手写数字识别卷积网络

1. 环境配置
   1. 程序运行环境

Python3.7+Anaconda+Pytorch+Cuda

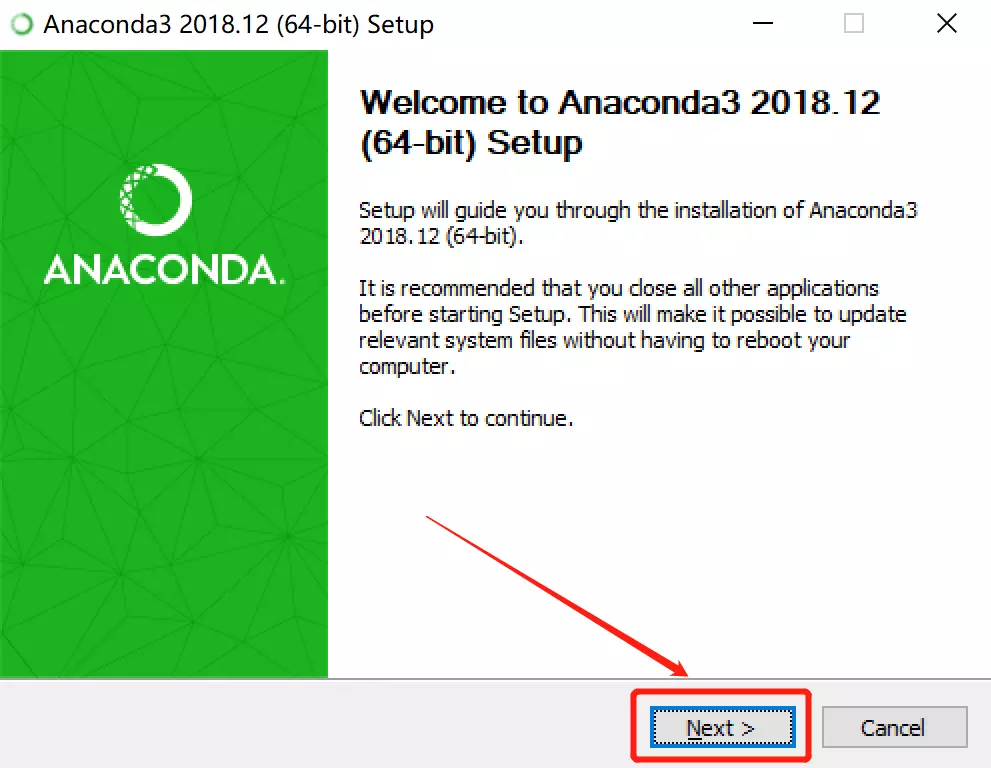
* 1. Anaconda环境配置

第一步：下载安装Anaconda

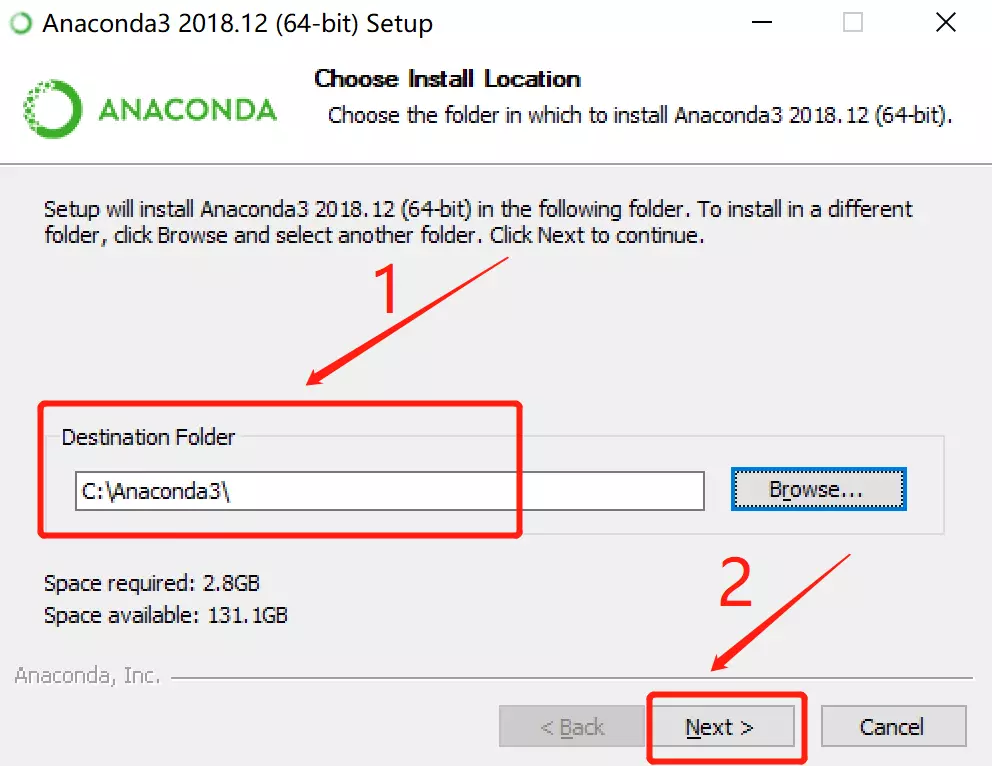
下载地址：<https://www.anaconda.com/download/>

