Lab4-Pytorch + CNN

1. 实验概览

1.1 实验4-1. PyTorch入门-CIFAR-10图片分类

实验的目的是利用深度学习对CIFAR-10数据集进行分类。在这次实验中，采用简单的模型ResNet20作为模型，通过采用不同的训练策略，如调整学习率、训练轮次等，来提高模型的性能，最终达到较好的分类效果。

1.2 实验4-2. Search by CNN features

实验的目的是利用Pytorch官方提供的现成的预训练好的深度模型，如VGG19、ResNet50等，提取图像特征，并通过计算得到的特征向量间的欧氏距离来进行图像检索。在本实验中，一共使用了三种模型，分别是VGG19、ResNet50和AlexNet，得到检索结果，完成以图搜图的任务。

2. 解决思路及关键代码

2.1 实验4-1. PyTorch入门-CIFAR-10图片分类

2.1.1 代码补充实现

在`exp2.py`中，我补充了`test`函数中的内容，使其能够计算模型在测试集上的损失与准确率，代码如下：

```
        for batch_idx, (inputs, targets) in enumerate(testloader):
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()
        acc = 100. * correct / total
        test_acc.append(acc)
```

2.1.2 训练策略调整

对`lr`变量进行了调整，将其转换成列表`lrs`，存储了不同的学习率，以便于在训练过程中进行学习率调整。对训练策略也进行调整：

- 加载已经预训练了5轮的pretrain_model.pth

```
restore_model_path = 'pretrained/ckpt_4_acc_63.320000.pth'
model.load_state_dict(torch.load(restore_model_path)['net'])
```

- 使用0.1的学习率进行5轮训练，每轮训练后进行测试。
- 使用0.01的学习率再进行5轮训练，每轮训练后进行测试。

```
for i in range(len(lrs)):
    print(f"========== Training with learning rate {lrs[i]} ==========")
    for epoch in range(start_epoch, end_epoch + 1):
        train(epoch + i * (end_epoch + 1), lr=lrs[i])
        test(epoch + i * (end_epoch + 1))
```

最后的模型在测试集上的准确率达到了80%以上

2.2 实验4-2. Search by CNN features

2.2.1 提取图像特征

在这一步中，使用了三种在ImageNet上预训练好的模型，分别是VGG19、ResNet50和alexnet，对图像进行特征提取。这三个模型的代码分别位于extract_feature_vgg19.py、extract_feature_resnet50.py和extract_feature_alexnet.py中。

通过Pytorch官方提供的模型数据，首先导入模型，然后通过定义好的图片归一化操作和预处理方式得到预处理后的图像。（这里三个模型的归一化参数均相同）

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                  std=[0.229, 0.224, 0.225])
trans = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    normalize,
])
```

之后，我找到了这三种模型的倒数第二层avgpool，通过仿照示例中的方法，得到了图像的特征，并且将这三种模型的输出特征向量保存到了save_path中。

```
def features_resnet50(x):
    x = model.conv1(x)
    x = model.bn1(x)
    x = model.relu(x)
    x = model.maxpool(x)
    x = model.layer1(x)
    x = model.layer2(x)
    x = model.layer3(x)
    x = model.layer4(x)
    x = model.avgpool(x)
    return x
```

```
def features_vgg19(x):
    x = model.features(x)
    x = model.avgpool(x)
    return x
```

```
def features_alexnet(x):
    x = model.features(x)
    x = model.avgpool(x)
    return x
```

2.2.2 计算欧氏距离并检索

在这一步中，我首先通过2.2.1中的方法，对数据库Dataset目录中的所有图片进行特征提取，并将提取的特征向量分别保存到Feature_vgg19、Feature_resnet50和Feature_alexnet目录中。以vgg19为例：

```
def extract_dataset_feature_vgg19(dataset_path, save_path):
    for i in range(1, 53):
        input_image_path = dataset_path + f'/{i}.jpg'
        save_path_i = save_path + f'/{i}.npy'
        extract_features_vgg19(input_image_path, save_path_i)
```

之后通过函数`cal_simularity`计算两个特征向量之间的欧氏距离：首先将两个特征向量使用`.flatten()`展平，然后进行归一化并计算欧式距离。

