Deep Learning 2024 Final exam_B103012002_林凡皓

一、 Implementing Backbone and Feature Pyramid Network

這次期末考是主要是實作 FCOS(fully-convolutional one-stage object detection model)，此模型架構如下圖所示，

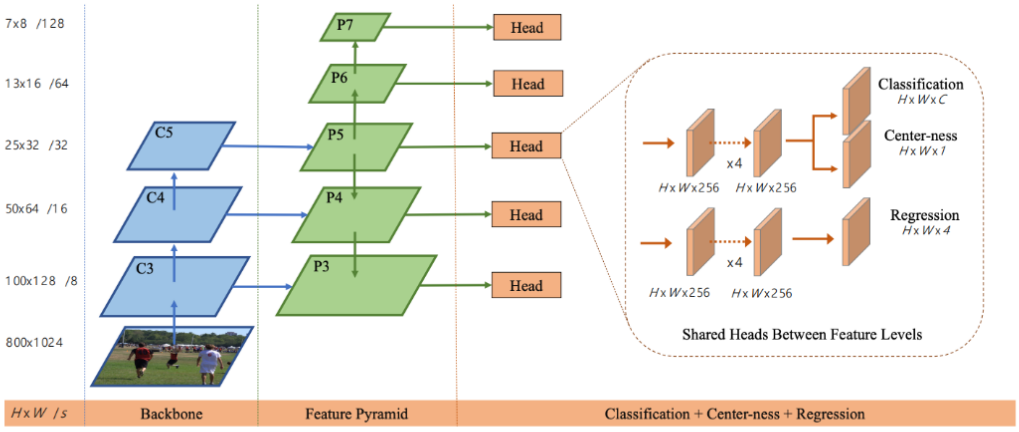


Figure 2 – The network architecture of FCOS, where C3, C4, and C5 denote the feature maps of the backbone network and P3 to P7 are the feature levels used for the final prediction. $H \times W$ is the height and width of feature maps. '/s' ($s = 8, 16, ..., 128$) is the down-sampling ratio of the feature maps at the level to the input image. As an example, all the numbers are computed with an 800×1024 input.

此架構主要可以分成三個部分，backbone、feature pyramid network (FPN)和 head (prediction layers)。這邊主要先實作 backbone 和 feature pyramid network 的部分，這些實作內容在 common.py 檔案中。

- Backbone :

  Backbone 採用 RegNetX-400MF，並載入 pre-trained weight。這部分不需要自己實現。初始化 backbone 結果如下

```
For dummy input images with shape: (2, 3, 224, 224)
Shape of c3 features: torch.Size([2, 64, 28, 28])
Shape of c4 features: torch.Size([2, 160, 14, 14])
Shape of c5 features: torch.Size([2, 400, 7, 7])
```

- FPN :

  FPN 為一種處理多尺度的網路架構，它的目的主要在增強對不同尺度的偵測能力，FPN 會接收 backbone 的特徵 c3、c4、c5 並將它們轉換為新的特徵 p3,、p4、p5。在架構途中會看到還有 p6、p7 的部分，這邊為了方便訓練與實作，將其拿掉。

  1. 實現方法 :

     要做到特徵的轉換可以透過 convolutional layer 來實現，這邊初始化三個 convolutional layer 分別用來處理 p3、p4、p5。在 forward path 中，透過 zip 將 c 與 p 做配對，並利用 for loop 將每一組特徵

都迭代過。每一次迭代過程中，先取得對應的 convolutional layer，並將特徵 c 轉換為特徵 p。對於 p4、p5 而言，我們可以將它們與上一級的特徵座內插，並將結果加到當前的特徵中。

2. 執行結果：

```
Extra FPN modules added:
ModuleDict(
  (fpn_c3): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1))
  (fpn_c4): Conv2d(160, 64, kernel_size=(1, 1), stride=(1, 1))
  (fpn_c5): Conv2d(400, 64, kernel_size=(1, 1), stride=(1, 1))
)
For dummy input images with shape: torch.Size([2, 3, 224, 224])
Shape of p3 features: torch.Size([2, 64, 28, 28])
Shape of p4 features: torch.Size([2, 64, 14, 14])
Shape of p5 features: torch.Size([2, 64, 7, 7])
```

二、 Implementing FCOS prediction network (head)

這邊主要實現 head prediction network 的部分，用來在每個地方預測物件的類別、邊界。此網路定義在 FCOSPredictionNetwork 中。

1. 實現方式：

因為要估測位置方框與類別，這邊會需要 stem_cls 與 stem_box，每一個 stem 都包含一個 convolutional layer 與 ReLU，而 cls 和 box 各自含有兩個 stem。接著會需要初始化預測類別、邊界與中心的 convolutional layer，這邊主要要注意 output channel 數量，預測類別的 output channel 為類別數量，預測邊界 output channel 為 4(上下左右)，預測中心 output channel 為 1。最後對每一層的 convolutional layer 作權重的初始化，權重的部分都初始化為 $N \sim (0, 0.01)$，bias 都初始化為 0。

2. 執行結果：

```
FCOS prediction network parameters:
FCOSPredictionNetwork(
  (stem_cls): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
  )
  (stem_box): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
  )
  (pred_cls): Conv2d(64, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pred_box): Conv2d(64, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pred_ctr): Conv2d(64, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
)
Classification logits:
Shape of p3 predictions: torch.Size([2, 784, 20])
Shape of p4 predictions: torch.Size([2, 196, 20])
Shape of p5 predictions: torch.Size([2, 49, 20])
Box regression deltas:
Shape of p3 predictions: torch.Size([2, 784, 4])
Shape of p4 predictions: torch.Size([2, 196, 4])
Shape of p5 predictions: torch.Size([2, 49, 4])
Centerness logits:
Shape of p3 predictions: torch.Size([2, 784, 1])
Shape of p4 predictions: torch.Size([2, 196, 1])
Shape of p5 predictions: torch.Size([2, 49, 1])
```

三、 Assigning a GT target to every model prediction

FCOS 會在每個位置做物件類別、邊界與中心的預測，在訓練過程中我們需要為這三個預測設定一個 ground truth。因為這三個預測都是在 FPN (p3, p4, p5)上的一個位置，所以我們可以將這個問題看成是幫每個 FPN 特徵位置分配一個 ground truth 邊界與 ground truth 類別。

