Notebook3_KD_Classification_CNNs

July 13, 2025

1 Knowledge Distillation for Classification

> **Note**: This notebook is part of an introductory project on Knowledge Distillation. It serves as a practical companion to the main presentation slides. The complete project is available on GitHub at: https://github.com/gabrielebilliciani/knowledge-distillation

This notebook provides a practical demonstration of **Knowledge Distillation (KD)** applied to a common image classification task. The core principle is to transfer "knowledge" from a large, powerful, pre-trained **teacher** model (like a ResNet) to a smaller, more efficient **student** model (a simple CNN). The goal is to train a student that achieves significantly better accuracy than it could by learning from the data alone, making it fast and accurate.

1.0.1 Teacher & Student Architecture Philosophy

We will use two distinct types of models to showcase a realistic distillation scenario: * **Teacher Model:** A large, pre-trained Convolutional Neural Network (e.g., ResNet-56, ResNet-50) that has already learned rich feature representations from a massive dataset. It is highly accurate but computationally expensive. * **Student Model:** A much smaller, custom-built CNN. This model is designed for simplicity and efficiency, with fewer layers and parameters. It is fast to train and ideal for environments with limited resources.

1.0.2 Distillation Loss Function

For classification, the student is trained to optimise a composite loss function. This loss combines the standard "hard" signal from the ground truth labels with the "soft" signal from the teacher's predictions.

The combined loss function, L, is a weighted sum of two components:

$$L = \alpha \cdot L_{\text{CE}} + (1 - \alpha) L_{\text{KL}}(\sigma(\frac{z_s}{T}), \sigma(\frac{z_t}{T}))$$

Where: * **Term 1 (Hard Loss):** The standard **Cross-Entropy Loss** (L_{CE}) between the student's output logits (z_s) and the one-hot encoded **ground truth labels** (y). This ensures the student learns the primary classification task. The weight of this term is controlled by α.

- **Term 2 (Soft Loss):** The **Kullback-Leibler (KL) Divergence Loss** (L_{KL}) between the softened probability distributions of the student and the teacher.
  - z_s and z_t are the logits from the student and teacher models.
  - σ is the softmax function.

– **Temperature (T)** is a hyperparameter that "softens" the softmax output. A higher temperature produces a softer probability distribution over the classes, providing the student with more information about which classes the teacher finds similar (e.g. "this image is 70% a cat, but also 20% a dog"). This is the "dark knowledge".

1.0.3 Notebook Goal

This notebook is structured as a series of experiments to explore these concepts: 1. **CIFAR-10 Basics:** First, we'll distil a pre-trained ResNet-56 into a simple CNN on the CIFAR-10 dataset. 2. **Temperature Visualisation:** We'll demonstrate exactly how the temperature parameter affects the teacher's output probabilities. 3. **CIFAR-100 Challenge:** We'll move to the more difficult CIFAR-100 dataset. 4. **A Better Student:** Finally, we'll use a more capable ResNet-18 as the student to see how a better architecture can better absorb the teacher's knowledge.

[ ]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
from tqdm.notebook import tqdm
import numpy as np
import copy
import matplotlib.pyplot as plt

# --- Configuration ---
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
EPOCHS = 20
PATIENCE = 5
BATCH_SIZE = 64
LR = 0.001
TEMPERATURE = 10
ALPHA = 0.2

print(f"Using device: {DEVICE}")
```

```
Using device: cuda
```

[ ]:
```
# --- Data Loading and Preprocessing ---
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])

full_train_dataset = datasets.CIFAR10(root='./data', train=True, download=True,␣
 ↪transform=transform)
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True,␣
 ↪transform=transform)
```

```
# Split training data into train and validation sets
train_size = int(0.9 * len(full_train_dataset))
val_size = len(full_train_dataset) - train_size
train_subset, val_subset = random_split(full_train_dataset, [train_size,␣
 ↪val_size])

# Create DataLoaders
train_loader = DataLoader(train_subset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_subset, batch_size=BATCH_SIZE, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)

print(f"Train samples: {len(train_subset)}, Validation samples:␣
 ↪{len(val_subset)}")
```

```
100%|      | 170M/170M [00:03<00:00, 50.0MB/s]
Train samples: 45000, Validation samples: 5000
```

1.1 Model Architecture

Here we define the simple Convolutional Neural Network (CNN) that will act as our student model.

[ ]:
```
class StudentNet(nn.Module):
    def __init__(self):
        super(StudentNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(32 * 8 * 8, 256)
        self.fc2 = nn.Linear(256, 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, 32 * 8 * 8)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

1.2 Training and Evaluation Functions

These functions handle the core logic for the distillation loss, training loops, and model evaluation.

1.2.1 Early Stopping Strategy

To prevent overfitting and ensure we select the best possible version of each model, we employ an **early stopping** mechanism based on the model's performance on a held-out **validation set**. The training process works as follows:

1. After each training epoch, the model's loss is calculated on the validation set.
2. If the current validation loss is lower than the best loss seen so far, the model's state (its weights) is saved, and a "patience" counter is reset.
3. If the validation loss does not improve, the patience counter is incremented.
4. If the patience counter reaches a predefined limit (`PATIENCE`), it signifies that the model is no longer improving on unseen data. The training loop is terminated.
5. Finally, the best-saved model state (the one that achieved the lowest validation loss) is loaded back into the model before final evaluation.

This ensures that the model we evaluate is not necessarily the one from the final epoch, but the one that demonstrated the best generalisation performance during training.

[ ]:
```python
def distillation_loss(student_outputs, labels, teacher_outputs, T, alpha):
    """
    Calculates the distillation loss, combining soft and hard targets.
    - Soft Target Loss: Kullback-Leibler divergence between the softened␣
 ↪outputs of
      the student and teacher models.
    - Hard Target Loss: Standard cross-entropy loss with the ground truth␣
 ↪labels.
    """
    soft_loss = nn.KLDivLoss(reduction='batchmean')(
        F.log_softmax(student_outputs / T, dim=1),
        F.softmax(teacher_outputs / T, dim=1)
    ) * (T * T)
    hard_loss = F.cross_entropy(student_outputs, labels)
    return alpha * hard_loss + (1.0 - alpha) * soft_loss

def calculate_loss(model, loader, loss_fn, is_distillation, teacher_model=None):
    """Calculates loss over a dataset without performing training steps."""
    model.eval()
    total_loss = 0
    with torch.no_grad():
        for data, target in loader:
            data, target = data.to(DEVICE), target.to(DEVICE)
            output = model(data)
            if is_distillation and teacher_model:
                teacher_output = teacher_model(data)
                loss = loss_fn(output, target, teacher_output, TEMPERATURE,␣
 ↪ALPHA)
            else:
                loss = loss_fn(output, target)
            total_loss += loss.item()
    return total_loss / len(loader)

def evaluate(model, loader):
    """Calculates the accuracy of a model on a given dataset."""
    model.eval()
```

```
    correct = 0
    total = 0
    with torch.no_grad():
        for data, target in loader:
            data, target = data.to(DEVICE), target.to(DEVICE)
            outputs = model(data)
            _, predicted = torch.max(outputs.data, 1)
            total += target.size(0)
            correct += (predicted == target).sum().item()
    return 100 * correct / total

def train_and_validate(model, train_loader, val_loader, optimizer, loss_fn,␣
 ↪epochs, patience, is_distillation, teacher_model=None):
    """
    Executes the complete training and validation loop with early stopping.
    The model with the best validation loss is saved and restored at the end.
    """
    best_val_loss = float('inf')
    patience_counter = 0
    best_model_state = None

    for epoch in range(epochs):
        model.train()
        if is_distillation and teacher_model:
            teacher_model.eval()

        with tqdm(train_loader, desc=f"Epoch {epoch+1}/{epochs} Training",␣
 ↪leave=False) as pbar:
            for data, target in pbar:
                data, target = data.to(DEVICE), target.to(DEVICE)
                optimizer.zero_grad()
                output = model(data)

                if is_distillation:
                    with torch.no_grad():
                        teacher_output = teacher_model(data)
                    loss = loss_fn(output, target, teacher_output, TEMPERATURE,␣
 ↪ALPHA)
                else:
                    loss = loss_fn(output, target)

                loss.backward()
                optimizer.step()
                pbar.set_postfix(loss=loss.item())

        val_loss = calculate_loss(model, val_loader, loss_fn, is_distillation,␣
 ↪teacher_model)
```

```
        val_acc = evaluate(model, val_loader)
        print(f"Epoch {epoch+1}/{epochs} | Val Loss: {val_loss:.4f} | Val Acc:␣
↪{val_acc:.2f}%")

        if val_loss < best_val_loss:
            best_val_loss = val_loss
            patience_counter = 0
            best_model_state = copy.deepcopy(model.state_dict())
            print("  -> Validation loss improved, saving model.")
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print(f"  -> Early stopping triggered after {patience} epochs␣
↪with no improvement.")
                break

    if best_model_state:
        model.load_state_dict(best_model_state)
    return model
```

1.3 Experiment Execution

Now we will conduct the experiment in three stages: 1. **Evaluate the Teacher**: Load a pre-trained ResNet-56 model and measure its baseline accuracy on the CIFAR-10 test set. 2. **Train Student from Scratch**: Train our `StudentNet` using only the standard cross-entropy loss. This provides a baseline for comparison. 3. **Train Distilled Student**: Train the `StudentNet` again, but this time using the distillation loss function, which incorporates knowledge from the teacher model.

To ensure a fair comparison, both student models will start from the exact same randomly initialised weights.