```python
def cal_similarity(feature1, feature2):
    feature1 = feature1.flatten()
    feature2 = feature2.flatten()
    # 归一化
    feature1 = feature1 / np.linalg.norm(feature1)
    feature2 = feature2 / np.linalg.norm(feature2)
    # 欧氏距离
    distance = np.linalg.norm(feature1 - feature2)
    return distance
```

最后，对每一张在`Query_image`目录中的待检索图片，通过2.2.1中的方法分别计算三种特征向量并保存到`Query_feature`目录中，然后通过函数`query_k_image()`对每一张待检索图片，计算其与数据库中所有图片的欧氏距离并返回相似度的前k张图片的标签与距离。

```python
def query_k_image(query_image_feature_path, dataset_path, k=5):
    query_feature = np.load(query_image_feature_path)
    distances = []
    indices = []
    for i in range(1, 53):
        feature_path = f'{dataset_path}/{i}.npy'
        try:
            feature = np.load(feature_path)
        except FileNotFoundError:
            print(f'Feature {i} not found!')
            continue
        dist = cal_similarity(query_feature, feature)
        distances.append(dist)
        indices.append(i)
    distances = np.array(distances)
    indices = np.array(indices)
    # 从小到大排序
    sorted_indices = np.argsort(distances)
    topk_index = indices[sorted_indices[:k]]
    topk_distance = distances[sorted_indices[:k]]

    return topk_index, topk_distance
```

使用 matplotlib 将检索结果可视化，并保存到`Output`中。

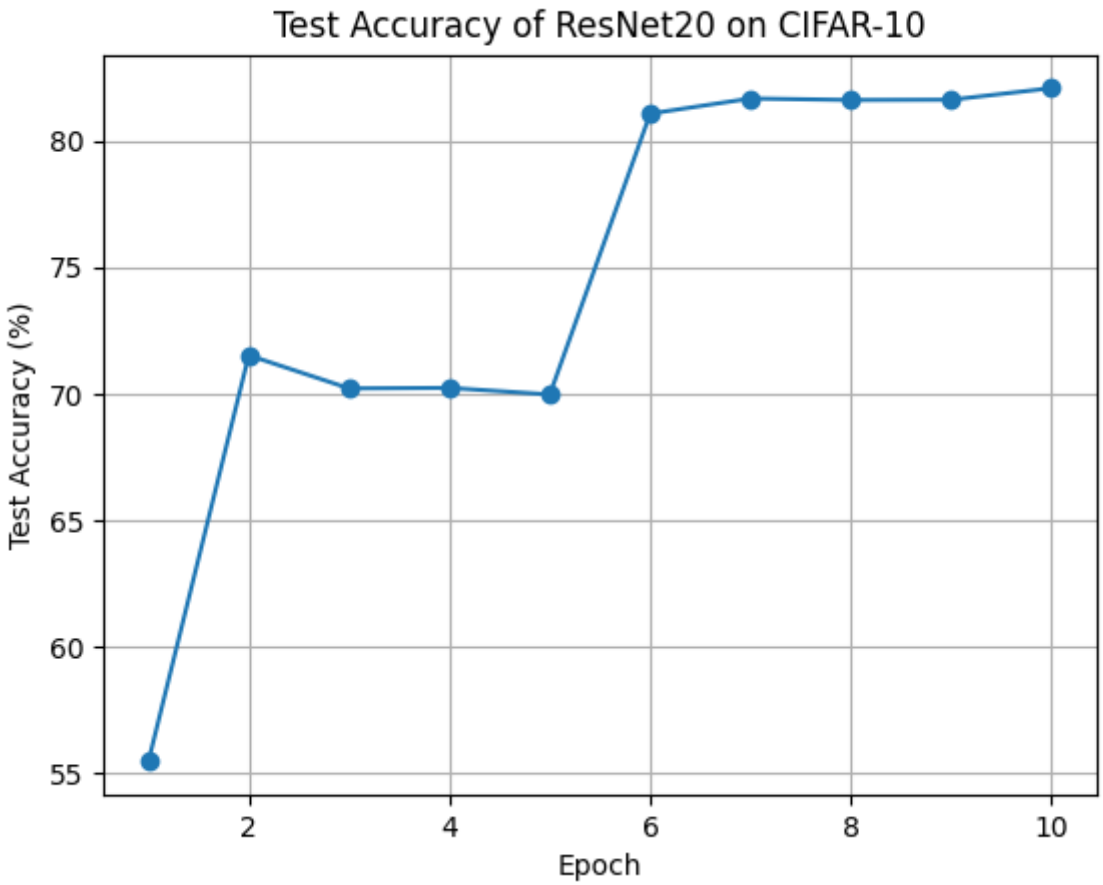
```
def draw(output_path, query_image_path, model, data, i):
    plt.figure(figsize=(28, 5))
    plt.subplot(1, 6, 1)
    query_image = plt.imread(query_image_path)
    plt.imshow(query_image)
    plt.axis('off')
    plt.title('Query Image')
    for j, (idx, sim) in enumerate(data):
        print('Index:', idx, 'Similarity:', sim)
        image_path = f'./Dataset/{idx}.jpg'
        image = plt.imread(image_path)
        plt.subplot(1, 6, j + 2)
        plt.imshow(image)
        plt.axis('off')
        plt.title(f'Top {j + 1} similar: Index: {idx}, Similarity: {sim:.2f}')
        plt.axis('off')
        plt.suptitle(f'{model}')
        plt.tight_layout()
        plt.savefig(f'{output_path}/{model}_{i}.png')


