**Lab3: Leukemia classification**

Student id / name: A113599 / 楊淨富

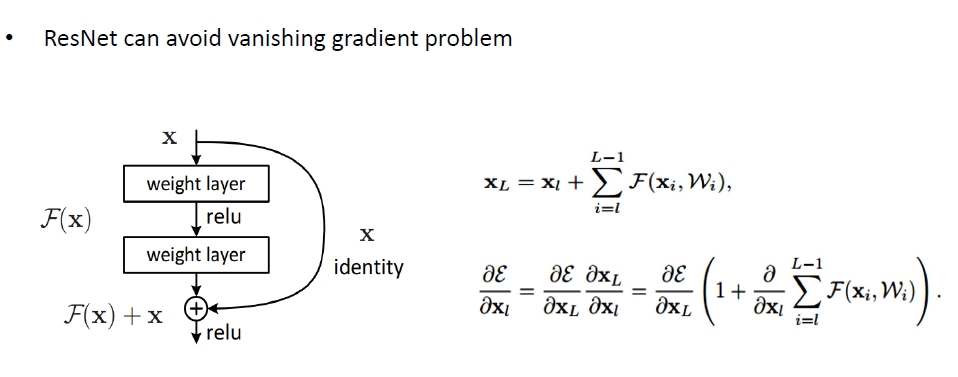
**I. Introduction:**

訓練3個model，分別為ResNet18, ResNet50, ResNet152來判斷給定的圖片是否為急性淋巴性白血病。此外，該lab禁止使用pre-trained model，即必須自行train出參數，並且需上傳對於test data(unlabeled)的預測結果至Kaggle上進行競賽。

**II. Implementation Details:**

1. **The detail of your model**

* ResNet



在標準的神經網路中，每一層對輸入數據進行轉換，並將一層的輸出作為下一層的輸入。在反向傳播過程中，計算並沿著這些層傳播梯度。然而，在非常深的網路中，隨著梯度通過每一層，它們可能呈指數級遞減，使得初始層難以學習有意義的表示。這個問題稱為「梯度消失」，在具有許多層的網路中特別突出， 這會使得神經網路難以有效地更新權重，導致訓練緩慢或停滯。

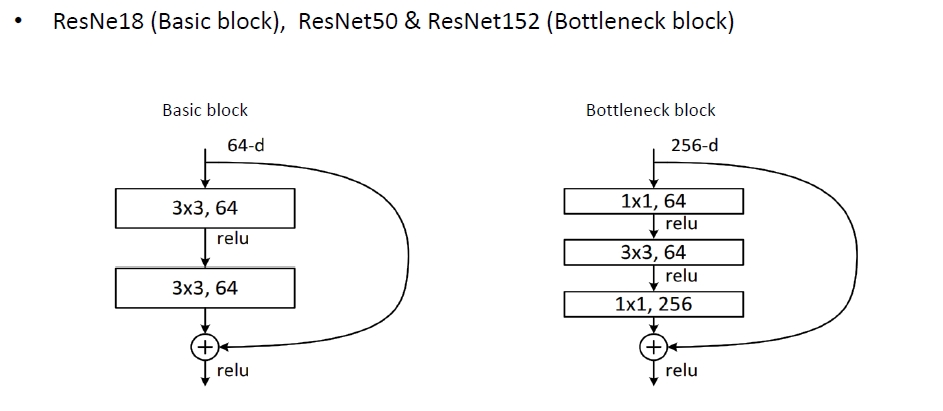
ResNet是透過residual connections(or skip connection)來避免梯度消失的問題。它並非在一層中學習從輸入到輸出的直接映射(identity mapping)，ResNet學習殘差映射(residual mapping)—即輸入和輸出之間的差異。這個過程可以表示為：

輸出 = 輸入 + F(輸入)

這種residual mapping使得gradient在back propagation更容易流動。此外，如果F(x) = x(恆等函數)，則gradient可以直接通過skip connection流動，輸入的梯度，即∂輸出/∂輸入是固定的，不受層數增加而改變，輸出的梯度不會消失（變得過小）或爆炸（變得過大），而是保持恆定值，有效避免了梯度消失。

