Video Streaming and tracking HW1 - Classification report

310581044 陳柏勳

1. How to reproduce your result(Including package, environment, reproduced method)

1. 使用 package 包含

Train.py

```
import pandas as pd
import numpy as np
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils import data
from torchvision.transforms import transforms
from torch.utils.data import DataLoader
from torch.utils.data import Dataset
import imageio
import torch.optim as optim
from tqdm import tqdm

import matplotlib.pyplot as plt
import matplotlib
from model import VGG11
matplotlib.style.use('ggplot')
```

model.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

test.py

```
import torch
import cv2
import glob as glob
import torchvision.transforms as transforms
import numpy as np
from model import VGG11
import pandas as pd
```

和 report 提供不一樣的是 train.py 多了 torchvision 只用來對圖片轉成 tensor 和做 normalize，numpy 用來轉成數列以便輸出，imageio 用來在 calss vggloader 中的 get_item function 讀圖片 imageio.imread，tqdm 用來把訓練時進度條顯示出來，from model import VGG11 則是我自己寫的 vgg 架構，用以把 vgg model import 進來，和 train.py 分開是方便對 vgg 架構做各種實驗，剩下的都是上述有講過的子 library，為了方便在把他 import 進來而已。而 model.py 只用了 torch 和其子 library。

Test.py 則是多了 cv2 和 glob，cv2 用 cv2.imread 來讀取圖片路徑用來的到 inference 的圖片，glob 則是用來操作文件，把每個 test 資料夾中的圖片都讀取，glob 我從 pip list 或 conda list 都找不到相關 module 版本，查詢後發現這是 python 自帶的，所以下面就沒附上版本。

2.environment

下面附上各 package 的 version 先從 report 中提到的 library，再按照上部分講解的順序描述：

Train.py

Python: 3.8.13
Pytorch: 1.12.1
Pandas: 1.4.4
Matplotlib: 3.5.2
Torchvision:0.13.1
Numpy: 1.23.1
Imageio:2.19.3
Tqdm:4.64.1

test.py

cv2:4.6.0

3.reproduced method

code 重新執行過程：我是在 anaconda3 jupyter notebook 下跑的，將 train.py 的 code 從頭到尾跑一次，會讀取 model.py 的模型架構並且讀取 dataset 和 csv 檔做訓練，爾後會先得到 model weight 和下面第四點 train datset , validation dataset acc loss 相關圖表，接著再把 test.py 執行，此 code 會讀取 train.py 的 weight 及 test 的圖片然後做 inference 預測出結果並生成 test.csv。

2. Number of Model parameters

使用 total_trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)作為判斷依據

我一開始測試的網路層數是 VGG19 得到參數量是 139,611,210，經過一些實驗後決定減少 VGG convolution layer 數成類似 VGG11 的架構，修改後

的 VGG 參數量則是 **128,807,306**，詳細的修改部分下面會解釋。

3. Explain modelstructure (as detail as possible)

這次的 model 架構主要參考的是 vgg paper 中下圖上半部提供的架構

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

其中 abde 架構為較常用部分，分別是 VGG11、VGG13、 VGG16、VGG19，而第二個下半部表格提供了個架構的參數數量，也和這次任務有關可以參考故也附上，而從這張圖及網路上查詢而知，減少參數量的主要方法主要是修改卷積層相關的數值，這次會先介紹基本架構，之後討論對卷積層做的實驗，最後再解釋各種超參數調整。

1. VGG 基本架構

是由數次的 3*3 convolution layer 和 maxpool 層互相堆疊，後面再加上數次 fully connection layer，最後再有一層 softmax output 出各 label 機率。

這部分做的實驗包含第一點是否在 convolution layer 後增加 nn.BatchNorm2d()，在 vgg paper 中作者沒有說是否使用了 BatchNorm 方法在他們的中；BatchNorm 算法是為了解決在非常深的神經網絡中梯度消失的問題，而實測不管是在 VGG11 還是 VGG19 都不會有太多影響，所以我這邊沒有加入。

而第二點是在 fully connection layer 後增加 nn.Dropout(0.5)，試著比較有無增加，發現有加 Dropout 方法的 model 差距約 3%，故選擇增加，而之後我也有嘗試只加在第一層 fully connection layer 後或是第一、兩層都有加，實驗結果差距不大，故我選擇在第一、二層後都加上 Dropout，上述兩者是否使用都對參數量沒有影響。

2.revised from VGGnet19

首先介紹一般的 VGG19 架構，首先由於這次是 RGB 圖片，input channel 為 3 經過 3*3 convolution layer 成下面 output channel 數目(即 3->64 64->64)而每次 convolution 會接一個 relu function，每次 channel 數目改變會接上一個 MaxPool 層(即下方 M 代表意義)，綜上所述，convolution layer 共有 16 層。

64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', 512, 512, 512, 512, 'M'

