Lab5 实验报告

李青雅 523030910004 电院 2301

2024 年 12 月 28 日

目录

1 实验概览

基于本次对于机器学习与深度学习的初步认识，了解神经网络及神经元的概念，理解模型训练过程的参量变化及参量意义，在此基础上，使用深度学习和卷积神经网络（ResNet20）对 CIFAR-10 数据集进行分类，评估模型在图像分类任务上的性能。基于对于机器学习与深度学习的进一步认识，了解图像检索的基本方式以及以图搜图的过程，通过深度学习模型使得模型能够学习到图像的某种特定的特征来将图像转换成一个能够代表图像的向量，从而用向量之间的相似程度代表图像之间的相似程度，从而解决以图搜图的任务。

2 实验环境

Python, Pytorch, Anaconda

3 实验一：CIFAR-10 图片分类

3.1 实验背景

我们首先来了解一下什么是神经网络，深度学习以及模型训练。

3.1.1 神经网络

神经网络是模拟人脑神经元连接方式的一种计算模型，用于处理复杂的非线性问题。它包含以下关键部分：

- 输入层：接受外部输入数据，每个节点对应一个特征。例如，在图像处理中，输入层每个节点可以表示一张图片的一个像素值。
- 隐藏层：对输入数据进行多层非线性变换。隐藏层的神经元通过激活函数将线性变换结果映射到非线性空间，允许模型学习复杂模式。
- 输出层：输出最终的预测结果，例如分类任务中输出类别的概率分布。

神经网络的核心在于“权值”的学习（权值是连接神经元之间的参数）。通过训练，模型不断调整这些权值，从而优化输入到输出的映射。

3.1.2 深度学习

深度学习是机器学习的一个子领域，指的是具有多层隐藏层的神经网络模型。与传统神经网络不同，深度学习强调“深度”，即隐藏层的数量非常多，可能达到几十甚至上百层。

3.1.3 模型训练

神经网络的训练过程是通过数据样本优化模型参数的过程，核心步骤包括以下内容：

1. 损失函数（Loss Function）：损失函数衡量模型输出与真实标签之间的差距
2. 反向传播（Backpropagation）：通过计算损失函数对模型参数的梯度，确定每个参数的调整方向和大小。

3. 优化算法：

   - 优化算法（如 SGD、Adam）用于更新模型参数，使得损失函数逐渐减小。
   - 学习率（Learning Rate）是关键的超参数，它决定了参数更新的步长。过大可能导致参数跳过最优解，过小可能导致收敛速度过慢。

4. 训练数据的使用：

   - 训练集（Training Set）：用于训练模型，即调整模型参数。
   - 测试集（Test Set）：用于最终评估模型的泛化能力。

5. 训练轮次（Epoch）和批次（Batch）：

   - Epoch：指模型对整个训练数据集完整训练一遍。
   - Batch：由于一次处理所有数据的计算量太大，数据集会被分成多个小批次，每次用一个批次的数据更新参数。

6. 参数调优：

   - 过拟合：模型在训练集上表现很好，但在测试集上表现差。解决方法包括正则化、数据增强、减少模型复杂度等。
   - 欠拟合：模型无法很好地拟合训练数据。解决方法包括增加模型复杂度、训练时间等。

3.2 数据集介绍

CIFAR-10 数据集：

- 包含 10 个类别的 60,000 张 32x32 彩色图像。
- 50,000 张训练图片和 10,000 张测试图片。
- 类别包括飞机、汽车、鸟类、猫、鹿、狗、青蛙、马、船、卡车。

3.3 实验流程

3.3.1 训练过程

我们需要在 50,000 张训练图像上进行参数更新，其中先以学习率 0.1 训练 5 个 epoch，然后以学习率 0.01 训练 5 个 epoch。每个 epoch 的训练过程如下：首先，使用损失函数计算模型输出与目标之间的损失。然后，将每次训练的损失累加到 *train_loss* 中。接下来，记录目标总数和正确预测的数量。之后，将每个 epoch 的训练结果（包括损失和准确率）写入文件 result.txt 中。最后，计算当前 epoch 的训练准确率。

训练好模型之后，我们需要将模型参数以及检查点等保存下来。首先，打印当前 epoch 的测试准确率。然后，如果检查点目录不存在，则创建该目录。最后，将模型状态保存到文件中。