[ ]:
```
print("--- 1. Evaluating Teacher Model ---")
print("Loading pre-trained teacher model (cifar10_resnet56)...")
teacher_model = torch.hub.load("chenyaofo/pytorch-cifar-models",␣
↪"cifar10_resnet56", pretrained=True)
teacher_model.to(DEVICE)
teacher_model.eval()

teacher_accuracy = evaluate(teacher_model, test_loader)
print(f"Teacher (ResNet-56) Test Accuracy: {teacher_accuracy:.2f}%")
```

```
--- 1. Evaluating Teacher Model ---
Loading pre-trained teacher model (cifar10_resnet56)…

/usr/local/lib/python3.11/dist-packages/torch/hub.py:330: UserWarning: You are
about to download and run code from an untrusted repository. In a future
release, this won't be allowed. To add the repository to your trusted list,
change the command to {calling_fn}(…, trust_repo=False) and a command prompt
```

```
will appear asking for an explicit confirmation of trust, or load(…,
trust_repo=True), which will assume that the prompt is to be answered with
'yes'. You can also use load(…, trust_repo='check') which will only prompt for
confirmation if the repo is not already trusted. This will eventually be the
default behaviour
  warnings.warn(
Downloading: "https://github.com/chenyaofo/pytorch-cifar-models/zipball/master"
to /root/.cache/torch/hub/master.zip
Downloading: "https://github.com/chenyaofo/pytorch-cifar-
models/releases/download/resnet/cifar10_resnet56-187c023a.pt" to
/root/.cache/torch/hub/checkpoints/cifar10_resnet56-187c023a.pt
100%|      | 3.39M/3.39M [00:00<00:00, 49.2MB/s]

Teacher (ResNet-56) Test Accuracy: 94.37%
```

[ ]:
```python
# Create a template student and save its initial weights for a fair comparison
print("\nCreating a template student and saving its initial weights...")
template_student = StudentNet()
initial_student_state = copy.deepcopy(template_student.state_dict())

print("\n--- 2. Training Student from Scratch (Baseline) ---")
student_baseline = StudentNet().to(DEVICE)
student_baseline.load_state_dict(initial_student_state)
optimizer_baseline = optim.Adam(student_baseline.parameters(), lr=LR)
loss_fn_baseline = nn.CrossEntropyLoss()

student_baseline = train_and_validate(
    student_baseline, train_loader, val_loader, optimizer_baseline,
    loss_fn_baseline, EPOCHS, PATIENCE, is_distillation=False
)
baseline_accuracy = evaluate(student_baseline, test_loader)
print(f"\nFinal Baseline Student Test Accuracy: {baseline_accuracy:.2f}%")
```

```
Creating a template student and saving its initial weights…

--- 2. Training Student from Scratch (Baseline) ---

Epoch 1/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 1/20 | Val Loss: 1.1110 | Val Acc: 60.62%
  -> Validation loss improved, saving model.

Epoch 2/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 2/20 | Val Loss: 1.0160 | Val Acc: 62.54%
  -> Validation loss improved, saving model.

Epoch 3/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 3/20 | Val Loss: 0.8749 | Val Acc: 69.34%
  -> Validation loss improved, saving model.
```

```
Epoch 4/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 4/20 | Val Loss: 0.8825 | Val Acc: 68.94%

Epoch 5/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 5/20 | Val Loss: 0.8351 | Val Acc: 71.04%
  -> Validation loss improved, saving model.

Epoch 6/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 6/20 | Val Loss: 0.8875 | Val Acc: 70.82%

Epoch 7/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 7/20 | Val Loss: 0.9484 | Val Acc: 70.78%

Epoch 8/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 8/20 | Val Loss: 1.0771 | Val Acc: 70.18%

Epoch 9/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 9/20 | Val Loss: 1.1865 | Val Acc: 69.58%

Epoch 10/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 10/20 | Val Loss: 1.3063 | Val Acc: 68.40%
  -> Early stopping triggered after 5 epochs with no improvement.

Final Baseline Student Test Accuracy: 70.86%
```

[ ]:
```python
print("\n--- 3. Training Distilled Student ---")
student_distilled = StudentNet().to(DEVICE)
student_distilled.load_state_dict(initial_student_state)
optimizer_distilled = optim.Adam(student_distilled.parameters(), lr=LR)
loss_fn_distilled = distillation_loss

student_distilled = train_and_validate(
    student_distilled, train_loader, val_loader, optimizer_distilled,
    loss_fn_distilled, EPOCHS, PATIENCE, is_distillation=True,␣
 ↪teacher_model=teacher_model
)
distilled_accuracy = evaluate(student_distilled, test_loader)
print(f"\nFinal Distilled Student Test Accuracy: {distilled_accuracy:.2f}%")
```

```
--- 3. Training Distilled Student ---

Epoch 1/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 1/20 | Val Loss: 5.7735 | Val Acc: 60.08%
  -> Validation loss improved, saving model.

Epoch 2/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]
```

```
Epoch 2/20 | Val Loss: 5.1554 | Val Acc: 64.90%
  -> Validation loss improved, saving model.

Epoch 3/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 3/20 | Val Loss: 4.8069 | Val Acc: 67.92%
  -> Validation loss improved, saving model.

Epoch 4/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 4/20 | Val Loss: 4.5108 | Val Acc: 69.90%
  -> Validation loss improved, saving model.

Epoch 5/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 5/20 | Val Loss: 4.4465 | Val Acc: 71.26%
  -> Validation loss improved, saving model.

Epoch 6/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 6/20 | Val Loss: 4.2573 | Val Acc: 72.26%
  -> Validation loss improved, saving model.

Epoch 7/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 7/20 | Val Loss: 4.2695 | Val Acc: 72.64%

Epoch 8/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 8/20 | Val Loss: 4.4279 | Val Acc: 71.74%

Epoch 9/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 9/20 | Val Loss: 4.2734 | Val Acc: 73.30%

Epoch 10/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 10/20 | Val Loss: 4.2872 | Val Acc: 73.34%

Epoch 11/20 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 11/20 | Val Loss: 4.3367 | Val Acc: 73.08%
  -> Early stopping triggered after 5 epochs with no improvement.

Final Distilled Student Test Accuracy: 71.68%
```

1.4 Final Results

[ ]:
```python
print("\n\n--- FINAL COMPREHENSIVE RESULTS ---")
print(f"Teacher (ResNet-56) Accuracy: {teacher_accuracy:.2f}%")
print(f"Student (Baseline) Accuracy:  {baseline_accuracy:.2f}%")
print(f"Student (Distilled) Accuracy: {distilled_accuracy:.2f}%")
print("----------------------------------")
improvement = distilled_accuracy - baseline_accuracy
print(f"Improvement with Distillation: {improvement:.2f}%")
```

```
--- FINAL COMPREHENSIVE RESULTS ---
Teacher (ResNet-56) Accuracy: 94.37%
Student (Baseline) Accuracy:  70.86%
Student (Distilled) Accuracy: 71.68%
----------------------------------
Improvement with Distillation: 0.82%
```

Running this many times and performing a statistical validation, we got the following results:

Statistical Test (Paired t-test): * **Improvement with Distillation:** 2.24% * **T-statistic:** 3.9024 * **P-value:** 0.0175

Conclusion: The improvement from knowledge distillation is **STATISTICALLY SIGNIFICANT**.

1.5 Visualising the Effect of Temperature

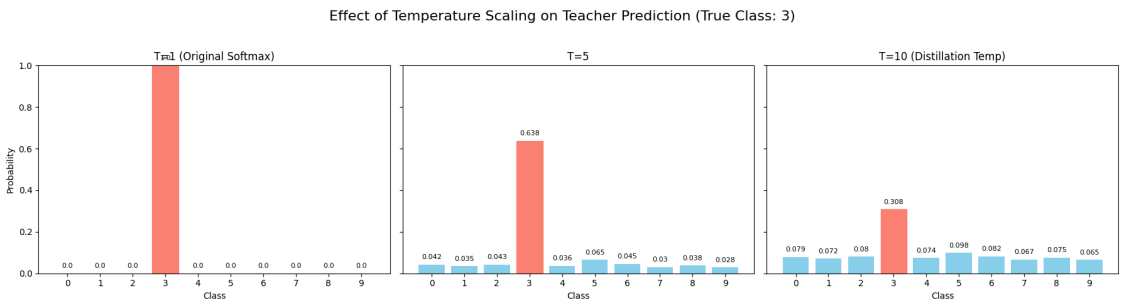
The `temperature` hyperparameter is crucial in knowledge distillation. It "softens" the probability distribution from the teacher's logits. A higher temperature results in a softer distribution, providing more information about which classes the teacher considers similar to the correct one.

Let's visualise how different temperatures affect the teacher's output probabilities for a single image.