Anaconda 是跨平台的，有 Windows、macOS、Linux 版本，根据自己的操作系统选择相应的版本下载。这里选择Python3.7版本的进行下载。

下载完成后会得到一个类似于Anaconda2-5.2.0-Windows-x86\_64.exe的文件，双击打开进行安装即可。

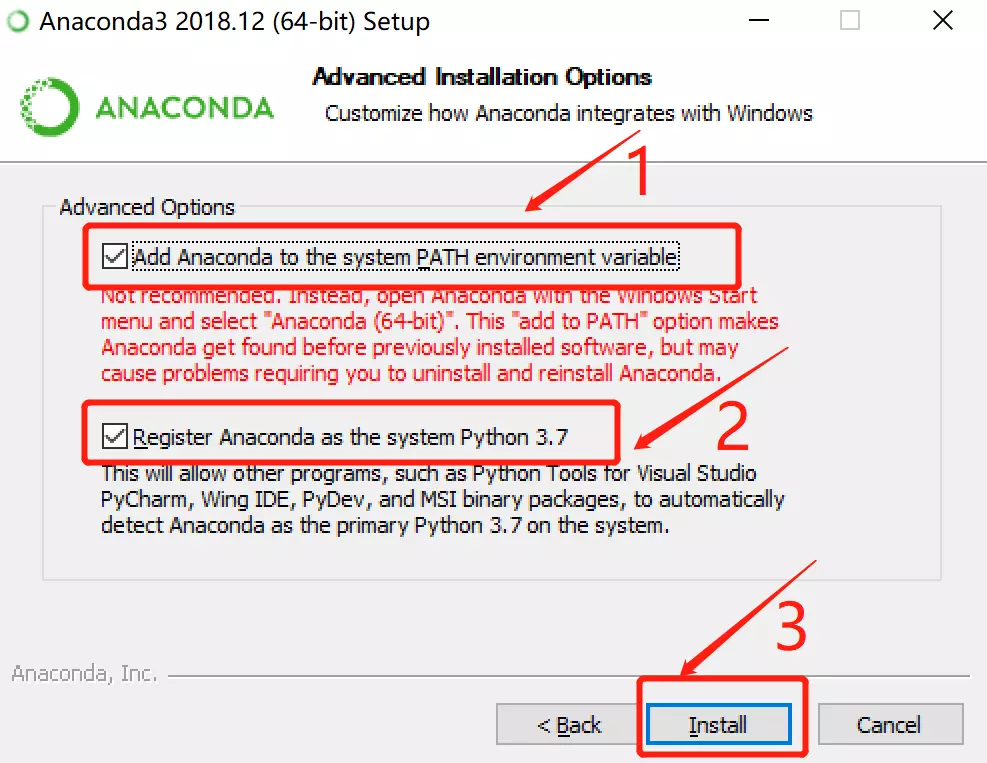


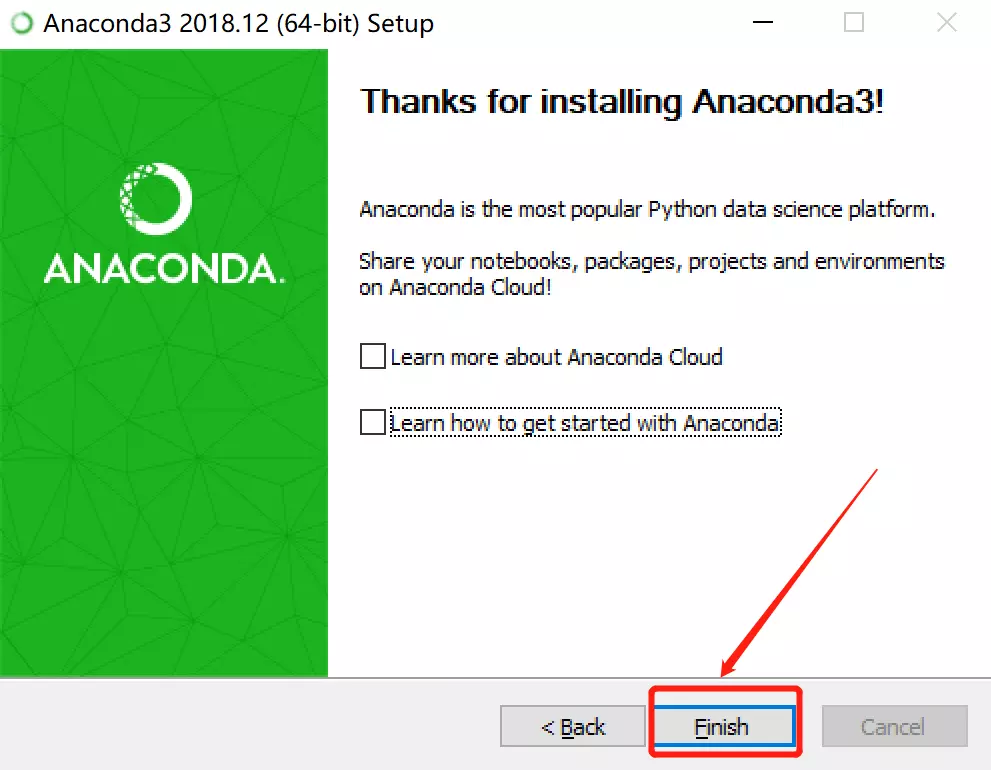
一路点击Next，到这一步：



选择安装路径，也可以安装在默认路径下。

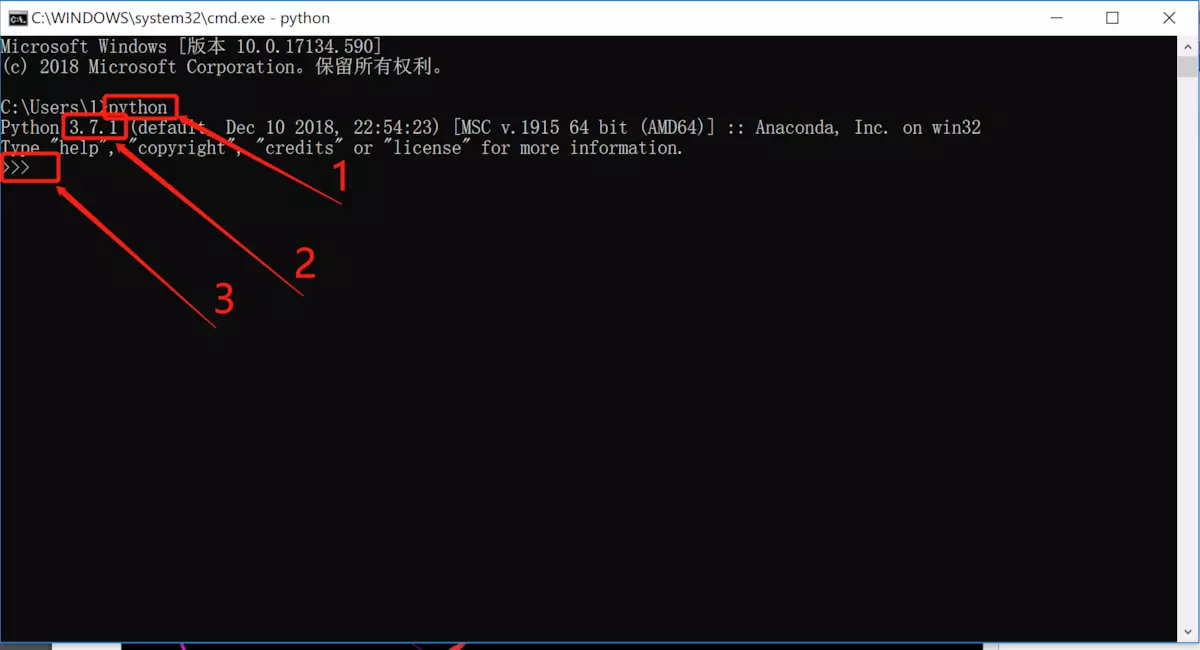
保持两个都在勾选状态，点击安装，等待安装完毕。





安装结束后，打开电脑的CMD命令行（使用WIN键+R， 然后输入CMD，点击OK），

在其中输入 python，如果返回的信息为下图中的2，并且提示符为下图中的3，则安装顺利完成。



第二步：创建Conda虚拟环境

使用 conda create -n your\_env\_name python=X.X（2.7、3.7等） anaconda 命令创建python版本为X.X、名字为your\_env\_name的虚拟环境。your\_env\_name文件可以在Anaconda安装目录envs文件下找到。

使用如下命令即可激活你的虚拟环境：

Linux: source activate your\_env\_name(虚拟环境名称)

Windows: activate your\_env\_name(虚拟环境名称)

