```
# 导入必要的库和模块
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt

import time
import tarfile

# 模拟耗时操作（如数据下载）
for i in range(100):
    time.sleep(0.1)  # 模拟耗时操作，每次休眠 0.1 秒
    print(f"当前进度: {i+1}/100")  # 输出当前进度，显示下载进度

# 解压 CIFAR-10 数据集
with tarfile.open('cifar-10-python.tar.gz', 'r:gz') as tar:
    tar.extractall('./data')  # 将压缩文件解压到./data 目录

print("解压完成")  # 提示解压完成

# 定义数据预处理转换
transform = transforms.Compose([
    transforms.ToTensor(),  # 将图像转换为 PyTorch 张量
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # 对图像进行标准化
])
```

```
# 加载 CIFAR-10 训练集
trainset = torchvision.datasets.CIFAR10(
    root='./data', # 指定数据集根目录
    train=True,    # 表示加载训练集
    download=False, # 数据集已经存在，不需要下载
    transform=transform # 应用预处理转换
)
trainloader = torch.utils.data.DataLoader(
    trainset,      # 加载的训练集
    batch_size=4,  # 每次加载 4 个样本
    shuffle=True,  # 在每个训练周期开始时打乱数据顺序
    num_workers=2  # 使用两个子进程来加载数据
)

# 加载 CIFAR-10 测试集
testset = torchvision.datasets.CIFAR10(
    root='./data', # 指定数据集根目录
    train=False,   # 表示加载测试集
    download=False, # 数据集已经存在，不需要下载
    transform=transform # 应用预处理转换
)
testloader = torch.utils.data.DataLoader(
    testset,       # 加载的测试集
    batch_size=4,  # 每次加载 4 个样本
    shuffle=False, # 不打乱测试数据顺序
    num_workers=2  # 使用两个子进程来加载数据
)
```

```
# 定义数据集的类别名称
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

# 定义 CNN 模型
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)  # 第一个卷积层，输入通道 3，输出通道 6，卷积核大小 5
        self.pool = nn.MaxPool2d(2, 2)  # 最大池化层，窗口大小 2，步长 2
        self.conv2 = nn.Conv2d(6, 16, 5) # 第二个卷积层，输入通道 6，输出通道 16，卷积核大小 5
        self.fc1 = nn.Linear(16 * 5 * 5, 120) # 第一个全连接层，输入特征数 16*5*5，输出 120
        self.fc2 = nn.Linear(120, 84)    # 第二个全连接层，输入 120，输出 84
        self.fc3 = nn.Linear(84, 10)     # 第三个全连接层，输入 84，输出 10（类别数）

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x))) # 应用第一个卷积层，激活函数 ReLU，然后池化
        x = self.pool(F.relu(self.conv2(x))) # 应用第二个卷积层，激活函数 ReLU，然后池化
        x = x.view(-1, 16 * 5 * 5)          # 将张量展平为一维向量
        x = F.relu(self.fc1(x))            # 应用第一个全连接层和 ReLU 激活
        x = F.relu(self.fc2(x))            # 应用第二个全连接层和 ReLU 激活
        x = self.fc3(x)                   # 应用第三个全连接层（输出层）
        return x

# 检查是否有可用的 GPU
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}") # 输出当前使用的设备（CPU 或 GPU）
```

```
net = Net().to(device) # 创建模型实例并移动到 GPU 上

# 定义损失函数和优化器
criterion = nn.CrossEntropyLoss() # 使用交叉熵损失函数
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9) # 使用随机梯度下降优化器

# 计时开始
start_time = time.time()

# 记录每个周期的准确率和损失
train_accuracies = []
train_losses = []
test_accuracies = []

# 训练模型
for epoch in range(30): # 训练 30 个周期
    running_loss = 0.0 # 初始化运行损失
    correct = 0       # 初始化正确预测计数
    total = 0         # 初始化总样本计数
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data # 获取输入数据和标签
        inputs, labels = inputs.to(device), labels.to(device) # 将数据移动到 GPU 上
        optimizer.zero_grad() # 清除梯度
        outputs = net(inputs) # 前向传播
        loss = criterion(outputs, labels) # 计算损失
        loss.backward() # 反向传播
        optimizer.step() # 更新参数
```

```
        running_loss += loss.item()  # 累加损失
        _, predicted = torch.max(outputs.data, 1)  # 获取预测结果
        total += labels.size(0)  # 累加总样本数
        correct += (predicted == labels).sum().item()  # 累加正确预测数

    # 记录训练准确率和损失
    train_loss = running_loss / (i + 1)
    train_accuracy = 100 * correct / total
    train_accuracies.append(train_accuracy)
    train_losses.append(train_loss)

    # 输出每个周期的损失和准确率
    print(f'Epoch {epoch + 1}, Loss: {train_loss}, Accuracy: {train_accuracy}%')

print('Finished Training')  # 提示训练完成

# 计时结束
end_time = time.time()

# 测试模型
correct = 0  # 初始化正确预测计数
total = 0    # 初始化总样本计数
class_correct = list(0. for i in range(10))  # 每个类别的正确预测计数
class_total = list(0. for i in range(10))    # 每个类别的总样本计数

with torch.no_grad():  # 禁用梯度计算
    for data in testloader:
        images, labels = data  # 获取测试数据和标签
        images, labels = images.to(device), labels.to(device)  # 将数据移动到 GPU 上
```

```
        outputs = net(images)  # 前向传播
        _, predicted = torch.max(outputs.data, 1)  # 获取预测结果
