机器学习实验报告——神经网络

221900180 田永铭

2024 年 4 月 24 日

目录

一、实验内容

使用深度神经网络实现对图像数据集的分类。

数据集：Caltech 101

采用模型：VGG

二、实验环境

- 环境工具：Anaconda
- python 环境：python 3.10
- pytorch 版本：CUDA 11.8
- GPU 环境：本实验采用 GPU 跑，CUDA 版本 12.1，驱动器版本 551.78
- 代码运行平台:jupyter notebook

三、实验原理

以下是本次实验用到的实验原理：

深度学习模型：

深度学习模型是一类人工神经网络，通过多层非线性变换来对数据进行建模和学习。其中，卷积神经网络（CNN）是一种主要用于处理图像数据的深度学习模型。VGG-16 是一个经典的深度学习模型，具有 16 层卷积和全连接层，被广泛用于图像分类任务。

数据集：

Caltech-101 数据集包含 101 个不同类别的图像，用于图像分类任务的基准测试。每个类别都包含数百张图像，总共包含约 9,000 张图像。

图像预处理：

图像预处理是在训练之前对图像进行的一系列操作，以提高模型的性能和鲁棒性。在本实验中，采用了随机裁剪、水平翻转、颜色调整等预处理操作，以增加数据的多样性和泛化能力。

模型训练与评估：

模型训练包括选择合适的损失函数（如交叉熵）、优化器（如随机梯度下降）和学习率调度器。训练过程通过最小化损失函数来调整模型参数，以使模型能够更好地拟合训练数据。在每个训练周期结束时，评估模型在测试集上的性能，通常使用准确率作为评估指标。

t-SNE 降维：

t-SNE 是一种流行的非线性降维技术，用于将高维数据映射到二维或三维空间以进行可视化。t-SNE 通过在高维空间中保持样本之间的局部关系来生成低维表示，使得样本在可视化空间中的分布能够反映其在高维空间中的相似性。

四、根据要求依次汇报

汇报主要分为以下几个方面：

1. 使用的分类模型，以及网络结构的描述
2. 训练时候所使用的优化器，学习率，epoch 数等细节信息
3. 编写代码
4. 在 Caltech 101 数据集上的训练精确度和测试精度
5. 使用预训练的参数进行微调训练, 并且绘制训练精确度曲线、训练损失曲线
6. 描述训练时数据增广策略是什么，使用此数据增广时，最终测试精度是多少？
7. 更换换超参数训练
8. 加分项：使用 TSNE 技术可视化训练样本的特征，可以任取 10 个类的样本进行可视化
9. 加分项：训练中遇到的任何异常情况和调试过程

4.1 使用的分类模型，以及网络结构的描述

我使用的是 VGG-16 这个经典的深度学习模型. 其名称中的"VGG" 表示 Visual Geometry Group，数字"16" 表示网络包含的层级数量。

输入层：

输入层接受输入图像数据，通常是三通道（RGB）的彩色图像。

卷积层部分：

VGG-16 网络以一系列卷积层开始，这些卷积层使用小型的卷积核来提取图像特征。连续使用多个 3x3 大小的卷积核，每个卷积核输出一个通道，然后再将多个输出通道堆叠起来形成一个特征图。网络中有 13 个卷积层，其中包括一些具有 2x2 大小的最大池化层，用于降低特征图的空间维度，并增加网络的平移不变性。

全连接层部分：

在卷积层之后是三个全连接层，这些层将卷积层的输出特征图转换为最终的分类结果。全连接层包括两个隐藏层和一个输出层，其中输出层的神经元数量等于数据集中的类别数。为了减少过拟合，全连接层之间通常会添加一些正则化方法，如 Dropout。

激活函数：

在每个卷积层和全连接层之间，VGG-16 使用修正线性单元（ReLU）作为激活函数，以引入非线性特征。

输出层：

输出层是一个具有 Softmax 激活函数的全连接层，用于将模型的原始输出转换为概率分布，以便进行多类别分类。

4.2 训练时候所使用的优化器，学习率，epoch 数等细节信息

- **优化器：**使用了随机梯度下降（SGD）优化器。SGD 优化器采用了动量（momentum）参数为 0.9。正则化参数 weight_decay 设置为 2e-4。
- **学习率：**由于是否选择预训练、学习的代数不同，我的学习率设置的不同，这会在后面详细讲出。
- **Epoch 数：**同上，由于是否选择预训练，我的训练代数也不同，为 10 代或者 20 代，后面会详细讲出。

