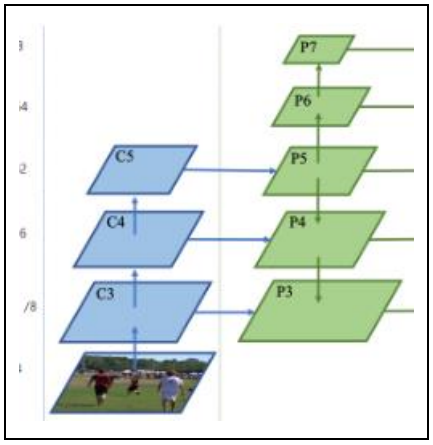
Final One-Stage Object Detector

M113040105 劉東霖

I. Implementing Backbone and Feature Pyramid Network

依據下圖的方式來實作:



DetectorBackboneWithFPN 的__INIT__主要是定義 c3 和 c4 和 c5 的卷積層。dummy_out_shape 主要的格式由 level name(c3,c4,c5)和 feature shape(每個 level name 對應的 shape)組成，下圖為每個 level 對應的 feature 。

```
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])
```

程式實作的方式如下圖:

1. 根據 dummy_out_shapes 中的形狀信息，初始化 FPN 1*1 的 Conv 層。這些 1*1 的 Conv 層將根據輸入的特徵形狀進行初始化。程式碼如下:

```
self.fpn_params = nn.ModuleDict()

# Replace "pass" statement with your code
self.fpn_params['conv5'] = nn.Conv2d(dummy_out['c5'].shape[1], out_channels, 1)
self.fpn_params['conv4'] = nn.Conv2d(dummy_out['c4'].shape[1], out_channels, 1)
self.fpn_params['conv3'] = nn.Conv2d(dummy_out['c3'].shape[1], out_channels, 1)
```

2. 定義 FPN 3*3 的輸出 Conv 層，這些層將輸出 FP 特徵金字塔的最終特徵。程式碼如下:

```
self.fpn_params['conv_out5'] = nn.Conv2d(out_channels, out_channels, 3, stride=1, padding=1)
self.fpn_params['conv_out4'] = nn.Conv2d(out_channels, out_channels, 3, stride=1, padding=1)
self.fpn_params['conv_out3'] = nn.Conv2d(out_channels, out_channels, 3, stride=1, padding=1)
```

金字塔的模型架構如下:

```
Extra FPN modules added:
ModuleDict(
  (conv5): Conv2d(400, 64, kernel_size=(1, 1), stride=(1, 1))
  (conv4): Conv2d(160, 64, kernel_size=(1, 1), stride=(1, 1))
  (conv3): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1))
  (conv_out5): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv_out4): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv_out3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(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])
```

DetectorBackboneWithFPN 的 forward 主要是實作 upsample 的部分

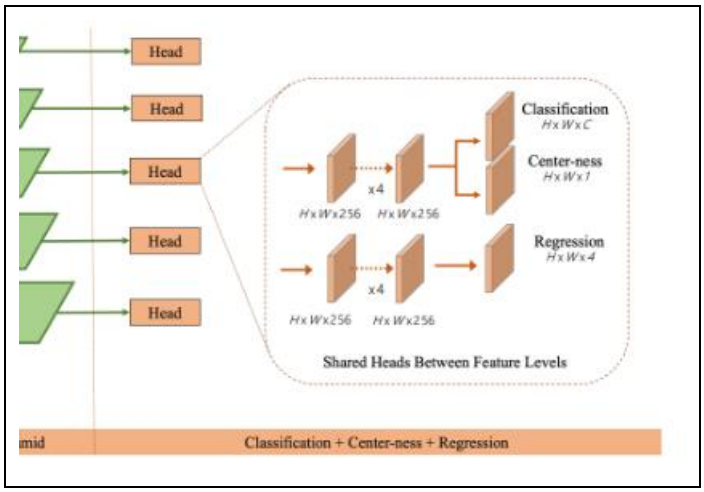
1. 將"c5"特徵通過 self.fpn_params['conv5'] 得到 out5。
2. 使用 upsample F.interpolate 將 out5 變成跟 c4 圖片的大小一樣，插值的模式使用 nearest，得到 scale5。
3. 將"c4"特徵通過 self.fpn_params['conv4'] ，並加上 scale5 得到 out4。
4. 使用 upsample 將 out4 變成跟 c3 圖片的大小一樣，插值的模式使用 nearest，得到 scale4。
5. 將"c3"特徵通過 self.fpn_params['conv3'] ，並加上 scale4 得到 out3。
6. fpn_feats 字典中的"p5"、"p4"和 "p3"特徵將分別是 self.fpn params['conv_out5'](out5)、self.fpn_params['conv_out4'](out4) 和 self.fpn_params['conv_out3'](out3) 的輸出。

程式碼如下:

```
out5=self.fpn_params['conv5'](backbone_feats["c5"])
scale5=F.interpolate(out5,size=(backbone_feats["c4"].shape[2],  backbone_feats["c4"].shape[3]),mode="nearest")
out4=self.fpn_params['conv4'](backbone_feats["c4"])+scale5
scale4=F.interpolate(out4,size=(backbone_feats["c3"].shape[2],  backbone_feats["c3"].shape[3]),  mode="nearest")
out3=self.fpn_params['conv3'](backbone_feats["c3"])+scale4

fpn_feats["p5"]  =  self.fpn_params['conv_out5'](out5)
fpn_feats["p4"]  =  self.fpn_params['conv_out4'](out4)
fpn_feats["p3"]  =  self.fpn_params['conv_out3'](out3)
```

