哈尔滨工业大学

模式识别与深度学习

实验二：卷积神经网络实现

姓　　名：	**刘天瑞**
院　（系）：	**未来技术学院**
专　　业：	**视听觉信息处理**
学　　号：	**7203610121**
指导教师：	**左旺孟**
提交日期：	**2023.5.9**

摘 要

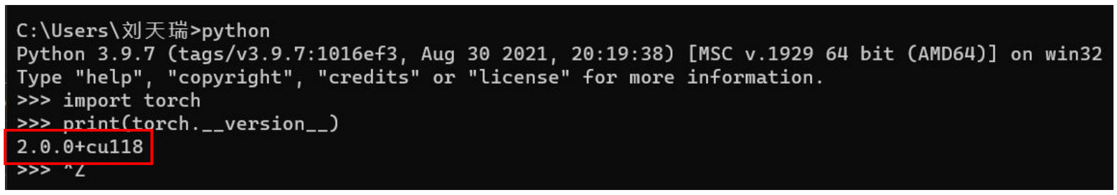
本次深度学习实验为模式识别与深度学习课程的实验二：即卷积神经网络的实现，实验的主要任务是基于 Python 的 PyTorch 库工具来实现 AlexNet 卷积神经网络结构，并且在 CalTech 101 数据集上进行模型验证。在本次实验中，我复现并且改进了 AlexNet 的卷积神经网络结构，同时还使用 Tensorboard 图形显示工具进行训练得到的数据可视化，最终在数据集上达到了 78.7%一个较为不错的准确率。