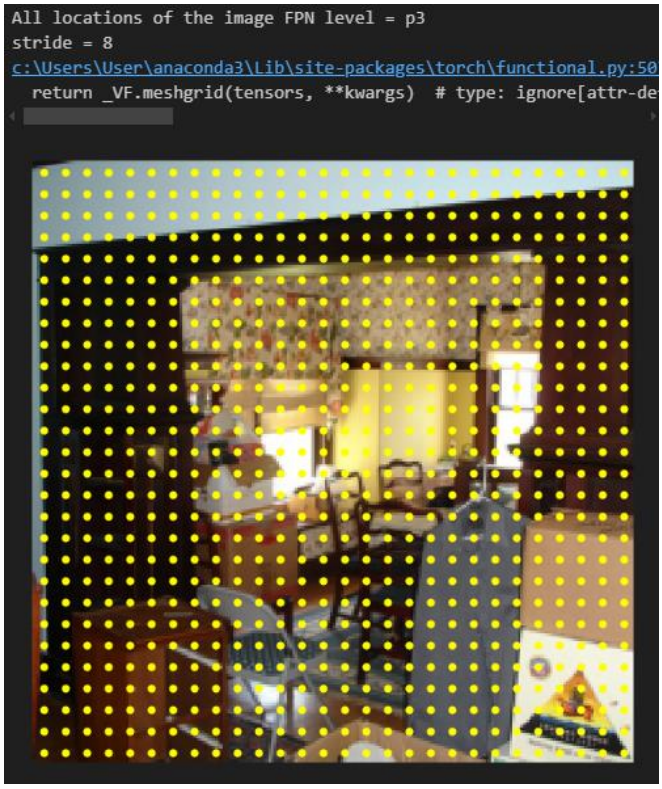
Ground truth 邊界是從 dataloader 中取得的，為一個五維向量(x1, y1, x2, y2, C)，其中(x1, y1)為邊界的左上角，(x2, y2)為邊界的右下角，C 是物件類別。我們會用(xc, yc)來表示 FPN 上的每一個位置，這些座標示圖片上的一個位置，也是特徵的 receptive fields 的中心。

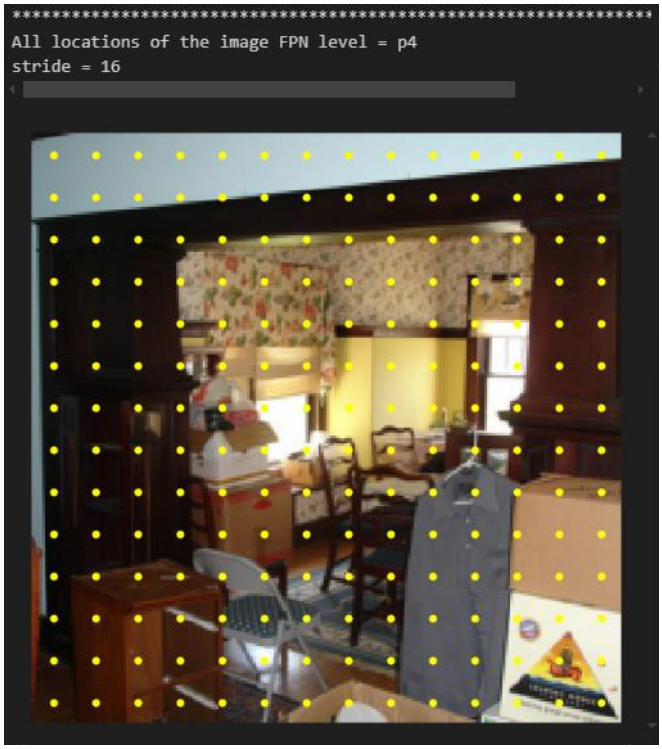
這部分的實現在 common.py 中的 get_fpn_locaton_coords 中。

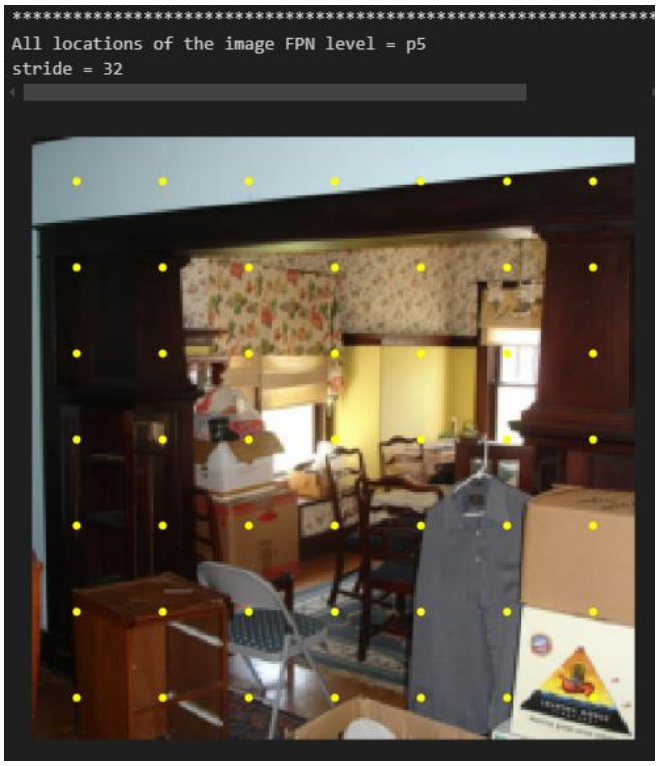
1. 實現方式：

   先初始化一個 dictionary，key 為 FPN 層級的名稱，value 都先初始化為 None。接著透過 for loop 遍歷每一個 FPN 層級，並取得其 shape 和步伐。接著去計算縱向與橫向的位移矩陣，並可以利用 torch.meshgrid 的創建格子點。將縱向與橫向矩陣 reshape 成一維向量後堆疊起來，再加上 level_stride // 2 來將座標調整到每個位置的中心。最後再將計算好的位置存入一開始初始化的 dictionary 中。

2. 執行結果：







四、 Matching feature map locations with GT boxes

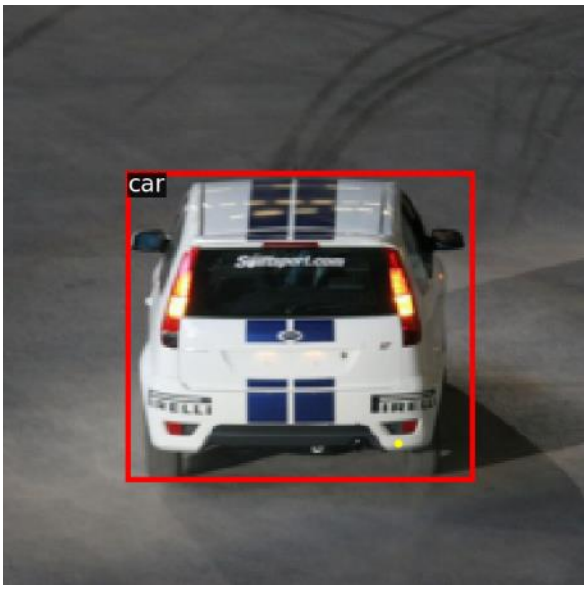
接下來要將每個特徵位置與 ground truth 邊界匹配，在 FCOS 中有兩個規則來進行這種匹配：

- 位置如果在一個 box 之中，則此位置與 box 匹配。如果該位置在兩個 boxes 之中，則該位置與較小的 box 匹配。如果位置不在任何 box 之

中，該位置被歸類為背景。

- 對於特定的 FPN 層級，FCOS 只考慮基於大小的 boxes 子集合，比較大的 box 被設定為 p5，比較小的 box 被設定為 p3。

透過這種方式匹配，每一個位置會收到一個 ground truth 的邊界與類別，即向量(x1, x2, y1, y2, C)，或是一個背景標籤(-1, -1, -1, -1, -1)，這部分不需要自己實現，執行結果如下



