Lab5-Pytorch+CNN

1. 实验概览

1.1 实验5.1 CIFAR-10数据集分类

这次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，如学习率调整、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5.2 使用Pytorch提取图像特征并检索

这个实验的目的是利用Pytorch官方提供的现成的预训练好的深度模型，如VGG16、ResNet50等，提取图像特征，并通过计算得到的特征向量间的夹角来进行图像检索。在这个实验中，一共使用了三种模型，分别是VGG16、ResNet50和IncentionV3，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5.1 CIFAR-10数据集分类

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：

```
##### 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
########################################
```

并且，我还对`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。并且对训练策略也进行了调整：

- 首先使用0.1的学习率进行训练，一共训练两次，一次中进行两轮训练并在每轮训练后进行测试。
- 然后使用0.01的学习率进行训练，一共训练一次，一次中进行两轮训练并在每次训练后进行测试。
- 一共训练30轮，最后模型在测试集上的准确率达到了86%以上。

```
lrs = [0.1, 0.1, 0.01]

for i in range(len(lrs)):
    for epoch in range(start_epoch, end_epoch + 1):
        train(epoch, lr=lrs[i])
        test(epoch)
        train(epoch, lr=lrs[i])
        test(epoch)
```

2.2 实验5.2 使用Pytorch提取图像特征并检索

2.2.1 提取图像特征

在这一步中，我分别使用了三种在ImageNet上预训练好的模型，分别是VGG16、ResNet50和InceptionV3，对图像进行特征提取。这三个模型的代码分别位于`extract_feature_vgg.py`、`extract_feature_resnet.py`和`extract_feature_inception.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）

之后，我找到了这三种模型的倒数第二层，分别为`avgpool`、`avgpool`和`Mixed_7c`，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。

```
'''
    Extract feature from pretrained model-Inception_V3
    File: extract_feature_inception.py
'''

print('Load model: InceptionV3')
model = torch.hub.load('pytorch/vision', 'inception_v3', pretrained=True)

normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])

trans = transforms.Compose([
    transforms.Resize(299),
    transforms.CenterCrop(299),
    transforms.ToTensor(),
    normalize,
])

def features_inception(x):
    x = model.Conv2d_1a_3x3(x)
    x = model.Conv2d_2a_3x3(x)
    x = model.Conv2d_2b_3x3(x)
    x = model.Conv2d_3b_1x1(x)
    x = model.Conv2d_4a_3x3(x)
    x = model.Mixed_5b(x)
    x = model.Mixed_5c(x)
    x = model.Mixed_5d(x)
    x = model.Mixed_6a(x)
    x = model.Mixed_6b(x)
    x = model.Mixed_6c(x)
    x = model.Mixed_6d(x)
```

```
    x = model.Mixed_6e(x)
    x = model.Mixed_7a(x)
    x = model.Mixed_7b(x)
    x = model.Mixed_7c(x)
    return x


def extract_features_inception(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_inception(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)


'''
    Extract feature from pretrained model-vgg16
    File: extract_feature_vgg.py
'''


print('Load model: VGG16')
model = torch.hub.load('pytorch/vision', 'vgg16', 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_vgg16(x):
    x = model.features(x)
    x = model.avgpool(x)
    return x

def extract_features_vgg16(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_vgg16(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)


'''
    Extract feature from pretrained model-resnet50
    File: extract_feature_resnet50.py
'''

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)
```

2.2.2 计算夹角并检索

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_inception`、`Feature_vgg16`和`Feature_resnet50`目录中。

之后通过函数`cal_simularity`计算两个特征向量之间的夹角：首先将两个特征向量使用`.flatten()`展平，然后通过`np.dot`计算两个向量的点积，最后通过`np.linalg.norm`计算两个向量的模长，最终得到夹角。