关闭虚拟环境使用如下命令即可：

linux: source deactivate

Windows: deactivate

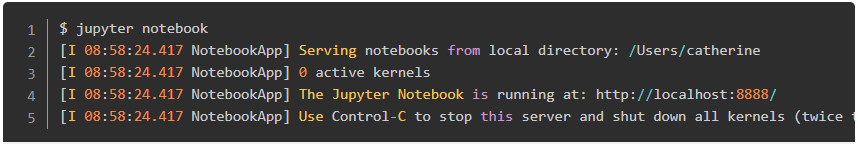
第三步：运行Jupyter notebook

首先在虚拟环境命令行中运行命令：

conda install jupyter notebook

安装完成之后在命令行中切换到文件所在目录下，打开Jupyter Notebook：

jupyter notebook



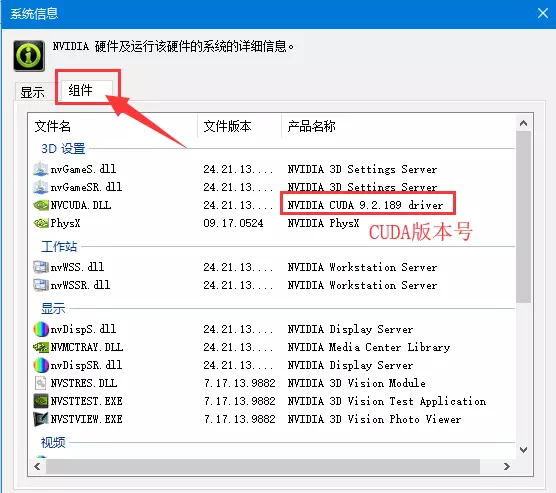
出现这种信息表示打开正常，复制链接到浏览器即可使用。

* 1. Cuda环境配置

首先查看显卡对应的cuda版本，如果版本不对会无法使用。

打开控制面板，点击NVIDIA控制面板，选择进入导航栏的帮助中的系统信息，点击组件，可以看到显卡相关信息。





选择对应版本的Cuda之后进行安装：

下载地址：<https://developer.nvidia.com/cuda-toolkit-archive>

选择自定义安装，只需选择CUDA下面这4项就够了





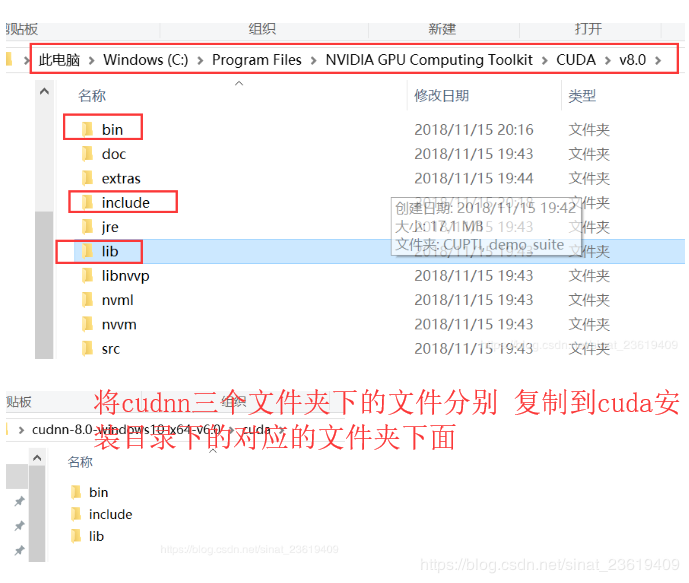