II. Implementing FCOS prediction network (head)

這裡主要分為兩個部分，第一個部分為預測 class 的部分，第二個部分為預測 box 位置的部分。



FCOSPredictionNetwork __init__:

1. 創立預測 class 的第一個卷積層，輸入為 in channel，輸出通道數為 stem_channel[0]，kernel size 為 3，stride 和 padding 都是 1，並把他 append 到 stem_cls 裡面。再 append 一個 relu 到 stem_cls 裡。
2. 創立預測 box 的第一個卷積層，輸入為 in channel，輸出通道數為 stem_channel[0]，kernel size 為 3，stride 和 padding 都是 1，並把他 append 到 stem_box 裡面。再 append 一個 relu 到 stem_box 裡。
3. 用 for 循環剩下的卷積層和 relu
4. 用 sequential 把 stem_cls 和 stem_box 分別存入 self.stem_cls 和 self.stem_box。
5. 定義 self.pred_cls 為一個卷積層，目的為預測 class。輸入為 stem_channel 的最後一層，輸出為 class 數目，kernel size 為 3，stride 和 padding 都是 1。
6. 定義 self.pred_box 為一個卷積層，目的為預測 box。輸入為 stem_channel 的最後一層，輸出為 4(中心點位置和長寬)，kernel size 為 3，stride 和 padding 都是 1。
7. 定義 self.pred_ctr 為一個卷積層，目的為預測 centerness。輸入為 stem_channel 的最後一層，輸出為 1，kernel size 為 3，stride 和 padding 都是 1。

程式碼如下，題目有要求卷積層的權重為常態分佈，平均值為 0，標準差為 0.01:

```
stem_cls = []
stem_box = []
# Replace "pass" statement with your code
conv1=nn.Conv2d(in_channels,stem_channels[0],3,1,1)
nn.init.normal_(conv1.weight,mean=0,std=0.01)
nn.init.zeros_(conv1.bias)
stem_cls.append(conv1)
stem_cls.append(nn.ReLU())

conv2=nn.Conv2d(in_channels,stem_channels[0],3,1,1)
nn.init.normal_(conv2.weight,mean=0,std=0.01)
nn.init.zeros_(conv2.bias)
stem_box.append(conv2)
stem_box.append(nn.ReLU())
for i in range(len(stem_channels)-1):
    conv1=nn.Conv2d(stem_channels[i],stem_channels[i+1],3,1,1)
    nn.init.normal_(conv1.weight,mean=0,std=0.01)
    nn.init.zeros_(conv1.bias)
    stem_cls.append(conv1)
    stem_cls.append(nn.ReLU())

    conv2=nn.Conv2d(stem_channels[i],stem_channels[i+1],3,1,1)
    nn.init.normal_(conv2.weight,mean=0,std=0.01)
    nn.init.zeros_(conv2.bias)
    stem_box.append(conv2)
    stem_box.append(nn.ReLU())
# Wrap the layers defined by student into a `nn.Sequential` module:
self.stem_cls = nn.Sequential(*stem_cls)
self.stem_box = nn.Sequential(*stem_box)
```

```
conv_cls=nn.Conv2d(stem_channels[-1],num_classes,3,1,1)
nn.init.normal_(conv_cls.weight,mean=0,std=0.01)
nn.init.zeros_(conv_cls.bias)
self.pred_cls=nn.Conv2d(stem_channels[-1],num_classes,3,1,1)

conv_box=nn.Conv2d(stem_channels[-1],4,3,1,1)
nn.init.normal_(conv_box.weight,mean=0,std=0.01)
nn.init.zeros_(conv_box.bias)
self.pred_box=conv_box

conv_ctr=nn.Conv2d(stem_channels[-1],1,3,1,1)
nn.init.normal_(conv_ctr.weight,mean=0,std=0.01)
nn.init.zeros_(conv_ctr.bias)
self.pred_ctr=conv_ctr
```

FCOSPredictionNetwork forward:

1. 用一個 for 把每一層抓出來。
2. 把當前那一層經過 self.stem_cls 卷積層分類，並經過 self.pred_cls 預測類別，最後再 reshape 成(b,h*w,num_classes)，b 和 h*w 和 num_classes 分別為 batch size 和長乘寬和 class 數目。
3. 把當前那一層經過 self.stem_box 卷積層，並經過 self.pred_box 找出 box 資訊，最後再 reshape 成(b,h*w,4)，b 和 h*w 分別為 batch size 和長乘寬。
4. 把當前那一層經過 self.stem_cls 卷積層，並經過 self.pred_ctr 找出 centerness 資訊，最後再 reshape 成(b,h*w,1)，b 和 h*w 分別為 batch size 和長乘寬。

程式碼如下:

```
class_logits = {}
boxreg_deltas = {}
centerness_logits = {}

# Replace "pass" statement with your code
for i in ['p3','p4','p5']:
    pred=self.pred_cls(self.stem_cls(feats_per_fpn_level[i]))
    b,c,h,w=pred.shape
    class_logits[i]=pred.reshape(b,-1,h*w).transpose(1,2)
    boxreg_deltas[i]=self.pred_box(self.stem_box(feats_per_fpn_level[i])).reshape(b,-1,h*w).transpose(1,2)
    centerness_logits[i]=self.pred_ctr(self.stem_cls(feats_per_fpn_level[i])).reshape(b,-1,h*w).transpose(1,2)
```

head 的模型架構如下:

```
FCOS prediction network parameters:
FCOSPredictionNetwork(
  (stem_cls): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
  )
  (stem_box): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
  )
  (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])
```

