Machine Learning Final — Pneumonia Detection

- b05902128 鄭百凱 b05902117 陳柏文: RetinaNet Model的訓練 & 測試
- b05902125 葛淯為 b05902033 高晟瑋: 各種方法研究 & Report撰寫

Introduction & Motivation

- 由於醫療技術的進步，X光片的照射也越來越清楚，相對的成本也降低許多，然而在現今社會中，如果要判斷肺部疾病的話，需要藉由去大醫院照X光片接著交給醫生來判斷是否有如肺炎的疾病存在。但這方法相當耗費醫療資源，若我們可以只藉由如X光片來寫一個程式直接判斷是否存在肺部疾病，而且其準確率不低於一定水準的情況下，將可以大量減少時間以及醫療成本。
- 但其實這並非是個簡單的工作，我們並不能直接寫個程式抓住固定的幾個地方來判斷，原因之一是每個人的肺部大小不一，若直接抓特定地方可能使判斷位置錯誤而造成誤判。其二是因為肺部疾病發生的位置與顯現明顯的點每個人有所不同，而造成沒有判別到的情況。在這樣的情況下，我們便想如果能藉由機器自主學習每個人的病例，加以套用在更廣大的群眾之中，或許便可以做到當初我們所想要的利用程式判斷肺部疾病，或是更進一步的加以預測哪些人比較有可能感染肺部疾病。
- 這並非是個簡單的Network便可以解決，困難點之一為到底需要多大的training set才可以確保我們的Network套用在大部分所有人的情況下皆可以有很棒的結果，之二為上述所提到，每個人的肺部大小以及發生的位置皆不相同，若使用一般簡單的CNN來train的話，很有可能造成train不起來，或是觀察疾病位置及發生完全不符合我們所想的情況。或許我們無法完全解決困難一，但我們卻可以藉由改善我們Network來達到解決困難二的結果。
- 現在大部分處理這種情況的Network概念皆是利用region的概念，對每一個區塊進行判斷而不是整張圖。而助教所提供的便是R-CNN、RetinaNet及YOLO，下面我們會介紹我們所實作的方法以及實際上做起來的效果。

Data Preprocessing

keras-retinanet對於自定義的csv dataset做training的時候需要兩個csv檔：annotations.csv跟class.csv

1. annotations.csv 跟給定的train_labels.csv差不多， 各個title分別為["圖片路徑", "x1", "y1", "x2", "y2", "category"]，唯一的差別是要將train_labels.csv的"weight"跟"height"轉換成"x2"跟"y2"。

train/train00000.png					
train/train00001.png					
train/train00002.png	316	318	486	796	Pneumonia
train/train00002.png	660	375	806	777	Pneumonia
train/train00003.png	570	282	839	691	Pneumonia
train/train00003.png	83	227	379	665	Pneumonia
train/train00004.png	552	164	928	840	Pneumonia
train/train00004.png	66	160	439	768	Pneumonia

2. class.csv 此次作業只對單一的class進行預測，所以只有Pneumonia一種

Pneumonia	0	

Methods

當前detection主要有one stage與two stage兩種方法:

1. one stage的方法是直接對所生成的bounding box進行細部的分類，因為one stage生成的正負樣本數量差距很大，所以可能造成training的效果不佳。
2. two stage是在第一個stage運用region proposal network的方法找出可能的object location，再進行一個二元分類，分辨出樣本是屬於前景或背景，在這個過程中，大量屬於背景的bounding box會被刪除。所以可以稍微減小正負樣本類別不均衡的現象，讓將來再做training的時候可以有比較好的performance。但是缺點是two stage方法的速度很慢。
3. 以下是兩個常用的衡量數據 IoU: Intersection over Union，代表兩張圖的交集區域占整體區域的多少。 mAP: Mean Average Precision

RCNN

- RCNN是屬於two stage的方法，其概念如上述所提及，利用selective search algorithm找出region proposal，這演算法主要分為兩個部分：Hierarchical Grouping Algorithm以及Diversification Strategies。
- Hierarchical Grouping Algorithm主要可以分為四個步驟：
    1. 計算所有鄰近區間的相似性
    2. 兩個最相似的區域被組合在一起
    3. 計算合併區域與相鄰區域的相似度
    4. 重複 2, 3直到圖像變成一個地區

