深度學習 作業一

學號：312831002　姓名：廖健棚

作業問題

這份報告需要比較您的 ResNet 模型與上週範例代碼中的模型之間的差異，然後嘗試解釋為什麼 ResNet 的性能優於範例模型。在最後回答

ResNet model 實作(程式碼)

```
class ResBlock(nn.Module):
    def __init__(self, input_dim, block_dim, stride=1, activation=nn.ReLU(inplace=True)):
        super(ResBlock, self).__init__()
        self.input_dim = input_dim
        self.output_dim = block_dim*4
        self.stride = stride
        self.activation = nn.Sequential(activation)
        self.block = nn.Sequential(
            nn.Conv2d(input_dim,block_dim,1,stride=self.stride,bias=False),
            nn.BatchNorm2d(block_dim),
            activation,
            nn.Conv2d(block_dim,block_dim,3,stride=1,padding=1,bias=False),
            nn.BatchNorm2d(block_dim),
            activation,
            nn.Conv2d(block_dim,self.output_dim,1,stride=1,bias=False),
            nn.BatchNorm2d(self.output_dim)
        )
        self.downsample = nn.Sequential(
            nn.Conv2d(input_dim, self.output_dim,1,stride=self.stride,bias=False),
            nn.BatchNorm2d(self.output_dim)
        )
    def forward(self, x):
        tmp = self.block(x)
        if self.stride != 1 or self.input_dim != self.output_dim:
            res = self.downsample(x)
        else:
            res = x
        tmp = tmp + res
        return self.activation(tmp)
```

ResBlock 程式碼

```
class ResNet50(nn.Module):
    def __init__(self, n_class, activation=nn.ReLU(inplace=True)):
        super(ResNet50, self).__init__()
        self.foot = nn.Sequential(  # [H,W]: 224 -> 112 -> 56
            nn.Conv2d(3,64,7,stride=2, padding = 3, bias=False),
            nn.BatchNorm2d(64),
            activation,
            nn.MaxPool2d(3,stride=2, padding=1)
        )
        self.block1 = nn.Sequential(# [H,W]: 56 -> 56
            ResBlock(64,64,1, activation),
            ResBlock(256,64,1, activation),
            ResBlock(256,64,1, activation)
        )
        self.block2 = nn.Sequential(# [H,W]: 56 -> 28
            ResBlock(256,128,2, activation),
            ResBlock(512,128,1, activation),
            ResBlock(512,128,1, activation),
            ResBlock(512,128,1, activation)
        )
        self.block3 = nn.Sequential(# [H,W]: 28 -> 14
            ResBlock(512,256,2, activation),
            ResBlock(1024,256,1, activation),
            ResBlock(1024,256,1, activation),
            ResBlock(1024,256,1, activation),
            ResBlock(1024,256,1, activation),
            ResBlock(1024,256,1, activation)
        )
        self.block4 = nn.Sequential(# [H,W]: 14 -> 7
            ResBlock(1024,512,2, activation),
            ResBlock(2048,512,1, activation),
            ResBlock(2048,512,1, activation),
            nn.AdaptiveAvgPool2d((1,1))# [H,W]: 7 -> 1, output_size(1, 1)
        )
        self.fc = nn.Sequential(
            nn.Linear(2048, n_class)
        )
```

```
def forward(self, x):
        x = self.foot(x)
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        x = self.block4(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x
```

ResNet50 程式碼

ResBlock 講解

ResBlock 是構成 ResNet 中的基本單元，用於建構深度殘差神經網路。 一個 ResBlock 包含了數個 Convolution Layer(簡寫:Conv)和 Batch Normalize layer(簡寫:BNorm)。 以下是 ResBlock 在初始化 ResBlock 時需要提供的參數：

- input_dim：輸入通道的維度。
- block_dim：ResBlock 內部捲積核的維度。
- stride：Conv 的步幅，預設為 1。
- activation：啟動函數，預設為 ReLU。
- self.block：這是 ResBlock 的主要組成部分，由 Sequential 包著數個 Conv 和 BNorm。以下是 self.block 的內容：