3.3.2 训练结果

我们将每个 epoch 的训练结果保存到了文件 result.txt 中，并且将 test acc 的变化趋势画在了图中。如下图所示：

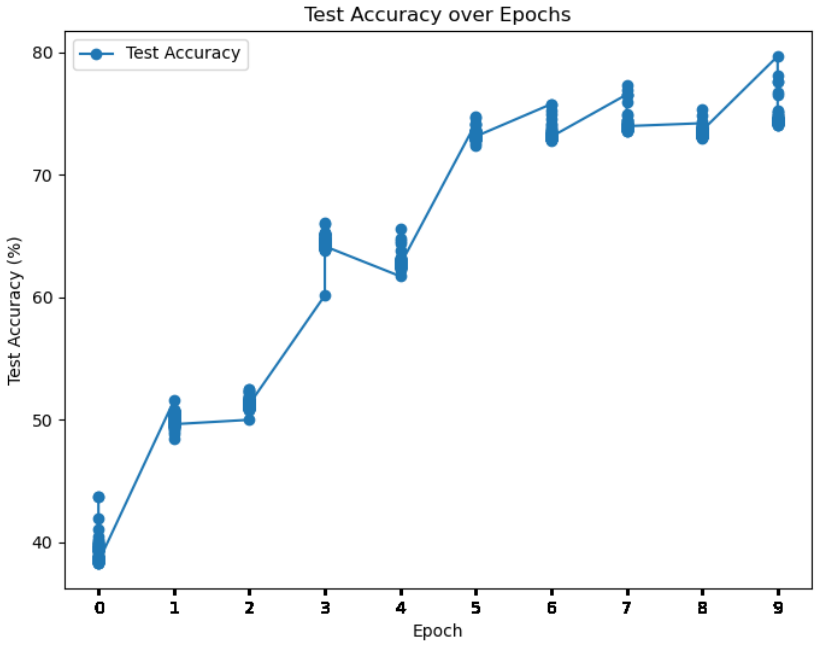

图 1: 训练准确率变化

这里每个 epoch 中有多个点，实则是每个 batch 都生成了一个点（见 exp2.py 中 test 函数）。也就是说，每个 epoch 中记录的点数和 batch 数量相同。对于我们使用的 CIFAR-10 的测试集，总共有 10000 张图片，其中我们设定了 batch_size=128，则 batch 数量为 79。因此，result.txt 中每个 epoch 的日志行数量是 79 行，对应图中每个 epoch 的 79 个点。

3.3.3 思考

1. Train acc 和 Test acc 有什么关联和不同？train acc 是模型在训练集上的准确率，通常是训练过程的直接反映，因为模型通过优化损失函数专门拟合训练数据，训练集的准确率可以快速提升。而 test acc 是模型在测试集上的准确率，用于评估模型对新数据的预测能力。在训练初期，train acc 和 test acc 同时提升。如果 train acc 很高而 test acc 不高，可能表明模型过拟合。

2. 在 lr 从 0.1 变到 0.01 后，acc 发生了什么变化？为什么？lr 下降后，acc 明显提升，但在下降完后的几个 epoch 中，acc 的提升速率变慢。笔者推测这是因为初期学习率较高，模型权重更新的步长较大，test acc 提升较快。然而，过高的学习率可能会使得模型无法充分逼近最优点，导致优化震荡或过早停止。而学习率降低后，权重更新步长减小，模型在损失函数的局部区域内进行更细致的优化。train acc 和 test acc 可能都有小幅提升，尤其是 test acc 提升更显著（见于图中在 lr 变化后 test acc 上升趋势变得更稳定）。

3.3.4 提高测试准确率

通过查询资料，笔者发现可以从改变优化器类型，调度学习率和增强数据三个方面提高测试准确率。于是在下图左中将优化器类型从 SDG 改为 AdamW，在每轮 epoch 结束后使用 Cosine Annealing 调整学习率，以及对训练集使用 Mixup 数据增强，提升模型的泛化能力。

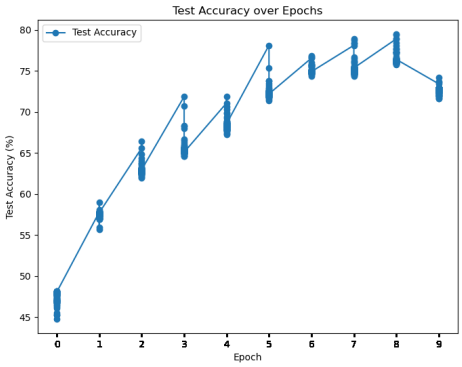

图 2: 测试准确率变化

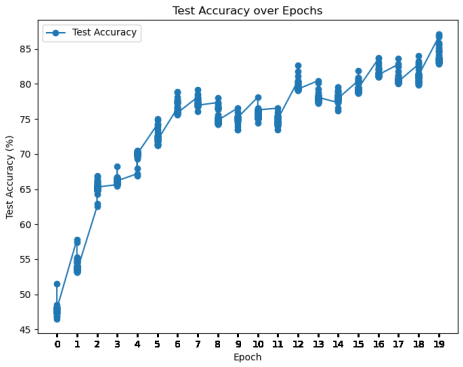

图 3: 提高 epoch 数量

但是调整完之后，笔者发现效果并不是很好，并且在最后一次 epoch 中，test acc 反而下降了，可能出现了过拟合。于是笔者增加了 epoch 数量，并且将 0 到 7 次 epoch 的学习率设置为 0.1，8 到 19 次 epoch 的学习率设置为 0.01。如上图右中所示，test acc 在 epoch 增加后有明显提升。

4 实验二：图像检索

4.1 实验背景

早期的图像检索技术研究主要是基于文本的图像检索，通过对图片的文本描述来查找图像库中具有相应标签的图像。随着技术的发展，基于内容的图像检索，即以图搜图也成为了各大网站的标配功能。以图搜图是如何实现的呢？以图搜图的核心其实就是在被检索的图像库中找到和参与检索的图片最相似的图片集合。那么问题就变成了如何定义图片之间的相似程度。

以下图为例，与待检索图像“Query”最相近的应该是图像“f”；但是图像库中的其他图片与“Query”的相似程度应该怎么排序呢？如果从考虑图像颜色分布的情况来看，图像“b”和“e”也会有较高的相似程度；如果考虑交通工具这个范畴，图像“c”则会有较高的排序。

