

ML Final Project Report

1. 隊名及隊員

隊名: NTU_r07943004_荳荳與水豚

隊員: R07943004 莊育權、R07943023 楊仲萱、R07943123 馬咏治

2. 題目

Human Protein Atlas Images Classification

3. Introduction

A. Database [1]

此 Database 會給多個細胞圖，需要根據這些圖片去預測出裡面有哪些不同種的蛋白質。蛋白質種類總共有 28 種，如下表所示：

| Number | Class Name | Number | Class Name |
|--------|---------------------------|--------|-------------------------------|
| 0 | Nucleoplasm | 14 | Microtubules |
| 1 | Nuclear membrane | 15 | Microtubule ends |
| 2 | Nucleoli | 16 | Cytokinetic bridge |
| 3 | Nucleoli fibrillar center | 17 | Mitotic spindle |
| 4 | Nuclear speckles | 18 | Microtubule organizing center |
| 5 | Nuclear bodies | 19 | Centrosome |
| 6 | Endoplasmic reticulum | 20 | Lipid droplets |
| 7 | Golgi apparatus | 21 | Plasma membrane |
| 8 | Peroxisomes | 22 | Cell junctions |
| 9 | Endosomes | 23 | Mitochondria |
| 10 | Lysosomes | 24 | Aggresome |
| 11 | Intermediate filaments | 25 | Cytosol |
| 12 | Actin filaments | 26 | Cytoplasmic bodies |

| | | | |
|----|----------------------|----|--------------|
| 13 | Focal adhesion sites | 27 | Rods & rings |
|----|----------------------|----|--------------|

而每張圖片當中會有四個 channel，分別是紅、綠、藍和黃，而這四種 channel 分別代表著 microtubules、the protein of interest、nucleus 和 endoplasmic reticulum 的資訊。

B. Motivation

由於我們三個人都有做跟生醫相關的研究，此 Database 已經事先研究過裡面分類的種類有嚴重 Class imbalance 的問題，而這個問題也時常在生醫訊號或者生醫影像上做 Classification 產生，主要原因是每個疾病或者現象，通常都會有一種類比較容易出現，而有些比較特殊的疾病一定會比較小機率出現，同時如果醫生要對每一筆資料去做 label 實在太耗時間，對於這些問題，就會需要用其他技巧彌補這個不足。因此我們三位覺得以這個當作我們 Final Project 的題目，也對於未來我們的研究有所幫助！

4. Data Preprocessing

A. RGBY

由於此 Database 中的每個影像都有四種不同顏色的 channel，每個 channel 也分別帶有不同的生物特性，而通常 library 裡 pretrained 好的 model 輸入維度只有三種 channel(紅、綠、藍)，因此在這部分我們會嘗試以下兩種方式：

1)由於黃色是紅色加綠色，因此我們將黃色 channel 訊號一半的值給紅色，另一半給綠色。這樣的優點是我們不用另外去更動到 pretrained 好的 model，缺點是黃色所代表的資訊可能就會遺失掉。

2)不管是我們自己所設計的 model，或者是 library 裡 pretrained 好的 model，我們都將其輸入 data 的 channel 設定成 4，這樣就可以確保黃色 channel 的資訊不會不見。

B. Data Augmentation

由於此 Database 所帶有的資料量，對於訓練一個複雜的 model 來說，仍然過於小，因此很容易就會產生 overfitting 的現象。在處理資料過少這方面的問題，再把資料丟進去 model 做 training 的時候，會先經過 Data Augmenter，如下圖程式所示：

```
def augmentor(self, image):
    augment_img = iaa.Sequential([
        iaa.OneOf([
            iaa.Affine(rotate=90),
            iaa.Affine(rotate=180),
            iaa.Affine(rotate=270),
            iaa.Affine(shear=(-16, 16)),
            iaa.Fliplr(0.5),
            iaa.Flipud(0.5),
        ])
    ], random_order=True)

    image_aug = augment_img.augment_image(image)
    return image_aug
```