  Conv_1：1x1 的捲積核，根據 stride 參數是否為 1 來決定是否改變圖高和圖寬，並用於調整輸出通道的維度。
  BNorm_1：用於正規化 Conv_1 的輸出。
  activation：根據使用者輸入的 activation function。
  Conv_2：3x3 的捲積核，stride 固定為 1，用於進行特徵提取。
  BNorm_2：用於正規化 Conv_2 的輸出。
  activation：同上。
  Conv_3：1x1 的卷積核，stride 固定為 1，用於調整輸出通道的維度。
  BNorm_3：用於正規化 Conv_3 的輸出。

- self.downsample：這是一個用於相容維度的部分，當 stride 不等於 1 或輸入通道維度不等於輸出通道維度時，它將用於調整輸入的維度以相容 ResBlock 的輸出。

forward：ResBlock 的前向傳播方式。 首先，通過進行 Conv 和 BNorm 操作。 然後，如果 strid 不等於 1 或輸入通道維度不等於輸出通道維度，將使用進行相應的調整。最後，將調整后的輸出與原始輸入相加，然後接上 activation，並回傳最終的輸出。

ResNet50 講解

ResNet50 是一個深度殘差神經網路模型，它由多個 ResBlock 組成，用於圖像分類任務。 以下是 ResNet50 在初始化 ResNet50 時，需要提供的參數：

- n_class：輸出類別的數量。
- activation：啟動函數，預設為 ReLU。
- self.foot：模型的開頭的部分，包括以下層：

  Conv：7x7 的卷積核，stride 為 2，用於圖像的初始特徵提取。
  BNorm：用於正規化 Conv_1 的輸出。
  activation：根據使用者輸入的 activation function
  MaxPool：3x3 的最大池化，留下 9 格中最大，stride 為 2，用於縮小圖像尺寸。

- self.block[1,2,3,4] ：這些 block 分別包含多個 ResBlock，用於構建深度網路。每個部分包括多個 ResBlock，提取不同階段的特徵。在 ResNet50 中，這些部分依序縮小圖像尺寸並增加通道維度。在 block4 中最後使用 AdpativeAvgPool 將圖片大小縮小至 1x1。
- self.fc：全連接層，將最後一個 ResBlock 的輸出拉平後連接到這裡，然後經過一個全連接層 Linear(2048, n_class)，最終輸出預測的類別機率。

forward：ResNet50 的前向傳播方式。首先，圖像經過 self.foot，然後依次經過 block1、block2、block3、block4，拉平後後經過全連接層得到最終的類別預測。

ResNet50 是一個具有深度 Residual Block 的深度卷積神經網路，用於圖像分類等任務，具有良好的性能和訓練效果。

實驗

嘗試使用三種不同的 activation function 並比較他們的表現。

本實驗中使用環境:

硬體

- CPU: Intel(R) Core(TM) i5-10400 6 核
- GPU: NVIDIA GeForce RTX 3080

模型

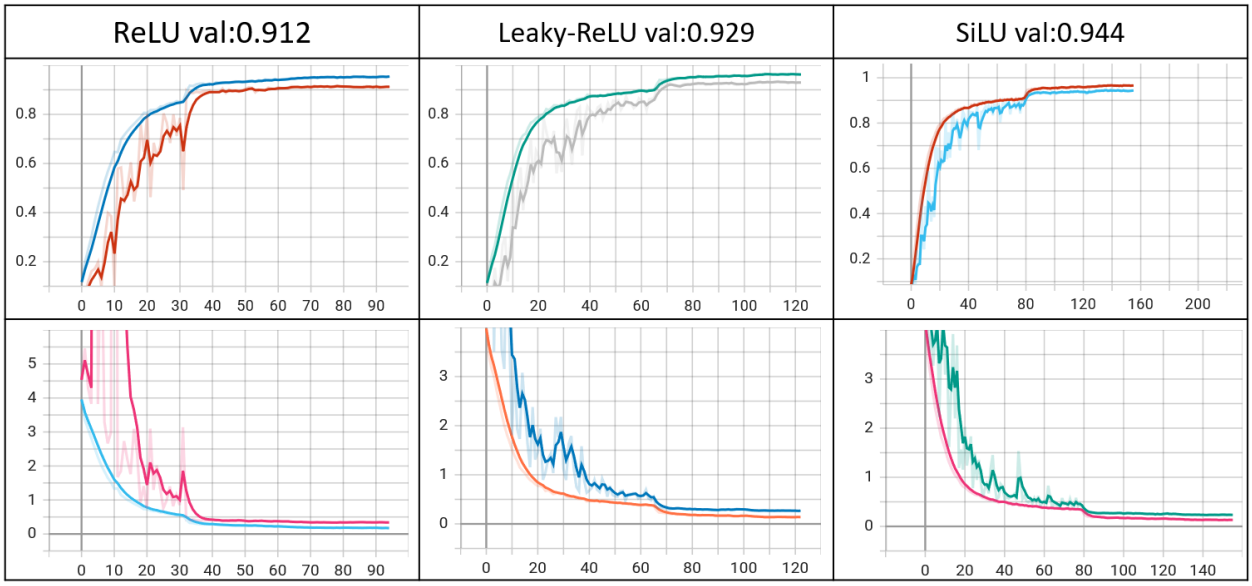
- ResNet50 架構
- Data Preprocess：Resize(256), Crop(224), ToTensor (), Normalize()
- Data Augmentation：Random[Crop, HFlip, Rotation, Grayscale], ColorJitter
- Optimizer：Nadam
- Criterion：CrossEntropyLoss
- Learning rate：0.001
- L2 weight decay：1e-4
- Scheduler：ReduceLROnPlateau
- Early stop：15 times
- Train Data：使用 6149 筆的測試資料
- Valid Data：使用原本 1020 筆的驗證資料
- Test Data：使用 1020 筆的訓練資料

