```
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 time
from torch.utils.data import DataLoader, random_split

# ---------------- GPU Setup ----------------
print("PyTorch CUDA Available:", torch.cuda.is_available())
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# ---------------- Data Preparation ----------------
batch_size = 256
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Load full training dataset
full_train_dataset = torchvision.datasets.CIFAR10(
    root="./data", train=True, download=True, transform=transform
)

# Split training dataset into training (80%) and validation (20%) sets
train_size = int(0.8 * len(full_train_dataset))
val_size = len(full_train_dataset) - train_size
train_dataset, val_dataset = random_split(full_train_dataset, [train_size, val_

# Load test dataset
test_dataset = torchvision.datasets.CIFAR10(
    root="./data", train=False, download=True, transform=transform
)

# Create DataLoaders
train_loader = DataLoader(
    train_dataset, batch_size=batch_size, shuffle=True, num_workers=0, pin_memc
)
val_loader = DataLoader(
    val_dataset, batch_size=batch_size, shuffle=False, num_workers=0, pin_memor
)
test_loader = DataLoader(
    test_dataset, batch_size=batch_size, shuffle=False, num_workers=0, pin_memc
)

print(f"Dataset sizes: Training={train_size}, Validation={val_size}, Test={len(
```

```
PyTorch CUDA Available: True
Using device: cuda
Dataset sizes: Training=40000, Validation=10000, Test=10000
```

```
# ---------------- CNN Model Definitions ----------------
class CNN_1A(nn.Module):
    # Simpler Model (2 Conv Layers)
    def __init__(self):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        with torch.no_grad():
            test = torch.zeros(1, 3, 32, 32)
            flat = self.features(test).view(1, -1).size(1)
        self.classifier = nn.Linear(flat, 10)

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        return self.classifier(x)

class CNN_1B(nn.Module):
    # More Complex Model (3 Conv Layers)
    def __init__(self):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(64, 128, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        with torch.no_grad():
            test = torch.zeros(1, 3, 32, 32)
            flat = self.features(test).view(1, -1).size(1)
        self.classifier = nn.Linear(flat, 10)

    def forward(self, x):
        x = self.features(x)
```

```
        x = x.view(x.size(0), -1)
        return self.classifier(x)
```

```
# ---------------- Validation Function ----------------
def validate_model(model, val_loader, criterion):
    """Calculates the loss on the validation dataset (unseen data)."""
    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for imgs, lbls in val_loader:
            imgs, lbls = imgs.to(device), lbls.to(device)
            outputs = model(imgs)
            loss = criterion(outputs, lbls)
            val_loss += loss.item() * imgs.size(0)

    avg_val_loss = val_loss / len(val_loader.dataset)
    return avg_val_loss
```

```
# ---------------- Training and Validation Loop ----------------
def train_and_validate(model, model_name, train_loader, val_loader, criterion,
    """Trains the model, tracks both losses, and checks for overfitting."""
