VRDL Homework 2

0716072 吳季嘉

GitHub link of my code

https://github.com/072jiajia/VRDL_HW2

Reference

https://github.com/signatrix/efficientdet (Reference Code)
https://arxiv.org/pdf/1911.09070.pdf (EfficientDet)

Speed benchmark

為了維持我的模型的準確度，我在 inference 的階段會將所有照片的短邊 resize 成 160，長邊會跟著等比例放大。所以我在附圖中附上對於 testing dataset 中，長寬比最大的照片，長寬比最小的照片以及長寬比最靠近平均值的照片的預測時間。

相較於將長邊 resize 成定值，並將短邊剩餘的地方補 0 的方法，resize 短邊的方法可以在長寬比很大或很小的時候維持 performance。

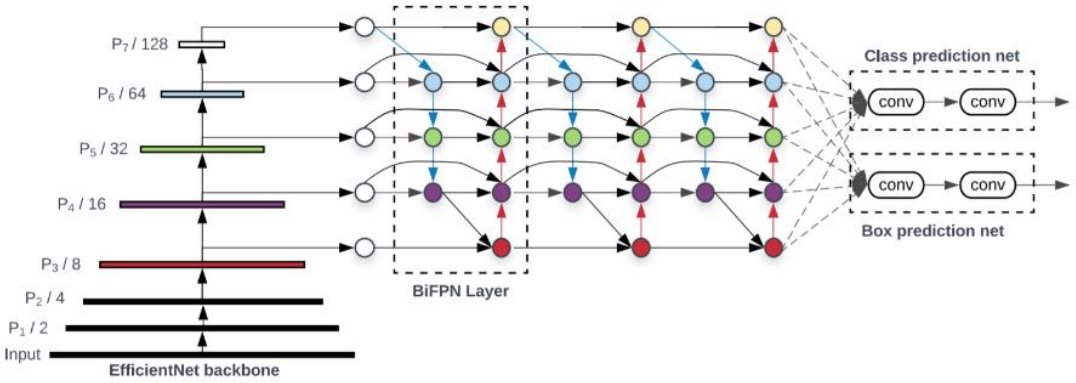
```
def detect(model, image, scale):
        image = image.to(device).float()
        nms_scores, nms_class, nms_anchors = model([image])
        return nms_scores, nms_class, nms_anchors / scale

print('mean H/W 10999.png (resized image size =', image_mean_ratio.shape, ')')
%timeit detection = detect(model, image_mean_ratio, scale_mean_ratio)
print()

print('max H/W 12115.png (resized image size =', image_max_ratio.shape, ')')
%timeit detection = detect(model, image_max_ratio, scale_max_ratio)
print()

print('min H/W 2749.png (resized image size =', image_min_ratio.shape, ')')
%timeit detection = detect(model, image_min_ratio, scale_min_ratio)
```

```
mean H/W 10999.png (resized image size = torch.Size([1, 3, 160, 384]) )
10 loops, best of 3: 32.7 ms per loop

max H/W 12115.png (resized image size = torch.Size([1, 3, 160, 1040]) )
10 loops, best of 3: 89.2 ms per loop

min H/W 2749.png (resized image size = torch.Size([1, 3, 160, 176]) )
10 loops, best of 3: 23.5 ms per loop
```

Brief Introduction

在這份作業中，我的模型參考了 EfficientDet。考慮到偵測的物件是數字，所以我對 Anchor 的長寬比做了更改，又因為 Dataset 中都是比較小型的圖片，我將論文中的模型的層數以及 Anchor 的大小做了調整。

Related Works



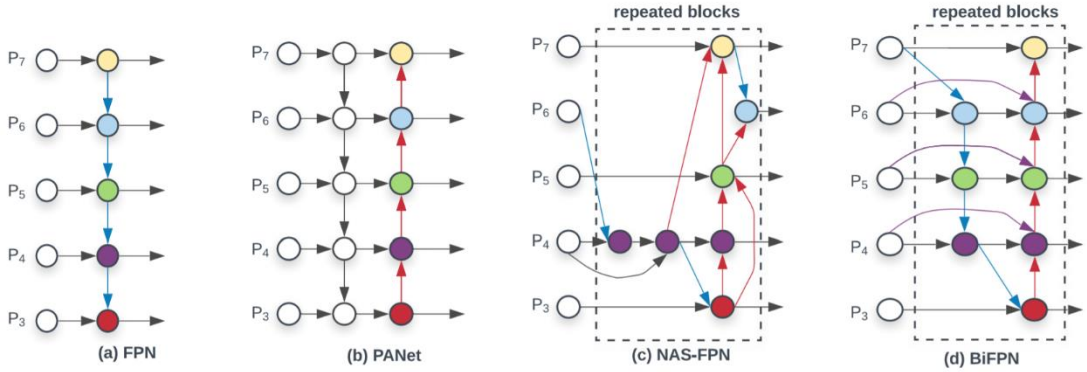
EfficientDet

這是 2020 年的一篇論文提出的物件偵測模型。它透過一張圖片產生不同大小的 feature map，並對它們做不同大小的物件偵測。

Backbone

使用 EfficientNet 作為 backbone 來產生 feature pyramid。EfficientNet 是 2019 年的一篇論文中提到的較高效的網路架構。它在相同大小的模型中有較快的預測時間以及較高的準確率。