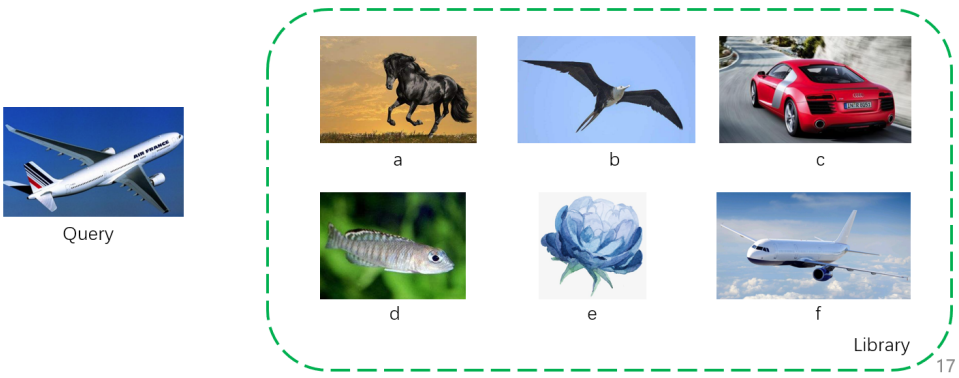

图 4: 图像检索示例

所以如果我们能从图像的内容中提取出相应的语义信息，如颜色，纹理，种类等等，那么这个以图搜图的检索任务就自然得到了解决。深度学习模型通过一种端到端学习的范式，使得模型能够学习到图像的这种特征。所以我们可以使用模型来将图像转换成一个能够代表图像的向量，从而用向量之间的相似程度代表图像之间的相似程度，从而解决以图搜图的任务。

本次实验中，笔者将使用深度模型提取图像的特征，从而帮助构建以图搜图的搜索引擎。

4.2 实验流程

4.2.1 使用 Pytorch 提取图像特征

笔者参考给出的示例代码 extract_features.py，用 pytorch 提取了示例图像 panda.jpg 的特征。直接运行程序得到如下结果：

```
Load model: ResNet50
Using cache found in C:\Users\une_g/.cache\torch\hub\pytorch_vision_main
E:\Anaconda3\envs\pytorch_GPU\Lib\site-packages\torchvision\models\_utils.py:208: UserWarning: The parameter 'pretr
ained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
E:\Anaconda3\envs\pytorch_GPU\Lib\site-packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than
 a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the future. The current beha
vior is equivalent to passing `weights=ResNet50_Weights.IMAGENET1K_V1`. You can also use `weights=ResNet50_Weights.
DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Prepare image data!
Extract features!
Time for extracting features: 0.13
Save features!
```

图 5: 提取图像特征

在终端我们发现了警告信息，提示我们 torchvision 中的 pretrained 参数在 0.13 后已经被弃用，由于笔者的 torchvision 版本为 0.20，所以修改了代码中的模型导入方式。将

$$model = torch.hub.load('pytorch/vision','resnet50', pretrained = True)$$

改为了

$$model = resnet50(weights = ResNet50\_Weights.IMAGENET1K\_V1)$$

再次运行代码，警告消失：

```
Load model: ResNet50
Prepare image data!
Extract features!
Time for extracting features: 0.09
Save features!
```

图 6: 提取图像特征

在代码中，我们定义了如下的 features 函数用来提取图像特征：

```
def features(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
```

这里我们提取的是模型倒数第二层的结果。在 ResNet50 模型中，正常调用模型会返回模型在 ImageNet 数据集上的最终分类结果，也就是通过最后一层全连接层输出的 1000 维分类向量。这是针对 ImageNet 数据集的分类任务的结果，而在图像检索任务中，我们需要的是图像的特征表示，而不是分类结果。

在深度神经网络中，倒数第二层（avgpool）的输出是经过特征提取后的高层特征表示，这些特征是模型对输入图像理解的浓缩表示，不受最后一层分类任务的限制。因此，提取倒数第二层的输出可以更好地用于图像检索等非分类任务。

由于不同的模型搭建方式不同，所以在不同模型下的 features 函数也不尽相同。例如，在 VGG16 模型下，我们的函数如下所示：

```
def features(x):
    x = model.features(x)
    x = model.avgpool(x)
    x = torch.flatten(x, 1)  # 展平成 1D 向量
    return x
```

4.2.2 构建图像库

由于笔者的电脑在笔者年轻时存了很多好看的图和梗图，所以就不从网上爬图片了。为了更好地提取图片特征，笔者写了一个 Python 脚本将图片的名字改为.png 格式，并将这些图片放在了名为 piclib 的文件夹中，此文件夹即为图像库。

4.2.3 检索图片

图像经过模型后保存得到的特征为一个向量，计算向量之间的相似程度有两种方法：

1. 方法一：首先将向量归一化，计算向量之间的欧氏距离，通过距离的远近来反映向量之间的相似程度。

2. 方法二：通过计算向量之间的夹角，夹角越大则越不相似，反之越相似。

当使用方法一时，欧氏距离越接近 0，说明两张图片越相似。笔者在实现方法二时，返回是两个向量夹角的余弦值，余弦值越接近 1，说明两张图片越相似。其中排序的方法为冒泡排序，由于图片较少，比较的次数并不多。当数据较大时，就应该考虑时间复杂度更低的堆排序和归并排序（O(nlogn)）。代码将在下文给出。

