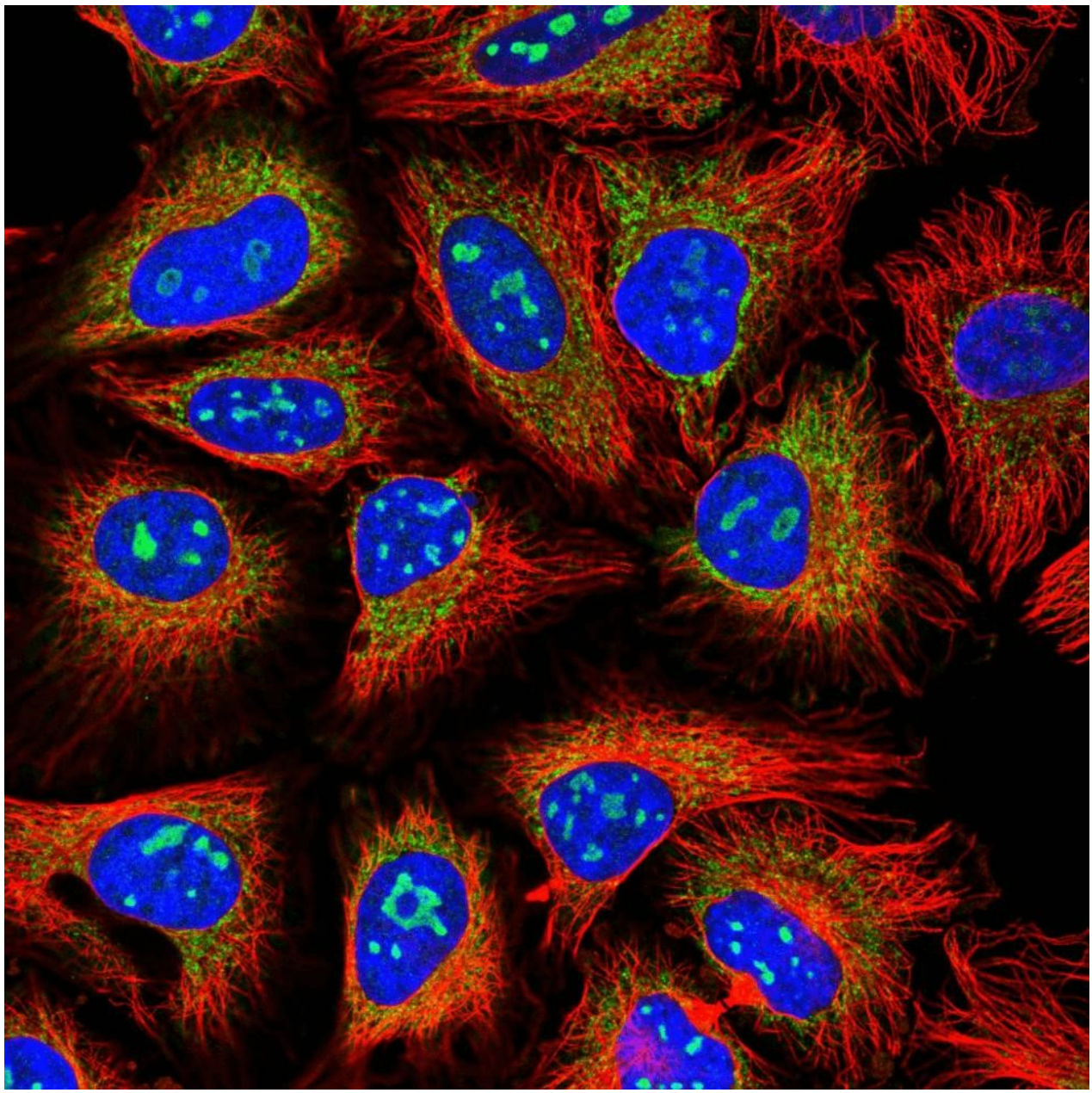
ML FINAL Report

Human Protein Atlas Image Classification

tags: `NTU_ML2018FALL`

Kaggle



https://www.kaggle.com/c/human-protein-atlas-image-classification/

(https://www.kaggle.com/c/human-protein-atlas-image-classification/)