五、 GT target for box regression

Box regression 需要預測從特徵位置(上圖中黃色點)到邊界框邊緣(上圖中紅色框)的距離，這邊簡稱為 LTRB。所有位置與 ground truth 邊界都是以絕對座標表示，範圍從(0, 224)，取決於圖像解析度。我們不能夠使用這種座標來訓練，因為這麼大的數值容易造成梯度爆炸，因此我們會將 LTRB 按照 FPN 層級的 stride 做 normalization，normalize 後的數值通常被稱為 deltas。在進行 inference 時，由於模型會與測出 normalize 後的數值，因此我們需要對預測出來的數值進行逆轉換，將神經網路的輸出結果轉換為圖像中的預測框。這部分需要實現 fcos_get_deltas_from_locations 與 fcos_apply_deltas_to_locations 兩個函數。

- fcos_get_deltas_from_locations

1. 實現方式：

   先分別去計算特徵位置與 ground truth box 的距離，接著將計算出來的四個距離堆疊成一個矩陣 deltas。最後處理背景的部分，將為背景的 delta 部分設定為(-1, -1, -1, -1)。

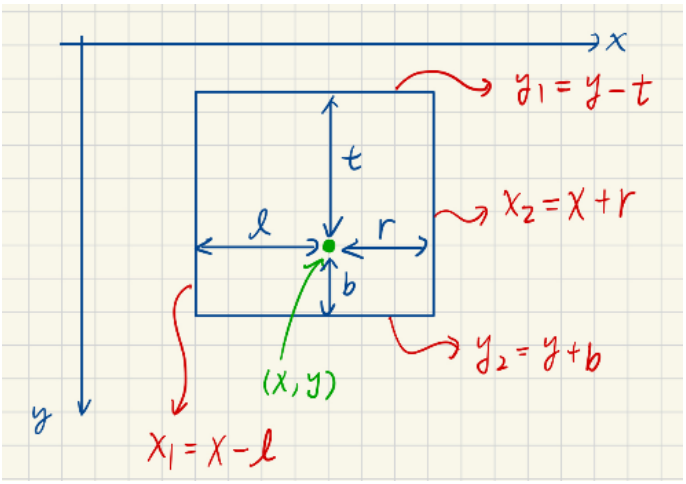
2. 執行結果：

```
Rel error in reconstructed boxes: 0.0
Background deltas should be all -1    : tensor([[-1., -1., -1., -1.]])
Output box should simply be the center: tensor([[100., 200., 100., 200.]])
```

- fcos_apply_deltas_to_locations

1. 實現方式：

   此函數用來根據 deltas 的資訊來取得方框邊界。一開始要將 normalize 的 deltas 變回沒有 normalize 的樣子。由於模型預測出來的 delta 可能是負，但是這並不符合規範，因此將負的數值都 clamp 到 0。最後再根據以下關係取得邊界座標



2. 執行結果：

```
Rel error in reconstructed boxes: 0.0
Background deltas should be all -1    : tensor([[-1., -1., -1., -1.]])
Output box should simply be the center: tensor([[100., 200., 100., 200.]])
```

六、 GT targets for centerness regression

$$centerness = \sqrt{\frac{\min(left, right) \times \min(top, bottom)}{\max(left, right) \times \max(top, bottom)}}$$

此數值在 left = right 且 top = bottom 時會最大，代表說特徵點位於 ground truth 框的中心。計算 centerness 的目的在於讓模型能夠專注在檢測方框中心的部分，進而減少在邊緣處的誤檢。

- 實現方法：

  透過 deltas 取得特徵位置到方框邊緣的距離，並根據 centerness 的公式計算出結果，最後再將背景的部分都設定為-1，讓模型忽略掉這些數值。

- 執行結果：

```
Rel error in centerness: 0.0
```

七、 Loss Functions

FCOS 有三個預測輸出的 layer，這三層分別用不同的 loss function 來優化。

- Object classification

  對於 FCOS 來說，大多數位置都會被分配為背景，導致類別不平衡的問題，因此 FCOS 會使用 Focal loss 來處理這個問題。Focal loss 為 cross entropy 的一種擴展，公式為

  $$FL(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t)$$

- Box regression

  一般來說，FCOS 會使用 Generalized intersecton-over-union(GIoU)來做為優化 box regression 的 loss function，但是由於要簡化問題，這邊直接使用 L1 loss 作為 loss function，L1 loss 公式為

  $$L1(x,y) = |x-y|$$

- Centerness regression

  centerness 為[0, 1]之間的數值，因此採用 binary cross entropy 來做為 Loss function，binary cross entropy 公式為

  $$BCE(p,y) = -[y\log(p) + (1-y)\log(1-p)]$$

- Total loss

  每一個位置的 loss 會包含 object classification loss、box regression loss 和 centerness regression loss，該位置的 total loss 為這三個部分相加後，以 foreground locations 的數量做平均。每個圖像中 Foreground locations 的數量高度可變，因此為了穩定訓練，total loss 通常會以 exponential moving average of foreground locations 來做平均。

這部分不需要自己實現，執行結果如下

```
Classification loss (dummy inputs/targets):
tensor([[[0.0269, 0.3310, 0.0659, 0.0148, 0.1416],
         [0.0164, 0.0468, 0.3919, 0.0590, 0.1108]]])
Total classification loss (un-normalized): 1.2051334381103516
```

```
Box regression loss (L1): tensor([[ 0.7243,  0.3815,  9.8869, 12.4257],
        [ 0.0000,  0.0000,  0.0000,  0.0000]])
Centerness loss (BCE): tensor([0.6874, 0.0000])
```

八、 Object detection model

這邊主要將剛才實現的所有東西應用到建立 FCOS 類別。

- __init__

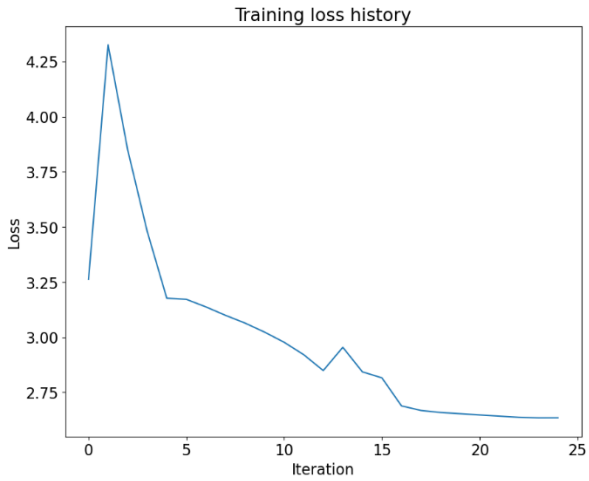
  初始化部分主要初始化 backbone network 與 prediction network，這兩個網路都已經在前面實現好了，因此只需要呼叫它們並將 channel size 設定好即可。

