

EAILab 1 Report

學號：NM6131051 姓名：張芷晴

Task 1

一、資料切分：

```
def _split_dataset(dataset, split_ratio):  
    # 學生實作部分：split dataset into training and validation sets  
    # hint: return Subset(dataset, train_indices), Subset(dataset, valid_indices)  
    np.random.seed(23)  
    total = len(dataset)  
    ind = np.random.permutation(total)  
    split = int(total * (1-split_ratio))  
    train_indices = ind[:split]  
    valid_indices = ind[split:]  
  
    return Subset(dataset, train_indices), Subset(dataset, valid_indices)
```

```
train set: 54000 images  
validation set: 6000 images  
test set: 10000 images  
x shape: (1, 784)  
y shape: (1, 10)
```

首先，先設定亂數種子 `np.random.seed(23)`，確保每次執行程式時資料的切分結果一致。然後我根據 `split_ratio` 來計算切分點，`split = int(total * (1 - split_ratio))`，代表有一定比例的資料會被分配到訓練集，其餘的則成為驗證集。最後，使用 `Subset(dataset, train_indices)` 和 `Subset(dataset, valid_indices)` 建立對應的子資料集。

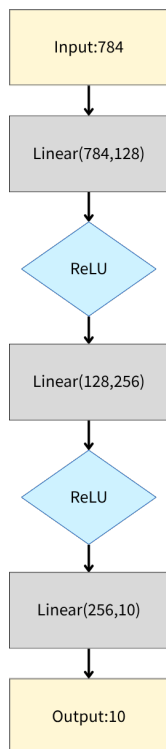
經過切分後，整體的資料分配如下：

- 訓練集：54,000 張影像
- 驗證集：6,000 張影像
- 測試集：10,000 張影像

二、Hyper Parameters:

- Learning Rate:0.0001
- Epochs:8

三、模型架構



```

class MLP(Module):
    def __init__(self) -> None:
        # 學生實作部分: design your Model architecture here
        in_feature = 784
        hidden1 = 128
        hidden2 = 256
        out = 10
        self.fc1 = Linear(in_feature,hidden1)
        self.at1 = ReLU()
        self.fc2 = Linear(hidden1,hidden2)
        self.at2 = ReLU()
        self.fc3 = Linear(hidden2,out)

    def forward(self, x):
        # 學生實作部分: compute forward pass through your model
        x = self.fc1.forward(x)
        x = self.at1.forward(x)
        x = self.fc2.forward(x)
        x = self.at2.forward(x)
        x = self.fc3.forward(x)

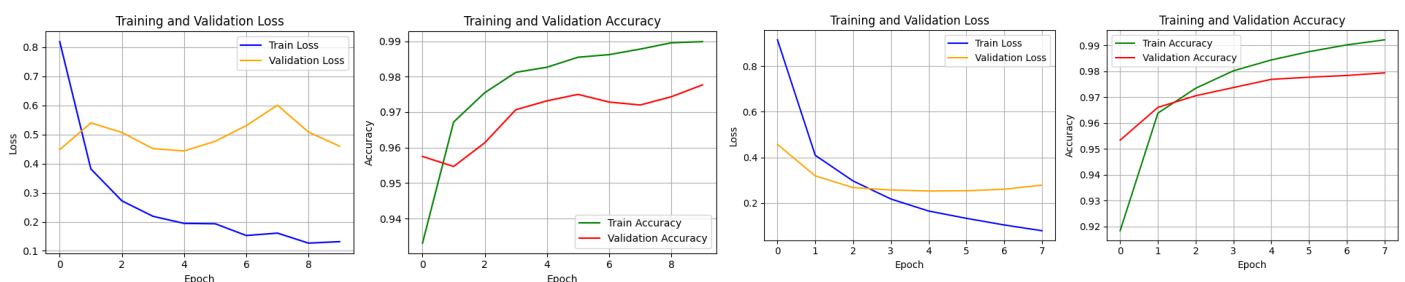
        return x
    def backward(self, dy):
        # 學生實作部分: compute backward pass through your model
        dy = self.fc3.backward(dy)
        dy = self.at2.backward(dy)
        dy = self.fc2.backward(dy)
        dy = self.at1.backward(dy)
        dy = self.fc1.backward(dy)

        return dy
  