print('The query image is:', query_image_path)
print('The top 5 most similar images using ResNet50:')
draw(output_path, query_image_path, 'ResNet50', zip(min_k_index_resnet50,
min_k_similarity_resnet50), i)
print('The top 5 most similar images using VGG19:')
draw(output_path, query_image_path, 'VGG19', zip(min_k_index_vgg19,
min_k_similarity_vgg19), i)
print('The top 5 most similar images using AlexNet:')
draw(output_path, query_image_path, 'AlexNet', zip(min_k_index_alexnet,
min_k_similarity_alexnet), i)
```

3. 实验结果及分析

3.1 实验4-1. PyTorch入门-CIFAR-10图片分类

在实验4.1中，通过调整学习率和训练策略，对ResNet20模型进行了训练，最终在测试集上的准确率达到了80%以上。其中test acc的变化趋势如下：



由上图可以看出，当学习率为0.1时，模型在测试集上的准确率在5轮训练之后达到了70%左右，这是因为学习率较大，模型的收敛速度较快，但是容易陷入局部最优解。当学习率调整为0.01时，模型在测试集上的准确率在之后5轮训练中逐步稳定，最终达到了82.10%。

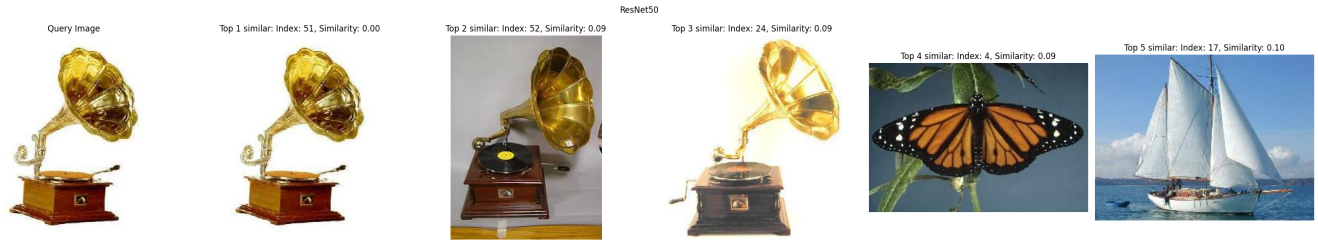
3.2 实验4-2. Search by CNN features