- forward

  forward path 一開始將圖片送入 backbone network 中取得 FPN feature，接著在送入 prediction network 中取得預測的類別、方框與中心。接著要取得每一個 FPN 層級的絕對座標，這部分可以透過 common.py 中實現的 get_fpn_location_coords 來取得。透過先前定義的 fcos_match-location_to_gt 來將 ground truth box 指派給特徵位置，並利用 fcos_get_deltas_from-locations 取得 boxes 的 ground truth deltas。再來要計算 loss，這部分可以仿照 Loss functions 部分作者幫我們實現好的 code。最後將計算好的 classification loss、box regression loss 與 centerness loss 回傳。

九、 Overfit small data

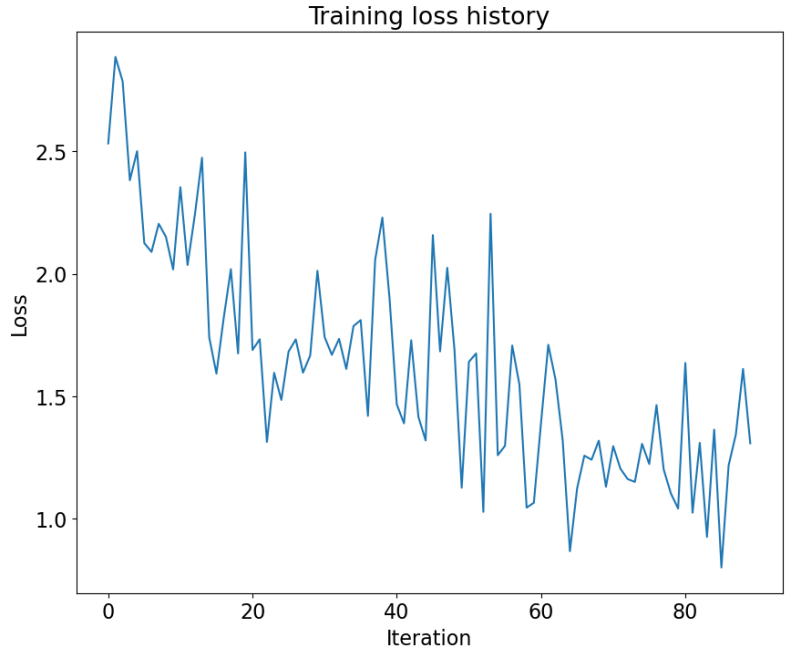
利用小資料來確認模型的實現是否正確，training loss curve 如下



十、 Train a net

確認好整個模型可以正常訓練後，用整個資料集做訓練，訓練參數配置都是期末考預設的參數，訓練 9000 epochs 後，loss 與 training loss curve 如下

```
[Iter 8500][loss: 2.558][loss_cls: 0.570][loss_box: 1.458][loss_ctr: 0.530]
[Iter 8600][loss: 3.105][loss_cls: 0.635][loss_box: 1.788][loss_ctr: 0.682]
[Iter 8700][loss: 3.281][loss_cls: 0.666][loss_box: 1.867][loss_ctr: 0.748]
[Iter 8800][loss: 2.955][loss_cls: 0.556][loss_box: 1.716][loss_ctr: 0.683]
[Iter 8900][loss: 2.087][loss_cls: 0.431][loss_box: 1.196][loss_ctr: 0.460]
```



十一、 Non-Maximum Suppression (NMS)

為了等一下推論，這邊需要先實現 nms。

- 實現方法 ：

  透過計算分類概率和中心度的幾何平均取的 level_pred_scores，並取得之中最大值做為最有信心的類別與分數。接著去檢查信心程度是否高於 threshold，保留那些信心程度高於 threshold 的類別與分數。透過 fcos_apply_deltas_to_locations 計算預測方框，並利用原始圖片修剪超出圖片的方框。最後匯集所有層級的預測結果，並執行 NMS 來過濾重疊的預測。

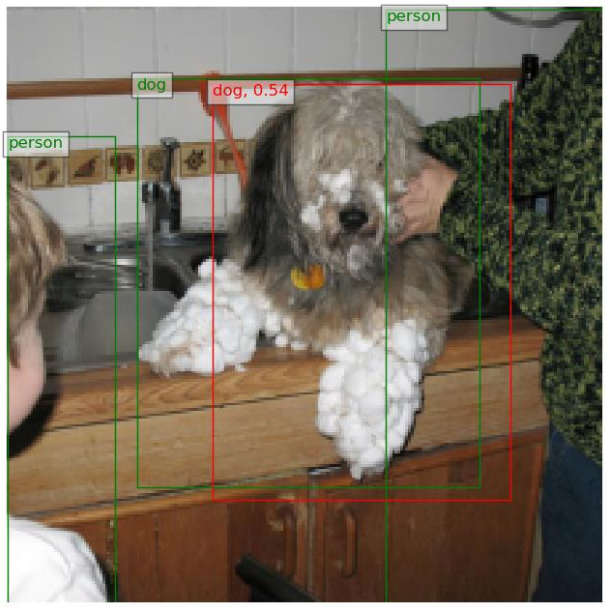
- 驗證結果 ：

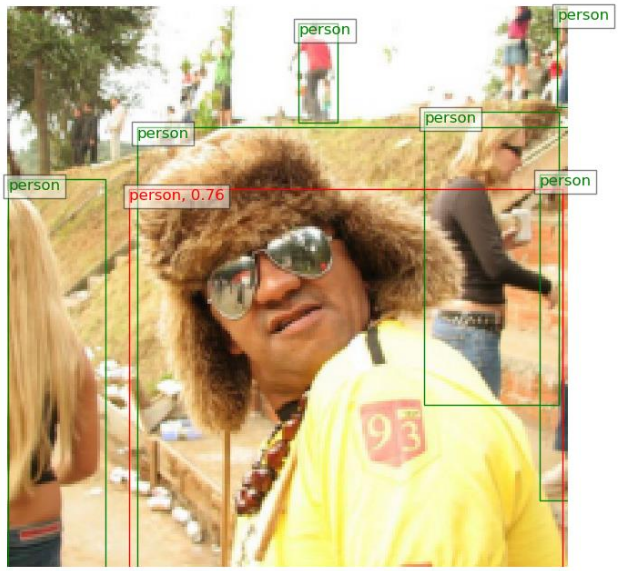
  與 torchvision 的 NMS 比較結果如下

```
Testing NMS:
Your         CPU  implementation: 0.909588s
torchvision CPU  implementation: 0.032657s
torchvision CUDA implementation: 0.040685s
Speedup CPU : 27.852636x
Speedup CUDA: 22.356602x
Difference CPU :  0.0012674271229404788
Difference CUDA:  0.0
```

