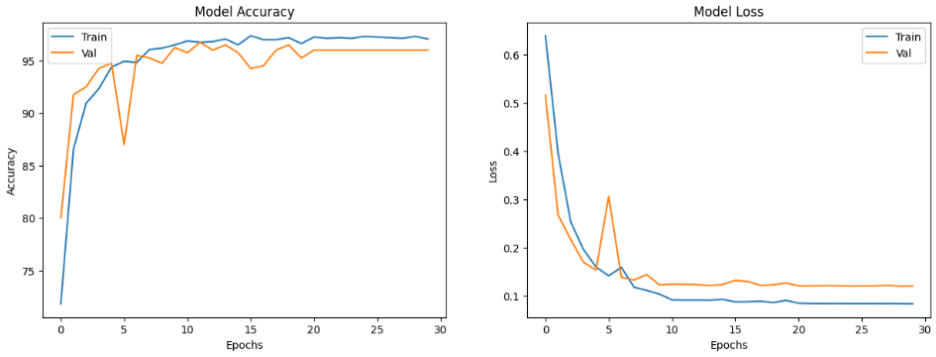
# National Tsing Hua University
# Fall 2023 11210IPT 553000
# Deep Learning in Biomedical Optical Imaging
# Homework 3

林伯諭

*Student ID: 111066538*

## 1. Task A: Reduce Overfitting

我們可以從 Train Loss 以及 Val Loss 的增減來看知道訓練模型是甚麼樣的狀態，當 Train Loss 下降但 Val Loss 不動或是上升那就代表 Overfitting，當 Overfitting 時有幾種方法可以改善他，以下列出三種方法 Regularization、減少層數、減少 Node 數，這也是我在改善 ConvModel 的三個方法，首先我將層數變成一層但是我發現還是 Overfitting，所以我影入了 BatchNormld 函數，雖然有改善但是還是 Overfitting，所以我將 Node 改為 16，經過測試，發現已經不會產生 Overfitting 了其結果如圖一:




圖一

並在表一紀錄了改變前以及改變後的差異:

表一

| | Final Train Loss | Final Train acc | Final Val Loss | Final Val acc | 最佳 Val Loss 位置以及數值 |
|---|---|---|---|---|---|
| 改動前 | 0.0094 | 99.81% | 0.0983 | 96.25% | Epoch 20/30(0.973) |
| 改動後 | 0.0839 | 97.06% | 0.1206 | 96% | Epoch 29/30(0.1202) |

可以從表一看到改動前的 Final Train acc 減掉 Final Val acc 等於 3.56%，並且可以看到 Val Loss 在 Epoch 20/30 之後不降反升，這就是典型的 Overfitting。同時也可以看到改動後的 Final Train acc 減掉 Final Val acc 等於 1.06%，這明顯比改動前好上不少，也可以看到 Val Loss 的最佳直是在 Epoch 29/30，這意思是他還有在學習，只不過有點小波動。

首先我將所有的數據 Regularization，並將其位置跟函數放在圖二:

```
def forward(self, x):
        BN=nn.BatchNorm1d(32)
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
```

圖二

發現到他沒有非常顯著的改善，所以我將 layer 改為一層，並且我想說 Overfitting 是因為過度跟隨了訓練集，因此我把 kernel_size 改成 5，那是因為想透過元素與周圍運算來淡化元素的獨立性，如圖三:

```
class ConvModel(nn.Module): #會OVER F I T T I N G要修
    def __init__(self):
        super().__init__()

        # 1 channel, and using 3x3 kernels for simplicity, 256*256
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5, stride=1, padding='same')

        self.pool1 = nn.MaxPool2d(kernel_size=8, stride=8) # 128*128
```

