数字媒体处理技术实验报告——实验（三）

21R0361班 2021113117-王宇轩

目录

1. 实验环境

- 操作系统：Windows
- 编程语言：Python
- 所用软件：PyCharm
- 主要库调用：

名称	介绍
PyTorch	本次实验所选用的开源机器学习库
Torchvision	PyTorch的拓展库，本次实验选用的计算机视觉库
Tensorboard	训练结果可视化

2. 文件列表

文件名	内容
main.py	源程序
实验报告.pdf	实验报告

文件名	内容
验收视频.mp4	模型进行训练并输出评估结果的录屏，用于验收

3. 实验内容

3.1 数据集的处理

本次实验选用Caltech-101数据集，共有（除背景外）101个类别，各个类别数据量不一致，大多数类别都有约50张图像，最多的数据量可达800张，可于 http://www.vision.caltech.edu/Image_Datasets/Caltech101/下载。

主要使用 `torchvision.datasets` 的 `ImagFolder` 将数据集的文件夹名作为类别名来加载数据集。**需要将BACKGROUND_Google文件夹移除，否则因为加载了102个类别，模型会报错。**

这部分代码主要是进行数据集的加载和处理，详细的操作请参考代码中的注释。这里对图像进行了Resize至224*224。至于特征提取的工作则是在模型网络中完成的，因为采用了卷积神经网络。

```
# 数据预处理
transform = transforms.Compose([
    transforms.Resize((224, 224)),              # 调整图像大小
    transforms.ToTensor(),                      # 转换为张量
])

# 加载 Caltech101 数据集
dataset = ImageFolder(root='./caltech-101/101_ObjectCategories', transform=transform)
# 计算数据集大小
dataset_size = len(dataset)
# 计算划分的样本数量
train_size = int(0.8 * dataset_size)
val_size = int(0.1 * dataset_size)
test_size = dataset_size - train_size - val_size
# 随机划分训练集、验证集和测试集
train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size, val_size,
test_size])
# 创建数据加载器
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=batch_size,
shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size,
shuffle=True)
```

3.2 模型网络定义

这里的模型参考了AlexNet的网络结构，使用了8层卷积神经网络。

```
net = nn.Sequential(
    # 这里使用一个11*11的更大窗口来捕捉对象。
    # 同时，步幅为4，以减少输出的高度和宽度。
    # 另外，输出通道的数目远大于LeNet
    nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=1), nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    # 减小卷积窗口，使用填充为2来使得输入与输出的高和宽一致，且增大输出通道数
    nn.Conv2d(96, 256, kernel_size=5, padding=2), nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    # 使用三个连续的卷积层和较小的卷积窗口。
    # 除了最后的卷积层，输出通道的数量进一步增加。
    # 在前两个卷积层之后，汇聚层不用于减少输入的高度和宽度
    nn.Conv2d(256, 384, kernel_size=3, padding=1), nn.ReLU(),
    nn.Conv2d(384, 384, kernel_size=3, padding=1), nn.ReLU(),
    nn.Conv2d(384, 256, kernel_size=3, padding=1), nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Flatten(),
    # 这里，全连接层的输出数量是LeNet中的好几倍。使用dropout层来减轻过拟合
    nn.Linear(6400, 4096), nn.ReLU(),
    nn.Dropout(p=0.7),
    nn.Linear(4096, 4096), nn.ReLU(),
    nn.Dropout(p=0.7),
    # 最后是输出层。由于这里使用caltech-101，所以类别数为101
    nn.Linear(4096, 101))
```