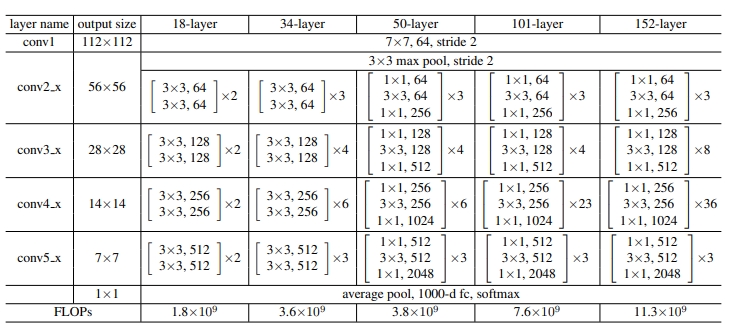
若F(x)非恆等函數，梯度仍然可以通過跳躍連接流動，有助於學習過程。這些跳躍連接的存在確保梯度在網路中具有更短的傳播路徑，在訓練過程中有助於保持資訊的良好流動，減輕了梯度消失問題，並成功地實現了非常深的網路的訓練。

因此，ResNet架構可以更深（例如ResNet-101、ResNet-152等）而不會嚴重降低性能，並且已被廣泛應用於各種電腦視覺任務，例如：圖像分類、目標檢測等任務中。



* Basic block應用在ResNet18, ResNet34
  + 由兩層CNN為主組成，kernel\_size = (3, 3)，並且在第一個CNN搭配ReLU，並且會將residual和最後的結果相加並再次經過ReLU。每次ReLU前都可以先batch normalization。
* Bottleneck block應用在ResNet50, ResNet101, ResNet152
  + 由三層CNN，kernel\_size 分別為(1, 1), (3, 3)和(1, 1)為主所組成，主要目的是：

1. 降低維度: 使用1x1卷積層可以減少input的channel。因為1x1卷積在深度(通道)維度上進行計算，相當於通道間的線性組合。
2. 減少計算量: 相比於直接使用3x3卷積，Bottleneck block中的1x1卷積可以大幅度地減少計算量。1x1卷積在通道數上的計算成本要遠小於3x3卷積，因為它涉及的參數數量更少。
3. 增加非線性：通過在1x1卷積和3x3卷積之間添加ReLU，Bottleneck block可以增加非線性，有助於網路學習更複雜的特徵表示。



接下來解釋ResNet18, ResNet50, ResNet152的配置: (參考原paper，如上圖)

* ResNet18: 5層layer
  + 第一層(conv1): kernel\_size=(7, 7) stride=2，之後可再搭配batch normalization和ReLU。
  + 第二層(con2\_x): 先透過max pooling，再使用兩個Residual block。
  + 第三至第五層(conv3\_x, conv4\_x, conv5\_x): 和第二層最大的差別為，basic block的stride改為2，為了減少資料量，且為了使輸入和輸出的尺寸一致，所以需要加上down sample的動作。此外，不使用max pooling。

此外，ResNet18有一些Hyper parameters，以下是我tune的參數:

* Batch size = 64
* Learning rate = 0.0001
* Epochs = 201
* Optimizer: torch.optim.SGD()
* Loss function: torch.nn.CrossEntropyLoss()
* ResNet50/ResNet152: 5層layer
  + 第一層(conv1): 和ResNet18相同。
  + 第二層(conv2\_x): 先透過max pooling，再使用了三個bottleneck block。
  + 第三至第五層(conv3\_x, conv4\_x, conv5\_x): 差別在於使用的bottleneck數，以及第二個CNN所使用的stride由1改為2。此外，不使用max pooling。
  + ResNet50和ResNet152僅差在第二至第四層的bottleneck數，ResNet50為[3, 4, 6, 3]，ResNet152為[3, 8, 36, 3]。

ResNet50的Hyper parameters:

* Batch size = 32
* Learning rate = 0.0001
* Epochs = 201
* Optimizer: torch.optim.SGD()
* Loss function: torch.nn.CrossEntropyLoss()

ResNet152的Hyper parameters:

* Batch size = 32
* Learning rate = 0.0001
* Epochs = 31
* Optimizer: torch.optim.SGD()
* Loss function: torch.nn.CrossEntropyLoss()

1. **The details of your Dataloader**



* \_\_init\_\_(self, root, mode): 初始化，root(string)為dataset所在的路徑，mode則根據對應的mode(train/ valid/ test)來取得csv檔中的image path跟label。
* \_\_len\_\_(self): 回傳dataset的size
* \_\_getitem()\_\_(self, index): 根據index去取得image的path，並import PIL並使用它的 Image.open()來開啟圖片，再使用 transforms.Compose() 這個function來將圖片轉換成tensor，轉換時可以搭配transforms.CenterCrop(),transforms.Resize(),transforms.RandomRotation() transforms.RandomHorizontalFlip(),等等function來對圖片進行處理。最後再根據mode傳回img(以及label)的tensor。