後面會再接上 3 層的 fully connection layer，第一層由於這次圖片是 224*224 經過前面的 convolution layer 變成 input feature 512*7*7 而 output channel 為 4096，由於是 fully connection 所以後面的 feature channel 數量不變，這邊 3 層加上前面的 16 層就是 VGG19 的基本架構。

接下來就是實驗如何在保持正確率的情況下能確實減少參數，我在網上主要找到的方法都是要修改 convolution layer 尺寸，像是用多個小尺寸 convolution kernel 替代大 convolution kernel，用 Pointwise Convolution, Depthwise Separable Conv 等等，所以基本上把焦點放在 16 層 convolution layer 身上即可，但目前的另一大問題是我在 train VGG19 的正確率大該都在 58%,59%徘徊，如果直接套用了把每個 3*3 convolution kernel 替換成小型 1*1kernel 參數量減少同時正確率可能會跌破 55%，所以不能粗暴的就直接縮小，我在第三部分開頭附上架構圖中的 C 其實就是對 VGG16 其中三層較大的 convolution layer 換成 1*1kernel，參數量減少了約 400 萬但 paper 中 error 部分其實就和 VGG13 沒差多少了。以這樣推論，如果仿造 C 的架構在多換 3 層 1*1kernel 或許正確率就會降低到接近 VGG11 了，而參數量共少 800 萬。

所以我換了另一個方向來想，如果直接砍掉較大的 convolution layer，只要最後的一層 convolution layer output channel 仍是 512 就能維持模型輸出一致而這樣也算是減少 convolution 總數，所以我就試著把都 256->256，

512->512 這種 input output 同 channel 數盡可能刪掉，之後才發現這樣其實很像 VGG11 的架構，最後呈現出是 8 層 convolution layer 和三層 fully connection layer，channel number 如下：

64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'

即使大尺寸也至少要留下一層 input ouput channel 一樣的是為了確保能學到大尺寸的 feature，而這樣算出來的參數量是 **128,807,306**，少了約 1100 萬的數量，比直接改 kernel size 還要少!但可惜的是，此時正確率也在 55%上下徘徊了，所以如果再繼續更改 kernel size 可以更進一步將低參數量但可能會讓正確率無法達標，所以我就沒有再做下去了。

但也可以藉由上面實驗看出，在減少參數量上，和改讓 kernel size 是 1*1 的小 kernel 比起來，直接減少 channel 大的 convolution layer 是更有效益的做法。

3.超參數調整：

這裡主要討論優化 function 的調整，這次嘗試了 adam 和 SGD 兩種方法，發現神奇一件的事，使用 adam 訓練時無法收斂，但 SGD 卻能正常的訓練成功，之後上網查詢發現，adam 在某些情況下會有泛化問題及訓練無法收斂，也許剛好就是碰上這種情況(文章沒有解釋為何無法收斂)。故這次選擇使用 SGD，而 SGD 相關參數我選擇使用 lr=0.01, momentum=0.9。

Loss function 部分則是用 crossentropy loss，主要原因是他是 paper 中所使用的 loss function 而如果隨意更改可能訓練會失敗，所以我這次沒有調整他。

momentum 部分 0.9 是 paper 所提供數值，我有試著用 0.5、0.1 實驗看看，結果 0.1 訓練不起來，0.5 正確率只有 47%。

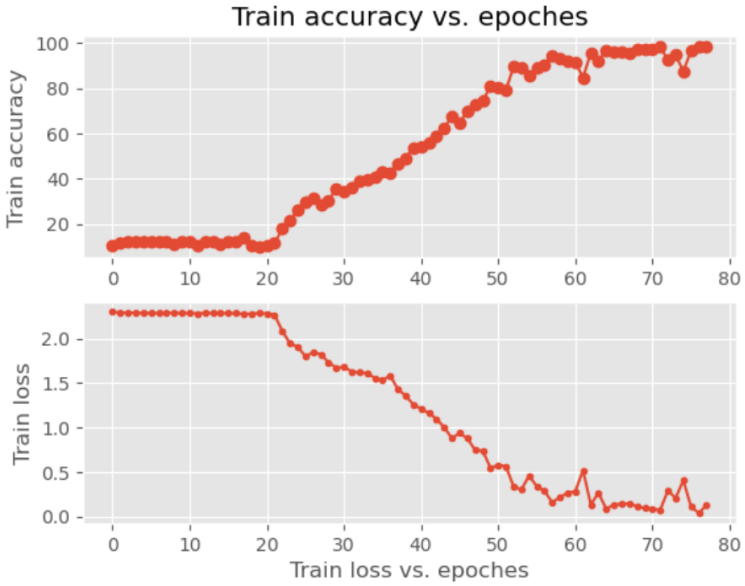
而 learning rate 部分我有測試 0.1,0.001,0.05 等數值，結果 0.1,0.001 直接訓練失敗無法收斂，0.05 要在 epoch 超過 70 次才開始學得到東西，所以最後選擇用 lr=0.01。