3.3 模型训练

```
# 设置设备（使用 GPU 如果可用）
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
net = net.to(device)
print('training on', device)
# 定义优化器和损失函数
optimizer = torch.optim.SGD(net.parameters(), lr=lr)
loss = nn.CrossEntropyLoss()


# 初始化权重
def init_weights(m):
    if type(m) == nn.Linear or type(m) == nn.Conv2d:
        nn.init.xavier_uniform_(m.weight)


net.apply(init_weights)

# 设置训练模式
net.train()
# 迭代训练
total_loss = 0
total_step = len(train_loader)   # 确定步数
for epoch in range(num_epochs):  # 迭代
```

```
    for i, (images, labels) in enumerate(train_loader):
        # 将输入数据和标签加载到设备上（如GPU）
        images = images.to(device)
        labels = labels.to(device)

        outputs = net(images)
        lo = loss(outputs, labels)
        optimizer.zero_grad()
        lo.backward()
        optimizer.step()
        # 损失函数值求和
        total_loss += lo
        print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'.format(epoch + 1, num_epochs,
i + 1, total_step, lo.item()))

        # writer.add_scalar('Train Loss', lo.item(), epoch)
    # 计算平均权重并写入数据
    avr_loss = total_loss / total_step
    writer.add_scalar('Train Loss', avr_loss, epoch)
    total_loss = 0
```

Tensorboard输出的日志文件存于`./log`目录，调用命令`tensorboard --logdir ./log --host 127.0.0.1 --port 8008`即可生成可视化结果。

3.4 模型评估

```
# 评估模型
net.eval()
# 计算在训练集、验证集和测试集上测试精度
with torch.no_grad():
    correct_train = 0
    total_train = 0
    correct_val = 0
    total_val = 0
    correct_test = 0
    total_test = 0
    # 训练集
    for images_train, labels_train in train_loader:
        images_train = images_train.to(device)
        labels_train = labels_train.to(device)
        outputs_train = net(images_train)
        _, predicted_train = torch.max(outputs_train.data, 1)
        total_train += labels_train.size(0)
        correct_train += (predicted_train == labels_train).sum().item()
        train_acc = correct_train / total_train
    # 验证集
    for images_val, labels_val in val_loader:
        images_val = images_val.to(device)
        labels_val = labels_val.to(device)
        outputs_val = net(images_val)
        _, predicted_val = torch.max(outputs_val.data, 1)
```

```
        total_val += labels_val.size(0)
        correct_val += (predicted_val == labels_val).sum().item()
        val_acc = correct_val / total_val
    # 测试集
    for images_test, labels_test in test_loader:
        images_test = images_test.to(device)
        labels_test = labels_test.to(device)
        outputs_test = net(images_test)
        _, predicted_test = torch.max(outputs_test.data, 1)
        total_test += labels_test.size(0)
        correct_test += (predicted_test == labels_test).sum().item()
        test_acc = correct_test / total_test
    # 输出精度结果
    print('训练集测试精度：{.4f}'.format(train_acc))
    print('验证集测试精度：{.4f}'.format(train_acc))
    print('测试集测试精度：{.4f}'.format(train_acc))
```

