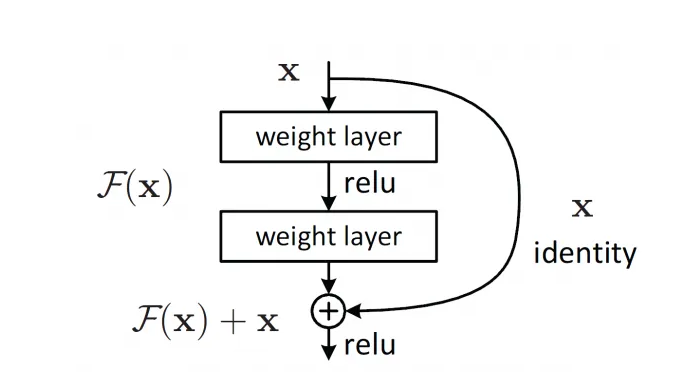
DeepLearning-HW2

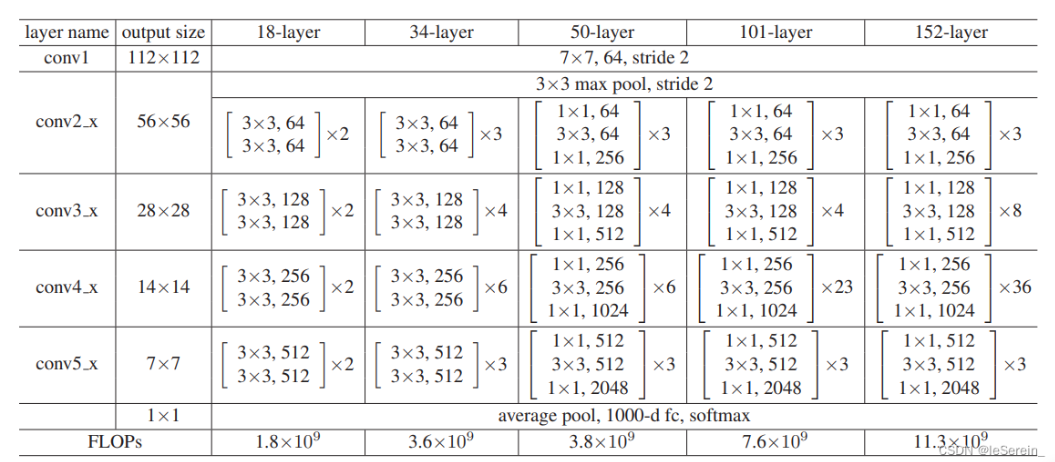
學號:313834004 姓名:周彥宏

實作ResNet model

這次在實作ResNet model, 選擇使用layer-18的架構下去做. ResNet主要架構其實就是CNN的擴展, 會多出一個叫做Residualblock的Class來去作出所謂的誤差計算, 如下圖所示: 

主要是將計算出的x套入公式, F(x)+x 是從F(x) = H(x) – x 所推導出來, 主要是用來解決CNN的一個問題, 當無法一次算出最接近的答案, 那可以通過誤差值來一階段一階段的找出最相近的答案, 得出H(x) – x = 0 使其等價。

**ResNet不同層數的架構:**



這次在設計的是ResNet-18的架構, 會有四個block\_layer 並且每層的Residualblock數量為2, 如果要架設18以上的就需要做修改. 值得注意的是當使用到50層以上後, 所使用的就不會是Residualblock 而會變成是Bottleneck Block,會有三層conv. 但由於太大架構的網路怕跑不動, 因此這次實作只使用18層的ResNet來實作。

而在input size上, 為了符合架構圖上所提供的size, 所以在資料前處理上一律將image resize(224,224). 關於input size經過conv, maxpool後, 會產生的outputsize我有將計算過程寫成程式方便我在使用不同網路時也可以很好計算.([計算程式](https://github.com/Qmooo/NYCU-DeepLearning/blob/main/Warmup/calculate_output_size.py))

資料前處理

**訓練集:**

下圖是我所做的資料前處理還有Data augmentation得部分, 我使用了隨機垂直翻轉, 機率則是用預設的0.5. **主要原因為:** 我觀察了花的資料後, 覺得做水平翻轉對於花這種對稱形狀並沒有很大的意義, 因此我選擇了垂直翻轉, 讓一些花可能有上下分明的特稱可以有更多的資料.