III. get_fpn_location_coords

這裡主要是將特徵金字塔 feature 裡的每個位置映射到圖像上的點，這樣做是為了在不同層級與正確的 box 之間實現統一座標的表示。

get_fpn_location_coords 主要寫的內容如下:

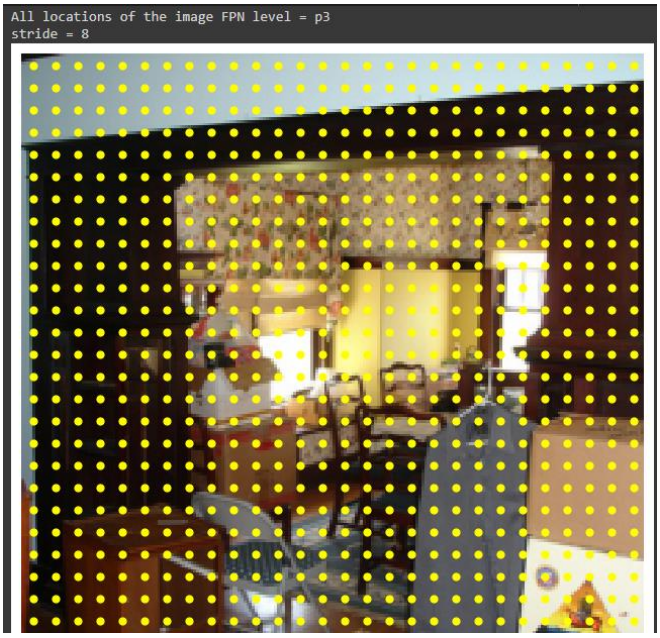
1. 創立一個空 tensor location，大小為(h*w,2) ，用於儲存當前層級的位置座標。
2. 使用兩個 for 走整張圖。
3. location 的第 0 個和第 1 個內容為每一個 receptive field 中心點的座標。
4. 把每一層的 location 存到 location_coords 裡面。

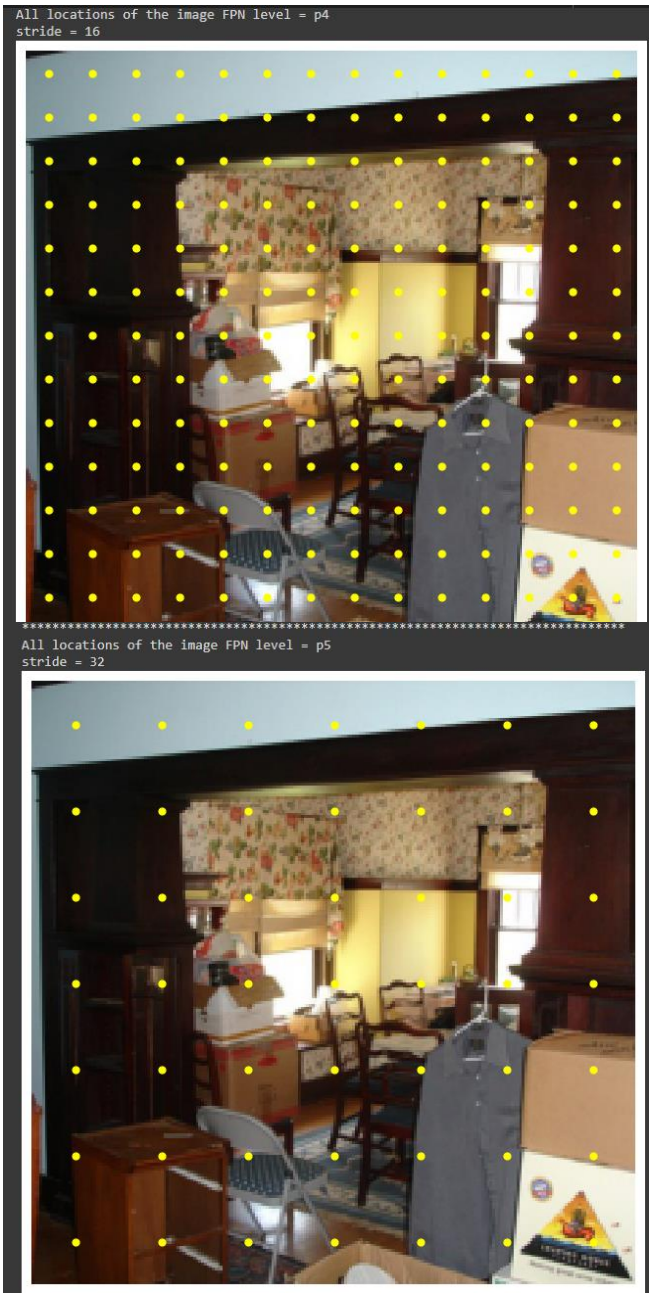
程式碼如下:

```
for level_name, feat_shape in shape_per_fpn_level.items():
        level_stride = strides_per_fpn_level[level_name]

        ###########################################################################
        # TODO: Implement logic to get location co-ordinates below.               #
        ###########################################################################
        # Replace "pass" statement with your code
        b,c,h,w=feat_shape
        location=torch.zeros(size=(h*w,2),dtype=dtype,device=device)
        for i in range(h):
            for j in range(w):
                location[i*w+j][0],location[i*w+j][1]=level_stride*(i+0.5),level_stride*(j+0.5)
        location_coords[level_name]=location
        ###########################################################################
        #                                                      END OF YOUR CODE
        ###########################################################################
```