圖三

這時我發現 Val Loss 不再上升了但是他還沒有開始下降，所以我將輸出的矩陣改為 16 並且通過測試發現將 MaxPool 的 kernel_size 改成 6，雖然這會導致(256-6)/16+1=16.625，不會有很好的轉換，但他出來的成果是最好的，其修改的地方為圖四:

```
import torch.nn as nn
import torch.nn.functional as F

class ConvModel(nn.Module): #會OVER F I T T I N G要修
    def __init__(self):
        super().__init__()

        # 1 channel, and using 3x3 kernels for simplicity, 256*256
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5, stride=1, padding='same')

        self.pool1 = nn.MaxPool2d(kernel_size=6, stride=16) # 128*128
```

圖四

如果用測試集去測試此模型可以發現他的 Test accuracy 為 79.25%，這數值與原本 CNN 模型的 78.25%高了 1%。

## 2. Task B: Performance Comparison between CNN and ANN

表二為 CNN 以及 ANN 的比較表，裡便包含了幾項較重要的參數 Final Train acc、Final Val acc、Final Val Loss、最佳 Val Loss 位置以及數值以及 Test accuracy:

表二

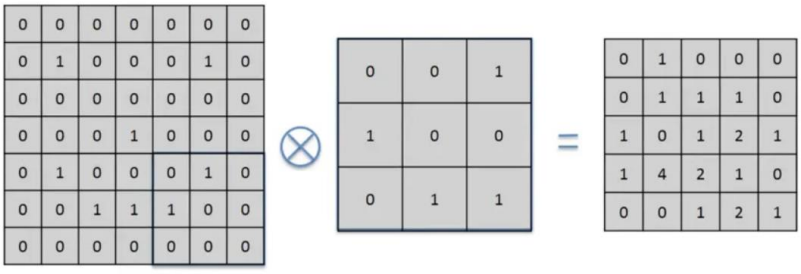
| | Final Train acc | Final Val acc | Final Val Loss | 最佳 Val Loss 位置 | Test accuracy | Time |
|---|---|---|---|---|---|---|
| CNN | 100% | 97% | 0.1323 | Epoch 13/30(0.1130) | 78.25% | 73s |
| ANN | 97.25% | 94% | 0.1811 | Epoch 20/30(0.1774) | 71.75% | 15s |

可以從表二看到 CNN 的優勢其實很明顯，就是他的 Test accuracy 比 ANN 高了 6.5%，並且也可以看到他的 Train acc 以及 Val acc 都高了次少。而 ANN 的優點也很明顯就是他的運行時間為 15 秒，為 CNN 的 73 秒的四分之一，而且雖然 ANN 與 CNN 都有 Overfitting 的情況產生，但是從最佳 Val Loss 位置中可以看到要改善 ANN 的 Overfitting 比起 CNN 簡單一點。

## ANN 與 CNN 結構比較

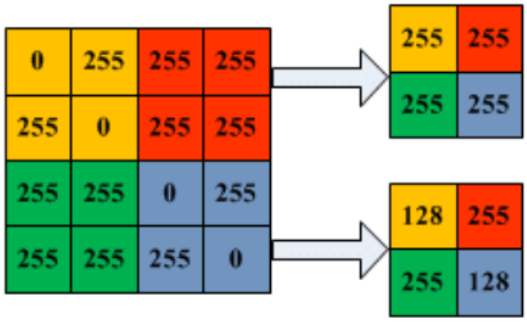
首先 ANN 主要的構成有三個部分，第一部分是 Input Layer，第二部分是 Hidden Layer，第三部分是 Output Layer，其中 Hidden Layer 的曾數並非只有一層，也可能包含數層。首先我們會將資料從 Input Layer 輸入，輸入節點會將不同的資料分類並傳至 Hidden Layer，當資料輸入至 Hidden Layer 之後資料會開是被分系並處理，其中假如 X 為 input features label、weights:W、training label:Y、bias:B、Activation function: $\sigma$ ，單一 Node 的輸入與輸出的關係為 $Y=\sigma(WX+B)$，如果只有一個 Layer 那這個值會直接被傳到 Output Layer，但如果為複數層那將會重複上述之動作 $Y2=\sigma(WY1+B)$。當數值被傳到 Output Layer 後 Output Layer 會將資料分類，如果為二元分類踏他將只有一個 Node，而多類分類的話那會有數個 Node。

