lab5.1-PyTorch 入门 &lab5.2-Search by CNN features 实验报告

电院 2303 徐恺阳 523030910085

December 2024

1 实验概览

lab5.1 实验内容如下：

1. 机器学习与深度学习；
2. PyTorch 深度学习训练。

lab5.2 实验内容如下：

1. 图像检索；
2. 卷积神经网络；
3. 使用 Pytorch 提取图像特征。

2 实验环境

本实验基于 Python 3.10，用到的库有 OpenCV、Numpy、Matplotlib 和 Pytorch。

3 练习一

3.1 题目描述

利用深度学习对 CIFAR-10 图片进行分类。CIFAR-10 数据集由 10 个类的 60000 个 32x32 彩色图像组成，每个类有 6000 个图像。共有 50000 个训练图像和 10000 个测试图像。我们采用简单的 ResNet20 （一个 20 层的神经网络）作为模型，在 50000 个训练图片上对其参数进行更新。

注意：训练时，我们只能使用 50000 个训练图像。这样，在测试时模型没见过这 10000 张测试图片，其准确率评估才准确。

1. 补充 exp2.py 的代码，使得程序可以完整运行；
2. 使用 resnet20 模型，训练一个 cifar-10 的分类器；
3. 展示 test_acc 变化趋势，以各种可能的方式使最终的测试准确率尽可能高。

推荐训练策略：以 0.1 的学习率 (learning rate, lr) 训练 5 个 epoch，再以 0.01 的 lr 训练 5 个 epoch。

3.2 解题思路

Step1: 模型搭建

搭建 ResNet20，已在 models.py 中实现。

Step2: 补全 test(epoch) 函数

仿照 train(epoch) 函数，

- inputs, targets: 获取当前批次的输入和目标标签。
- outputs = model(inputs): 将输入数据通过模型前向传播，得到输出。
- loss = criterion(outputs, targets): 使用损失函数（criterion）计算输出和目标标签之间的损失。
- test_loss += loss.item(): 将当前批次的损失累加到总损失中。
- _, predicted = outputs.max(1): 从模型输出中获取预测结果，
- max(1) 表示在维度 1（通常是类别维度）上找到最大值的索引，这些索引即为预测的类别。
- total += targets.size(0): 增加总样本数。
- correct += predicted.eq(targets).sum().item(): 增加正确预测的数量。

补全代码如下：

```
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()
    test_accuracies.append(100. * correct / total)
acc = 100. * correct / total
print('Test Loss: %.3f | Test Acc: %.3f%% (%d/%d)' % (test_loss / (batch_idx + 1), acc, correct, total))
```

图 1: 补全代码

Step3: 可视化 train_acc 和 test_acc 变化趋势

分别用 train_accuracies 和 test_accuracies 记录每一次 epoch 中 train_acc 和 test_acc 的值，并在训练完成后进行可视化。

代码如下：

```
plt.figure(figsize=(10, 6))
plt.plot(range(10), train_accuracies, marker='o', linestyle='-', label='Train Accuracy', color='blue')
plt.plot(range(10), test_accuracies, marker='s', linestyle='--', label='Test Accuracy', color='orange')
plt.title("Train and Test Accuracy Trend During Training", fontsize=16)
plt.xlabel("Epoch", fontsize=14)
plt.ylabel("Accuracy (%)", fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)
plt.xticks("epoches", fontsize=12)
plt.yticks(fontsize=12)
plt.legend(fontsize=12)
plt.tight_layout()
plt.savefig("Train and Test Accuracy Trend During Training.jpg")
```

图 2: 可视化部分代码

3.3 代码运行结果

训练正确率和测试正确率如下：

```
Epoch [9] Batch [388/391] Loss: 0.677 | Traininig Acc: 76.375% (37931/49664)
Epoch [9] Batch [389/391] Loss: 0.677 | Traininig Acc: 76.384% (38033/49792)
Epoch [9] Batch [390/391] Loss: 0.677 | Traininig Acc: 76.398% (38138/49920)
Epoch [9] Batch [391/391] Loss: 0.676 | Traininig Acc: 76.408% (38204/50000)
==> Testing...
Test Loss: 0.683 | Test Acc: 76.360% (7636/10000)
Test Acc: 76.360000
```