```

MLP 的架構依序由一個輸入層、兩個隱藏層和一個輸出層所組成。模型先把影像攤平成 784 維的向量當作輸入，接著經過第一層 128 個神經元的隱藏層，在進入第二層 256 個神經元的隱藏層。這兩層都使用 ReLU 作為 activation function，最後模型通過一個 fully connect layer 的輸出層，把學到的特徵轉成 10 個類別的預測分數。

四、訓練結果

1. 不同 learning rate 的實驗結果



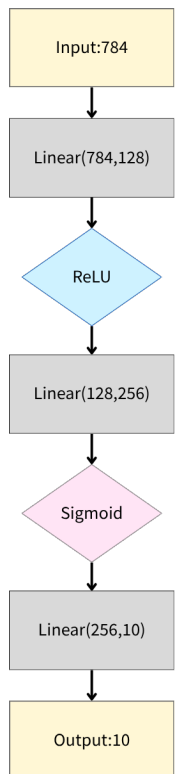
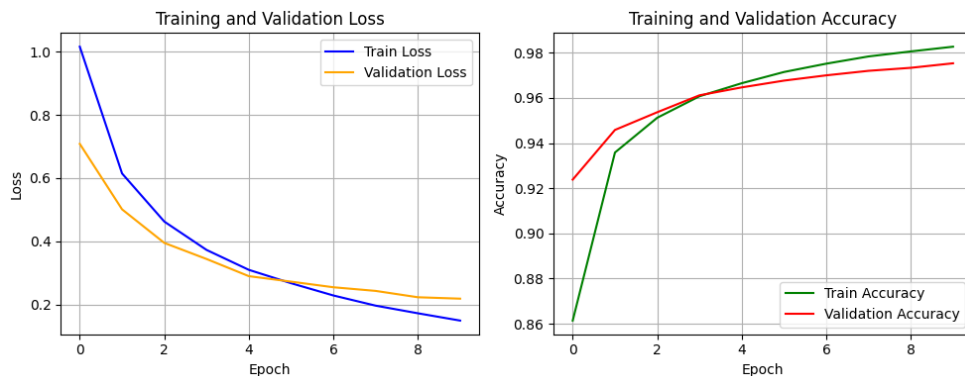
▲ 左圖為將 learning rate 設為 0.001，右圖為將 learning rate 設為 0.0001

第一次訓練時 lr 設為 0.001，從訓練曲線可以觀察到，模型的 training loss 持續下降，但 validation loss 到後期開始上升，出現了輕微的 overfitting 現象。於是之後將 lr 設為 0.0001，讓模型學習速度放慢，調整後的 train loss 和 valid loss

變化趨勢更為穩定，最終 $\text{test_acc} = 0.9789$ 。

2. 更改 activation function

原先模型的兩層隱藏層都使用 ReLU 作為 activation function，新的模型架構將其中一層改為 Sigmoid，雖然 test_acc 沒有什麼變化，準確度為 0.9741，但 validation loss 和 train loss 的走向變得更一致，顯示模型的學習行為變得更加平衡，收斂過程也相對平滑。