比起 ANN，CNN 的架構分為四個部分，第一部分是 Input Layer，第二部分是 Convolution Layer，第三部分是 Pooling Layer、第四部份是 Output Layer，在這邊 Input Layer 與 Output Layer 的作用與上面一樣，所以我只著重於 Convolution Layer 以及 Pooling Layer。Convolution Layer 的作用是對 Input 進來的矩陣分塊進行處理如圖五:




圖五

從圖五中可以看到他將大矩陣中的某一塊分割出來後，與特定的矩陣做 Convolution 並輸出一個新的矩陣，那是為了將原矩陣的特徵提取出來。而 Pooling Layer 的功用是將 Convolution Layer 輸出矩陣的維度縮小，大多人是用以下兩種方法來使矩陣縮小的，第一種是 Max Pooling，就是將一塊區域內的矩陣取最大值並放到新矩陣的對應區域，而第二種是 Average Pooling，她的工作方法與 Max Pooling 類似但她是取一矩陣元素的平均值，會在圖六解釋其運作原理。

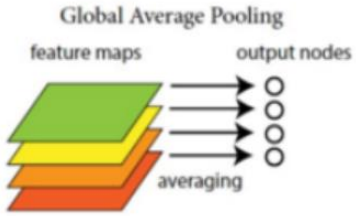



圖六

圖六是對原本的矩陣分別做 Max Pooling(上方)以及 Average Pooling(下方)，在這裡她的 kernel size=2，stride=2，可以知道輸出矩陣大小=(輸入大小-kernel size)/ stride+1=2。

### 3. Task C: Global Average Pooling in CNNs

因為 GAP 是將最後的 Pooling 層分層後輸出如圖七:



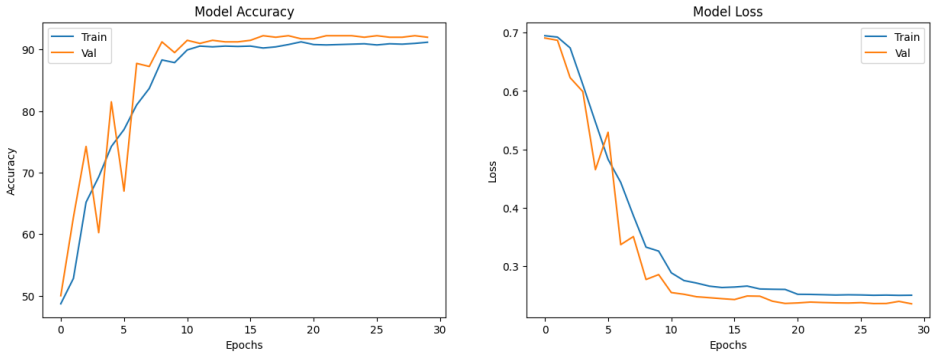

圖七

在圖七中可以看到 GAP 會將每層的 feature maps 輸出成一個直，例如一 9*9*3 的矩陣，做 GAP 後他的輸出會變為 1*1*3，因此不需要做尺寸計算。

從原本的模型可以看到有 Underfitting 的情形產生，要改善 Underfitting 最簡單的方法就是使用更複雜的模型，所以我將原本三層的 Hidden Layer 改為四層，而我將這兩種的輸出表格放在表三，並將其訓練結果記錄在圖八:

表三

| | Final Train acc | Final Val acc | Final Val Loss | 最佳 Val Loss 位置 | Test accuracy |
|---|---|---|---|---|---|
| 原架構 | 88.44% | 89% | 0.3018 | Epoch 29/30(0.3018) | 75.5% |
| 新架構 | 91.19% | 92% | 0.2358 | Epoch 30/30(0.2358) | 77.75% |




圖八

可以從表三看到加上一層後她的效能就有改變了，基本上每個數值都有明顯的上升，而且也沒有 Overfitting 的情形產生。