

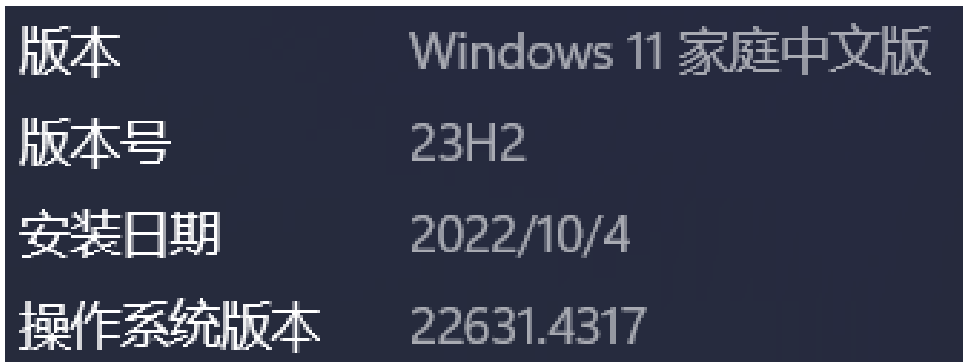
# 第 3 次实验报告

2022111663 刘博飞

2024 年 10 月 16 日

## 1 环境介绍

### 1. 操作系统:Windows 系统

A screenshot of the Windows system information window. It displays the following details: Version (Windows 11 家庭中文版), Version number (23H2), Installation date (2022/10/4), and Operating system version (22631.4317).

版本	Windows 11 家庭中文版
版本号	23H2
安装日期	2022/10/4
操作系统版本	22631.4317

图 1: 操作系统信息

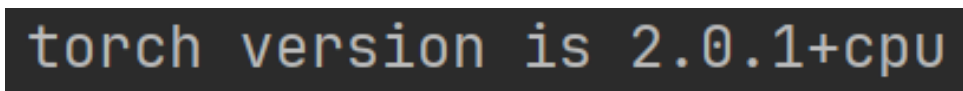
### 2. CPU 型号:

A screenshot showing the CPU information. The processor is identified as 11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz, with a current frequency of 2.50 GHz.

处理器	11th Gen Intel(R) Core(TM) i5-1155G7 @ 2.50GHz 2.50 GHz
-----	---

图 2: CPU 型号

### 3. torch 版本:

A screenshot of a terminal window showing the output of the command to check the PyTorch version. The output is 'torch version is 2.0.1+cpu'.

```
torch version is 2.0.1+cpu
```

图 3: torch 版本

#### 4. resize 尺寸:224x224

```
transform = transforms.Compose([
    transforms.Lambda(to_rgb), # 确保所有图像都有3个通道
    transforms.Resize((224, 224)),
```

图 4: resize 尺寸

## 2 任务 1: 基于 PyTorch 实现 AlexNet 结构

### 2.1 数据集下载与配置

从教学群下载 Caltech101 数据集, 得到 101\_ObjectCategories 文件夹, 存储了 102 类不同类型的图片, 但是发现不能直接从其中提取数据, 因此根据原论文说明配置了数据集文件夹, 结构如下:

```
D:\skill_files\code\python\pr_ml\lab3\data>tree
卷 Data 的文件夹 PATH 列表
卷序列号为 D4F0-14C8
D:.
├──caltech101
│   └──101_ObjectCategories
│       ├──accordion
│       ├──airplanes
│       ├──anchor
│       └──ant
```

图 5: 数据集文件夹结构

### 2.2 数据集加载

使用 dataset 和 Dataloader 定义并加载数据集, 发现 pytorch 中已经集成了 Caltech101 数据集的个性化加载方法, 其路径为: torchvision.datasets.Caltech101(root,download,transform).

遗憾的是, Caltech101 并没有划分训练集和测试集(验证集), 因此需要手动划分。这里使用了 pytorch 定义的集合划分函数 random\_split(dataset, [train\_size, test\_size]), 其路径为: torch.utils.data.random\_split. 本实验定义训练集占 80%, 测试集和验证集相同且占剩余的 20%.

