

National Tsing Hua University

Fall 2023 11210IPT 553000

Deep Learning in Biomedical Optical Imaging

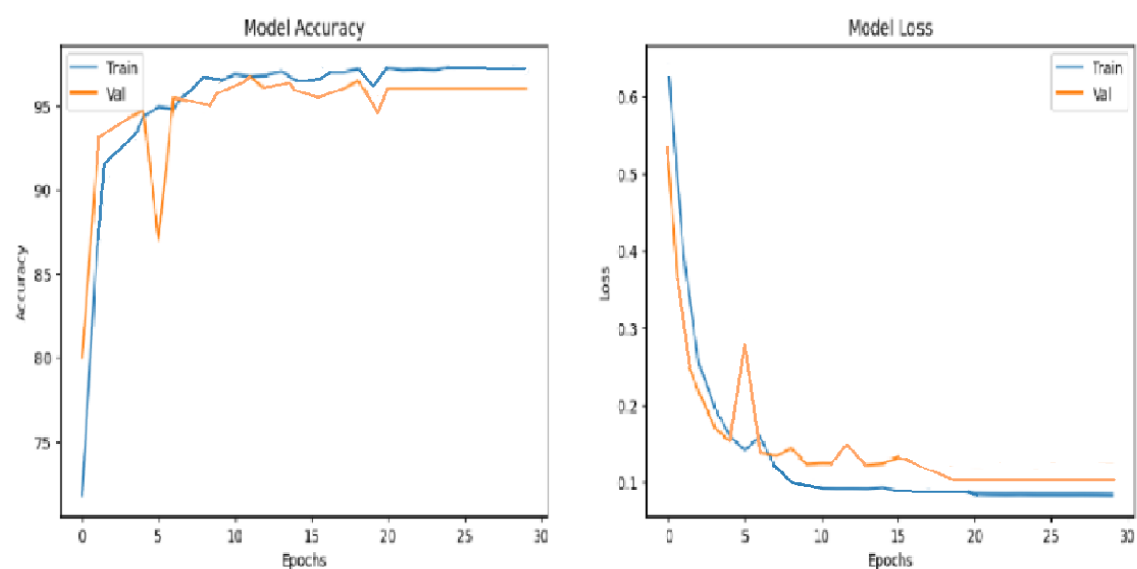
Homework 3

110066540 陳哲瑋

### 1. Task A: Reduce Overfitting

觀察 Train Loss 以及 Val Loss 的增減幅度就會知道訓練模型所呈現的狀態，當 Train Loss 下降但 Val Loss 反而上升或不動，就代表它已經 Overfitting 了，當 Overfitting 時有大概有三種方法可以改善他：

Regularization、減少層數、減少 Node 數，我嘗試用這三種方法來改善 ConvModel，首先先將層數變成一層但是結果還是 Overfitting，所以我查找了網路加入了 BatchNorm1d 函數，結果雖然改善但是還是 Overfitting，最後嘗試將 Node 改為 16，發現竟然已經不會產生 Overfitting 了其結果如下圖所示：



以下是改的過程首先為加入 BatchNorm1d 函數:

```
BN=nn.BatchNorm1d(32)
x = F.relu(self.conv1(x))
x = self.pool1(x)
```

但改善的效果不佳，故先將 layer 改為一層，後來查了網路發現有可能 Overfitting 的其中一原因是結果過度的跟隨訓練集於是將 kernel\_size 改為 5，這樣可以透過改變周圍運算與元素來減少元素的獨立性。如下圖所示:

```
self.conv1 = nn.Conv2d(1, 32, kernel_size=5, stride=1, padding='same')
self.pool1 = nn.MaxPool2d(kernel_size=6, stride=16) # 128*128
```

最後發現 Val Loss 已不再上升了也還沒有開始下降，所以我大膽的將輸出的矩陣改為 16，通過模擬發現將 MaxPool 的 kernel\_size 改成 6，會導致不良的轉換因為公式為 $(256 - \text{kernel\_size}) / \text{stride}$ ，但從多次模擬的結果顯示他所呈現的效果是最好的，如下圖所示:

```
self.conv1 = nn.Conv2d(1, 32, kernel_size=6, stride=1, padding='same')
self.pool1 = nn.MaxPool2d(kernel_size=6, stride=16) # 128*128
```