十二、 Inference

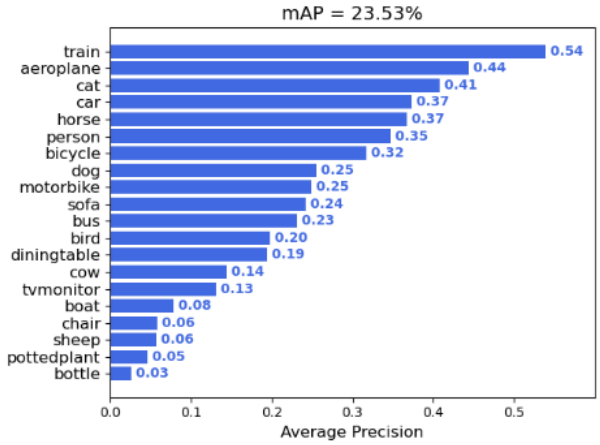
接著將剛才訓練好的模型用來做物件偵測，部分偵測結果如下





十三、 Evaluation

利用 PASCAL VOC validation set 計算 mAP，計算結果如下



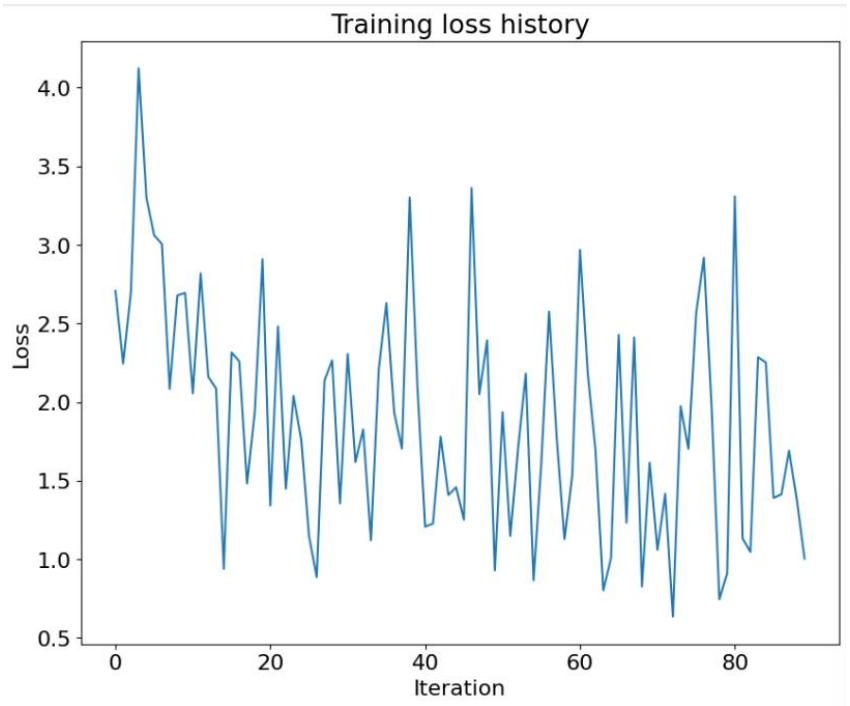
目前模型 mAP 為 23.53 %，高於這次期末考標準的 22 %。雖然說有達到期末考的標準，但是跟 state of art 的模型比較起來(mAP > 80%)相差很多，因此我嘗試一些方式試圖提升 mAP。

十四、 Try to Improve mAP

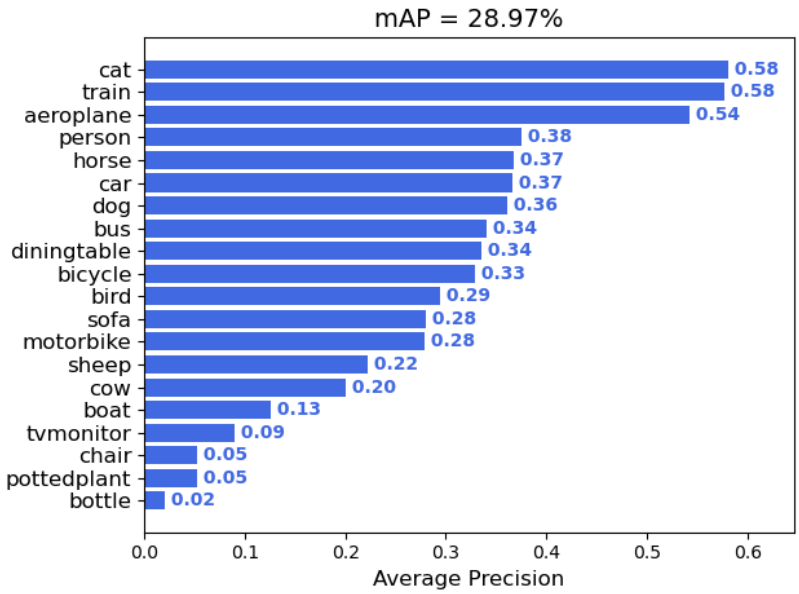
- Change backbone to ResNet50

  期末考預設是使用 RegNetX-400MF 作為 backbone network，我嘗試將他修改為 ResNet50。這部分主要使修改 common.py 的實現，我有將 code 貼到 B103012002_one_stage_detector.py 最後，修改部分主要是在一開始建立_cnn 的地方，改為呼叫 torchvision.model.resnet50，一樣去載入 pre-train weight。由於自己電腦顯卡記憶體不足，因此這部分是在 colab 做訓練。訓練 9000 epochs 結果如下

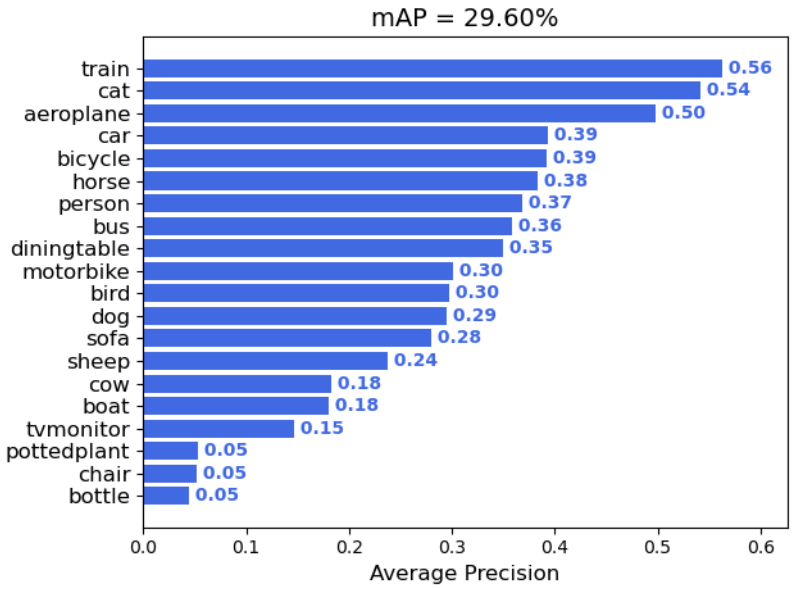
```
[Iter 8500][loss: 1.391][loss_cls: 0.330][loss_box: 0.566][loss_ctr: 0.495]
[Iter 8600][loss: 1.413][loss_cls: 0.296][loss_box: 0.499][loss_ctr: 0.619]
[Iter 8700][loss: 1.691][loss_cls: 0.311][loss_box: 0.685][loss_ctr: 0.695]
[Iter 8800][loss: 1.381][loss_cls: 0.233][loss_box: 0.531][loss_ctr: 0.617]
[Iter 8900][loss: 1.004][loss_cls: 0.200][loss_box: 0.381][loss_ctr: 0.422]
save drive/My Drive/Final_exam/onestage_ResNet10_v4.pt, loss : 0.5126364231109619 --> 0.49798691272735596
```