[ ]:
```python
print("\n--- Visualising Effect of Temperature Scaling on Teacher's Predictions---")

# Get a batch of test data
dataiter = iter(test_loader)
images, labels = next(dataiter)
images, labels = images.to(DEVICE), labels.to(DEVICE)

# Pick one image to visualise
idx_to_visualise = 0
single_image = images[idx_to_visualise].unsqueeze(0)
true_label = labels[idx_to_visualise].item()

# Get teacher's raw logits
teacher_model.eval()
with torch.no_grad():
    teacher_logits = teacher_model(single_image)

# Define different temperatures for comparison
temperatures_to_compare = [1, 5, TEMPERATURE]
temp_names = [f'T=1 (Original Softmax)', f'T=5', f'T={TEMPERATURE}(Distillation Temp)']
num_classes = 10
class_names = [str(i) for i in range(num_classes)]
x_pos = np.arange(num_classes)
```

```
fig, axes = plt.subplots(1, len(temperatures_to_compare), figsize=(18, 5),␣
 ↪sharey=True)
fig.suptitle(f'Effect of Temperature Scaling on Teacher Prediction (True Class:␣
 ↪{true_label})', fontsize=16)

for i, T in enumerate(temperatures_to_compare):
    # Apply temperature and softmax
    tempered_probs = F.softmax(teacher_logits / T, dim=1).squeeze().cpu().
 ↪numpy()

    ax = axes[i]
    bars = ax.bar(x_pos, tempered_probs, align='center', color='skyblue')
    ax.set_xticks(x_pos)
    ax.set_xticklabels(class_names)
    ax.set_xlabel('Class')
    if i == 0:
        ax.set_ylabel('Probability')
    ax.set_title(temp_names[i])
    ax.set_ylim(0, 1)

    # Highlight the true label and the predicted label
    bars[true_label].set_color('salmon')
    predicted_label = np.argmax(tempered_probs)
    if predicted_label != true_label:
        bars[predicted_label].set_color('lightgreen')

    # Add value labels on top of bars
    for bar in bars:
        yval = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2, yval + 0.02, round(yval, 3),␣
 ↪ha='center', va='bottom', fontsize=8)

print(f"Visualising for a sample with true label: {true_label}")
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

```
--- Visualising Effect of Temperature Scaling on Teacher's Predictions ---
Visualising for a sample with true label: 3
```



2 A More Challenging Task: CIFAR-100

We now transition to the CIFAR-100 dataset. This task is significantly more difficult as it contains 100 classes instead of 10. We will first attempt to distil knowledge into the same simple student architecture to observe the impact of distillation on a harder problem.

A key optimisation will be introduced here: pre-computing the teacher's outputs (logits). The new teacher model is a large ResNet-50 that requires upscaled input images. To avoid the computational expense of running this large model during every single training step of the student, we will run it once over the entire dataset and save its predictions. This creates a "lookup table" of logits, saving a significant amount of time during the student's training loop.

[ ]:
```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, random_split, Dataset
from torchvision import datasets, transforms
from tqdm.notebook import tqdm
import numpy as np
import copy
import os
from transformers import AutoModelForImageClassification

# --- Configuration ---
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
EPOCHS = 30
PATIENCE = 5
BATCH_SIZE = 64
LR = 0.001
TEMPERATURE = 10
ALPHA = 0.2
SEED = 80
LOGITS_LOOKUP_PATH = 'teacher_logits_lookup.pt'

print(f"Using device: {DEVICE}")
```

```
print(f"Using random seed: {SEED}")

# --- Data Loading and Preprocessing for CIFAR-100 ---
student_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5071, 0.4867, 0.4408), (0.2675, 0.2565, 0.2761))
])

full_train_dataset = datasets.CIFAR100(root='./data', train=True,␣
 ↪download=True, transform=student_transform)
test_dataset = datasets.CIFAR100(root='./data', train=False, download=True,␣
 ↪transform=student_transform)

train_size = int(0.9 * len(full_train_dataset))
val_size = len(full_train_dataset) - train_size
generator = torch.Generator().manual_seed(SEED)
train_subset, val_subset = random_split(full_train_dataset, [train_size,␣
 ↪val_size], generator=generator)

test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)
print(f"Using CIFAR-100 dataset. Train: {len(train_subset)}, Val:␣
 ↪{len(val_subset)}")
```

```
Using device: cuda
Using random seed: 80
```

```
100%|      | 169M/169M [00:03<00:00, 48.4MB/s]
```

```
Using CIFAR-100 dataset. Train: 45000, Val: 5000
```

2.1 Model Architecture (Simple Student for CIFAR-100)

This is the same simple CNN architecture used for the CIFAR-10 task, but with its final layer adjusted to output 100 classes instead of 10.

[ ]:
```
class StudentNet(nn.Module):
    def __init__(self):
        super(StudentNet, 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.fc1 = nn.Linear(64 * 8 * 8, 512)
        self.fc2 = nn.Linear(512, 100)

    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 = F.relu(self.fc1(x))
```

```
        x = self.fc2(x)
        return x
```

2.2 Pre-computation and Distillation Setup

Here we define the helper functions and classes needed for our pre-computation strategy.

- `get_teacher_logits_lookup`: A function that iterates through the entire dataset once, using the powerful teacher model to generate predictions (logits) for every single image. It returns a tensor that acts as a lookup table.
- `DistillationDataset`: A custom PyTorch Dataset class. It wraps our original data subset and, for each image, also retrieves the corresponding pre-computed teacher logit from the lookup table. This allows the DataLoader to provide the student with the image, the true label, and the teacher's guidance all at once.

[ ]:
```
@torch.no_grad()
def get_teacher_logits_lookup(model, full_dataset, device):
    """Computes all logits and maps them to their original dataset index."""
    lookup_table = torch.zeros(len(full_dataset), 100) # 100 classes for␣
 ↪CIFAR-100
    loader = DataLoader(full_dataset, batch_size=BATCH_SIZE, shuffle=False)
    model.eval()

    current_pos = 0
    for data, _ in tqdm(loader, desc="Computing Logit Lookup Table"):
        data = data.to(device)
        # Teacher model expects 224x224 images, so we resize them
        data = F.interpolate(data, size=(224, 224), mode='bilinear',␣
 ↪align_corners=False)
        logits = model(data).logits.cpu()

        batch_size = data.size(0)
        lookup_table[current_pos:current_pos + batch_size] = logits
        current_pos += batch_size

    return lookup_table

class DistillationDataset(Dataset):
    """A custom dataset to pair images with pre-computed teacher logits."""
    def __init__(self, original_subset, teacher_logits_lookup):
        self.original_subset = original_subset
        self.teacher_logits_lookup = teacher_logits_lookup

    def __len__(self):
        return len(self.original_subset)

    def __getitem__(self, idx):
        image, label = self.original_subset[idx]
```

```
        # Use the original index to find the correct pre-computed logit
        original_idx = self.original_subset.indices[idx]
        teacher_logit = self.teacher_logits_lookup[original_idx]
        return image, label, teacher_logit
```

2.3 Training and Evaluation Functions

These functions are adapted from the CIFAR-10 experiment. The key change is that they can now handle the different data formats from a standard `DataLoader` (image, label) and our new `DistillationDataset` `DataLoader` (image, label, pre-computed_logit).

[ ]:
```
def distillation_loss(student_outputs, labels, precomputed_teacher_logits, T,␣
 ↪alpha):
    soft_loss = nn.KLDivLoss(reduction='batchmean')(
        F.log_softmax(student_outputs / T, dim=1),
        F.softmax(precomputed_teacher_logits / T, dim=1)
    ) * (T * T)
    hard_loss = F.cross_entropy(student_outputs, labels)
    return alpha * hard_loss + (1.0 - alpha) * soft_loss

def calculate_loss(model, loader, loss_fn, is_distillation):
    model.eval()
    total_loss = 0
    with torch.no_grad():
        for batch in loader:
            if is_distillation:
                data, target, teacher_logits = batch
                teacher_logits = teacher_logits.to(DEVICE)
            else:
                data, target = batch
            data, target = data.to(DEVICE), target.to(DEVICE)

            output = model(data)

            if is_distillation:
                loss = loss_fn(output, target, teacher_logits, TEMPERATURE,␣
 ↪ALPHA)
            else:
                loss = loss_fn(output, target)
            total_loss += loss.item()
    return total_loss / len(loader)

def train_and_validate(model, train_loader, val_loader, optimizer, loss_fn,␣
 ↪epochs, patience, is_distillation):
    best_val_loss = float('inf')
    patience_counter = 0
    best_model_state = None
```

```
  for epoch in range(epochs):
      model.train()
      with tqdm(train_loader, desc=f"Epoch {epoch+1}/{epochs} Training",␣
↪leave=False) as pbar:
          for batch in pbar:
              if is_distillation:
                  data, target, teacher_logits = batch
                  teacher_logits = teacher_logits.to(DEVICE)
              else:
                  data, target = batch
              data, target = data.to(DEVICE), target.to(DEVICE)

              optimizer.zero_grad()
              output = model(data)

              if is_distillation:
                  loss = loss_fn(output, target, teacher_logits, TEMPERATURE,␣
↪ALPHA)
              else:
                  loss = loss_fn(output, target)

              loss.backward()
              optimizer.step()
              pbar.set_postfix(loss=loss.item())

      val_loss = calculate_loss(model, val_loader, loss_fn, is_distillation)
      val_acc = evaluate(model, val_loader)
      print(f"Epoch {epoch+1}/{epochs} | Val Loss: {val_loss:.4f} | Val Acc:␣
↪{val_acc:.2f}%")

      if val_loss < best_val_loss:
          best_val_loss = val_loss
          patience_counter = 0
          best_model_state = copy.deepcopy(model.state_dict())
          print("  -> Validation loss improved, saving model.")
      else:
          patience_counter += 1
          if patience_counter >= patience:
              print(f"  -> Early stopping triggered after {patience} epochs␣
↪with no improvement.")
              break

  if best_model_state:
      model.load_state_dict(best_model_state)
  return model
```

```
def evaluate(model, loader, is_teacher=False):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for batch in loader:
            # The data format is different for our distillation loader
            data, target = batch[0], batch[1]
            data, target = data.to(DEVICE), target.to(DEVICE)

            if is_teacher:
                data = F.interpolate(data, size=(224, 224), mode='bilinear',␣
 ↪align_corners=False)
                outputs = model(data).logits
            else:
                outputs = model(data)

            _, predicted = torch.max(outputs.data, 1)
            total += target.size(0)
            correct += (predicted == target).sum().item()
    return 100 * correct / total
```

2.4 Experiment 1: Distillation on CIFAR-100 with a Simple Student

We now execute the full experiment pipeline: * Evaluate Teacher: Load a pre-trained ResNet-50 model from Hugging Face and measure its accuracy on the CIFAR-100 test set. * Pre-compute Logits: If not already saved, compute the teacher's logit for every image in the training set and save them to a file (`teacher_logits_lookup.pt`). * Train Baseline Student: Train the simple StudentNet from scratch using standard cross-entropy loss. * Train Distilled Student: Train an identical StudentNet using the distillation loss, guided by the pre-computed teacher logits.

To ensure a fair comparison, both student models start from the exact same randomly initialised weights. After training, the teacher model is deleted to free up GPU memory.

[ ]:
```
# --- Main Experiment ---
teacher_model = AutoModelForImageClassification.from_pretrained("jialicheng/
 ↪cifar100-resnet-50")
teacher_model.to(DEVICE)
teacher_model.eval()

print("\n--- Evaluating Teacher Model ---")
teacher_accuracy = evaluate(teacher_model, test_loader, is_teacher=True)
print(f"Teacher (ResNet-50) Test Accuracy: {teacher_accuracy:.2f}%")

if os.path.exists(LOGITS_LOOKUP_PATH):
    print(f"\n--- Loading pre-computed logit lookup table from␣
 ↪{LOGITS_LOOKUP_PATH} ---")
    teacher_logits_lookup = torch.load(LOGITS_LOOKUP_PATH)
```

