Overview

這次的 Lab 我們針對胸腔的 X 光照片去偵測受測者是否患有肺炎，簡單觀測一下我們的 dataset，資料分布呈現如下：

	INFECTED	NORMAL	TOTAL
TRAIN	3944	1342	5286
TEST	?	?	624

可以看到整體而言，INFECTED 的訓練資料比 NORMAL 來的多，若是手動切 validation set 的話（後面將會提到）要特別留意這一點。另外，這次要求要用 Pytorch Library 去建立自己的 nn 模型對影像做二分類。

Approach Methods (Referenced Literatures)

不幸的現在正值 cov-19 肺炎，所以也因此最近陸陸續續的有更多相關的研究出現。首先，我有蒐集一些相關的 paper 閱讀，了解並參考別人所用的 model 和訓練方式，之所以會這樣做是因為我覺得一來可以培養自己閱讀 paper 的能力，二來，可能會學到意想不到的訓練和資料前處理技巧。Paper 閱讀的部分我選擇了兩篇：

1. An Efficient Deep Learning Approach to Pneumonia Classification in Healthcare (Stephen et al., 2019)

2. A Novel Transfer Learning Based Approach for Pneumonia Detection in Chest X-ray Images (Chouhan et al., 2020)

簡單述說一下兩篇論文的分類架構和內容：

- ***(一)第一篇論文***是自己建構一個 convolution 架構對影像做分類，如下圖。作者們利用了四層的 convolutional layer 加上 ReLU 和 Max Pooling，然後將結果 flatten 然後通過兩層 dense layer，最後用 sigmoid 對影像做分類。

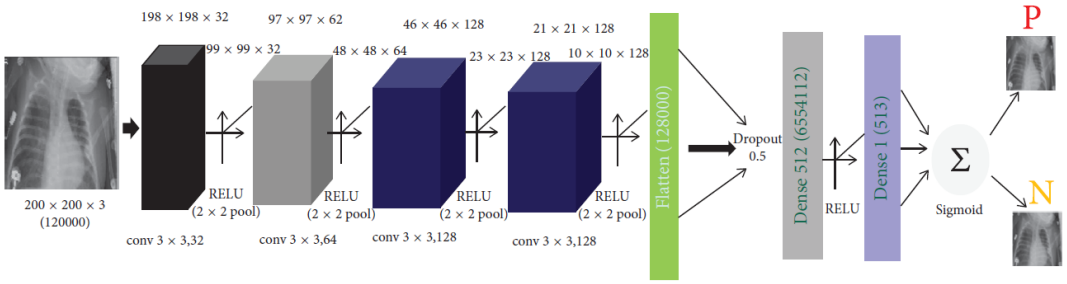

FIGURE 3: The proposed architecture.

因為他們利用的 training set 也只不過三千多張將近四千張圖片而已，所以在過程中，他們也有利用 data augmentation 的方式將 dataset 充胖，而所使用到的手法如下：

TABLE 1: Settings for the image augmentation.

Method	Setting
Rescale	1/255
Rotation range	40
Width shift	0.2
Height shift	0.2
Shear range	0.2
Zoom range	0.2
Horizontal flip	True

最後，文章有去探討利用不同的影像大小（resize 的維度）會有什麼樣的訓練效果，可以從下圖中看到 200 的圖片大小可以產出最好的訓練結果，雖然感覺其他的 data size 效果也不錯。

TABLE 3: Performance of the classification model on different data sizes.