4.2.4 实验结果

我们选取目标图片为下图，分别用两种方法进行检索。

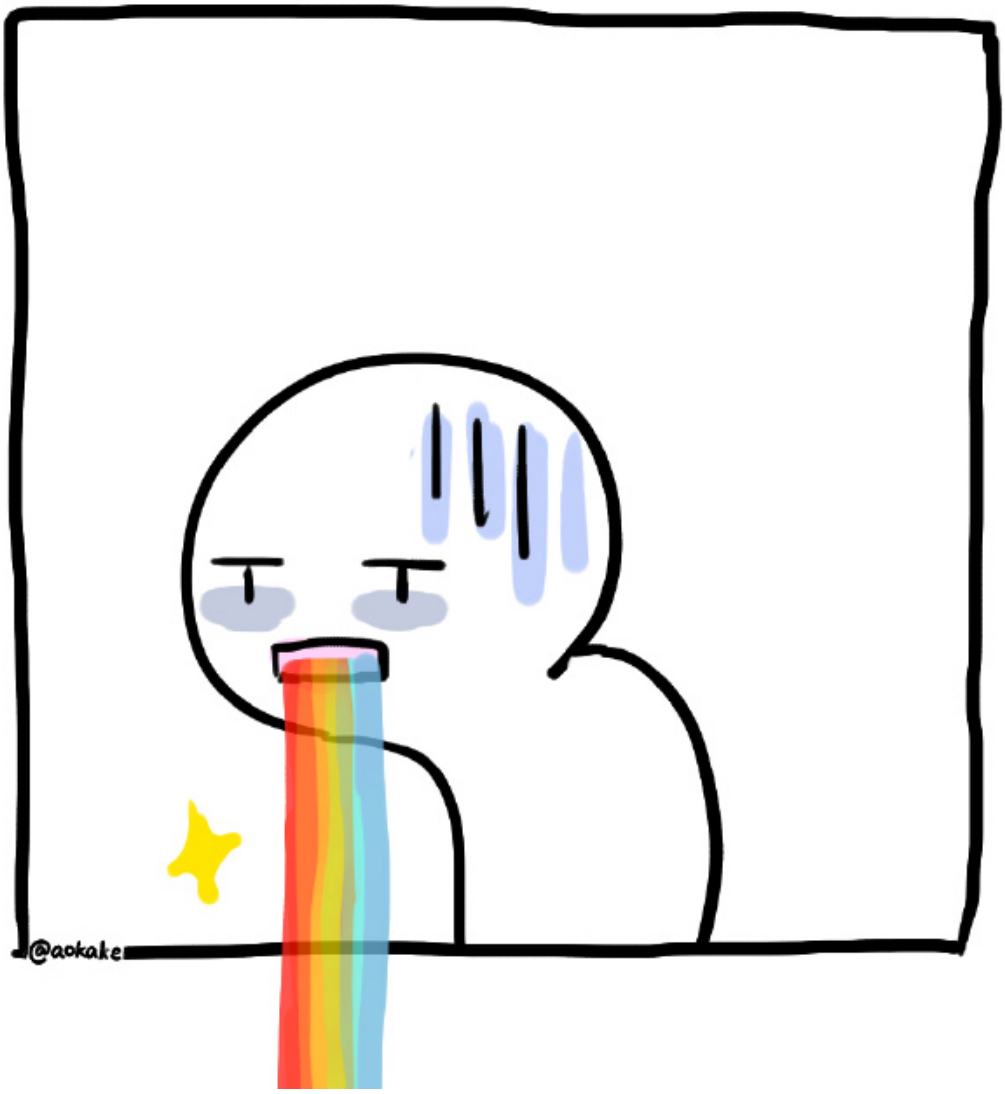

图 7: 目标图片

方法一的检索结果如下：

```
[np.float32(0.0), '160.png']
[np.float32(0.08687053), '158.png']
[np.float32(0.08832728), '159.png']
[np.float32(0.09039075), '161.png']
[np.float32(0.09225822), '182.png']
```

图 8: 欧氏距离检索结果

可以发现欧氏距离依次增加，说明图片相似度依次降低。

其代表的图片从左往右依次为：

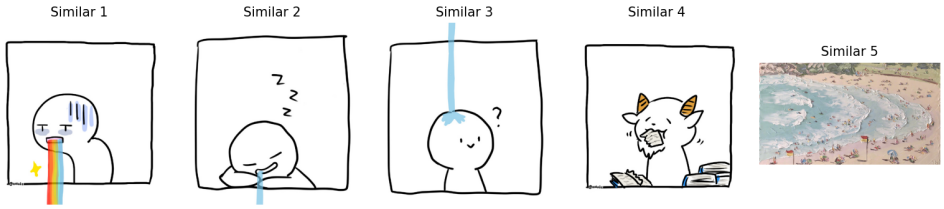

图 9: 欧氏距离检索结果

方法二的检索结果如下：

```
[np.float32(0.9999999), '160.png']
[np.float32(0.9962262), '158.png']
[np.float32(0.99609834), '159.png']
[np.float32(0.9959146), '161.png']
[np.float32(0.9957437), '182.png']
```