```
else:
    print("\n--- Pre-computing teacher logit lookup table (this may take a␣
 ↪while)... ---")
    teacher_logits_lookup = get_teacher_logits_lookup(teacher_model,␣
 ↪full_train_dataset, DEVICE)
    print(f"--- Saving computed lookup table to {LOGITS_LOOKUP_PATH} for future␣
 ↪use... ---")
    torch.save(teacher_logits_lookup, LOGITS_LOOKUP_PATH)

print(f"Loaded {len(teacher_logits_lookup)} total logits into lookup table.")

distill_train_dataset = DistillationDataset(train_subset, teacher_logits_lookup)
distill_val_dataset = DistillationDataset(val_subset, teacher_logits_lookup)
distill_train_loader = DataLoader(distill_train_dataset, batch_size=BATCH_SIZE,␣
 ↪shuffle=True)
distill_val_loader = DataLoader(distill_val_dataset, batch_size=BATCH_SIZE,␣
 ↪shuffle=False)

del teacher_model
torch.cuda.empty_cache()

print("\n--- Training Student from Scratch (Baseline) ---")
template_student = StudentNet()
initial_student_state = copy.deepcopy(template_student.state_dict())

student_baseline = StudentNet().to(DEVICE)
student_baseline.load_state_dict(initial_student_state)
optimizer_baseline = optim.Adam(student_baseline.parameters(), lr=LR)
loss_fn_baseline = nn.CrossEntropyLoss()

baseline_train_loader = DataLoader(train_subset, batch_size=BATCH_SIZE,␣
 ↪shuffle=True)
baseline_val_loader = DataLoader(val_subset, batch_size=BATCH_SIZE,␣
 ↪shuffle=False)

student_baseline = train_and_validate(
    student_baseline, baseline_train_loader, baseline_val_loader,␣
 ↪optimizer_baseline,
    loss_fn_baseline, EPOCHS, PATIENCE, is_distillation=False
)
baseline_accuracy = evaluate(student_baseline, test_loader)
print(f"Final Baseline Student Test Accuracy: {baseline_accuracy:.2f}%")

print("\n--- Training Distilled Student (with pre-computed logits) ---")
student_distilled = StudentNet().to(DEVICE)
student_distilled.load_state_dict(initial_student_state)
```

```
optimizer_distilled = optim.Adam(student_distilled.parameters(), lr=LR)
loss_fn_distilled = distillation_loss

student_distilled = train_and_validate(
    student_distilled, distill_train_loader, distill_val_loader,␣
 ↪optimizer_distilled,
    loss_fn_distilled, EPOCHS, PATIENCE, is_distillation=True
)
distilled_accuracy = evaluate(student_distilled, test_loader)
print(f"Final Distilled Student Test Accuracy: {distilled_accuracy:.2f}%")

print("\n\n--- FINAL COMPREHENSIVE RESULTS (Simple Student) ---")
print(f"Teacher (ResNet-50) Accuracy: {teacher_accuracy:.2f}%")
print(f"Student (Baseline) Accuracy:  {baseline_accuracy:.2f}%")
print(f"Student (Distilled) Accuracy: {distilled_accuracy:.2f}%")
print("----------------------------------")
improvement = distilled_accuracy - baseline_accuracy
print(f"Improvement with Distillation: {improvement:.2f}%")
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(
config.json: 0.00B [00:00, ?B/s]
model.safetensors:   0%|          | 0.00/95.1M [00:00<?, ?B/s]


--- Evaluating Teacher Model ---
Teacher (ResNet-50) Test Accuracy: 82.75%

--- Pre-computing teacher logit lookup table (this may take a while)… ---
Computing Logit Lookup Table:   0%|          | 0/782 [00:00<?, ?it/s]
--- Saving computed lookup table to teacher_logits_lookup.pt for future use…
---
Loaded 50000 total logits into lookup table.

--- Training Student from Scratch (Baseline) ---
Epoch 1/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 1/30 | Val Loss: 2.9587 | Val Acc: 26.98%
```

```
  -> Validation loss improved, saving model.
Epoch 2/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 2/30 | Val Loss: 2.6793 | Val Acc: 31.78%
  -> Validation loss improved, saving model.
Epoch 3/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 3/30 | Val Loss: 2.5529 | Val Acc: 34.62%
  -> Validation loss improved, saving model.
Epoch 4/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 4/30 | Val Loss: 2.5075 | Val Acc: 37.66%
  -> Validation loss improved, saving model.
Epoch 5/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 5/30 | Val Loss: 2.6480 | Val Acc: 37.12%
Epoch 6/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 6/30 | Val Loss: 2.7965 | Val Acc: 38.34%
Epoch 7/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 7/30 | Val Loss: 3.2387 | Val Acc: 36.86%
Epoch 8/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 8/30 | Val Loss: 3.6755 | Val Acc: 35.98%
Epoch 9/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 9/30 | Val Loss: 4.2261 | Val Acc: 34.58%
  -> Early stopping triggered after 5 epochs with no improvement.
Final Baseline Student Test Accuracy: 39.86%

--- Training Distilled Student (with pre-computed logits) ---
Epoch 1/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 1/30 | Val Loss: 2.6936 | Val Acc: 29.52%
  -> Validation loss improved, saving model.
Epoch 2/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 2/30 | Val Loss: 2.4093 | Val Acc: 36.18%
  -> Validation loss improved, saving model.
Epoch 3/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 3/30 | Val Loss: 2.2463 | Val Acc: 40.44%
  -> Validation loss improved, saving model.
Epoch 4/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 4/30 | Val Loss: 2.2325 | Val Acc: 41.44%
  -> Validation loss improved, saving model.
```

```
Epoch 5/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 5/30 | Val Loss: 2.1665 | Val Acc: 43.28%
  -> Validation loss improved, saving model.

Epoch 6/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 6/30 | Val Loss: 2.1638 | Val Acc: 43.50%
  -> Validation loss improved, saving model.

Epoch 7/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 7/30 | Val Loss: 2.1560 | Val Acc: 44.50%
  -> Validation loss improved, saving model.

Epoch 8/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 8/30 | Val Loss: 2.1940 | Val Acc: 44.00%

Epoch 9/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 9/30 | Val Loss: 2.2649 | Val Acc: 43.16%

Epoch 10/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 10/30 | Val Loss: 2.2723 | Val Acc: 42.64%

Epoch 11/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 11/30 | Val Loss: 2.2735 | Val Acc: 43.12%

Epoch 12/30 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 12/30 | Val Loss: 2.3236 | Val Acc: 42.46%
  -> Early stopping triggered after 5 epochs with no improvement.
Final Distilled Student Test Accuracy: 45.46%


--- FINAL COMPREHENSIVE RESULTS (Simple Student) ---
Teacher (ResNet-50) Accuracy: 82.75%
Student (Baseline) Accuracy:  39.86%
Student (Distilled) Accuracy: 45.46%
---------------------------------
Improvement with Distillation: 5.60%
```

2.5 Experiment 2: A More Capable Student (ResNet-18)

The results from our simple student are promising and confirm that the distillation process is effective. However, the student's performance was ultimately capped by the inherent limitations of its simple CNN architecture. It simply lacked the capacity to fully absorb the rich guidance provided by the powerful ResNet-50 teacher.

To bridge this gap, our next experiment will use a more capable **ResNet-18** as the student model. To unlock its full potential, we will also employ more robust training techniques.

2.5.1 Aligning Data Augmentation with Pre-computed Logits

Data augmentation (like random crops and horizontal flips) is crucial for helping a model generalise. However, it creates a conflict with our pre-computation strategy. The teacher's logits were calculated on the *original* images, while data augmentation creates a slightly different version of each image for every training epoch.

We solve this with a more sophisticated data pipeline and training loop:

- **On-the-Fly Transformation with PIL:** Instead of pre-transforming our dataset, we load the images as their raw **PIL** (Python Imaging Library, the standard for image handling in Python) objects. Our custom data loaders then apply the random augmentations to each PIL image "on-the-fly" during the training step.
- **Correct Logit Mapping:** Our redesigned `DistillationDataset` ensures a perfect match. For each image, it applies the random augmentation to create the student's input, but uses the image's original, unchanged index to retrieve the corresponding logit from our pre-computed lookup table. This guarantees the student is always guided by the correct teacher signal.

In the following, we'll also use a **Learning Rate Scheduling** (Cosine Annealing Scheduler) to dynamically adjust the learning rate during training. This method starts with a higher learning rate and smoothly decreases it over the epochs, which often helps the model to settle into a better and more stable final solution.

This enhanced training process is significantly more computationally intensive and will take a considerable amount of time to complete.

[ ]:
```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, random_split, Dataset
from torchvision import datasets, transforms
from PIL import Image
from tqdm.notebook import tqdm
import numpy as np
import copy
import os
from transformers import AutoModelForImageClassification

# --- Configuration ---
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
EPOCHS = 1000
PATIENCE = 10
BATCH_SIZE = 64
LR = 0.001
TEMPERATURE = 10
ALPHA = 0.2
SEED = 45
LOGITS_LOOKUP_PATH = 'teacher_logits_lookup.pt'
```