1. **Describing your evaluation through the confusion matrix**

混淆矩陣（Confusion Matrix）是在機器學習和統計學中用來評估分類模型的性能的一種方法。它是一個矩陣，用於展示模型在測試集上對不同類別的預測結果。

混淆矩陣的行代表實際的類別，列代表預測的類別。對於一個二分類問題，混淆矩陣如下

|  |  |  |
| --- | --- | --- |
|  | 實際正例 | 實際反例 |
| 預測正例 | TP | FP |
| 預測反例 | FN | TN |

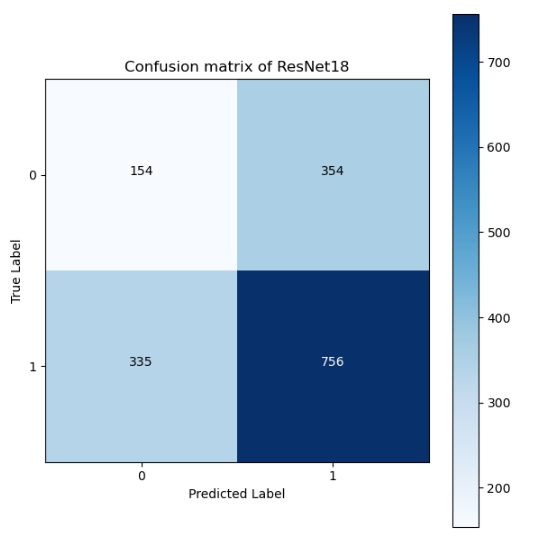
* TP (True Positive)：真正例，表示實際為正例且被正確預測為正例的數量。
* TN (True Negative)：真反例，表示實際為反例且被正確預測為反例的數量。
* FP (False Positive)：假正例，表示實際為反例但被錯誤預測為正例的數量。
* FN (False Negative)：假反例，表示實際為正例但被錯誤預測為反例的數量。

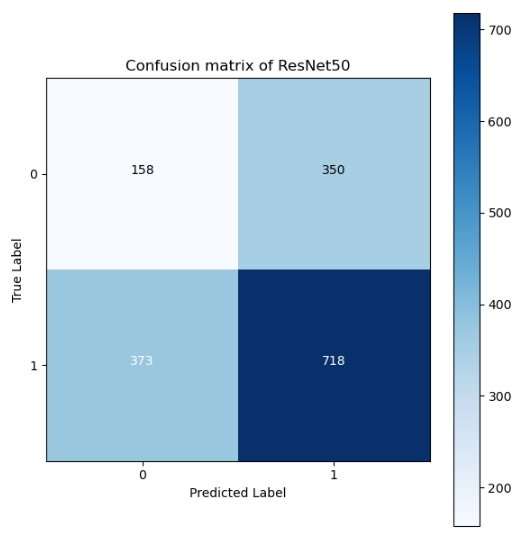
混淆矩陣也可以用於多分類問題，其中類別數目超過兩個。在這種情況下，混淆矩陣的大小會隨著類別數目的增加而增加，但基本原理保持不變。

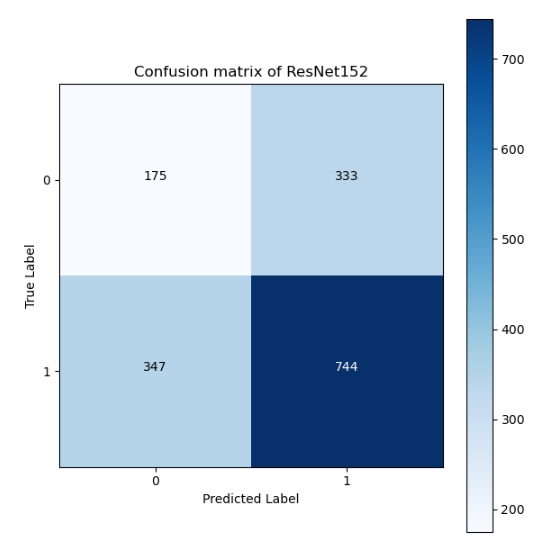
通過分析混淆矩陣，我們可以獲得模型在不同類別上的表現情況，找出模型在哪些類別上出現了錯誤，並進一步優化模型的設計。混淆矩陣是評估和理解分類模型性能的重要工具之一。



