生成式人工智慧 Lab01

1. 作業

附件為辨識 cifar-10 的網路模型，但呈現出 overfitting 的樣子，請優化這份程式碼，解決 overfitting 的狀況，相關的優化方法於老師的 ppt 有提過

繳交內容需包括:

1. 改良後的程式碼
2. 成果截圖
3. 請說明如何改良、改良後 loss 以及 accuracy 在折線圖上的變化以及改良的部分，

2. 繳交作業

> *NOTE：修改程式碼有使用 GPT 輔助。*

2.1. 改良後的程式碼

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import torch.nn.functional as F

# 加入 Data Augmentation
transform_train = transforms.Compose([
```

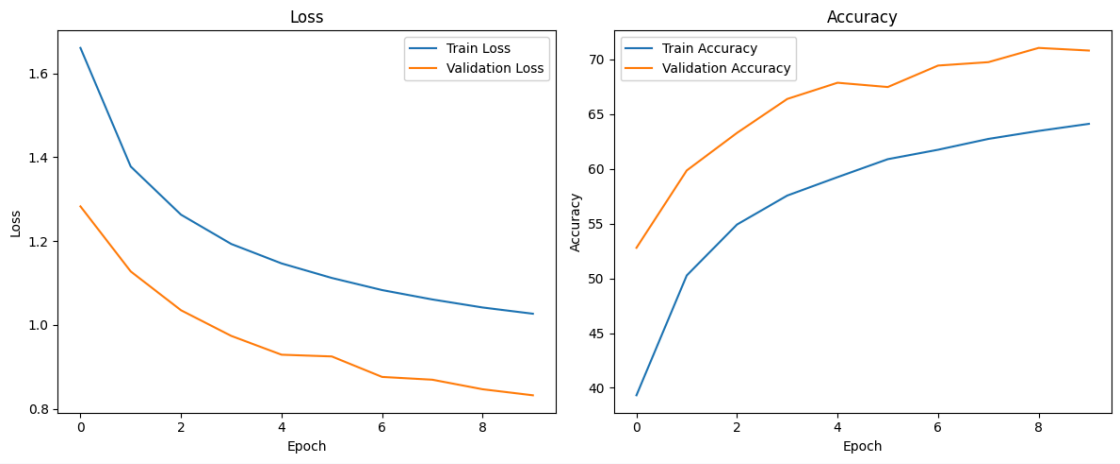
```
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=32, shuffle=True)
testloader = torch.utils.data.DataLoader(testset, batch_size=32, shuffle=False)

# CNN 模型加上 Dropout
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.5)
        self.fc1 = nn.Linear(64 * 8 * 8, 128)  # 降低參數量
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 8 * 8)
```

```
        x = self.dropout(F.relu(self.fc1(x)))
        x = self.fc2(x)
        return x

model = CNNModel()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

train_losses, val_losses, train_accuracies, val_accuracies = [], [], [], []

# 訓練
for epoch in range(10):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for inputs, labels in trainloader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
    train_losses.append(running_loss / len(trainloader))
    train_accuracies.append(100 * correct / total)

    model.eval()
    val_loss = 0.0
    correct = 0
```

```
        total = 0
        with torch.no_grad():
            for inputs, labels in testloader:
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
        val_losses.append(val_loss / len(testloader))
        val_accuracies.append(100 * correct / total)

    # 繪圖
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(train_losses, label='Train Loss')
    plt.plot(val_losses, label='Validation Loss')
    plt.title('Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(train_accuracies, label='Train Accuracy')
    plt.plot(val_accuracies, label='Validation Accuracy')
    plt.title('Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout()
```

```
plt.savefig("result.png")  # 存圖
plt.show()
```

2.2. 成果截圖



genai_lab01_image.png

2.3. 如何改良、改良後 loss 以及 accuracy 在折線圖上的變化以及改良的部分

> *NOTE：會作一些小整理*

2.3.1. Overfitting

定義：在原始模型架構中，雖然訓練集準確率快速上升且 loss 明顯下降，但在驗證集上卻出現

- Validation Loss 明顯高於 Training Loss
- Validation Accuracy 無法持續提升，甚至下降
  此現象可判斷為過擬合（Overfitting），即模型對訓練資料學習過度、記憶特定樣本特徵，導致在未知資料上無法良好泛化。

背後原因：

- 模型容量過大（過多參數）
- 缺乏正則化手段（如 Dropout、L2）
- 訓練樣本過於單一（未進行資料增強）
- 訓練迴圈次數與學習率控制不當

2.3.2. 優化策略 01：Data Augmentation

目的： 提高資料多樣性，減少模型對特定樣本的依賴。
方法： 對 CIFAR-10 訓練圖像進行 `RandomHorizontalFlip` 及 `RandomCrop` 。

```
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    ...
])
```

2.3.3. 優化策略 02：Dropout

目的： 隨機丟棄部分神經元，防止神經元之間過度依賴，避免 co-adaptation。
位置： 應用於全連接層之後，設為 `Dropout(0.5)` 。

```
self.dropout = nn.Dropout(0.5)
```

2.3.4. 優化策略 03：模型簡化（降低參數量）

目的： 減少模型容量，降低過度擬合風險。
方式： 原始模型中全連接層 `fc1` 使用 256 個神經元，改為 128。

```
self.fc1 = nn.Linear(64 * 8 * 8, 128)  # 原為 256
```

2.3.5. 訓練結果與圖像分析

指標	原始模型	優化後模型	說明
訓練 Loss	持續下降	同樣下降	模型有效學習特徵
驗證 Loss	停滯甚至上升	隨訓練下降	Overfitting 減輕
訓練 Accuracy	高	略下降但穩定	表示未過度記憶訓練樣本
驗證 Accuracy	明顯偏低	與訓練 accuracy 差距縮小	泛化性能提升