安装完成之后测试一下，命令行输入 nvcc -V，有显示即安装成功。

下载完Cuda之后接着要下载cuDNN：

下载地址：<https://developer.nvidia.com/rdp/cudnn-download>

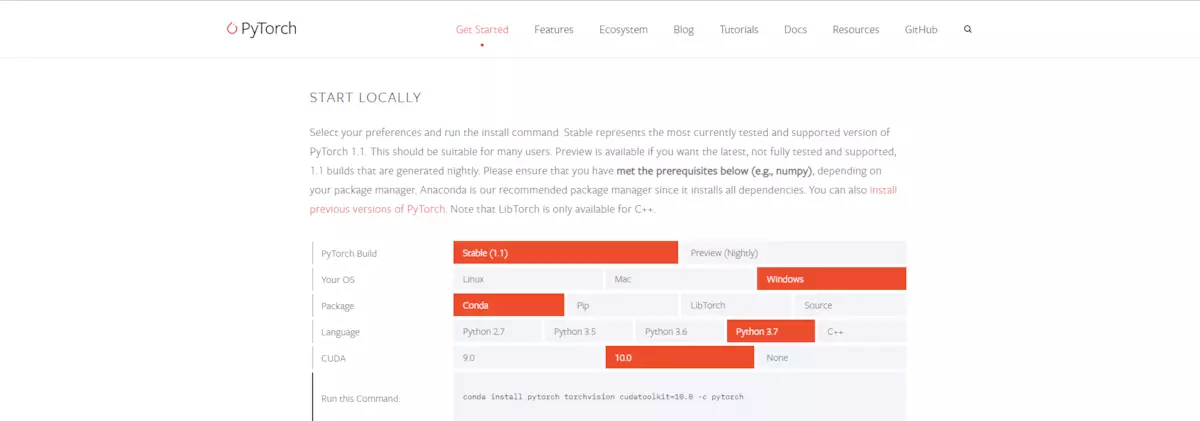
注意要下载与Cuda版本对应的cuDNN。





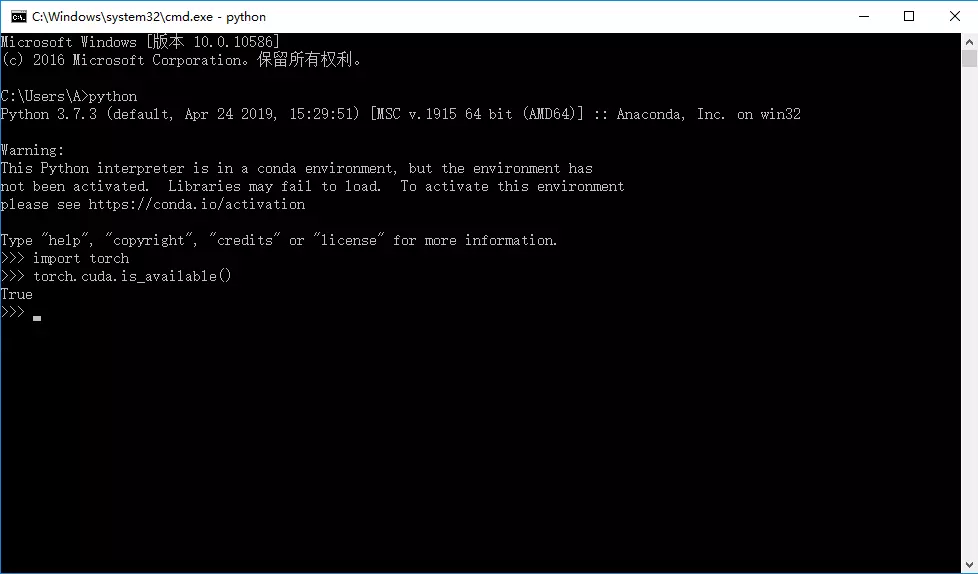
* 1. Pytorch环境配置

下载地址：<https://pytorch.org/get-started/previous-versions/>



在Conda虚拟环境中运行网站给出的命令即可。

验证pytorch的gpu是否可用，返回结果为True则表示GPU可用。



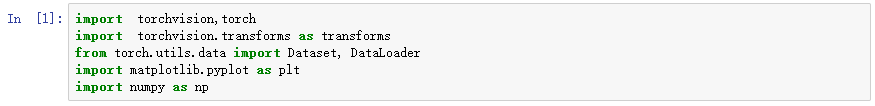
1. 程序运行指导

初次运行程序时可能会缺少一些库文件，使用pip安装相应的库即可。

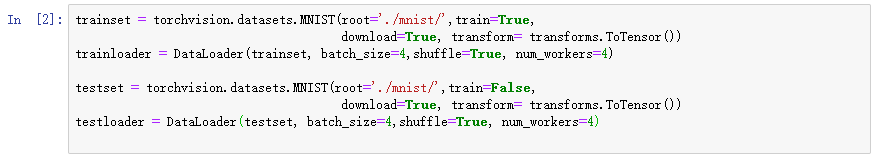
程序有两个版本，一个是.ipynb文件，一个是.py文件，均可以运行。ipynb文件方便观察程序运行的每一步的结果，便于调试和理解。ipynb文件需使用Jupyter notebook运行。