Algorithm 1: Hierarchical Grouping Algorithm

Input: (colour) image
Output: Set of object location hypotheses L

Obtain initial regions $R = \{r_1, \cdots, r_n\}$ using [13]
Initialise similarity set $S = \emptyset$
foreach *Neighbouring region pair* (r_i, r_j) **do**
  Calculate similarity $s(r_i, r_j)$
  $S = S \cup s(r_i, r_j)$

while $S \neq \emptyset$ **do**
  Get highest similarity $s(r_i, r_j) = \max(S)$
  Merge corresponding regions $r_t = r_i \cup r_j$
  Remove similarities regarding r_i : $S = S \setminus s(r_i, r_*)$
  Remove similarities regarding r_j : $S = S \setminus s(r_*, r_j)$
  Calculate similarity set S_t between r_t and its neighbours
  $S = S \cup S_t$
  $R = R \cup r_t$

Extract object location boxes L from all regions in R

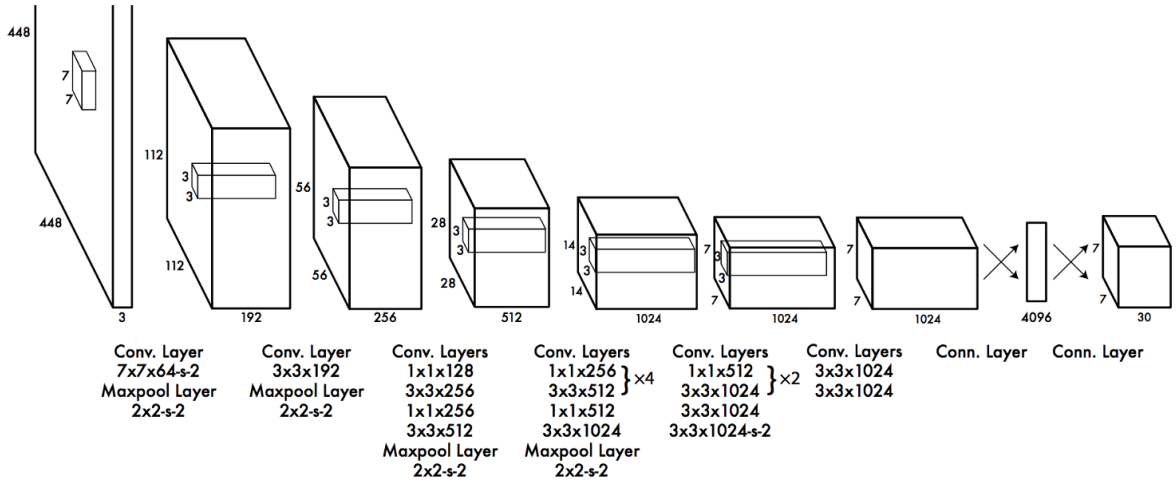
- 細節不多敘述，主要概念便是每次迴圈中形成更大區域新增到候選列表中，從小segment建立到大segment。Diversification Strategies主要提供三個strategies使得抽樣更多樣化：

1. 利用不同的色彩空間
2. 採用不同的相似度測量
3. 通過改變起始區域

- 利用上述所說的演算法後，找出region proposal，接著CNN計算每個區域，然後對每個區域使用SVM進行分類，除此之外還會預測偏移值使得物品更容易在bounding box內。
- 但這方法就如同上述所提到，因為要先找到每個region proposal，必須先進行algorithm，以及找出region之後需要對每個region做CNN以及分類，所需花費時間為CNN的數千倍，所以基本上是使用fast rcnn，在最後一層Convolution layers得到一個H * W的feature map，然後在feature map上面各自max pooling，每個region的到相同大小的矩陣。
- 更快的方法是faster RCNN，利用feature map直接找出region proposals，也就是RPN(Region Proposal network)，在feature map上取sliding window，每個sliding window的中心點稱為anchor point，然後利用不同大小的box對這個anchor point計算可能含有物體的機率，其最大的box稱為anchor box，最後得到有可能的bounding box。

