109 學年度第 2 學期

機器學習

Final_Project

繁體中文場景文字辨識競賽

初階：場景文字檢測

隊伍：麻瓜磨成菸草灰
系所：電機所
教師：江振國老師
日期：2021.06.20

一、 環境

請說明使用的作業系統、語言、套件(函式庫)、預訓練模型、額外資料集等，如使用預訓練模型及額外資料集請列出來源。

作業系統	Colab
程式語言	Python
套件	Pytorch
套件	Pyyaml
套件	tqdm
套件	tensorboardX
套件	opencv-python 4.1.2.30
套件	anyconfig
套件	munch
套件	scipy
套件	sortedcontainers
套件	shapely
套件	pyclipper
套件	gevent
套件	gevent-websocket
套件	flask
套件	editdistance
套件	scikit-image
套件	imgaug 0.2.8
預訓練模型	Resnet-50
訓練集	AICUP 所提供之 TrainDataset
測試集	AICUP 所提供之 PublicTestDataset
測試集	AICUP 所提供之 PrivateTestDataset

表 1 環境配置

二、 演算方法與模型架構

說明演算法設計、模型架構與模型參數，包括可能使用的特殊處理方式。

模型架構：本次競賽我們所使用的模型架構是以 Resnet-50 的預訓練模型進行訓練，因此以 RestNet-50 為主體，其架構圖如下圖 1。

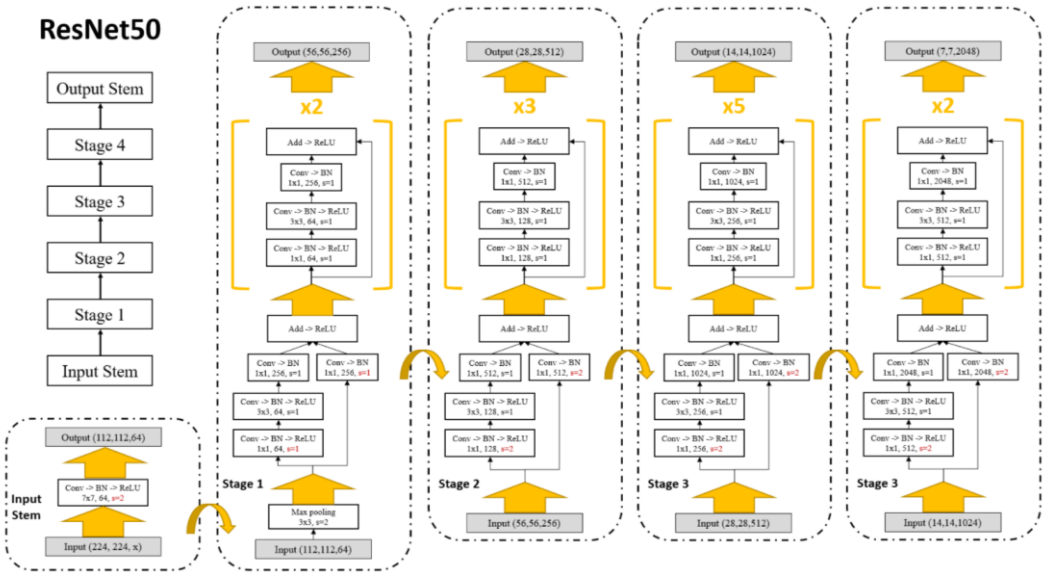

圖 1 ResNet-50 架構圖

ResNet 的網路設計其實很簡單，沒有甚麼過多複雜的程序，就是單純、簡單的增加一條路線做加法，而這樣所組合的卷積層在論文中稱為一個 block，如此簡單的方法可以極度的簡化深層網路訓練，使其變得容易許多，而在要做更加深層的網路架構時，ResNet 設計了 bottleneck block，降低了 3x3 卷積層的寬度，一旦減少了寬度即可大幅減少其計算量，網路圖如下圖 2。

