实验二：卷积神经网络实现

姓名：李旻翀
学号：1190200208

日期：May 25, 2022

摘　　要

本实验为模式识别与深度学习课程的实验二：卷积神经网络实现，主要任务是基于 PyTorch 实现 AlexNet 网络结构，并在 CalTech 101 数据集上验证。在本实验中，我复现并修改了 AlexNet 的网络结构，并使用 tensorboard 进行训练数据可视化，最终在数据集上达到了 0.72 的准确率。

关键词: 模式识别与深度学习，AlexNet，CalTech101 数据集，tensorboard

1　深度学习框架与实验环境

本次实验的环境为 Python3.8 + PyTorch1.11.0 + cuda11.3 + cudnn8.2.1，与实验一相同。

2　实验背景知识

2.1　AlexNet 简介

Alexnet 模型由 5 个卷积层，3 个池化层，3 个全连接层构成。AlexNet 是 LeNet 神经网络的发展，但有了许多新的亮点:

1. AlexNet 引入了数据增广技术，可以对图像进行颜色变换、裁剪、翻转等操作。
2. 采用 ReLU 激活函数代替 Sigmoid，提升了训练速度，并在一定规模数据上的性能表现超过了使用 Sigmoid 的网络模型。
3. AlexNet 引入了 Dropout 用于解决模型训练过程中容易出现过拟合的问题。

AlexNet 的网络结构如下:

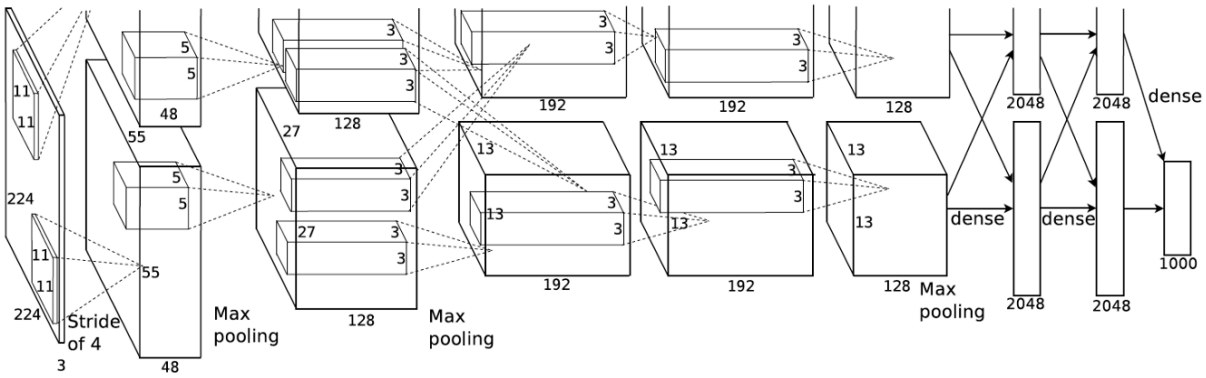

图 1: AlexNet 网络结构

2.2 CalTech 101 数据集介绍

Caltech 101 是图像识别分类常用数据集。其总共包含 9146 张图像, 分为 101 个不同类别（如 face, piano, ant 等）和背景类别。与图像一起提供的还有一组注释，描述了每个图像的轮廓。

在本实验中，按照实验要求去掉了 Caltech101 数据集中的 BACKGROUND_Google 类。用剩下的数据完成实验。

3 实验过程

3.1 搭建网络

AlexNet 原论文中采用两块 GPU 进行运算后再拼接向量（见图1），由于设备所限，我只实现了一块 GPU 对应的网络结构。

具体而言，我搭建的网络结构分为 features 与 classifier 两部分。features 部分负责提取图片中的特征，而 classifier 部分负责具体的图片分类。

features 部分的网络结构如下:

- 卷积层 1

  输入：224 × 224 × 3 的图像；卷积核数量：48（对应单片 GPU）; 卷积核大小：11 × 11 × 3; stride = 4（步长为 4）；padding = 2（表示扩充边缘两行两列）; 完成卷积后，接入一层 ReLU 并进行 max pooling。max pooling 的参数为 kernal_size = 3, stride = 2, padding = 0。

  输出：55 x 55 x 48 的 feature。

- 卷积层 2

  输入：55 x 55 x 48 的 feature（上一层的输出）；卷积核数量：128; 卷积核大小：5 × 5 × 48; stride = 1；padding = 2; 完成卷积后，同样接入 ReLU 并进行 max pooling（max pooling 参数同之前）。

  输出：27 x 27 x 128 的 feature。

- 卷积层 3

  输入：上一层的输出；卷积核数量：192; 卷积核大小：3 × 3 × 128; stride = 1；padding = 1; 完成卷积后，接入 ReLU，但不进行 max pooling。

  输出：13 x 13 x 192 的 feature。
- 卷积层 4

  输入：上一层的输出；卷积核数量：192; 卷积核大小：3 × 3 × 192; stride = 1；padding = 1; 完成卷积后，接入 ReLU，但不进行 max pooling。

  输出：13 x 13 x 192 的 feature。
- 卷积层 5

  输入：上一层的输出；卷积核数量：128; 卷积核大小：3 × 3 × 192; stride = 1；padding = 2; 完成卷积后，接入 ReLU 并进行 max pooling（max pooling 参数同之前）。