組別：ChineseTaipei中O台北

B04611017 林弘曄
R07943018 周育辰
D06922023 顏志軒
R06922051 鍾詠先

GITHUB

https://github.com/linnil1/ML2018FALL_FINAL (https://github.com/linnil1/ML2018FALL_FINAL)

題目

這個題目需要從顯微鏡照片中判斷出有哪些蛋白質。例如下圖是一張有不同細胞的顯微照片。

這個題目的困難點在於，過去蛋白質分類都是少數細胞種類中判斷出單一型態的蛋白質。而現在，為了更了解人類細胞的複雜性，需要從多種細胞中判斷出多個型態的蛋白質。

一開始選這個題目沒什麼，就只是覺得能參加 kaggle competition 很酷

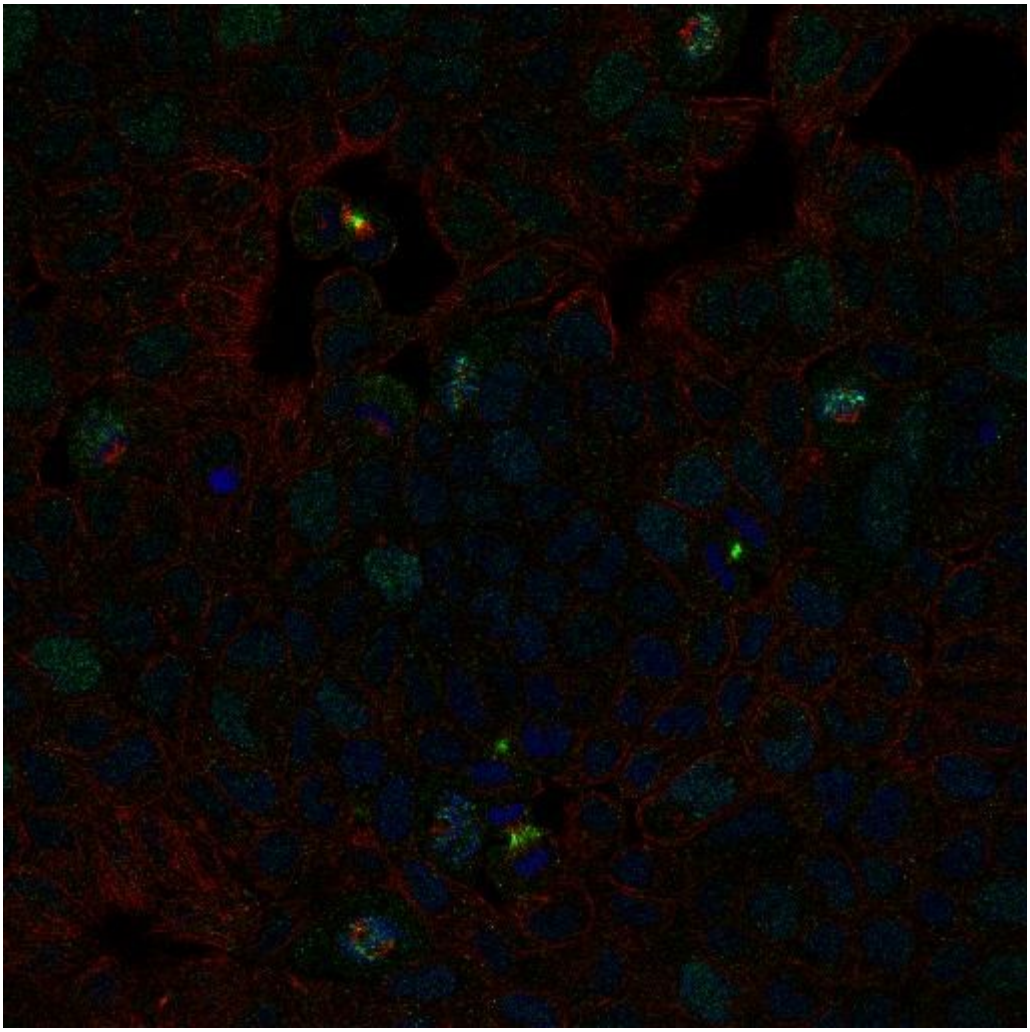
Data Pre-processing

Pre-processing用的是一個Python的套件Augmentor。

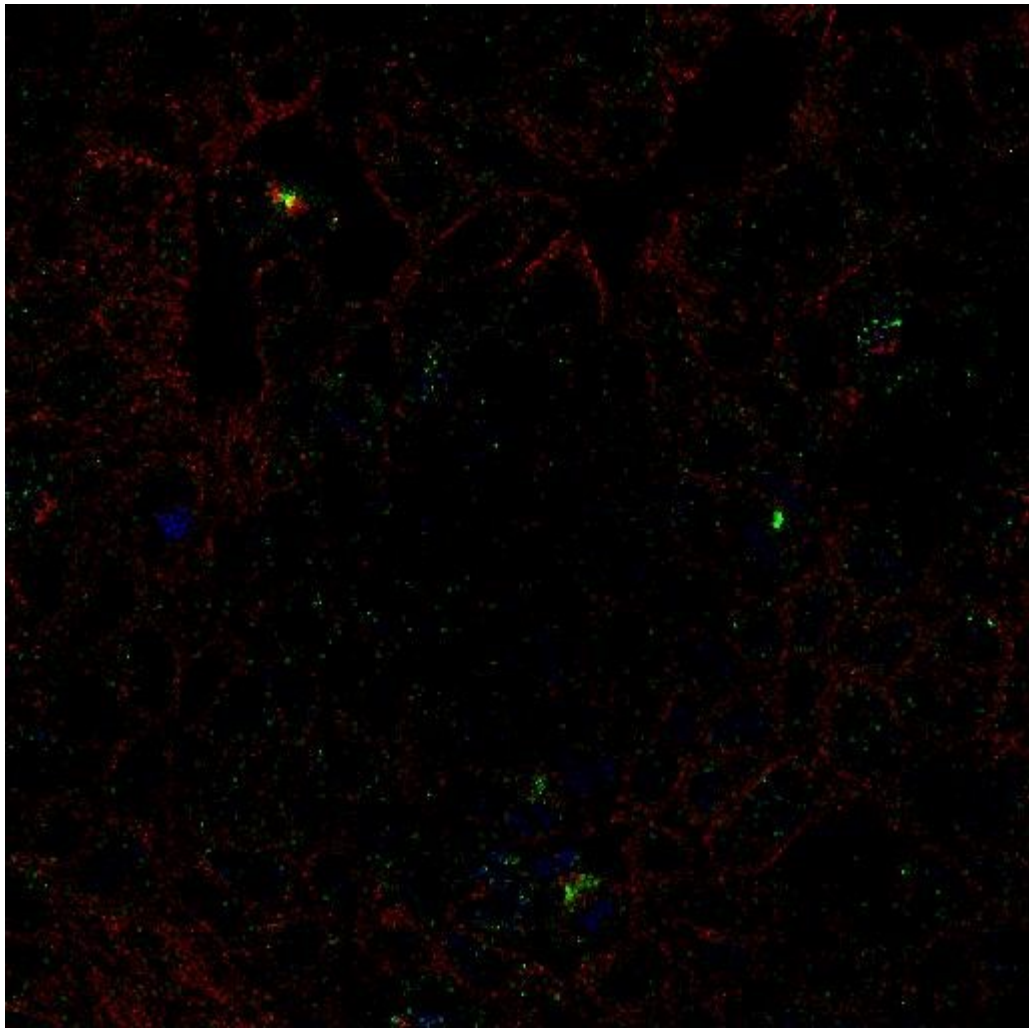
這個套件可以將圖片翻轉、旋轉、縮放、變形、投影、調整亮度、加上random noise等等，來增加訓練資料的數量，提高訓練的準確性。