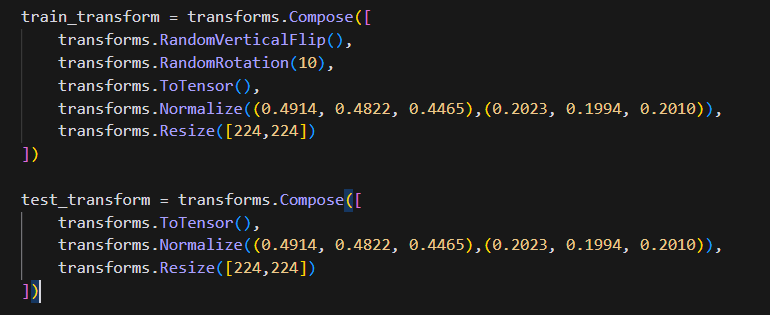
接著我使用了隨機選轉, 角度區間設為[-10,10], 主要是想讓花透過不同角度會有機會去呈現不一樣的特徵點.

最後就是做歸一化和resize, 歸一化的部分, 我不使用範例的0.5, 而是上網參照一些不一樣的人所做的實驗設置, 透過輸出來觀察花的圖片, 最後發現當我使用這主數據時, 最能夠將花的本體顏色凸顯出來, 並且可以有效的將背景色變為接近統一色調.Resize則是上面提到為了符合網路架構的input size所設定.

**驗證集和測試集:**

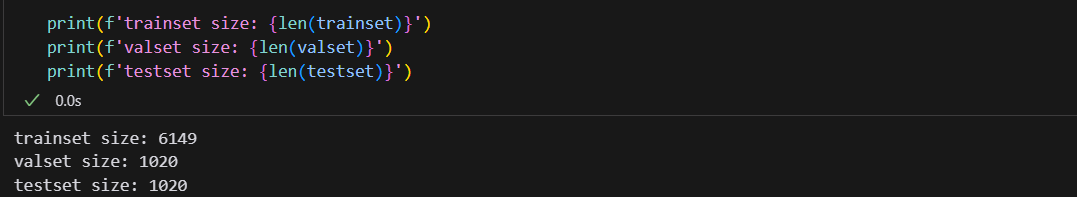
在這部分我只有做了簡單的歸一化和Resize, 相關原因和訓練集相同.

**嘗試步驟:**

嘗試過將圖片做灰階來判斷, 原先想法是灰階圖, 就不會注重花的顏色, 而是會去學習花的本體特徵, 但最後效果顯示十分不佳. 發現其實**顏色也是一個非常重要的特徵點, 對於這份資料集來說.**

**資料集大小**

資料集切分是照著Flower102內建的去切分, 唯一要注意的是**我的訓練集**使用的是**Flower102的測試集**, 而**我的測試集**是使用**Flower102的訓練集**.

主要原因為資料量大小差異過大, 所以選擇使用6149筆資料的來做訓練. 

模型參數&訓練參數&優化器選擇&lr\_scheduler

|  |  |  |  |
| --- | --- | --- | --- |
| Activation function | ReLU | LeakReLU | Sigmod |
| Learning Rate | 0.001 | 0.001 | 0.001 |
| lr\_scheduler | MultiStepLR  (milestones=[40,70],gamma=0.2) | MultiStepLR  (milestones=[40,70],gamma=0.2) | MultiStepLR  (milestones=[40,70],gamma=0.2) |

表(一)

**模型參數:**

在模型參數部分, 都是依照ResNet論文的架構下去實作,並沒有額外新增或是減少Conv, Pool, or BatchNorm layer,只有在設定Activation function時, 選擇了表(一)

三個來做,最後會有實作的比較圖表.