4.3 编写代码

导入所需的包：

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import models
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms
from tqdm import tqdm
```

下载数据集，并且定义训练和测试图像变换操作

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

```
 std=[0.229, 0.224, 0.225])
])
```

定义数据集和数据加载器

```
dataset_path = 'D:/@extremely_important_files/class_and_homework/
sophomore_second_semester/Machine␣Learning/homework/
02/dataset/caltech-101/caltech-101/101_ObjectCategories'
dataset = datasets.ImageFolder(
root=dataset_path, transform=train_transform)

train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(
dataset, [train_size, test_size])

test_dataset.dataset.transform = test_transform

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

定义从头训练的 VGG 模型：

```
class VGG(nn.Module):
def __init__(self, features, num_classes=1000, init_weights=False):
super(VGG, self).__init__()
self.features = features                          # 卷积层提取特征
self.classifier = nn.Sequential(          # 全连接层进行分类
nn.Dropout(p=0.5),
nn.Linear(512*7*7, 2048),
nn.ReLU(True),
nn.Dropout(p=0.5),
nn.Linear(2048, 2048),
nn.ReLU(True),
nn.Linear(2048, num_classes)
)
if init_weights:
self._initialize_weights()
```

```
def forward(self, x):
# N x 3 x 224 x 224
x = self.features(x)
# N x 512 x 7 x 7
x = torch.flatten(x, start_dim=1)
# N x 512*7*7
x = self.classifier(x)
return x

def _initialize_weights(self):
for m in self.modules():
if isinstance(m, nn.Conv2d):
# nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
nn.init.xavier_uniform_(m.weight)
if m.bias is not None:
nn.init.constant_(m.bias, 0)
elif isinstance(m, nn.Linear):
nn.init.xavier_uniform_(m.weight)
# nn.init.normal_(m.weight, 0, 0.01)
nn.init.constant_(m.bias, 0)
```

定义训练与测试流程（这步有缺陷，在测试集上做验证的时候有冗余代码，后面删掉了）：

```
def train_model(num_epochs, model, criterion, optimizer,
 scheduler,train_loader, device):
model.train()
for epoch in range(num_epochs):
print(f'training␣on␣epoch:{epoch}')
running_loss = 0.0
total = 0
correct = 0
for images, labels in tqdm(train_loader):
images, labels = images.to(device), labels.to(device)
optimizer.zero_grad()
outputs = model(images)
```

```
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
running_loss += loss.item() * images.size(0)
_, predicted = torch.max(outputs, 1)
total += labels.size(0)
correct += (predicted == labels).sum().item()
scheduler.step()
epoch_loss = running_loss / len(train_loader.dataset)
epoch_accuracy = 100 * correct / total
print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}, Train Accuracy: {epoch_accuracy:.2f}%')

def evaluate_model(num_epochs, model,
 criterion, test_loader, device):
model.eval()  # 将模型设置为评估模式
with torch.no_grad():  # 在评估过程中不需要计算梯度
total = 0
correct = 0
for epoch in range(num_epochs):
print(f'evaluating on epoch:{epoch}')
for images, labels in tqdm(test_loader):
images, labels = images.to(device), labels.to(device)
outputs = model(images)
_, predicted = torch.max(outputs, 1)
total += labels.size(0)
correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f'Test Accuracy: {accuracy:.2f}%')
```

主函数，实例化模型，加载到 gpu 上训练

```
# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
# 实例化模型
```

```
model = VGG(models.vgg16(weights=None).features, num_classes=102, init_weights=T
# 采用gpu
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
# Define loss function and optimizer
epochs = 10
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(
model.parameters(), lr=0.01, momentum=0.9, weight_decay=2e-4)
scheduler = optim.lr_scheduler.CosineAnnealingLR(
optimizer, T_max=epochs)
train_model(epochs, model, criterion, optimizer,
scheduler, train_loader, device)
evaluate_model(epochs,model, criterion, test_loader, device)
```