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

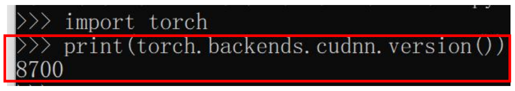
目 录

一、 深度学习框架与实验环境

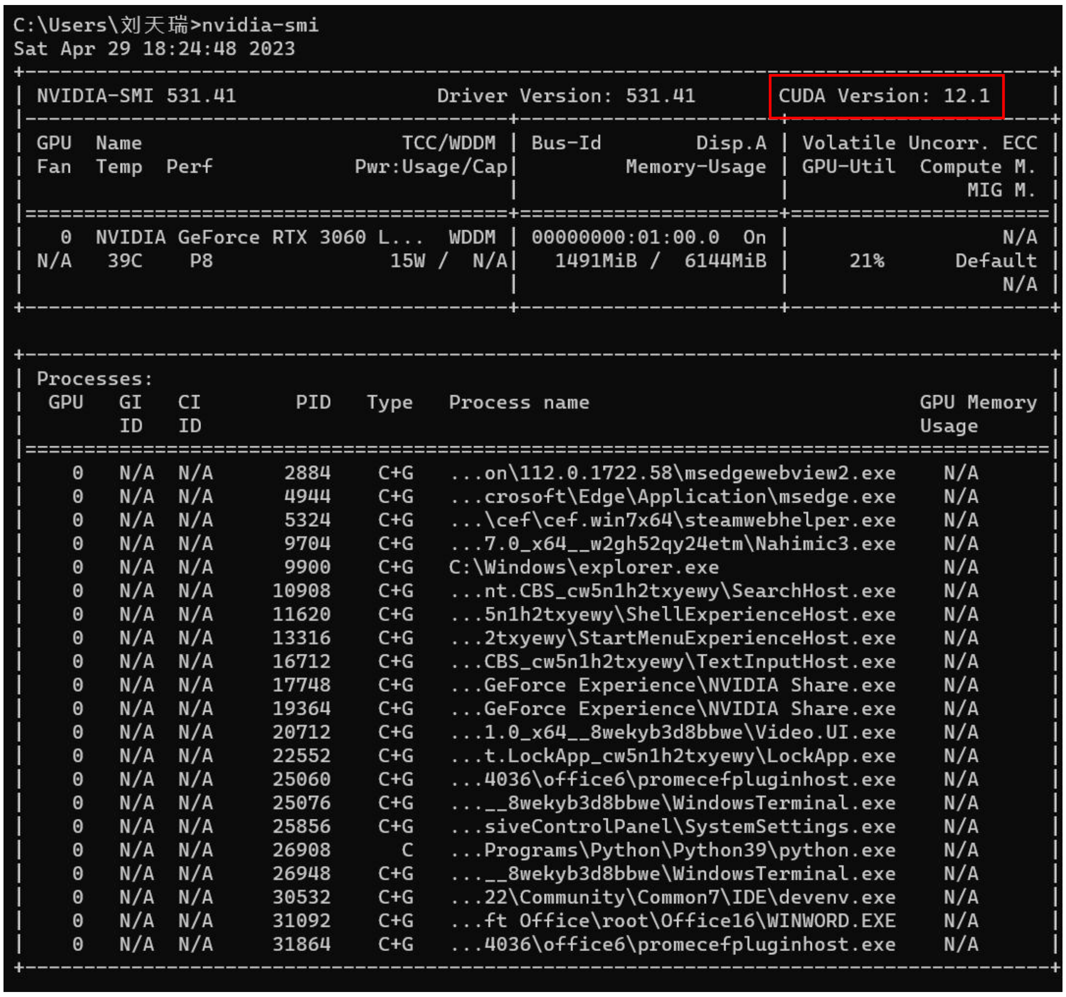
本次实验采用的深度学习框架是 Python 中强大的深度学习库 Pytorch，整个实验是在 Visual Studio + Pip 环境下完成的。Pip 是一个开源的 Python 发行版本，可以很方便地安装许多深度学习所需要的模块包，而 Visual Studio 则是一个功能强大的 IDE，可以在其中完成 python 代码编写深度学习过程、进行训练测试等深度学习环节。由于我在大二时参加过美赛有接触到一些深度学习相关的工具知识，因此在本次实验中的配置环境过程相对得心应手。配置环境的具体流程比较繁琐，查看系统 Pytorch 库以及 cuda 版本如下图 1 所示（英伟达表示 cuda 可升级到的最高版本）：



```
C:\Users\刘天瑞>python
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> print(torch.__version__)
2.0.0+cu118
>>> ^Z
```



```
>>> import torch
>>> print(torch.backends.cudnn.version())
8700
```



```
C:\Users\刘天瑞>nvidia-smi
Sat Apr 29 18:24:48 2023
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 531.41                 Driver Version: 531.41       CUDA Version: 12.1     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                      TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf            Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA GeForce RTX 3060 L...  WDDM | 00000000:01:00.0  On |                  N/A |
| N/A   39C    P8               15W /  N/A|   1491MiB /  6144MiB |     21%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|    0   N/A  N/A      2884    C+G   ...on\112.0.1722.58\msedgewebview2.exe    N/A      |
|    0   N/A  N/A      4944    C+G   ...crosoft\Edge\Application\msedge.exe    N/A      |
|    0   N/A  N/A      5324    C+G   ...\cef\cef.win7x64\steamwebhelper.exe    N/A      |
|    0   N/A  N/A      9704    C+G   ...7.0_x64__w2gh52qy24etm\Nahimic3.exe    N/A      |
|    0   N/A  N/A      9900    C+G   C:\Windows\explorer.exe                   N/A      |
|    0   N/A  N/A     10908    C+G   ...nt.CBS_cw5n1h2txyewy\SearchHost.exe    N/A      |
|    0   N/A  N/A     11620    C+G   ...5n1h2txyewy\ShellExperienceHost.exe    N/A      |
|    0   N/A  N/A     13316    C+G   ...2txyewy\StartMenuExperienceHost.exe    N/A      |
|    0   N/A  N/A     16712    C+G   ...CBS_cw5n1h2txyewy\TextInputHost.exe    N/A      |
|    0   N/A  N/A     17748    C+G   ...GeForce Experience\NVIDIA Share.exe    N/A      |
|    0   N/A  N/A     19364    C+G   ...GeForce Experience\NVIDIA Share.exe    N/A      |
|    0   N/A  N/A     20712    C+G   ...1.0_x64__8wekyb3d8bbwe\Video.UI.exe    N/A      |
|    0   N/A  N/A     22552    C+G   ...t.LockApp_cw5n1h2txyewy\LockApp.exe    N/A      |
|    0   N/A  N/A     25060    C+G   ...4036\office6\promecefpluginhost.exe    N/A      |
|    0   N/A  N/A     25076    C+G   ...__8wekyb3d8bbwe\WindowsTerminal.exe    N/A      |
|    0   N/A  N/A     25856    C+G   ...siveControlPanel\SystemSettings.exe    N/A      |
|    0   N/A  N/A     26908      C   ...Programs\Python\Python39\python.exe    N/A      |
|    0   N/A  N/A     26948    C+G   ...__8wekyb3d8bbwe\WindowsTerminal.exe    N/A      |
|    0   N/A  N/A     30532    C+G   ...22\Community\Common7\IDE\devenv.exe    N/A      |
|    0   N/A  N/A     31092    C+G   ...ft Office\root\Office16\WINWORD.EXE    N/A      |
|    0   N/A  N/A     31864    C+G   ...4036\office6\promecefpluginhost.exe    N/A      |
+---------------------------------------------------------------------------------------+
```

