LAB5 Report

(一) 模型比較和分析

```
Teacher from scratch: loss-0.4908088000514839, accuracy-86.14
Student from scratch: loss-0.4700029639503624, accuracy-86.47
Response-based student: loss-0.6681125254570683, accuracy-87.55
Filter-baswd student: loss-2.0340768264818796, accuracy-86.9600000000
```

1. teacher 與 student 基準表現:
    - 學生模型的表現稍好於教師模型（accurancy 稍大一些），顯示學生模型能有效利用 CIFAR-10 數據進行學習。
    - 這表明學生模型（ResNet18）即使參數量較少，依然具有良好的學習能力。
2. Response-based 的知識蒸餾:
    - 此方法顯著提升學生模型的準確率（87.55%），說明教師的軟目標能有效幫助學生學習。
    - 雖然損失值高於基準學生模型，但這可能與知識蒸餾中的 KL 散度計算有關，實際分類性能更具參考價值。
3. Feature-based 的知識蒸餾:
    - 雖然準確率達到 86.96%，但損失值（2.0341）偏高。
    - 這可能與特徵對齊或損失函數的設計有關，需進一步優化。

（二） Response-Based KD 參數選擇

```python
def loss_re(student_logits, teacher_logits, labels, T=4.0, alpha=0.7):
```

1. 溫度 (T)

最終選擇 T = 4.0

在實驗中，嘗試了 T=1.0,4.0,8.0 的不同溫度值。結果顯示 T=4.0 提供了最佳的準確率和平衡效果，既能讓學生模型模仿教師的知識，又避免因概率過於平滑而丟失關鍵信息。

2. 平衡係數 Alpha

最終選擇 Alpha =0.7

實驗測試 Alpha = 0.3, 0.7 兩種參數，發現 Alpha = 0.7 最適合。此值讓學生模型能充分利用教師的知識，同時保留對數據標籤的直接學習能力。

（三） feature-based KD 的實作

（以下包含選了哪些 layer 的 feature 及損失函數設計）

在 Feature-Based 知識蒸餾中，我選擇了 ResNet 結構中的 Layer 1 (feature1) 、Layer 2 (feature2)、 Layer 3 (feature3) 、Layer 4 (feature4)幾層特徵進行蒸餾，選擇這些層是為了確保學生模型從低階到高階，能全面學習教師模型的多層次語義表徵，以下是實作過程：

步驟 1: 獲取教師與學生模型的多層特徵

```
# ResNet 殘差塊
feature1 = self.layer1(x)
feature2 = self.layer2(feature1)
feature3 = self.layer3(feature2)
feature4 = self.layer4(feature3)

# 平均池化 + 全連接層
out = self.avgpool(feature4)
out = torch.flatten(out, 1)
out = self.fc(out)

# 返回分類輸出與隱藏特徵
return out, [feature1, feature2, feature3, feature4]
```

教師與學生的輸出：在進行特徵蒸餾時，教師和學生模型的 forward 方法會返回類似的特徵列表：

- 教師模型：teacher_features = [feature1, feature2, feature3, feature4]
- 學生模型：student_features = [feature1, feature2, feature3, feature4]

步驟 2: 特徵對齊 (Feature Alignment)

由於教師與學生模型的特徵圖大小可能不同，需要對學生模型的特徵進行對齊。我使用 F.adaptive_avg_pool2d 方法將學生的特徵調整為與教師特徵相同的大小

```
# 確保特徵大小一致
sf = F.adaptive_avg_pool2d(sf, tf.shape[2:])
```

步驟 3: 計算特徵損失

逐層計算學生與教師特徵之間的 L2 損失，累加為總的特徵損失

```
for sf, tf in zip(student_features, teacher_features):
    feature_loss += F.mse_loss(sf, tf)
```

步驟 4: 加入分類損失

除了特徵損失，還需加入分類損失，確保學生模型能正確分類數據

```
classification_loss = F.cross_entropy(student_logits, labels)
```

步驟 5: 加權組合損失

特徵損失與分類損失通過加權組合形成總損失，實驗中，我使用 alpha = 1.0 和 beta = 0.5 作為加權係數

```
total_loss = alpha * feature_loss + beta * classification_loss
```

整體 Feature-Based KD 的損失函數設計公式

$$Loss_{feature} = \alpha \cdot \sum_{i=1}^{n} MSE(f_s^i, f_t^i) + \beta \cdot CE(ys, y)$$