會想要選擇主要的原因是, ReLU和LeakReLU是**Non-Saturated function**, 而Sigmod則是**Saturated function**. 而Non-Saturated function優點則是可以解決**梯度消失**的問題, 也可以加快收斂的速度, 因此選擇兩種不同的來做比較.

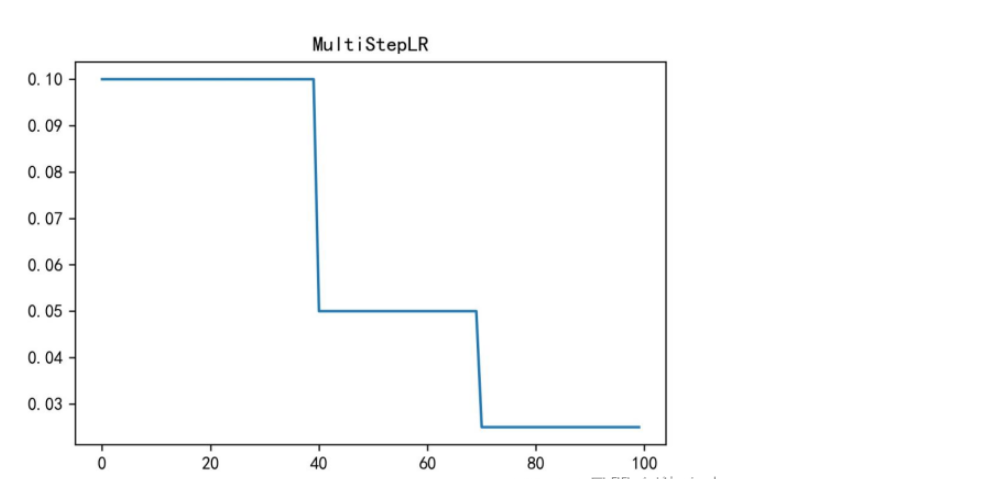
而另外會選擇LeakReLU是想要和ReLU做比較, 因為ReLU有個缺點是他會將所有x內的**負值都設為0**, 很容易造成神經元死掉. 而LeakReLU則是會將負值賦予一個非零的斜率, 我想要觀察這兩者會不會有很大的差異.

優化器選擇:

在優化器的部分, 我選擇Adam而不是SGD. SGD是用微分的方法算出梯度, 並往梯度方向去更新參數, 而Adam有保留SGD的方法, 可以使用過去的梯度方向做梯度的速度調整, 和利用過去梯度的平方值來去做learning rate的調整.

lr\_scheduler

接著我想要根據epoch的次數來去減少我的learning rate, 因此我使用了lr\_scheduler這個方法, 相關設定在表(一). 這次使用的是MultiStep, 遞減倍數設為0.2, Learning Rate分布如下圖一樣:

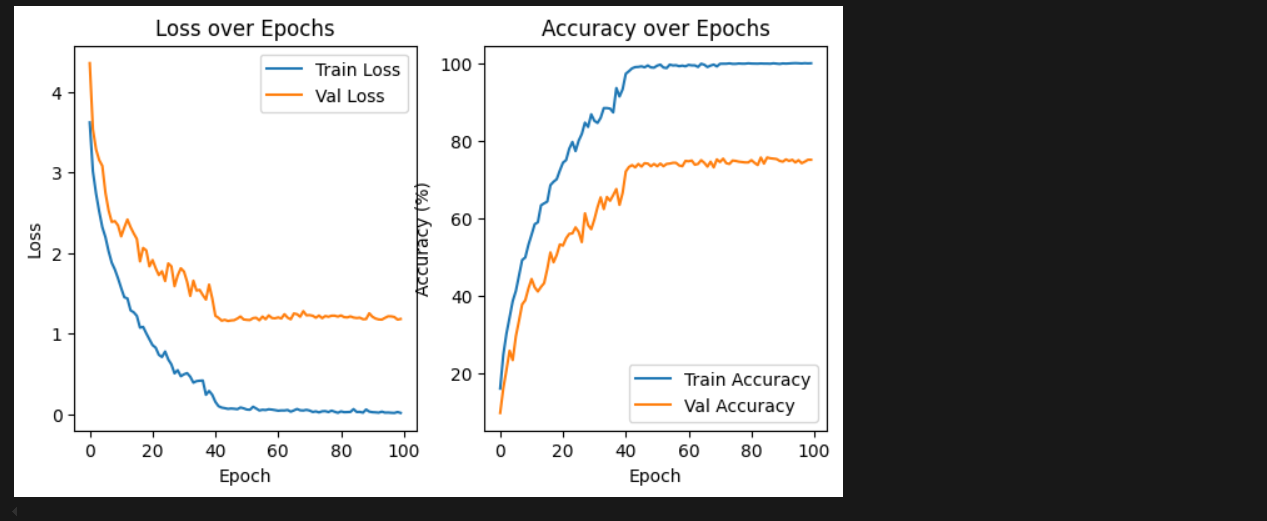


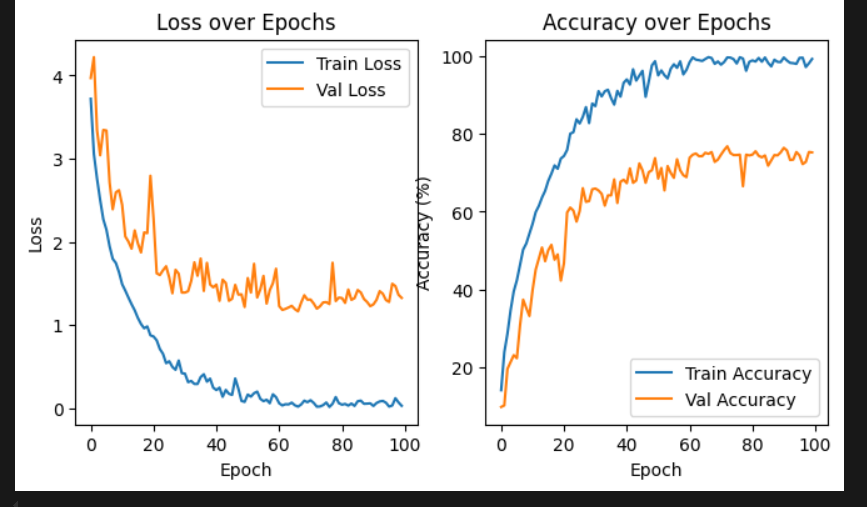
可以讓我的Learning Rate照著epoch次數來做調整.

實驗結果:

|  |  |  |  |
| --- | --- | --- | --- |
| Case/feature | Activation function | Epochs | Accuracy |
| Case1 | ReLU | 100 | 75% |
| Case2 | Sigmod | 100 | 46% |
| Case3 | LeakReLU | 100 | 74% |
| Example | ReLU | 100 | 36% |

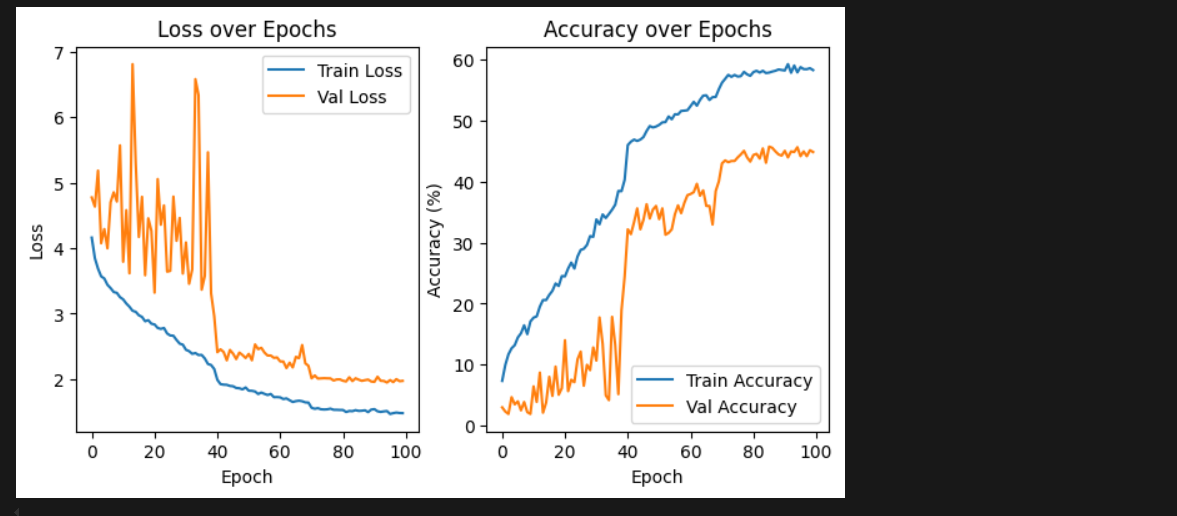
Case1:





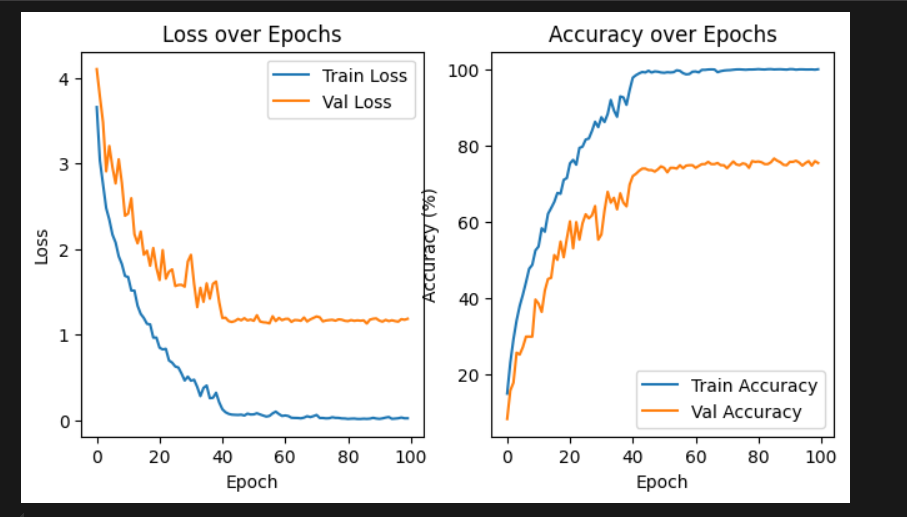
可以發現在訓練時, Loss在40個epochs之後有一個很明顯的下降並慢慢收斂, 是因為使用了lr\_scheduler的關係, 在40次會調降learning rate. 並最後收斂也是十分漂亮, 雖然準確率效果並不理想, 但不論是在驗證集或是訓練集的收斂上都是不錯的.另一張比較圖一樣是使用MultiStepLR(miles[60],gamma=0.5), 一樣Loss在60epochs會有不錯的下降, 但是最後收斂因為gamma設0.5所以沒有收斂得很好. 後續會將數值改成40,70也是透過這張比較圖來決定.

Case2:



可以發現case2在使用sigmod的效果十分不好, 在訓練中loss雖然收斂的很不錯, 但是在驗證集可以很明顯發現前期的收斂效果非常不好, Loss到最後也是十分高, 所以跟ReLU的比較可以說使用ReLU會來的好很多.

Case3:



Case3相對於Case1來說差距其實也沒有很大, 收斂效果也因為使用lr\_scheduler顯得很不錯, 但對於比較兩者差異來說, 也許是沒有特別大的梯度和learning rate所以沒有辦法發揮LeakReLU的優勢.

Example:

最後則是Example對於使用ResNet的比較, 範例所使用的是單純兩層Conv+3層全連接層的CNN架構, 可以很明顯發現Residualblock是有多麼的有效, 可以透過誤差計算來增加準確率, 效果很明顯好的非常多. 在實作完後, 造成這種情況的原因主要在優化器的選擇, 和誤差計算的使用, 這是十分聰明且有趣的一個想法.