實驗方式

設定 ResNet50 參數 activation，替換掉模型中所有的啟動函數。

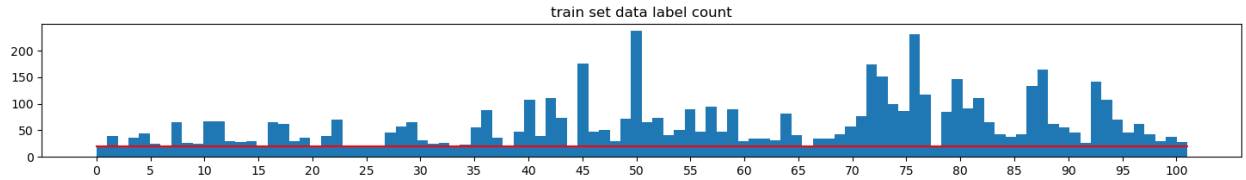
- ReLU：常見的啟動函數，輸入大於 0 不變，小於 0 變為 0
- Leaky-ReLU：Leaky ReLU 在負數輸入值時不將其為 0，而是將其乘以一個小的正數
- SiLU：透過使用 Sigmoid 函數對輸入值進行平滑處理

訓練期間 Accuracy & Loss (Train & Valid)



實驗結果

測試資料各標籤分布(紅線: 20 筆以下)



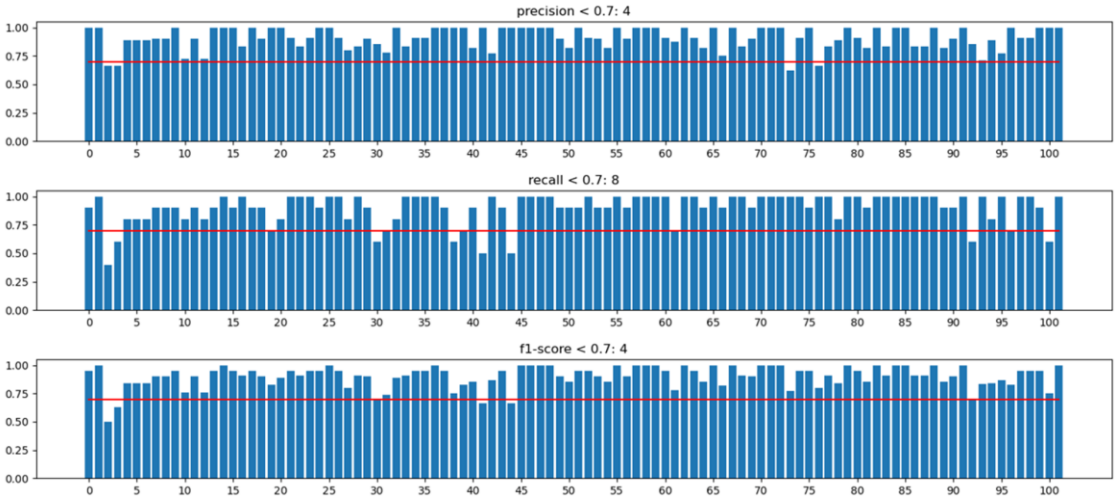
統計 Accuracy, Precision, Recall, F1-score