值得注意的一点是, 在 Caltech101 数据集中同时含有彩色图像和灰色图像, 前者包含三通道信息, 而后者仅包含一通道, 因此需要将一通道转换为三通道, 采用了 image.convert('RGB') 函数进行转换。转换完成后再使用 resize 函数化成 224x224 尺寸。

最后使用 torch.utils.data.DataLoader(dataset, batch\_size, shuffle) 加载 MNIST 数据集, 在本实验中, DataLoader 有着如下的结构: [(batch\_size, 3, 224, 224), (batch\_size)] \* batch\_num, 在一个 batch 中, 前者是图像的色彩矩阵, 后者是每张图的标签。

## 2.3 网络框架搭建

AlexNet 网络结构是一个比较成熟的卷积神经网络结构，其组成如下：

输入图像  $3 \times 224 \times 224$

1. 卷积层，输入通道数 3，输出通道数 96，采用  $11 \times 11$  的卷积核，步幅为 4，输出结果为  $96 \times 54 \times 54$ ，再做非线性激活（ReLU 激活函数）
2. 池化层，通道数不变，卷积核  $3 \times 3$ ，步幅为 2，输出结果为  $96 \times 26 \times 26$
3. 卷积层，输入通道数 96，输出通道数 256，卷积核  $5 \times 5$ ，填充为 2，输出结果为  $256 \times 26 \times 26$ ，再做非线性激活
4. 池化层，通道数不变，卷积核  $3 \times 3$ ，步幅为 2，输出结果为  $256 \times 12 \times 12$
5. 卷积层，输入通道数 256，输出通道数 384，卷积核  $3 \times 3$ ，填充为 1，输出结果为  $384 \times 12 \times 12$ ，再做非线性激活
6. 卷积层，输入通道数 384，输出通道数 384，卷积核  $3 \times 3$ ，填充为 1，输出结果为  $384 \times 12 \times 12$ ，再做非线性激活
7. 卷积层，输入通道数 384，输出通道数 256，卷积核  $3 \times 3$ ，填充为 1，输出结果为  $256 \times 12 \times 12$ ，再做非线性激活
8. 池化层，通道数不变，卷积核  $3 \times 3$ ，步幅为 2，输出结果为  $256 \times 5 \times 5$
9. 全连接层，输入大小  $256 \times 5 \times 5 = 6400$ ，输出结果大小 4096
10. 全连接层，输入大小 4096，输出结果大小 4096
11. 全连接层，输入大小 4096，输出结果 1000

示意图如下图所示：

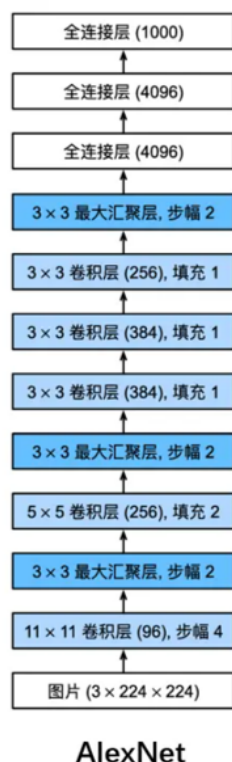


图 6: AlexNet 结构

我们还可以运用卷积尺寸计算公式来检查 AlexNet 各个环节的尺寸设计是否有误，计算公式如下：

$$OutputSize = \lfloor (n_h - k_h + p_h + s_h) / s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w) / s_w \rfloor \quad (1)$$

其中， $n$  为输入的长宽， $k$  为卷积核大小， $p$  为横向/纵向总的 padding 行/列数， $s$  为步长

## 2.4 损失函数和优化器初始化

本实验中选择了 `nn.CrossEntropyLoss` 和 `torch.optim.SGD` 作为损失函数和优化器。

## 2.5 基于训练集训练 AlexNet

根据上述神经网络结构的构建，我们可以很容易的对 AlexNet 进行训练，前向传播、反向传播以及优化的语句如下：

```
1  # 前向传播
2  outputs = model(images)
3  loss = criterion(outputs, labels)
4
5  # 反向传播和优化
6  optimizer.zero_grad()
7  loss.backward()
8  optimizer.step()
```

## 3 任务 2: 使用 AlexNet 来对 Caltech101 数据集进行分类任务

使用训练好的 AlexNet 测试的思路为：将测试集数据投入已经训练好的模型，得到训练结果并与标签对比，计算准确率。

## 4 任务 3: 通过 tensorboard 可视化网络的训练过程（loss 函数的变化）

### 4.1 tensorboard 的使用

```
1  from torch.utils.tensorboard import SummaryWriter
2  writer = SummaryWriter('runs/origin_no_drop')
3  # 日志目录
4  writer.add_scalar('Loss/train', loss.item(), step)
```