        total += labels.size(0)  # 累加总样本数
        correct += (predicted == labels).sum().item()  # 累加正确预测数

        # 统计每个类别的正确预测和总样本数
        c = (predicted == labels).squeeze()
        for i in range(len(labels)):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1

# 记录测试准确率
test_accuracy = 100 * correct / total
test_accuracies.append(test_accuracy)

# 输出测试准确率
print(f'Test Accuracy: {test_accuracy}%')

# 输出每个类别的准确率
for i in range(10):
    if class_total[i] != 0:
        print(f'Accuracy of {classes[i]} : {100 * class_correct[i] / class_total[i]}%')
    else:
        print(f'Accuracy of {classes[i]}: 0% (No samples in test set)')

# 输出训练时间
print(f'Training Time: {(end_time - start_time):.2f} seconds')
```

```
#绘制训练和测试准确率图表
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(test_accuracies, label='Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.title('Training and Test Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_losses, label='Train Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.legend()

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

```
当前进度: 86/100
当前进度: 87/100
当前进度: 88/100
当前进度: 89/100
当前进度: 90/100
当前进度: 91/100
当前进度: 92/100
当前进度: 93/100
当前进度: 94/100
当前进度: 95/100
当前进度: 96/100
当前进度: 97/100
当前进度: 98/100
当前进度: 99/100
当前进度: 100/100
解压完成
Using device: cuda:0
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/conv.py:456: UserWarning: Plan failed with a cudnnException: CUDNN_BACKEND_EX
ECUTION_PLAN_DESCRIPTOR: cudnnFinalize Descriptor Failed cudnn_status: CUDNN_STATUS_NOT_SUPPORTED (Triggered internally at ../aten/sr
c/ATen/native/cudnn/Conv_v8.cpp:919.)
  return F.conv2d(input, weight, bias, self.stride,
/usr/local/lib/python3.10/dist-packages/torch/autograd/graph.py:744: UserWarning: Plan failed with a cudnnException: CUDNN_BACKEND_EXE
CUTION_PLAN_DESCRIPTOR: cudnnFinalize Descriptor Failed cudnn_status: CUDNN_STATUS_NOT_SUPPORTED (Triggered internally at ../aten/src/
ATen/native/cudnn/Conv_v8.cpp:919.)
  return Variable._execution_engine.run_backward(  # Calls into the C++ engine to run the backward pass
Epoch 1, Loss: 1.707234972230196, Accuracy: 36.95%
```

```
Epoch 1, Loss: 1.707234972230196, Accuracy: 36.95%
Epoch 2, Loss: 1.32760055034101, Accuracy: 52.248%
Epoch 3, Loss: 1.191439954866171, Accuracy: 57.694%
Epoch 4, Loss: 1.0987366149416566, Accuracy: 61.216%
Epoch 5, Loss: 1.0293424591638147, Accuracy: 63.542%
Epoch 6, Loss: 0.9754010478535108, Accuracy: 65.642%
Epoch 7, Loss: 0.9296682361439988, Accuracy: 67.01%
Epoch 8, Loss: 0.8955322258555144, Accuracy: 68.242%
Epoch 9, Loss: 0.8608931998819671, Accuracy: 69.54%
Epoch 10, Loss: 0.8257967552061821, Accuracy: 70.808%
Epoch 11, Loss: 0.8011284896801458, Accuracy: 71.554%
Epoch 12, Loss: 0.7771735379579047, Accuracy: 72.342%
Epoch 13, Loss: 0.7628289761277102, Accuracy: 73.104%
Epoch 14, Loss: 0.7449441306237015, Accuracy: 73.592%
Epoch 15, Loss: 0.7250168524540745, Accuracy: 74.446%
Epoch 16, Loss: 0.7078402423407324, Accuracy: 74.796%
Epoch 17, Loss: 0.6958120898552891, Accuracy: 75.332%
Epoch 18, Loss: 0.6859128045611351, Accuracy: 75.58%
Epoch 19, Loss: 0.6705203574449545, Accuracy: 76.162%
Epoch 20, Loss: 0.6574327594212092, Accuracy: 76.628%
Epoch 21, Loss: 0.6528054773030907, Accuracy: 76.958%
Epoch 22, Loss: 0.64652872858853, Accuracy: 77.324%
Epoch 23, Loss: 0.6441116073382187, Accuracy: 77.404%
Epoch 24, Loss: 0.6357097702718358, Accuracy: 77.438%
Epoch 25, Loss: 0.6320777602239372, Accuracy: 77.8%
Epoch 26, Loss: 0.6224390435715983, Accuracy: 78.32%
Epoch 27, Loss: 0.6178601043018169, Accuracy: 78.414%
Epoch 28, Loss: 0.6206121492371234, Accuracy: 78.418%
Epoch 29, Loss: 0.6153990050262731, Accuracy: 78.526%
Epoch 30, Loss: 0.6108759035744082, Accuracy: 78.898%
Finished Training
Test Accuracy: 60.56%
Accuracy of plane : 55.0%
Accuracy of car : 74.4%
Accuracy of bird : 45.6%
Accuracy of cat : 42.5%
Accuracy of deer : 59.6%
Accuracy of dog : 43.6%
Accuracy of frog : 70.1%
Accuracy of horse : 69.3%
Accuracy of ship : 78.2%
Accuracy of truck : 67.3%
Training Time: 3626.88 seconds
```