Exercise 1 Report

資工四 陳昱瑋 409410118

Introduction

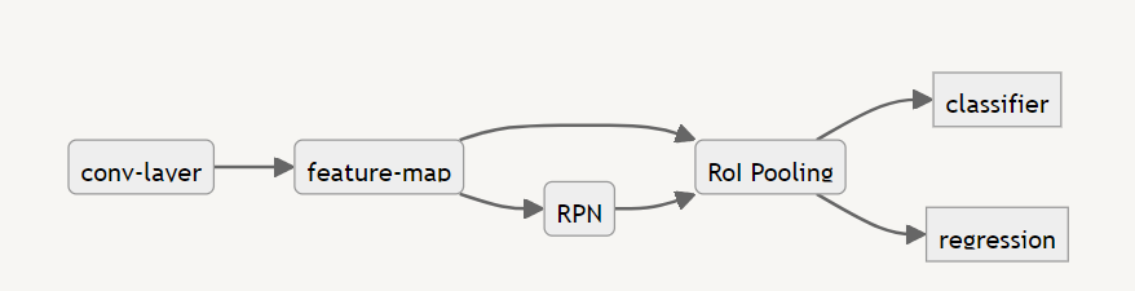
使用detectron2進行object detection的任務。detectron2為facebook所提供的framework，可以讓使用者可以模組化、客製化ML，根據需求可以進行參數、架構調整。

這次主要透過detectron2針對Cityscapes資料集(Pascal VOC format)進行model training，比較透過Backbone ResNet50與VGG16的在efficiency、accuracy、total loss 等等方面上的差別，在實驗結果之上進行分析與歸納，驗證ResNet50與VGG16之表現。

Experimental Setup

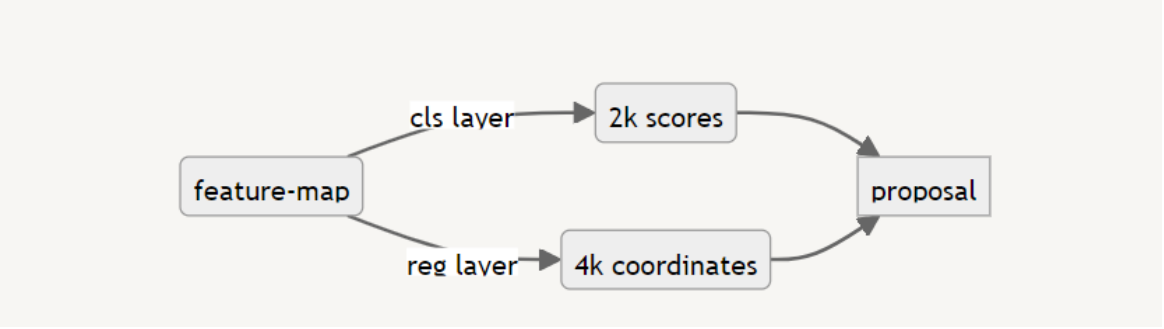
Detail of the model(Faster R-CNN)

為Fast R-CNN的變種，引入RPN(region proposal network)，不再僅限feature map之特定區域進行selective search，而是進行anchor prediction。



這裡關注於RPN與RoI Pooling進行解釋。

- **RPN**



  - 首先會對每個點產生9個(K=9)不同aspect ratio、scale的anchors，整張圖就會有width*height*K個anchors。將這些anchors輸入到RPN(region proposal network)進行anchor prediction，透過sliding window(3*3 conv filter)對每個位置的點分別計算。
  - 會分成 classification 和 regression
    - Classification:

        - 決定proposal為target object之機率
        - anchor與ground truth比較，IoU>0.7 ⇒ P=1
    - Regression:
        - 決定proposal的coordinate
        - 透過 Smooth L1 loss 找出 t_i
- 最後使用loss function