### 4.2 实验结果分析

本次实验采用了 AlexNet 网络对 Caltech101 数据集进行训练和测试，并且加入了两种模型优化方法：Data Augmentation 和 Dropout 正则化。

其中，Data Augmentation 采用了两种图像变换方法：随机水平翻转和随机旋转  $[-20, 20]$  度，代码如下：

```
1 transforms.RandomHorizontalFlip(), # 随机水平翻转
2 transforms.RandomRotation(20), # 随机旋转  $[-20, 20]$  度
```

Dropout 正则化在线性全连接层使用，默认丢弃比例为 50%。模型训练的 batch 大小为 64，epoch 数量为 20，得到验证集准确率、训练集损失、验证集损失如下所示：

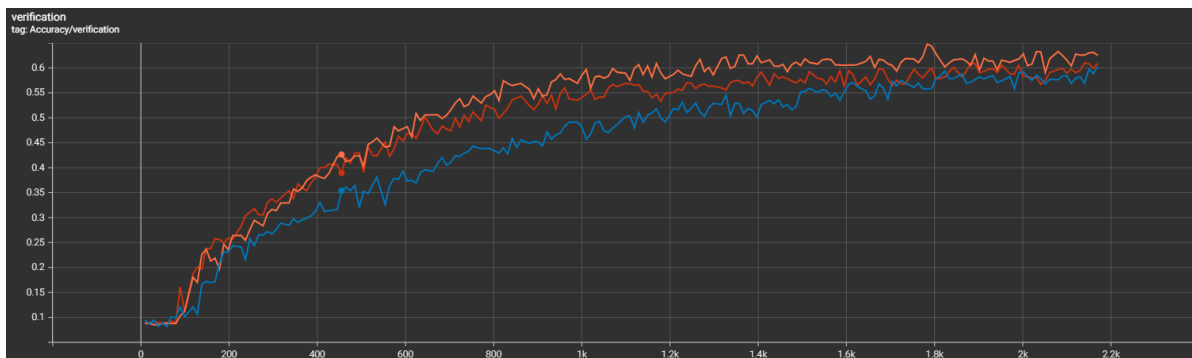


图 7: 验证集准确率

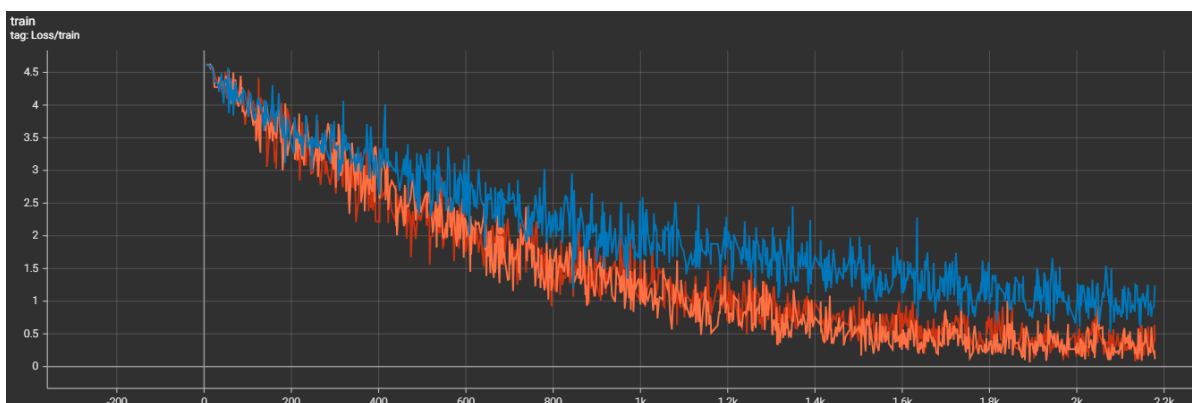


图 8: 训练集损失

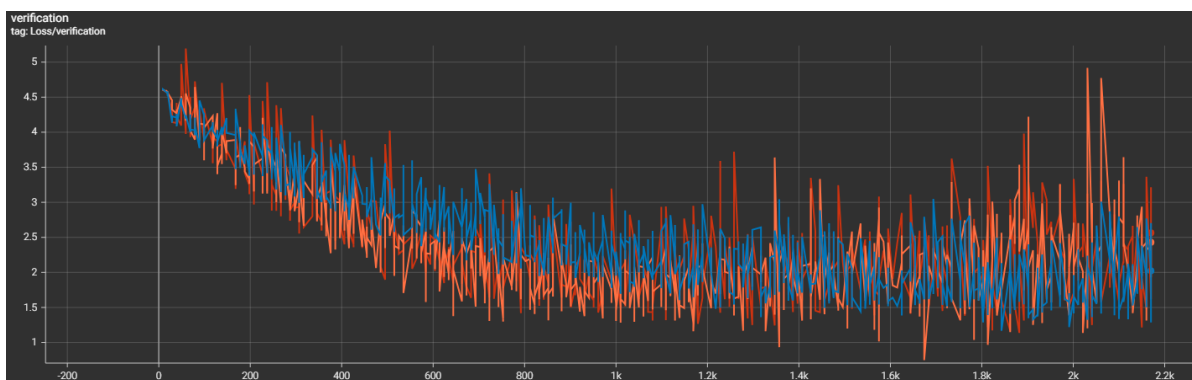


图 9: 验证集损失

其中，红色曲线是 AlexNet 原始曲线，橙色曲线是加入 Dropout 正则化的曲线，蓝色曲线是加入数据增强方法的曲线。从曲线结果来看，加入 dropout 正则化的模型准确率较高，而加入数据增强方法的模型验证集损失震荡较小。

Dropout 通过随机丢弃部分神经元来减少模型的过拟合。这有助于模型学到更加稳健的特征，提升了模型的最终准确率。因为它抑制了某些神经元的过度适应，模型在训练过程中可能变得更加通用，从而在测试集上表现得更好。

数据增强（随机水平翻转和旋转）增加了训练集的多样性，帮助模型应对不同的输入变换，特别是在验证集上表现得更加稳健。这可以解释为什么验证集损失较小并且震荡减小，因为模型通过看到更多不同的图像变种，能够更好地拟合验证数据，而不会过度依赖某些特定模式。

综上所述，如果要求模型的准确率较高，可以进行适当比例的 dropout，而如果希望模型有更好的适应性而不过度依赖某种特定模式则可以考虑加入数据增强方法。另外，我认为如果对数据增强组别增加 epoch 数，最终的准确率可能会继续上升一点，但是由于硬件限制，我的模型单次训练时间较长，所以没有尝试。

## A AlexNet 模型架构

```
1  def __init__(self, num_classes=102):  # Caltech101 有 102 个类
2  super(AlexNet, self).__init__()
3  self.features = nn.Sequential(  # (3,224,224)