輸出的誤差和每一個位置的中心點座標如下:

```
First five locations per FPN level (absolute image co-ordinates):
p3: [[4.0, 4.0], [4.0, 12.0], [4.0, 20.0], [4.0, 28.0], [4.0, 36.0]]
rel error:  0.0
p4: [[8.0, 8.0], [8.0, 24.0], [8.0, 40.0], [8.0, 56.0], [8.0, 72.0]]
rel error:  0.0
p5: [[16.0, 16.0], [16.0, 48.0], [16.0, 80.0], [16.0, 112.0], [16.0, 144.0]]
rel error:  0.0
********************************************************************************
```





IV. GT Targets for box regression and centerness regression

1. fcos_get_deltas_from_locations

這裡主要是計算特徵位置到 ground truth 邊緣的距離 delta，實作過程如下:

- 如果 ground truth 框的 shape 是(N,5)的話，其最後一個值為背景標誌，剩下 4 個為(left,top,right,bottom)。如果背景標誌為-1，代表他是背景，將 delta 設為(-1,-1,-1,-1)。否則，delta[0]=左邊緣到特徵位置的水平距離除以步長，delta[1]=上邊緣到特徵位置的垂直距離除以步長，delta[2]=右邊緣到特徵位置的水平距離除以步長，delta[3]=下邊緣到特徵位置的垂直距離除以步長。

- 如果 ground truth 框的 shape 是(N,4)的話，這 4 個值分別為(left,top,right,bottom)。如果其中一個為-1，代表他是背景，將 delta 設為(-1,-1,-1,-1)。否則，跟前一個計算的內容一樣

程式碼如下:

```
deltas = torch.zeros(locations.shape[0], 4, device=locations.device)
xc,yc=locations[:,0],locations[:,1]
x1,y1,x2,y2=gt_boxes[:,0],gt_boxes[:,1],gt_boxes[:,2],gt_boxes[:,3]

l=(xc-x1)/stride
t=(yc-y1)/stride
r=(x2-xc)/stride
b=(y2-yc)/stride
if(gt_boxes.shape[1]==5):
    background= gt_boxes[:,4]
else:
    background=gt_boxes[:,0]
deltas[:,0],deltas[:,1],deltas[:,2],deltas[:,3]=l,t,r,b
deltas[background==-1,:] = -1
```

2. fcos_apply_deltas_to_locations

這裡的目的是根據 delta 和 FPN 特徵位置來獲取最終的邊界框座標。

- 首先，從 location 提取中心點位置，然後對 delta 進行檢查，如果為負的，將他裁減為 0。
- 再來計算 box 的位置。箱子的左上角的 x 座標為中心點 x 減掉 left*stride，左上角的 y 座標為中心點 y 減掉 top*stride，右下角的 x 座標為中心點加 right*stride，右下角的 y 座標為中心點 y 加 bottom*stride。

程式碼如下:

```
output_boxes=torch.zeros_like(deltas)
xc,yc=locations[:,0],locations[:,1]
l,t,r,b=deltas[:,0],deltas[:,1],deltas[:,2],deltas[:,3]

l = torch.clamp(l, min=0)
t = torch.clamp(t, min=0)
r = torch.clamp(r, min=0)
b = torch.clamp(b, min=0)

x1=xc-l*stride
y1=yc-t*stride
x2=xc+r*stride
y2=yc+b*stride

output_boxes[:,0],output_boxes[:,1],output_boxes[:,2],output_boxes[:,3]=x1,y1,x2,y2
```

3. fcos_make_centerness_targets

依據下圖的公式，把中心點的程度算出來:

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

程式碼如下，並把 gt boxes 為背景的設為-1:

```
centerness = None
# Replace "pass" statement with your code
l,t,r,b=deltas[:,0],deltas[:,1],deltas[:,2],deltas[:,3]
centerness=torch.sqrt(torch.min(l,r)*torch.min(t,b)/torch.max(l,r)/torch.max(t,b))
centerness[l==-1]=-1
```