這邊我有修改訓練所使用的函數，設定讓程式自動儲存 loss 最低的模型。將訓練好的模型計算 mAP 結果如下



mAP 提升到 28.97 %，雖然有提升但是提升不多。從這樣的結果我猜測可能是訓練資料不足導致模型訓練效果不好，因此我嘗試做資料增生，這邊做了 color jitter、Gaussian blur 和 gray scale，最後的訓練資料量為原先的 3 倍，這部分的 code 我有貼到 B103012002_one_stage_detector.py 的最上面。增生後訓練結果如下



可以看到 mAP 提升的很有限，這邊我猜測可能我做的資料增生方式會導致原先圖片的特徵遭到破壞，導致模型學習效果不好，因此我覺得做資料增生在這次考試的效果會很有限，因為能使用的增生方式不多(要能夠保持圖片特徵不被破壞且預測方框位置不可以更動到)，導致說訓練資料量會很有限，再加上 one stage detector 在準確度表現上本來就很有限，因此我將架構換成 faster rcnn，並利用 transfer learning

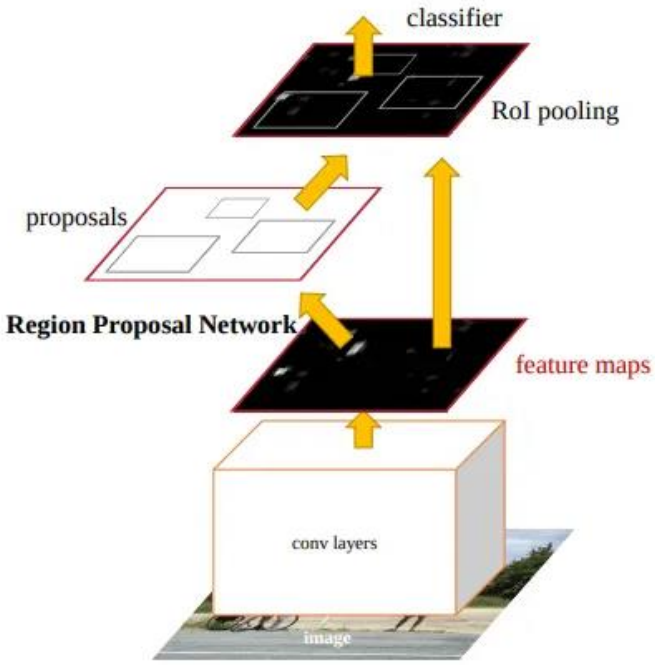
來解決資料量不足的問題。

- Faster RCNN transfer learning

  在 torchvision 中有 pre-train 好的 faster RCNN 可以直接使用，因此 transfer learning 挑戰的地方並不是在如何建立架構以及如何訓練，而是在資料處理部分。這邊會針對模型、資料處理、訓練、推論與模型評估做詳細解釋。

  1. 模型架構：

     模型架構部分我採用 torchvision 中的 fasterrcnn_resnet50_fpn 架構進行訓練，transfer learning pre-trained weight 我採用在 COCOv1 資料集上預訓練好的權重。Fasterrcnn_resnet50_fpn 的架構為參考論文"Faster R-CNN: Towards Real-Time Objection Detection with Region Proposal Network"所提出的架構，架構如下



     Backbone network 採用 ResNet50，且與期末考一樣採用 FPN 架構。由於我們採用的資料集與 COCOv1 所定義的類別數目不同，因此 head (prediction network)的部分會需要做修改。我們採用的資料集一共有 21 個類別，所以需要將原先針對 COCOv1 的 head 換成是分 21 個類別的 head，這部分可以用 torchvision 中的 FastRCNNPredictor 的做更換。

  2. 資料處理：

     從 torchvision 的 document 中可以看到 fasterrcnn_resnet50_fpn 有

預期要接收的輸入資料格式與輸出格式，規範如下

The input to the model is expected to be a list of tensors, each of shape `[C, H, W]`, one for each image, and should be in `0-1` range. Different images can have different sizes.

The behavior of the model changes depending on if it is in training or evaluation mode.

During training, the model expects both the input tensors and a targets (list of dictionary), containing:

- boxes (`FloatTensor[N, 4]`): the ground-truth boxes in `[x1, y1, x2, y2]` format, with `0 <= x1 < x2 <= W` and `0 <= y1 < y2 <= H`.
- labels (`Int64Tensor[N]`): the class label for each ground-truth box

The model returns a `Dict[Tensor]` during training, containing the classification and regression losses for both the RPN and the R-CNN.

During inference, the model requires only the input tensors, and returns the post-processed predictions as a `List[Dict[Tensor]]`, one for each input image. The fields of the `Dict` are as follows, where `N` is the number of detections:

- boxes (`FloatTensor[N, 4]`): the predicted boxes in `[x1, y1, x2, y2]` format, with `0 <= x1 < x2 <= W` and `0 <= y1 < y2 <= H`.
- labels (`Int64Tensor[N]`): the predicted labels for each detection
- scores (`Tensor[N]`): the scores of each detection

期末考所提供的 dataloader 並不符合這樣的格式，因此所有的 code 都需要更改，這部分的 code 都在 B103012002_one_stage_detector.ipynb 最後 Faster RCNN Transfer Learning 中，這邊會依序解釋所有自己寫的函數。

I. VOC_CLASSES：為一個 encoder，將 PASCAL VOC 資料集編碼成 ID。
II. get_transform：將圖片轉換成 pytorch tensor，並做 normalizaton。
III. collate_fn：使用 pytorch 的 DataLoader 來載入資料的時候，DataLoader 會將資料打包成 batch，一般情況下 DataLoader 會將每個 batch 中所以樣本組合成一個矩陣。如果資料的樣本形狀都相同的話，這樣的方法使可行的，但是對於物件偵測來說，每個圖片中的物體數量與尺寸都可能不同，因此需要自行定義一個函數來解決這個問題。此函數會接收一個含有 tuple 的 list，並回傳一個包含兩個 tuple，images 和 targets 的 tuple。
IV. prepare_targets：此函數主要是將我們的資料變成符合 fasterrcnn_resnet50_fpn 的輸入格式。這部分會需要去解析一

下 PASCAL VOC 資料集的資料結構，其資料結構如下

```
target = {
    'annotation': {
        'object': [
            {
                'name': 'cat',
                'bndbox': {
                    'xmin': '48',
                    'ymin': '240',
                    'xmax': '195',
                    'ymax': '371'
                }
            },
            ...
        ]
    }
}
```

target 是圖像的標記數據，根據 fasterrcnn_resnet50_fpn 輸入資料規範，我們需要整理出 boxes 和 label。Boxes 所需要的資料都在'bndbox'中，因此可以透過 dictionary key 來取得資訊，並整理成 0 <= x1 <= x2、0<= y1<= y2 的格式。Label 的部分也是透過 dictionary key 來取得，再透過一開始的 encoder 將類別轉換為 ID。。