	Test accuracy	Precision	Recall	F1-score
ReLU	0.902941	0.911882	0.902941	0.900080
Leaky ReLU	0.926471	0.932657	0.926471	0.925030
SiLU	0.938235	0.944589	0.938235	0.936527

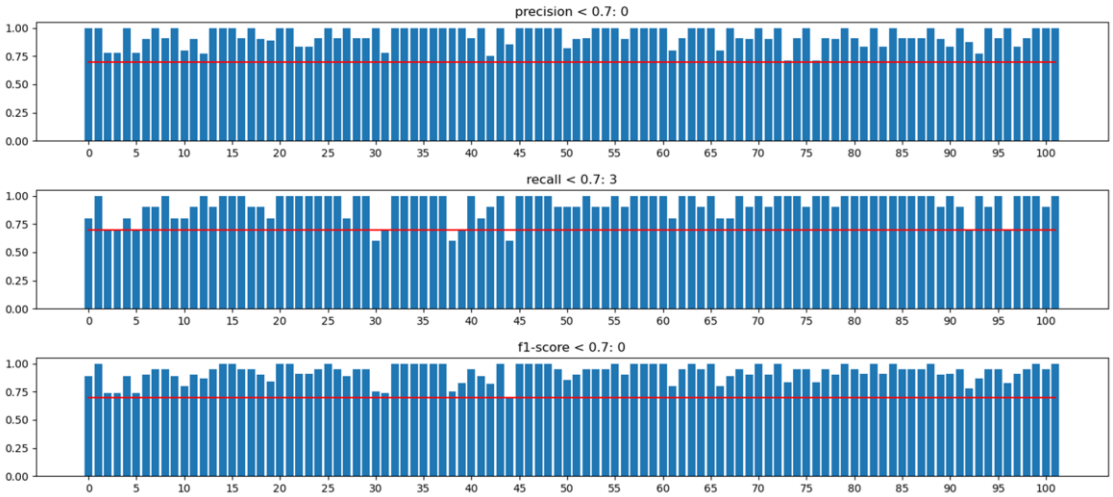
根據 label 製作的直條圖

直條圖由 label 及數值所組成，圖中我將 0.7 設為及格門檻，在 ReLU precision 圖中表現可以得知有 4 組類別未達 0.7，其中 label 3、4、73、76 的得分不是很高，另外 ReLU recall 途中有 8 組未達標，可以得知雖然 precision 很高，但 recall 分數偏低，預測狀況沒有回饋到真實資料，例如:在 label 3 的分類 precision 接近 0.7 但 recall 卻只有 0.4，並且從訓練資料 label 3 的數量，可以得出 label 3 相較其他比較少資料也比較難，也導致訓練效果不佳。其他 activation function 也是以此類推。