3.5 调试超参数

模型涉及到的超参数如下：

```
# 超参数
batch_size = 64
lr, num_epochs = 0.05, 35
```

此外，模型`net`中`Dropout`中的参数`p`也是需要调节的参数，以上给出的参数是已知效果最佳的，具体的调参过程于**4. 实验结果与分析**给出。

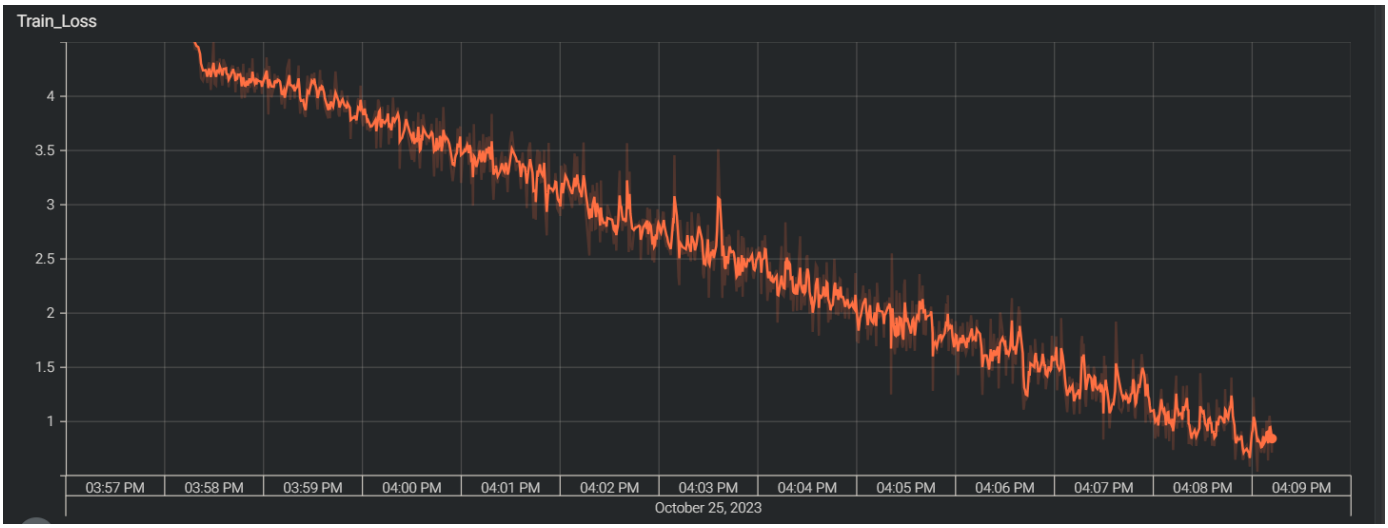
3.6 测试结果与调试分析

最初，超参数选取如下：

```
lr, num_epochs = 0.01, 10
```

并取`batch_size = 128`结果是，模型收敛效果较差。

将`num_epochs`调至25，损失函数图像和训练集、验证集测试精度如下：



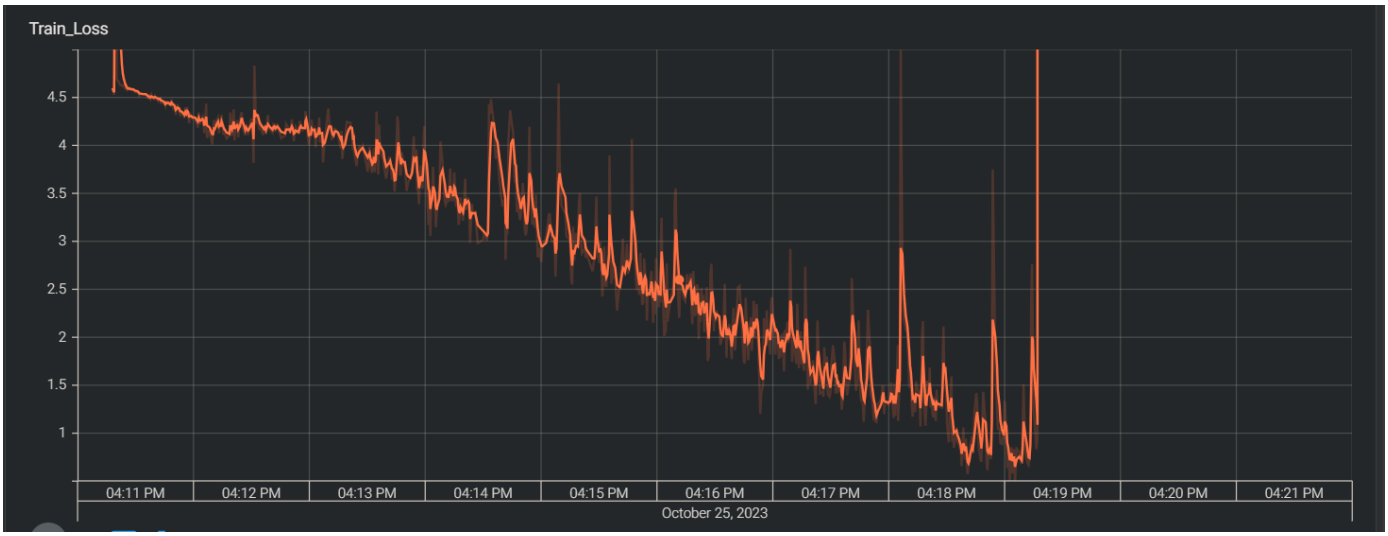

```
训练集测试精度：0.7742400230514335
验证集测试精度：0.46490218642117376
```

