Introduction

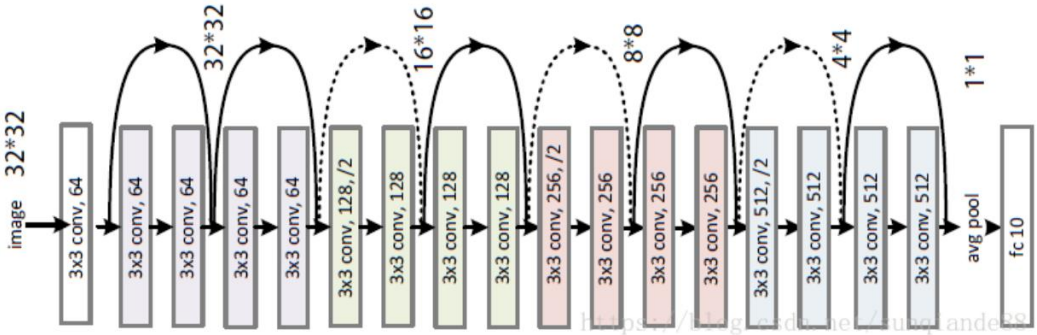
此次lab利用pytorch實作ResNet18以及ResNet50兩種ResNet，且需自行定義自己的dataloader，最後輸出兩種net的pretrained和非pretrained的train and test accuracy rate。這邊data的transform只做最基本的，將所有pixel value調整至0-1之間，並將其shape從[H, W, C] 改成 [C, H, W]。在實作ResNet時，因為各大layer是由多個小layer所組成，這裡利用class的概念將相似的小Layer包成一個物件，到時執行forward時比較簡潔且好控制，也可以很容易的去判斷是否需要downsample。從Input layer後的第二個大隱藏layer開始，每個大隱藏layer的第一小層(resnet18為例，一小層有兩個Conv2d)都需要作downsample，調整輸出入大小以符合下一小層的輸出入。ResNet藉由將上幾層layer的output加到後幾層layer的input的方法，來解決gradient vanishing and exploding的問題。

Experiment setups

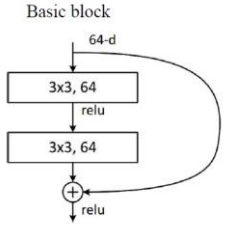
A. The detail of model

1. ResNet18

資料來源: https://blog.csdn.net/sunqiande88/article/details/80100891



虛線部分為downsampling，出現在每層大layer中的第一小層，此時stride=2，image的size為512*512，最後一層Linear輸出時的size為5。實作時，以每兩層conv2d做為一小層，再利用這些小層堆疊出整個Net。



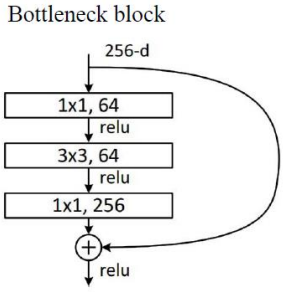
每一小層的架構，chanel大小會不同

2. ResNet50

資料來源: https://reurl.cc/b510Av

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	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	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

和ResNet18的寫法大致相同，只是每一小層涵蓋了三層conv2d，kernel_size即in_chanel和out_chanel有所改動，同樣是每大層的第一小層需要做downsampling(第一層除外)。



每一小層的架構，chanel大小會不同

兩種net所需要注意的主要是in_chanel和out_chanel的大小，以及stride的值。

B. The details of dataloader

和原先提供架構基本一致，再getitem的地方有做改動而已

```
##step1
path = self.root + self.img_name[index] + '.jpeg'
img = Image.open(path)

##step2
GroundTruth = self.label[index]

##step3
img_np = np.asarray(img)/255
img_np = np.transpose(img_np, (2,0,1))
img_ten = torch.from_numpy(img_np)

##step4
return img_ten, GroundTruth
```