此 augmentor 函式會將影像做前置處理，例如旋轉、移動、翻轉等處理，因此就會產生很多被前置處理過的影像，而這些影像就可以用來增加訓練用的資料量。

5. Model Description

A. 類 DenseNet 的 CNN

一開始我們參考類似於 CVPR2017 最佳論文[2]中所提出的 DenseNet 架構，運用 keras 的 concatenate 將後面一層的網路架構跟前面的網路架構都去做連結，以下為我們的架構程式碼：

```
init = Input(input_shape)
x = BatchNormalization(axis=-1)(init)
x = Conv2D(8, (3, 3), activation='relu')(x)
x = BatchNormalization(axis=-1)(x)
x = Conv2D(8, (3, 3), activation='relu')(x)
x = BatchNormalization(axis=-1)(x)
x = Conv2D(16, (3, 3), activation='relu')(x)
x = BatchNormalization(axis=-1)(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Dropout(dropRate)(x)
c1 = Conv2D(16, (3, 3), padding='same', activation='relu')(x)
c2 = Conv2D(16, (5, 5), padding='same', activation='relu')(x)
c3 = Conv2D(16, (7, 7), padding='same', activation='relu')(x)
c4 = Conv2D(16, (1, 1), padding='same', activation='relu')(x)
x = Concatenate()([c1, c2, c3, c4])
x = BatchNormalization(axis=-1)(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Dropout(dropRate)(x)
```

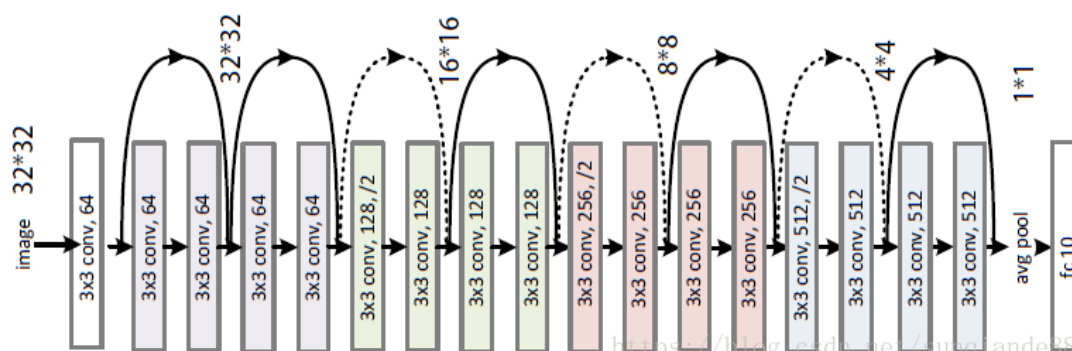
```

x = Conv2D(32, (3, 3), activation='relu')(x)
x = BatchNormalization(axis=-1)(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Dropout(dropRate)(x)
x = Conv2D(64, (3, 3), activation='relu')(x)
x = BatchNormalization(axis=-1)(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Dropout(dropRate)(x)
x = Conv2D(128, (3, 3), activation='relu')(x)
x = BatchNormalization(axis=-1)(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Dropout(dropRate)(x)
x = Flatten()(x)
x = Dropout(0.5)(x)
x = Dense(28, activation='relu')(x)
x = BatchNormalization(axis=-1)(x)
x = Dropout(0.1)(x)
x = Dense(28)(x)
x = Activation('sigmoid')(x)

```

B. Pretrained Resnet18

第二種 model 是使用 library 內建已經用 ImageNet 訓練好的 Resnet18，架構來自 CVPR2015 最佳論文獎[3]，架構如下圖所示 [4]:

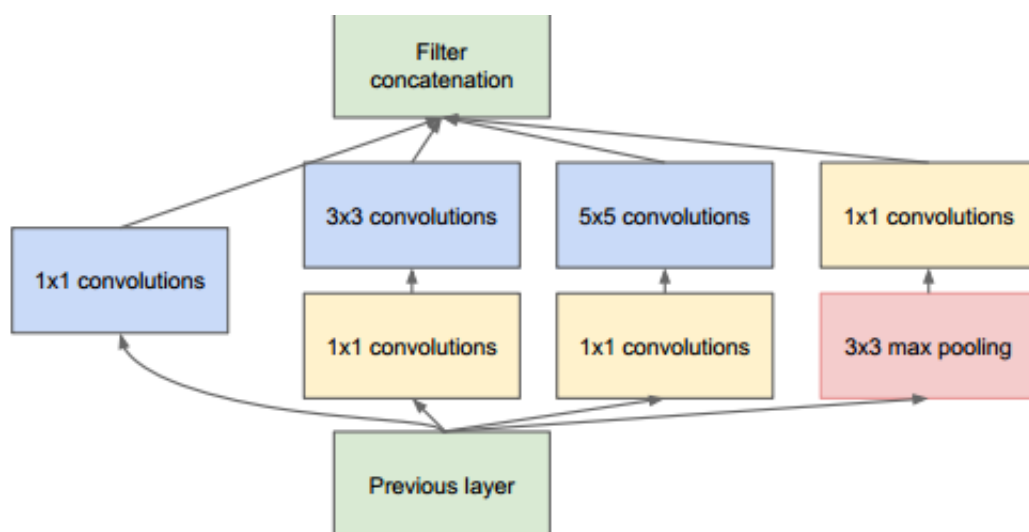


其基本精神為，假定某段神經網絡的輸入是 x ，期望輸出是 $H(x)$ ，如果我們直接把輸入 x 傳到輸出作為初始結果，那麼此時我們需要學習的目標就是 $F(x)=H(x)-x$ 。ResNet 的結構可以極快地加速超深神經網絡的訓練，模型的準確率也有非常大的提升。

C. Pretrained Inception V2

第三種模型架構是用 Google 所提出來的 Inception [5]，並且是使用已經用 ImageNet 訓練過的 model。其本身 Inception Google 有陸陸續續提出不同的版本，而其 Version 1 的基本精神為，傳統的 Convolution Network 是將 Convolution layer stack 在一起，而 Inception 最大的改變就是 Inception 模塊疊加的形式跟傳統的不太一樣。按論文裡面的說法就是，用 Inception（稠密的可利用的組件）近似一個稀疏結構。將 1x1, 3x3, 5x5 的 conv 和 3x3 的 pooling，stack 在一起，一方面增加了網絡的寬度，另一方面增加了網絡對尺度的適

應性。主要特點是提高了網絡內部計算資源的利用率，架構如下圖所示[6]:



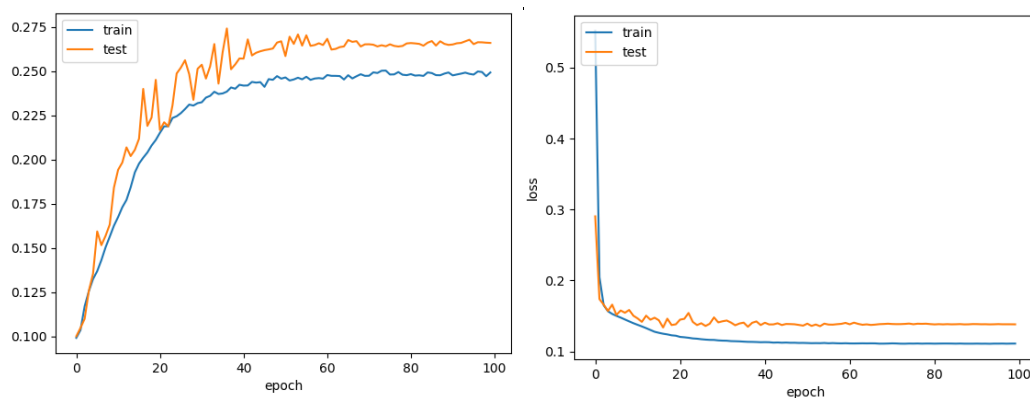
而 InceptionV2 又被稱之為 BN-Inception，其在 Version 1 的基礎上，進行了改進，一方面加入了 Batch Normalization Layer，減少了 Internal Covariate Shift（內部 neuron 的數據分佈發生變化），使每一層的輸出都規範化到一個 $N(0, 1)$ 的高斯分布曲線，另外一方面學習 VGG 用 2 個 3×3 的 conv 替代 inception 模塊中的 5×5 ，既降低了參數數量，也加速計算。