Feature Pyramid Network



論文中提出 Bi-direction FPN 讓不同大小的 feature map 可以做更 high-level 的特徵提取。相較過去的 FPN, PANet, NAS-FPN 有更好的結果。

Anchor Box

在 Feature Pyramid 中，每個 pixel 都會產生 3 種大小和 3 種長寬比的共 9 個 anchor box，每個 anchor box 會去計算各自包含的物件類別以及它的 bounding box。Bounding box 的算法為計算 4 個值，dx, dy, dh, dw。Bounding box 的 x, y, h, w 是由 anchor 的 anchor_x, anchor_y, anchor_h, anchor_w 和 dx, dy, dh, dw 產生，其算法為

x = anchor_x + dx * anchor_w
y = anchor_y + dy * anchor_h
h = exp(dh) * anchor_h
w = exp(dw) * anchor_w

Loss Function

對於所有 anchor box 計算與它最靠近的 annotation 的 IoU，如果 IoU 大於 0.5 則為正樣本(數字)，小於 0.4 則為負樣本(背景)，0.4 和 0.5 之間則忽略。

LossFunction 分成 2 個部分，classification loss 和 regression loss

Classification Loss

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

使用 Focal Loss, 對於所有正樣本和負樣本，計算它的 Binary Cross Entropy Loss 並乘上 Focal Weight，其中平衡係數α_t用來解決物件和背景資料量不對稱的問題，並使用γ來調整簡單測資和困難測資對梯度的影響。

因為一個 batch 的 Loss 是由各個物件和背景的 Loss 加總而產生，又因為背景和數字出現次數的不對稱，所以當將數字判斷為背景時將 loss 乘上較大的α，而當將背景判定為數字時將 loss 乘上較小的α，以此避免 model 為了降低 loss 而寧可將全部都判斷為背景。

此外，對於 predict 出來較差的物件，我們希望它們可以佔在 Loss 中比較大的比例，像是有 100 個很好分的背景和 1 個很難分的數字，我們希望可以著重學習到分辨 1 是物件而不是分辨出那裡是背景，所以乘上一個$(1-p_t)^\gamma$來讓難分辨的物件能在 loss 中佔有更大的比例。

Regression Loss

對於所有正樣本，計算該 anchor 要轉換成 ground truth 所需要的 GT_dx, GT_dy, GT_dh, GT_dw，用 Smooth L1 來計算 regression loss。

Methodology

Data pre-process

在 training 階段，我將所有的照片做等比例的縮放，將短邊縮放成 144~176 之間的一個隨機值，再對整張照片做 128 * 128 的 random crop 並調整標記的位置與大小。考慮到 Loss Function 的計算方法，即便物件被 crop 到一半也不會對訓練造成太大影響。最後再以 ImageNet 的 mean 和 std 去 normalize input image。

Model

我的模型參考了以上的論文，但我將 EfficientNet 改為 ResNet。在 input image 是 n * m 大小時，使用 ResNet 所產生的(n/2)*(m/2), (n/4)*(m/4), (n/8)*(m/8), (n/16)*(m/16)這四個大小的 feature map 來組成 feature pyramid，並將 BiFPN 的結構中的第 3 層刪掉。在 anchor 的部分，因為這個 dataset 是要偵測數字，所以我只將 anchor 設定 2 種 height / width，分別為 2.5 和 1.25

Validation and Testing

將所有的照片的等比例縮放，短邊為 160，考慮到 BiFPN 的會做 down sampling 和 up sampling，將長邊 padding 為 16 的倍數以符合模型的架構。將縮放過的整張圖片放入模型進行預測。

Hyperparameters

這個模型共訓練的 200 個 epochs，使用 AdamW 作為 optimizer，learning rate 定為 0.0001，為了避免梯度過大將梯度限制在 -0.1 到 0.1 之間。使用 ReduceLROnPlatea 作為 schedular，當 3 個 epochs 內 validation loss 沒有下降就調降 learning rate 為 0.1 倍。

Focal Loss 的部分我將α_t設為 0.25，γ設為 5。

Experiment & Result

為了加速每張圖片的預測速度，我嘗試了 EfficientNet-b0 到 b2 以及速度和它們差不多的 ResNet-34，結果使用 EfficientNet 的 model 都很快的 overfitting 且無法通過 baseline，所以我最後決定採用 ResNet-34 作為這次作業的 backbone。

Summary

在這份作業，因為圖片的大小很小，所以我嘗試修改現有的模型縮小並加快他的速度。結果也顯示，我的 inference 速度是有比 baseline 快的。只可惜我的 mAP 並不高。或許是因為我的網路深度不夠以及我的 Feature Pyramid 只有 4 層。