  输出：27 x 27 x 128 的 feature。

完成上述步骤后，我们可以得到 13 x 13 x 128 的特征，将其通过 flatten 方法展平后输入 classifier 部分，该部分网络结构如下：

- 全连接层 1

  输入：128 × 6 × 6 的图像；在进入该层前，设置 dropout 为 0.5,；完成后，接入一层 ReLU。

  输出：长度为 2048 的一维向量。
- 全连接层 2

  输入：长度为 2048 的一维向量（上一层的输出）；在进入该层前，设置 dropout 为 0.5,；完成后，接入一层 ReLU。

  输出：仍为 2048 的一维向量。
- 全连接层 3

  输入：上一层的输出。

  输出：长度为 num_class 的一维向量（在本实验中，对应 101 个类）。

3.2 数据集构建

由于 Caltech101 数据集不能通过 pytorch 直接导入，所以需要自己编写数据集类，以便训练时使用。

在本实验中，数据集类保存在 caltech101.py 中。完成的主要工作是对 __init__，__getitem__，__len__ 三个方法的重写，注意在 __getitem__ 方法中，需要利用 PIL 库读入图片，并将图片转为 RGB 格式，方便后续处理。

在完成上述步骤以后，便可以在主函数中调用该数据集。

3.3 模型参数

3.3.1 损失函数与优化器

为了取得相对较好的效果，采用交叉熵损失与 Adam 优化器。具体参数如下：

```
# 定义损失函数和优化器
loss_func = torch.nn.CrossEntropyLoss().cuda()
optimizer = torch.optim.Adam(params=model.parameters(), lr=0.0005)
```

3.3.2 超参数

超参数设定如下：

```
# 超参数
batch_size = 64
epoch = 50
```

3.4 引入 tensorboard

通过 conda 安装 tensorboard。在训练相关代码执行前，实例化一个 SummaryWriter 来保存相关信息。在训练过程中，将网络结构，当次训练损失，当次训练准确率等信息加入 SummaryWriter，在训练结束后即可查询。

在本次实验中，我将 loss, accuracy, 网络结构这三个数据写入 SummaryWriter，在训练结束之后可以在 tensorboard 中查看。

4 实验结果与分析

在本机 GPU 为 GTX 1660 Ti 6GB 的环境下，训练完 50 个 epoch 大致需要半小时，最终，AlexNet 在该数据集上取得了 0.72 的准确率，如图所示：

```
Epoch:42   Train Loss: 1.1613   Acc:0.5971
Epoch:43   Train Loss: 1.1174   Acc:0.6026
Epoch:44   Train Loss: 1.1133   Acc:0.6048
Epoch:45   Train Loss: 1.0912   Acc:0.6114
Epoch:46   Train Loss: 1.0715   Acc:0.6169
Epoch:47   Train Loss: 1.0457   Acc:0.6136
Epoch:48   Train Loss: 1.0006   Acc:0.5862
Epoch:49   Train Loss: 0.9902   Acc:0.5862
Epoch:50   Train Loss: 0.9882   Acc:0.6048
Finished
```

图 2: 训练结果

```
G:\Anaconda3\envs\py38_torch_cudnn\python.exe "C:/UserData/Desktop/DL Lab2/train.py"
Load model | Acc:0.7241
```

图 3: 模型在测试集上的表现

在程序根目录下运行命令：tensorboard –logdir=runs，在 tensorboard 中查看相关结果如下：

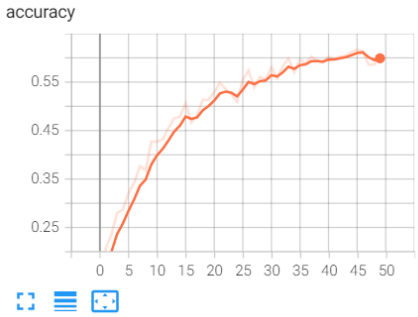

图 4: Accuracy

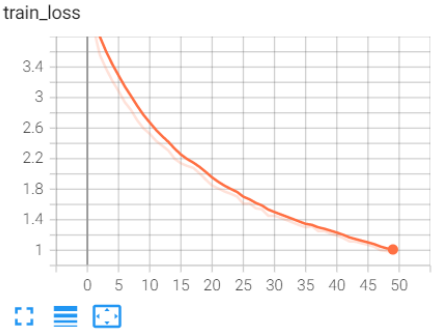

图 5: train_loss

5 说明文档

5.1 实验文件夹结构

- train.py：训练程序文件
- alexnet.py：自己实现的 AlexNet 类
- caltech101.py：自己实现的 Caltech101 数据集类
- ./data：保存 Caltech101 数据集（提交时已删除）
- ./model：保存训练好的模型（本实验中仅保留最优模型）
- ./utils：保存训练程序中用到的功能函数（数据集读取，划分，载入）
- ./runs：保存 tensorboard 日志文件

5.2 程序运行方法

运行 python train.py –mode train，会重新训练模型（需要助教重新下载数据集到./data 下）。
运行 python train.py –mode test，会载入保存的最优模型进行测试并打印结果。