一開始有嘗試過各種各種操作，包含上圖的投影等等，後來發現 Argumentor 的旋轉使圖片失真很多，所以最終版本圖片只有做 左右 or 上下翻轉。

For example:
原圖



僅旋轉



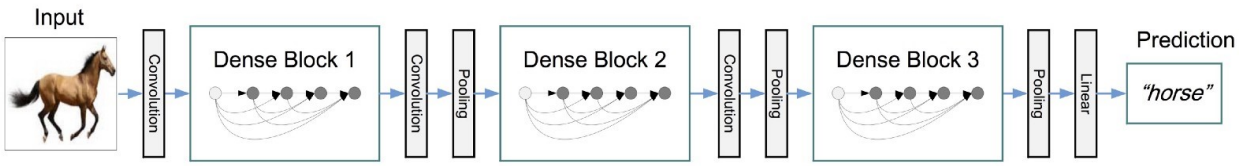
整張圖連人類看都覺得不像了，更何況是機器。
我覺得是它旋轉時，使用外差，然而 0 很多， 所以特徵都被背景消滅光光

以結果論，會差 0.1 的 F1 score，一開始不知道傻傻的train了很久QQ

Backbone

一開始有試過Densenet121，但是由於Densenet121不夠穩定，所以最後使用的是 pretrained 的 Densenet201。

https://arxiv.org/pdf/1608.06993.pdf (https://arxiv.org/pdf/1608.06993.pdf)



兩種model的詳細架構如下表：

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Table 1: DenseNet architectures for ImageNet. The growth rate for all the networks is $k = 32$. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

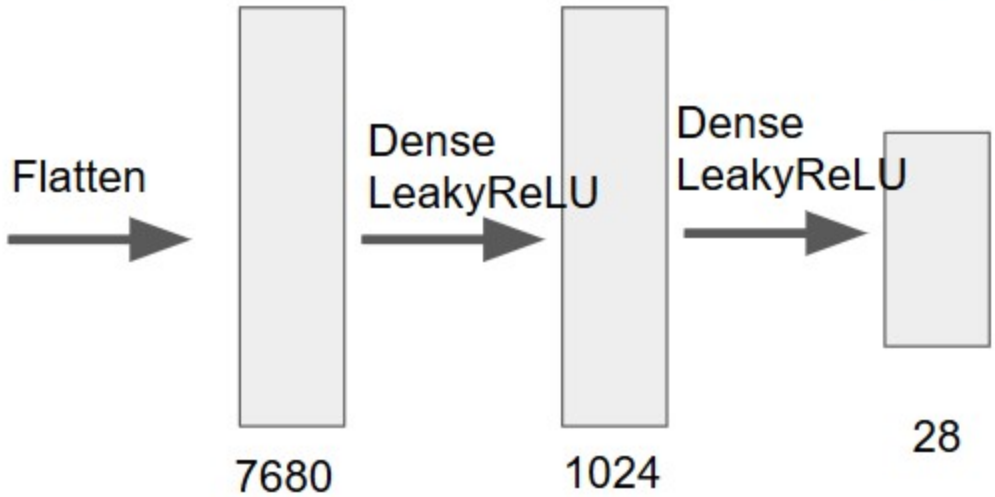
和Densenet121相比，Densenet201在比較接近output端的地方用了比較深的Dense block。

因為Densenet201有 RGBY 四個input，但一開始只有3個input，所以要多增加一個layer像這樣

```
nn.Conv2d(4, 64, kernel_size=7, stride=2, padding=3, bias=False)
```

然後因為輸入圖片512x512，太大張，所以使用 FiveFold(256)，也就是把角落跟中間截出 256x256 五張圖片

因為原本densenet input 224x224，所以後面的weight 也要改掉，如下圖



最後28維接 Sigmoid 變成 0~1 之間的值.

每個 Dense 都有一個 dropout，這裡是設 0.4 0.5。

最後再將五張圖片的的 output 取 max 來 當作 一張的 output.