可见模型收敛效果提升明显，精度结果有所提升。

在此基础上尝试调大学习率，希望可以加快收敛速度，取：

```
batch_size = 128
lr, num_epochs = 0.1, 25
```

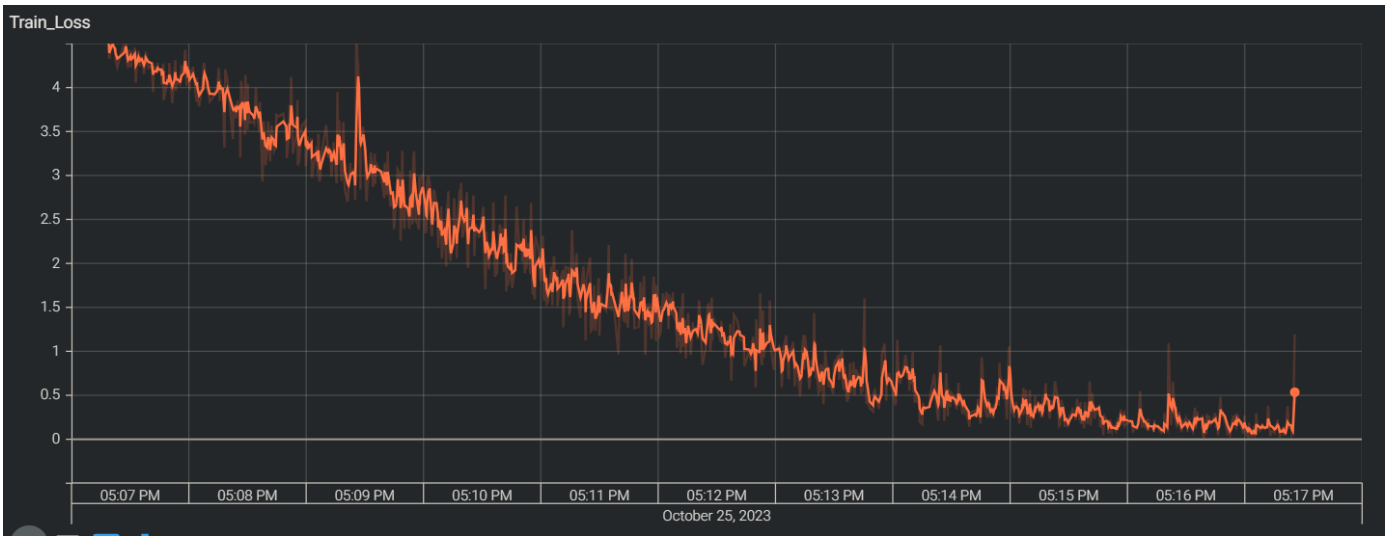
产生了如下的损失函数曲线，此结果表明学习率过大，模型并未收敛。



减小学习率，并且取更小的batch_size：

```
batch_size = 64
lr, num_epochs = 0.05, 25