在实验4.2中，我通过使用三种在ImageNet上预训练好的模型，分别是VGG19、ResNet50和alexnet，对图像进行特征提取，并通过计算得到的特征向量间的欧氏距离来进行图像检索。最终的检索结果如下：(从待检索图像左侧开始，依此为相似度第1~5高的图像)

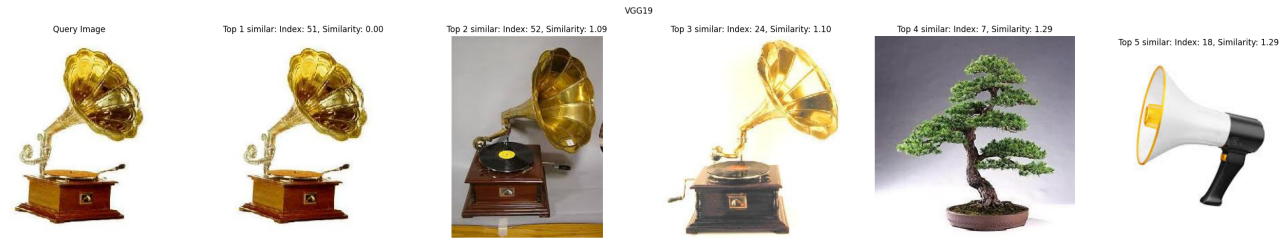
目标图像1

检索结果：

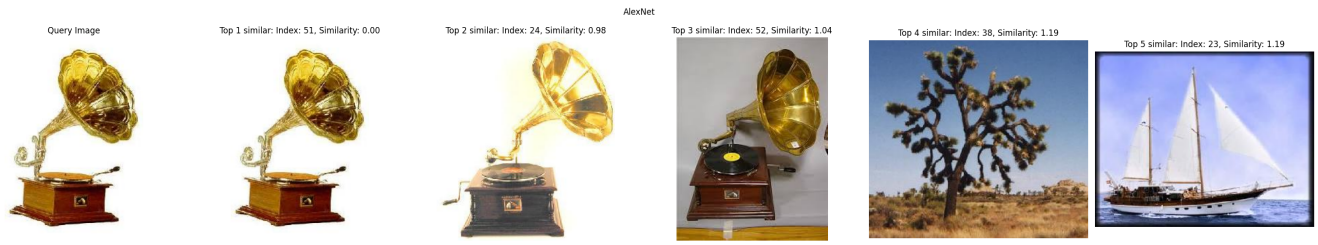
ResNet50 Similarity : 0.00 0.09 0.09 0.09 0.10



Vgg19 Similarity : 0.00 1.09 1.10 1.29 1.29



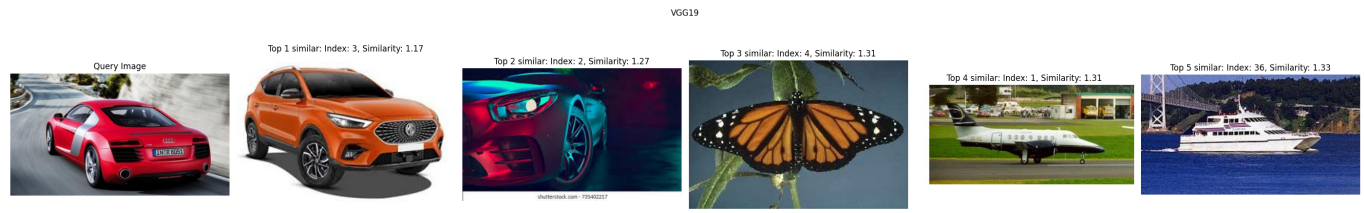
AlexNet Similarity : 0.00 0.98 1.04 1.19 1.19



目标图像2

ResnNet50 Similarity : 0.09 0.09 0.09 0.09 0.09

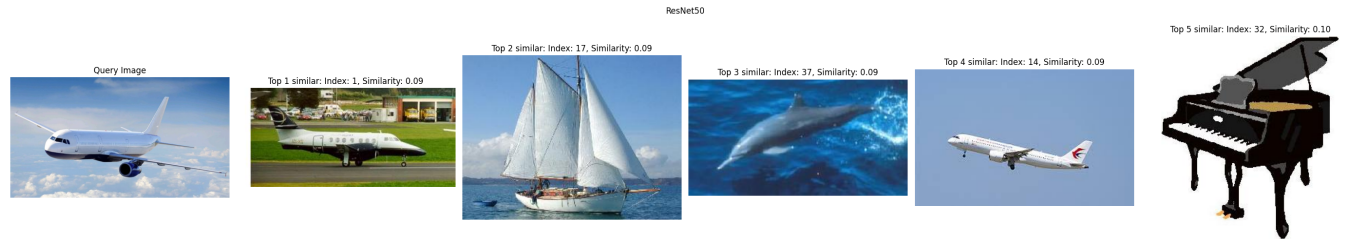
Vgg19 Similarity : 1.17 1.27 1.31 1.31 1.33



AlexNet Similarity : 1.08 1.21 1.21 1.22 1.22

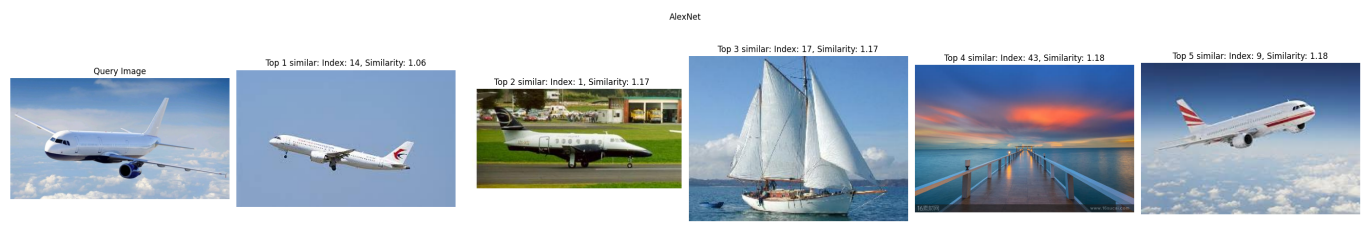
目标图像3

ResNet50 Similarity : 0.09 0.09 0.09 0.09 0.10



Vgg19 Similarity : 1.00 1.13 1.17 1.25 1.27

AlexNet Similarity : 1.06 1.17 1.17 1.18 1.18