图 3: 训练准确率和测试准确率

train_acc 和 test_acc 变化趋势如下：

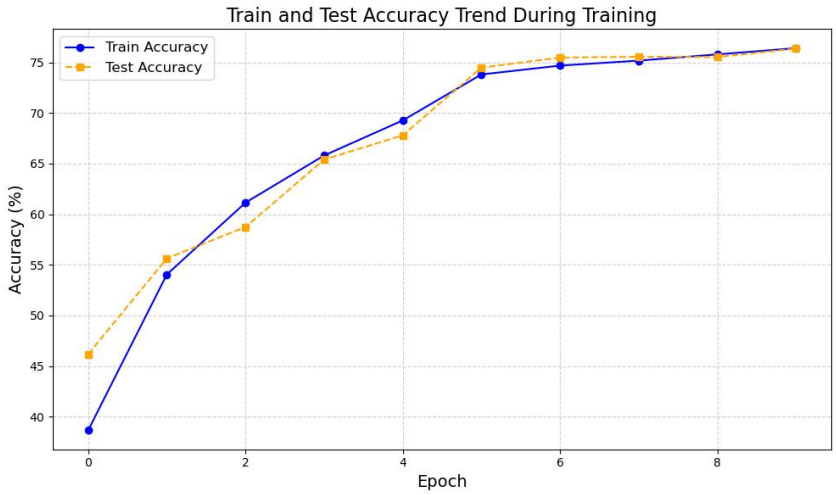

图 4: train_acc 和 test_acc 变化趋势

3.4 分析与思考

观察到，进入第二阶段的迭代之前，训练正确率仍上升较快，且有较高上升空间，所以尝试增大两个阶段的 epoach，并且调整每个阶段的 lr，以提高测试准确率。

- 第一阶段：以 0.15 的 lr 迭代 8 轮；
- 第二阶段：以 0.05 的 lr 迭代 8 轮；

训练正确率和测试正确率如下：

```
Epoch [15] Batch [388/391] Loss: 0.435 | Traininig Acc: 84.810% (42120/49664)
Epoch [15] Batch [389/391] Loss: 0.435 | Traininig Acc: 84.807% (42227/49792)
Epoch [15] Batch [390/391] Loss: 0.435 | Traininig Acc: 84.818% (42341/49920)
Epoch [15] Batch [391/391] Loss: 0.435 | Traininig Acc: 84.820% (42410/50000)
==> Testing...
Test Loss: 0.541 | Test Acc: 81.880% (8188/10000)
Test Acc: 81.880000
```

图 5: 训练准确率和测试准确率

- 第一阶段：以 0.15 的 lr 迭代 12 轮；
- 第二阶段：以 0.05 的 lr 迭代 8 轮；

训练正确率和测试正确率如下：

```
Epoch [19] Batch [387/391] Loss: 0.363 | Traininig Acc: 87.353% (43271/49536)
Epoch [19] Batch [388/391] Loss: 0.363 | Traininig Acc: 87.333% (43373/49664)
Epoch [19] Batch [389/391] Loss: 0.363 | Traininig Acc: 87.329% (43483/49792)
Epoch [19] Batch [390/391] Loss: 0.363 | Traininig Acc: 87.332% (43596/49920)
Epoch [19] Batch [391/391] Loss: 0.363 | Traininig Acc: 87.330% (43665/50000)
==> Testing...
Test Loss: 0.490 | Test Acc: 83.860% (8386/10000)
Test Acc: 83.860000
```