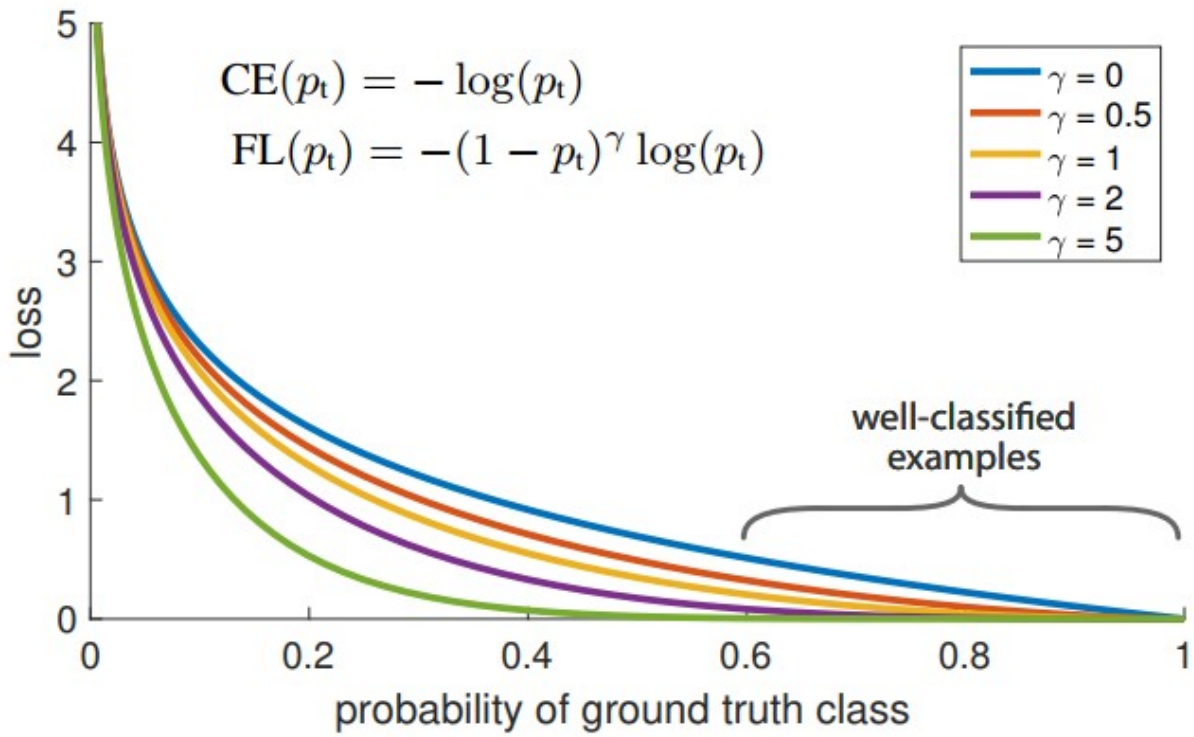
為什麼要這麼做
因為 看到
https://www.kaggle.com/kwentar/visualization-examples-of-each-class-in-rgb
(https://www.kaggle.com/kwentar/visualization-examples-of-each-class-in-rgb)
發現一張圖片通常都包含很多同樣的 protein

所以 切五份 是很合理的，但有些會有少量protein在裡面，所以取 max 也很合理。

Loss Function

由於是multi-class classification的問題，
且每個class的資料量很不平衡，
因此使用 RetinaNet 的 FocalLoss
(https://arxiv.org/pdf/1708.02002.pdf (https://arxiv.org/pdf/1708.02002.pdf))

因為是 imbalance 的 資料，所以在這種狀況下
$\alpha = 1$ & $\gamma = 2$ 時最好



Training

Training 的時候， 要注意 先把 DenseNet 剩下的 weight fix 住
然後一定要 train 好 train 滿，不然整個下去 train validation 會不好
我用至少 15 epoches 去train (最後大蓋有f1=0.3)