6. Experiment and Discussion

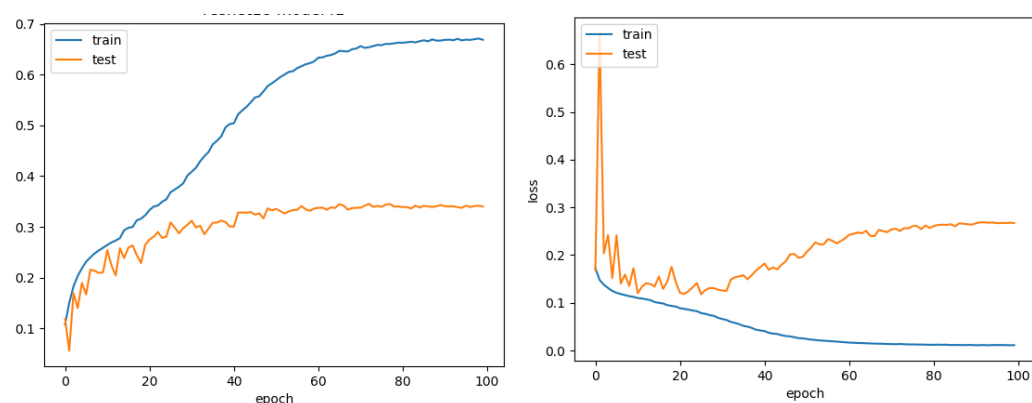
A. Different Classifiers

在這部分我們實驗了上述所提及的三個模型架構，看每個模型架構對於 f1 loss 的影響。

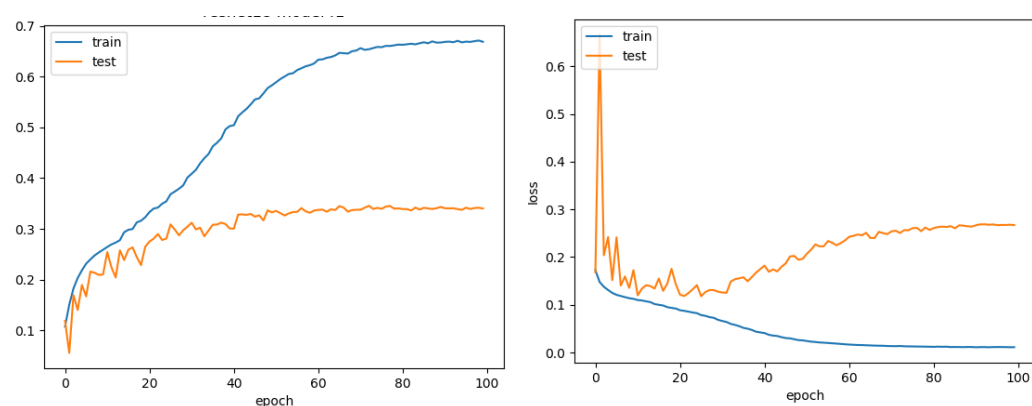
下圖為類 DenseNet 未被 pretrained 過的 CNN 在訓練過程中的 f1_score 和 f1_loss:



下圖為 ResNet18 在訓練過程中的 f1_score 和 f1_loss:



下圖為 InceptionV2 在訓練過程中的 f1_score 和 f1_loss:



由上面這六張圖可以發現以下兩點:

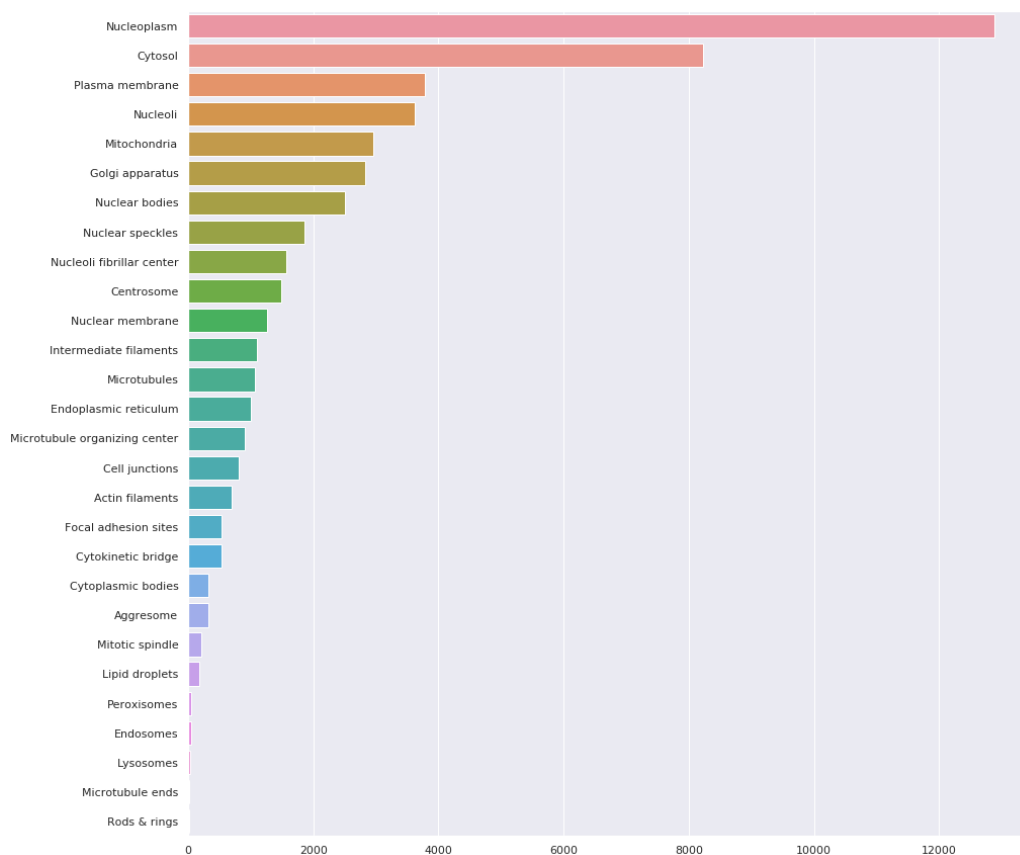
1) 後面兩個模型 ResNet 和 InceptionV2 表現得比沒有經過 pretrained 過 DenseNet，可以得知有經過 pretrained 過的模型，可能因為其權重的 initialization 是有意義的，比起沒有經過 pretrained 的模型權重隨機亂給的 performance 還來的好很多。

2) 後面兩個模型 ResNet 和 InceptionV2 雖然表現比較好，但可以很明顯地發現他們都有很嚴重 overfitting 的問題，而較為簡單的類 DenseNet 卻比較不明顯，主要可能是因為後面兩個模型複雜度太高，很容易就 over training。

| | Train | | Test | |
|-------------|----------|---------|----------|---------|
| | F1 score | F1 loss | F1 score | F1 loss |
| DenseNet | 0.247 | 0.12 | 0.261 | 0.14 |
| ResNet18 | 0.656 | 0.02 | 0.333 | 0.25 |
| InceptionV2 | | | | |

B. Class Imbalance

在一開始就有提到此 Human Protein Database 有很嚴重 Class Imbalance 的問題，而整個蛋白質種類出現的次數，如下圖所示[7]:



因此可以發現此 Database 的蛋白質類別有很嚴重 Class Imbalance 的問題，而一開始在嘗試訓練模型的時候，去看每個模型預測出來的結果，可以發現裡面所預測出來的類別，在 Database 很少出現的類別幾乎很少被預測出來，而比較常出現的前幾名，其實幾乎在蠻多圖上都可以看到的。造成這個現象大概原因就是某些類別在訓練資料中太少出現，而模型在訓練的過程中會一直看到前幾名一直出現的類別，因此前幾名的類別會被訓練的較好，且整個模型可能都被前幾名出現的類別索引導，因此較常出現的類別會在模型做預測的時候，較容易出現。

因此在處理 Class Imbalance 這方面的問題，我們這裡嘗試了兩種方法:

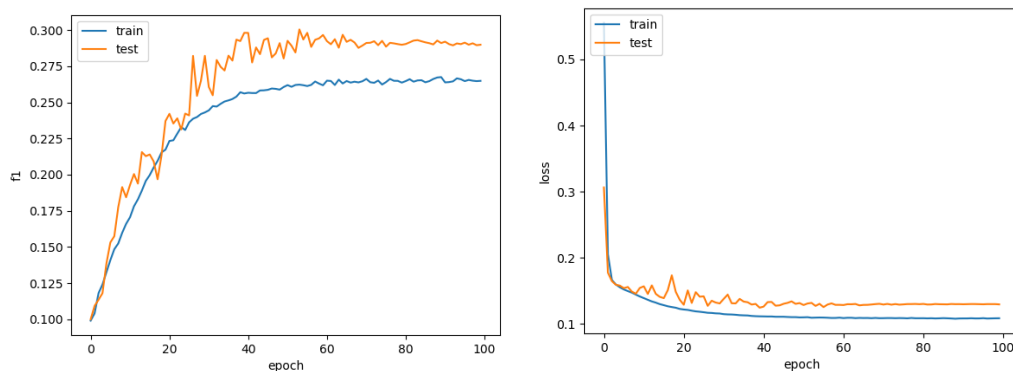
1) 在文獻上，比較常用到的方法就是對於每個類別的權重(Class Weight)去做調整。例如，一開始在訓練模型的時候，如果每個種類的權重都沒有去做更改的話，對於 Cost function 來說，每個類別只要犯錯的懲罰都是一樣的，但當如果給類別權重的話，如下程式圖所示:

```
weight = [1,1,1,1,1,1,1,1,1,4,4,4,1,1,1,1,4,1,1,1,1,2,1,1,1,1,1,1,4]
class_weight = {0: weight[0],
1: weight[1],
2: weight[2],
3: weight[3],
4: weight[4],
5: weight[5],
6: weight[6],
7: weight[7],
8: weight[8],
9: weight[9],
10: weight[10],
11: weight[11],
12: weight[12],
13: weight[13],
14: weight[14],
15: weight[15],
16: weight[16],
17: weight[17],
18: weight[18],
19: weight[19],
20: weight[20],
21: weight[21],
22: weight[22],
23: weight[23],
24: weight[24],
25: weight[25],
26: weight[26],
27: weight[27]}
```

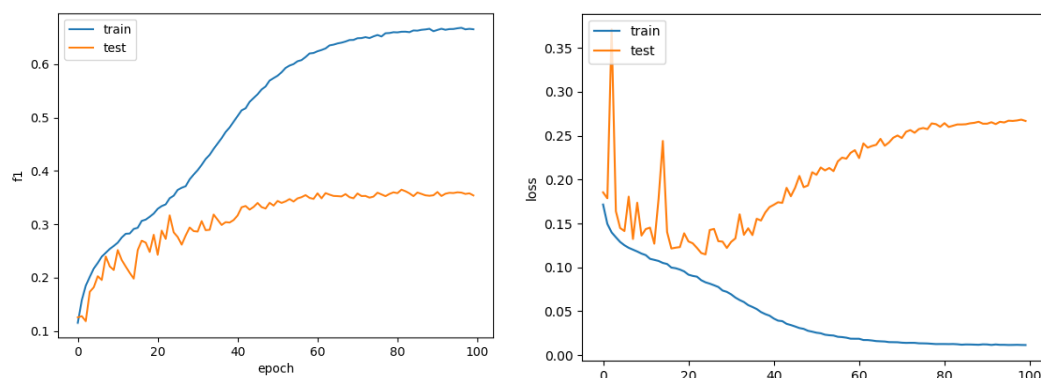
當最後一個類別只要犯錯，其所受到的懲罰就會是平常的四倍，也就代表著我們將較少出現類別在模型 Cost function 的重要性調高，這樣模型在訓練的過程中，為了降低 Cost function 就比較不會去忽略較少出現的 Class。

在這部分我們使用類 DenseNet 的 CNN 和 ResNet18 去比較，加上 Class Weight 之後，訓練模型的結果是否有比較好:

下圖為類 DenseNet 的 CNN 加上 Class Weight 後的 f1 score 和 f1 loss:



下圖為 ResNet18 加上 Class Weight 後的 f1 score 和 f1 loss:



可以去比較在 A 部分 Different Classifiers 的模擬結果和上面這些圖的模擬結果，可以觀察出加上權重之後，在訓練模型時的 f1 score 的確比沒有加上權重的 f1 score 還要來的高:

| | Train | | Test | |
|----------|----------|--------------------|----------|--------------------|
| | F1 score | F1 score w/ weight | F1 score | F1 score w/ weight |
| DenseNet | 0.247 | 0.261 | 0.261 | 0.285 |
| ResNet18 | 0.656 | 0.71 | 0.333 | 0.358 |