首先看ipynb文件的运行：

在Jupyter Notebook中打开文件，然后对于每一个Cell，按住shift+enter运行。



这个cell中加载了程序需要的一些第三方库文件。



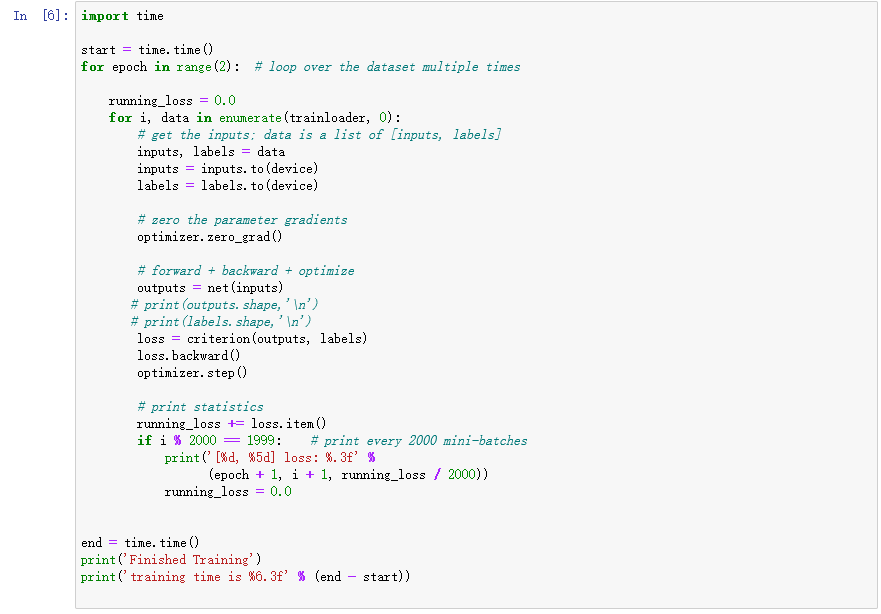
这个cell加载了minist手写数字的数据集。这个数据集已经下载好了，可以直接调用，如果没有下载的话运行这个cell会自动去下载。