如果用測試集去模擬此模型可以發現他 Test accuracy 為 79.18%，這數值與原本 CNN 模型的 78.20 %高了快 1%左右。

```
labels_float = labels.float().unsqueeze(1) # Convert labels to float and match shape with outputs
predicted = torch.sigmoid(outputs) > 0.5

test_correct += (predicted.float() == labels_float).sum().item()
test_total += labels.size(0)

print(f'Test accuracy is {100. * test_correct / test_total}%')
```

Test accuracy is 79.18%

下表展示了修改參數前與修改參數後的差異:

	最後的 Train loss	最後的 Train acc	最後的 Val loss	最後的 Val acc	最佳 Val loss 的位置	最佳 Val loss 的數值
修改前	0.0093	99.16%	0.0978	95.47%	Epoch21/30	0.961
修改後	0.0829	96.97%	0.1315	95.81%	Epoch28/30	0.1304

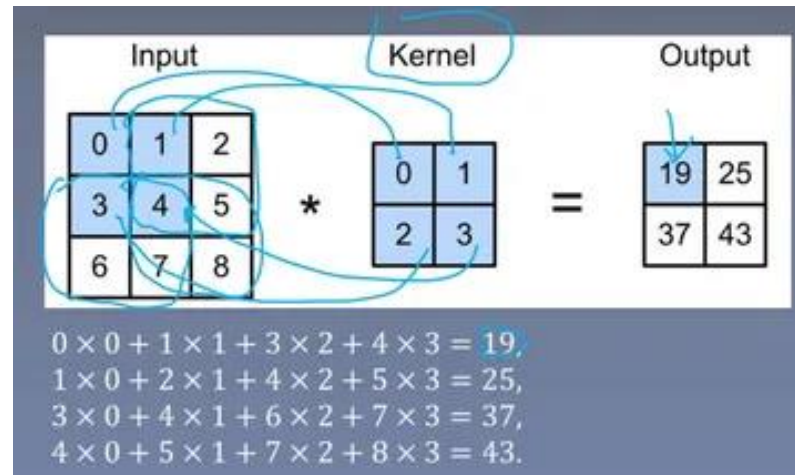
可以從發現到修改參數前之最後的 Train acc-最後的 Val acc=3.69%，且可看到 Val Loss 在 Epoch 21/30 後反而上升了，這表示它 Overfitting 了。修改參數後之最後的 Train acc 減掉最後的 Val acc=1.16%，比改動前好了許多，且 Val Loss 的最佳值是在 Epoch 29/30 的地方，說明了此次改動對於整個機械學習是有相當好的提升。

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

ANN 與 CNN 結構的差異:

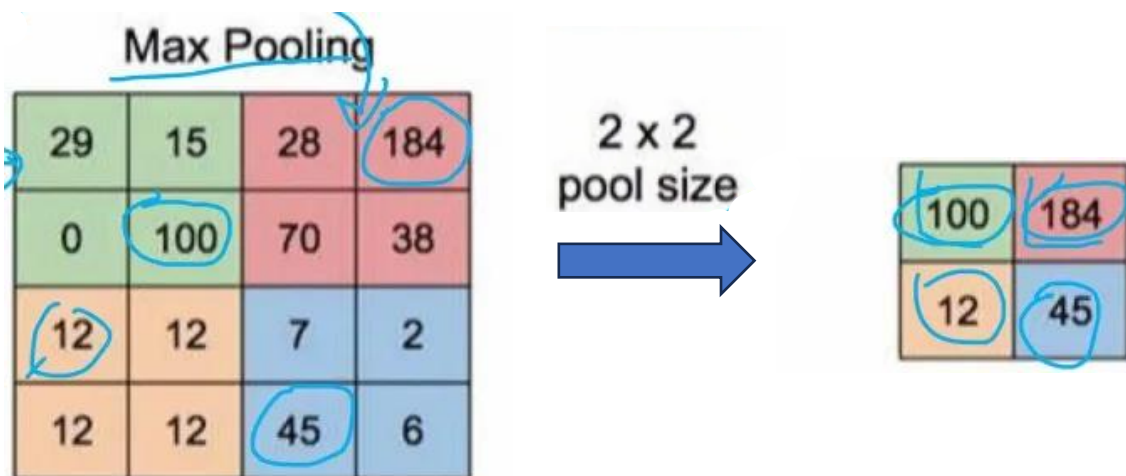
CNN 一般會分為 Convolution、Pooling、Input、Output 四個 Layer

Convolution Layer 作用為將 Input Layer 傳進來的矩陣分成不同區塊進行處理，如下圖所示:

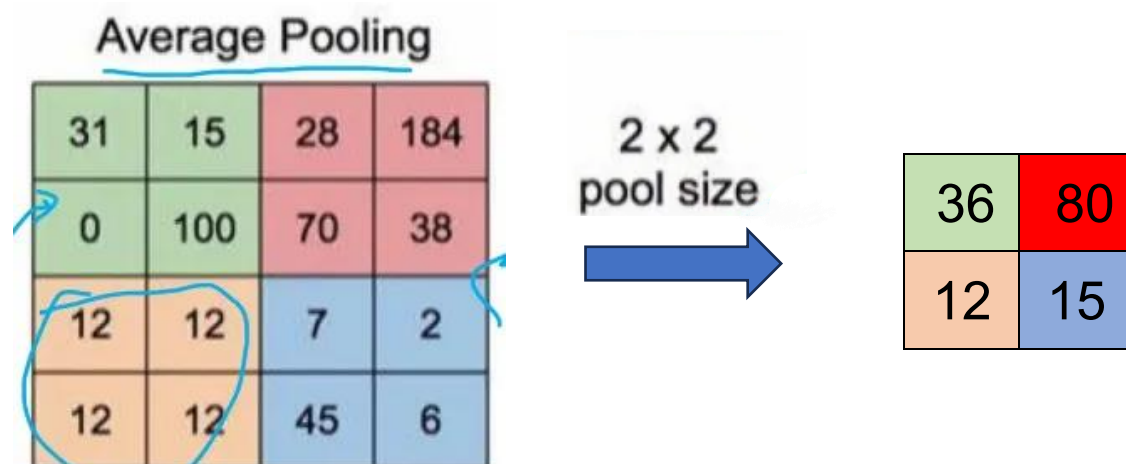


而 Pooling Layer 的功用是將 Convolution Layer 輸出矩陣的維度縮小，主要有兩種方法來縮小矩陣

一種是 Max Pooling:



另一種為 Average Pooling:



ANN 一般則會分為 Input、Hidden、Output 三個 Layer，資料從 Input Layer 傳至 Hidden Layer 後會被分類並處理，如果  $y$  為 input、Activation function 用  $\sigma$  表示、weights 用  $w$  表示、training label 用  $x$  表示、bias 則用  $b$  表示，單 Node 的輸入與輸出的關係可寫為  $x = \sigma(wy + b)$ ，如果只有單 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  $x_2 = \sigma(wx_1 + b)$ 。當數值被傳到 Output Layer 後會將此資料分類，來區分他為單一 Node 又或為數個 Node。

下表格為 CNN 以及 ANN 的比較表，其中有幾個重要的參數分別為 Final Train acc、Final Val acc、Final Val Loss、最佳 Val Loss 的位置、最佳 Val loss 的數值及 Test accuracy:

	Final Train acc	Final Val acc	Final Val Loss	最佳 Val Loss 的位置	最佳 Val loss 的數值	Test accuracy	Time
CNN	100%	96.89%	0.1297	12/30	0.1091	77.86%	75s
ANN	98.1%	95.16%	0.1859	22/30	0.1812	72.56%	18s

