In [1]:
```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```python
# ---------------------------- 1. 配置实验参数 ----------------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")  # 优先
batch_size = 64
learning_rate = 0.001
epochs = 10  # 训练轮次（CIFAR10简单，10轮足够看到效果）
```

In [3]:
```python
# ---------------------------- 2. 定义AlexNet模型（适配CIFAR10） ----------------
# 原始AlexNet针对224×224图像，CIFAR10是32×32，需调整卷积核和池化层参数
class AlexNet(nn.Module):
    def __init__(self, num_classes=10):  # CIFAR10有10个类别
        super(AlexNet, self).__init__()
        # 特征提取层：5个卷积层 + 3个最大池化层
        self.features = nn.Sequential(
            # 卷积层1：3→64，核11→3（适配小图像），步长4→1，填充2
            nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=2),
            nn.ReLU(inplace=True),
            # LRN层1：论文配置n=5，k=2，alpha=1e-4，beta=0.75
            nn.LocalResponseNorm(size=5, alpha=1e-4, beta=0.75, k=2),
            nn.MaxPool2d(kernel_size=2, stride=2),  # 输出：64×16×16

            # 卷积层2：64 -> 192，核5 -> 3，填充2
            nn.Conv2d(64, 192, kernel_size=3, padding=2),
            nn.ReLU(inplace=True),
            # LRN层2
            nn.LocalResponseNorm(size=5, alpha=1e-4, beta=0.75, k=2),
            nn.MaxPool2d(kernel_size=2, stride=2),  # 输出：192×8×8

            # 卷积层3：192→384，核3，填充1
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),

            # 卷积层4：384→256，核3，填充1
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),

            # 卷积层5：256→256，核3，填充1
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),  # 输出：256×4×4
        )

        # 分类层：3个全连接层
        self.classifier = nn.Sequential(
            nn.Dropout(0.5),  # 防止过拟合
            nn.Linear(256 * 4 * 4, 4096),  # 输入：256×4×4=4096
            nn.ReLU(inplace=True),
```

```
            nn.Dropout(0.5),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),

            nn.Linear(4096, num_classes)  # 输出10个类别概率
        )

    def forward(self, x):
        x = self.features(x)  # 特征提取
        x = x.view(x.size(0), -1)  # 展平（batch_size, 256×4×4）
        x = self.classifier(x)  # 分类
        return x
```

In [5]:
```
# ---------------------------- 3. 数据准备（CIFAR10数据集） ----------------------
# 数据预处理：归一化、随机增强（训练集）
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),  # 随机裁剪
    transforms.RandomHorizontalFlip(),  # 随机水平翻转
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])

train_dataset = datasets.CIFAR10(
    root='./datasets', train=True, download=True, transform=transform_train
)
test_dataset = datasets.CIFAR10(
    root='./datasets', train=False, download=True, transform=transform_test
)

# 数据加载器（批量读取数据）
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False,

# CIFAR10类别名称（用于后续可视化）
classes = ['airplane', 'automobile', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck']
```

```
100.0%
```

In [6]:
```
# ---------------------------- 4. 初始化模型、损失函数、优化器 --------------------
model = AlexNet(num_classes=10).to(device)  # 模型移到GPU/CPU
criterion = nn.CrossEntropyLoss()  # 交叉熵损失（分类任务常用）
optimizer = optim.Adam(model.parameters(), lr=learning_rate)  # Adam优化器

# ---------------------------- 5. 训练函数 ----------------------------
def train(model, train_loader, criterion, optimizer, epoch):
    model.train()  # 训练模式（启用Dropout）
    running_loss = 0.0
    for batch_idx, (inputs, targets) in enumerate(train_loader):
        inputs, targets = inputs.to(device), targets.to(device)
```

```
        # 前向传播
        outputs = model(inputs)
        loss = criterion(outputs, targets)

        # 反向传播+参数更新
        optimizer.zero_grad()  # 清空梯度
        loss.backward()  # 计算梯度
        optimizer.step()  # 更新参数

        running_loss += loss.item()

        # 打印训练进度
        if batch_idx % 100 == 99:  # 每100个batch打印一次
            print(f'Epoch [{epoch+1}/{epochs}], Batch [{batch_idx+1}/{len(tr
            running_loss = 0.0
```