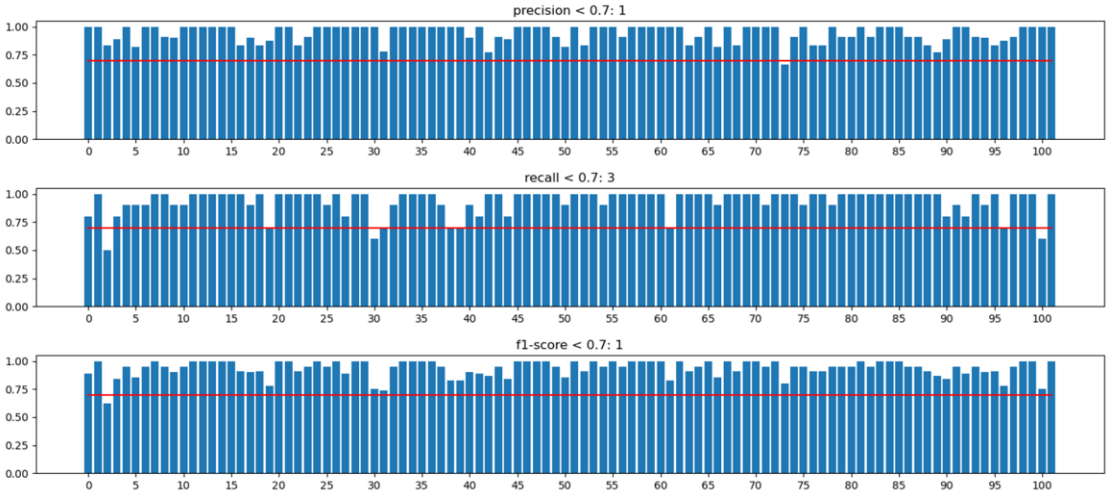
ReLU



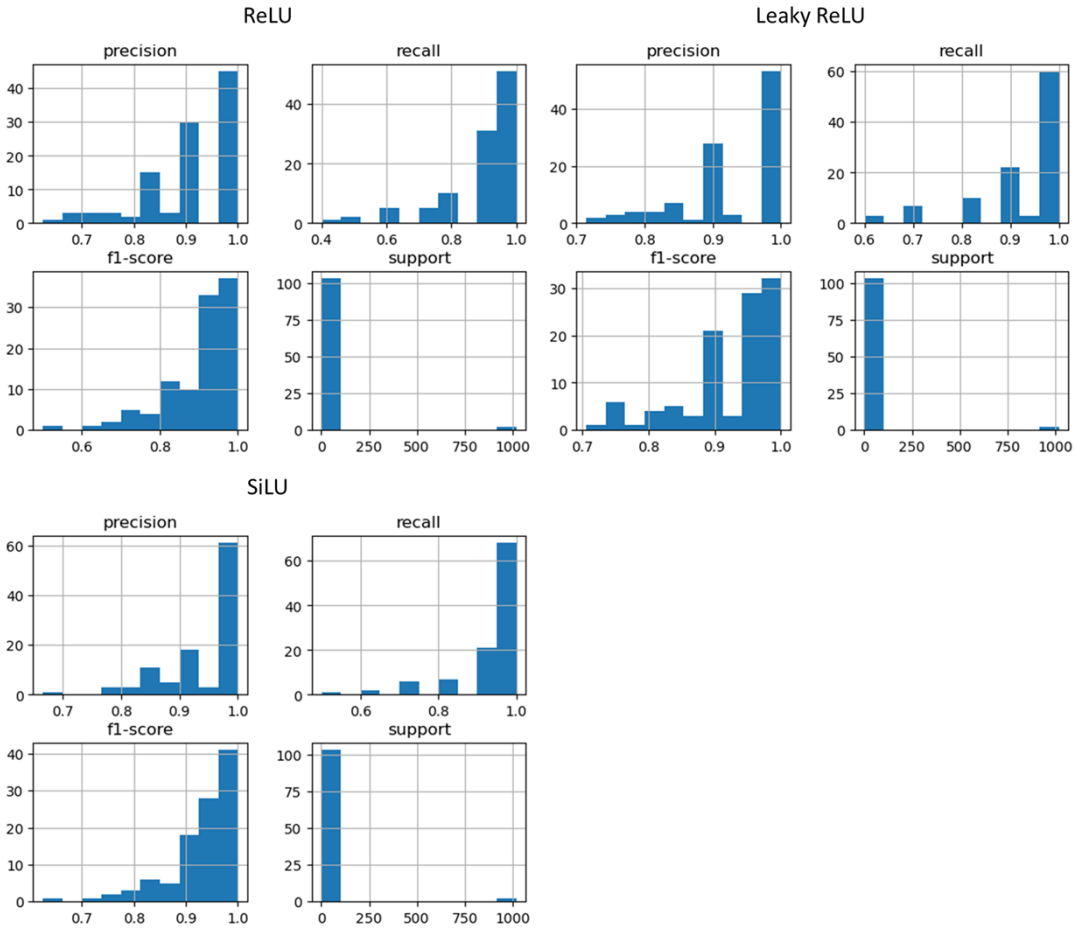
Leaky ReLU



SiLU

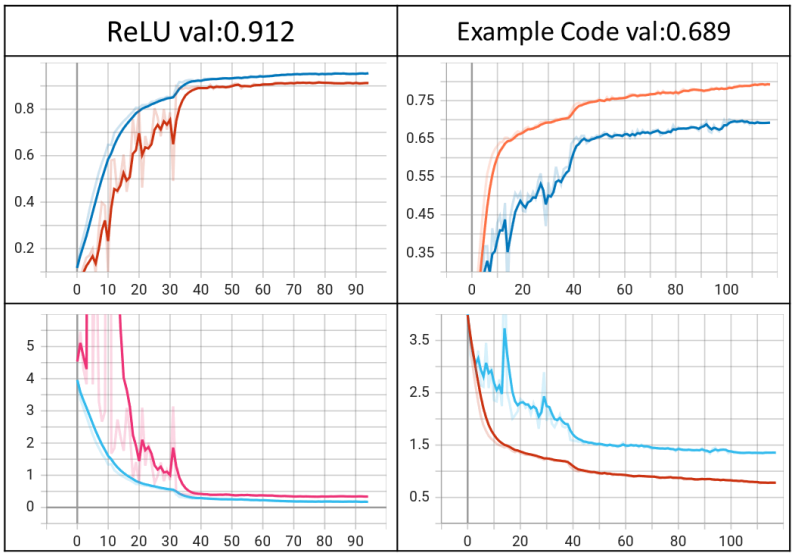


統計各評分指標的占比



解釋為什麼 ResNet 會比範例程式碼好

訓練期間 Accuracy & Loss (Train & Valid)



實驗結果

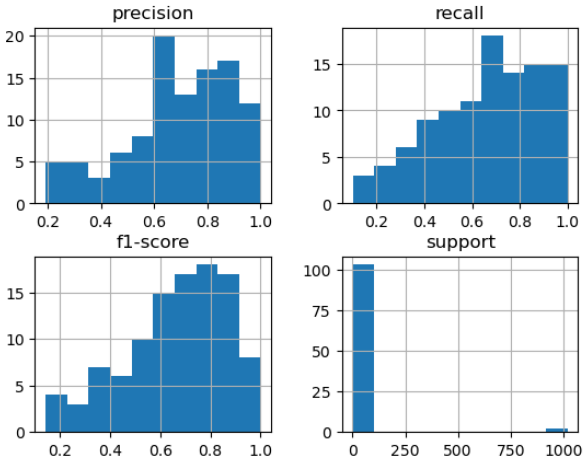
統計 Accuracy, Precision, Recall, F1-score

	Test accuracy	Precision	Recall	F1-score
ReLU	0.902941	0.911882	0.902941	0.900080
Example code	0.669608	0.708112	0.669608	0.662007

根據 label 製作的直條圖

可以發現相較 ResNet 的訓練結果明顯有很大的落差。

統計各評分指標的占比



ResNet 與傳統 CNN 的差距

ResNet 引入 Residual Block，這個部分讓模型可以疊得更深。在傳統的 CNN 中，層與層之間是順序連接的，而在 ResNet 中，每個 Residual Block 都包括跳過連接，允許資訊在網路中跳躍傳遞。這意味著在訓練過程中，網路可以學習將輸入資訊直接傳遞到後續層，從而減輕了梯度消失及梯度爆炸問題，使得非常深的網路足以訓練起來。