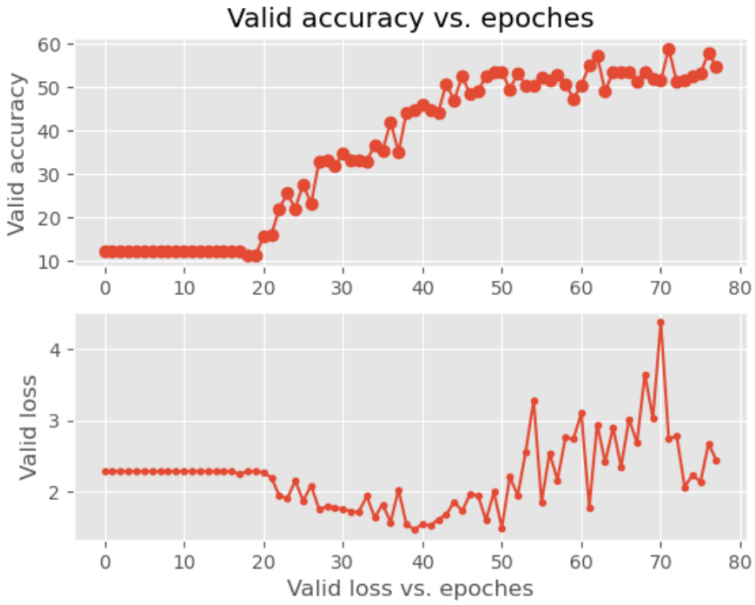
另外 batch size 部分越小代表參數更新越快但也越容易不穩，在這次 dataset 較小的情況下，我認為 batch size 要小一點而讓參數盡量多更新幾次，所以選擇 16，但之後試 batch size=32,64 accuracy 也沒差很多，所以我想這不是很大的影響要素，另外這是在類似 VGG11 model 實驗下做的結果，以原本 VGG19 下由於會跑出 out of cuda，所以最大只能用 batch size =8。

而 epoch 部分是我在這次遇到的一個大問題，我有設計在每一次 epoch 結束會 output 出 train valid 的 accuracy 和 loss 以及各 class 的正確率，發現前面至少 20 次的 epoch 的 valid set 都是一種 class 正確率 100%，其他都 0%，要隨著數量到一定次數 model 才會開始學到東西各 class 正確率逐漸提升，而由於我這次 code 沒有用存最好的那次 epoch 結果，所以就是選擇大概範圍，約在 epoch 在 75~80 次效果是最好的。

4. Results

這次將 training and validation loss curve, training and validation accuracy curve 分別儲存成圖並貼到這，





可以看到不論是 train dataset 還是 valid dataset 隨著 epoch 數量增加，loss 逐漸下降，而 accuracy 逐漸上升。

5. Problems encountered and discussion

我認為在減少參數量這個任務遇到遇到的最大問題，第一個是 dataset 的數量太少，導致訓練結果不太好，我首先有先試過 pytorch 內建的 minst dataset 在 epoch 第二次結束就又超過 90%準確率，這實驗是為了確保我的 model 架構是可以跑的，但同樣的 code 用到這次要用的 dataset 上面卻變成要至少 epoch 25 次以上才慢慢地學到東西，到約 70 次才能穩定，而這點和 batch size 似乎沒有太大關聯，我測試過各種 batch size(8 16 32 64)，原本以為減少 batch size 增加參數更新速度有用，結果還是都是一樣沒有影響，之後看了 minst dataset 的 trainset size 發現大概是這次任務的 30 幾倍，所以之後想說或許 vgg 就是要看到這樣的數據量後才能開始學到東西。

另外由於我每次訓練有把不同的 class 正確率印出來，在上面的討論中我認為還沒學到東西指的是只有一個 class 是 100%正確率其他都是 0%，這種情況下我認為還沒 train 起來，網路還在 warm up 的情形，但我在還是 vgg19 架構時做的一些實驗我的 epoch 設了 100 次都還是這種情況，我是把他當成無法收斂訓練失敗，我推測這種情況是因為網路太深造成訓練不成功(因為我之後把 layer 減少在 50 次左右 accuracy 就慢慢提高)，想詢問助教這種情況是沒有收斂還是因為 dataset 太小所以其實我把 epoch 設的更大就能解決，或是時我的 model 哪裡有設計問題？

另外在設計架構上，由於這次是以參數量為主，但我一開始還不太確定那些是會影響參數量的因素，像是要不要 dropout 、要不要 stride、padding 大小之類的，所以變成花了大量時間土法煉鋼，一項一項去測試，但好險最後結果還不錯，這次經驗也讓我以後遇到相關問題能知道該如何解決。

這次的 project 讓我對神經網路有更多的了解，以往大多是套用 pretrained 好的架構直接使用，少有重頭開始訓練的機會，所以我覺得這會是一次很棒的經驗。