其中f_s^i, f_t^i分別為學生和教師第 i 層的特徵； α 和 β 控制特徵損失與分類損失的比重。

```
def loss_fe(student_features, teacher_features, student_logits, labels, alpha=1.0, beta=0.5):
    """
    計算基於特徵的蒸餾損失（包含分類損失）。
    """
    feature_loss = 0.0
    for sf, tf in zip(student_features, teacher_features):
        # 確保特徵大小一致
        sf = F.adaptive_avg_pool2d(sf, tf.shape[2:])
        feature_loss += F.mse_loss(sf, tf)  # 使用 L2 損失

    # 加入分類損失
    classification_loss = F.cross_entropy(student_logits, labels)

    # 加權後返回總損失
    total_loss = alpha * feature_loss + beta * classification_loss
    return total_loss
```

（四）Response-Base 的損失函數設計

Response-Based 蒸餾的損失函數由兩部分組成

1. 分類損失 (Classification Loss):

   衡量學生模型對真實標籤的預測準確性

$$CE\ Loss = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(\hat{y}_i)$$

   其中 y_i是真實標籤的分佈（通常為 one-hot 編碼）；$\hat{y}_i$是學生模型的預測概率。

2. 蒸餾損失 (Knowledge Distillation Loss):

   衡量學生模型輸出分佈與教師模型輸出分佈的差異，、使用 KL 散度作為損失函數。

$$KL\ Loss = T^2 \cdot KL(softmax(\frac{y_t}{T}), softmax(\frac{y_s}{T}))$$

   其中 T 是蒸餾溫度；y_t 和 y_s 分別是教師和學生模型的 logits 輸出。

最後再將以上兩部分進行加權組和：

$$Total\ Loss = \alpha \cdot KL\ Loss + (1-\alpha) \cdot CE\ Loss$$

```python
def loss_re(student_logits, teacher_logits, labels, T=4.0, alpha=0.7):
    # CrossEntropy 損失（標準分類損失）
    ce_loss = F.cross_entropy(student_logits, labels)

    # KL 散度損失（蒸餾損失）
    student_softmax = F.log_softmax(student_logits / T, dim=1)
    teacher_softmax = F.softmax(teacher_logits / T, dim=1)
    kd_loss = F.kl_div(student_softmax, teacher_softmax, reduction='batchmean') * (T * T)

    # 總損失
    loss = alpha * kd_loss + (1 - alpha) * ce_loss

    return loss
```

（五）實作過程中遇到的困難及你後來是如何解決的

在實作 Feature-Based 知識蒸餾的過程中，我一開始信心滿滿，以為只要按照既定流程寫完損失函數，學生模型的準確率自然會穩步提升。然而實際情況卻完全不是這麼回事。即使損失值看似在下降，但模型的準確率卻一直不見起色。這個現象一度讓我感到迷惑，也促使我重新檢查整個實作的每一個細節。

首先，我發現自己在損失函數的設計上忽略了一個重要問題：沒有引入分類損失，只有特徵損失。這導致學生模型過於專注於模仿教師的特徵，而缺乏對數據真實標籤的學習能力。結果，模型的準確率雖然表面上不低，但這只是因為特徵損失壓低了總損失，並沒有真正提升模型的分類性能。於是，我將分類損失加了進去，讓損失函數同時考慮特徵蒸餾和分類準確率。調整後，學生模型的表現終於開始與教師模型接近。

在這個基礎上，我又發現特徵對齊的方式存在問題。一開始我使用的是 F.adaptive_avg_pool2d 來對齊教師和學生的特徵圖，雖然這種方法簡單直接，但對於一些低階特徵，比如 Layer 1 和 Layer 2，它會丟失部分細節。於是，我改用了插值法 (F.interpolate) 來對齊特徵，尤其在高階特徵的對齊上，插值法的效果更好。這個小改動讓模型的損失下降速度更穩定，也讓蒸餾過程更加可靠。

接下來，我又發現了另一個問題：初期使用的均方誤差（MSE）損失對結果的影響並不理想。MSE 對於極端值過於敏感，經常會放大某些特徵差異，導致模型收斂過程中出現波動。經過反覆實驗，我將 MSE 改成了 L2 損失，用歐幾里得距離來衡量教師與學生特徵的差異。這個改動對穩定模型訓練起到了明顯作用，並讓最終的準確率提高了大約 1%。

經過這些改進，最終的 Feature-Based 知識蒸餾結果才達到了預期。準確率從一開始的停滯不前，提升到了穩定的 86.96%，幾乎與教師模型持平。我從這次實作中學到了很多，特別是每個細節的設計和調整都可能對結果產生深遠影響，哪怕是一個小小的對齊方式或損失函數的選擇，都可能決定整個模型的最終表現。