程式執行結果如下:

```
import torch

from one_stage_detector import fcos_get_deltas_from_locations, fcos_apply_deltas_to_locations

# Three hard-coded input boxes and three points lying inside them.
# Add a dummy class ID = 1 indicating foreground
input_boxes = torch.Tensor(
        [[10, 15, 100, 115, 1], [30, 20, 40, 30, 1], [120, 100, 200, 200, 1]]
)
input_locations = torch.Tensor([[30, 40], [32, 29], [125, 150]])

# Here we do a simple sanity check - getting deltas for a particular set of boxes
# and applying them back to centers should give us the same boxes. Setting a random
# stride = 8, it should not affect reconstruction if it is same on both sides.
_deltas = fcos_get_deltas_from_locations(input_locations, input_boxes, stride=8)
output_boxes = fcos_apply_deltas_to_locations(_deltas, input_locations, stride=8)

print("Rel error in reconstructed boxes:", rel_error(input_boxes[:, :4], output_boxes))


# Another check: deltas for GT class label = -1 should be -1.
background_box = torch.Tensor([[10, 15, 100, 115, -1]])
input_location = torch.Tensor([[100, 200]])

_deltas = fcos_get_deltas_from_locations(input_location, background_box, stride=8)
output_box = fcos_apply_deltas_to_locations(_deltas, input_location, stride=8)

print("Background deltas should be all -1       :", _deltas)
print("Output box should simply be the center:", output_box)
```

```
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.]])
```

```
[18] import torch

     from one_stage_detector import fcos_make_centerness_targets

     # Three hard-coded input boxes and three points lying inside them.
     # Add a dummy class ID = 1 indicating foreground
     input_boxes = torch.Tensor(
             [
                     [10, 15, 100, 115, 1],
                     [30, 20, 40, 30, 1],
                     [-1, -1, -1, -1, -1]    # background
             ]
     )
     input_locations = torch.Tensor([[30, 40], [32, 29], [125, 150]])

     expected_centerness = torch.Tensor([0.30860671401, 0.1666666716, -1.0])

     _deltas = fcos_get_deltas_from_locations(input_locations, input_boxes, stride=8)
     centerness = fcos_make_centerness_targets(_deltas)
     print("Rel error in centerness:", rel_error(centerness, expected_centerness))
```

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

V. Object detection module

FCOS 的__init__主要是初始化骨幹和預測分支的部分，程式碼如下:

```
# Feel free to delete these two lines: (but keep variable names same)
self.backbone = None
self.pred_net = None
# Replace "pass" statement with your code
self.backbone=DetectorBackboneWithFPN(out_channels=fpn_channels)
self.pred_net=FCOSPredictionNetwork(num_classes,fpn_channels,stem_channels)
```

再來我把 FCOS forward 需要實作的部分寫出來。

1. 把 images 放入 backbone 得到特徵，在把特徵帶入 pred_net 得到 pred_cls_logits 和 pred_boxreg_deltas 和 pred_ctr_logits。程式碼如下:

```
pred_cls_logits, pred_boxreg_deltas, pred_ctr_logits = None, None, None
# Replace "pass" statement with your code
fpn_feat=self.backbone(images)
pred_cls_logits, pred_boxreg_deltas, pred_ctr_logits=self.pred_net(fpn_feat)
```

2. 把 fpn_feat 裡頭的名字和特徵的形狀存入一個字典裡，再用 get_fpn_location_coords 來獲取每個 FPN 層級中位置的絕對座標(xc, yc)。程式碼如下:

```
shape_per_fpn_level={}
for name,feat in fpn_feat.items():
    shape_per_fpn_level[name]=feat.shape
locations_per_fpn_level=get_fpn_location_coords(shape_per_fpn_level,self.backbone.fpn_strides,device=images.device)
```

3. 再來要將 ground truth box 分配給 feature locations。我們先把 locations_per_fpn_level 和每一個 ground truth box 和每一特徵層的 strides self.backbone.fpn_strides 帶入 fcos_match_locations_to_gt 來讓特徵位置(feature location)和 ground truth box 進行比對，以確定那些特徵位置該被視為正樣本或副樣本，並用一個 list 存起來。程式碼如下:

```
matched_gt_boxes = []
# Replace "pass" statement with your code
for gt_contain in gt_boxes:
    matched_gt_boxes.append(fcos_match_locations_to_gt(locations_per_fpn_level,self.backbone.fpn_strides,gt_contain))
```

4. 計算每一層的特徵位置到 ground truth 邊緣的距離 delta，並用一個字典存起來，為了分辨每一層的距離。最後，再把字典裡的內容 append 到一個 list。

```
for i in range(gt_boxes.shape[0]):
    gt_delta_dict = {}
    for key in locations_per_fpn_level.keys():
        gt_delta_dict[key] = fcos_get_deltas_from_locations(locations_per_fpn_level[key], matched_gt_boxes[i][key], self.backbone.fpn_strides[key])

    matched_gt_deltas.append(gt_delta_dict)
```

5. 計算 loss_cls、loss_box 和 loss_ctr。這些 loss 用於訓練 FCOS 目標檢測模型，以指導模型學習正確分類和精確預測邊界框的能力。