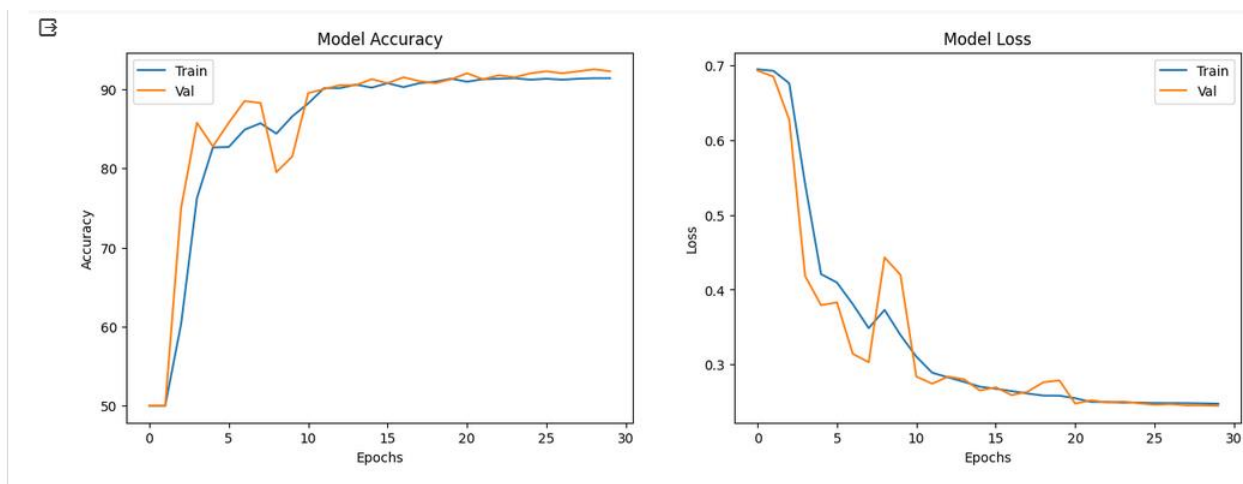
從結果可以明顯看到 CNN 的有相當明顯的優點，CNN 的 Train acc 及 Val acc 都比 ANN 高，CNN 的 Test accuracy 也比 ANN 高了 5.3%。而 ANN 的優點就是其運行時間為 18 秒，比 CNN 的 75 秒短了相當多，兩者都會有 Overfitting 情況產生，要改善 ANN 的 Overfitting 情況也比 CNN 更容易因為其最佳 Val Loss 位置在 22 比 CNN 的 12 還要後面也是其少數的優點。

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

GAP 是將最後的 Pooling 層分層後輸出，發現原本的模型會產生 Underfitting 的情形，要改善這問題的方法就是將模型的複雜度提升，故將原本三層的 Hidden Layer 改為四層，以下為表格來做比對:

	Final Train acc	Final Val acc	Final Val Loss	最佳 Val Loss 的位置	最佳 Val loss 的數值	Test accuracy
三層的 Hidden Layer	88.18%	88.96%	0.2924	27/30	0.3198	78.46%
四層的 Hidden Layer	91.38%	92.25%	0.2449	30/30	0.2449	80.75%

以下為訓練結果：



從表格可很明顯看到多加一層後整體就有相當明顯的效果了，每個數值都有明顯的上升，**loss** 也都比較少，所以證明此方法是相當可行且有效的。

#### Load Trained Model and Evaluate

```
# Load the trained weights
model.load_state_dict(torch.load('model_classification.pth'))

# Set the model to evaluation mode
model.eval()

test_correct = 0
test_total = 0

with torch.no_grad():
    for images, labels in test_loader:

        images = images.cuda()
        images = images / 255.

        labels = labels.cuda()

        outputs = model(images)

        labels_float = labels.float().unsqueeze(1) # Convert labels to float and match shape with outputs
        predicted = torch.sigmoid(outputs) > 0.5

        test_correct += (predicted.float() == labels_float).sum().item()
        test_total += labels.size(0)

print(f'Test accuracy is {100. * test_correct / test_total}%')
```