* Normalize預設為False，即顯示對應的分類圖片數量，所以區域數字總和為圖片總數量。
* Normalize若設為True則改為顯示分類圖片的比例，所以每一列(row)總和應為1(100%)。且對角線為預測相應結果正確的對應比例。
* 以下三張圖分別為ResNet18, ResNet50, ResNet152的confusion matrix。資料集為valid set的1599張圖片。
* 三個模型train出來後的confusion matrix結果相似。在FP/FN的數量差不多。也可以發現我在預測valid set中非急性淋巴性白血病的圖片有較高的準確率。







**III. Data Preprocessing**

**A. How you preprocessed your data?**

* 利用transform.CenterCrop()來擷取圖片中心，可以加快運算速度。
* 為了讓訓練時可以看過更多圖片類型，增加訓練的彈性，即data augmentation，加入了transform.RandomHorizontalFlip()跟transform.RandomRotation()來讓圖片有隨機水平翻轉和旋轉一個小角度的設定。
* 加入transform.Normalize()來使得圖片的資訊更均勻。

**B. What makes your method special?**

* 根據A的描述，由於使用CenterCrop，可以使圖片大小變小，但又不至於失去太多圖片的特徵，因為圖片主要是集中在中央，可以大幅提升運算速度。
* 可以使用resize來進一步減少圖片大小，也可以加快運算速度，但測試後發現會讓結果變差太多，所以最後沒有使用。
* 另外有關random的函數，可以在training時，看過更多種圖片，增加model彈性，豐富data set，有助於model generalization。
* transforms.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))這組數字是從ImageNet訓練資料集中抽樣算出來的，經過正規化後，可以加快模型的收斂速度。

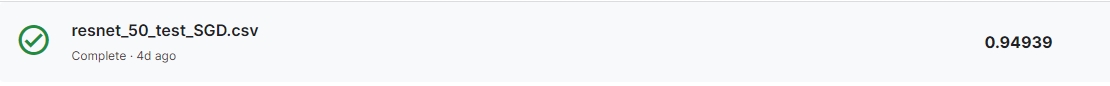
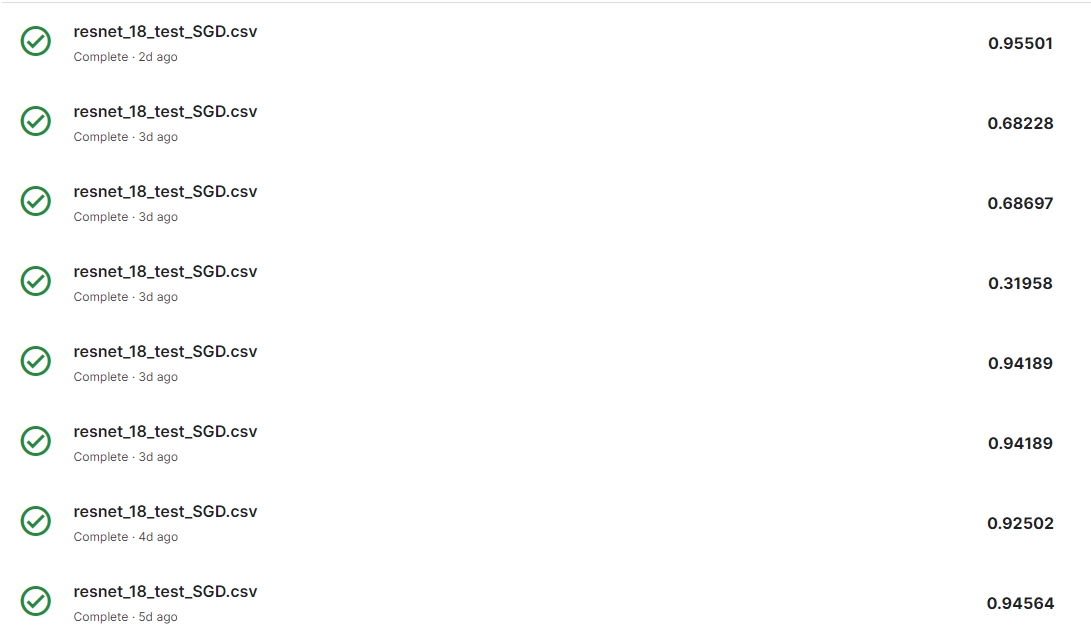
**IV. Experimental results**