In [ ]:

```
# ------------------------- 6. 测试函数 -------------------------
def test(model, test_loader, criterion):
    model.eval()  # 测试模式（禁用Dropout）
    test_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():  # 禁用梯度计算（节省内存）
        for inputs, targets in test_loader:
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, targets)

            test_loss += loss.item()
            _, predicted = outputs.max(1)  # 取概率最大的类别
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()  # 统计正确个数

    # 计算测试集损失和准确率
    avg_loss = test_loss / len(test_loader)
    accuracy = 100. * correct / total
    print(f'\nTest Set: Average Loss: {avg_loss:.4f}, Accuracy: {correct}/{t
    return accuracy

# ------------------------- 7. 启动训练和测试 -------------------------
print(f"Training on {device}...")
best_accuracy = 0.0
# for epoch in range(epochs):
#     train(model, train_loader, criterion, optimizer, epoch)
#     current_accuracy = test(model, test_loader, criterion)

#     # 保存最优模型
#     if current_accuracy > best_accuracy:
#         best_accuracy = current_accuracy
#         torch.save(model.state_dict(), 'results/alexnet_cifar10_best.pth')
#         print(f"Saved best model with accuracy: {best_accuracy:.2f}%\n")

# print(f"Training Finished! Best Accuracy: {best_accuracy:.2f}%")
```

```
Training on cpu...
Epoch [1/10], Batch [100/782], Loss: 2.1783
Epoch [1/10], Batch [200/782], Loss: 1.9728
Epoch [1/10], Batch [300/782], Loss: 1.8626
Epoch [1/10], Batch [400/782], Loss: 1.7660
Epoch [1/10], Batch [500/782], Loss: 1.6747
Epoch [1/10], Batch [600/782], Loss: 1.6193
Epoch [1/10], Batch [700/782], Loss: 1.5739

Test Set: Average Loss: 1.5522, Accuracy: 4296/10000 (42.96%)

Saved best model with accuracy: 42.96%

Epoch [2/10], Batch [100/782], Loss: 1.5317
Epoch [2/10], Batch [200/782], Loss: 1.4714
Epoch [2/10], Batch [300/782], Loss: 1.4165
Epoch [2/10], Batch [400/782], Loss: 1.4126
Epoch [2/10], Batch [500/782], Loss: 1.3548
Epoch [2/10], Batch [600/782], Loss: 1.3543
Epoch [2/10], Batch [700/782], Loss: 1.3433

Test Set: Average Loss: 1.1950, Accuracy: 5690/10000 (56.90%)

Saved best model with accuracy: 56.90%

Epoch [3/10], Batch [100/782], Loss: 1.2687
Epoch [3/10], Batch [200/782], Loss: 1.2437
Epoch [3/10], Batch [300/782], Loss: 1.2085
Epoch [3/10], Batch [400/782], Loss: 1.2221
Epoch [3/10], Batch [500/782], Loss: 1.1993
Epoch [3/10], Batch [600/782], Loss: 1.1686
Epoch [3/10], Batch [700/782], Loss: 1.1540

Test Set: Average Loss: 1.1346, Accuracy: 5916/10000 (59.16%)

Saved best model with accuracy: 59.16%

Epoch [4/10], Batch [100/782], Loss: 1.1187
Epoch [4/10], Batch [200/782], Loss: 1.0914
Epoch [4/10], Batch [300/782], Loss: 1.0966
Epoch [4/10], Batch [400/782], Loss: 1.1024
Epoch [4/10], Batch [500/782], Loss: 1.0780
Epoch [4/10], Batch [600/782], Loss: 1.0473
Epoch [4/10], Batch [700/782], Loss: 1.0496

Test Set: Average Loss: 0.9714, Accuracy: 6542/10000 (65.42%)

Saved best model with accuracy: 65.42%

Epoch [5/10], Batch [100/782], Loss: 1.0126
Epoch [5/10], Batch [200/782], Loss: 1.0070
Epoch [5/10], Batch [300/782], Loss: 1.0295
Epoch [5/10], Batch [400/782], Loss: 0.9779
Epoch [5/10], Batch [500/782], Loss: 1.0153
Epoch [5/10], Batch [600/782], Loss: 0.9828
Epoch [5/10], Batch [700/782], Loss: 0.9814
```

```
Test Set: Average Loss: 0.8626, Accuracy: 6936/10000 (69.36%)

Saved best model with accuracy: 69.36%

Epoch [6/10], Batch [100/782], Loss: 0.9538
Epoch [6/10], Batch [200/782], Loss: 0.9270
Epoch [6/10], Batch [300/782], Loss: 0.9163
Epoch [6/10], Batch [400/782], Loss: 0.9300
Epoch [6/10], Batch [500/782], Loss: 0.9254
Epoch [6/10], Batch [600/782], Loss: 0.9244
Epoch [6/10], Batch [700/782], Loss: 0.9179

Test Set: Average Loss: 0.8268, Accuracy: 7071/10000 (70.71%)

Saved best model with accuracy: 70.71%

Epoch [7/10], Batch [100/782], Loss: 0.8774
Epoch [7/10], Batch [200/782], Loss: 0.8996
Epoch [7/10], Batch [300/782], Loss: 0.8732
Epoch [7/10], Batch [400/782], Loss: 0.8915
Epoch [7/10], Batch [500/782], Loss: 0.8569
Epoch [7/10], Batch [600/782], Loss: 0.8615
Epoch [7/10], Batch [700/782], Loss: 0.8778

Test Set: Average Loss: 0.8019, Accuracy: 7248/10000 (72.48%)

Saved best model with accuracy: 72.48%

Epoch [8/10], Batch [100/782], Loss: 0.8443
Epoch [8/10], Batch [200/782], Loss: 0.8351
Epoch [8/10], Batch [300/782], Loss: 0.8468
Epoch [8/10], Batch [400/782], Loss: 0.8354
Epoch [8/10], Batch [500/782], Loss: 0.8241
Epoch [8/10], Batch [600/782], Loss: 0.8234
Epoch [8/10], Batch [700/782], Loss: 0.7899

Test Set: Average Loss: 0.8030, Accuracy: 7221/10000 (72.21%)

Epoch [9/10], Batch [100/782], Loss: 0.7887
Epoch [9/10], Batch [200/782], Loss: 0.7999
Epoch [9/10], Batch [300/782], Loss: 0.8060
Epoch [9/10], Batch [400/782], Loss: 0.7931
Epoch [9/10], Batch [500/782], Loss: 0.8028
Epoch [9/10], Batch [600/782], Loss: 0.7951
Epoch [9/10], Batch [700/782], Loss: 0.7935

Test Set: Average Loss: 0.7150, Accuracy: 7519/10000 (75.19%)

Saved best model with accuracy: 75.19%

Epoch [10/10], Batch [100/782], Loss: 0.7467
Epoch [10/10], Batch [200/782], Loss: 0.7944
Epoch [10/10], Batch [300/782], Loss: 0.7535
Epoch [10/10], Batch [400/782], Loss: 0.7797
Epoch [10/10], Batch [500/782], Loss: 0.7720
```

```
Epoch [10/10], Batch [600/782], Loss: 0.7614
Epoch [10/10], Batch [700/782], Loss: 0.7647

Test Set: Average Loss: 0.7327, Accuracy: 7531/10000 (75.31%)

Saved best model with accuracy: 75.31%

Training Finished! Best Accuracy: 75.31%
```