图 6: 训练准确率和测试准确率

train_acc 和 test_acc 变化趋势如下：

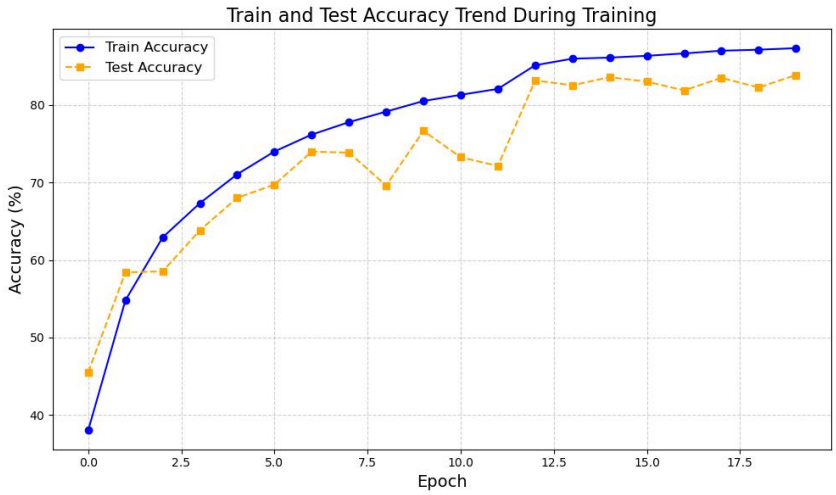

图 7: train_acc 和 test_acc 变化趋势

最终，测试准确率可达到 83.860%。

4 拓展思考

4.1 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？

答：

关联：

- 训练准确率 (Train Acc) 反映模型在训练集上的表现，它表示模型对已见过的训练数据的学习能力；
- 测试准确率 (Test Acc) 表明模型在未见过的测试集上的表现，反映模型的泛化能力。
- 如果训练过程正常且模型没有过拟合，Train Acc 和 Test Acc 应该在一定程度上保持一致，随着训练轮次的增加，它们都应该逐步提高。

不同点：

- 训练集用于模型参数的优化，因此模型会更擅长在训练集上进行预测；
- 测试集是未参与训练的数据，模型对其预测的准确性通常较低；

- 因此，Train Acc 通常高于 Test Acc。

lr 从 0.1 变到 0.01 后，

- 学习率从 0.1 下降到 0.01 后，Train Acc 和 Test Acc 都会逐步趋于平稳；
- Train Acc 增长减缓或趋于稳定，表明模型对训练集的拟合已经接近充分；
- Test Acc 可能会继续缓慢上升，表明模型在测试集上的泛化能力得到了进一步优化。

5 练习二

5.1 题目描述

构建一个不小于 50 张的图像库，用不限于 Resnet50 的模型提取图像的特征，并使用上一页 PPT 提到的方法，计算图片之间的相似度。

使用一些不在图像库中的图片进行测试，完成以图搜图的检索任务。给出检索的结果，与被检索图片 Top5 相似的图片和排序，说明排序的方法，给出排序的得分情况。

5.2 解题思路

Step0: 模型导入

分别导入 ResNet50、alexnet 和 vgg13 模型，并加载预训练好的模型，代码如下：

```
print("Load model: ResNet50")
resnet50 = torch.hub.load("pytorch/vision", "resnet50", pretrained=True)
print("Load model: alexnet")
alexnet = torch.hub.load("pytorch/vision", "alexnet", pretrained=True)
print("Load model: vgg13")
vgg13 = torch.hub.load("pytorch/vision", "vgg13", pretrained=True)
```

图 8: 模型加载

Step1: 数据预处理

定义处理图片的归一化操作和预处理方式，代码如下：

```
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,
])
```

图 9: 数据预处理

Step2: 提取图像的特征

以 ResNet50 为例，ResNet50 包含以下几个主要模块:

1. conv1

2. bn1
3. relu
4. maxpool
5. layer1
6. layer2
7. layer3
8. avgpool
9. fc

我们把模型最终分类的前一层，即 avgpool 层，当作模型学习到的图像特征，并用这种特征来完成我们的图像检索任务。因此，我们需要定义一个特征提取函数来提取模型倒数第二层的输出结果，代码如下：

```
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
```

图 10: 特征函数

alexnet 和 vgg13 模型对应的特征提取函数省略，直接调用 feature() 函数实现，代码如下：

```
def alexnet_features(x):
    x = alexnet.features(x)
    x = alexnet.avgpool(x)

    return x

def vgg13_features(x):
    x = vgg13.features(x)
    x = vgg13.avgpool(x)

    return x
```