4      nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=2),  # Conv
5          ↪ Layer 1 (96,54,54)
6      nn.ReLU(inplace=True),
7      nn.MaxPool2d(kernel_size=3, stride=2),  # Pooling Layer 1
8          ↪ (96,26,26)
9      nn.Conv2d(96, 256, kernel_size=5, padding=2),  # Conv Layer 2
10         ↪ (256,26,26)
11     nn.ReLU(inplace=True),
12     nn.MaxPool2d(kernel_size=3, stride=2),  # Pooling Layer 2
13         ↪ (256,12,12)
14     nn.Conv2d(256, 384, kernel_size=3, padding=1),  # Conv Layer 3
15         ↪ (384,12,12)
16     nn.ReLU(inplace=True),
17     nn.Conv2d(384, 384, kernel_size=3, padding=1),  # Conv Layer 4
18         ↪ (384,12,12)
19     nn.ReLU(inplace=True),
20     nn.Conv2d(384, 256, kernel_size=3, padding=1),  # Conv Layer 5
21         ↪ (256,12,12)
22     nn.ReLU(inplace=True),
23     nn.MaxPool2d(kernel_size=3, stride=2)  # Pooling Layer 3
24         ↪ (256,5,5)
25 )
26
27 self.classifier = nn.Sequential(
28     nn.Dropout(),
29     nn.Linear(256 * 6 * 6, 4096),  # 256 * 6 * 6 是池化后的特征图尺寸
30     nn.ReLU(inplace=True),
31     nn.Dropout(),
32     nn.Linear(4096, 4096),
33     nn.ReLU(inplace=True),
34     nn.Linear(4096, num_classes),
35 )
```