最后，先对每一张在`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夹角，并返回夹角最小的前k张图片的标签与夹角。并使用 matplotlib 将检索结果可视化，并保存到`Output`中。

```
# Calculate the similarity between two features using angle
def cal_similarity(feature1, feature2):
    feature1 = feature1.flatten()
    feature2 = feature2.flatten()
    similarity = np.dot(feature1, feature2) / (np.linalg.norm(feature1) *
np.linalg.norm(feature2))
    return np.arccos(similarity) / np.pi * 180


# Extract features of all images in the dataset using ResNet50
def extract_dataset_feature_resnet50(dataset_path, save_path):
    for i in range(1, 51):
        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)


# Extract features of all images in the dataset using VGG16
def extract_dataset_feature_vgg16(dataset_path, save_path):
    for i in range(1, 51):
        input_image_path = dataset_path + f'/{i}.jpg'
        save_path_i = save_path + f'/{i}.npy'
        extract_features_vgg16(input_image_path, save_path_i)


def extract_dataset_feature_inception(dataset_path, save_path):
    for i in range(1, 51):
        input_image_path = dataset_path + f'/{i}.jpg'
        save_path_i = save_path + f'/{i}.npy'
        extract_features_inception(input_image_path, save_path_i)


# Query the k most similar images in the dataset
def query_k_image(query_image_feature_path, dataset_path, k=5):
    query_feature = np.load(query_image_feature_path)

    similarity = []
    index = []

    for i in range(1, 51):
        feature_path = dataset_path + f'/{i}.npy'
```

```
        feature = np.load(feature_path)
        if feature is None:
            print('Feature not found!')
            continue

        sim = cal_similarity(query_feature, feature)
        similarity.append(sim)
        index.append(i)

    similarity = np.array(similarity)
    index = np.array(index)

    min_index = index[np.argsort(similarity)[:k]]
    min_similarity = similarity[np.argsort(similarity)[:k]]

    return min_index, min_similarity

def main():
    os.makedirs('./Feature_resnet50', exist_ok=True)
    os.makedirs('./Feature_vgg16', exist_ok=True)
    os.makedirs('./Feature_inception', 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_vgg16 = './Feature_vgg16'
    save_path_inception = './Feature_inception'
    output_path = './Output'

    # If the features of the dataset have been extracted, comment the following
three lines
    extract_dataset_feature_resnet50(dataset_path, save_path_resnet50)
    extract_dataset_feature_vgg16(dataset_path, save_path_vgg16)
    extract_dataset_feature_inception(dataset_path, save_path_inception)

    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('\n')
        print('Query the most similar image using resnet50!')

        # Extract features of the query image using resnet50
        extract_features_resnet50(query_image_path,
'./Query_feature/query_resnet50.npy')

        # Query the most similar image using resnet50
        query_image_feature_path = './Query_feature/query_resnet50.npy'
        min_k_index_resnet50, min_k_similarity_resnet50 =
query_k_image(query_image_feature_path, './Feature_resnet50', k=5)

        print('\n')
        print('Query the most similar image using vgg16!')

        # Extract features of the query image using vgg16
```

```
        extract_features_vgg16(query_image_path,
'./Query_feature/query_vgg16.npy')

        # Query the most similar image using vgg16
        query_image_feature_path = './Query_feature/query_vgg16.npy'
        min_k_index_vgg16, min_k_similarity_vgg16 =
query_k_image(query_image_feature_path, './Feature_vgg16', k=5)

        print('\n')
        print('Query the most similar image using inception!')

        # Extract features of the query image using inception
        extract_features_inception(query_image_path,
'./Query_feature/query_inception.npy')

        # Query the most similar image using inception
        query_image_feature_path = './Query_feature/query_inception.npy'
        min_k_index_inception, min_k_similarity_inception =
query_k_image(query_image_feature_path, './Feature_inception', k=5)

        print('\n')

        print('The query image is:', query_image_path)
        print('The top 5 most similar images using ResNet50:')

        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(zip(min_k_index_resnet50,
min_k_similarity_resnet50)):
            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('ResNet50')
        plt.tight_layout()
        plt.savefig(f'{output_path}/ResNet50_{i}.png')

        print('The top 5 most similar images using VGG16:')

        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(zip(min_k_index_vgg16,
min_k_similarity_vgg16)):
            print('Index:', idx, 'Similarity:', sim)
```