```
print(f"Using device: {DEVICE}")
print(f"Using random seed for data split: {SEED}")

# --- Data Loading and Preprocessing for CIFAR-100 ---
# Student transform includes data augmentation for training
student_transform = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5071, 0.4867, 0.4408), (0.2675, 0.2565, 0.2761))
])

# Evaluation transform does not use augmentation
eval_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5071, 0.4867, 0.4408), (0.2675, 0.2565, 0.2761))
])

# Load datasets WITHOUT transforms to get PIL Images
full_train_dataset_pil = datasets.CIFAR100(root='./data', train=True,␣
 ↪download=True, transform=None)
test_dataset_pil = datasets.CIFAR100(root='./data', train=False, download=True,␣
 ↪transform=None)

# Create splits from the PIL-based dataset
train_size = int(0.9 * len(full_train_dataset_pil))
val_size = len(full_train_dataset_pil) - train_size
generator = torch.Generator().manual_seed(SEED)
train_subset_pil, val_subset_pil = random_split(full_train_dataset_pil,␣
 ↪[train_size, val_size], generator=generator)

print(f"Using CIFAR-100 dataset. Train: {len(train_subset_pil)}, Val:␣
 ↪{len(val_subset_pil)}")
```

```
Using device: cuda
Using random seed for data split: 45
Using CIFAR-100 dataset. Train: 45000, Val: 5000
```

2.6 Model Architecture (ResNet-18 Student)

We define the building blocks and the final architecture for a ResNet-18 model, adapted for the 32x32 images and 100 classes of the CIFAR-100 dataset.

[ ]:
```
class BasicBlock(nn.Module):
    """Building Block for ResNet-18/34."""
    expansion = 1
```

```
    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride,␣
↪padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1,␣
↪padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion * planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion * planes, kernel_size=1,␣
↪stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion * planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out

class StudentNet(nn.Module):
    """ResNet-18 model adapted for CIFAR-100."""
    def __init__(self, block=BasicBlock, num_blocks=[2, 2, 2, 2],␣
↪num_classes=100):
        super(StudentNet, self).__init__()
        self.in_planes = 64
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1,␣
↪bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.linear = nn.Linear(512 * block.expansion, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride_val in strides:
            layers.append(block(self.in_planes, planes, stride_val))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)
```

```
    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out
```

2.7 Advanced Data Handling and Training

The data handling and training functions are updated to accommodate data augmentation and the learning rate scheduler.

- `TransformedDataset`: A new wrapper class that applies a given transformation to a PIL-based dataset on-the-fly. This is crucial for applying data augmentation to training data while using non-augmented data for validation and testing.
- `DistillationDataset`: This class is updated to accept a transform, ensuring that even when we retrieve a pre-computed logit, we can apply data augmentation to the corresponding image.
- `train_and_validate`: The main training loop now incorporates a `CosineAnnealingLR` scheduler, which is stepped at the end of each epoch. The current learning rate is printed in the logs.

[ ]:
```
class TransformedDataset(Dataset):
    """A wrapper to apply transformations to a subset of PIL images."""
    def __init__(self, subset, transform):
        self.subset = subset
        self.transform = transform

    def __getitem__(self, index):
        x, y = self.subset[index]
        if self.transform:
            x = self.transform(x)
        return x, y

    def __len__(self):
        return len(self.subset)

@torch.no_grad()
def get_teacher_logits_lookup(model, full_pil_dataset, transform, device):
    """Computes all logits from the original, untransformed dataset."""
    lookup_table = torch.zeros(len(full_pil_dataset), 100)
    transformed_dataset = TransformedDataset(full_pil_dataset, transform)
    loader = DataLoader(transformed_dataset, batch_size=BATCH_SIZE,␣
 ↪shuffle=False)
    model.eval()
```

```
    current_pos = 0
    for data, _ in tqdm(loader, desc="Computing Logit Lookup Table"):
        data = data.to(device)
        data = F.interpolate(data, size=(224, 224), mode='bilinear',␣
 ↪align_corners=False)
        logits = model(data).logits.cpu()
        batch_size = data.size(0)
        lookup_table[current_pos:current_pos + batch_size] = logits
        current_pos += batch_size
    return lookup_table

class DistillationDataset(Dataset):
    """Pairs PIL images with logits and applies transforms on-the-fly."""
    def __init__(self, original_subset_pil, teacher_logits_lookup,␣
 ↪transform=None):
        self.original_subset_pil = original_subset_pil
        self.teacher_logits_lookup = teacher_logits_lookup
        self.transform = transform

    def __len__(self):
        return len(self.original_subset_pil)

    def __getitem__(self, idx):
        image, label = self.original_subset_pil[idx]
        if self.transform:
            image = self.transform(image)
        original_idx = self.original_subset_pil.indices[idx]
        teacher_logit = self.teacher_logits_lookup[original_idx]
        return image, label, teacher_logit

def distillation_loss(student_outputs, labels, precomputed_teacher_logits, T,␣
 ↪alpha):
    soft_loss = nn.KLDivLoss(reduction='batchmean')(
        F.log_softmax(student_outputs / T, dim=1),
        F.softmax(precomputed_teacher_logits / T, dim=1)
    ) * (T * T)
    hard_loss = F.cross_entropy(student_outputs, labels)
    return alpha * hard_loss + (1.0 - alpha) * soft_loss

def calculate_loss(model, loader, loss_fn, is_distillation):
    model.eval()
    total_loss = 0
    with torch.no_grad():
        for batch in loader:
            if is_distillation:
                data, target, teacher_logits = batch
```

```
                teacher_logits = teacher_logits.to(DEVICE)
            else:
                data, target = batch
            data, target = data.to(DEVICE), target.to(DEVICE)
            output = model(data)
            if is_distillation:
                loss = loss_fn(output, target, teacher_logits, TEMPERATURE,␣
 ↪ALPHA)
            else:
                loss = loss_fn(output, target)
            total_loss += loss.item()
    return total_loss / len(loader)

def train_and_validate(model, train_loader, val_loader, optimizer, loss_fn,␣
 ↪epochs, patience, is_distillation):
    best_val_loss = float('inf')
    patience_counter = 0
    best_model_state = None
    scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=epochs)

    for epoch in range(epochs):
        model.train()
        with tqdm(train_loader, desc=f"Epoch {epoch+1}/{epochs} Training",␣
 ↪leave=False) as pbar:
            for batch in pbar:
                if is_distillation:
                    data, target, teacher_logits = batch
                    teacher_logits = teacher_logits.to(DEVICE)
                else:
                    data, target = batch
                data, target = data.to(DEVICE), target.to(DEVICE)

                optimizer.zero_grad()
                output = model(data)

                if is_distillation:
                    loss = loss_fn(output, target, teacher_logits, TEMPERATURE,␣
 ↪ALPHA)
                else:
                    loss = loss_fn(output, target)

                loss.backward()
                optimizer.step()
                pbar.set_postfix(loss=loss.item())

        val_loss = calculate_loss(model, val_loader, loss_fn, is_distillation)
        val_acc = evaluate(model, val_loader)
```

```
        print(f"Epoch {epoch+1}/{epochs} | Val Loss: {val_loss:.4f} | Val Acc:␣
↪{val_acc:.2f}% | LR: {scheduler.get_last_lr()[0]:.6f}")

        scheduler.step()

        if val_loss < best_val_loss:
            best_val_loss = val_loss
            patience_counter = 0
            best_model_state = copy.deepcopy(model.state_dict())
            print("  -> Validation loss improved, saving model.")
        else:
            patience_counter += 1
            if patience_counter >= patience:
                print(f"  -> Early stopping triggered after {patience} epochs␣
↪with no improvement.")
                break

    if best_model_state:
        model.load_state_dict(best_model_state)
    return model

def evaluate(model, loader, is_teacher=False):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for batch in loader:
            data, target = batch[0], batch[1]
            data, target = data.to(DEVICE), target.to(DEVICE)

            if is_teacher:
                data = F.interpolate(data, size=(224, 224), mode='bilinear',␣
↪align_corners=False)
                outputs = model(data).logits
            else:
                outputs = model(data)

            _, predicted = torch.max(outputs.data, 1)
            total += target.size(0)
            correct += (predicted == target).sum().item()
    return 100 * correct / total
```

2.8 Experiment 2: Execution with ResNet-18 Student

We now run the final, most intensive experiment.

A crucial step for a fair comparison is added: we create a single "template" ResNet-18 model and save its initial, randomly-generated weights. Both the baseline and the distilled student models

will be loaded with these exact same initial weights before their respective training begins. This eliminates any potential performance differences arising from a lucky or unlucky random start.

The pipeline is as follows:

1. Evaluate the teacher model's performance.
2. Load the pre-computed teacher logits (if the file exists) or compute them. The computation now uses the `eval_transform` to ensure logits are calculated on clean, non-augmented images.
3. Create and save a common initial state for the ResNet-18 student models.
4. Train the baseline ResNet-18 student using data augmentation.
5. Train the distilled ResNet-18 student, also using data augmentation, but with the distillation loss function.
6. Compare the final test accuracies.

[ ]:
```
# --- Main Experiment ---
teacher_model = AutoModelForImageClassification.from_pretrained("jialicheng/
 ↪cifar100-resnet-50")
teacher_model.to(DEVICE)
teacher_model.eval()

print("\n--- Evaluating Teacher Model ---")
teacher_test_loader = DataLoader(TransformedDataset(test_dataset_pil,␣
 ↪eval_transform), batch_size=BATCH_SIZE)
teacher_accuracy = evaluate(teacher_model, teacher_test_loader, is_teacher=True)
print(f"Teacher (ResNet-50) Test Accuracy: {teacher_accuracy:.2f}%")

if os.path.exists(LOGITS_LOOKUP_PATH):
    print(f"\n--- Loading pre-computed logit lookup table from␣
 ↪{LOGITS_LOOKUP_PATH} ---")
    teacher_logits_lookup = torch.load(LOGITS_LOOKUP_PATH)
else:
    print("\n--- Pre-computing teacher logit lookup table (this will take a␣
 ↪while)... ---")
    # Logits are computed on the clean, non-augmented data
    teacher_logits_lookup = get_teacher_logits_lookup(teacher_model,␣
 ↪full_train_dataset_pil, eval_transform, DEVICE)
    print(f"--- Saving computed lookup table to {LOGITS_LOOKUP_PATH} for future␣
 ↪use... ---")
    torch.save(teacher_logits_lookup, LOGITS_LOOKUP_PATH)

print(f"Loaded {len(teacher_logits_lookup)} total logits into lookup table.")

del teacher_model
torch.cuda.empty_cache()

print("\n--- Creating and saving a common initial state for student models ---")
initial_student_model = StudentNet().to(DEVICE)
initial_student_state_dict = copy.deepcopy(initial_student_model.state_dict())
```

```
print("Initial state saved. This will be used for both baseline and distilled␣
 ↪students.")
del initial_student_model
torch.cuda.empty_cache()


print("\n--- Training Student from Scratch (Baseline) ---")
student_baseline = StudentNet().to(DEVICE)
student_baseline.load_state_dict(initial_student_state_dict)

optimizer_baseline = optim.Adam(student_baseline.parameters(), lr=LR)
loss_fn_baseline = nn.CrossEntropyLoss()

# Create baseline datasets by applying the correct transforms to the PIL subsets
baseline_train_dataset = TransformedDataset(train_subset_pil, student_transform)
baseline_val_dataset = TransformedDataset(val_subset_pil, eval_transform)
baseline_train_loader = DataLoader(baseline_train_dataset,␣
 ↪batch_size=BATCH_SIZE, shuffle=True)
baseline_val_loader = DataLoader(baseline_val_dataset, batch_size=BATCH_SIZE,␣
 ↪shuffle=False)

student_baseline = train_and_validate(
    student_baseline, baseline_train_loader, baseline_val_loader,␣
 ↪optimizer_baseline,
    loss_fn_baseline, EPOCHS, PATIENCE, is_distillation=False
)
final_test_loader = DataLoader(TransformedDataset(test_dataset_pil,␣
 ↪eval_transform), batch_size=BATCH_SIZE)
baseline_accuracy = evaluate(student_baseline, final_test_loader)
print(f"Final Baseline Student (ResNet-18) Test Accuracy: {baseline_accuracy:.
 ↪2f}%")


print("\n--- Training Distilled Student (with pre-computed logits) ---")
student_distilled = StudentNet().to(DEVICE)
student_distilled.load_state_dict(initial_student_state_dict)

optimizer_distilled = optim.Adam(student_distilled.parameters(), lr=LR)
loss_fn_distilled = distillation_loss

# Create distillation datasets, applying the correct transforms
distill_train_dataset = DistillationDataset(train_subset_pil,␣
 ↪teacher_logits_lookup, transform=student_transform)
distill_val_dataset = DistillationDataset(val_subset_pil,␣
 ↪teacher_logits_lookup, transform=eval_transform)
```

```
distill_train_loader = DataLoader(distill_train_dataset, batch_size=BATCH_SIZE,␣
 ↪shuffle=True)
distill_val_loader = DataLoader(distill_val_dataset, batch_size=BATCH_SIZE,␣
 ↪shuffle=False)

student_distilled = train_and_validate(
    student_distilled, distill_train_loader, distill_val_loader,␣
 ↪optimizer_distilled,
    loss_fn_distilled, EPOCHS, PATIENCE, is_distillation=True
)
distilled_accuracy = evaluate(student_distilled, final_test_loader)
print(f"Final Distilled Student (ResNet-18) Test Accuracy: {distilled_accuracy:.
 ↪2f}%")
```

```
--- Evaluating Teacher Model ---
Teacher (ResNet-50) Test Accuracy: 82.75%

--- Loading pre-computed logit lookup table from teacher_logits_lookup.pt ---
Loaded 50000 total logits into lookup table.

--- Creating and saving a common initial state for student models ---
Initial state saved. This will be used for both baseline and distilled students.

--- Training Student from Scratch (Baseline) ---
Epoch 1/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 1/1000 | Val Loss: 3.5614 | Val Acc: 15.14% | LR: 0.001000
  -> Validation loss improved, saving model.
Epoch 2/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 2/1000 | Val Loss: 3.0496 | Val Acc: 25.78% | LR: 0.001000
  -> Validation loss improved, saving model.
Epoch 3/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 3/1000 | Val Loss: 2.5818 | Val Acc: 34.90% | LR: 0.001000
  -> Validation loss improved, saving model.
Epoch 4/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 4/1000 | Val Loss: 2.1904 | Val Acc: 41.08% | LR: 0.001000
  -> Validation loss improved, saving model.
Epoch 5/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 5/1000 | Val Loss: 2.0513 | Val Acc: 45.50% | LR: 0.001000
  -> Validation loss improved, saving model.
Epoch 6/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
```

```
Epoch 6/1000 | Val Loss: 2.0786 | Val Acc: 45.20% | LR: 0.001000

Epoch 7/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 7/1000 | Val Loss: 1.8645 | Val Acc: 51.14% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 8/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 8/1000 | Val Loss: 1.6840 | Val Acc: 54.34% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 9/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 9/1000 | Val Loss: 1.5985 | Val Acc: 56.36% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 10/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 10/1000 | Val Loss: 1.5498 | Val Acc: 58.22% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 11/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 11/1000 | Val Loss: 1.5078 | Val Acc: 59.94% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 12/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 12/1000 | Val Loss: 1.4362 | Val Acc: 61.22% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 13/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 13/1000 | Val Loss: 1.4472 | Val Acc: 62.02% | LR: 0.001000

Epoch 14/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 14/1000 | Val Loss: 1.4242 | Val Acc: 62.26% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 15/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 15/1000 | Val Loss: 1.4283 | Val Acc: 63.46% | LR: 0.001000

Epoch 16/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 16/1000 | Val Loss: 1.4199 | Val Acc: 63.96% | LR: 0.000999
  -> Validation loss improved, saving model.

Epoch 17/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 17/1000 | Val Loss: 1.4979 | Val Acc: 63.24% | LR: 0.000999

Epoch 18/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 18/1000 | Val Loss: 1.5110 | Val Acc: 64.20% | LR: 0.000999

Epoch 19/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 19/1000 | Val Loss: 1.5568 | Val Acc: 64.54% | LR: 0.000999
```

```
Epoch 20/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 20/1000 | Val Loss: 1.5099 | Val Acc: 63.80% | LR: 0.000999
Epoch 21/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 21/1000 | Val Loss: 1.5293 | Val Acc: 65.00% | LR: 0.000999
Epoch 22/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 22/1000 | Val Loss: 1.6717 | Val Acc: 64.22% | LR: 0.000999
Epoch 23/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 23/1000 | Val Loss: 1.6465 | Val Acc: 64.32% | LR: 0.000999
Epoch 24/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 24/1000 | Val Loss: 1.7055 | Val Acc: 64.62% | LR: 0.000999
Epoch 25/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 25/1000 | Val Loss: 1.7608 | Val Acc: 64.88% | LR: 0.000999
Epoch 26/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 26/1000 | Val Loss: 1.8029 | Val Acc: 64.46% | LR: 0.000998
  -> Early stopping triggered after 10 epochs with no improvement.
Final Baseline Student (ResNet-18) Test Accuracy: 63.94%

--- Training Distilled Student (with pre-computed logits) ---
Epoch 1/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 1/1000 | Val Loss: 3.0917 | Val Acc: 17.20% | LR: 0.001000
  -> Validation loss improved, saving model.
Epoch 2/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 2/1000 | Val Loss: 2.4684 | Val Acc: 30.40% | LR: 0.001000
  -> Validation loss improved, saving model.
Epoch 3/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 3/1000 | Val Loss: 1.9943 | Val Acc: 39.38% | LR: 0.001000
  -> Validation loss improved, saving model.
Epoch 4/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 4/1000 | Val Loss: 1.7348 | Val Acc: 46.82% | LR: 0.001000
  -> Validation loss improved, saving model.
Epoch 5/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 5/1000 | Val Loss: 1.5247 | Val Acc: 52.62% | LR: 0.001000
  -> Validation loss improved, saving model.
Epoch 6/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
```

```
Epoch 6/1000 | Val Loss: 1.3759 | Val Acc: 55.86% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 7/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 7/1000 | Val Loss: 1.3024 | Val Acc: 57.94% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 8/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 8/1000 | Val Loss: 1.3712 | Val Acc: 56.90% | LR: 0.001000

Epoch 9/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 9/1000 | Val Loss: 1.1107 | Val Acc: 62.44% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 10/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 10/1000 | Val Loss: 1.0872 | Val Acc: 64.34% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 11/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 11/1000 | Val Loss: 1.0247 | Val Acc: 65.18% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 12/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 12/1000 | Val Loss: 0.9880 | Val Acc: 66.24% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 13/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 13/1000 | Val Loss: 0.9603 | Val Acc: 68.02% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 14/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 14/1000 | Val Loss: 0.9513 | Val Acc: 67.92% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 15/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 15/1000 | Val Loss: 0.9049 | Val Acc: 68.90% | LR: 0.001000
  -> Validation loss improved, saving model.

Epoch 16/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 16/1000 | Val Loss: 0.8991 | Val Acc: 69.62% | LR: 0.000999
  -> Validation loss improved, saving model.

Epoch 17/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 17/1000 | Val Loss: 0.8406 | Val Acc: 71.54% | LR: 0.000999
  -> Validation loss improved, saving model.

Epoch 18/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 18/1000 | Val Loss: 0.8852 | Val Acc: 70.50% | LR: 0.000999
```

```
Epoch 19/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 19/1000 | Val Loss: 0.8574 | Val Acc: 71.50% | LR: 0.000999
Epoch 20/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 20/1000 | Val Loss: 0.8467 | Val Acc: 71.20% | LR: 0.000999
Epoch 21/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 21/1000 | Val Loss: 0.7989 | Val Acc: 72.78% | LR: 0.000999
  -> Validation loss improved, saving model.
Epoch 22/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 22/1000 | Val Loss: 0.8202 | Val Acc: 72.62% | LR: 0.000999
Epoch 23/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 23/1000 | Val Loss: 0.8316 | Val Acc: 73.32% | LR: 0.000999
Epoch 24/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 24/1000 | Val Loss: 0.7805 | Val Acc: 73.66% | LR: 0.000999
  -> Validation loss improved, saving model.
Epoch 25/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 25/1000 | Val Loss: 0.7713 | Val Acc: 73.32% | LR: 0.000999
  -> Validation loss improved, saving model.
Epoch 26/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 26/1000 | Val Loss: 0.7737 | Val Acc: 73.26% | LR: 0.000998
Epoch 27/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 27/1000 | Val Loss: 0.7971 | Val Acc: 73.14% | LR: 0.000998
Epoch 28/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 28/1000 | Val Loss: 0.7560 | Val Acc: 74.22% | LR: 0.000998
  -> Validation loss improved, saving model.
Epoch 29/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 29/1000 | Val Loss: 0.7429 | Val Acc: 75.06% | LR: 0.000998
  -> Validation loss improved, saving model.
Epoch 30/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 30/1000 | Val Loss: 0.7305 | Val Acc: 74.84% | LR: 0.000998
  -> Validation loss improved, saving model.
Epoch 31/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 31/1000 | Val Loss: 0.7720 | Val Acc: 73.84% | LR: 0.000998
Epoch 32/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
```

```
Epoch 32/1000 | Val Loss: 0.7249 | Val Acc: 75.08% | LR: 0.000998
  -> Validation loss improved, saving model.

Epoch 33/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 33/1000 | Val Loss: 0.7328 | Val Acc: 74.62% | LR: 0.000997

Epoch 34/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 34/1000 | Val Loss: 0.7269 | Val Acc: 75.14% | LR: 0.000997

Epoch 35/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 35/1000 | Val Loss: 0.7178 | Val Acc: 74.76% | LR: 0.000997
  -> Validation loss improved, saving model.

Epoch 36/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 36/1000 | Val Loss: 0.7165 | Val Acc: 75.32% | LR: 0.000997
  -> Validation loss improved, saving model.

Epoch 37/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 37/1000 | Val Loss: 0.7101 | Val Acc: 75.64% | LR: 0.000997
  -> Validation loss improved, saving model.

Epoch 38/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 38/1000 | Val Loss: 0.7044 | Val Acc: 75.64% | LR: 0.000997
  -> Validation loss improved, saving model.

Epoch 39/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 39/1000 | Val Loss: 0.6989 | Val Acc: 75.54% | LR: 0.000996
  -> Validation loss improved, saving model.

Epoch 40/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 40/1000 | Val Loss: 0.6923 | Val Acc: 76.04% | LR: 0.000996
  -> Validation loss improved, saving model.

Epoch 41/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 41/1000 | Val Loss: 0.7143 | Val Acc: 75.84% | LR: 0.000996

Epoch 42/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 42/1000 | Val Loss: 0.6848 | Val Acc: 76.40% | LR: 0.000996
  -> Validation loss improved, saving model.

Epoch 43/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 43/1000 | Val Loss: 0.6924 | Val Acc: 75.64% | LR: 0.000996

Epoch 44/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 44/1000 | Val Loss: 0.7033 | Val Acc: 75.66% | LR: 0.000995

Epoch 45/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
```

```
Epoch 45/1000 | Val Loss: 0.6642 | Val Acc: 76.80% | LR: 0.000995
  -> Validation loss improved, saving model.
Epoch 46/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 46/1000 | Val Loss: 0.6782 | Val Acc: 76.34% | LR: 0.000995
Epoch 47/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 47/1000 | Val Loss: 0.6856 | Val Acc: 76.76% | LR: 0.000995
Epoch 48/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 48/1000 | Val Loss: 0.7060 | Val Acc: 75.76% | LR: 0.000995
Epoch 49/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 49/1000 | Val Loss: 0.6691 | Val Acc: 76.48% | LR: 0.000994
Epoch 50/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 50/1000 | Val Loss: 0.6680 | Val Acc: 76.88% | LR: 0.000994
Epoch 51/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 51/1000 | Val Loss: 0.6870 | Val Acc: 76.32% | LR: 0.000994
Epoch 52/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 52/1000 | Val Loss: 0.6653 | Val Acc: 77.20% | LR: 0.000994
Epoch 53/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 53/1000 | Val Loss: 0.6626 | Val Acc: 77.32% | LR: 0.000993
  -> Validation loss improved, saving model.
Epoch 54/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 54/1000 | Val Loss: 0.6630 | Val Acc: 76.68% | LR: 0.000993
Epoch 55/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 55/1000 | Val Loss: 0.6706 | Val Acc: 77.40% | LR: 0.000993
Epoch 56/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 56/1000 | Val Loss: 0.6525 | Val Acc: 77.32% | LR: 0.000993
  -> Validation loss improved, saving model.
Epoch 57/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 57/1000 | Val Loss: 0.6606 | Val Acc: 77.06% | LR: 0.000992
Epoch 58/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 58/1000 | Val Loss: 0.6690 | Val Acc: 76.82% | LR: 0.000992
Epoch 59/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 59/1000 | Val Loss: 0.6619 | Val Acc: 76.60% | LR: 0.000992
Epoch 60/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
```

```
Epoch 60/1000 | Val Loss: 0.6435 | Val Acc: 77.08% | LR: 0.000991
  -> Validation loss improved, saving model.
Epoch 61/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 61/1000 | Val Loss: 0.6485 | Val Acc: 77.30% | LR: 0.000991
Epoch 62/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 62/1000 | Val Loss: 0.6505 | Val Acc: 77.46% | LR: 0.000991
Epoch 63/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 63/1000 | Val Loss: 0.6472 | Val Acc: 77.24% | LR: 0.000991
Epoch 64/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 64/1000 | Val Loss: 0.6415 | Val Acc: 78.02% | LR: 0.000990
  -> Validation loss improved, saving model.
Epoch 65/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 65/1000 | Val Loss: 0.6416 | Val Acc: 77.44% | LR: 0.000990
Epoch 66/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 66/1000 | Val Loss: 0.6398 | Val Acc: 77.88% | LR: 0.000990
  -> Validation loss improved, saving model.
Epoch 67/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 67/1000 | Val Loss: 0.6358 | Val Acc: 77.46% | LR: 0.000989
  -> Validation loss improved, saving model.
Epoch 68/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 68/1000 | Val Loss: 0.6445 | Val Acc: 77.22% | LR: 0.000989
Epoch 69/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 69/1000 | Val Loss: 0.6442 | Val Acc: 77.62% | LR: 0.000989
Epoch 70/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 70/1000 | Val Loss: 0.6329 | Val Acc: 78.22% | LR: 0.000988
  -> Validation loss improved, saving model.
Epoch 71/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 71/1000 | Val Loss: 0.6472 | Val Acc: 77.20% | LR: 0.000988
Epoch 72/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 72/1000 | Val Loss: 0.6361 | Val Acc: 77.34% | LR: 0.000988
Epoch 73/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 73/1000 | Val Loss: 0.6367 | Val Acc: 77.80% | LR: 0.000987
Epoch 74/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 74/1000 | Val Loss: 0.6404 | Val Acc: 77.40% | LR: 0.000987
```

```
Epoch 75/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 75/1000 | Val Loss: 0.6285 | Val Acc: 78.28% | LR: 0.000987
  -> Validation loss improved, saving model.
Epoch 76/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 76/1000 | Val Loss: 0.6342 | Val Acc: 78.18% | LR: 0.000986
Epoch 77/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 77/1000 | Val Loss: 0.6316 | Val Acc: 77.60% | LR: 0.000986
Epoch 78/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 78/1000 | Val Loss: 0.6347 | Val Acc: 78.04% | LR: 0.000985
Epoch 79/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 79/1000 | Val Loss: 0.6422 | Val Acc: 77.54% | LR: 0.000985
Epoch 80/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 80/1000 | Val Loss: 0.6258 | Val Acc: 77.96% | LR: 0.000985
  -> Validation loss improved, saving model.
Epoch 81/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 81/1000 | Val Loss: 0.6265 | Val Acc: 78.26% | LR: 0.000984
Epoch 82/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 82/1000 | Val Loss: 0.6474 | Val Acc: 77.44% | LR: 0.000984
Epoch 83/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 83/1000 | Val Loss: 0.6239 | Val Acc: 77.74% | LR: 0.000984
  -> Validation loss improved, saving model.
Epoch 84/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 84/1000 | Val Loss: 0.6213 | Val Acc: 78.04% | LR: 0.000983
  -> Validation loss improved, saving model.
Epoch 85/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 85/1000 | Val Loss: 0.6216 | Val Acc: 78.38% | LR: 0.000983
Epoch 86/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 86/1000 | Val Loss: 0.6289 | Val Acc: 78.04% | LR: 0.000982
Epoch 87/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 87/1000 | Val Loss: 0.6309 | Val Acc: 78.20% | LR: 0.000982
Epoch 88/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 88/1000 | Val Loss: 0.6343 | Val Acc: 77.18% | LR: 0.000981
Epoch 89/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
```

```
Epoch 89/1000 | Val Loss: 0.6217 | Val Acc: 77.80% | LR: 0.000981

Epoch 90/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 90/1000 | Val Loss: 0.6223 | Val Acc: 78.08% | LR: 0.000981

Epoch 91/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 91/1000 | Val Loss: 0.6156 | Val Acc: 78.56% | LR: 0.000980
  -> Validation loss improved, saving model.

Epoch 92/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 92/1000 | Val Loss: 0.6230 | Val Acc: 78.22% | LR: 0.000980

Epoch 93/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 93/1000 | Val Loss: 0.6192 | Val Acc: 78.58% | LR: 0.000979

Epoch 94/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 94/1000 | Val Loss: 0.6200 | Val Acc: 78.20% | LR: 0.000979

Epoch 95/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 95/1000 | Val Loss: 0.6135 | Val Acc: 78.56% | LR: 0.000978
  -> Validation loss improved, saving model.

Epoch 96/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 96/1000 | Val Loss: 0.6170 | Val Acc: 78.10% | LR: 0.000978

Epoch 97/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 97/1000 | Val Loss: 0.6105 | Val Acc: 78.30% | LR: 0.000977
  -> Validation loss improved, saving model.

Epoch 98/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 98/1000 | Val Loss: 0.6213 | Val Acc: 77.90% | LR: 0.000977

Epoch 99/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 99/1000 | Val Loss: 0.6201 | Val Acc: 77.96% | LR: 0.000976

Epoch 100/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 100/1000 | Val Loss: 0.6184 | Val Acc: 78.24% | LR: 0.000976

Epoch 101/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 101/1000 | Val Loss: 0.6126 | Val Acc: 78.24% | LR: 0.000976

Epoch 102/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 102/1000 | Val Loss: 0.6230 | Val Acc: 77.68% | LR: 0.000975

Epoch 103/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 103/1000 | Val Loss: 0.6189 | Val Acc: 78.24% | LR: 0.000975

Epoch 104/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
```

```
Epoch 104/1000 | Val Loss: 0.6128 | Val Acc: 78.66% | LR: 0.000974
Epoch 105/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 105/1000 | Val Loss: 0.6116 | Val Acc: 78.74% | LR: 0.000974
Epoch 106/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 106/1000 | Val Loss: 0.6226 | Val Acc: 78.44% | LR: 0.000973
Epoch 107/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 107/1000 | Val Loss: 0.6105 | Val Acc: 78.52% | LR: 0.000973
  -> Validation loss improved, saving model.
Epoch 108/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 108/1000 | Val Loss: 0.6166 | Val Acc: 78.60% | LR: 0.000972
Epoch 109/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 109/1000 | Val Loss: 0.6043 | Val Acc: 78.86% | LR: 0.000971
  -> Validation loss improved, saving model.
Epoch 110/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 110/1000 | Val Loss: 0.6094 | Val Acc: 78.58% | LR: 0.000971
Epoch 111/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 111/1000 | Val Loss: 0.6090 | Val Acc: 78.58% | LR: 0.000970
Epoch 112/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 112/1000 | Val Loss: 0.6066 | Val Acc: 78.54% | LR: 0.000970
Epoch 113/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 113/1000 | Val Loss: 0.6100 | Val Acc: 78.40% | LR: 0.000969
Epoch 114/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 114/1000 | Val Loss: 0.6054 | Val Acc: 78.50% | LR: 0.000969
Epoch 115/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 115/1000 | Val Loss: 0.6068 | Val Acc: 78.74% | LR: 0.000968
Epoch 116/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 116/1000 | Val Loss: 0.6035 | Val Acc: 78.60% | LR: 0.000968
  -> Validation loss improved, saving model.
Epoch 117/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 117/1000 | Val Loss: 0.6138 | Val Acc: 78.28% | LR: 0.000967
Epoch 118/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 118/1000 | Val Loss: 0.6061 | Val Acc: 78.46% | LR: 0.000967
Epoch 119/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
```

```
Epoch 119/1000 | Val Loss: 0.6112 | Val Acc: 78.74% | LR: 0.000966
Epoch 120/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 120/1000 | Val Loss: 0.6048 | Val Acc: 78.84% | LR: 0.000965
Epoch 121/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 121/1000 | Val Loss: 0.6159 | Val Acc: 78.32% | LR: 0.000965
Epoch 122/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 122/1000 | Val Loss: 0.6058 | Val Acc: 78.54% | LR: 0.000964
Epoch 123/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 123/1000 | Val Loss: 0.6097 | Val Acc: 78.58% | LR: 0.000964
Epoch 124/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 124/1000 | Val Loss: 0.5997 | Val Acc: 78.52% | LR: 0.000963
  -> Validation loss improved, saving model.
Epoch 125/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 125/1000 | Val Loss: 0.6065 | Val Acc: 78.58% | LR: 0.000963
Epoch 126/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 126/1000 | Val Loss: 0.6063 | Val Acc: 78.60% | LR: 0.000962
Epoch 127/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 127/1000 | Val Loss: 0.5949 | Val Acc: 79.10% | LR: 0.000961
  -> Validation loss improved, saving model.
Epoch 128/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 128/1000 | Val Loss: 0.6018 | Val Acc: 78.90% | LR: 0.000961
Epoch 129/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 129/1000 | Val Loss: 0.6026 | Val Acc: 78.52% | LR: 0.000960
Epoch 130/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 130/1000 | Val Loss: 0.6072 | Val Acc: 78.98% | LR: 0.000959
Epoch 131/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 131/1000 | Val Loss: 0.6052 | Val Acc: 78.46% | LR: 0.000959
Epoch 132/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 132/1000 | Val Loss: 0.5984 | Val Acc: 79.36% | LR: 0.000958
Epoch 133/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 133/1000 | Val Loss: 0.6020 | Val Acc: 78.78% | LR: 0.000958
Epoch 134/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]
Epoch 134/1000 | Val Loss: 0.6083 | Val Acc: 78.64% | LR: 0.000957
```

```
Epoch 135/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 135/1000 | Val Loss: 0.6004 | Val Acc: 79.06% | LR: 0.000956

Epoch 136/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 136/1000 | Val Loss: 0.5990 | Val Acc: 78.98% | LR: 0.000956

Epoch 137/1000 Training:   0%|          | 0/704 [00:00<?, ?it/s]

Epoch 137/1000 | Val Loss: 0.6044 | Val Acc: 78.34% | LR: 0.000955
  -> Early stopping triggered after 10 epochs with no improvement.
Final Distilled Student (ResNet-18) Test Accuracy: 77.74%
```

2.9 Final Results (ResNet-18 Student)

Finally, we compare the performance of the powerful ResNet-50 teacher, our baseline ResNet-18 student, and the distilled ResNet-18 student.

[ ]:
```python
print("\n\n--- FINAL COMPREHENSIVE RESULTS (ResNet-18 Student) ---")
print(f"Teacher (ResNet-50) Accuracy: {teacher_accuracy:.2f}%")
print(f"Student (Baseline - ResNet18) Accuracy:  {baseline_accuracy:.2f}%")
print(f"Student (Distilled - ResNet18) Accuracy: {distilled_accuracy:.2f}%")
print("----------------------------------")
improvement = distilled_accuracy - baseline_accuracy
print(f"Improvement with Distillation: {improvement:.2f}%")
```

```
--- FINAL COMPREHENSIVE RESULTS (ResNet-18 Student) ---
Teacher (ResNet-50) Accuracy: 82.75%
Student (Baseline - ResNet18) Accuracy:  63.94%
Student (Distilled - ResNet18) Accuracy: 77.74%
----------------------------------
Improvement with Distillation: 13.80%
```

2.10 Key Observations

Some critical insights emerge from these experiments:

- **Distillation is More Impactful on Complex Tasks:** The accuracy improvement from distillation was notably greater on the 100-class CIFAR-100 dataset than on the 10-class CIFAR-10. This suggests that the value of the teacher's "dark knowledge" is magnified when the classification task is more complex and the decision boundaries are harder to define.
- **Bridging the Performance Gap:** The final experiment with the ResNet-18 student is particularly compelling. Despite being a significantly smaller and more computationally efficient model than the ResNet-50 teacher, the distilled student achieved a test accuracy that nearly closed the performance gap between them. This powerfully illustrates the primary promise of knowledge distillation: to create compact, efficient models that retain the high performance of much larger counterparts, making them viable for deployment in resource-constrained environments.

- **Distillation Enables More Stable and Prolonged Training**: The logs clearly show that the distilled student model continued to improve its validation loss for many more epochs compared to the baseline student, which quickly overfit and triggered early stopping. This demonstrates that the teacher's soft targets act as a powerful regulariser, providing a consistent learning signal that prevents the student from simply memorising the hard labels. The resulting accuracy improvement on the complex CIFAR-100 task is a direct result of this guided and more robust training process.

3 Concluding Remarks and Future Directions

The experiments in this notebook successfully demonstrate the core principles and effectiveness of knowledge distillation. By transferring knowledge from a large, pre-trained teacher model to smaller student models using softened logits, we observed significant performance improvements across both the CIFAR-10 and CIFAR-100 datasets.

However, this implementation explores only one form of distillation signal. Further things which could easily be added to the current notebook are the following (but there are many more!):

- **Intermediate Feature Matching:** A highly effective technique involves compelling the student to mimic the teacher's internal "thought process." This can be achieved by adding a loss term that minimises the difference between the feature maps of the student's intermediate layers and those of the teacher. For instance, one could match the output of the final feature extraction block in the student's `StudentNet` with the corresponding block in the teacher, forcing the student to learn the teacher's method of feature representation, not just its final prediction.
- **Dynamic Distillation Schedules:** The influence of the different loss components does not have to be static. Advanced methods often involve scheduling the hyperparameters during training. For example, the `alpha` parameter could be annealed over time, starting with a high reliance on the teacher's guidance (`alpha` is low) and gradually increasing the weight of the ground-truth labels as the student model becomes more capable.