4.4 在 Caltech 101 数据集上的训练精确度和测试精度

注意：此处我还没有加上画图代码，这将在后面一部分展示

运用以上初代代码，我分别训练了 10 代和 20 代，得到的训练准确度和测试精度如下图所示：

10 代：测试集合 66.34% 准确度，训练集 55.60% 准确度

20 代：测试集合 97.51% 准确度，训练集 57.52% 准确度

4.5 使用预训练的参数进行微调训练，并且绘制训练精确度曲线、训练损失曲线

接下来，为了更高的精度，我们采用预训练好的 VGG 参数进行微调，并且绘制曲线图，修改关键部分代码如下：

```
# 定义模型
# 加载torch原本的vgg16模型，设置pretrained=True，即使用预训练模型
model = models.vgg16(pretrained='Default')
num_fc = model.classifier[6].in_features # 获取最后一层的输入维度
# 修改最后一层的输出维度，即分类数
model.classifier[6] = torch.nn.Linear(num_fc, 102)
# 对于模型的每个权重，使其不进行反向传播，即固定参数
```

图 1: VGG_without_pretraining_10_epochs

图 2: VGG_without_pretraining_20_epochs

```
for param in model.parameters():
param.requires_grad = False
# 将分类器的最后层输出维度换成了num_cls，这一层需要重新学习
for param in model.classifier[6].parameters():
param.requires_grad = True
# 定义全局变量用于记录训练过程中的精度和损失
```

```
train_losses = []
train_accuracies = []
# 定义全局变量用于记录每个epoch的特征向量和对应的标签
all_features = []
all_labels = []

#在训练模型的时候加上：
# 保存特征向量和标签
all_features.append(model.features(images).detach().cpu().numpy())
all_labels.append(labels.detach().cpu().numpy())
# 保存数据
train_losses.append(epoch_loss)
train_accuracies.append(epoch_accuracy)

# 绘制训练集精度和损失曲线
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(train_accuracies, label='Train Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.title('Training Accuracy')

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

plt.show()
```

由此训练 10 代，发现训练集精度能达到 93.71%，测试集精度能达到 85.89%。观察绘制的图片（如下），发现训练准确度是凹函数，但逐渐上升；训练损失为凸函数，逐渐下降。(此步学习率为 0.01)。我发现，训练的速度急剧上升，只需要微调就可以几代达到 90% 的准确度。10 代已经达到接近 90% 的准确度，而且还没收敛，所以增加训练的代数，包括下一部分采用数据增广策略，都能再增加这个数字。经过和助教讨论，单纯讨论达到的最大精度是没有意义的，我已经通过预训练达到了很好地效果。

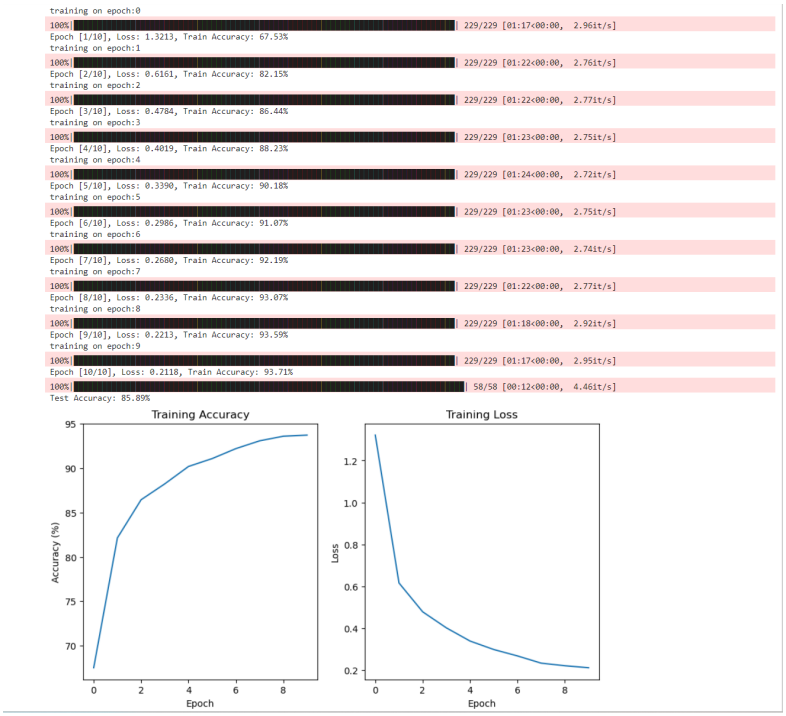

图 3

4.6 描述训练时数据增广策略是什么，使用此数据增广时，最终测试精度是多少

为了更好的效果，我进行数据增广，策略是添加添加随机水平翻转，随机颜色调整和添加随机旋转，代码如下：

```
transforms.RandomHorizontalFlip(),  # 添加随机水平翻转
transforms.ColorJitter(
brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),  # 随机颜色调整
transforms.RandomRotation(30),  # 添加随机旋转
```