图 10: 夹角余弦值检索结果

可以发现夹角余弦值依次减小，说明图片相似度依次降低。
其代表的图片从左往右依次为：

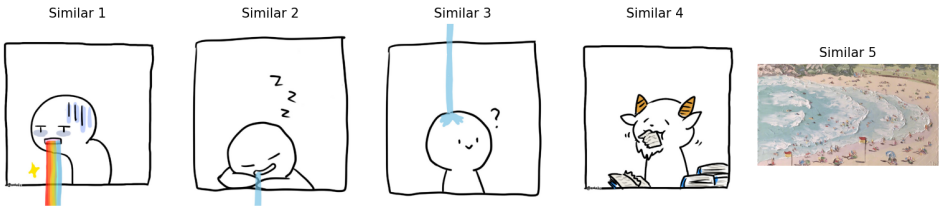

图 11: 夹角余弦值检索结果

可以看到两种方法完成的检索结果一致，并且较为准确。

4.2.5 使用非图像库图片实现以图搜图

笔者在网上找了一张可爱小狗的图片作为目标图片，进行了以图搜图的实验。结果如下：

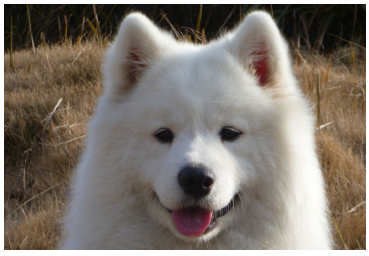

图 12: 目标图片

```
[np.float32(0.09457179), '6.png']
[np.float32(0.094702944), '92.png']
[np.float32(0.09529764), '9.png']
[np.float32(0.095405236), '200.png']
[np.float32(0.09559344), '31.png']
```

图 13: 欧氏距离检索结果

```
[np.float32(0.99552864), '6.png']
[np.float32(0.9955148), '92.png']
[np.float32(0.99545866), '9.png']
[np.float32(0.9954484), '200.png']
[np.float32(0.99542993), '31.png']
```

图 14: 夹角余弦值检索结果

两者的检索结果一致，如下图所示：

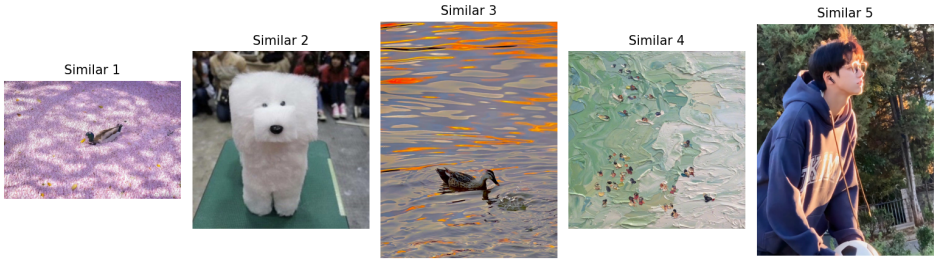

图 15: ResNet50 检索结果

4.2.6 VGG16 的使用和与 ResNet50 的对比

除了 ResNet50，还有其他预训练的模型，如 VGG16、MobileNet 或其他 ResNet 变种。这里笔者使用了 VGG16 模型，目标图像则同上文一样为白色小狗图像。结果如下：

```
[np.float32(1.1309096), '91.png']
[np.float32(1.1428136), '92.png']
[np.float32(1.2378625), '137.png']
[np.float32(1.2438799), '166.png']
[np.float32(1.2653372), '76.png']
```

图 16: 欧氏距离检索结果

```
[np.float32(0.36052185), '91.png']
[np.float32(0.34698862), '92.png']
[np.float32(0.2338483), '137.png']
[np.float32(0.22638157), '166.png']
[np.float32(0.19946104), '76.png']
```

图 17: 夹角余弦值检索结果

二者的检索结果均为下图：

图 18: VGG16 检索结果

可以看到，VGG16 返回了与目标图像相似的前 5 张图片，这些图片均包含与目标图像相似的白色狗，且物体主体特征更为接近。VGG16 的第 1 张检索结果与目标图像非常相似（白色狗，背景简单），接下来的检索结果也大多是狗的图片。说明对目标物体的外形、纹理以及局部特征的提取

表现优秀。而 ResNet50 对整体的视觉特征提取表现良好，能够识别出与目标图像风格、颜色、背景接近的图片。但检索结果中有一些与目标物体（白色狗）无直接关联的图片（例如纹理图片和人物图片），说明 ResNet50 在某些情况下可能更多依赖图像的整体风格和背景信息，而非具体物体的特征。

5 实验体会心得与思考

哇，最后一个 lab 了（如果不算最后的车牌识别大作业的话）。也算是炼上丹了，在第一个 Pytorch 的实验中狠狠感受到了一把调参的魅力，诸如 Mixup 要混多少，epoch 要训几轮，优化器用哪种，学习率调成多少、在第几次 epoch 里面开始调整，调整完之后学习率下降了怎么办，显卡跑不跑得起来，这些都是我曾经没考虑过的问题。

现在好多实验室都直接 all in ai 了，模式识别确实和传统的图像处理很不一样，感觉就是黑箱。至于里面为什么会是这样，为什么是这个结果，没人能具体切实地说清楚，所以看十几二十年前的论文才会有一种精致的美感吧。不过，这也算是大势所趋了。