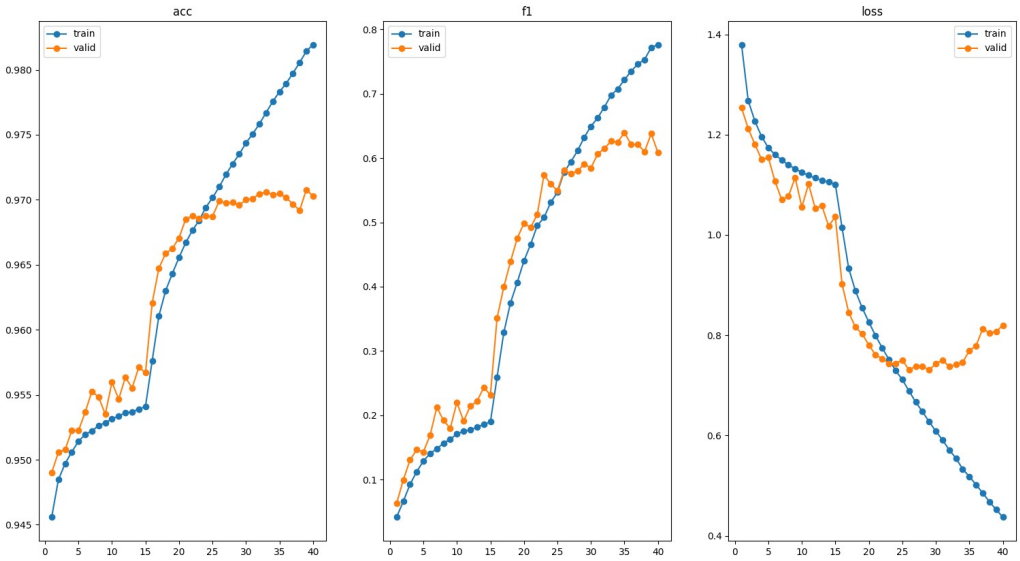
之後再全部一起 train 加上並在最後兩層加上 Dropout 0.4 and 0.5 train 個 20 epoches.

~~注：一個epoch 要 train 至少 1518 s 累到不想train力~~

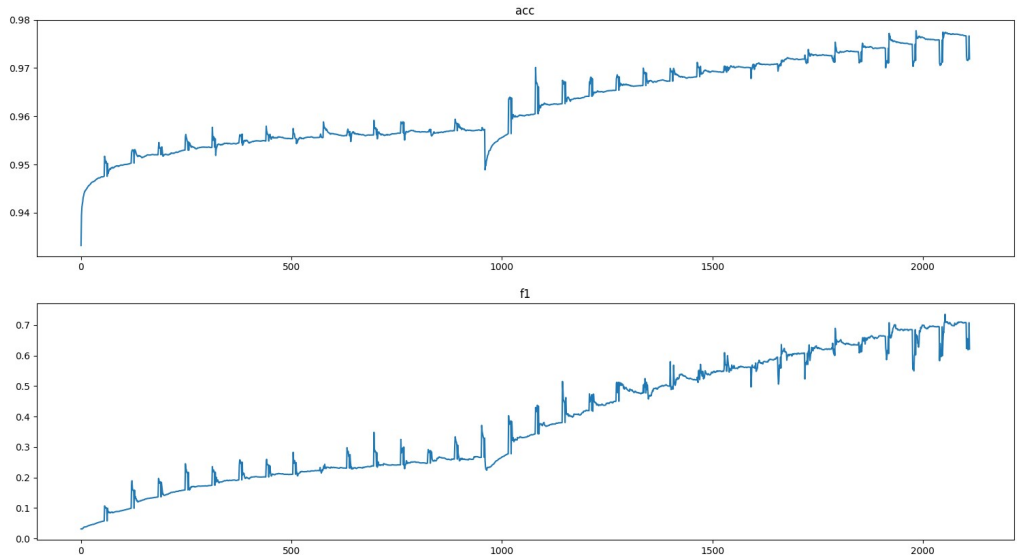
這樣單個 model 的 Kaggle LeaderBoard 會在 0.463 左右

注:
前面 train 2 個 epoch， 最後大概只有 0.28
前面 train 10 個 epoch， 最後大概只有 0.38
前面 train 18 個 epoch， 最後大概只有 0.46(目前弄出最大)
後面就飽和了