    model.to(device)
    train_losses = []
    val_losses = []

    print(f"\n--- Starting Training for {model_name} (Epochs: {epochs}) ---")
    start = time.time()

    for epoch in range(epochs):
        # --- Training Step ---
        model.train()
        running_loss = 0.0
        for imgs, lbls in train_loader:
            imgs, lbls = imgs.to(device), lbls.to(device)
            optimizer.zero_grad()
            outputs = model(imgs)
            loss = criterion(outputs, lbls)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        avg_train_loss = running_loss / len(train_loader)
        train_losses.append(avg_train_loss)

        # --- Validation Step (Quick Check for Generalization) ---
        avg_val_loss = validate_model(model, val_loader, criterion)
        val_losses.append(avg_val_loss)

        # Console Output for quick monitoring
        print(f"Epoch [{epoch+1}/{epochs}] | Train Loss: {avg_train_loss:.4f} |
```

```
        # Simple Early Stopping Heuristic Check
        if epoch > 10 and avg_val_loss > val_losses[-2]:
            print(f"--- {model_name} WARNING: Validation loss increased at epoc

    training_time = time.time() - start
    return train_losses, val_losses, training_time
```

```
# ---------------- Final Evaluation ----------------
def evaluate_final_accuracy(model, test_loader):
    model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for imgs, lbls in test_loader:
            imgs, lbls = imgs.to(device), lbls.to(device)
            outputs = model(imgs)
            _, preds = torch.max(outputs, 1)
            total += lbls.size(0)
            correct += (preds == lbls).sum().item()
    accuracy = 100 * correct / total
    return accuracy
```

```
# ---------------- Run Experiment ----------------
num_epochs = 50
criterion = nn.CrossEntropyLoss()
weight_decay_rate = 1e-4 # Used for L2 Regularization

# --- CNN 1A (Simpler Model) ---
model_a = CNN_1A()
optimizer_a = optim.Adam(model_a.parameters(), lr=0.001, weight_decay=weight
train_losses_a, val_losses_a, a_time = train_and_validate(
    model_a, "CNN 1A", train_loader, val_loader, criterion, optimizer_a, epo
)
a_accuracy = evaluate_final_accuracy(model_a, test_loader)
a_params = sum(p.numel() for p in model_a.parameters())

# --- CNN 1B (More Complex Model) ---
model_b = CNN_1B()
optimizer_b = optim.Adam(model_b.parameters(), lr=0.001, weight_decay=weight
train_losses_b, val_losses_b, b_time = train_and_validate(
    model_b, "CNN 1B", train_loader, val_loader, criterion, optimizer_b, epo
)
b_accuracy = evaluate_final_accuracy(model_b, test_loader)
b_params = sum(p.numel() for p in model_b.parameters())
```

```
Epoch [22/50] | Train Loss: 0.2594 | VAL Loss: 0.9321
--- CNN 1B WARNING: Validation loss increased at epoch 22. Potential Overfitti
Epoch [23/50] | Train Loss: 0.2424 | VAL Loss: 1.0182
--- CNN 1B WARNING: Validation loss increased at epoch 23. Potential Overfitti
Epoch [24/50] | Train Loss: 0.2210 | VAL Loss: 0.9699
Epoch [25/50] | Train Loss: 0.1920 | VAL Loss: 1.0110
--- CNN 1B WARNING: Validation loss increased at epoch 25. Potential Overfitti
Epoch [26/50] | Train Loss: 0.1687 | VAL Loss: 1.1092
--- CNN 1B WARNING: Validation loss increased at epoch 26. Potential Overfitti
Epoch [27/50] | Train Loss: 0.1630 | VAL Loss: 1.1347
--- CNN 1B WARNING: Validation loss increased at epoch 27. Potential Overfitti
Epoch [28/50] | Train Loss: 0.1436 | VAL Loss: 1.1617
--- CNN 1B WARNING: Validation loss increased at epoch 28. Potential Overfitti
Epoch [29/50] | Train Loss: 0.1355 | VAL Loss: 1.2394
