1. Introduction (20%)

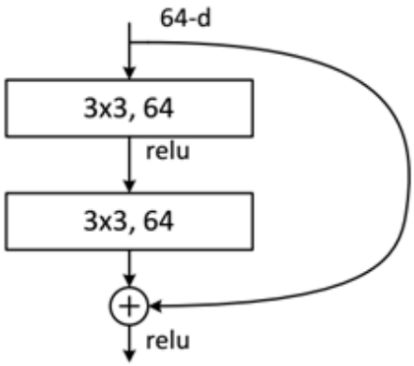
這次作業要求實做出ResNet18 以及 ResNet50兩種神經網路架構，並且在這兩個神經網路上透過助教所提供的model.py來實做有pretrain過的model，以及直接實做沒有pretrain過的mode，一共會需要實做出四種models，來比較誰的accuracy較好。這次作業的另一個重點是要透過助教提供的4個csv檔（train_data,label & test_data,label）以及雲端上的data.zip檔，將data folder裡面所有的3*512*512的RGB視網模圖片根據training or test的編號來load到程式中。

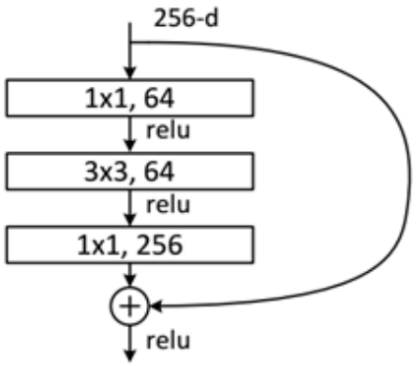
2. Experiment setups (30%)

A. The details of your model (ResNet)

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

參考不同ResNet的各個架構圖，會發現ResNet18需要的基本元件是BasicBlock，而ResNet50需要的基本元件Bottleneck 。形狀如下:





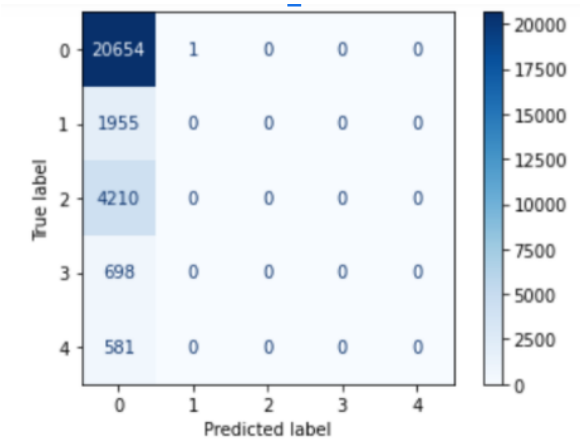
因此在設計時需要先分別對BasicBlock及Bottleneck做設計，可以理解成ResNet18 = ResNet+BasicBlock，ResNet50=ResNet+Bottleneck，而後設計ResNet的架構，由於5個conv layer構成的，第一層為預設，其他四層則有各自的設定，所以主要區分不同的ResNet是依據這四層，因此會設計make_layer讓我們能根據block來設計不同的ResNet，舉例來說：
ResNet18 = ResNet(BasicBlock,[2,2,2,2]),ResNet50=ResNet(Bottleneck, [3,4,6,3])
其中block代表的是構成不同conv層的單元數目，其他大小的ResNet亦可參考此設計方式。
且在最後一層自己設定fully connected的大小。

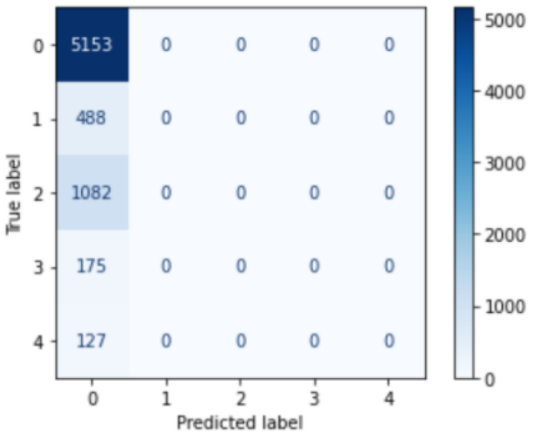
B. The details of your Dataloader

這次的Dataloader主要實作的部分在__getitem__，他是一個magic function，使用方法是當我們對有__getitem__物件取index時(可以是多個維度，看回傳值有幾維決定)，會自動執行__getitem__函示的內容並回傳抓到的item()，實作細節為:根據getData()得到path，透過path抓到圖片後再把label和img轉成我們要tensor form，以利我們訓練神經網路。
#在dataloader的時候應該做圖片旋轉來讓圖片多樣性增加，使結果更好。

C. Describing your evaluation through the confusion matrix

這邊的confusion matrix並不是最好的，因為我訓練的結果有些問題，但由於時間的緣故先將現有的狀況繳交上來，我會再嘗試其他方法。
Confusion matrix的每個(row,col)代表值的含意是:真值為row被我預測成col 的數目，左圖是train的，右圖是test的，可以看到training的confusion幾乎把所有的人都pred成0了，所以test也學習training把所有人幾乎都pred成0，且可以明顯看出資料存在不均等的問題，從圖上來看會發現大多數的人都是class 0的。