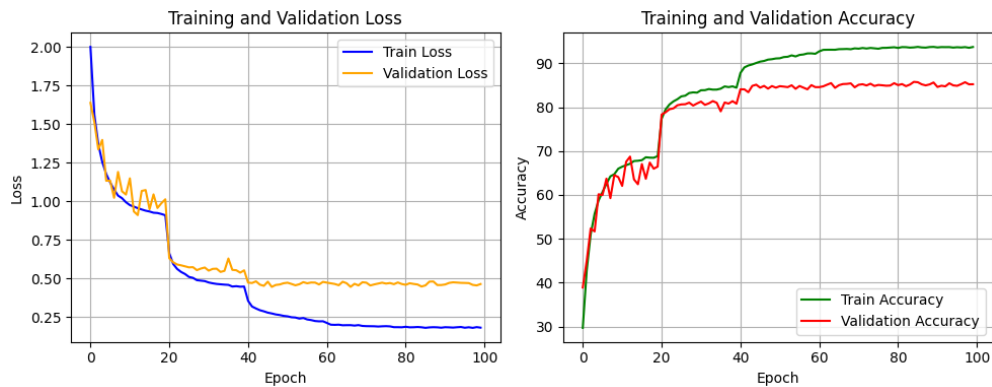
Task2

1. Learning rate

在此次實驗中除了手動調整 learning rate 以外，也有嘗試使用 lr_scheduler。這次採用的是 StepLR，並設定每過 20 個 epoch，就把 lr 乘上 0.1，讓模型在訓練初期能夠快速學習，在後期放慢步伐，讓權重微調的更細緻、穩定。但因為一次訓練 150 個 epoch，一次完整的訓練要花費近一小時的時間，所以沒有嘗試別的 scheduler 方法或調整 step_size 和 gamma，未來可以嘗試看看 LambdaLR()、ExponentialLR()、MultiStepLR()等其他方法。

```
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=20, gamma=0.1)
```

Learning rate	0.0001	0.001	StepLR
Test_acc	82.89	85.16	85.86



▲ 使用 stepLr 時的 train/valid loss and accuracy

2. Data augmentation

● RandomCrop

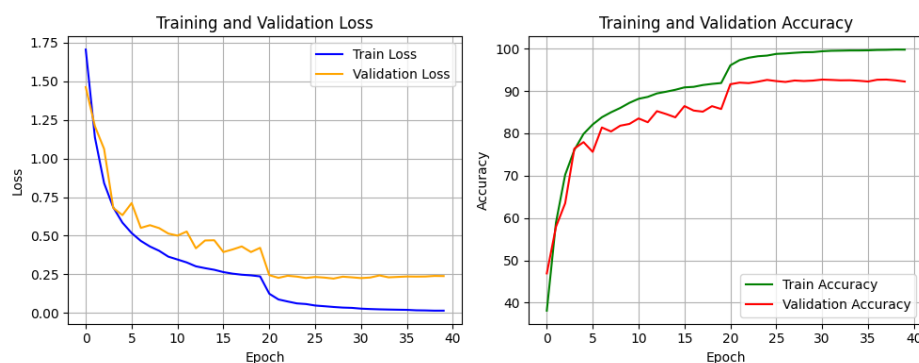
此實驗比較使用不同 data augmentation 方法的差異。完全不使用任何 data augmentation 的狀況下，test_acc = 76.58，加上 RandomCrop 和 RandomHorizontalFlip 之後 test_acc 提升到 85.16。在其他實驗中 data augmentation 也都採用 RandomCrop 和 RandomHorizontalFlip。

```
transforms.RandomCrop(32, padding=4),
transforms.RandomHorizontalFlip(),
```

● Resize

原始的 ResNet 是採用 ImageNet 訓練集去做 training，並且是先放大到 256，再隨機裁成 224 的大小。CIFAR-10 的原圖只有 32 x 32，所以嘗試將 CIFAR-10 透過 Resize 放大到 256 再隨機裁成 224。此時測試集的準確度提高到 92.72%。

```
transforms.Resize(256),
transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),
transforms.RandomHorizontalFlip(),
transforms.ToTensor(), # Transform to tensor
transforms.Normalize(mean=train_mean, std=train_std), # Normalization
])
```



▲ Resize 成 224 的 train/valid loss and accuracy

3. Batch size

以下為不同 Batch size 的實驗結果比較，實驗中除了調整 batch size 外，其餘參數（lr、model architecture、optimizer、epoch....）皆保持一致。