V. VOCDataset： 定義一個繼承 torchvision.datasets.VOCDetection 的類別，用來載入訓練資料。此類別初始化繼承 torchvision.datasets.VOCDetection 的方法，需要傳入資料路徑、資料年份(2007、2012)和資料類型(train、validation、test)。此外也需要初始化資料預處理的方法(transform)。Getter 的部分先透過 torchvision.datasets.VOCDetection 的 getter 取得圖片和標記數據，並透過 prepare_targets 將標記數據轉換為輸入格式，同時透過 transform 對圖片做預處理。最後將處理好的圖片與標記數據回傳。

VI. train_model : train_model 需要傳入待訓練模型、data_loader、optimizer、device 和 epochs 數量。一開始需要將 model 轉換為訓練模式，也就是開啟 batch normalization 和 dropout。接著利用 for loop 迭代每一次 epochs。每一次 epochs 都先從

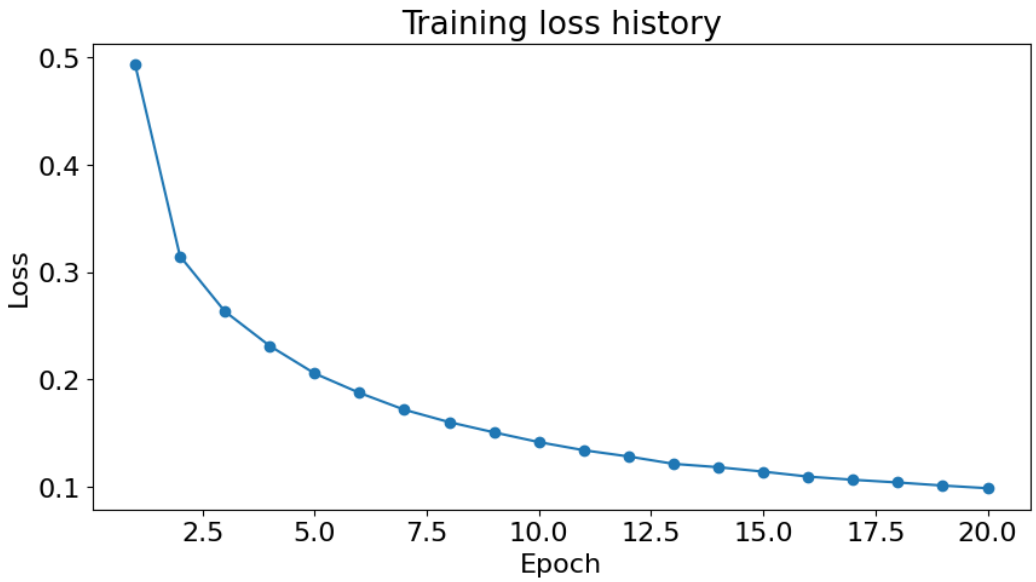
data_loader 中取的圖片與標記數據，接著去計算 loss 並根據 optimizer 做優化。計算當前 epochs 的 average loss 然後將這些資訊 print 出來並儲存到 loss history 中。最後再根據 loss history 繪製 training loss curve。

VII. main： 主要運行的程式碼。一開始要先載入資料，與期末考一樣採用 PASCAL VOC2007 training set 做訓練，optimizer 採用 SGD (lr = 0.001, momentum = 0.9, weight_decay = 0.0005)，訓練 20 個 epochs。

3. 訓練 ：

訓練結果如下

```
Epoch 1/20, Loss: 0.4931784912461428
Epoch 2/20, Loss: 0.314856212854171
Epoch 3/20, Loss: 0.26345446904440767
Epoch 4/20, Loss: 0.23134926639253098
Epoch 5/20, Loss: 0.2057605250556168
Epoch 6/20, Loss: 0.18764380154182775
Epoch 7/20, Loss: 0.17194661611376955
Epoch 8/20, Loss: 0.1604003270717262
Epoch 9/20, Loss: 0.15086658080543974
Epoch 10/20, Loss: 0.14180298681578618
Epoch 11/20, Loss: 0.1342453495975283
Epoch 12/20, Loss: 0.12843582155458408
Epoch 13/20, Loss: 0.12157484755176101
Epoch 14/20, Loss: 0.11844224698073287
Epoch 15/20, Loss: 0.11433178397843972
Epoch 16/20, Loss: 0.10969616859097132
Epoch 17/20, Loss: 0.10673939159138501
Epoch 18/20, Loss: 0.10422633613104658
Epoch 19/20, Loss: 0.10136989358928707
Epoch 20/20, Loss: 0.09887775052249503
```