3.Experimental results (30%)

A.The highest testing accuracy

```
    path = "CKPT/pre_ResNet18_final_ckpt"
    print(validation(pre_net,test_data,len(test_data),path))
    path = "CKPT/new_ResNet18_final_ckpt"
    print(validation(net,test_data,len(test_data),path))
```

```
0.7335231316725979
0.7335231316725979
```

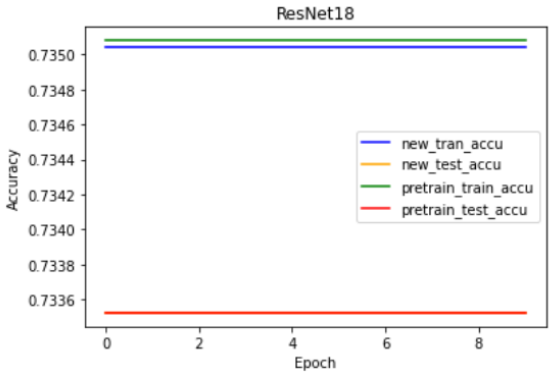
```
else:
    path = "CKPT/pre_ResNet50_final_ckpt"
    print(validation(pre_net,test_data,len(test_data),path))
    path = "CKPT/new_ResNet50_final_ckpt"
    print(validation(net,test_data,len(test_data),path))
```

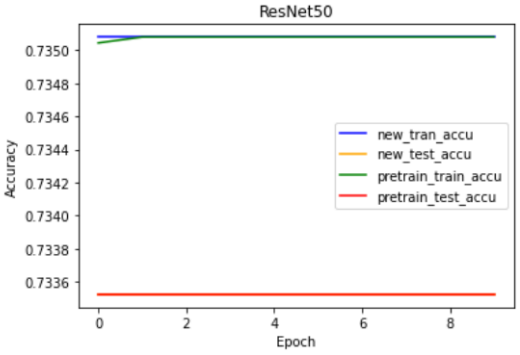
```
0.7335231316725979
0.7335231316725979
```

最高的狀況不太理想，四個model皆卡在73%左右，有請教做到82%以上的同學，主要的方法是做圖片旋轉以增加圖片的多樣性，demo前我會嘗試這個方法以增加accuracy。

B.Comparison figures

我在訓練時遇到了一些問題，在做pretrain跟沒有pretrain的四個model都會像下圖一樣，卡在只有第一個epoch練起來後後面的結果都沒變的狀況，因此會發現圖大多幾乎是一直線，且new_test_accu 訓練的結果和pretrain_test_accu一樣，我認為沒有pretrain的model沒有學習率是正常的，但pretrain的model練不起來的原因我想是因為我在實作時沒有加入圖片旋轉的緣故，導致pretrain的ResNet的學習率很差。






```
pretrain network:
Train: epoch 1
TrainAccuracy is : 0.73507954019716
TestAccuracy is : 0.7335231316725979
執行時間：2092.848972 秒
Train: epoch 2
TrainAccuracy is : 0.73507954019716
TestAccuracy is : 0.7335231316725979
執行時間：2106.176200 秒
Train: epoch 3
TrainAccuracy is : 0.73507954019716
TestAccuracy is : 0.7335231316725979
執行時間：2118.089262 秒
Train: epoch 4
TrainAccuracy is : 0.73507954019716
TestAccuracy is : 0.7335231316725979
執行時間：2085.569059 秒
```

```
Train: epoch 5
TrainAccuracy is : 0.73507954019716
TestAccuracy is : 0.7335231316725979
執行時間：2087.780766 秒
Train: epoch 6
TrainAccuracy is : 0.73507954019716
TestAccuracy is : 0.7335231316725979
執行時間：2087.042780 秒
Train: epoch 7
TrainAccuracy is : 0.73507954019716
TestAccuracy is : 0.7335231316725979
執行時間：2088.393601 秒
Train: epoch 8
TrainAccuracy is : 0.73507954019716
TestAccuracy is : 0.7335231316725979
執行時間：2087.531054 秒
Train: epoch 9
TrainAccuracy is : 0.73507954019716
TestAccuracy is : 0.7335231316725979
執行時間：2087.382022 秒
Train: epoch 10
TrainAccuracy is : 0.73507954019716
TestAccuracy is : 0.7335231316725979
執行時間：2081.760543 秒
```

4.Discussion (20%)

若結果正常，會發現ResNet沒有pretrain的在10個epoch下幾乎沒有學習率，反而是有pretrain過的model在這樣短短的epochs還能夠有較顯著的學習，這是因為ResNet通常需要大量的資料及長時間的訓練才會有成果的緣故，用pretrain能改善這個問題。

和成功做出82%以上的同學打聽到，除了透過圖片旋轉之外以增加圖片多樣性外，有同學一開始先用lr=0.001的紀錄最好的CKPT，再透過lr=0.0001的去訓練這個CKPT，這樣的結果能夠使accuracy更高，這某種程度上算是lr decay的作法，但是由於這次作業需要訓練的時間較長這樣做能夠"人工的"來篩選有前途的CKPT來做lr decay，以大幅減少訓練時間。