从上述结果可以看出，VGG19和AlexNet在检索结果上表现较好，ResNet50的检索结果相对较差，体现在其检索结果存在很多Similarity小而相近，对图像的区分度不够高，模型的效果不够稳定。

3.3 实验总结

本次实验中，我通过Pytorch对CIFAR-10数据集进行分类，并通过调整学习率和训练策略，最终在测试集上的准确率达到了80%以上。并且通过使用Pytorch提取图像特征并检索，最终得到了以图搜图的结果。在实验中，我学习到了如何通过调整学习率和训练策略来提高模型的性能，同时也学习到了如何使用Pytorch提取图像特征，并通过计算特征向量间的欧氏距离来进行图像检索。

4. 思考题解答

- **Train acc 和 Test acc 有什么关联和不同？在 lr 从 0.1 变到 0.01 后，acc 发生了什么变化？为什么?**

  Train acc 反映了模型学得多好，Test acc 反映了模型学得是否通用。

  Train acc 和 Test acc 都是用来评估模型性能的指标，Train acc 是模型在训练集上的准确率，而 Test acc 是模型在测试集上的准确率。Train acc 反映了模型在训练集上的拟合能力，而 Test acc 反映了模型在未知数据上的泛化能力。在训练过程中，Train acc 通常会随着训练轮次的增加而逐渐提高，而 Test acc 通常会在一定轮次后达到一个稳定值，Train acc 一般会大于 Test acc。

  大学习率用于快速逼近最优区，小学习率用于稳定收敛、细调参数。

  在 lr 从 0.1 变到 0.01 后，模型在测试集上的准确率发生了变化，从准确率变化较大到逐渐稳定。这是因为当学习率较大时，模型的收敛速度较快，但是容易陷入局部最优解，导致模型的准确率波动较大。而当学习率较小时，模型的收敛速度较慢，但是模型的泛化能力较强，模型的准确率会逐渐稳定。

5. 附录

5.1 实验4.1

`exp2.py`：