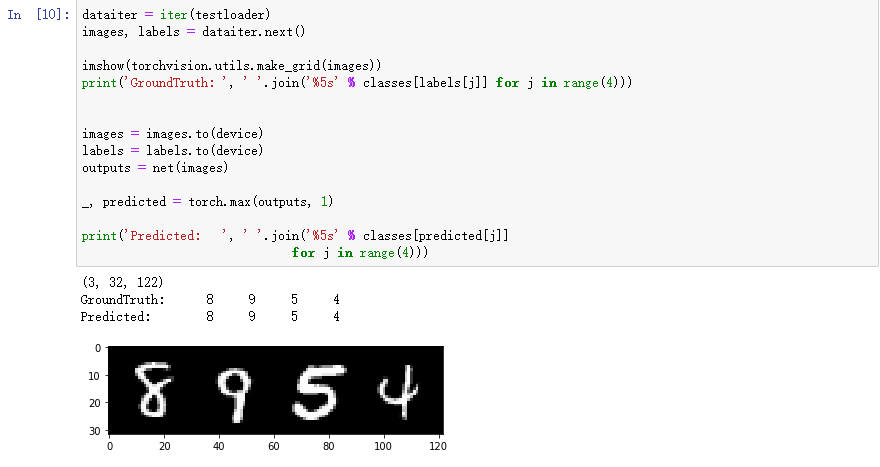
如图，这个cell是对数据加载的正确性进行一次检测，确保训练时数据不会出错。



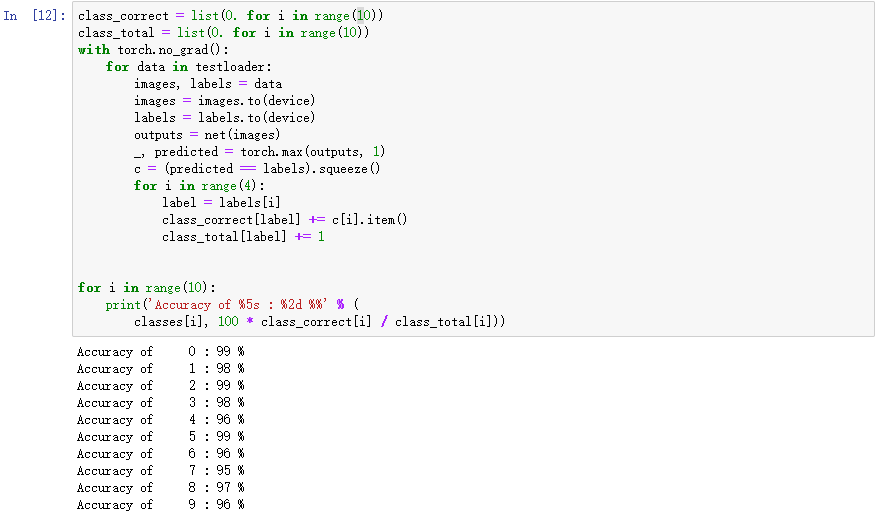
这个cell主要是进行网络的搭建，这里没有使用目前比较流行的图像处理的深度卷积网络模型，而是使用简单的两层卷积配合池化层和三个全连接层一起对图像进行处理。其中device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")控制了这个模型是在CPU还是在GPU上进行训练。ipynb文件需要手动调换训练的平台，py文件可以直接输入参数来调整。



网络搭建完之后开始对网络进行训练，在Minist的训练集上进行两次迭代，同时记录训练的时间。

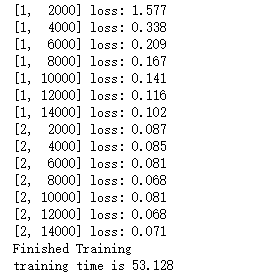
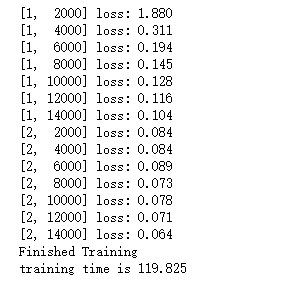


对训练的结果进行一次测试。