第一步求出圖片的檔案名稱，並load到img裡

第二步得到該檔案名的groundtruth

第三步將img的值控制在0-1之間，並改變其shape，以tensor型式送出

C. Describe the evaluation through the confusion matrix

Confusion matrix可以得到在GroundTruth底下，predicted其他結果及正確結果的機率各是多少，以groundtruth為0，predicted為1的例子來說，算法是將所有groundtruth為0的所有資料數量放在分母，將groundtruth為0，prediction為1的資料數量放在分子，出來的數即為該格的結果，如此一來，每列總和都會是1，也可以從中看出groundtruth和prediction的關係。

因為不小心沒存到整個model，時間關係來不及產生confusion matrix。

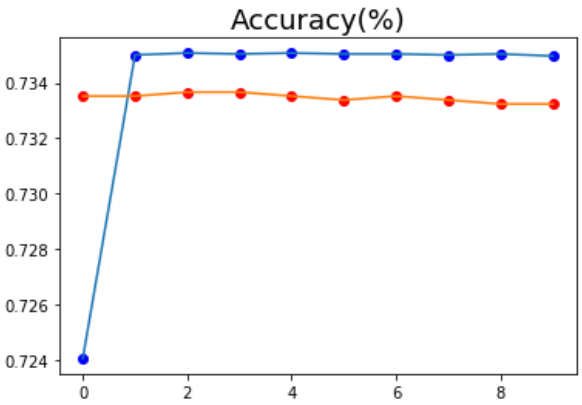
Experimental results

A. The highest testing accuracy and anything want to present

因為不小心沒存到整個model，時間來不及重chain，所以有所缺漏。

ResNet18 without pretrained:

```
epochs: 9
Train Accuracy:  0.7349727748318445
Test Accuracy:  0.733238434163701
Max accuracy:  0.7336654804270463
```



藍色為training accuracy, 紅色為testing accuracy

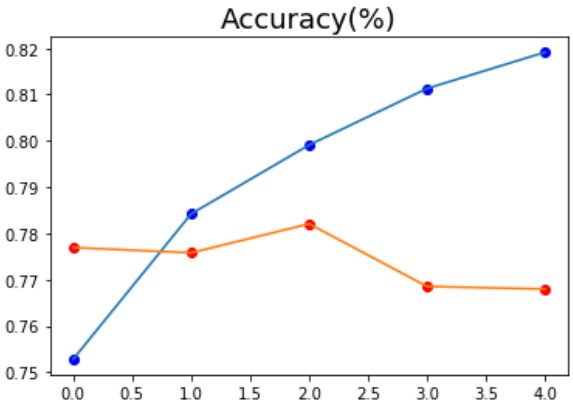
ResNet18 with pretrained:

```
epochs: 9
Train Accuracy:  0.8788568988220221
Test Accuracy:  0.741067615658363
Max accuracy:  0.7830604982206406
```

ResNet50 without pretrained:

ResNet50 with pretrained:

```
epochs: 4
Train Accuracy:  0.8191394711555572
Test Accuracy:  0.7679715302491104
Max accuracy:  0.7820640569395018
```



藍色為training accuracy, 紅色為testing accuracy

B. Comparison figures

	With pretrained	Without pretrained
ResNet18	0.7831	0.7336
ResNet50	0.7820	

Discussion

這次lab因為layer比較多，執行時間上大幅超過lab2，在lab2時我依舊使用cpu下去執行，勉強可以得出一些結果，但這次已無法單純使用cpu執行，效能太慢，所以必須得使用cuda來執行，但須先將其環境建立好，在這方面我花了不少的時間，也了解了可以藉由將data丟上網路上，再由遠端下載下來執行的方式來使用cuda跑程式，效能上真的差非常多。

Pretrained model為原本就定義好的resnet model，內部架構都已決定好，差別只在最後一層輸出層的定義，因為本次lab的output chanel為5，所以最後額外定義一Linear model去覆蓋原先的，將其output chanel強制改為5。

Momentum有類似物理慣性定律的概念，當這次得出的gradient更新方向和前次方向一致時，會加快weight的更新，反之，減少更新的幅度。