Data size	Training accuracy	Validation accuracy
100	0.9375	0.9226
150	0.9422	0.9343
200	**0.9531**	**0.9373**
250	0.9513	0.9297
300	0.9566	0.9267
Average	0.94814	0.93012

但是我在仿效這篇論文的架構自製一個類似的 convolutional network 發覺效果並不怎麼好，在自己 local 的 validation set 準確度到七十幾，但在 Kaggle 上只有六十幾接近七時而已，並不是很成功。我覺得可能的原因是因為可能使用的資料集不同，外加文中作者所使用到的 data augmentation 手法，我不確定我在 pytorch 裡面有沒有做到很好的仿效，或許這也是影響的因素之一。但是，我覺得從這個 approach 裏頭我學到了很重要的一個技巧：就是當資料量不夠的時候，可以使用 data augmentation 的方法對影像做一些處理擴充自己的資料量。

(二)第二篇論文則是使用 pretrain model 來做影像分類，我接下來會選擇這個做法是因為考量到或許是因為自己的架構從頭學習有礙於資料量和資料特質，訓練結果不如理想，如果可以利用 Transfer Learning，用一些 pretrain 好的 convolution model 參數來做影像分類，或許效果會比較好。文中作者分別用到了：AlexNet、DenseNet21、InceptionV3、ResNet18、GoogLeNet 這幾個架構，分別對資料集進行預測，然後利用 majority voting 的方式進行 classification，但是因為一次把所有架構都拿來做 training 有點工程浩大，外加有些架構自己建的過程中頻頻遇到問題，所以最後先選擇 paper 中效果最好的 resnet18 做 transfer learning。

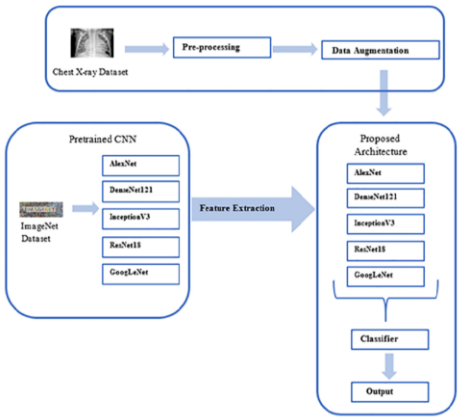

Figure 1. Outline of the methodology.

可以從下圖中看到是利用 resnet18 最後的 layer4（其實其中還包了很多 convolutional layers，所以有些人會將其稱作 block），將這個還有後面的 fully-connected network 中的參數根據手上的資料及做 update，至於其他層就將其 ”freeze”，也就是不對它進行 back-propagation 的 update，主要利用 pretrain 好的 convolutional layers 來幫助我們更有效的辨識手中資料。

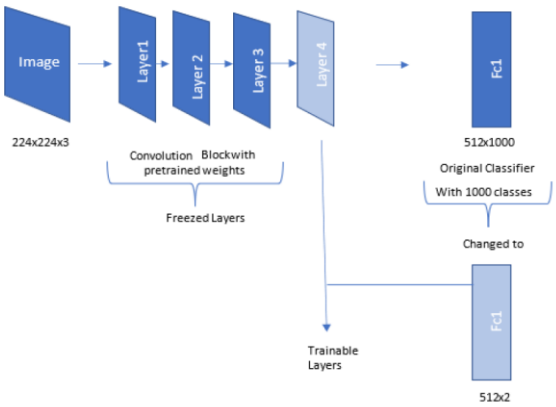

Figure 6. ResNet18 architecture with "frozen" and trainable layers.

可以看到文中若是利用作者們用所有架構做 majority voting 建出來的 ensemble model 效果會是最好的，明顯高過其他單一的模型，但就個體表現而言 ，ResNet18 整體表現是最好的。

Table 2. Comparative results for each model on the test set.