对每一类的分类的正确性做一次评估。

1. 训练时间对比

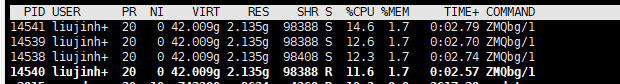
 

左边是使用GPU训练的结果，右边是CPU的结果，可见即使是在一个小网络上，GPU的速度比CPU也有明显的加快。几次的训练结果显示GPU上面的训练时间只有CPU上的一半。

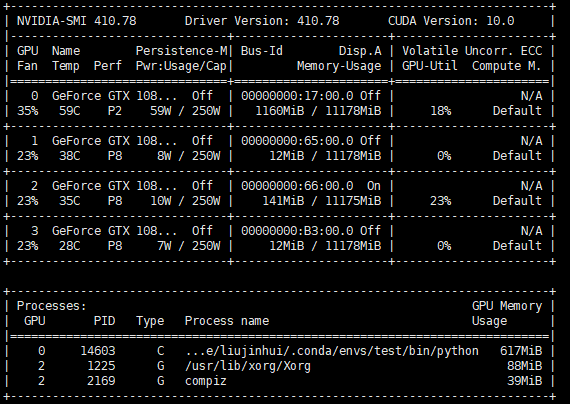
1. 空间占用对比

GPU:

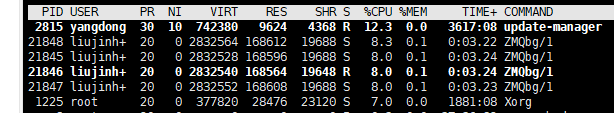
内存消耗：



显存消耗：



CPU:



可以观察到：使用GPU进行训练的时候，内存的消耗为2.13G比使用CPU训练的1.33G要大，同时还伴随着显存的消耗。

1. 结论

神经网络，尤其是卷积神经网络在GPU上进行训练可以极大的缩短训练时间，提高效率，但同时也对资源有更大的消耗。在具体任务时，可以使用其他方法优化代码从而降低内存和显存的消耗。在资源紧张的时候，小型任务可以使用CPU进行训练。