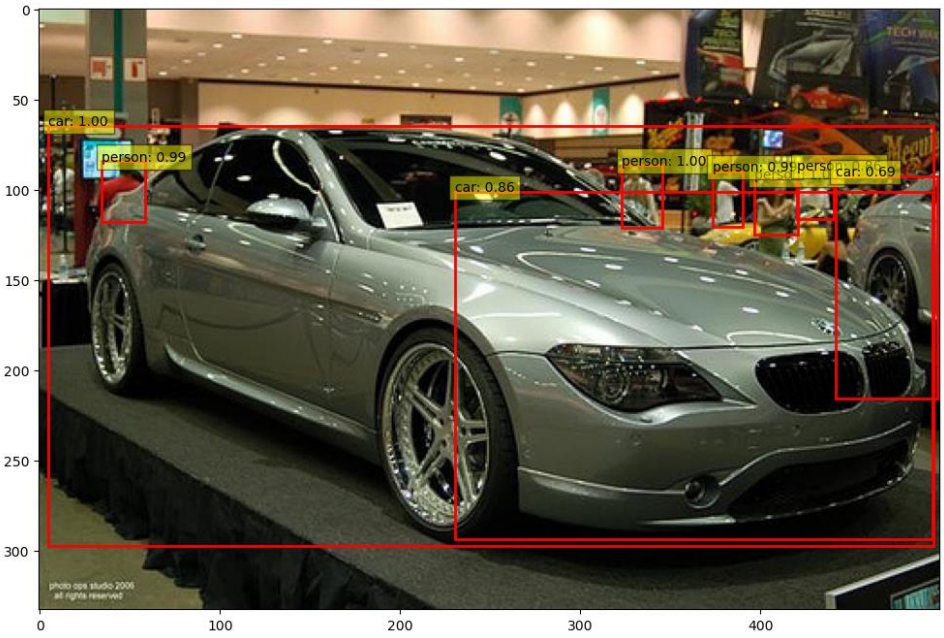


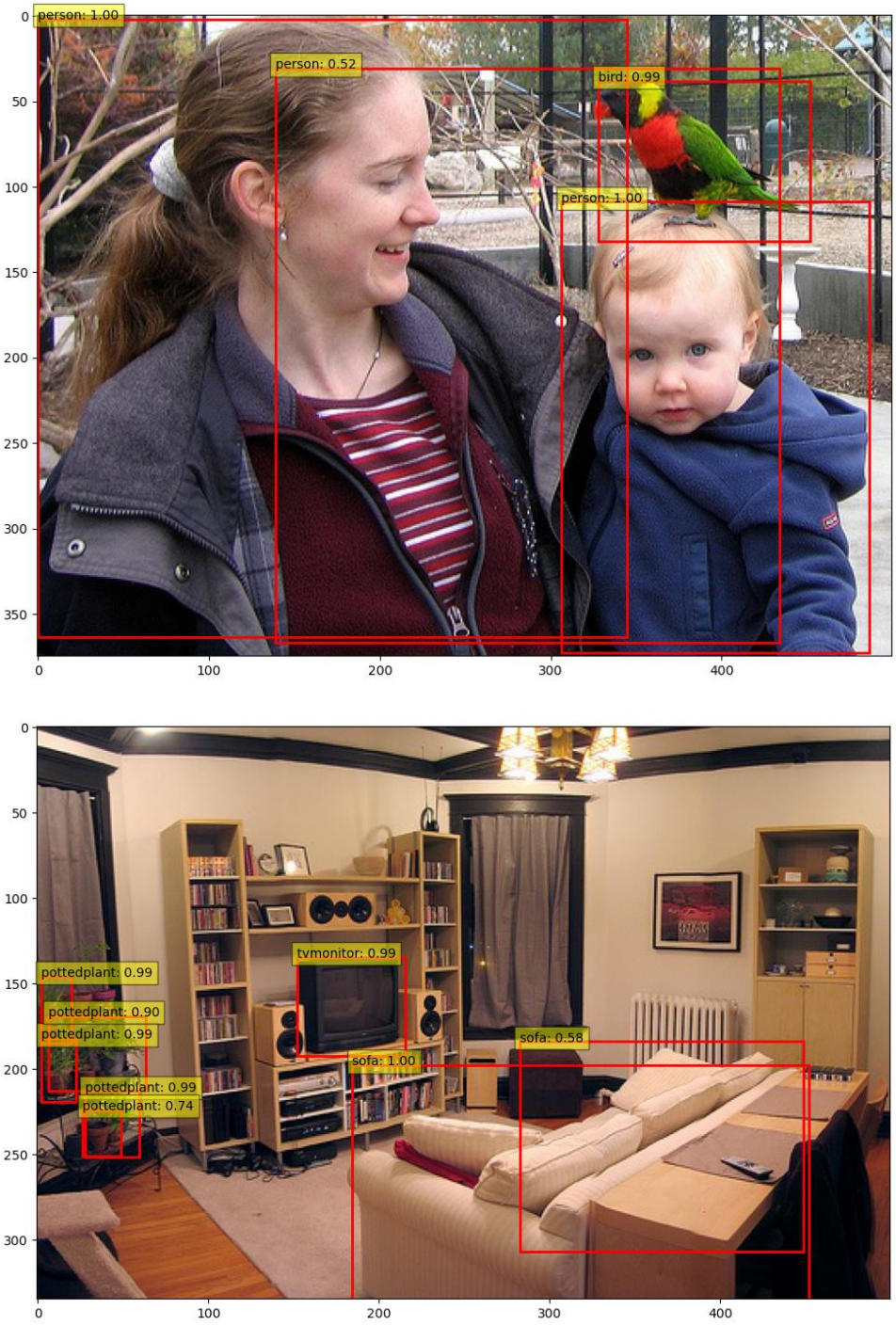
4. 推論與模型評估 ：

這部分也須根據 fasterrcnn_resenet50_fpn 的輸出格式定義 inference with detector。

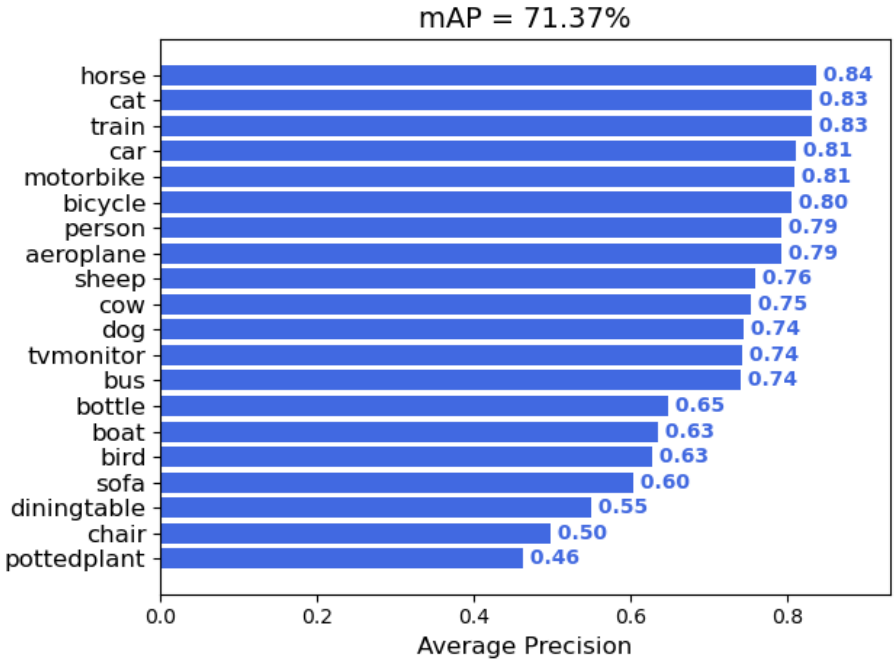
Inference with detector 主要是用來產生計算 mAP 所需要的資料。首先須將預處理的圖片轉換回來，並設置好儲存 ground truth 與 detection result 的資料夾路徑。接著利用 for loop 迭代測試資料中每一張圖片，利用訓練好的模型做偵測。這邊使用的資料與期末考使用的資料相同，都是 PASCAL VOC2007 validation set。接著透過 fasterrcnn_resnet50_fpn 的輸出資料結構取得 boxes、scores 和 label，並過濾掉信心度過低的方框(threshold 與期末考的數值同為 0.4)。關於模型處理重複方框的方法，torchvision fasterrcnn_resnet50_fpn 也是採用 NMS，預設 threshold 為 0.5，與期末考的設置都相同。

部分預測圖片結果如下





Faster RCNN transfer learning 的 mAP 表現如下



只需要訓練 20 epochs mAP 就可以提升到 71.37 %，與期末考實現的 one stage detector 的 23.53 %相比提升了 47.84 %。

十五、 Reference

[1] OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. https://chat.openai.com/

[2] Ren et al, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks” https://arxiv.org/abs/1506.01497

[3] PyTorch “fasterrcnn_resnet50_fpn” https://pytorch.org/vision/main/models/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn.html#torchvision.models.detection.fasterrcnn_resnet50_fpn

[4] Lin et al, “Feature Pyramid Networks for Object Detection” https://arxiv.org/abs/1612.03144