希望能好好复习期末，把车牌识别的大作业做好，给这个令人叹息的学期画一个完满的句号。

6 源代码

6.1 实验一：CIFAR-10 图片分类

6.1.1 模型训练 exp2.py

```
# SJTU EE208

'''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
import numpy as np

from models import resnet20

# 检查是否有GPU支持
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f'Using device: {device}')

start_epoch = 0
end_epoch = 7
lr = 0.1

# 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().to(device)  # 将模型移动到GPU
    # 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()
    # 使用SGD优化器
    #optimizer = optim.SGD(model.parameters(), lr=lr, weight_decay=5e-4)
    # 使用AdamW优化器
    optimizer = torch.optim.AdamW(model.parameters(), lr=0.001, weight_decay=5e-4)

    # Mixup 数据增强
    ##########################################################################
    def mixup_data(x, y, alpha=1.0):
        '''Mixup 数据增强'''
        if alpha > 0:
            lam = np.random.beta(alpha, alpha)
        else:
            lam = 1
        batch_size = x.size()[0]
        index = torch.randperm(batch_size).to(x.device)

        mixed_x = lam * x + (1 - lam) * x[index, :]
        y_a, y_b = y, y[index]
        return mixed_x, y_a, y_b, lam

    def mixup_criterion(criterion, pred, y_a, y_b, lam):
        '''Mixup 损失函数'''
        return lam * criterion(pred, y_a) + (1 - lam) * criterion(pred, y_b)
    ##########################################################################

    # 使用Mixup数据增强的训练
    def train(epoch):
        model.train()
```

```
        train_loss = 0
        correct = 0
        total = 0
        for batch_idx, (inputs, targets) in enumerate(trainloader):
            inputs, targets = inputs.to(device), targets.to(device)  # 将数据移动到GPU

            # 使用 Mixup 数据增强
            inputs, targets_a, targets_b, lam = mixup_data(inputs, targets, alpha=0.4)

            optimizer.zero_grad()
            outputs = model(inputs)

            # 使用 Mixup 损失函数
            loss = mixup_criterion(criterion, outputs, targets_a, targets_b, lam)
            loss.backward()
            optimizer.step()

            train_loss += loss.item()

            # 计算训练精度（以原始标签为基准，仅作为参考）
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += (lam * predicted.eq(targets_a).sum().item() +
                        (1 - lam) * predicted.eq(targets_b).sum().item())

            print('Epoch [%d] Batch [%d/%d] Loss: %.3f | Training Acc: %.3f%% (%d/%d)'
                  % (epoch, batch_idx + 1, len(trainloader), train_loss / (batch_idx + 1),
                     100. * correct / total, correct, total))


    # # 普通训练
    # def train(epoch):
    #     model.train()
    #     train_loss = 0
    #     correct = 0
    #     total = 0
    #     for batch_idx, (inputs, targets) in enumerate(trainloader):
    #         inputs, targets = inputs.to(device), targets.to(device)  # 将数据移动到GPU
    #         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()
        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.
            train_loss = 0
            correct = 0
            total = 0
            for batch_idx, (inputs, targets) in enumerate(testloader):
                inputs, targets = inputs.to(device), targets.to(device)  # 将数据移动到GPU
                # optimizer.zero_grad(): 优化器进行初始化
                optimizer.zero_grad()
                # inputs是x, model是function, outputs是f(x)
                outputs = model(inputs)
                # 损失（loss）：神经网络输出和目标之间的距离
                loss = criterion(outputs, targets)
                # train_loss是每次测试的损失
                train_loss += loss.item()
                # 记录输出的最大值（最好的匹配结果）
                _, predicted = outputs.max(1)
                # 记录目标总数
                total += targets.size(0)
                # 记录目标正确匹配数
                correct += predicted.eq(targets).sum().item()
                with open('result.txt', 'a') as f:
                    f.write('TEST::Epoch [%d] Batch [%d/%d] Loss: %.3f | Traininig Acc: %.3f%% (%d/%
                        d)\n'% (epoch, batch_idx + 1, len(testloader), train_loss / (batch_idx + 1)
                        , 100. * correct / total, correct, total))
                    # print('TEST::Epoch [%d] Batch [%d/%d] Loss: %.3f | Traininig Acc: %.3f%% (%d/%
                        d)'% (epoch, batch_idx + 1, len(testloader), train_loss / (batch_idx + 1),
                        100. * correct / total, correct, total))
            acc = 1.0 * correct / total
            ########################################
        # 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))

    # 学习率调度
    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=10)
    for epoch in range(start_epoch, end_epoch + 1):
        train(epoch)
        test(epoch)
        scheduler.step()

    start_epoch = 8
    end_epoch = 19
    lr = 0.01
    # 使用SGD优化器
    # optimizer = optim.SGD(model.parameters(), lr=lr, weight_decay=5e-4)
```

```
    # 使用AdamW优化器
    optimizer = torch.optim.AdamW(model.parameters(), lr=0.001, weight_decay=5e-4)
    for epoch in range(start_epoch, end_epoch + 1):
        train(epoch)
        test(epoch)
        scheduler.step()
```