而我們去觀察模型預測出來的類別，發現原本在訓練資料較少出現的類別，在有加權重模型的 output prediction 出現的比例比沒有加權重的還來的多，代表對較少出現的類別加上權重，的確可以讓模型不會只去 fit 較常出現的類別，也可以來減緩 Class Imbalance 的影響。

2) 第二種方法為，通常經過模型預測出來的數值會介於 0-1 之間，而正常來說我們會將 0.5 當做一個臨界點，大於 0.5 就算是這個類別，而小於的話就是沒有這個類別。而在這方面我們去對每一種 Class 找出最適合他的 threshold，可以使得每個類別的 f1 score 最高，如下程式碼所示:

```

rng = np.arange(0, 1, 0.001)
fls = np.zeros((rng.shape[0], 28))
lastFullValPred = np.empty((0, 28))
lastFullValLabels = np.empty((0, 28))
for i in range(len(vg)):
    im, lbl = vg[i]
    scores = bestModel.predict(im)
    lastFullValPred = np.append(lastFullValPred, scores, axis=0)
    lastFullValLabels = np.append(lastFullValLabels, lbl, axis=0)

for j,t in enumerate(rng):
    for i in range(28):
        p = np.array(lastFullValPred[:,i]>t, dtype=np.int8)
        scoref1 = off1(lastFullValLabels[:,i], p, average='macro')
        fls[j,i] = scoref1

T = np.empty(28)
for i in range(28):
    T[i] = rng[np.where(fls[:,i] == np.max(fls[:,i]))[0][0]]
print(T)
np.save("threshold.npy",T)

```

這部分由於是在 Inference 的時候，去用 validation data 去找最能使 f1 score 最高的 threshold，因此就沒有圖可以放。但實驗出來的結果，的確會比沒有去選 threshold 的還來的好：

| | w/o threshold | w/ threshold |
|--------------|---------------|--------------|
| kaggle pubic | 0.329 | 0.343 |

C. Ensemble

為了能更加提高我們的 f1 score，在這部分我們使用 ensemble 的方法將不同種模型去做整合。但在這部分由於每個訓練出來的模型檔案太過龐大，於是我們在這採取另外一種方法，我們將兩種不同模型 output 出來的 ans.csv 去做 merge 的動作，也就是 A 預測出來的類別跟 B 預測出來的類別一起合併，而出來的分數的確是會上升，可以推測出 ensemble 方法還是有用的：

| | w/o ensemble | w/ ensemble |
|--------------|--------------|--------------|
| kaggle pubic | 0.440 | 0.444 |

7. Conclusion

在這個比賽當中為了解決 Class Imabalance 和提高我們的排名，我們嘗試了使用很多種不同的方法，以下為我們嘗試方法的統整：

- 1) 調整 Data Augmentor 的參數
- 2) 運用不同種模型去做訓練
- 3) 解決 Class Imbalance 的問題
- 4) 調整 threshold
- 5) 使用 ensemble 的方法

我們一開始使用 pretrained 好的模型，發覺出來的分數都是 0.3 多，一直沒辦法過 strong baseline，後來才去 kernel 發現這筆 database 有很嚴重的 Data Imbalance 的問題，於是上網 survey 如何去處理 Data Imbalance 的方法，最後才終於過 strong。然而為了拚 ranking，我們原本只使用 ResNet18，最後才去嘗試 InceptionV2，並且經過不斷地調整 threshold，還有一些參數才上升至 0.46。而以下也統整出幾個我們覺得對於這個比賽有效提升分數的小技巧：

- 1) 調整每個類別的資料權重
- 2) 去訓練每個類別的 threshold
- 3) 直接使用 output file 達到 ensemble 的效果

因此在這比賽我覺得最重要的是，你如何去處理好 Data Imbalance 的問題，如果處理好基本上過 strong 不難

8. Reference

- [1]<https://www.kaggle.com/c/human-protein-atlas-image-classification>
- [2]<https://arxiv.org/abs/1608.06993>
- [3]<https://arxiv.org/abs/1512.03385>
- [4]<https://blog.csdn.net/sunqiande88/article/details/80100891>
- [5]<https://arxiv.org/abs/1502.03167>
- [6]<https://blog.csdn.net/yuanchheneducn/article/details/53045551>
- [7]<https://www.kaggle.com/allunia/protein-atlas-exploration-and-baseline>