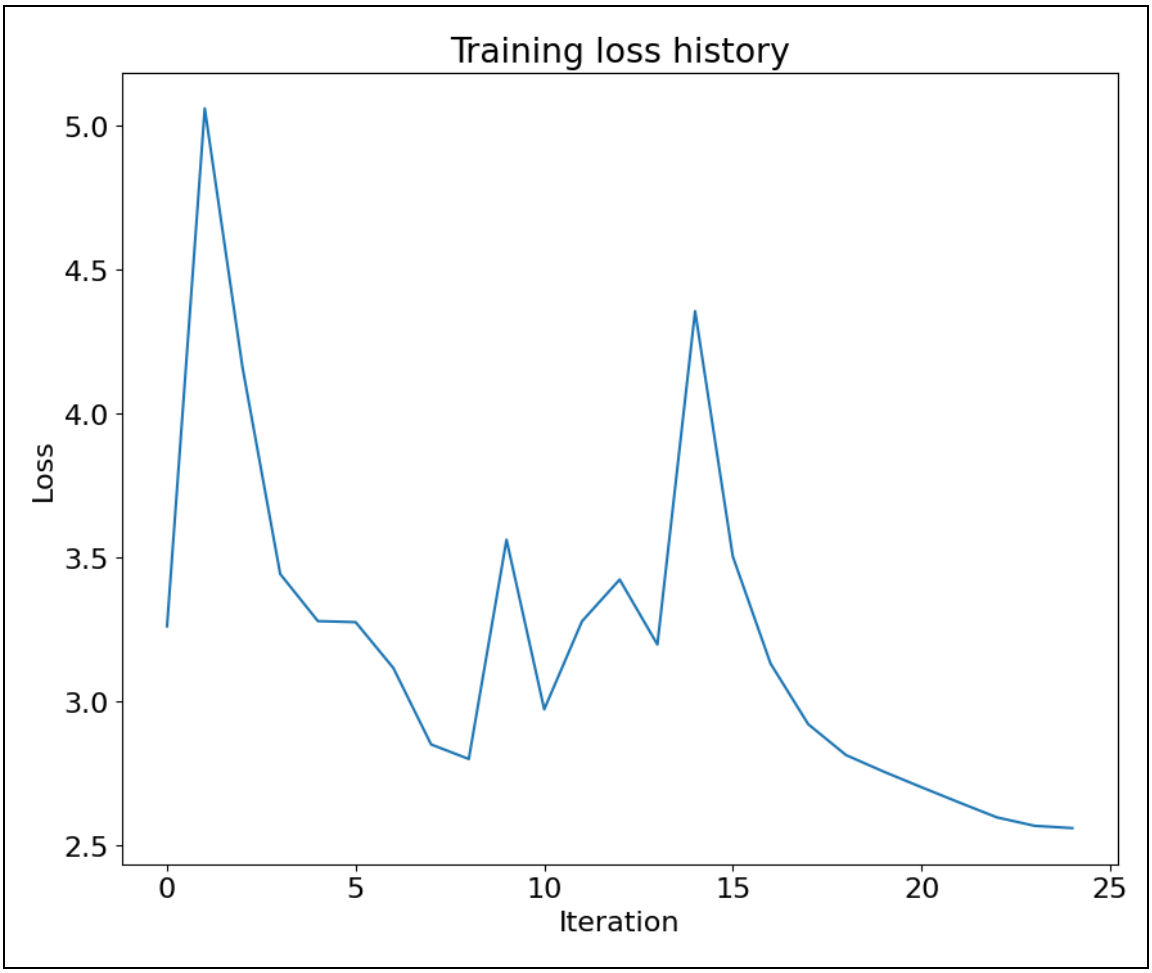
- Loss_cls: 首先，從目標物體的 ground truth 中提取類別標籤，並將其轉換為 one-hot 編碼的形式。然後，使用 sigmoid_focal_loss 去計算 loss_cls。
- Loss_box:把 pred_boxreg_deltas 和 matched_gt_deltas 帶入 l1 loss 計算出 Loss_box，並把 delta<0 的變成 0。
- Loss_ctr:把 pred_ctr_deltas 和 centerness 帶入 binary cross entropy with logits 計算出 Loss_ctr，並把 centerness<0 的變成 0。

程式碼如下:

```
loss_cls, loss_box, loss_ctr = None, None, None

gt_classes = matched_gt_boxes[:, :, 4].long() + 1
gt_classes = F.one_hot(gt_classes, num_classes=self.num_classes + 1)
gt_classes = gt_classes[:, :, 1:]

# cls loss
loss_cls = sigmoid_focal_loss(pred_cls_logits, gt_classes.float())

# box loss
loss_box = 0.25 * F.l1_loss(pred_boxreg_deltas, matched_gt_deltas, reduction="none")
loss_box[matched_gt_deltas < 0]=0.0

# ctr loss
centerness = fcos_make_centerness_targets(matched_gt_deltas.view(-1, 4))
loss_ctr = F.binary_cross_entropy_with_logits(pred_ctr_logits.flatten(), centerness, reduction="none")
loss_ctr[centerness < 0] = 0.0
```