YOLO

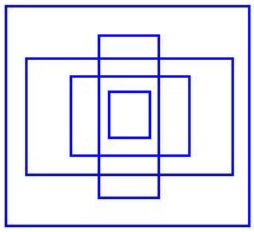
- YOLO主要好處為快速能用於real-time，以及運用於其他種類的class上依然能表現不錯的結果。
- YOLO屬於one stage的方法，相比起上述所提到RCNN，速度有大幅的提升，並且強調能夠real-time的判斷bounding box，主要概念便是裁減圖片成grid cell，每個grid cell只能判斷一個物體，對於每個grid cell，主要有三個重點：
    1. 分成B個Bounding box，每個bounding box有個box confidence score
    2. 不管有幾個bounding box都只能偵測一個物體
    3. 對於C個class都給予一個conditional class probability(條件機率相對所有class)
- 假設我們的grid cell有7 * 7的大小，那最後YOLO所要預測的shape便為 (7,7,B * 5 + C)，5來自於(x, y, w, h, box confidence score)，所以先利用CNN將7 * 7 * 1024，再用Fully connected layer壓到7 * 7 * 30，最後把box confidence score大於threshold(例如0.25)當作最後的prediction的結果。



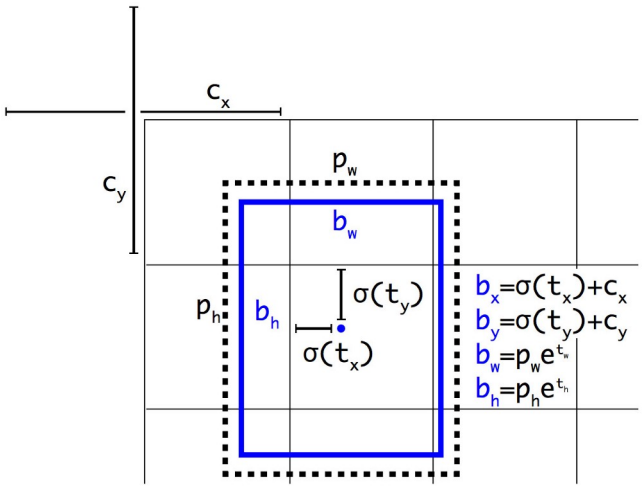
- YOLO的loss來自與ground truth相比，其loss function主要可以分為三個部分：
    1. the classification loss(與實際目標分類機率的平方差)
    2. the localization loss(與實際位置高和寬的平方差)
    3. the confidence loss(與實際信心分數的平方差)

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned}
$$

- 其內容與參數在這就不詳細的介紹。YOLO為了防止對同個物體的重複預測，使用Non-maximal suppression，其內容主要為：
    1. 利用confidence scores對prediction進行sort
    2. 從最高分數的prediction開始，如果當前的prediction與先前遍歷過的prediction的class相同且跟現在的prediction IoU > 0.5便忽略他
    3. 重複步驟2直到完成所有檢測
- 雖然YOLO檢測的速度很快，但其精確程度卻不如R-CNN，而YOLOv2更進一步的改善，增加Batch normalization並且移除dropout，使得mAP上升2.4%。以及利用448 * 448來自Imagenet的圖片輸入finetune從224 * 224轉變到高畫素的過程，大約10個epochs，使mAP上升4%。最主要的改變是新增anchor的概念，把YOLO最後的Fully connected layer用convolution layer以及anchor box來預測邊框，其中細節在這不細講，簡單的概念便是預測anchor中心的偏移值，使我們可以對中心延伸更多固定的形狀，使預測更能成功，如下面這張圖所示：



除此之外還做了Dimension Clusters，用K means以及IoU對邊框進行分類使得train更容易成功，還有Direct location prediction預測anchor可能的偏移範圍，基本概念如下圖：