```python
# SJTU EE208

'''Train CIFAR-10 with PyTorch.'''
import os
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.backends.cudnn as cudnn
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms

from models import resnet20

start_epoch = 0
end_epoch = 4
lrs = [0.1, 0.01]
test_acc = []

# Data pre-processing, DO NOT MODIFY
print('==> Preparing data..')
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    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)),
])

trainset = torchvision.datasets.CIFAR10(
    root='./data', train=True, download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True)

testset = torchvision.datasets.CIFAR10(
    root='./data', train=False, download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=128, shuffle=False)

classes = ("airplane", "automobile", "bird", "cat",
           "deer", "dog", "frog", "horse", "ship", "truck")

# Model
print('==> Building model..')
model = resnet20()
# If you want to restore training (instead of training from beginning),
```

```
# you can continue training based on previously-saved models
# by uncommenting the following two lines.
# Do not forget to modify start_epoch and end_epoch.
restore_model_path = 'pretrained/ckpt_4_acc_63.320000.pth'
model.load_state_dict(torch.load(restore_model_path)['net'])

# A better method to calculate loss
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=lrs[0], weight_decay=5e-4)


def train(epoch, lr):
    model.train()
    train_loss = 0
    correct = 0
    total = 0
    optimizer.param_groups[0]['lr'] = lr
    for batch_idx, (inputs, targets) in enumerate(trainloader):
        optimizer.zero_grad()
        outputs = model(inputs)
        # The outputs are of size [128x10].
        # 128 is the number of images fed into the model
        # (yes, we feed a certain number of images into the model at the same time,
        # instead of one by one)
        # For each image, its output is of length 10.
        # Index i of the highest number suggests that the prediction is classes[i].
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()
        print('Epoch [%d] Batch [%d/%d] Loss: %.3f | Traininig Acc: %.3f%% (%d/%d)'
              % (epoch, batch_idx + 1, len(trainloader), train_loss / (batch_idx + 1),
                 100. * correct / total, correct, total))


def test(epoch):
    print('==> Testing...')
    model.eval()
    test_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        ##### TODO: calc the test accuracy #####
        # Hint: You do not have to update model parameters.
        #       Just get the outputs and count the correct predictions.
        #       You can turn to `train` function for help.
        for batch_idx, (inputs, targets) in enumerate(testloader):
            outputs = model(inputs)
```

```
            loss = criterion(outputs, targets)
            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()
        acc = 100. * correct / total
        test_acc.append(acc)
        ########################################
    # Save checkpoint.
    print('Test Acc: %f' % acc)
    print('Saving..')
    state = {
        'net': model.state_dict(),
        'acc': acc,
        'epoch': epoch,
    }
    if not os.path.isdir('checkpoint'):
        os.mkdir('checkpoint')
    torch.save(state, './checkpoint/ckpt_%d_acc_%f.pth' % (epoch, acc))


for i in range(len(lrs)):
    print(f"========== Training with learning rate {lrs[i]} ==========")
    for epoch in range(start_epoch, end_epoch + 1):
        train(epoch + i * (end_epoch + 1), lr=lrs[i])
        test(epoch + i * (end_epoch + 1))

# 绘制 test acc 曲线
plt.plot(range(1, len(test_acc) + 1), test_acc, marker='o')
plt.xlabel('Epoch')
plt.ylabel('Test Accuracy (%)')
plt.title('Test Accuracy of ResNet20 on CIFAR-10')
plt.grid(True)
plt.savefig('./test_acc.png')

# 保存最终模型
state = {
    'net': model.state_dict(),
    'acc': test_acc[-1],
    'epoch': start_epoch + len(test_acc),
}
torch.save(state, './final.pth')
print("Final model saved as final.pth")
```

5.2 实验4.2

1. `main.py`