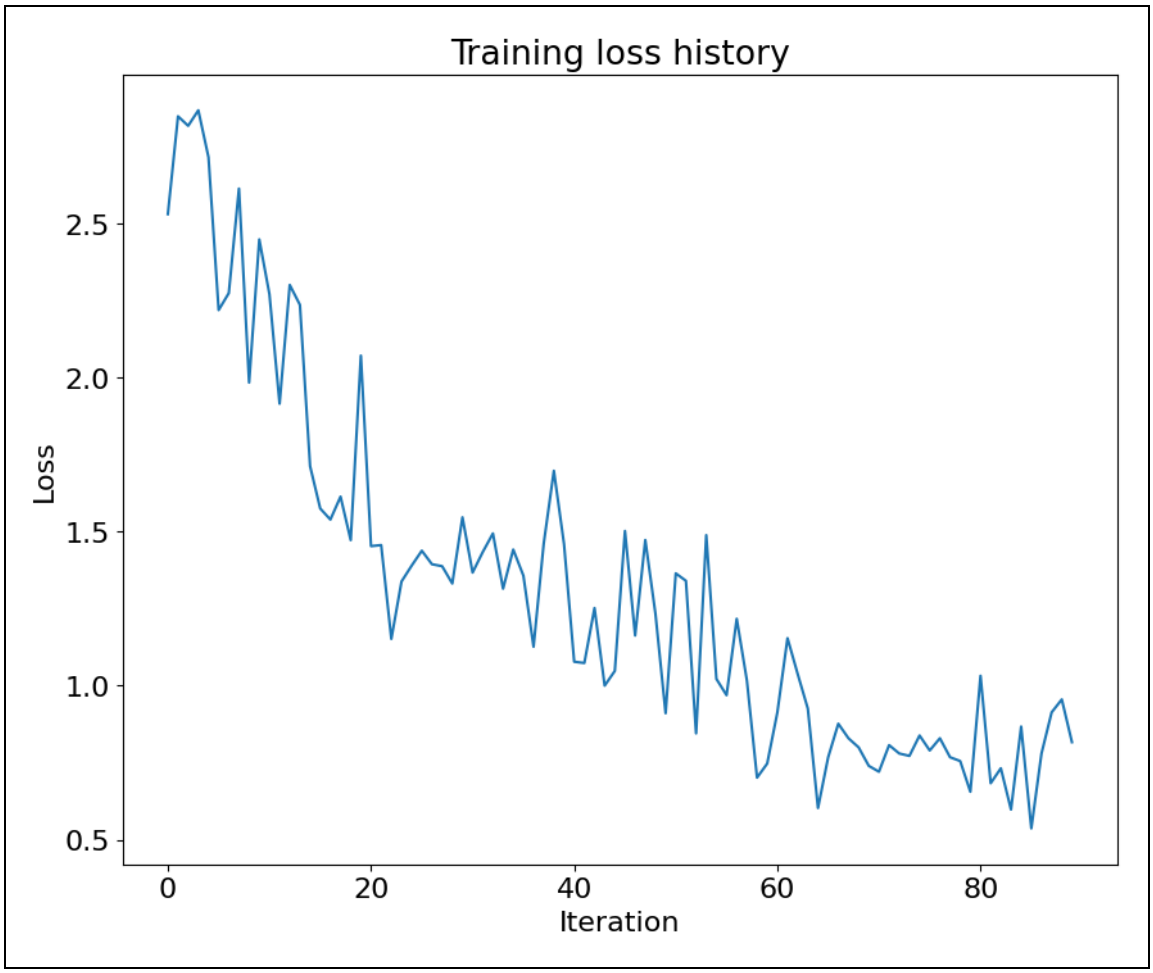
寫完 FCOS 該寫得之後，overfit small data，結果如下圖:

```
[Iter 300][loss: 3.504][loss_cls: 0.856][loss_box: 1.917][loss_ctr: 0.730]
[Iter 320][loss: 3.132][loss_cls: 0.798][loss_box: 1.623][loss_ctr: 0.711]
[Iter 340][loss: 2.921][loss_cls: 0.642][loss_box: 1.572][loss_ctr: 0.707]
[Iter 360][loss: 2.814][loss_cls: 0.585][loss_box: 1.524][loss_ctr: 0.704]
[Iter 380][loss: 2.756][loss_cls: 0.572][loss_box: 1.482][loss_ctr: 0.702]
[Iter 400][loss: 2.702][loss_cls: 0.565][loss_box: 1.438][loss_ctr: 0.700]
[Iter 420][loss: 2.649][loss_cls: 0.556][loss_box: 1.395][loss_ctr: 0.698]
[Iter 440][loss: 2.597][loss_cls: 0.549][loss_box: 1.352][loss_ctr: 0.696]
[Iter 460][loss: 2.568][loss_cls: 0.545][loss_box: 1.328][loss_ctr: 0.695]
[Iter 480][loss: 2.560][loss_cls: 0.544][loss_box: 1.322][loss_ctr: 0.694]
```



再來 train 整個 data，總共要訓練 9000 個 iteration，結果如下:

```
[Iter 8100][loss: 0.683][loss_cls: 0.038][loss_box: 0.148][loss_ctr: 0.497]
[Iter 8200][loss: 0.732][loss_cls: 0.039][loss_box: 0.135][loss_ctr: 0.558]
[Iter 8300][loss: 0.598][loss_cls: 0.036][loss_box: 0.146][loss_ctr: 0.416]
[Iter 8400][loss: 0.867][loss_cls: 0.050][loss_box: 0.223][loss_ctr: 0.595]
[Iter 8500][loss: 0.537][loss_cls: 0.032][loss_box: 0.123][loss_ctr: 0.382]
[Iter 8600][loss: 0.780][loss_cls: 0.051][loss_box: 0.212][loss_ctr: 0.516]
[Iter 8700][loss: 0.913][loss_cls: 0.061][loss_box: 0.243][loss_ctr: 0.610]
[Iter 8800][loss: 0.955][loss_cls: 0.057][loss_box: 0.252][loss_ctr: 0.646]
[Iter 8900][loss: 0.817][loss_cls: 0.057][loss_box: 0.216][loss_ctr: 0.544]
```



