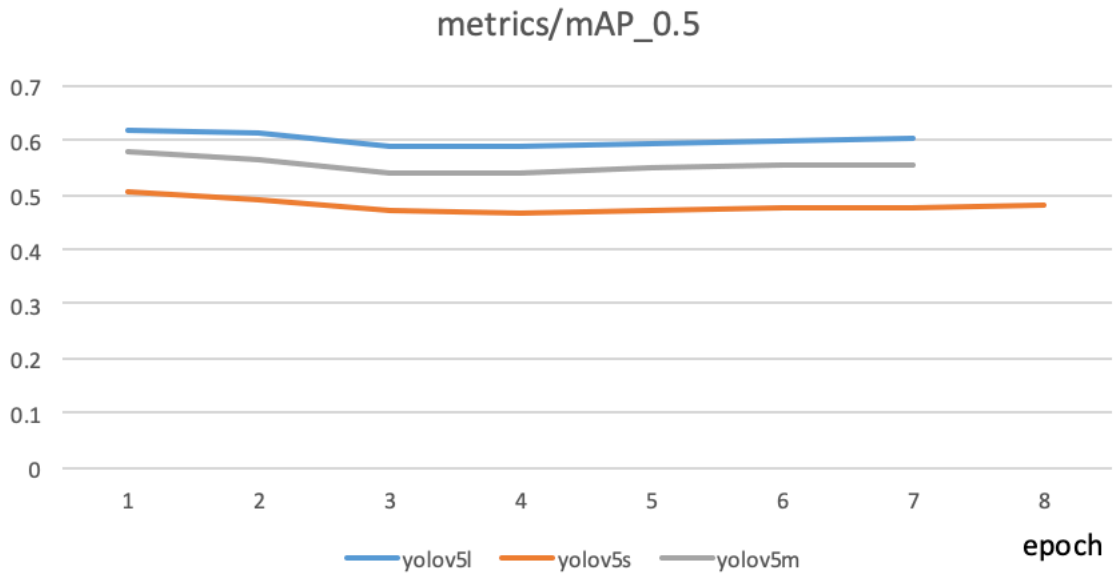

Fig. 7. YOLOv5 training result

training data 是使用 coco dataset，有 80 個 class：train image 有 118287 張圖片，validation image 有 5000 張圖片；在 model 方面分別使用了 yolov5s、yolov5m、yolov5l 進行訓練。可以從圖七看出在相同 epoch 底下，yolov5l 的 mAP 最高，也就是說它的

預測準確度比其他模型來的好。另外，進行了 quantize 後的結果如圖八所示。Quantize 的部分是拿 yolov5s 的 weight 進行 retrain，同時加入 FloatSD4 的 quantization。雖然 weights 和 bias 以 FloatSD4 的形式儲存，但是可以看到說，yolov5_qat 的 mAP 依然能維持在 0.46 附近，和 yolov5 相比也不會掉太多。

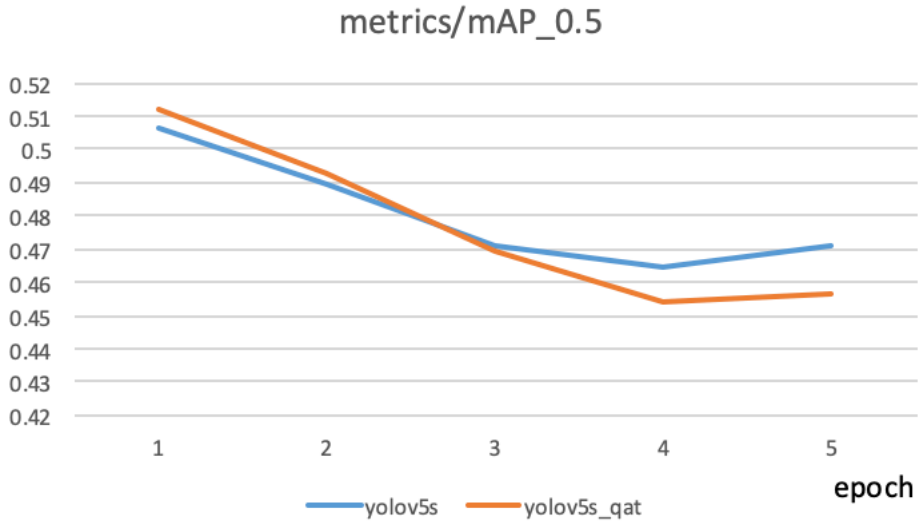

Fig. 8. mAP of Yolov5s and YOLOv5_qat

心得：

學期初在修專題的時候，其實不太清楚自己想要研究什麼。因緣際會下，聽到老師說碩士的學姊也在研究 YOLO。那時候，我就想說如果可以跟學姊合作的話，也許就可以學習業界最新的技術等等。稍微跟學姊討論過後，我就決定先做 YOLOv4。一開始，訓練沒有說特別順利，一方面我對機器學習蠻陌生的，另一方面在 colab 上面做訓練需要注意蠻多事情的。比方說：training data、圖片備份到雲端，如果沒先備份好的話，一不小心 colab 需要重新連線，所有資料就會消失。那時候，在訓練 YOLOv4 的時候，還要每隔三小時看電腦一下，因為 colab 會不時寄出小測驗，來判斷你是不是機器人。也因為這個原因，讓我想要在實驗室的 GPU 跑訓練，可是遇到 cuda 和 GCC 版本之類的問題時，蠻挫折的。但是也因為這個機會，讓我可以認識 YOLOv5，進而去比較 YOLOv4 和 YOLOv5 的優缺點。在 testing result 方面，YOLOv5 的程式寫的很完整，會自動幫你把 mAP、loss 記錄到 csv 檔裡面，很方便；YOLOv4 則是還需要寫程式才能取得你想要的資料進行分析。像圖一就是利用 YOLOv4 內部的畫圖工具幫我畫的圖，因為那時候我沒辦法單獨把 mAP 的結果取得在 excel 上面作圖，所以圖片畫面很亂。在環境方面，YOLOv5 本身就是 pytorch 的體系，在實驗室訓練的時候，也不會像 YOLOv4/pytorch 會需要下載 mish-cuda 而導致的問題。另外，YOLOv5 也有比較多人在維護，這方面在執行程式的時候蠻重要的，才不會一開始訓練的時候就挫折感十足，花不必要的時間在非訓練相關的事上。

進到了 quantize 的階段，我認為這是一個嶄新的學習，更有所謂「研究」的感覺。遇到問題的時候，我得上網自己找資料，針對錯誤的部分自己去修復，再也沒有同學或學長姐可以依靠。雖然需要花費很多的時間，可是當你親手解決一個又一個的錯誤，看到程式能重新 run 的時候，內心的感動是無法形容的！經過這個學期的專題研究，讓我知道事先準備工作是需要的，需要先上網查資料、看論文，才不會花了很

多時間做不必要的事，就像我連 YOLOv4 是 C++的編譯方式都不清楚等等問題；經過這個學期的專題研究，也讓我對機器學習這塊，有一番全新的認識！