```python
import os
import numpy as np
import matplotlib.pyplot as plt
from extract_feature_resnet50 import extract_features_resnet50
from extract_feature_vgg19 import extract_features_vgg19
from extract_feature_alexnet import extract_features_alexnet

def extract_dataset_feature_resnet50(dataset_path, save_path):
    for i in range(1, 53):
        input_image_path = dataset_path + f'/{i}.jpg'
        save_path_i = save_path + f'/{i}.npy'
        extract_features_resnet50(input_image_path, save_path_i)

def extract_dataset_feature_vgg19(dataset_path, save_path):
    for i in range(1, 53):
        input_image_path = dataset_path + f'/{i}.jpg'
        save_path_i = save_path + f'/{i}.npy'
        extract_features_vgg19(input_image_path, save_path_i)

def extract_dataset_feature_alexnet(dataset_path, save_path):
    for i in range(1, 53):
        input_image_path = dataset_path + f'/{i}.jpg'
        save_path_i = save_path + f'/{i}.npy'
        extract_features_alexnet(input_image_path, save_path_i)

def cal_similarity(feature1, feature2):
    feature1 = feature1.flatten()
    feature2 = feature2.flatten()
    # 归一化
    feature1 = feature1 / np.linalg.norm(feature1)
    feature2 = feature2 / np.linalg.norm(feature2)
    # 欧氏距离
    distance = np.linalg.norm(feature1 - feature2)
    return distance

def query_k_image(query_image_feature_path, dataset_path, k=5):
    query_feature = np.load(query_image_feature_path)
    distances = []
    indices = []
    for i in range(1, 53):
        feature_path = f'{dataset_path}/{i}.npy'
        try:
            feature = np.load(feature_path)
        except FileNotFoundError:
            print(f'Feature {i} not found!')
            continue
        dist = cal_similarity(query_feature, feature)
        distances.append(dist)
        indices.append(i)
    distances = np.array(distances)
```

```
    indices = np.array(indices)
    # 从小到大排序
    sorted_indices = np.argsort(distances)
    topk_index = indices[sorted_indices[:k]]
    topk_distance = distances[sorted_indices[:k]]

    return topk_index, topk_distance

def draw(output_path, query_image_path, model, data, i):
    plt.figure(figsize=(28, 5))
    plt.subplot(1, 6, 1)
    query_image = plt.imread(query_image_path)
    plt.imshow(query_image)
    plt.axis('off')
    plt.title('Query Image')
    for j, (idx, sim) in enumerate(data):
        print('Index:', idx, 'Similarity:', sim)
        image_path = f'./Dataset/{idx}.jpg'
        image = plt.imread(image_path)
        plt.subplot(1, 6, j + 2)
        plt.imshow(image)
        plt.axis('off')
        plt.title(f'Top {j + 1} similar: Index: {idx}, Similarity: {sim:.2f}')
        plt.axis('off')
        plt.suptitle(f'{model}')
        plt.tight_layout()
        plt.savefig(f'{output_path}/{model}_{i}.png')

def main():
    os.makedirs('./Feature_resnet50', exist_ok=True)
    os.makedirs('./Feature_vgg19', exist_ok=True)
    os.makedirs('./Feature_alexnet', exist_ok=True)
    os.makedirs('./Query_feature', exist_ok=True)
    os.makedirs('./Output', exist_ok=True)

    dataset_path = './Dataset'
    save_path_resnet50 = './Feature_resnet50'
    save_path_vgg19 = './Feature_vgg19'
    save_path_alexnet = './Feature_alexnet'
    output_path = './Output'

    extract_dataset_feature_resnet50(dataset_path, save_path_resnet50)
    extract_dataset_feature_vgg19(dataset_path, save_path_vgg19)
    extract_dataset_feature_alexnet(dataset_path, save_path_alexnet)

    query_images = ['./1.jpg', './2.jpg', './3.jpg']
    for i, query_image_path in enumerate(query_images):
        query_image_path = f'./Query_image/{i+1}.jpg'

        print('Query the most similar image using resnet50!')
        # Extract features of the query image using resnet50
```