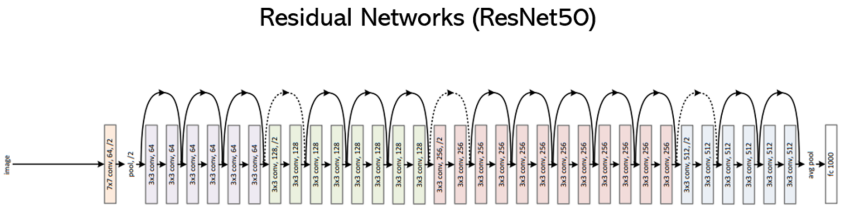
$$L(\{p_i\}\{t_i\}) = \frac{1}{N_{cls}} \sum_i \overbrace{L_{cls}(p_i, p_i^*)}^{\text{Log Loss}} + \lambda \frac{1}{N_{reg}} \sum_i p_i^* \overbrace{L_{reg}(t_i, t_i^*)}^{\text{Smooth L1 loss}}$$

- **RoI Pooling**

  把RPN輸出的proposal設為RoI的區域進行Pooling(H*W*C)，將所有不同大小的proposal以相同大小(fixed-length)輸出，進行classification與regression。

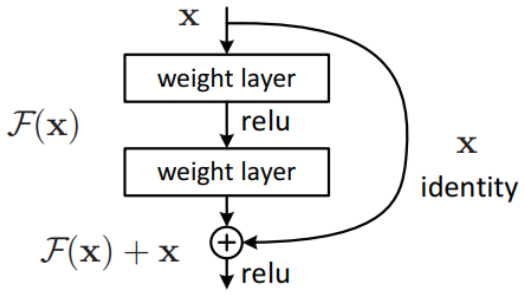
Detail of the backbone(ResNet50 and VGG16) and the differences between them

- ResNet50



ResNet50_From_Scratch_Tensorflow | This repository implements the basic building blocks of Deep Residual networks which is trained on SIGNS dataset to detect numbers on hand images (jananisbabu.github.io)

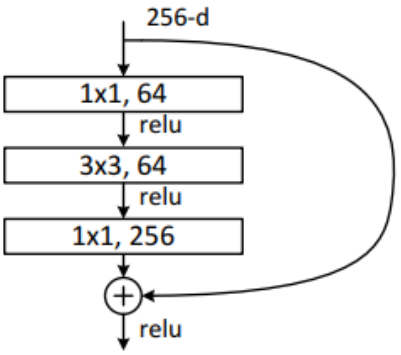
- Basic Idea



1512.03385.pdf (arxiv.org)

為了避免較深的訓練造成gradient vanishing，提出了 **Residual Mapping** 的想法。假設 input 為 x，expected output 為 $H(x)$，將 $H(x)$ 轉換成 $F(x) = H(x) - x$，改成求 $F(x) = 0$ 的誤差最小解，即$H(x) = F(x) + x$。這就導入了 **Shortest Connections** 的概念，實作上也不會增加運算成本，簡單的就可以得到 $H(x)$。

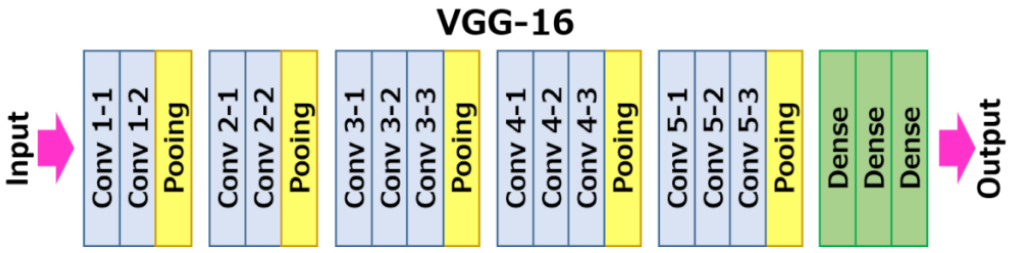
- Bottleneck



1512.03385.pdf (arxiv.org)

由於在大於50層的網路會使用到更多時間，為節省大量參數，提出Bottleneck技術。使用1*1 conv 進行降維再升維，讓 3*3 的Bottleneck不需讀那麼多的參數，又同時可以處理更高維度的input。