使用此数据增广，训练 10 代后测试集精度为 90.38%。相比之前提升 5 个百分点，效果比较明显。

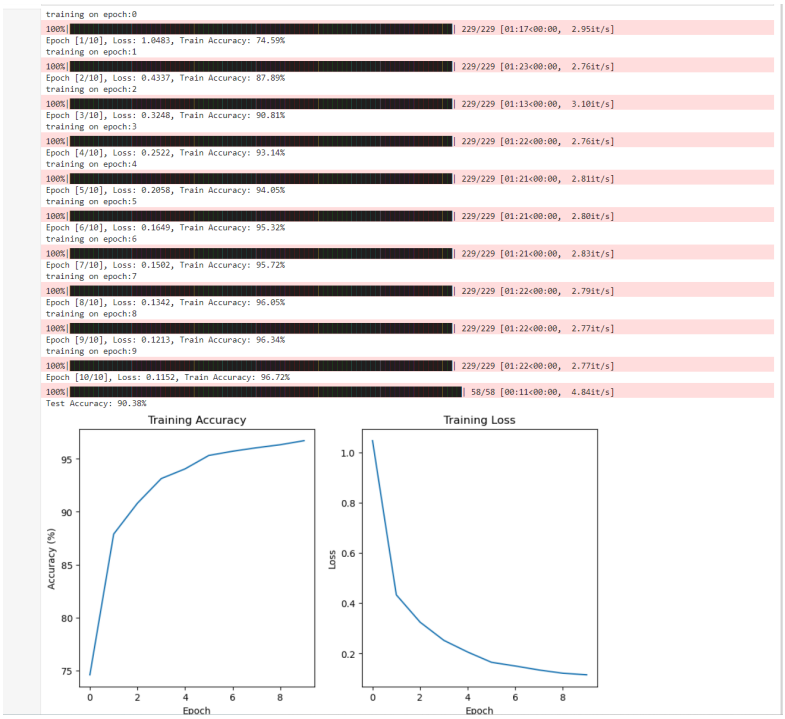

图 4

4.7 更换换超参数训练

因为用了预训练模型，所以学习率应该略微降低，学习率从 0.01 分别改为 0.0001 和 0.001，测试集准确度分别是 75.62% 和 89.61%, 由此知，学习率在 0.01 到 0.001 之间较好，0.0001 学习速度太慢，10 代的效果不好。

4.8 加分项：使用 TSNE 技术可视化训练样本的特征，可以任取 10 个类的样本进行可视化

采用 TSNE 技术可视化训练样本的特征，任取 10 个类的样本进行可视化，代码如下：

```
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
features = []
labels = []

with torch.no_grad():
for i, (image_batch, label_batch) in enumerate(test_loader):
```

```
image_batch, label_batch = image_batch.cuda(), label_batch.cuda()
label_batch = label_batch.long().squeeze()
inputs = image_batch
logits = model(inputs)  # 只返回logits, 不需要feature
if i == 0:
logits_bank = logits
label_bank = label_batch
else:
logits_bank = torch.cat((logits_bank, logits))
label_bank = torch.cat((label_bank, label_batch))

# 计算 t-SNE
logits_bank = logits_bank.cpu().numpy()
label_bank = label_bank.cpu().numpy()
tsne = TSNE(n_components=2)
output = tsne.fit_transform(logits_bank)

# 绘制 t-SNE 图
colors = plt.cm.Spectral(np.linspace(0, 1, 10))  # 生成10种不同的颜色
for i in range(10):    # 对每类的数据画上特定颜色的点
index = (label_bank == i)
plt.scatter(output[index, 0], output[index, 1], s=5, color=colors[i])
plt.legend(["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"])
plt.show()
```

由此，绘制出训练 10 代的可视化图片如下：

由此我们可以发现，经过降维，我们能直观地看到同一类样本靠的很近，不同类靠的比较远，完全达到了预期的结果。

4.9 加分项：训练中遇到的任何异常情况和调试过程

4.9.1 配置 GPU 环境加速训练

由于这次训练量不是很小，CPU 跑 1 代数据需要 15 分钟左右，这过于慢速。于是需要配置 GPU 环境，在配置过程中，我遇到了很多麻烦，例如 cuda 和 pytorch 和我的 gpu 配置不兼容的问题，通过查阅资料最终解决，协调好了版本兼容问题，并且利用 anaconda 配置好了实验环境，达到 1 分半训练 1 代的速度。参考文献为GPU 版本 pytorch 安装教程