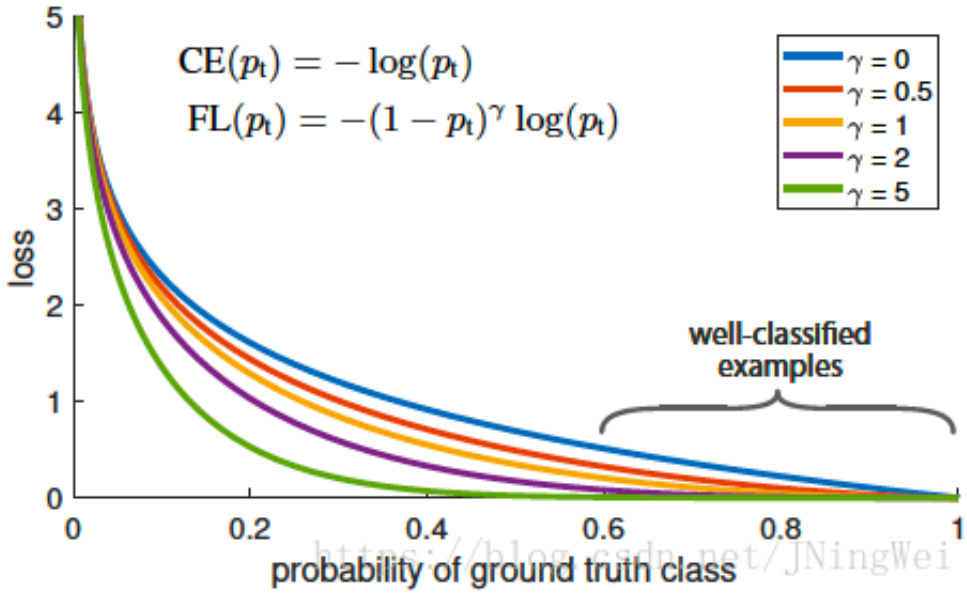


- 經過上述改進後，精確度可以達到與R-CNN一樣好，但是比R-CNN還要快速，而且可以適應不同大

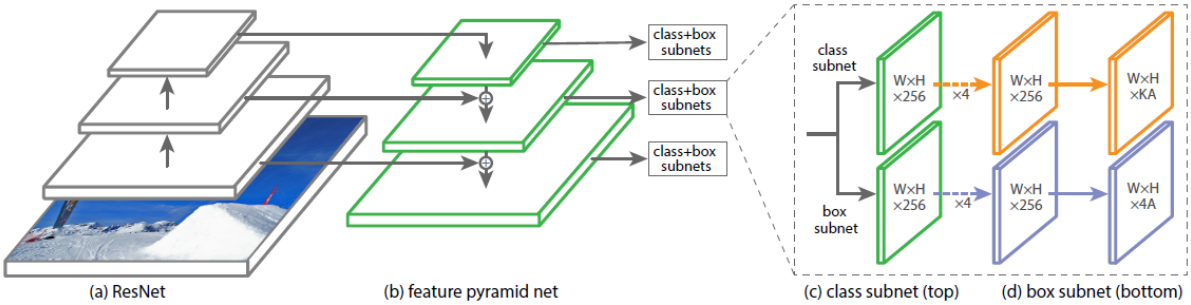
小的圖片。

RetinaNet

- 在one stage的detector中存在著嚴重的foreground-background class imbalance problem，原因是因為在演算法開始之前都會產生很大量的bounding box，然而一張圖片中物體卻沒有那麼多，導致大部分的bounding box都是background，而進一步的使detector最後要classify時把所有bounding box歸類為background卻依然可以得到很高的accuracy。two stage由於上述所提到的RPN對anchor進行二分，讓background的數量不是那麼多，缺點也如上面所提到的速度相對慢了許多。
- 為了在one stage上面解決這問題，在RetinaNet提出一種新的概念：Focal loss。在我們所熟知的CrossEntropy(CE)：$CE(p_t) = -log(p_t)$ 加上一個常數項形成$CE(p_t) = -\alpha log(p_t)$。alpha可以視為是一個用來平衡前景和背景不均的參數。再加入另一個變數項便成為所謂的 Focal Loss(FL)：$FL(p_t) = -\alpha_t(1-p_t)^\gamma log(p_t)$ 。gamma的功用是讓能夠清楚辨識的樣本所造成的loss相對下降，而容易被混淆的樣本的loss相對上升。讓我們所關心的物體，也就是容易被混淆的Pneumonia的bounding box更重要。 底下附上參數的實驗結果：