- VGG16



VGG-16 | CNN model for Classification and Detection - All about Machine Learning (techcraft.org)

16 代表有16個 conv layer，每個Block後面配合一個MAXPOOLING，最後接三個FC layer。

然而VGGNet有著致命的缺點

- Slow to train
- Acquire large disk space due to weights are quite large

其深度與FC layer數量，導致VGGNet需要大量的空間。但在較小的network architecture 可能會有較佳的表現 (如 SqueezeNet、GoogLeNet 等等)。

Description of the data loading process

```
from detectron2.data.datasets import register_pascal_voc

cls_names = ('truck', 'car', 'rider', 'person', 'train', 'motorcycle', 'bic
ycle', 'bus')
register_pascal_voc("my_dataset", '/path/to/Cityscapes_dataset', "trainva
l", 2007, cls_names)
register_pascal_voc("my_test", '/path/to/Cityscapes_dataset', "test", 2007,
cls_names)
```

使用 `register_pascal_voc` 裡的function `register_pascal_voc` ，進行data loading 。

Explanation of the evaluation setup

```
from detectron2.evaluation import PascalVOCDetectionEvaluator, inference_on
_dataset

evaluator = PascalVOCDetectionEvaluator("my_test")
val_loader = build_detection_test_loader(cfg, "my_test")
print(inference_on_dataset(predictor.model, val_loader, evaluator))
```

並於 pascal_voc_evaluation.py 印出每個 class 的AP

```
for cls_id, cls_name in enumerate(self._class_names):
        lines = predictions.get(cls_id, [""])

    with open(res_file_template.format(cls_name), "w") as f:
             f.write("\n".join(lines))

    for thresh in range(50, 100, 5):
            rec, prec, ap = voc_eval(
                res_file_template,
            self._anno_file_template,
              self._image_set_path,
            cls_name,
            ovthresh=thresh / 100.0,
            use_07_metric=self._is_2007,
       )
       aps[thresh].append(ap * 100)

       print(cls_name)
       res = 0
       for x, cls_x in aps.items():
               res = res + cls_x[cls_id]
       print("AP: "+str(res/50))
       print("AP50: "+str(aps[50][cls_id]))
       print("AP75: "+str(aps[75][cls_id]))
]))
```

同時使用 wandb 視覺化 training 之各個數據的變化

```
import wandb
wandb.login(relogin=True, key='...')
wandb.init(
            # set the wandb project where this run will be logged
            project="Exercise_1",
            name="backbone-name",
            sync_tensorboard=True
        )
```

Experimental Results

Accuracy in each category

- ResNet50

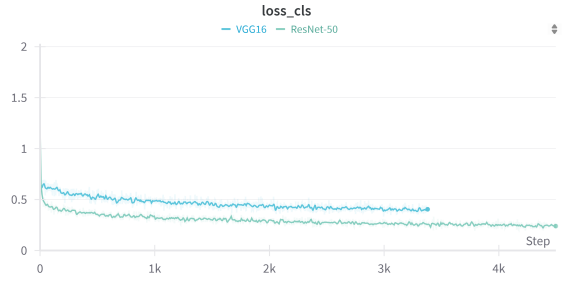
```
truck
AP: 3.7171682892271134
AP50: 31.338383838383837
AP75: 20.745920745920746
car
AP: 9.046388259463322
AP50: 62.661848007180495
AP75: 50.77474732238093
rider
AP: 6.263308047646255
AP50: 51.69606753022406
AP75: 30.086574827029832
person
AP: 5.738178152299821
AP50: 50.96133221738626
AP75: 28.273660903748304
train
AP: 2.329988965283083
AP50: 34.34343434343434
AP75: 2.02020202020202
motorcycle
AP: 4.027045120374719
AP50: 39.87173943333766
AP75: 13.203463203463203
bicycle
AP: 4.516625220783926
AP50: 41.96947257013663
AP75: 19.64384797516233
bus
AP: 6.042065819593401
AP50: 49.33982683982684
AP75: 31.677622540212635
OrderedDict({'bbox': {'AP': 26.05047992166978, 'AP50': 45.2727630974887
6, 'AP75': 24.553254942265}})
```