```

结果如下：

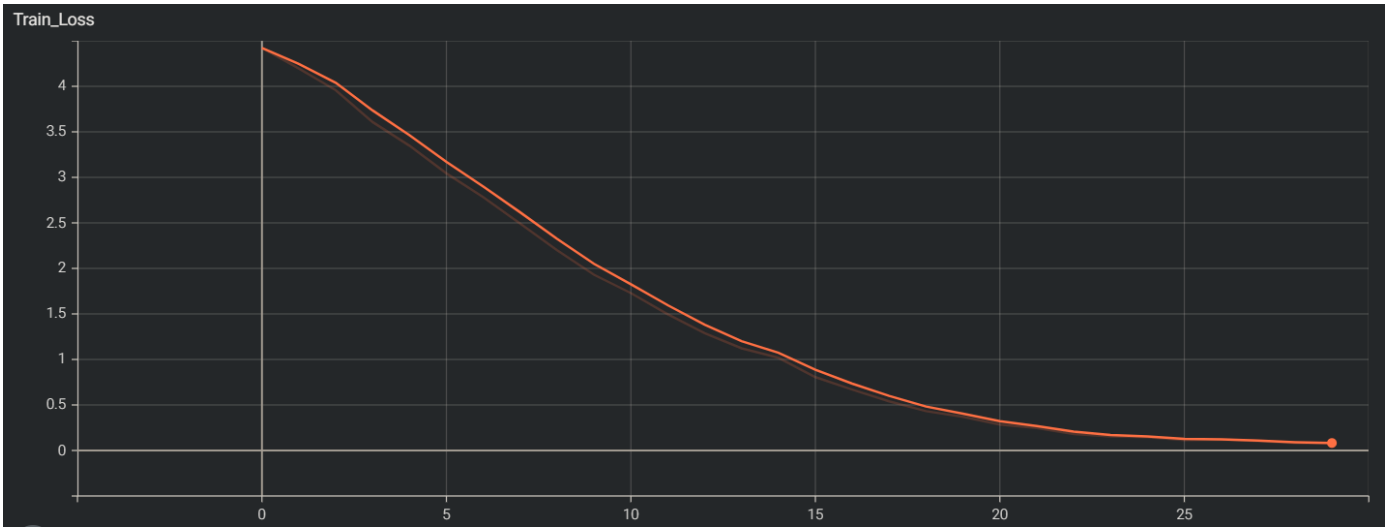



```
训练集测试精度：0.94215865421012101
验证集测试精度：0.61542310582001210
```

从损失函数曲线可看出，模型已充分收敛，但精度还有提升空间。增加迭代次数：

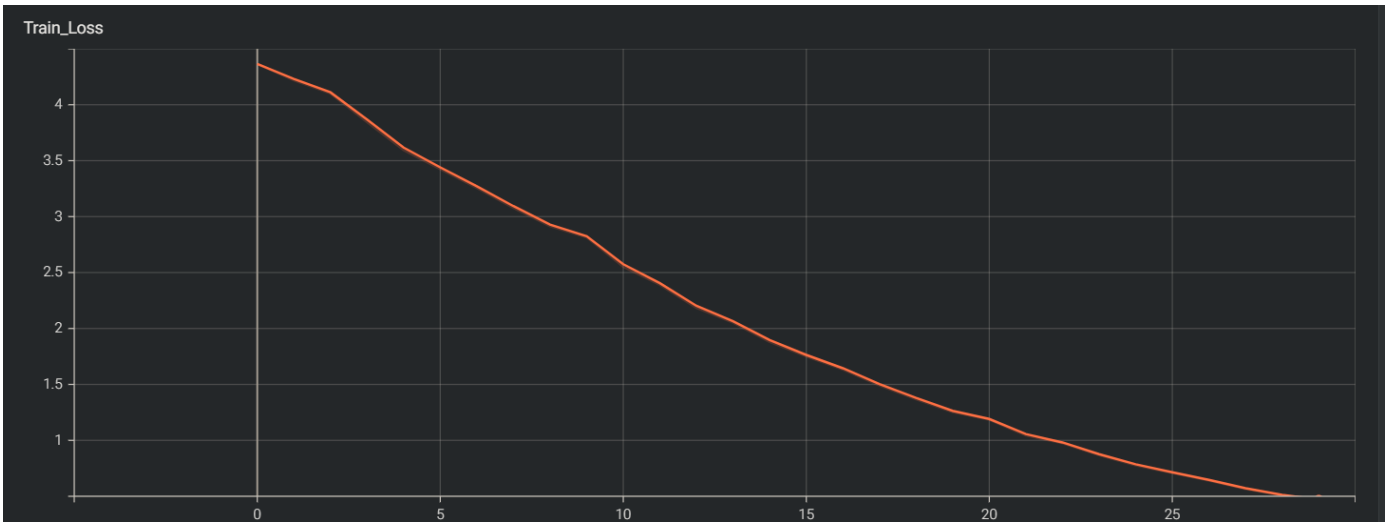
```
batch_size = 64
lr, num_epochs = 0.05, 30
```