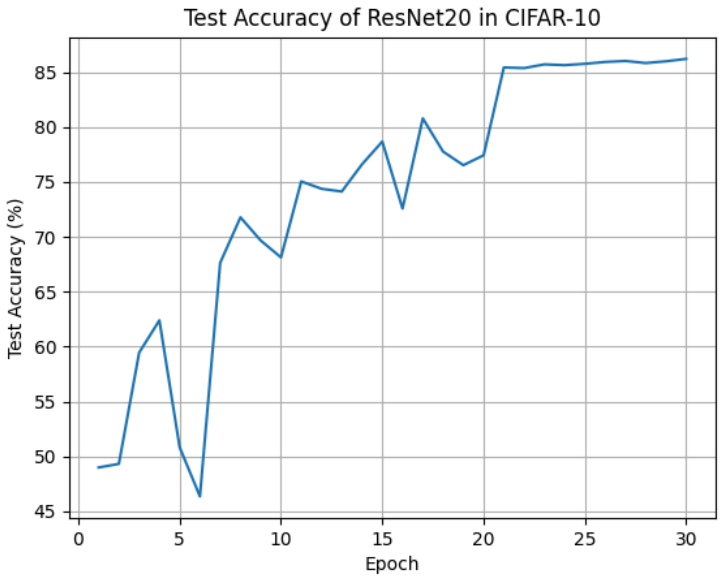
```
            image_path = f'./Dataset/{idx}.jpg'
            image = plt.imread(image_path)
            plt.subplot(1, 6, j+2)
            plt.imshow(image)
            plt.axis('off')
            plt.title(f'Top {j+1} similar: Index: {idx}, Similarity: {sim:.2f}')
        plt.axis('off')
        plt.suptitle('VGG16')
        plt.tight_layout()
        plt.savefig(f'{output_path}/VGG16_{i}.png')

        print('The top 5 most similar images using Inception V3:')

        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(zip(min_k_index_inception,
min_k_similarity_inception)):
            print('Index:', idx, 'Similarity:', sim)
            plt.subplot(1, 6, j+2)
            image_path = f'./Dataset/{idx}.jpg'
            image = plt.imread(image_path)
            plt.imshow(image)
            plt.axis('off')
            plt.title(f'Top {j+1} similar: Index: {idx}, Similarity: {sim:.2f}')
        plt.axis('off')
        plt.suptitle('Inception V3')
        plt.tight_layout()
        plt.savefig(f'{output_path}/Inception_{i}.png')
```

3. 实验结果及分析

3.1 实验5.1 CIFAR-10数据集分类

在实验5.1中，我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86%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20轮训练之后基本上达到了80%左右，但是在训练过程中，模型的准确率波动较大，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10轮训练后逐步稳定，最终达到了86.21%。

3.2 实验5.2 使用Pytorch提取图像特征并检索

在实验5.2中，我通过使用三种在ImageNet上预训练好的模型，分别是VGG16、ResNet50和InceptionV3，对图像进行特征提取，并通过计算得到的**特征向量间的夹角**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)

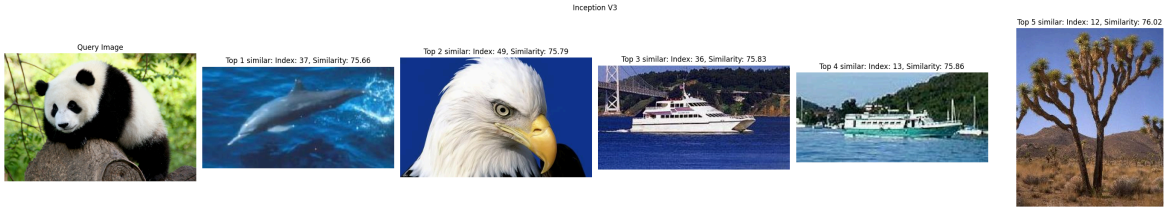
图像1

检索结果:

ResNet50: 得分：5.21，5.24，5.32，5.37，5.38

VGG16: 得分：80.72，82.81，83.55，83.58，83.75

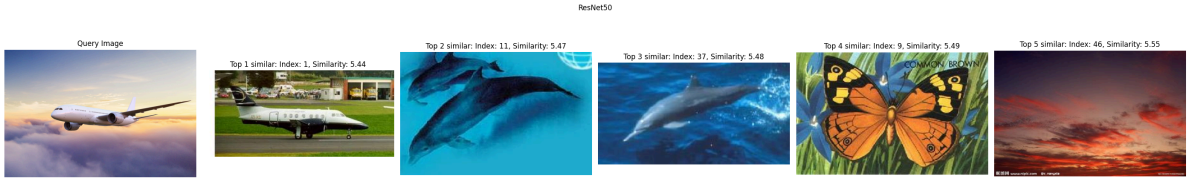
InceptionV3: 得分：75.66，75.79，75.85，75.86，76.02