图 1

二、 主要研究内容以及实验背景知识

2.1 AlexNet 网络模型简介

Alexnet 网络模型是由 5 个卷积层，3 个池化层以及 3 个全连接层所构成的。同时 AlexNet 是 LeNet 神经网络的进一步发展，其中包含了许多新的亮点如下所示：

1. AlexNet 网络模型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. AlexNet 网络模型采用了 ReLU( )激活函数来代替 Sigmoid( )函数，从而提升了训练速度，并且在一定规模数据上的性能表现超过了使用 Sigmoid( )函数的网络模型；

3. AlexNet 网络模型引入了 Dropout 参数用于解决模型训练过程中容易出现过拟合的问题。

AlexNet 模型的网络结构如下图 2 所示：

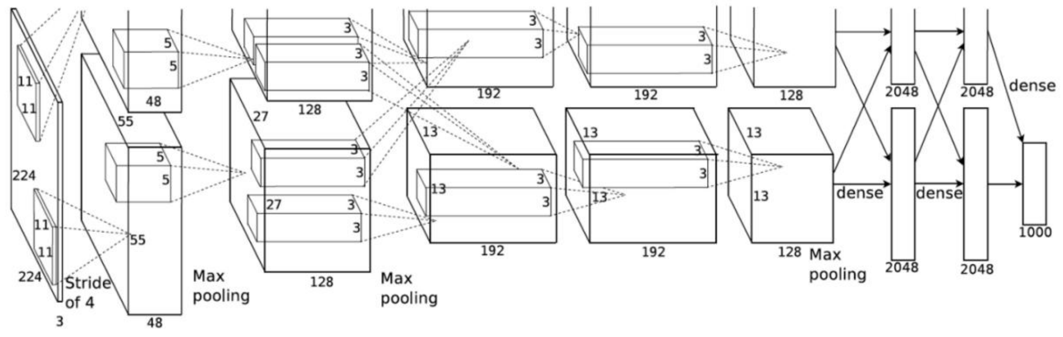


图 2

2.2 CalTech 101 数据集介绍

Caltech 101 数据集是图像识别分类领域常用的数据集。其总共包含有 9146 张图像，这些图像被分为 101 个不同类别（例如 face, piano, ant 等等）和背景类别。而与图像一起被提供的还有一组注释，其用来描述每个图像的准确轮廓。 在本次深度学习实验中，我按照实验的具体要求忽略去掉了 Caltech 101 数据集中的 文件夹内容：BACKGROUND_Google 类，再用剩下的数据来完成接下来的

实验。

三、实验详细过程

3.1　加载构建并读取数据集

由于 Caltech 101 数据集不能够通过 Pytorch 直接进行导入，所以我需要自己编写数据集类，以便在训练数据时使用。在本次实验中，数据集类我将其保存在了 caltech101.py 程序代码中。完成的主要工作是对__init__，__getitem__以及__len__ 三个方法的重写。值得注意的是，在__getitem__方法中，需要利用到 PIL 库来读入图片，并且将图片转换 RGB 格式，从而方便后续进一步处理。在完成上述步骤以后，便可以在主函数 main( )中调用该数据集了。