6.1.2 生成 test acc 变化 pics.py

```
import re
import matplotlib.pyplot as plt

# 初始化存储数据的列表
epochs = []
test_accs = []

# 从 result.txt 文件中读取测试准确率
with open('result.txt', 'r') as f:
    for line in f:
        # 匹配 TEST:: 的日志行
        if "TEST::" in line:
            # 提取 Epoch 和 Test Accuracy
            match = re.search(r'Epoch \[(\d+)] .* Traininig Acc: ([\d\.]+)', line)
            if match:
                epoch = int(match.group(1))
                test_acc = float(match.group(2))
                epochs.append(epoch)
                test_accs.append(test_acc)

# 检查提取的数据
print("Epochs:", epochs)
print("Test Accuracies:", test_accs)

# 绘制测试准确率变化趋势
plt.figure(figsize=(8, 6))
plt.plot(epochs, test_accs, marker='o', label="Test Accuracy")
plt.title("Test Accuracy over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Test Accuracy (%)")
plt.xticks(epochs)  # 显示每个 Epoch 的刻度
plt.grid(False)
plt.legend()
plt.show()
```

6.2 实验二：图像检索

6.2.1 图片处理 dealing_photos.py

```
import os
from PIL import Image

# 设置图片库路径
image_dir = r"E:\ICE2607\lab5-PyTorch_CNN\CNN\piclib"
output_dir = os.path.join(image_dir, "renamed_images")  # 新文件夹存储重命名的图片

# 创建输出文件夹
```

```
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    # 遍历文件夹并重命名
    def rename_images(image_dir, output_dir):
        # 获取图片文件列表
        files = [f for f in os.listdir(image_dir) if f.lower().endswith(('.jpg', '.jpeg', '.png', '.
            webp'))]
        files.sort()  # 可选，按文件名排序
        count = 1  # 从1开始编号

        for file in files:
            file_path = os.path.join(image_dir, file)

            # 打开图片并保存为 .png 格式
            try:
                img = Image.open(file_path)
                new_name = f"{count}.png"  # 重命名为1, 2, 3...的格式
                new_path = os.path.join(output_dir, new_name)

                img.save(new_path, format="PNG")
                print(f"Renamed and converted: {file} -> {new_name}")
                count += 1  # 编号递增
            except Exception as e:
                print(f"Failed to process {file}: {e}")

    rename_images(image_dir, output_dir)
    print("All files have been renamed and saved as .png!")
```