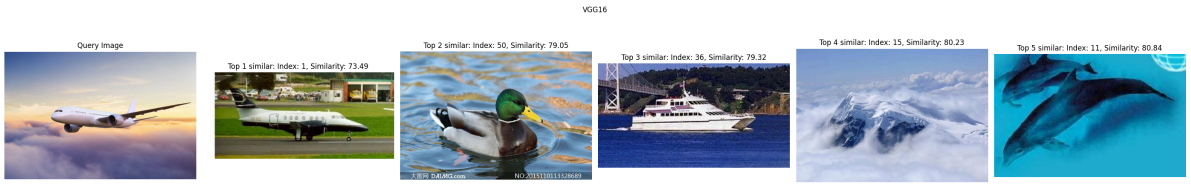
图像2

检索结果：

ResNet50: 得分：5.44，5.47，5.48，5.49，5.55



VGG16: 得分：73.49，79.05，79.32，80.32, 80.84



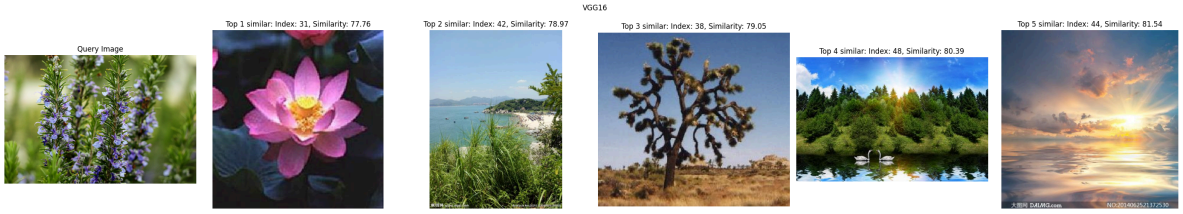
InceptionV3: 得分：75.42，75.42，75.75，75.79，75.90

图像3

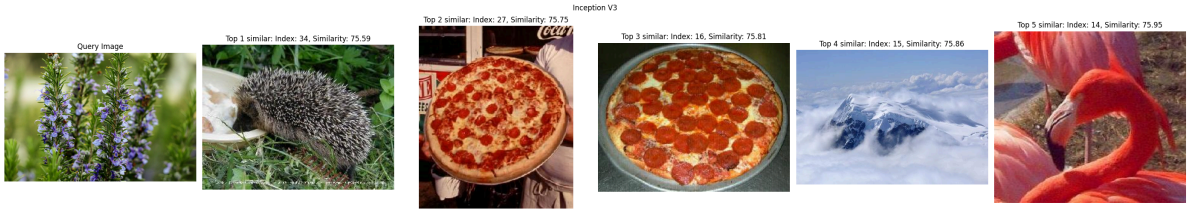
检索结果:

ResNet50: 得分：5.14，5.16，5.22，5.22，5.22

VGG16: 得分：77.76，78.97，79.05，80.39，81.54



InceptionV3: 得分：75.59，75.75，75.82，75.86，75.95



从上述结果可以看出，Resnet50和VGG16在检索结果上表现较好，InceptionV3的检索结果相对较差，无法找到与待检索图像相似度较高的图像。这可能是因为InceptionV3的模型结构较为复杂，特征提取的效果不如ResNet50和VGG16。不过在图片3中，ResNet50与InceptionV3检索到的最相似的图像均与待检索图像的相似度不高，说明模型的效果还不够稳定。

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86%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，我学习到了如何使用Pytorch提取图像特征，并通过计算特征向量间的夹角来进行图像检索，同时也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。

4. 思考及感想

4.1 思考题解答

1. 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 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。并且Train acc 一般会大于 Test acc，在多次训练后，Test acc 会逐渐趋近于 Train acc。
- 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。

4.2 实验感想

通过本次实验，我了解了如何使用Pytorch对CIFAR-10数据集进行分类，并通过调整学习率和训练策略，提高模型的性能。同时，我还学习到了如何使用Pytorch提取图像特征，并通过计算特征向量间的夹角来进行图像检索。通过这次实验，我对深度学习的训练策略和模型性能有了更深入的了解，同时也学习到了如何使用Pytorch进行图像检索，以及学习到基本的深度学习的知识。

