```
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
```

```
# ---------------- 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)
torch.backends.cudnn.benchmark = True  # Optimize GPU kernels
```

```
PyTorch CUDA Available: True
Using device: cuda
```

```
# ---------------- Data ----------------
batch_size = 256  # Increased batch size
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,0.5,0.5), (0.5,0.5,0.5))
])

train_dataset = torchvision.datasets.CIFAR10(root="./data", train=True, download=True, transform=transform)
test_dataset = torchvision.datasets.CIFAR10(root="./data", train=False, download=True, transform=transform)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=0, pin_memory=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_workers=0, pin_memory=True)
```

```
# ---------------- CNN Models ----------------
class CNN_1A(nn.Module):
    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)
        )
        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):
    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)
        )
        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)
```

```
# ---------------- Training Function ----------------
def train_model(model, train_loader, criterion, optimizer, epochs=30):
    scaler = torch.amp.GradScaler(device="cuda")  # Mixed precision
    model.to(device)
    train_losses = []

    start = time.time()
    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        for imgs, lbls in train_loader:
            imgs, lbls = imgs.to(device, non_blocking=True), lbls.to(device, non_blocking=True)
            optimizer.zero_grad()
            with torch.amp.autocast(device_type="cuda"):
                outputs = model(imgs)
                loss = criterion(outputs, lbls)
            scaler.scale(loss).backward()
            scaler.step(optimizer)
            scaler.update()
            running_loss += loss.item()
        avg_loss = running_loss / len(train_loader)
        train_losses.append(avg_loss)
        print(f"Epoch [{epoch+1}/{epochs}] Loss: {avg_loss:.4f}")
    end = time.time()
    training_time = end - start
    return train_losses, training_time
```

```
# ---------------- Evaluation ----------------
def evaluate_model(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
```

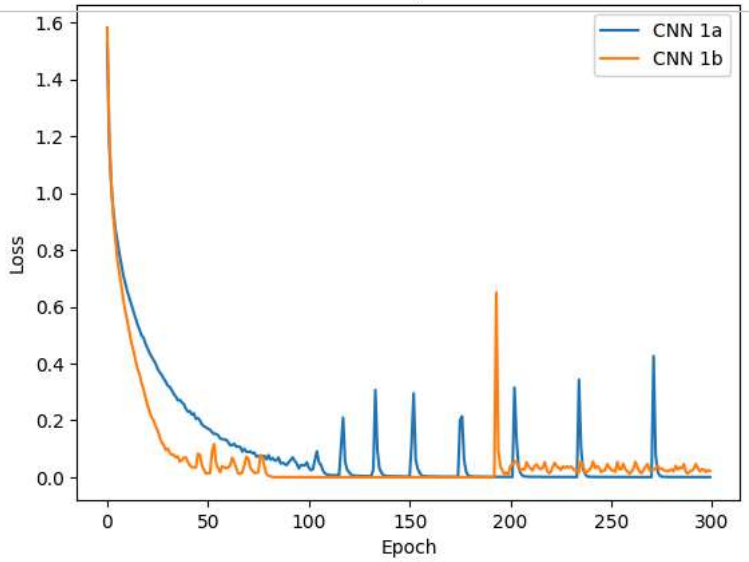
```
# ---------------- Run Experiments ----------------
num_epochs = 300
criterion = nn.CrossEntropyLoss()

# CNN 1a
model_a = CNN_1A()
optimizer_a = optim.Adam(model_a.parameters(), lr=0.001)
train_losses_a, a_time = train_model(model_a, train_loader, criterion, optimizer_a, epochs=num_epochs)
a_accuracy = evaluate_model(model_a, test_loader)
a_params = sum(p.numel() for p in model_a.parameters())

# CNN 1b
model_b = CNN_1B()
optimizer_b = optim.Adam(model_b.parameters(), lr=0.001)
train_losses_b, b_time = train_model(model_b, train_loader, criterion, optimizer_b, epochs=num_epochs)
b_accuracy = evaluate_model(model_b, test_loader)
b_params = sum(p.numel() for p in model_b.parameters())
```

```
Epoch [261/300] Loss: 0.0192
Epoch [262/300] Loss: 0.0292
Epoch [263/300] Loss: 0.0545
Epoch [264/300] Loss: 0.0347
Epoch [265/300] Loss: 0.0201
Epoch [266/300] Loss: 0.0138
Epoch [267/300] Loss: 0.0239
Epoch [268/300] Loss: 0.0376
Epoch [269/300] Loss: 0.0466
Epoch [270/300] Loss: 0.0241
Epoch [271/300] Loss: 0.0250
Epoch [272/300] Loss: 0.0270
Epoch [273/300] Loss: 0.0302
Epoch [274/300] Loss: 0.0404
Epoch [275/300] Loss: 0.0295
Epoch [276/300] Loss: 0.0301
Epoch [277/300] Loss: 0.0260
Epoch [278/300] Loss: 0.0219
Epoch [279/300] Loss: 0.0210
Epoch [280/300] Loss: 0.0244
Epoch [281/300] Loss: 0.0276
Epoch [282/300] Loss: 0.0209
Epoch [283/300] Loss: 0.0403
Epoch [284/300] Loss: 0.0268
Epoch [285/300] Loss: 0.0349
Epoch [286/300] Loss: 0.0306
Epoch [287/300] Loss: 0.0465
Epoch [288/300] Loss: 0.0193
Epoch [289/300] Loss: 0.0132
Epoch [290/300] Loss: 0.0210
Epoch [291/300] Loss: 0.0226
Epoch [292/300] Loss: 0.0299
Epoch [293/300] Loss: 0.0464
Epoch [294/300] Loss: 0.0310
Epoch [295/300] Loss: 0.0201
Epoch [296/300] Loss: 0.0283
Epoch [297/300] Loss: 0.0266
Epoch [298/300] Loss: 0.0178
Epoch [299/300] Loss: 0.0244
Epoch [300/300] Loss: 0.0214
```

```
# ---------------- Results ----------------
print(f"\nCNN 1a - Time: {a_time:.2f}s, Accuracy: {a_accuracy:.2f}%, Params: {a_params:,}")
print(f"CNN 1b - Time: {b_time:.2f}s, Accuracy: {b_accuracy:.2f}%, Params: {b_params:,}")
```

```
CNN 1a - Time: 3414.04s, Accuracy: 68.29%, Params: 60,362
CNN 1b - Time: 3453.85s, Accuracy: 72.88%, Params: 113,738
```

```
# ---------------- Plot ----------------
plt.figure()
plt.plot(train_losses_a, label="CNN 1a")
plt.plot(train_losses_b, label="CNN 1b")
plt.title("Training Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```


Training Loss
CNN 1a
CNN 1b
Loss
Epoch
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250
300