In [ ]:
```python
# ------------------------- 8. 单图推理演示（加载最优模型） --------------------
def infer_single_image(model_path, image_path, transform):
    # 加载模型
    model = AlexNet(num_classes=10).to(device)
    model.load_state_dict(torch.load(model_path))
    model.eval()

    # 预处理图像
    from PIL import Image
    image = Image.open(image_path).convert('RGB')
    image_tensor = transform(image).unsqueeze(0).to(device)  # 增加batch维度

    # 推理
    with torch.no_grad():
        output = model(image_tensor)
        _, predicted = output.max(1)
        predicted_class = classes[predicted.item()]

    # 显示图像和结果
    plt.imshow(image)
    plt.title(f'Predicted: {predicted_class}')
    plt.axis('off')
    plt.show()
```

In [10]:
```python
# 从test_batch中加载图片
import pickle
def load_cifar10_batch(batch_path):
    with open(batch_path, 'rb') as f:
        batch = pickle.load(f, encoding='bytes')
    images = batch[b'data']
    labels = batch[b'labels']
    images = images.reshape(-1, 3, 32, 32).astype(np.uint8)
    return images, labels
```

In [20]:
```python
images, labels = load_cifar10_batch('./datasets/cifar-10-batches-py/test_bat
img, label = images[0], labels[0]
# save the image
from PIL import Image
img1_pil = Image.fromarray(np.transpose(img, (1, 2, 0)))
img1_pil.save('test_image.jpg')
print(label)
```

```
3
```

In [15]:
```python
# 示例：用一张自己的图片测试（替换为你的图片路径）
# 注意：图片尺寸建议32×32（或自动缩放），类别需在CIFAR10中
infer_single_image('./results/alexnet_cifar10_best.pth', 'test_image.jpg', t
```

Predicted: cat