VI. NMS

NMS function 的定義如下。

i. 先把左上和右下的座標點抓出來(x1,y1,x2,y2)，並把 scores 由大排到小(排序後索引為 index)。
ii. 算出 box 的面積為(x2-x1)*(y2-y1)。
iii. 用一個 while 判斷 index 是否為空的。
iv. 把分數最大的賦予給 idx，並用一個 list 把它存起來。把最高分數的排除掉，如果 index 空了就 break 掉。
v. 把 x1[index], y1[index], x2[index], y2[index]分別存為 xx1,yy1,xx2,yy2，為不是分數最大的索引。
vi. 利用 torch.max 讓 xx1,yy1,去跟 x1[idx],y1[idx]比大小。利用 torch.min 讓 xx2,yy2 去跟 x2[idx]和 y2[idx]比大小。
vii. 計算交集的部分 inter

viii. 利用 torch.index_select 算出剩餘框對應的面積 rem_area，再把 rem_area-inter 計算剩餘框非交集的部分。最後再與 area[idx]相加得到所有剩餘框與當前框的聯合面積

ix. 計算 iou=inter/聯集，再把 iou 小於臨界值的去除掉。

程式碼如下:

```
x1,y1,x2,y2=boxes[:,0],boxes[:,1],boxes[:,2],boxes[:,3]
_,  index  =  torch.sort(scores,  descending=True)
keep=[]
area=(x2-x1)*(y2-y1)
while  len(index)>0:
    idx=index[0]
    keep.append(idx)
    index=index[1:]
    if  len(index)==0:
        break
    xx1,xx2,yy1,yy2=x1[index],x2[index],y1[index],y2[index]
    xx1,xx2,yy1,yy2=torch.max(xx1,x1[idx]),torch.max(xx2,x2[idx]),torch.max(yy1,y1[idx]),torch.max(yy2,y2[idx])
    w=torch.clamp(xx2-xx1,min=0.0)
    h=torch.clamp(yy2-yy1,min=0.0)
    inter=w*h
    rem_area=torch.index_select(area,  dim=0,index=index)
    union=(rem_area-inter)+area[idx]
    iou=inter/union
    index=index[iou<iou_threshold]
keep=torch.tensor(keep)
```

下圖為測試我的 nms 和 torchvision 的 nms 比較。結果表明，torchvision 的 nms 比我的 nms 在 CPU 上快了 12 倍，在 CUDA 上快了 66 倍。

```
Testing NMS:
Your         CPU  implementation: 0.948993s
torchvision CPU  implementation: 0.075314s
torchvision CUDA implementation: 0.014333s
Speedup CPU : 12.600438x
Speedup CUDA: 66.210307x
Difference CPU :  0.0018764619378015546
Difference CUDA:  0.0037739951446327247
```

VII. Inference

以下為 FCOS 的 inference 的需要實作的地方:

1. 使用 torch.max 在 level_pred_scores 中沿著第一維度（類別維度）找到最大值和對應的索引，分別獲得 level_pred_scores 和 level_pred_classes。
2. 定義一個閥值 idx，判斷 level_pred_scores 是否>=test_score_thresh，並把 level_pred_classes、level_pred_scores、level_pred_delta、level_pred_location 符合閥值定義的保留。
3. 帶入 level_deltas 和 level locations 和每一個特徵層的 stride，使用了 fcos_apply_deltas_to _locations 獲得預測框的最終座標。

4. 根據輸入圖像的高度和寬度，對預測框進行裁剪。將預測框的座標限制在圖像範圍內，使其不超出圖像的邊界。

程式碼如下：

```
# Step 1:
# Replace "pass" statement with your code
# Get the most confident class and its score for each box
level_pred_scores,level_pred_classes=torch.max(level_pred_scores,dim=1)
# Step 2:
# Replace "pass" statement with your code
idx=level_pred_scores>=test_score_thresh
level_pred_classes=level_pred_classes[idx]
level_pred_scores=level_pred_scores[idx]
# Step 3:
# Replace "pass" statement with your code
level_deltas = level_deltas[idx]
level_locations = level_locations[idx]
level_pred_boxes=fcos_apply_deltas_to_locations(level_deltas, level_locations,self.backbone.fpn_strides[level_name])
# Step 4: Use `images` to get (height, width) for clipping.
# Replace "pass" statement with your code
level_pred_boxes[:, 0] = torch.clamp(level_pred_boxes[:, 0],min=0,max=images.shape[2])
level_pred_boxes[:, 1] = torch.clamp(level_pred_boxes[:, 1],min=0,max=images.shape[3])
level_pred_boxes[:, 2] = torch.clamp(level_pred_boxes[:, 2],min=0,max=images.shape[2])
level_pred_boxes[:, 3] = torch.clamp(level_pred_boxes[:, 3],min=0,max=images.shape[3])
```

輸出結果如下：

VIII. Evaluation

這裡主要是要計算 mAP (mean Average Precision)。

Total inference time 如下:

```
import locale
locale.getpreferredencoding = lambda: 'UTF-8'
inference_with_detector(
        detector,
        val_loader,
        val_dataset.idx_to_class,
        score_thresh=0.4,
        nms_thresh=0.6,
        device=DEVICE,
        dtype=torch.float32,
        output_dir="mAP/input",
)
```

```
Total inference time: 151.6s
```

每個類別的 Average Precision 如下:

```
/content/mAP
49.37% = aeroplane AP
19.06% = bicycle AP
33.40% = bird AP
4.92% = boat AP
2.86% = bottle AP
40.44% = bus AP
40.65% = car AP
49.42% = cat AP
0.94% = chair AP
27.88% = cow AP
22.80% = diningtable AP
34.30% = dog AP
36.08% = horse AP
36.23% = motorbike AP
18.89% = person AP
3.27% = pottedplant AP
13.62% = sheep AP
21.63% = sofa AP
56.74% = train AP
5.00% = tvmonitor AP
mAP = 25.87%
Figure(640x480)
```

mean Average Precision 如下: