YOLO Object Detection on PASCAL VOC

B092040016 陳昱逢

Assignment 5

1 實作細節

本次作業主要實作 You Only Look Once (YOLO) 的 loss function，Figure 1 呈現 YOLO 的損失函數，主要分三個部分：（1）regression loss （2）confidence loss (3) classification loss

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \qquad (3)
\end{aligned}
$$

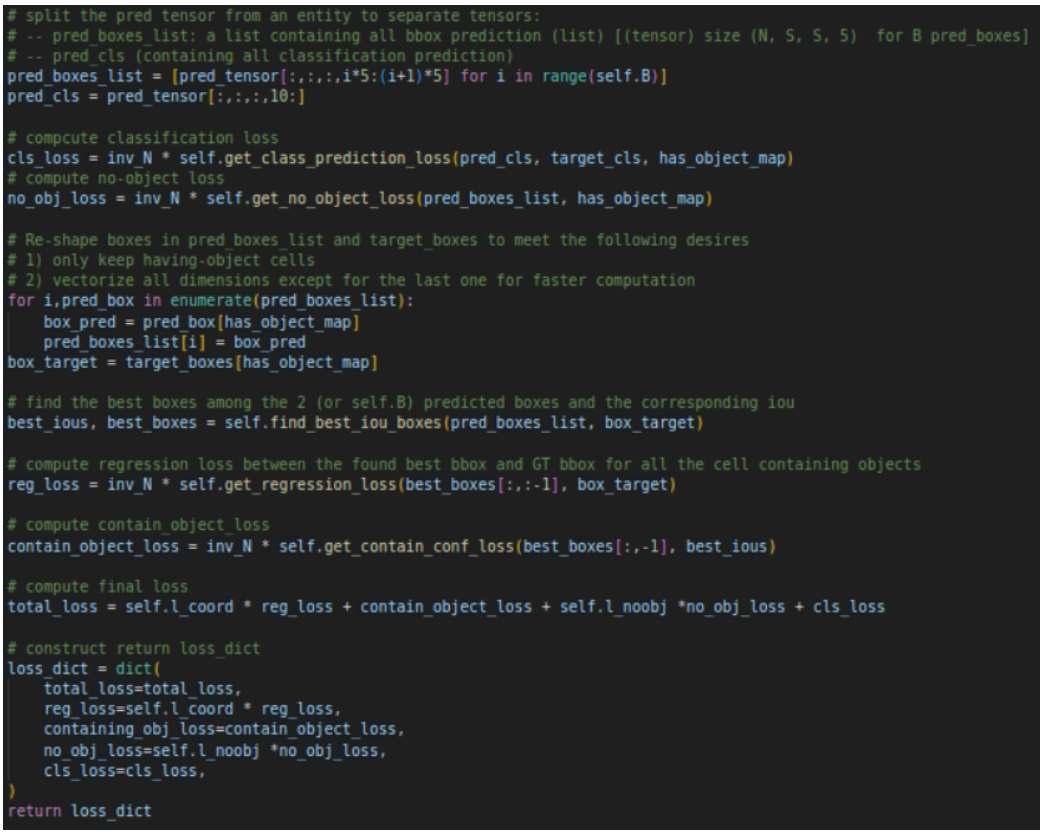
Figure 1: YOLO loss function

在了解到 YOLO loss function 是如何運作之後，我從 YOLO loss class 的 forward function 開始實作，再依序完成呼叫到的 function，分類錯誤以及沒有物體的信心度分數都只需簡單的計算 mse loss，而 regression loss 以及有物體的信心分數 loss 都需經過找到每個含物體的 grid 裡面 iou 最大的 box 再分別去算對應的 loss 。Figure 2 呈現我的主要 YOLO loss forward funtion。

2 準確度報告以及嘗試一些實驗

Table 1: 實驗測試

Model	mAP
resnet50	**0.4691**
resnet50 + 水平翻轉 / 隨機擦除	0.2156



```python
# split the pred tensor from an entity to separate tensors:
# -- pred_boxes_list: a list containing all bbox prediction (list) [(tensor) size (N, S, S, 5)  for B pred_boxes]
# -- pred_cls (containing all classification prediction)
pred_boxes_list = [pred_tensor[:,:,:,i*5:(i+1)*5] for i in range(self.B)]
pred_cls = pred_tensor[:,:,:,10:]

# compcute classification loss
cls_loss = inv_N * self.get_class_prediction_loss(pred_cls, target_cls, has_object_map)
# compute no-object loss
no_obj_loss = inv_N * self.get_no_object_loss(pred_boxes_list, has_object_map)

# Re-shape boxes in pred_boxes_list and target_boxes to meet the following desires
# 1) only keep having-object cells
# 2) vectorize all dimensions except for the last one for faster computation
for i,pred_box in enumerate(pred_boxes_list):
    box_pred = pred_box[has_object_map]
    pred_boxes_list[i] = box_pred
box_target = target_boxes[has_object_map]

# find the best boxes among the 2 (or self.B) predicted boxes and the corresponding iou
best_ious, best_boxes = self.find_best_iou_boxes(pred_boxes_list, box_target)

# compute regression loss between the found best bbox and GT bbox for all the cell containing objects
reg_loss = inv_N * self.get_regression_loss(best_boxes[:,:-1], box_target)

# compute contain_object_loss
contain_object_loss = inv_N * self.get_contain_conf_loss(best_boxes[:,-1], best_ious)

# compute final loss
total_loss = self.l_coord * reg_loss + contain_object_loss + self.l_noobj *no_obj_loss + cls_loss

# construct return loss_dict
loss_dict = dict(
    total_loss=total_loss,
    reg_loss=self.l_coord * reg_loss,
    containing_obj_loss=contain_object_loss,
    no_obj_loss=self.l_noobj *no_obj_loss,
    cls_loss=cls_loss,
)
return loss_dict
```

Figure 2: YOLO loss forward function

我在 epoch 45 的時候，validation set 上的測試結果達到最高的 mAP 是 0.4691。經過觀察，我發現大物體的 AP 較高，而小物體的較低，因此我嘗試了結合 focal loss，但效果沒有比較好，也嘗試了拿到訓練集我對影像做了 RandomHorizontalFlip 以及 RandomErasing，想透過遮住一些物體來讓模型能夠更聚焦在其他物體或是其他輪廓上，但結果也沒有比較好。Table 1 呈現了我此次實驗的結果。

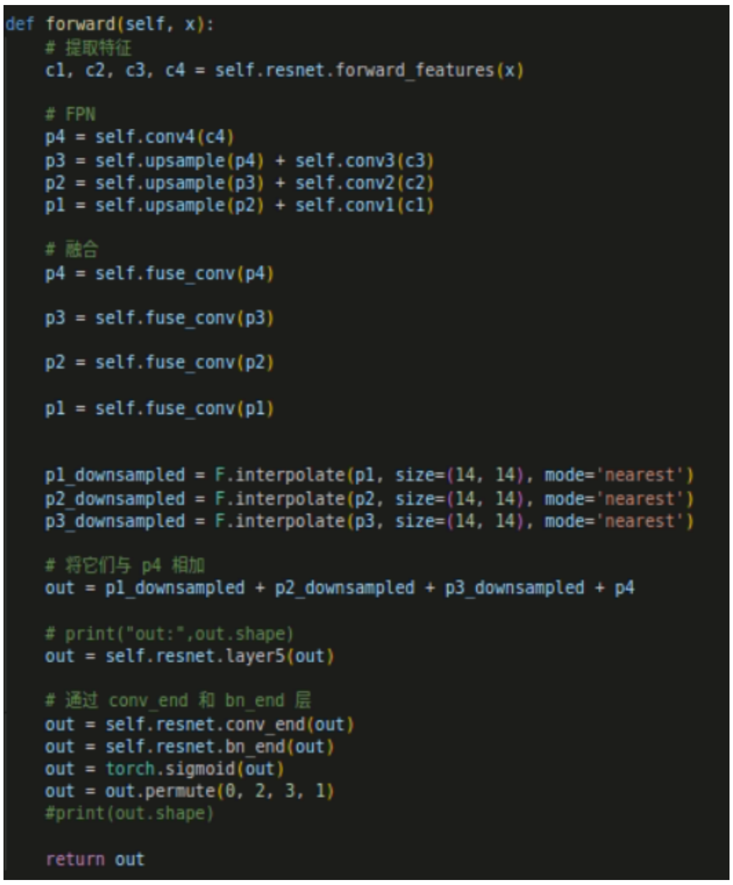
3 Extra credit

3.1 Video

這部分我主要實作方式是將影片的每一幀當做一個圖片來看待，每一幀送進我的模型偵測完後，經由 cv2 的套件將偵測結果畫上去，跟原來的幀做結合，在將每一幀輸出成影片，我的輸出影片連結有放在我的 A5.ipynb 裡面

3.2 Train more advanced model & other tricks

由於剛剛在 section 2 有討論過說，大物體小物體的 feature scale 不同，導致模型較關注大物體，因此針對這點，我想將 feature pyramid newtwork (FPN) 融入我的 CNN 架構中，來提取不同 level 的 feature map，我的做法主要是分別對 ResNet50 裡面的四層的輸出結果提取出來，上層的 layer 的 output 經過上采樣後與下層的相加，在經過 cnn 結合起來，最後把得到的四個 feature map 經由插值 resize 到同一個大小在輸入 detnet 裡進行物件偵測，詳細程式碼在 Figure 3, 但效果一樣不佳。



```
def forward(self, x):
    # 提取特征
    c1, c2, c3, c4 = self.resnet.forward_features(x)

    # FPN
    p4 = self.conv4(c4)
    p3 = self.upsample(p4) + self.conv3(c3)
    p2 = self.upsample(p3) + self.conv2(c2)
    p1 = self.upsample(p2) + self.conv1(c1)

    # 融合
    p4 = self.fuse_conv(p4)

    p3 = self.fuse_conv(p3)

    p2 = self.fuse_conv(p2)

    p1 = self.fuse_conv(p1)


    p1_downsampled = F.interpolate(p1, size=(14, 14), mode='nearest')
    p2_downsampled = F.interpolate(p2, size=(14, 14), mode='nearest')
    p3_downsampled = F.interpolate(p3, size=(14, 14), mode='nearest')

    # 将它们与 p4 相加
    out = p1_downsampled + p2_downsampled + p3_downsampled + p4

    # print("out:",out.shape)
    out = self.resnet.layer5(out)

    # 通过 conv_end 和 bn_end 层
    out = self.resnet.conv_end(out)
    out = self.resnet.bn_end(out)
    out = torch.sigmoid(out)
    out = out.permute(0, 2, 3, 1)
    #print(out.shape)

    return out
```

Figure 3: implementation of FPN