**A. The highest testing accuracy**

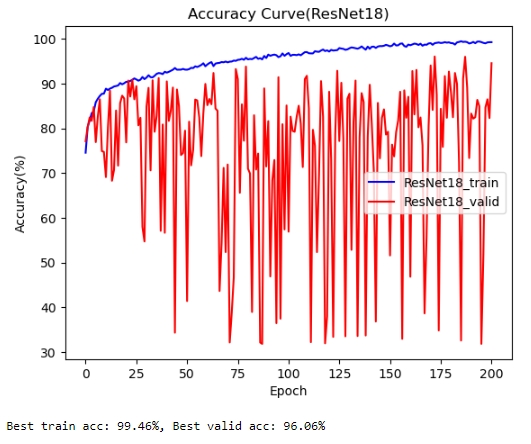
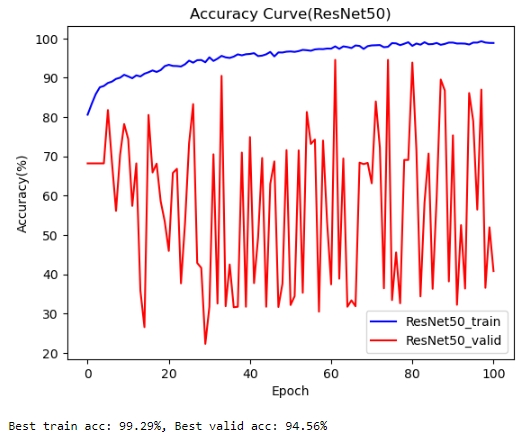
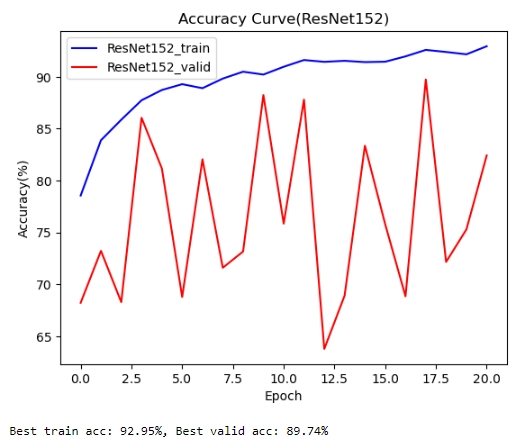
* Screenshot

|  |  |  |  |
| --- | --- | --- | --- |
|  | ResNet18 | ResNet50 | ResNet152 |
| Train accuracy | 99.46% | 99.29% | 92.95% |
| Valid accuracy | 96.06% | 94.56% | 89.74% |
| Test accuracy | 95.50% | 94.93% | 92.87% |

* Anything you want to present

****

**B. Comparison figures**

* Plotting the comparison figures(RseNet18, ResNet50, ResNet152) 
* 
* 

**V. Discussion**

**A. Anything you want to share**

* 一開始我在train ResNet18時，發現epoch=201的結果比epoch=101上傳到Kaggle的結果好，所以我覺得可以硬train一發epoch數非常大的training。而且畫出accuracy的圖片進行觀察時，accuracy上下震盪的幅度很大，突然就有可能找到一組很不錯的參數。但我一開始並沒有把training的code寫好(即每次都是重新train，而不是從上次train好的model繼續train)，加上前面花了很多時間不斷嘗試圖片裁切的參數，導致花了很多時間train出一堆垃圾。
* ResNet152的model state(.pth)非常大，一開始沒寫進gitignore，又不小心push到github，然後就出錯，要用LFS傳，但最好還是model都不要push。
* Kaggle比賽很好玩，前面排名的人很厲害，很想知道他們是用什麼方法做出來的，希望以後課程，助教可以請這些大神分享是如何利用自己獨門的技巧增加accuracy。
* 一開始train ResNet152時，發現算超級久，然後又沒寫存一個check point的code，導致我有時候interrupt kernel就沒辦法從上次的地方繼續train，超浪費時間。也是很後面才意識到這個問題，變成最後只train了21個epoch，成績也不是很好看，但後面寫好這部分的code時已經很接近deadline加上又有三個model要train，結論就是，以後training應該是最後面的事，不要急著train，而是應該把code架構搞好(即存model或是可以往下繼續train)。
* 一開始看不太懂Basic(Residual) block跟Bottleneck block的內部結構，導致一些channel數一直跳error(即matrix size不對)，花了一些時間進行推導。
* 我有換Adam來train，但結果似乎跟SGD差不多，所以後面就都用SGD了。