3.2　搭建 AlexNet 网络结构

AlexNet 网络模型的原始论文中采用了两块 GPU 进行运算后再拼接向量(如上图 2 所示)，而由于设备所限，我只实现了其中一块 GPU 对应的网络结构。

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

features 部分的网络结构如下所示：

• 卷积层 1：

输入：224 × 224 × 3 的图像；卷积核数量：48（对应单片的 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（最大池化处理）。max pooling 的参数为 kernal_size = 3, stride = 2, padding = 0；

输出：55 ×55 × 48 的 feature。

• 卷积层 2：

输入：55 ×55 × 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 处理（max pooling 的参数同之前一样）；

输出：27 ×27 ×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，但此时不进行 max pooling 处理；

输出：13 ×13 ×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，但此时不进行 max pooling 处理；

输出：13 ×13 ×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 处理（max pooling 参数同之前一样）；

输出：27 ×27 ×128 的 feature。

完成上述步骤后，我们可以得到大小为 13 ×13 ×128 的特征，并且将其通过 flatten 方法展平后输入至 classifier 部分，该部分的网络结构如下所示：

• 全连接层 1：

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

输出：仍为长度为 2048 的一维向量。

• 全连接层 3：

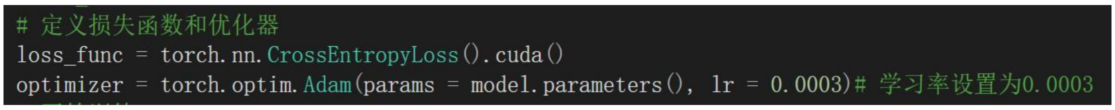
输入：上一层的输出；

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

3.3 定义并且设置模型参数

3.3.1 损失函数与优化器

为了取得相对较好的训练效果，我继续采用了交叉熵损失与 Adam 优化器。其具体参数如下图 3 所示：

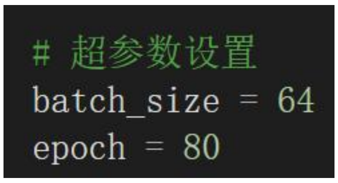


```
# 定义损失函数和优化器
loss_func = torch.nn.CrossEntropyLoss().cuda()
optimizer = torch.optim.Adam(params = model.parameters(), lr = 0.0003)# 学习率设置为0.0003
```