5. 附录-代码

5.1 实验5.1

1. `exp2.py`:

```
'''Train CIFAR-10 with PyTorch.'''
import os

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

file_path = os.path.dirname(__file__)
data_path = os.path.join(file_path, 'data')
save_path = os.path.join(file_path, 'checkpoint')

os.makedirs(data_path, exist_ok=True)

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

```
# 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_path, train=True, download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True)

testset = torchvision.datasets.CIFAR10(
    root=data_path, 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
        ########################################
    # Save checkpoint.
    print('Test Acc: %f' % acc)
    print('Saving..')
    state = {
        'net': model.state_dict(),
        'acc': acc,
        'epoch': epoch,
    }
    torch.save(state, save_path + '/ckpt_%d_acc_%f.pth' % (epoch, acc))


for i in range(len(lrs)):
    for epoch in range(start_epoch, end_epoch + 1):
        train(epoch, lr=lrs[i])
        test(epoch)
        train(epoch, lr=lrs[i])
        test(epoch)
```

2. `draw.py`:

```
import numpy as np
import matplotlib.pyplot as plt

test_acc = [
    49.00, 49.33, 59.46, 62.40, 50.84, 46.37, 67.64, 71.78, 69.67, 68.13, 75.05,
74.38, 74.13, 76.61, 78.68, 72.58, 80.79, 77.77, 76.53, 77.43, 85.43, 85.37,
85.71, 85.64, 85.76, 85.93, 86.02, 85.83, 85.99, 86.21
]

plt.plot(range(1, 31), test_acc)
plt.xlabel('Epoch')
plt.ylabel('Test Accuracy (%)')
plt.title('Test Accuracy of ResNet20 in CIFAR-10')
plt.grid()
plt.savefig('./test_acc.png')
```

5.2 实验5.2

1. `main.py`:

```
import os
import numpy as np
import matplotlib.pyplot as plt
from extract_feature_resnet50 import extract_features_resnet50
from extract_feature_vgg16 import extract_features_vgg16
from extract_feature_inception import extract_features_inception

# Calculate the similarity between two features using angle
def cal_similarity(feature1, feature2):
    feature1 = feature1.flatten()
    feature2 = feature2.flatten()
    similarity = np.dot(feature1, feature2) / (np.linalg.norm(feature1) *
np.linalg.norm(feature2))
    return np.arccos(similarity) / np.pi * 180


# Extract features of all images in the dataset using ResNet50
def extract_dataset_feature_resnet50(dataset_path, save_path):
    for i in range(1, 51):
        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)


# Extract features of all images in the dataset using VGG16
def extract_dataset_feature_vgg16(dataset_path, save_path):
    for i in range(1, 51):
        input_image_path = dataset_path + f'/{i}.jpg'
        save_path_i = save_path + f'/{i}.npy'
        extract_features_vgg16(input_image_path, save_path_i)


def extract_dataset_feature_inception(dataset_path, save_path):
    for i in range(1, 51):
```

```
        input_image_path = dataset_path + f'/{i}.jpg'
        save_path_i = save_path + f'/{i}.npy'
        extract_features_inception(input_image_path, save_path_i)


# Query the most similar image in the dataset
def query_image(query_image_feature_path, dataset_path):
    query_feature = np.load(query_image_feature_path)

    min_similarity = 180
    min_index = -1
    for i in range(1, 51):
        feature_path = dataset_path + f'/{i}.npy'
        feature = np.load(feature_path)
        if feature is None:
            print('Feature not found!')
            continue

        sim = cal_similarity(query_feature, feature)
        if sim < min_similarity:
            min_similarity = sim
            min_index = i
    return min_index, min_similarity

# Query the k most similar images in the dataset
def query_k_image(query_image_feature_path, dataset_path, k=5):
    query_feature = np.load(query_image_feature_path)

    similarity = []
    index = []

    for i in range(1, 51):
        feature_path = dataset_path + f'/{i}.npy'
        feature = np.load(feature_path)
        if feature is None:
            print('Feature not found!')
            continue

        sim = cal_similarity(query_feature, feature)
        similarity.append(sim)
        index.append(i)

    similarity = np.array(similarity)
    index = np.array(index)

    min_index = index[np.argsort(similarity)[:k]]
    min_similarity = similarity[np.argsort(similarity)[:k]]

    return min_index, min_similarity

def main():
    os.makedirs('./Feature_resnet50', exist_ok=True)
    os.makedirs('./Feature_vgg16', exist_ok=True)
    os.makedirs('./Feature_inception', 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_vgg16 = './Feature_vgg16'
    save_path_inception = './Feature_inception'
    output_path = './Output'