```
        extract_features_resnet50(query_image_path,
f'./Query_feature/query_resnet50_{i}.npy')
        # Query the most similar image using resnet50
        query_image_feature_path = f'./Query_feature/query_resnet50_{i}.npy'
        min_k_index_resnet50, min_k_similarity_resnet50 =
        query_k_image(query_image_feature_path, './Feature_resnet50', k=5)

        print('Query the most similar image using vgg19!')
        extract_features_vgg19(query_image_path,
f'./Query_feature/query_vgg19_{i}.npy')
        query_image_feature_path = f'./Query_feature/query_vgg19_{i}.npy'
        min_k_index_vgg19, min_k_similarity_vgg19 =
        query_k_image(query_image_feature_path, './Feature_vgg19', k=5)

        print('Query the most similar image using alexnet!')
        extract_features_alexnet(query_image_path,
f'./Query_feature/query_alexnet_{i}.npy')
        query_image_feature_path = f'./Query_feature/query_alexnet_{i}.npy'
        min_k_index_alexnet, min_k_similarity_alexnet =
        query_k_image(query_image_feature_path, './Feature_alexnet', k=5)

        print('The query image is:', query_image_path)
        print('The top 5 most similar images using ResNet50:')
        draw(output_path, query_image_path, 'ResNet50', zip(min_k_index_resnet50,
min_k_similarity_resnet50), i)
        print('The top 5 most similar images using VGG19:')
        draw(output_path, query_image_path, 'VGG19', zip(min_k_index_vgg19,
min_k_similarity_vgg19), i)
        print('The top 5 most similar images using AlexNet:')
        draw(output_path, query_image_path, 'AlexNet', zip(min_k_index_alexnet,
min_k_similarity_alexnet), i)

if __name__ == '__main__':
    main()
```

2. `extract_feature_resnet50.py`

```python
# SJTU EE208

import time
import numpy as np
import torch
import torchvision.transforms as transforms
from torchvision.datasets.folder import default_loader

print('Load model: ResNet50')
model = torch.hub.load('pytorch/vision', 'resnet50', pretrained=True)

normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
trans = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    normalize,
])

def features_resnet50(x):
    x = model.conv1(x)
    x = model.bn1(x)
    x = model.relu(x)
    x = model.maxpool(x)
    x = model.layer1(x)
    x = model.layer2(x)
    x = model.layer3(x)
    x = model.layer4(x)
    x = model.avgpool(x)
    return x

def extract_features_resnet50(input_image_path, save_path):
    print('Prepare image data!')
    test_image = default_loader(input_image_path)
    input_image = trans(test_image)
    input_image = torch.unsqueeze(input_image, 0)
    print('Extract features!')

    start = time.time()
    image_feature = features_resnet50(input_image)
    image_feature = image_feature.detach().numpy()
    print('Time for extracting features: {:.2f}'.format(time.time() - start))

    print('Save features!')
    np.save(save_path, image_feature)
```

3. `extract_feature_vgg19.py`

```
import time
import numpy as np
import torch
import torchvision.transforms as transforms
from torchvision.datasets.folder import default_loader

print('Load model: Vgg19')
model = torch.hub.load('pytorch/vision', 'vgg19', pretrained=True)

normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
trans = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    normalize,
])

def features_vgg19(x):
    x = model.features(x)
    x = model.avgpool(x)
    return x

def extract_features_vgg19(input_image_path, save_path):
    print('Prepare image data!')
    test_image = default_loader(input_image_path)
    input_image = trans(test_image)
    input_image = torch.unsqueeze(input_image, 0)
    print('Extract features!')

    start = time.time()
    image_feature = features_vgg19(input_image)
    image_feature = image_feature.detach().numpy()
    print('Time for extracting features: {:.2f}'.format(time.time() - start))

    print('Save features!')
    np.save(save_path, image_feature)
```

4. `extract_feature_alexnet.py`

```python
import time
import numpy as np
import torch
import torchvision.transforms as transforms
from torchvision.datasets.folder import default_loader

print('Load model: AlexNet')

model = torch.hub.load('pytorch/vision', 'alexnet', pretrained=True)

normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
trans = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    normalize,
])

def features_alexnet(x):
    x = model.features(x)
    x = model.avgpool(x)
    return x

def extract_features_alexnet(input_image_path, save_path):
    print('Prepare image data!')
    test_image = default_loader(input_image_path)
    input_image = trans(test_image)
    input_image = torch.unsqueeze(input_image, 0)
    print('Extract features!')

    start = time.time()
    image_feature = features_alexnet(input_image)
    image_feature = image_feature.detach().numpy()
    print('Time for extracting features: {:.2f}'.format(time.time() - start))

    print('Save features!')
    np.save(save_path, image_feature)
```