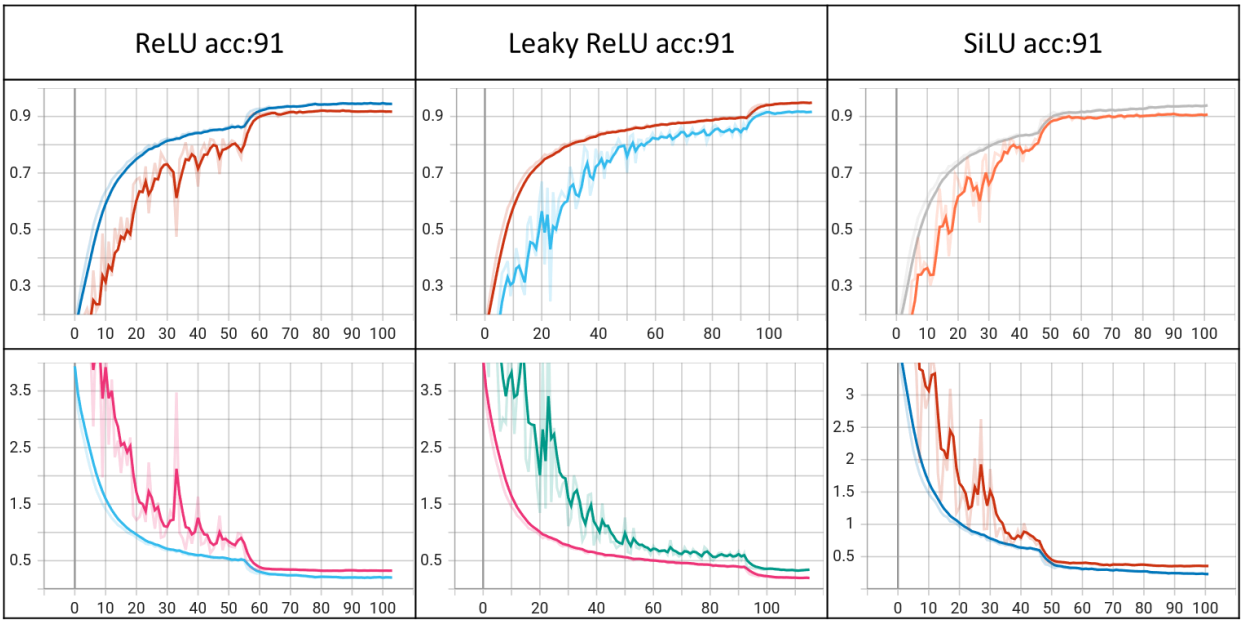
2015 年的 ResNet 論文中發現，當普通深度神經網路變得非常深時，通常會出現一個奇怪的現象，即訓練誤差不但沒有下降，反而會增加。這種情況被稱為退化（degradation）問題，它的意思是，隨著網路變得更深，模型的性能不再提升，甚至變得不穩定。這通常不是由於過度擬合造成的，而是因為網路變得太深，導致訓練誤差變得非常高。

ResNet 論文的作者為了克服訓練超深神經網路的困難，提出了一種稱為恆等映射（Identity Mapping）的想法。這種想法的核心是，無論神經網路有多深，模型的目標應該是學習將輸入 x 直接映射為 H(x)，即輸出應該等於輸入。換句話說，如果我們有一個 50 層的深網路和一個 20 層的網路（20 層網路只是 50 層網路的一部分），那麼它們的訓練誤差不應該相差太多。然而，實驗結果顯示，嘗試使用恆等映射來訓練深層神經網路幾乎無法在合理的時間內獲得足夠好的準確性，也就是無法實現恆等映射方式。因此，作者提出了殘差映射（Residual Mapping）的概念來訓練深層神經網路。這種方法的關鍵是將要學習的映射函數從 H(x)轉換為 F(x) := H(x) - x，也就是將輸入 x 減去自身，這就是所謂的殘差。作者認為，最佳化的殘差的方式比直接學習恆等映射更容易。模型只需努力將每個 Residual Block 學會如何將輸入的殘差變為 0，而不需要一次性學習完整的恆等映射函數。這樣，我們可以將原本要學習的映射函數 H(x)表示為 F(x) + x。

為了實現 F(x) + x，作者引入了 Shortcut Connections，這是一種恆等映射，將上一層的輸入 x 直接加到經過非線性轉換的輸出 F(x)上。這樣做不僅可以獲得輸出 F(x) + x，還可以避免增加額外的參數和計算量。整個神經網路結構仍然可以使用梯度下降等最佳化進行訓練，並且容易實現在常見的深度學習框架中。

其他模型(不小心漏掉一層 activation function)

在初期手刻模型時漏掉一個 activation function 在 ResBlock 最後回傳前，結果在做完 3 種 activation function 後竟然得分還不差，測試 accuracy 都還有 91%。



參考資料

i. HE, Kaiming, et al. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. p. 770-778.

ii. Lee, C. M. (2023, February 2). 直觀理解 ResNet 一簡介、 觀念及實作 (Python Keras) - Chi Ming Lee - Medium. Medium. https://medium.com/@rossleecooloh/%E7%9B%B4%E8%A7%80%E7%90%86%E8%A7%A3resnet-%E7%B0%A1%E4%BB%8B-%E8%A7%80%E5%BF%B5%E5%8F%8A%E5%AF%A6%E4%BD%9C-python-keras-8d1e2e057de2