Test accuracy is 80.75%

```
100% 30/30 [01:12<00:00, 2.44s/it]
Epoch 1/30, Train loss: 0.6942, Train acc: 50.00%, Val loss: 0.6925, Val acc: 50.00%, Best Val loss: 0.6925 Best Val acc: 50.00%
Epoch 2/30, Train loss: 0.6922, Train acc: 50.00%, Val loss: 0.6844, Val acc: 50.00%, Best Val loss: 0.6844 Best Val acc: 50.00%
Epoch 3/30, Train loss: 0.6755, Train acc: 60.31%, Val loss: 0.6269, Val acc: 75.00%, Best Val loss: 0.6269 Best Val acc: 75.00%
Epoch 4/30, Train loss: 0.5410, Train acc: 76.25%, Val loss: 0.4181, Val acc: 85.75%, Best Val loss: 0.4181 Best Val acc: 85.75%
Epoch 5/30, Train loss: 0.4207, Train acc: 82.62%, Val loss: 0.3793, Val acc: 82.75%, Best Val loss: 0.3793 Best Val acc: 85.75%
Epoch 6/30, Train loss: 0.4095, Train acc: 82.69%, Val loss: 0.3630, Val acc: 85.75%, Best Val loss: 0.3793 Best Val acc: 85.75%
Epoch 7/30, Train loss: 0.3806, Train acc: 84.88%, Val loss: 0.3142, Val acc: 88.50%, Best Val loss: 0.3142 Best Val acc: 88.50%
Epoch 8/30, Train loss: 0.3486, Train acc: 85.69%, Val loss: 0.3031, Val acc: 88.25%, Best Val loss: 0.3031 Best Val acc: 88.50%
Epoch 9/30, Train loss: 0.3730, Train acc: 84.38%, Val loss: 0.4429, Val acc: 79.50%, Best Val loss: 0.3031 Best Val acc: 88.50%
Epoch 10/30, Train loss: 0.3394, Train acc: 86.56%, Val loss: 0.4196, Val acc: 81.50%, Best Val loss: 0.3031 Best Val acc: 88.50%
Epoch 11/30, Train loss: 0.3105, Train acc: 88.19%, Val loss: 0.2839, Val acc: 89.50%, Best Val loss: 0.2839 Best Val acc: 89.50%
Epoch 12/30, Train loss: 0.2890, Train acc: 90.12%, Val loss: 0.2744, Val acc: 90.00%, Best Val loss: 0.2744 Best Val acc: 90.00%
Epoch 13/30, Train loss: 0.2828, Train acc: 90.12%, Val loss: 0.2838, Val acc: 90.50%, Best Val loss: 0.2744 Best Val acc: 90.50%
Epoch 14/30, Train loss: 0.2768, Train acc: 90.56%, Val loss: 0.2803, Val acc: 90.50%, Best Val loss: 0.2744 Best Val acc: 90.50%
Epoch 15/30, Train loss: 0.2703, Train acc: 90.19%, Val loss: 0.2653, Val acc: 91.25%, Best Val loss: 0.2653 Best Val acc: 91.25%
Epoch 16/30, Train loss: 0.2676, Train acc: 90.75%, Val loss: 0.2695, Val acc: 90.75%, Best Val loss: 0.2653 Best Val acc: 91.25%
Epoch 17/30, Train loss: 0.2645, Train acc: 90.25%, Val loss: 0.2591, Val acc: 91.50%, Best Val loss: 0.2591 Best Val acc: 91.50%
Epoch 18/30, Train loss: 0.2613, Train acc: 90.75%, Val loss: 0.2637, Val acc: 91.00%, Best Val loss: 0.2591 Best Val acc: 91.50%
Epoch 19/30, Train loss: 0.2585, Train acc: 90.94%, Val loss: 0.2763, Val acc: 90.75%, Best Val loss: 0.2591 Best Val acc: 91.50%
Epoch 20/30, Train loss: 0.2583, Train acc: 91.31%, Val loss: 0.2788, Val acc: 91.25%, Best Val loss: 0.2591 Best Val acc: 91.50%
Epoch 21/30, Train loss: 0.2550, Train acc: 90.94%, Val loss: 0.2478, Val acc: 92.00%, Best Val loss: 0.2478 Best Val acc: 92.00%
Epoch 22/30, Train loss: 0.2500, Train acc: 91.25%, Val loss: 0.2522, Val acc: 91.25%, Best Val loss: 0.2478 Best Val acc: 92.00%
Epoch 23/30, Train loss: 0.2500, Train acc: 91.31%, Val loss: 0.2494, Val acc: 91.75%, Best Val loss: 0.2478 Best Val acc: 92.00%
Epoch 24/30, Train loss: 0.2492, Train acc: 91.38%, Val loss: 0.2505, Val acc: 91.50%, Best Val loss: 0.2478 Best Val acc: 92.00%
Epoch 25/30, Train loss: 0.2491, Train acc: 91.19%, Val loss: 0.2481, Val acc: 92.00%, Best Val loss: 0.2478 Best Val acc: 92.00%
Epoch 26/30, Train loss: 0.2486, Train acc: 91.31%, Val loss: 0.2463, Val acc: 92.25%, Best Val loss: 0.2463 Best Val acc: 92.25%
Epoch 27/30, Train loss: 0.2486, Train acc: 91.19%, Val loss: 0.2472, Val acc: 92.00%, Best Val loss: 0.2463 Best Val acc: 92.25%
Epoch 28/30, Train loss: 0.2484, Train acc: 91.31%, Val loss: 0.2455, Val acc: 92.25%, Best Val loss: 0.2455 Best Val acc: 92.25%
Epoch 29/30, Train loss: 0.2481, Train acc: 91.26%, Val loss: 0.2455, Val acc: 92.50%, Best Val loss: 0.2455 Best Val acc: 92.50%
Epoch 30/30, Train loss: 0.2477, Train acc: 91.38%, Val loss: 0.2449, Val acc: 92.25%, Best Val loss: 0.2449 Best Val acc: 92.50%
```