图 3

3.3.2 超参数

超参数的设定值如下图 4 所示：



```
# 超参数设置
batch_size = 64
epoch = 80
```

图 4

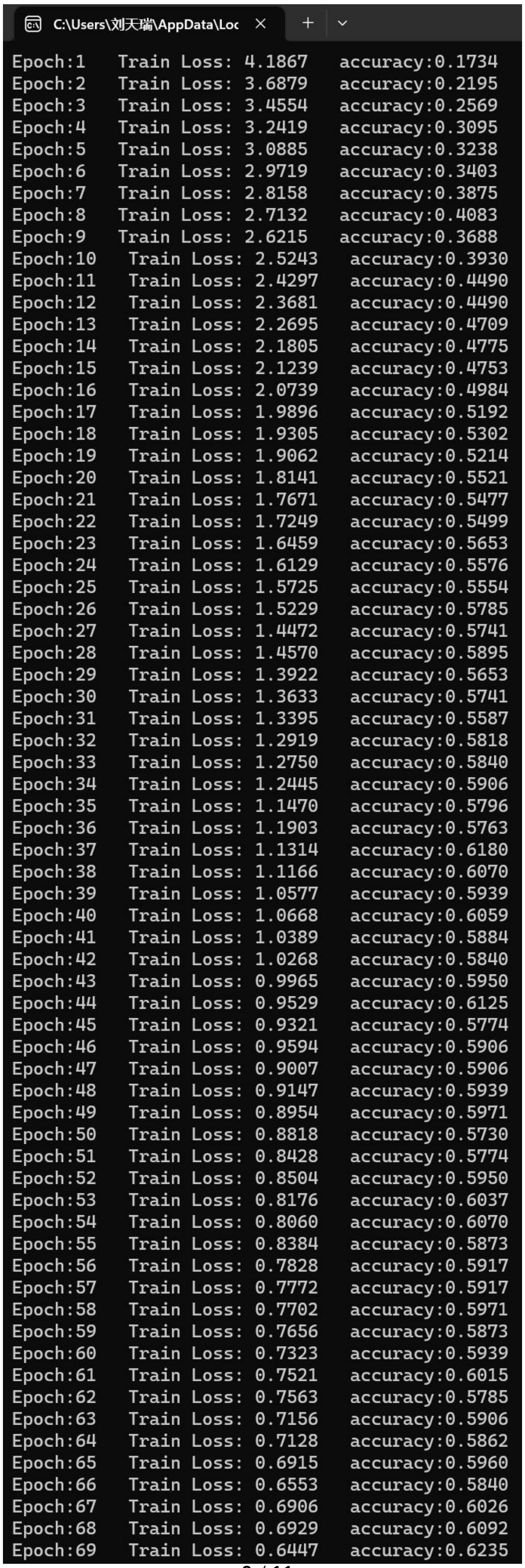
3.4 引入 Tensorboard 图形显示工具

通过 Pip 命令行指令来安装 Tensorboard。在训练相关代码执行前，需要实例化一个 SummaryWriter 来保存相关信息。在训练过程中，需要将网络结构，本轮训练损失以及本轮训练的准确率等信息加入到 SummaryWriter 实例中，在训练结束后即可查询。

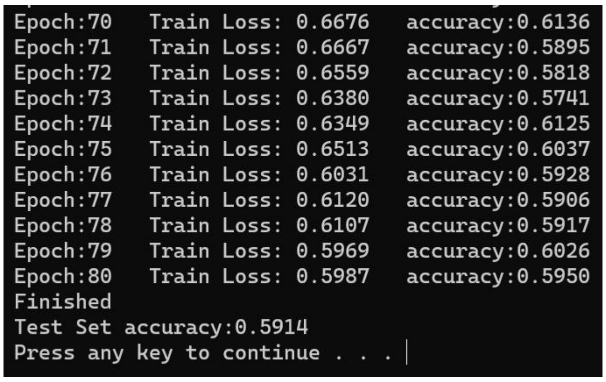
在本次实验中，我还将损失函数值,精确度值以及网络结构这三个数据写入到了 SummaryWriter 中，在训练结束之后就可以在 Tensorboard 线上网页上查看到。

四、实验结果与分析

在我这台计算机的 GPU 为 Nvidia GeForce RTX 3060 的环境下，训练 1 轮 epoch 大致需要 10s。而训练完 80 轮 epoch 后，AlexNet 网络模型在该数据集上取得了 78.7%的准确率，得到的实验命令行运行结果如下图 5 所示：