- RetinaNet主要架構如下：



  - 步驟(a)&(b): 在Resnet的Top加上Feature Pyramid net(FPN)，利用Resnet或其他CNN架構的pretrain model，從Input圖片建立multi-scale feature pyramid。anchor的面積隨著金字塔等級P3到P7變成32^2到512^2遞增，每個level有9個anchor，每個anchor有一個長度為K(K為class數目)的one-hot分類目標向量，以及長度為4的box regression向量(anchor與目標物之間的偏移值)。如果IoU在0.5以上則anchor被判斷為目標物所在，若IoU在0 ~ 0.4則anchor被判斷為background，若在0.4~0.5之間便忽略。
  - 步驟(c): classification subnet對於A個anchors預測物體屬於K個class是否存在的，這個subnet運用FCN，有著4個convolution layer且用ReLU activation，最後用3 * 3 conv layer with KA filters。
  - 步驟(d): box regression subnet便是用於物體位置的確認，與分類無關只判斷位置，對每層

pyramid做FCN計算偏移值使其更靠近ground truth，唯一跟classifiction subnet不同的是對每個位置使用4A linear output。對於每個FPN level只取前1000得分高的prediction，
    - 最後把這些合併以及運用non-maximum suppression(YOLO提過)當作最後輸出。在訓練過程中，total focal loss來自於100k個anchor的focal loss經過normalized。
- 我們可以看到相比YOLO以及Faster R-CNN，RetinaNet皆有較佳的表現，而且其時間也不會太久，所以這次的work主要先嘗試利用RetinaNet的架構。

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [16]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [20]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [17]	Inception-ResNet-v2 [34]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [32]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [27]	DarkNet-19 [27]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [22, 9]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [9]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet (ours)	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet (ours)	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2

Experiment & Discussion

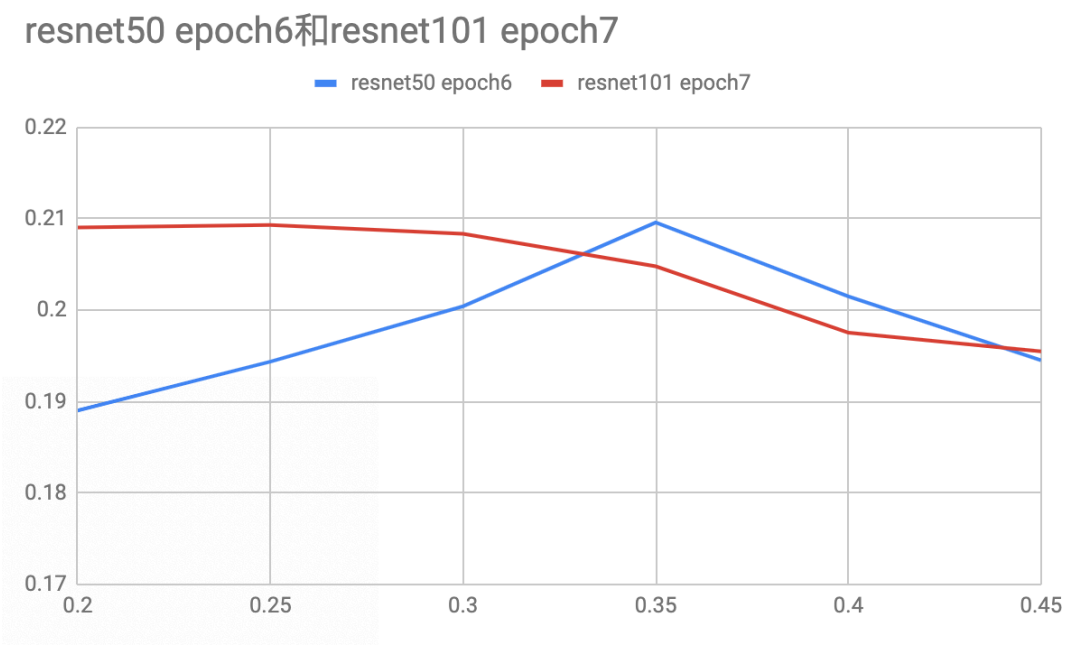
Yolo(v2)