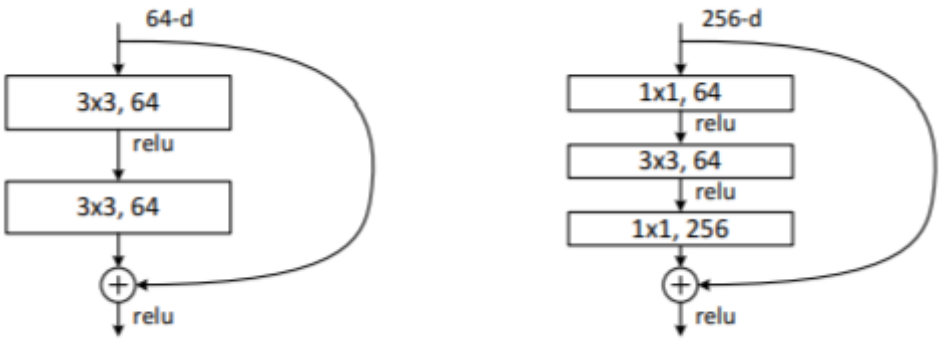

圖 2 ResNet 網路圖

ResNet 根據其網路深度又可以分成 ResNet-18、ResNet-34、ResNet-50、ResNet-101、ResNet-152，其差別主要可以分為兩大項。第一項是在我們上述所討論的 block，ResNet-18 與 ResNet-34 均使用一般的 residual block，而 ResNet-50、ResNet-101、ResNet-152 則是使用了 expansion 為 4 的 bottleneck block，如表 2。第二項差異則是其網路所堆疊的 building block 層數不同，ResNet 的網路層數與其名相同，舉例來說如果是 ResNet-18 則 building block 就是 18 層，本次競賽我們所使用的是 ResNet-50，也就是 50 層的 building block，其詳細資料如圖 3 所示。

名稱	Block
ResNet-18	residual block
ResNet-34	residual block
ResNet-50	bottleneck block
ResNet-101	bottleneck block
ResNet-152	bottleneck block

表 2 各 ResNet 的 block

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

圖 3 各 ResNet 的比較表

模型參數：

參數名稱	參數 Value
Training Epoch	80
Num_workers	4
Batch_size	12
Training Data	4000
Validation Data	0
Learning Rate	0.001
Optimizer	SGD
Save_interval	9000
Weight_decay	0.0001

表 3 模型參數配置

三、 資料處理

資料處理的部分可以切割成前處理與後處理，在訓練前所做的資料處理稱為前處理，而在訓練後的我們在此稱為後處理。而前面兩項為前處理，後面的兩項則是後處理。

首先我們先對 Training Data 做前處理，由 AICUP 所提供的 Label 檔案是.json 檔，因此我們需先將其轉換成.txt，才可以交給模型進行訓練。

其次是環境的配置，由於我們所使用 pytorch 版本是高於 1.3 的版本，因此先將範例程式的 deform_pool_cuda.cpp 與 deform_conv_cuda.cpp 兩個檔案進行修改，檔案內的 AT_CHECK 全部替換成 TORCH_CHECK，才能夠執行。

在模型訓練完畢後，我們需要去 Demo 我們的測試資料集，因此我們可以寫一個 for in range 的迴圈去加快、簡化我們的 Demo 時間。

最後是後處理的部分，我們需彙整 Demo 所產生出來的.txt 檔，因此在這邊我們寫了一個 Python 程式方便我們進行處理，最後即可將我們所有的 Demo 數據合在一起變成一個.csv。

讀取每一張照片的 json 檔中 shapes 資訊，並只提取 label、points 及 group_id 的資料，並將其儲存為 txt 檔，以利後續訓練模型時使用。

label: 為該標註框的文字內容，因初階賽只有文字定位，故文字內容為 null。

points: 為一個二維 list，每個 item 為一個(x,y)座標。由左至右分別為，以標註框左上方為起點，以順時針方向走訪到標註框左下方的點座標。

group_id: 為該標註框所屬的類別編號。

四、 訓練方式

本次競賽我們主要撰寫的部分是.yaml 檔，其內的參數是控制我們 ResNet-50 的 Value，我們使用的環境是 Colab，因此我們使用 Google 所提供的顯示卡 Tesla T4 進行訓練，其規格如圖 4。

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 465.27       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   38C    P8     9W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

圖 4 顯示卡資訊

不過我們在訓練模型時，遇到了一個相當大的問題，就是 Google 的 Colab 所提供顯示卡只能夠使用 12 小時，但是我們光是要 Training 一個 Epoch 就要半個小時，因此我們若是想要 Training 80 個 Epoch，就要 30min * 80 = 2400 min = 40hr，因此我們後來發現可以藉由儲存模型並切換 Google 的帳戶重新連線顯示卡，並且讀取我們上次所訓練到一定 Epoch 的模型繼續訓練，因此我們只需要 3、4 個 Google 帳戶就可以不間段的使用免費的顯示卡進行訓練，使用的指令如下圖 5 所示：

```
!CUDA_VISIBLE_DEVICES=0 python train.py experiments/seg_detector/AICUP_resnet50_deform_thre.yaml \
--num_gpus 1 --epochs 60 --num_workers 8 --batch_size 4 \
--resume outputs/workspace/DB/SegDetectorModel-seg_detector/deformable_resnet50/L1BalanceCELoss/model/final_60 --start_epoch 5
```