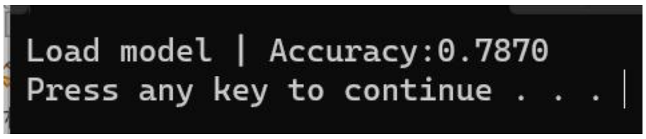
```
Epoch:1   Train Loss: 4.1867   accuracy:0.1734
Epoch:2   Train Loss: 3.6879   accuracy:0.2195
Epoch:3   Train Loss: 3.4554   accuracy:0.2569
Epoch:4   Train Loss: 3.2419   accuracy:0.3095
Epoch:5   Train Loss: 3.0885   accuracy:0.3238
Epoch:6   Train Loss: 2.9719   accuracy:0.3403
Epoch:7   Train Loss: 2.8158   accuracy:0.3875
Epoch:8   Train Loss: 2.7132   accuracy:0.4083
Epoch:9   Train Loss: 2.6215   accuracy:0.3688
Epoch:10   Train Loss: 2.5243   accuracy:0.3930
Epoch:11   Train Loss: 2.4297   accuracy:0.4490
Epoch:12   Train Loss: 2.3681   accuracy:0.4490
Epoch:13   Train Loss: 2.2695   accuracy:0.4709
Epoch:14   Train Loss: 2.1805   accuracy:0.4775
Epoch:15   Train Loss: 2.1239   accuracy:0.4753
Epoch:16   Train Loss: 2.0739   accuracy:0.4984
Epoch:17   Train Loss: 1.9896   accuracy:0.5192
Epoch:18   Train Loss: 1.9305   accuracy:0.5302
Epoch:19   Train Loss: 1.9062   accuracy:0.5214
Epoch:20   Train Loss: 1.8141   accuracy:0.5521
Epoch:21   Train Loss: 1.7671   accuracy:0.5477
Epoch:22   Train Loss: 1.7249   accuracy:0.5499
Epoch:23   Train Loss: 1.6459   accuracy:0.5653
Epoch:24   Train Loss: 1.6129   accuracy:0.5576
Epoch:25   Train Loss: 1.5725   accuracy:0.5554
Epoch:26   Train Loss: 1.5229   accuracy:0.5785
Epoch:27   Train Loss: 1.4472   accuracy:0.5741
Epoch:28   Train Loss: 1.4570   accuracy:0.5895
Epoch:29   Train Loss: 1.3922   accuracy:0.5653
Epoch:30   Train Loss: 1.3633   accuracy:0.5741
Epoch:31   Train Loss: 1.3395   accuracy:0.5587
Epoch:32   Train Loss: 1.2919   accuracy:0.5818
Epoch:33   Train Loss: 1.2750   accuracy:0.5840
Epoch:34   Train Loss: 1.2445   accuracy:0.5906
Epoch:35   Train Loss: 1.1470   accuracy:0.5796
Epoch:36   Train Loss: 1.1903   accuracy:0.5763
Epoch:37   Train Loss: 1.1314   accuracy:0.6180
Epoch:38   Train Loss: 1.1166   accuracy:0.6070
Epoch:39   Train Loss: 1.0577   accuracy:0.5939
Epoch:40   Train Loss: 1.0668   accuracy:0.6059
Epoch:41   Train Loss: 1.0389   accuracy:0.5884
Epoch:42   Train Loss: 1.0268   accuracy:0.5840
Epoch:43   Train Loss: 0.9965   accuracy:0.5950
Epoch:44   Train Loss: 0.9529   accuracy:0.6125
Epoch:45   Train Loss: 0.9321   accuracy:0.5774
Epoch:46   Train Loss: 0.9594   accuracy:0.5906
Epoch:47   Train Loss: 0.9007   accuracy:0.5906
Epoch:48   Train Loss: 0.9147   accuracy:0.5939
Epoch:49   Train Loss: 0.8954   accuracy:0.5971
Epoch:50   Train Loss: 0.8818   accuracy:0.5730
Epoch:51   Train Loss: 0.8428   accuracy:0.5774
Epoch:52   Train Loss: 0.8504   accuracy:0.5950
Epoch:53   Train Loss: 0.8176   accuracy:0.6037
Epoch:54   Train Loss: 0.8060   accuracy:0.6070
Epoch:55   Train Loss: 0.8384   accuracy:0.5873
Epoch:56   Train Loss: 0.7828   accuracy:0.5917
Epoch:57   Train Loss: 0.7772   accuracy:0.5917
Epoch:58   Train Loss: 0.7702   accuracy:0.5971
Epoch:59   Train Loss: 0.7656   accuracy:0.5873
Epoch:60   Train Loss: 0.7323   accuracy:0.5939
Epoch:61   Train Loss: 0.7521   accuracy:0.6015
Epoch:62   Train Loss: 0.7563   accuracy:0.5785
Epoch:63   Train Loss: 0.7156   accuracy:0.5906
Epoch:64   Train Loss: 0.7128   accuracy:0.5862
Epoch:65   Train Loss: 0.6915   accuracy:0.5960
Epoch:66   Train Loss: 0.6553   accuracy:0.5840
Epoch:67   Train Loss: 0.6906   accuracy:0.6026
Epoch:68   Train Loss: 0.6929   accuracy:0.6092
Epoch:69   Train Loss: 0.6447   accuracy:0.6235
```



```
Epoch:70   Train Loss: 0.6676   accuracy:0.6136
Epoch:71   Train Loss: 0.6667   accuracy:0.5895
Epoch:72   Train Loss: 0.6559   accuracy:0.5818
Epoch:73   Train Loss: 0.6380   accuracy:0.5741
Epoch:74   Train Loss: 0.6349   accuracy:0.6125
Epoch:75   Train Loss: 0.6513   accuracy:0.6037
Epoch:76   Train Loss: 0.6031   accuracy:0.5928
Epoch:77   Train Loss: 0.6120   accuracy:0.5906
Epoch:78   Train Loss: 0.6107   accuracy:0.5917
Epoch:79   Train Loss: 0.5969   accuracy:0.6026
Epoch:80   Train Loss: 0.5987   accuracy:0.5950
Finished
Test Set accuracy:0.5914
Press any key to continue . . .
```