--- CNN 1B WARNING: Validation loss increased at epoch 29. Potential Overfitti
Epoch [30/50] | Train Loss: 0.1318 | VAL Loss: 1.2448
--- CNN 1B WARNING: Validation loss increased at epoch 30. Potential Overfitti
Epoch [31/50] | Train Loss: 0.1025 | VAL Loss: 1.2864
--- CNN 1B WARNING: Validation loss increased at epoch 31. Potential Overfitti
Epoch [32/50] | Train Loss: 0.1155 | VAL Loss: 1.3701
--- CNN 1B WARNING: Validation loss increased at epoch 32. Potential Overfitti
Epoch [33/50] | Train Loss: 0.0918 | VAL Loss: 1.3137
Epoch [34/50] | Train Loss: 0.0772 | VAL Loss: 1.4187
--- CNN 1B WARNING: Validation loss increased at epoch 34. Potential Overfitti
Epoch [35/50] | Train Loss: 0.0676 | VAL Loss: 1.4493
--- CNN 1B WARNING: Validation loss increased at epoch 35. Potential Overfitti
Epoch [36/50] | Train Loss: 0.0529 | VAL Loss: 1.4797
--- CNN 1B WARNING: Validation loss increased at epoch 36. Potential Overfitti
Epoch [37/50] | Train Loss: 0.0566 | VAL Loss: 1.5425
--- CNN 1B WARNING: Validation loss increased at epoch 37. Potential Overfitti
Epoch [38/50] | Train Loss: 0.0661 | VAL Loss: 1.5125
Epoch [39/50] | Train Loss: 0.0676 | VAL Loss: 1.6113
--- CNN 1B WARNING: Validation loss increased at epoch 39. Potential Overfitti
Epoch [40/50] | Train Loss: 0.0523 | VAL Loss: 1.6669
--- CNN 1B WARNING: Validation loss increased at epoch 40. Potential Overfitti
Epoch [41/50] | Train Loss: 0.0575 | VAL Loss: 1.6599
Epoch [42/50] | Train Loss: 0.0423 | VAL Loss: 1.7458
--- CNN 1B WARNING: Validation loss increased at epoch 42. Potential Overfitti
Epoch [43/50] | Train Loss: 0.0461 | VAL Loss: 1.6789
Epoch [44/50] | Train Loss: 0.0672 | VAL Loss: 1.8855
--- CNN 1B WARNING: Validation loss increased at epoch 44. Potential Overfitti
Epoch [45/50] | Train Loss: 0.0780 | VAL Loss: 1.7397
Epoch [46/50] | Train Loss: 0.0471 | VAL Loss: 1.7559
--- CNN 1B WARNING: Validation loss increased at epoch 46. Potential Overfitti
Epoch [47/50] | Train Loss: 0.0290 | VAL Loss: 1.7648
--- CNN 1B WARNING: Validation loss increased at epoch 47. Potential Overfitti
Epoch [48/50] | Train Loss: 0.0238 | VAL Loss: 1.7726
--- CNN 1B WARNING: Validation loss increased at epoch 48. Potential Overfitti
Epoch [49/50] | Train Loss: 0.0110 | VAL Loss: 1.7886
--- CNN 1B WARNING: Validation loss increased at epoch 49. Potential Overfitti
Epoch [50/50] | Train Loss: 0.0048 | VAL Loss: 1.8097
--- CNN 1B WARNING: Validation loss increased at epoch 50. Potential Overfitti
```

```
# ---------------- Results Summary ----------------
print("\n--- Final Performance Summary ---")
```

```
print(f"CNN 1A (Simpler) - Params: {a_params:,} | Time: {a_time:.2f}s | Test Ac
print(f"CNN 1B (Complex) - Params: {b_params:,} | Time: {b_time:.2f}s | Test Ac
```

```
--- Final Performance Summary ---
CNN 1A (Simpler) - Params: 60,362 | Time: 599.14s | Test Accuracy: 68.26%
CNN 1B (Complex) - Params: 113,738 | Time: 598.73s | Test Accuracy: 72.52%
```

```
# ---------------- Plot Losses ----------------
plt.figure(figsize=(14, 6))

# Plot 1: CNN 1A Generalization Curve
plt.subplot(1, 2, 1)
plt.plot(train_losses_a, label="Train Loss", color='blue')
plt.plot(val_losses_a, label="Validation Loss", color='red')
plt.title("CNN 1A (Simpler) Loss Curve - Check for Overfitting")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)

# Plot 2: CNN 1B Generalization Curve
plt.subplot(1, 2, 2)
plt.plot(train_losses_b, label="Train Loss", color='blue')
plt.plot(val_losses_b, label="Validation Loss", color='red')
plt.title("CNN 1B (Complex) Loss Curve - Check for Overfitting")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```