圖 5 讀取模型繼續訓練的指令

指令介紹：

- resume：讀取先前的模型繼續訓練

-start_epoch：從哪個 Epoch 開始進行訓練。

五、 分析與結論

AI_CUP 的最終成績與排名：

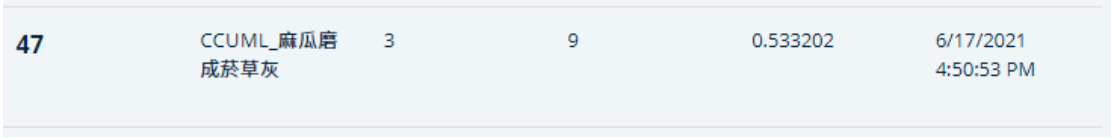

圖 6 排名圖

我們最後送出的成績是以 80 個 Epoch 進行訓練的，最後得到的 Training INFO 如下圖所示：

```
[INFO] [2021-06-16 22:48:46,726] step:   1800, epoch:  80, loss: 2.087735, lr: 0.006579
[INFO] [2021-06-16 22:48:46,736] bce_loss: 0.274690
[INFO] [2021-06-16 22:48:46,737] thresh_loss: 0.216716
[INFO] [2021-06-16 22:48:46,738] l1_loss: 0.049757
```

圖 7 Training INFO

我們最後是以這組 Model 進行測試，並且將其結果進行輸出，而前面的也有測試過其他的 Epoch 去進行成績的評估，如下圖所示。我們所使用的是以 5 個 Epoch 進行測試的模型，其實我們滿意外的，因為框出來的結果其實都還不錯但是分數卻滿低的，而且 Training 80 個 Epoch 跟 5 個集時也沒有高特別多，原本以為會達到 60 以上。

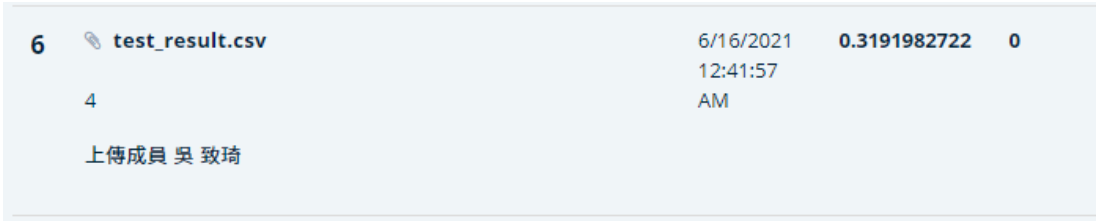

圖 8 五個 Epoch 的成績

後來我才發現，原來是優化器的設定關係，我們的優化器所儲存的的模型不不是儲存 Loss 最低，也就是效果最好的模型，而是儲存我們所訓練出來的最後的模型，只可惜當我發現這件事情的時候，我們已經沒有時間可以重新訓練了，而在看 Training 的.log 檔時有發現其實我們有訓練出很好的模型，如下圖，他的 Loss 只有 0.89，因此我認為只要我們使用這個模型下去做 demo 想必測試出來的分數一定會高於 55 分。

```
[INFO] [2021-06-16 13:16:41,365] step:  16200, epoch:  59, loss: 0.896122, lr: 0.006690
[INFO] [2021-06-16 13:16:41,368] bce_loss: 0.103495
[INFO] [2021-06-16 13:16:41,369] thresh_loss: 0.083082
```

困難點是在於，不知道朝哪一個面向去更改模型才會得到最好的辨識率提升，因此只能從各個方面下去亂嘗試，過程是相當耗費時間的，但其實也有從實驗的過程當中慢慢了解到參數對模型的影響哪個大。

在實驗的過程中有做了許多的嘗試，包含加深 CNN layer 以及提升 Dimension 還有加入輸出以後的殘差值並將其加入下一層的輸入，DBnet 有極限，到了一個程度以後，就無法再更好了，只能從優化器，或其他的方面去著手調整。

圖 9 模型測試結果圖 1

圖 10 模型測試結果圖 2

圖 11 模型測試結果圖 3

圖 12 模型測試結果圖 4

圖 13 模型測試結果圖 5

六、 程式碼

1.以.yaml 檔的形式附上

2.Colab 連結：

https://colab.research.google.com/drive/1i7moWU_NinNxFnnJqTewk9sgDtTnkPhq?usp=sharing

3.相關程式碼連結：

https://drive.google.com/drive/folders/1kP0Zkbmdobk0jjYpUOy9OOwlFFqK8RvL?usp=sharing

七、 使用的外部資源與參考文獻

[1] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[2] https://colab.research.google.com/notebooks/intro.ipynb?utm_source=scs-index