结果如下：




```
训练集测试精度：0.9987
验证集测试精度：0.6563
测试集测试精度：0.6743
```

此时封闭测试精度已经很好，但是过拟合的问题需要解决。尝试调大`Dropout`参数`p=0.7`，结果如下：

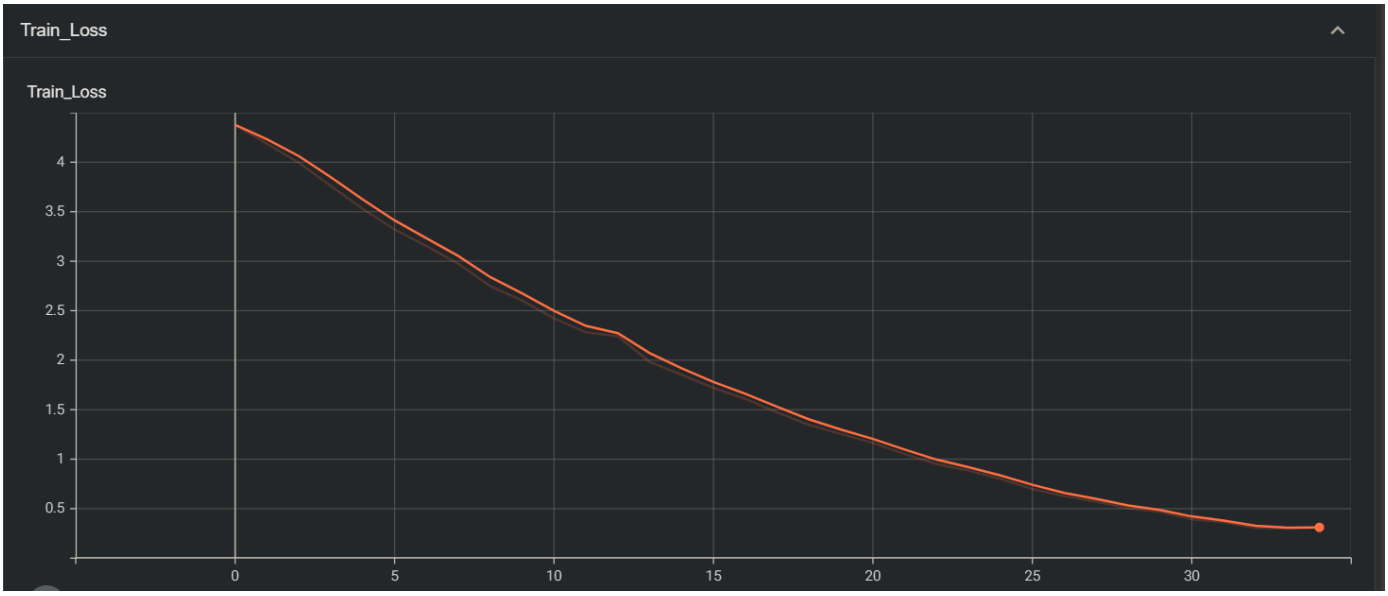



```
训练集测试精度：0.9166
验证集测试精度：0.6759
测试集测试精度：0.6697
```

过拟合现象有所好转，但精度有所下降，且模型并未充分收敛。增加迭代次数：

```
batch_size = 64
lr, num_epochs = 0.05, 35
```

结果如下：




```
训练集测试精度：0.9872
验证集测试精度：0.6736
测试集测试精度：0.7077
```

可见，模型收敛得较为充分，过拟合现象也有所减轻。以上就是已获得的最好结果。

4. 实验总结

熟悉、掌握了一些开源的机器学习平台，加深了对数据处理、特征提取的了解。