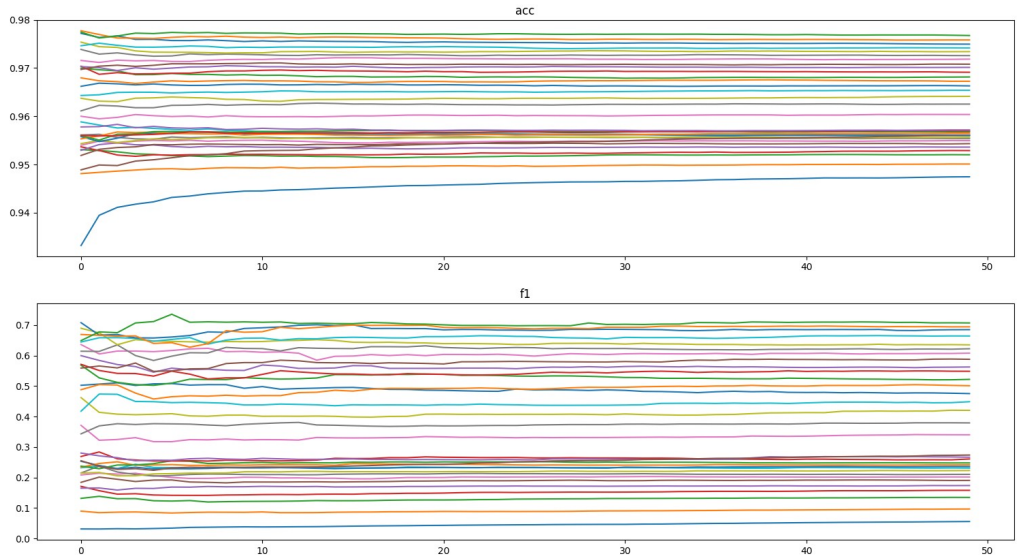
training history 大概長這樣(先train15個2的)



training per batch
每次一開始的時候 都會比較高，然後越降越低



但是每次都時有逐漸升高(不明顯)，所以才要train 約 30 epoches.
(下圖每條不同顏色的線分別代表不同次epoch跑出的結果)



補一個數據

```
epoch:19/20
Train: 140000/140000 100.00% acc: 0.9571 loss: 1.0103 f1: 0.2721828520298004
Valid:  15360/ 15360 100.00% acc: 0.9577 loss: 1.0122 f1: 0.30623894929885864
Time: 1277 (s)
epoch:40/50
Train: 140000/140000 100.00% acc: 0.9793 loss: 0.4900 f1: 0.7561458945274353
Valid:  15360/ 15360 100.00% acc: 0.9701 loss: 0.7645 f1: 0.63612961769104
Time: 1482 (s)
```

Ensemble

然後 ensemble 起來， 取最後一層 output 最大值會沒有效果，要取 mean 會比較好，
最後共有5個 model 落在 0.498 也大概飽和ㄌ。
kaggle result 0.473

如果是 voting，雖然 public score 差，但是 private score = 0.476

後記

- 原本想要用 external data (https://www.proteinatlas.org/) train 的，沒研究，就算了。
- 最後一層用 XGBoost 取代最高也才 Public LB 也才 0.45。(或者是我使用錯誤)
  https://xgboost.readthedocs.io/en/latest/parameter.html
  (https://xgboost.readthedocs.io/en/latest/parameter.html)
  我把原本Kaggle score 最高的model 拿來抽出最後一層 1024 * 5(個) -> 每張
  然後在用 XGBoost train
  因為只有 binary logistric， 或是只能有 single label 的 output 的 softmax， 所以我分開，所有的 class 都做一次，
  得到的結果是 validation f1 = 0.8 以上， 可是丟上 kaggle 只剩 0.42
  不管是 把 正的weight 調大調小，或者是把 Regularation 調很大，最高就 0.45。
- 不同的 learning rate 差異會非常大，拿不同的來 ensemble 效果特別好。
- 不同的 class 用不同的 threshold (上次分享聽到的)， 不過後來就沒做了。
- 因為我第一次使用 Pytorch，跟 HW3 不一樣的是， 要自己找圖形 Augument 套件，找到這個套件似乎堪用，但是好像似乎有些問題。

Reproduce

請參見 `Readme.md`

References

1. Image augmentation library in Python for machine learning. https://github.com/mdbloice/Augmentor (https://github.com/mdbloice/Augmentor)
2. https://www.kaggle.com/iafoss/pretrained-resnet34-with-rgby-0-460-public-lb (https://www.kaggle.com/iafoss/pretrained-resnet34-with-rgby-0-460-public-lb)
3. batch normalization paper (https://arxiv.org/pdf/1502.03167.pdf)
4. Focal Loss for Dense Object Detection (https://arxiv.org/pdf/1708.02002.pdf)
5. XGBoost (https://arxiv.org/abs/1603.02754)
6. Human Protein Data (https://www.proteinatlas.org/)