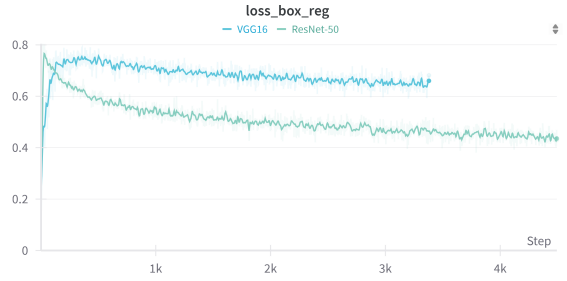
- VGG16

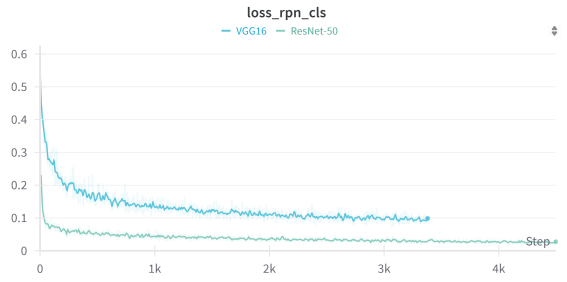
```
truck
AP: 0.09090909090909091
AP50: 4.545454545454546
AP75: 0.0
car
AP: 1.8078246161145173
```

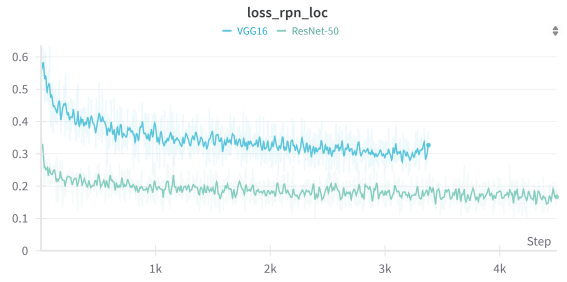
```
AP50: 25.02996680051936
AP75: 4.004103925568752
rider
AP: 1.019639333625711
AP50: 13.88888888888889
AP75: 2.4242424242424243
person
AP: 0.6655155471181206
AP50: 11.317384386681901
AP75: 1.0356731875719216
train
AP: 0.0
AP50: 0.0
AP75: 0.0
motorcycle
AP: 0.5454545454545455
AP50: 4.545454545454546
AP75: 4.545454545454546
bicycle
AP: 0.22623009179818557
AP50: 3.54913058699965
AP75: 0.6993006993006993
bus
AP: 0.6060606060606061
AP50: 9.090909090909092
AP75: 0.7575757575757575
OrderedDict({'bbox': {'AP': 3.101021144425486, 'AP50': 8.9958986056135,
'AP75': 1.6832938174642627}})
```

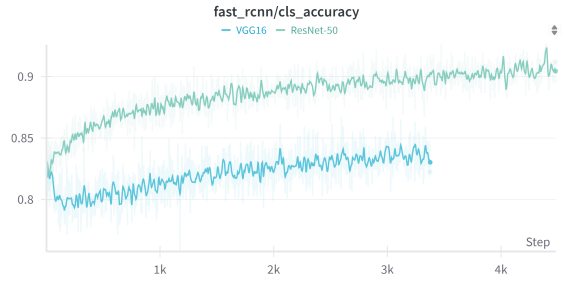
Visualization of training process

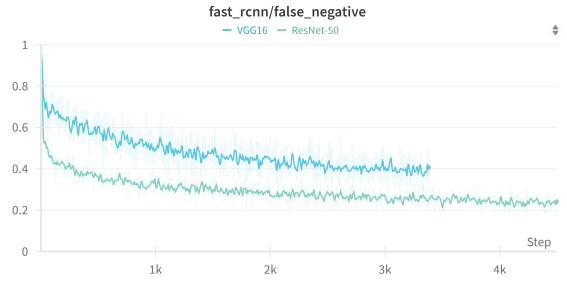












Summary

可以明顯看出VGG16的表現明顯差於ResNet50。

其中可以觀察到，VGG16在loss、accuracy、false negative的曲線幅度都又很大的擺動，可以推測其相較於ResNet50表現較為不穩定，VGG16在有大量weight的存在下，其不定性更大；而ResNet50則可以透過bottleneck的方式處理多維資料。