- 我們是參考 **Yolo-v2-pytorch** 這個repo來做訓練。訓練初期，threshold要設的非常非常小(大約0.005)才能夠偵測出物件，accuracy大概只有5%。大約10個epoch之後，就比較能看出成果，threshold可以設大一點(0.03左右)，但跟正常的threshold相比的話還是遠遠低了許多。而我們的model在10個epoch最高的準確率是11%，更多epoch之後就會overfit。Accuracy跟底下的RetinaNet相比差了許多。

RetinaNet

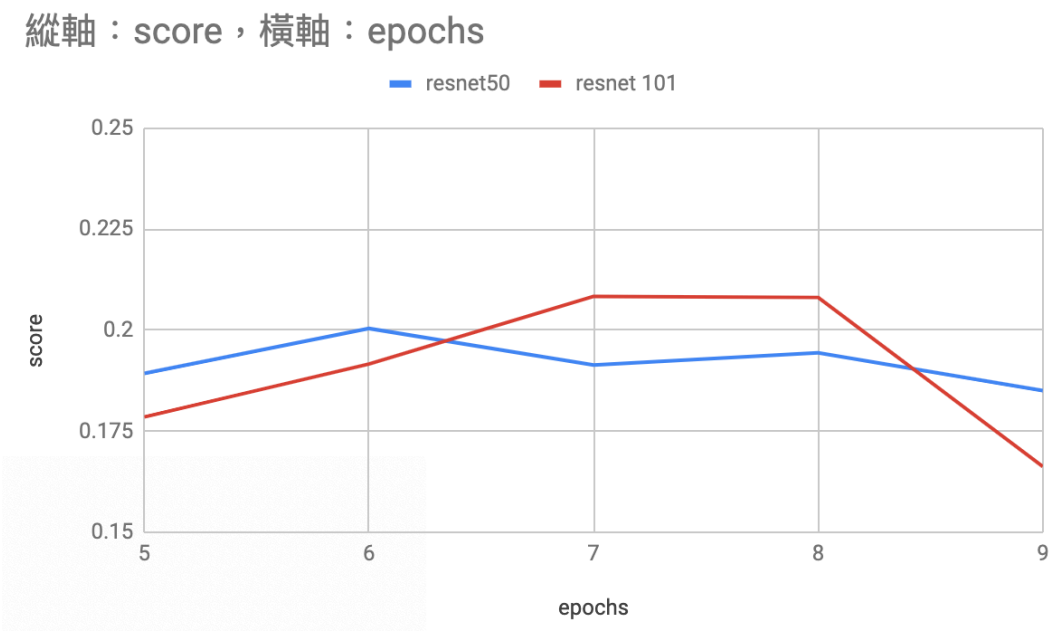
- 我們是參考 **keras-retinanet** 這個repo來做訓練，調整的東西包含: backbone model, epochs, threshold, loss function

- Threshold: 這個數值代表bounding box要至少有多少的信心才會output出來，下圖為針對兩個backbone model我們分別以第6個epoch&第7個epoch的model來做預測。



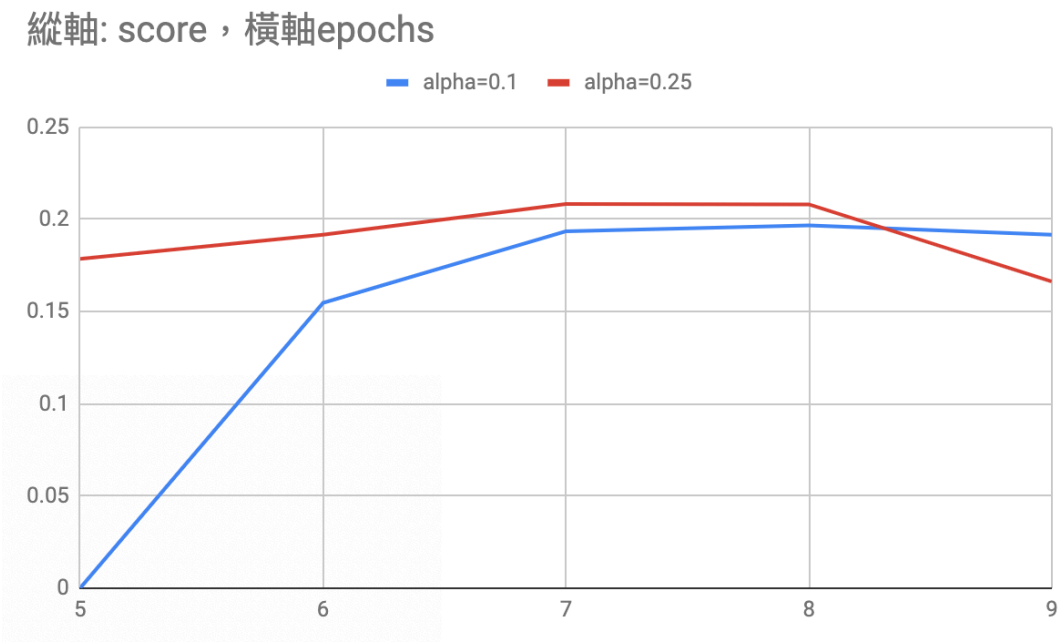
我們可以發現對每個不同的model其實threshold沒有一個固定的規律，基本上都要每個值試試看才行。