    # If the features of the dataset have been extracted, comment the following
three lines
    extract_dataset_feature_resnet50(dataset_path, save_path_resnet50)
    extract_dataset_feature_vgg16(dataset_path, save_path_vgg16)
    extract_dataset_feature_inception(dataset_path, save_path_inception)

    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('\n')
        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('\n')
        print('Query the most similar image using vgg16!')

        # Extract features of the query image using vgg16
        extract_features_vgg16(query_image_path,
f'./Query_feature/query_vgg16_{i}.npy')

        # Query the most similar image using vgg16
        query_image_feature_path = f'./Query_feature/query_vgg16_{i}.npy'
        min_k_index_vgg16, min_k_similarity_vgg16 =
query_k_image(query_image_feature_path, './Feature_vgg16', k=5)

        print('\n')
        print('Query the most similar image using inception!')

        # Extract features of the query image using inception
        extract_features_inception(query_image_path,
f'./Query_feature/query_inception_{i}.npy')

        # Query the most similar image using inception
        query_image_feature_path = f'./Query_feature/query_inception_{i}.npy'
        min_k_index_inception, min_k_similarity_inception =
query_k_image(query_image_feature_path, './Feature_inception', k=5)

        print('\n')

        print('The query image is:', query_image_path)
```

```
        print('The top 5 most similar images using ResNet50:')

        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(zip(min_k_index_resnet50,
min_k_similarity_resnet50)):
            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('ResNet50')
        plt.tight_layout()
        plt.savefig(f'{output_path}/ResNet50_{i}.png')

        print('The top 5 most similar images using VGG16:')

        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(zip(min_k_index_vgg16,
min_k_similarity_vgg16)):
            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('VGG16')
        plt.tight_layout()
        plt.savefig(f'{output_path}/VGG16_{i}.png')

        print('The top 5 most similar images using Inception V3:')

        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(zip(min_k_index_inception,
min_k_similarity_inception)):
            print('Index:', idx, 'Similarity:', sim)
            plt.subplot(1, 6, j+2)
```

```
            image_path = f'./Dataset/{idx}.jpg'
            image = plt.imread(image_path)
            plt.imshow(image)
            plt.axis('off')
            plt.title(f'Top {j+1} similar: Index: {idx}, Similarity: {sim:.2f}')
        plt.axis('off')
        plt.suptitle('Inception V3')
        plt.tight_layout()
        plt.savefig(f'{output_path}/Inception_{i}.png')


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

2. `extract_feature_resnet50.py`:

```
# 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_vgg16.py`:

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

print('Load model: VGG16')
model = torch.hub.load('pytorch/vision', 'vgg16', 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_vgg16(x):
    x = model.features(x)
    x = model.avgpool(x)
    return x

def extract_features_vgg16(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_vgg16(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_inception.py`:

```
import time
import numpy as np
```

```
import torch
import torchvision.transforms as transforms
from torchvision.datasets.folder import default_loader

print('Load model: InceptionV3')
model = torch.hub.load('pytorch/vision', 'inception_v3', pretrained=True)

normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])

trans = transforms.Compose([
    transforms.Resize(299),
    transforms.CenterCrop(299),
    transforms.ToTensor(),
    normalize,
])

def features_inception(x):
    x = model.Conv2d_1a_3x3(x)
    x = model.Conv2d_2a_3x3(x)
    x = model.Conv2d_2b_3x3(x)
    x = model.Conv2d_3b_1x1(x)
    x = model.Conv2d_4a_3x3(x)
    x = model.Mixed_5b(x)
    x = model.Mixed_5c(x)
    x = model.Mixed_5d(x)
    x = model.Mixed_6a(x)
    x = model.Mixed_6b(x)
    x = model.Mixed_6c(x)
    x = model.Mixed_6d(x)
    x = model.Mixed_6e(x)
    x = model.Mixed_7a(x)
    x = model.Mixed_7b(x)
    x = model.Mixed_7c(x)
    return x


def extract_features_inception(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_inception(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)
```