- loss box regression: VGG16 剛開始優於 ResNet50

  可以看到剛開始ResNet50的表現是比VGG16差的，但經過幾個epoches後ResNet50 loss box regression急遽下降，反而是VGG16 loss box regression 變得十分的高，由於ResNet50為Residual Network，故其架構可以避免gradient vanishing，因此ResNet50可以達到多層layer可以萃取更精細之feature卻不失lower layer feature；反觀VGG16，其需具大量weight，在訓練方面不只時間上很長，效果也不如ResNet50。

- AP: train、bus、truck 表現不佳

  每個category的AP表現，train的表現非常不好，推測原因為整個資料集火車的個數並沒有太多；反觀car的AP值十分的高，也可以觀察到資料集中車輛出現的次數十分頻繁。綜觀整體AP也可以發現ResNet50比VGG16好上非常多。

category	#instances	category	#instances	category	#instances
truck	489	car	27155	rider	1807
person	17994	train	171	motorcycle	739
bicycle	3729	bus	385		
total	52469				

- Diving into AP comparison

  由於ResNet50使用的是base為FPN架構，在小物件、具複雜形狀的物件上會有較好的表現，在下表的欄位 `ResNet50 / VGG16` 中可以看到bicycle, motorcycle上都有突出的準確度。

  同時也可以發現並不是instance數多就代表其準確度就會相對高，如truck(instances: 489)於ResNet50是31.3383，但train(instances: 171)於ResNet50卻有34.3434。整體看下來instance與AP關係可能為正相關，但必非正比。

  - AP50

category	instances	VGG16	ResNet50	ResNet50/VGG16
truck	489	4.5454	31.3383	6.8945
car	27155	25.0299	62.6618	2.5034
rider	1807	13.8888	51.6960	3.7221
person	17994	11.3173	50.9613	4.5029
train	171	0.0	34.3434	NaN
motorcycle	739	4.5454	39.8717	8.7718
bicycle	3729	3.5491	41.9694	11.8253
bus	385	9.0909	49.3398	5.4273

Discussion

透過這次的實作，遇到了很多問題。

1. detectron2 版本支援與作業系統與CUDA版本之支援問題

   由於本身是使用windows作業系統，為了解決這些版本問題使用wsl與conda來解決，在wsl 上使用 CUDA 也閱讀了很多文獻參考才成功裝成 detectron2。

2. 硬體資源不足

   由於default batch size: 16，這導致load進的資料量幾乎會占滿GPU memory(我的GPU只有4G)，導致每個epoch執行都非常的緩慢，須等很長的時間才會把數據寫到wandb。為解決畫圖效率與時間問題，我將batch size下降到了2，並提升iterations量，讓整體訓練量不變的情況下達到更快地寫出速度，也就是降低bandwith來換取更及時的數據回饋並保護硬體設備。

> Reference:
>
> CUDA on WSL (nvidia.com)
>
> CUDA Toolkit 12.4 Downloads | NVIDIA Developer

> facebookresearch/detectron2: Detectron2 is a platform for object detection, segmentation and other visual recognition tasks. (github.com)
>
> Getting Started with Detectron2 — detectron2 0.6 documentation
>
> ResNet50网络结构图及结构详解 - 知乎 (zhihu.com)
>
> 直觀理解ResNet —簡介、 觀念及實作(Python Keras) | by Chi Ming Lee | Medium
>
> [1512.03385] Deep Residual Learning for Image Recognition (arxiv.org)
>
> VGG-16 | CNN model for Classification and Detection - All about Machine Learning (techcraft.org)
>
> Add VGG backbones by ashnair1 · Pull Request #1584 · facebookresearch/detectron2 (github.com)