图 11: 特征函数

Step3: 提取图像的特征

以 ResNet50 为例，对于 Dataset 中的每一张图片，分别利用 Resnet50 的模型提取图像的特征，保存至 Resnet50_features 目录下，代码如下：

```
for i in range(1, 51):
    img_path = "Dataset\\" + str(i) + ".jpg"
    test_image = default_loader(img_path)
    input_image = trans(test_image)
    input_image = torch.unsqueeze(input_image, 0)
    start = time.time()
    image_feature = resnet50_features(input_image)
    image_feature = image_feature.detach().numpy()
    image_feature = normalization(image_feature)
    print(f"Time for extracting features for {str(i)}.jpg: {time.time() - start:.2f}")
    print("Save features for " + str(i) + ".jpg !")
    save_path = "ResNet50_features/features" + str(i)
    np.save(save_path, image_feature)
```

图 12: 特征提取

Step4: 计算图片库中两两图片之间的相似度

这里，我们采用“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”的方法，代码如下：

```
def calculate_similarity(features_path, num_images):
    # 加载所有图片的特征向量
    features = []
    for i in range(1, num_images + 1):
        feature_file = os.path.join(features_path, f"features{i}.npy")
        feature = np.load(feature_file)
        features.append(feature)

    features = np.array(features)

    # 初始化相似度矩阵
    similarity_matrix = np.zeros((num_images, num_images))

    # 归一化特征向量
    normalized_features = features / np.linalg.norm(features, axis=1, keepdims=True)

    # 计算两两图片之间的欧氏距离
    for i in range(num_images):
        for j in range(num_images):
            if i != j:
                distance = np.linalg.norm(normalized_features[i] - normalized_features[j])
                similarity_matrix[i, j] = distance

    return similarity_matrix
```

图 13: 计算两两图片之间的相似度

Step5: 以图搜图

这里，采用示例中的 panda.png 作为测试图片，以图片间的欧氏距离作为得分情况，得分越低，说明越相近。代码如下：

```
def calculate_similarity(test_feature, features):
    similarities = []
    for feature in features:
        distance = np.linalg.norm(test_feature - feature)
        similarities.append(distance)
    return similarities
```

图 14: 计算测试图片与图片库中图片之间的相似度

5.3 代码运行结果

分别以 resnet50、alexnet 和 vgg13 作为模型提取特征，与被检索图片 Top5 相似的图片和排序结果如下：

```
----------restnet50----------
Top 5 similar images and their scores:
Image 34.jpg: Score 20.74
Image 14.jpg: Score 20.74
Image 37.jpg: Score 20.74
Image 22.jpg: Score 20.74
Image 9.jpg: Score 20.74
```

图 15: 以 resnet50 为模型的 Top5 相似的图片和排序

```
----------alexnet----------
Top 5 similar images and their scores:
Image 34.jpg: Score 280.25
Image 6.jpg: Score 280.25
Image 1.jpg: Score 280.27
Image 8.jpg: Score 280.28
Image 49.jpg: Score 280.29
```

图 16: 以 alexnet 为模型的 Top5 相似的图片和排序

```
----------vgg13----------
Top 5 similar images and their scores:
Image 34.jpg: Score 302.13
Image 8.jpg: Score 302.20
Image 49.jpg: Score 302.20
Image 6.jpg: Score 302.20
Image 32.jpg: Score 302.21
```

图 17: 以 vgg13 为模型的 Top5 相似的图片和排序

5.4 分析与思考

panda.png 和与三种模型下与其最相似的图片库中的图片如下：

图 18: panda.png

图 19: Top1

可以看出，识别结果主要以颜色大体分布为分类依据。

6 实验感想

这次实验让我对深度学习和卷积神经网络有了更深入的理解。通过 PyTorch 框架，我了解了如何在 CIFAR-10 数据集上训练模型和提取图像特征。

此外，我通过使用 ResNet50 和 AlexNet 等模型提取图像特征，加深了对深度学习模型的理解。通过逐层分析模型的工作机制，我了解了卷积层、池化层及全连接层的作用，尤其是如何利用预训练模型进行特征提取和迁移学习。