- Backbone model: 這部分是影響第一步的抽feature過程是用什麼CNN架構，我們測試了兩個分別是resnet50 & resnet101，並且訓練9個epochs，結果如下：



我們可以看出Resnet101效果較好，但也較難較晚收斂。(threshold均為嘗試後最好的結果)

- Alpha(params of loss function): 我們測試了兩種的alpha值，結果如下圖，其中alpha=0.1的狀況前面五個epoch根本無法predict出任何結果，也較難收斂。(threshold均為嘗試後最好的結果)



- Gamma(params of loss function): 訓練過程中，我們嘗試了3種數值分別是1, 2, 4，其中我們發現gamma=1的效果最好，下表為訓練5個epochs後的predict成果。而本次kaggle上最好的結果也是gamma=1時的訓練結果。

Gamma	1	2	4
score	0.18655	0.17852	0.1465

Conclusion

- RetinaNet的實驗結果為：
  - resnet50的效果比resnet101好，和RetinaNet作者作出來的結果相同，推測是因為resnet101的model參數比較多。
  - alpha改為0.1會導致model train的速度很慢，要經過更多epoch才會接近收斂。
  - gamma小一點，accuracy會比較好。gamma變大沒有讓accuracy上升的可能原因是model並沒有辦法很有信心的辨認Pneumonia。
  - threshold的設定每個model都有所差異，每個值都要去測試看看，才能找出最好的threshold。
- 根據實驗後，我們可以知道R-CNN與YOLO都不比RetinaNet來的好，推測可能是因為這次dataset中大部分人的骨頭差異並不大，所以容易被判斷成background，加深判斷難度，而R-CNN結果不佳猜測是圖片顏色差異不大，如同上面所提到造成色彩空間太過於相似而導致。由此可知RetinaNet對於此Dataset的幫助比較大，之後或許可以考慮對於Pyramid level中的FCN多加一些不同的layer，例如fully connected layer，原本的輸出加上病人之前的病歷資料當作linear的input去train，也許可以使Perfomance更好。
- R-CNN、YOLO、RetinaNet都是設計來處理在影片中實時辨識物體，然而我們這次的task是用來辨識圖片，並不需要快速地辨識，因此這些架構並不完全符合這個task的需求。以RetinaNet為例，他的bakbone model都是CNN的model，能夠辨識高達1000種物體，而我們只需要偵測Pneumonia一種物體而已，因此在這個model中，有許多部份是用不上的。如果有一個專門辨識Pneumonia的CNN

model，再用RetinaNet的FPN架構，或許能夠有更好的效果。

- 在衝kaggle排名時其實很多時候都在調整threshold，有時候看起來表現沒有很好的model其實調整到對的threshold成績就會突飛猛進。

Reference

- https://medium.com/@syshen/%E7%89%A9%E9%AB%94%E5%81%B5%E6%B8%AC-object-detection-740096ec4540
- https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
- https://zhuanlan.zhihu.com/p/39927488
- https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088
- https://zhuanlan.zhihu.com/p/35325884
- https://blog.csdn.net/JNingWei/article/details/80038594
- https://towardsdatascience.com/review-retinanet-focal-loss-object-detection-38fba6afabe4
- https://zhuanlan.zhihu.com/p/48958966
- https://github.com/fizyr/keras-retinanet