Model	Epoch	Recall (%)	Precision (%)	AUC (%)	Test Accuracy (%)
AlexNet	200	98.97	90.21	97.83	92.86
DenseNet121	100	99.23	91.18	98.78	92.62
InceptionV3	100	98.46	90.30	97.33	92.01
GoogLeNet	50	99.48	90.44	98.29	93.12
ResNet18	200	99.48	91.58	99.36	94.23
Ensemble model	–	99.62	93.28	99.34	96.39

我在參照第二篇論文的時候有明顯的進步，預測上精準度馬上拉高了 10~20%，所以很明顯的，以我的經驗來說，利用 pretrain 效果真的會比從頭訓練到尾來的好，接下來的討論和分析會著重於我所嘗試的第二個 approach，也就是上面所提到的第二篇論文。

Description of Model

整體的 model 架構有在上面先說明（Resnet18），而在 hyperparameter 的部分，我利用 Adam 作為我的 optimizer，這樣在收斂的時候可以更快也可以更精確，需要調整的參數也相較沒有那麼多，只需要調初始的 learning rate 為 10^{-3}，接下來就讓 Adam optimizer 在每個 epoch 去調整我的 learning rate 大小更新參數。

Difficulties and How I Solved Them

整個 lab 的訓練過程中，除了進行第一個 approach 花了不少心血和經歷了大大的挫折外，另一個讓我花一點時間的就是 data augmentation 的部分。兩篇論文裡用到的資料量跟我們沒有差很多，所以作者都有對資料集進行 data augmentation，而我發現做和不做 data augmentation 真的會讓 validation 精準度差個 3~5%，所以其實有做 data augmentation 真的蠻重要的。Pytorch 是利用 transforms.Compose 對資料進行 data augmentation，而做 transforms 這件事我們只有在讀進影像的時候才能做，但是問題就在於，我們原先的方法是將資料一併讀入、transform 完再做 split，而 data

augmentation 只能做在 training set，不能做在 validation set，本來想寫一個 pytorch 的 data loader 來 handle 這個問題，但考慮到自己還是對這個 Library 不熟悉，所以最後，我選擇用 linux 的一些指令，除了原本的 train、test 兩資料夾，再建立第三個 val 資料夾放 validation 資料，而資料的搬移就利用像以下的指令：

```
n$ ls | shuf -n 360 | xargs -i mv {} ../../val/INFECTED/
```

把 train test split 有感染和未感染的比例分別算好，將 train 裏頭的資料進行 shuffle 然後將算好的 validation 個數移到新的資料夾，而在 load data 的時候分別向這些資料夾拿資料：

```
train_dataset = datasets.ImageFolder(root=TRAIN_DATA_PATH, transform=train_transform)
val_dataset = datasets.ImageFolder(root=VAL_DATA_PATH, transform=val_transform)
```

這樣就可以對 train 資料做 data augmentation 了！

Improve Accuracy

在優化模型的部分，影響最大的就是改用 pretrain 模型後，accuracy 真的被拉高了不少，其次就是先前提到的 data augmentation，我利用了以下手法：
（因為 pytorch 是在每次跑 batch 的時候去決定要不要做這個 random 動作，所以等效上就等於多加了額外處理過的資料）

1. RandomHorizontalFlip：針對資料做隨機的左右顛倒
2. RandomResizedCrop：把影像放大擷取
3. RandomAffine：對邊界進行伸縮
4. RandomRotation：將影像旋轉

Update

我後來又再回去改 approach 1 的 code，我發現其實在 classifier 的最後並不用建一個 sigmoid 的 activation function，這一拿掉 sigmoid 那一層，馬上就將 accuracy 衝到 81% (Kaggle 上)，雖然還是比 pretrained model 來的低 (85.737% on Kaggle)，但是我覺得這個準確度的差距值得探討，上網找了一些資料研究了一下這個問題後我發現，應該是因為我們在算 loss 的時候就已經有用到 cross entropy 了，而如果併著 sigmoid 使用的話，可能會影響到學習的進度 ，所以才會導致我一開始嘗試 approach 1 的時候效果那麼差，並不是一開始臆測的資料集異同問題（畢竟都是 X 光片嘛😇）。這個經驗讓我學到了在建立模型前真的要對自己的架構深入的了解，而如果要外加甚麼功能前，要先知道自己以前有放的東西會不會與其相衝突。