Batch size	16	32	64
Test_acc	84.31	85.16	83.72

從結果可以觀察到，當 batch size 設為 32 時，模型在測試集上的準確率最高。當 batch size 過小時，每次更新的樣本過少，導致梯度估計波動較大，學習過程不夠穩定。



▲ batch size = 64 時的 train/valid loss and accuracy

4. Different optimizer

此實驗為不同 optimizer 之比較，採用的 optimizer 有 Adam(test_acc = 85.16%)和 SGD (test_acc = 84.15%)，從實驗結果來看 Adam 效果比較好，但兩者差異不大。但若有使用 StepLR 的話，optimizer 採用 SGD 會比使用 Adam 時來的好。

```

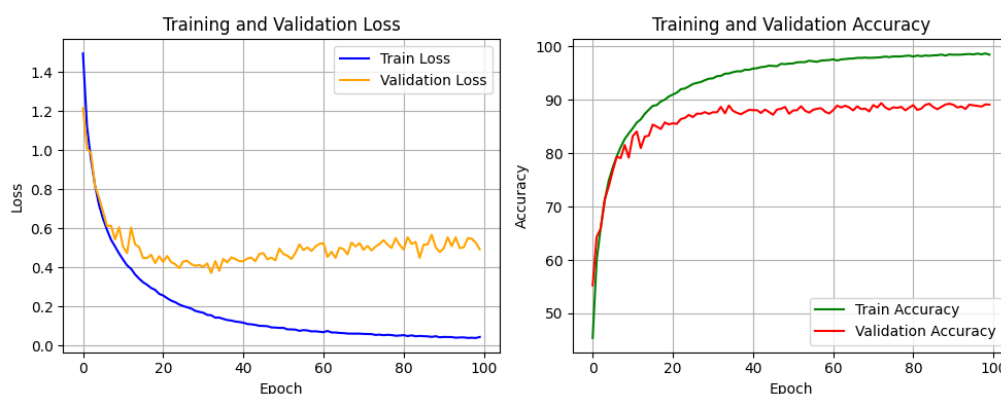
criterion = nn.CrossEntropyLoss()
# optimizer = optim.Adam(model.parameters(), lr=lr)
optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9, weight_decay=5e-4)

```

5. ResNet18 Architecture

原始的 ResNet paper 中，作者使用的是 ImageNet 資料集去做 training，所以 model 的輸入是 224 x 224，但 CIFAR-10 的圖片僅有 32 x 32 大小，所以有嘗試將第一層 convolution layer 的 kernel_size 從 7 x 7 改為 3x3，stride 也從 2 改為 1，在其他條件都相同的狀況下更改 filter size 讓測試集的準確度從 85.16% 上升到 90.03%。

```
class ResNet18(nn.Module):
    def __init__(self, num_classes=1000):
        super(ResNet18, self).__init__()
        # 學生實作部分: Define the ResNet-18 architecture using BasicBlock
        self.in_channels = 64
        # self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False) #original paper
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
```



▲ kernel_size 3 , stride = 1 時的 train/valid loss and accuracy

6. 這次作業遇到的問題和解決方法

在寫這次作業的過程中，主要遇到兩個問題。

在刻 ResNet18 的時候，光看 Lab 的講義有點看不懂，但網路上很多資源和教學，透過其他人對 ResNet 架構的講解與示範，逐步釐清了每個 module 的作用與 data 流向。這個過程讓我更加了解整個 ResNet 架構的設計。

在進行 Task 2 實驗時，發現模型的可調整參數非常多，例如 learning rate、epoch 數量、optimizer 的選擇，以及各種 data augmentation 方法。一開始因為沒有明確規劃實驗方向，就同時嘗試了多種設定，導致在中期階段不太能確定是哪個因素讓模型表現變好。重新思考後，我認為下次在開始實驗前應該先 明確規劃實驗組別與控制變因，例如一次只調整一個參數、其他條件保持固定，才能更清楚地分析各項設定對模型的影響。