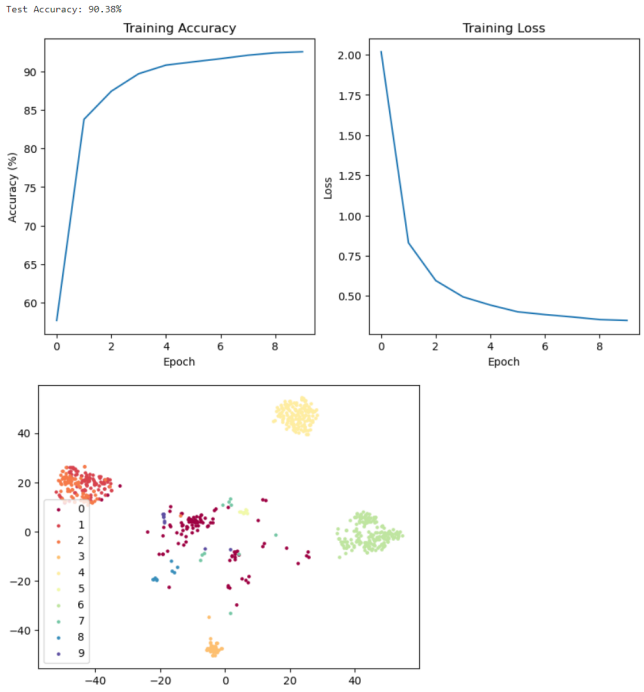

图 5

4.9.2 vgg16 预训练模型导入问题

一开始，由于疏忽，我并没有成功导入 vgg16 模型，后来想通过先下载好模型再导入，但是总是导入不成功。最终选择的是利用 pytorch 自带的 vgg16 模型，这一下给我的训练精度带来了很大的提升。

4.9.3 OSError: [WinError 127] 报错

由于训练途中为了下载一个奇怪的包，我将环境都更新到了最新，导致我的 cuda 和 pytorch 版本又不兼容了，报错 OSError: [WinError 127]，在此之后我又重新配置了一遍环境，成功解决。

4.9.4 jupyter notebook 启动后一片空白

我还遇到了 jupyter notebook 启动后一篇空白的问题，经过查阅资料知道，可以通过清除缓存区或者刷新页面来解决。

4.9.5 conda 显示有 torch，但是 jupyter notebook 显示没有这个模块

查阅资料得知出错原因：jupyter 和终端的 python.exe 的路径不一样。解决方案：打开 cmd，输入以下命令，安装 nb_conda_kernels 包。我直接在 anaconda 虚拟环境中安装了一个 kernel，就成功解决了。

4.9.6 tsne 可视化

一开始，我的 tsne 可视化画图始终报错或者没有绘图结果。我选择每次只训练 1 代，对代码进行调试。我解决了以下问题：

1. TSNE 报错: 因为 perplexity 参数必须小于样本数量。让我们检查一下 perplexity 参数的设置。在 TSNE 初始化时，默认 perplexity 是 30。但在数据集很小的情况下，如果选择的样本数量比 perplexity 更小，就会出现这个错误。解决方法是降低 perplexity 或者增加选择的样本数量。由于 perplexity 默认值是 30，我们可以尝试将 perplexity 设置为一个更小的值 10。另外，我们也可以尝试增加选择的样本数量，确保 perplexity 小于样本数量。
2. model 的输出中返回了超出了预期的值: 通常情况下，模型的输出应该是一个张量，但是在这里，似乎模型的输出返回了两个值，导致了 ValueError。model(inputs) 应该只返回一个张量，即模型的输出 logits。这个错误可能是因为使用了错误的模型结构或者模型的 forward 方法返回了不正确的值。解决方法：在这个修改后的代码中，只保留模型输出的 logits，而不再使用 feature。

五、总结

通过这次实验，我成功使用深度神经网络完成了对图像数据集的分类。在此过程中，我综合解决了各种问题，实现了各种功能，能力得到了提升，也对机器学习更加感兴趣了。