6.2.2 提取图像特征并检索 extract_features.py

```
    # SJTU EE208


    # 使用ResNet50模型

    # import time
    # import os
    # import matplotlib.pyplot as plt
    # from PIL import Image

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

    # from torchvision.models import resnet50, ResNet50_Weights

    # print('Load model: ResNet50')
    # model = resnet50(weights=ResNet50_Weights.IMAGENET1K_V1)

    # 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,
```

```
    # ])

    # print('Prepare image data!')
    # test_image = default_loader('white_dog.png')
    # input_image = trans(test_image)
    # input_image = torch.unsqueeze(input_image, 0)

    # # resnet50的特征提取
    # def features(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


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


    # print('Save features!')
    # np.save('features.npy', image_feature)


    # def dist(feature1,feature2):
    #     # 需要归一化，否则容易过大
    #     dis, t1, t2 = 0, 0, 0
    #     for i in feature1[0]:
    #         t1 += i[0][0]**2
    #     t1 = t1**0.5
    #     feature11 = [i[0][0]*1.0/t1 for i in feature1[0]]
    #     for i in feature2[0]:
    #         t2 += i[0][0]**2
    #     t2 = t2**0.5
    #     feature22 = [i[0][0]*1.0/t2 for i in feature2[0]]
    #     for i in range(len(feature11)):
    #         dis += (feature11[i]-feature22[i])**2
    #     dis = dis**0.5
    #     return dis

    # def angle(feature1,feature2):
    #     cnt, t1, t2 = 0, 0, 0
    #     for i in feature1[0]:
    #         t1 += i[0][0]**2
    #     t1 = t1**0.5
    #     feature11 = [i[0][0] for i in feature1[0]]
    #     for i in feature2[0]:
    #         t2 += i[0][0]**2
    #     t2 = t2**0.5
    #     feature22 = [i[0][0]for i in feature2[0]]
```

```
87      #     for i in range(len(feature22)):
88      #         cnt += feature11[i]*feature22[i]
89      #     cnt /= t1
90      #     cnt /= t2
91      #     return cnt
92
93      # # 展示与目标图片最相似的图片
94      # def show_similar_images(target_path, similar_images):
95      #     """
96      #     显示目标图片以及与之最相似的图片
97      #     :param target_path: 目标图片路径
98      #     :param similar_images: 与目标图片相似的图片列表，每个元素为 [距离, 文件名]
99      #     """
100     #     # 创建画布
101     #     num_images = len(similar_images) + 1  # 包括目标图片
102     #     plt.figure(figsize=(15, 5))
103
104     #     # 显示相似图片
105     #     for idx, (_, filename) in enumerate(similar_images):
106     #         plt.subplot(1, num_images, idx + 2)
107     #         similar_img = Image.open(os.path.join('piclib', filename))
108     #         plt.imshow(similar_img)
109     #         plt.title(f"Similar {idx + 1}")
110     #         plt.axis("off")
111
112     #     plt.tight_layout()
113     #     plt.show()
114
115
116     # distance = []
117     # # 调用函数去计算不同的欧氏距离情况比较
118     # start = time.time()
119     # for i in range(1,210):
120     #     filename = str(i) + '.png'
121     #     print('Prepare image data:  '+ filename + "  !")
122     #     # 此处读入i.png的图片，然后进行与前面待匹配图像一样的操作，故可略
123     #     temp_image = default_loader(os.path.join('piclib',filename))
124     #     test_image = trans(temp_image)
125     #     test_image = torch.unsqueeze(test_image, 0)
126
127     #     test_feature = features(test_image)
128     #     test_feature = test_feature.detach().numpy()
129
130     #     # 欧氏距离内存下的为具体数值以及文件名
131     #     distance.append([dist(image_feature,test_feature),filename])
132     #     # 角度下的为具体数值及文件名
133     #     # distance.append([angle(image_feature,test_feature),filename])
134
135     # print('Time for extracting features: {:.2f}'.format(time.time() - start))
136
137     # for i in range(len(distance)):
138     #     for j in range(len(distance)):
139     #         # 如果是dist，则为<；如果是角度 则为>
140     #         if(distance[i][0]<distance[j][0]):
141     #             distance[i], distance[j] = distance[j], distance[i]
142     # for i in range(5):
143     #     print(distance[i])
144
145
```

```
    # # 调用展示函数
    # target_image_path = 'white_dog.png'
    # most_similar_images = distance[:5]  # 取出距离最近的5张图片
    # # 显示目标图片
    # target_img = Image.open(target_image_path)
    # plt.imshow(target_img)
    # plt.title("Target Image")
    # plt.axis("off")
    # plt.show()
    # show_similar_images(target_image_path, most_similar_images)



    # 使用VGG16模型

    import time
    import os
    import matplotlib.pyplot as plt
    from PIL import Image

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

    from torchvision.models import vgg16, VGG16_Weights

    print('Load model: VGG16')
    model = vgg16(weights=VGG16_Weights.IMAGENET1K_V1)

    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,
    ])

    print('Prepare image data!')
    test_image = default_loader('white_dog.png')
    input_image = trans(test_image)
    input_image = torch.unsqueeze(input_image, 0)

    def features(x):
        x = model.features(x)
        x = model.avgpool(x)
        x = torch.flatten(x, 1)
        return x


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

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


    def dist(feature1, feature2):
        # 需要归一化，否则容易过大
        t1 = np.linalg.norm(feature1)
        t2 = np.linalg.norm(feature2)
        feature11 = feature1 / t1
        feature22 = feature2 / t2
        dis = np.linalg.norm(feature11 - feature22)
        return dis

    def angle(feature1, feature2):
        t1 = np.linalg.norm(feature1)
        t2 = np.linalg.norm(feature2)
        # 将特征展平为一维
        feature11 = feature1.flatten() / t1
        feature22 = feature2.flatten() / t2
        # 计算点积
        cnt = np.dot(feature11, feature22)
        return cnt


    # 展示与目标图片最相似的图片
    def show_similar_images(target_path, similar_images):
        """
        显示目标图片以及与之最相似的图片
        :param target_path: 目标图片路径
        :param similar_images: 与目标图片相似的图片列表，每个元素为 [距离，文件名]
        """
        # 创建画布
        num_images = len(similar_images) + 1  # 包括目标图片
        plt.figure(figsize=(15, 5))

        # 显示相似图片
        for idx, (_, filename) in enumerate(similar_images):
            plt.subplot(1, num_images, idx + 2)
            similar_img = Image.open(os.path.join('piclib', filename))
            plt.imshow(similar_img)
            plt.title(f"Similar {idx + 1}")
            plt.axis("off")

        plt.tight_layout()
        plt.show()


    distance = []
    # 调用函数去计算不同的欧氏距离情况比较
    start = time.time()
    for i in range(1,210):
        filename = str(i) + '.png'
        print('Prepare image data:  '+ filename + "  !")
        # 此处读入i.png的图片，然后进行与前面待匹配图像一样的操作，故可略
        temp_image = default_loader(os.path.join('piclib',filename))
        test_image = trans(temp_image)
        test_image = torch.unsqueeze(test_image, 0)

        test_feature = features(test_image)
```

```
264         test_feature = test_feature.detach().numpy()
265 
266         # 欧氏距离内存下的为具体数值以及文件名
267         # distance.append([dist(image_feature,test_feature),filename])
268         # 角度下的为具体数值及文件名
269         distance.append([angle(image_feature,test_feature),filename])
270 
271     print('Time for extracting features: {:.2f}'.format(time.time() - start))
272 
273     for i in range(len(distance)):
274         for j in range(len(distance)):
275             # 如果是dist，则为<；如果是角度 则为>
276             if(distance[i][0]>distance[j][0]):
277                 distance[i], distance[j] = distance[j], distance[i]
278     for i in range(5):
279         print(distance[i])
280 
281 
282     # 调用展示函数
283     target_image_path = 'white_dog.png'
284     most_similar_images = distance[:5]  # 取出距离最近的5张图片
285     # 显示目标图片
286     target_img = Image.open(target_image_path)
287     plt.imshow(target_img)
288     plt.title("Target Image")
289     plt.axis("off")
290     plt.show()
291     show_similar_images(target_image_path, most_similar_images)
```