图 5

模型在测试集上的表现如下图 6 所示：



```
Load model | Accuracy:0.7870
Press any key to continue . . .
```

图 6

最后在程序的根目录下 Python 终端里运行如下所示命令，其中主机端口 localhost port 为 6006：

tensorboard – logdir Runs

便可以在 Tensorboard 相关网页中查看到 train_loss、learning rate 与 Accuracy 的相关结果如下图 7 所示：

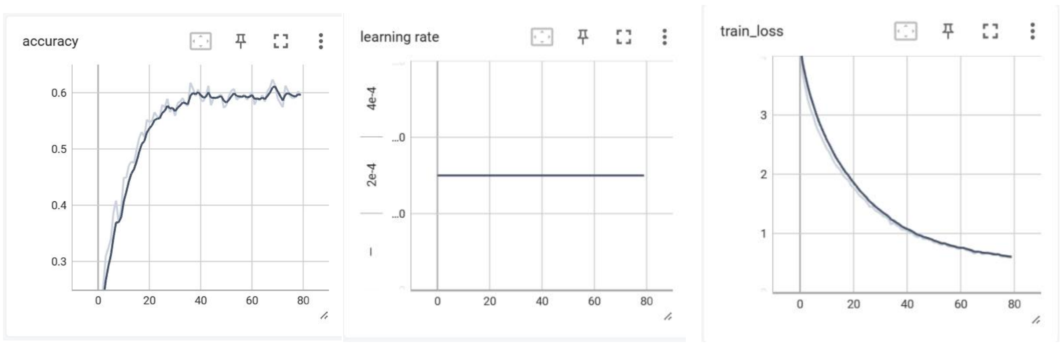


图 9

损失也在一定数量的 epoch 迭代轮次后保持在了 1.00 以下，并没有出现过拟合的现象。

最后我说明一下模型和数据文档归类。在实验的当前文件夹中，alexnet.py 是程序文件，即我自己实现的 AlexNet 网络模型类。train.py 则为训练程序文件。./Data 中是从网站上下载下来的保存 Caltech 101 数据集（提交时已删除），

而./Model 中则保存着已经训练好的模型（即本次实验中我仅保留的最优模型）。./Runs 文件夹保存了 Tensorboard 的日志文件。load_img_data.py 程序文件保存了训练程序中所用到的功能函数（数据集读取，划分，载入等等）。

程序运行方法如下所示：在 train.py 的 main 函数中如果只保留着 train_model() 函数，运行 python train.py –mode train，就会重新训练模型（需要助教重新下载数据集到./Data 文件夹路径下），则可以进行对 AlexNet 网络模型的训练过程，最后训练完成后的最佳模型就会保存在./Model/Best_AlexNet.ckpt 路径文件中；而如果只保留 load_model()函数，运行 python train.py –mode test，就会载入保存好的最优模型进行测试并且打印测试结果，同时在测试集上验证其效果。

源代码以及注释见附件。