BppAttack复现

目标模型：PreActResNet18

数据集:CIFAR10

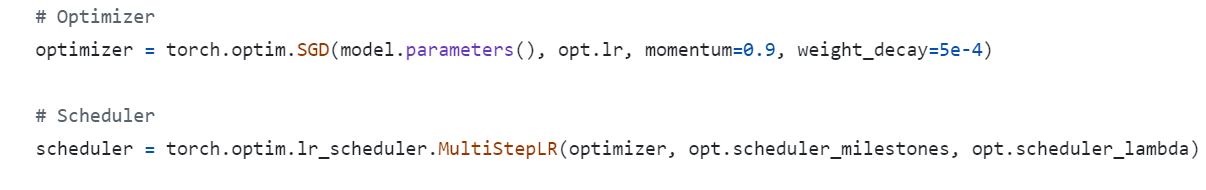
代码理解

BppAttack代码的实现是在WaNet代码的基础上进行修改的，部分代码逻辑核心与其保持一致，所以接下来重点解释不一致的地方即核心代码部分，即bppattack.py。

1. 初始化模型get\_model()



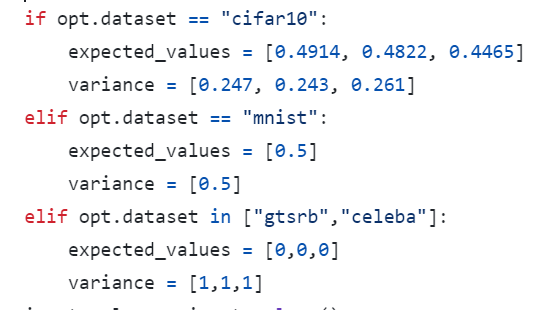
根据传来的参数opt（这是config.py文件里的，包含一大堆相关的变量）选择数据集和网络模型架构。



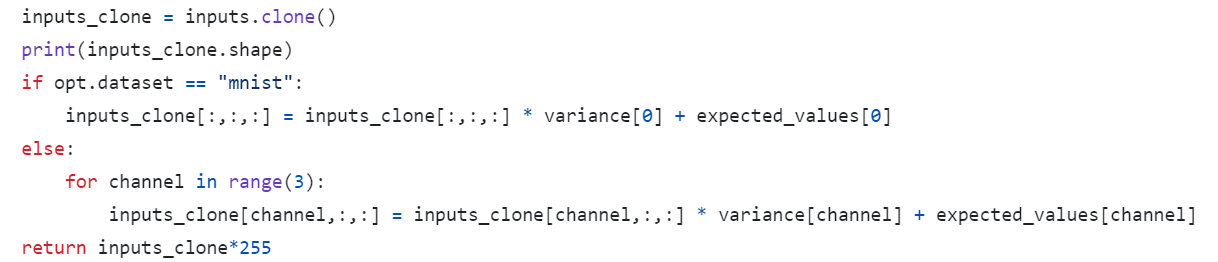
初始化优化器。使用随机梯度下降算法。（SGD）

初始化学习率调度器。给予优化器optimizer的数值来调整学习率，milestones表示里程点，达到该点后学习率乘以lambda数值。

1. 反标准化back\_to\_np()函数



根据数据集选择期望和方差，用于反标准化的计算。



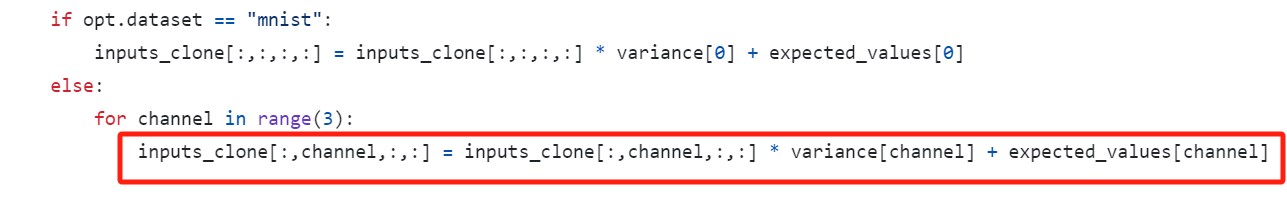
将标准化后的图像转化回原始图像的像素范围值。

先将输入的图片数据进行克隆，防止改变了原始图像数据。

后面即是进行反标准化操作：因为mnist数据集里都是单通道灰度图像（就是黑白的），所以只有一个channel；而其他数据集都是彩色图像（RGB），所以有三个channel。根据对应的期望和方差算出来原始数据。返回值要乘以255，以回到0-255的范围。

1. 反标准化back\_to\_np\_4d()函数

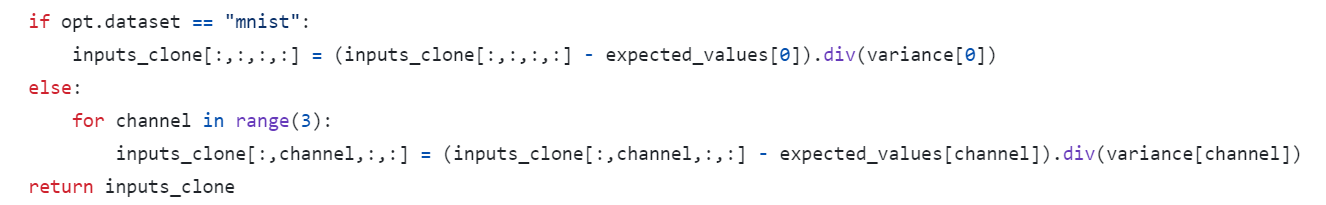
这个函数以上述（2）函数基本上一模一样，只有下述差别。



因为这个用来处理4d图像的，即（batch\_size, channels, height, width），比（2）的3d图像多一个batch\_size。其他一样。

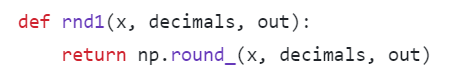
1. 标准化np\_4d\_to\_tensor()函数

这个函数也与上面两个差不多，只是过程是反着的。如下：



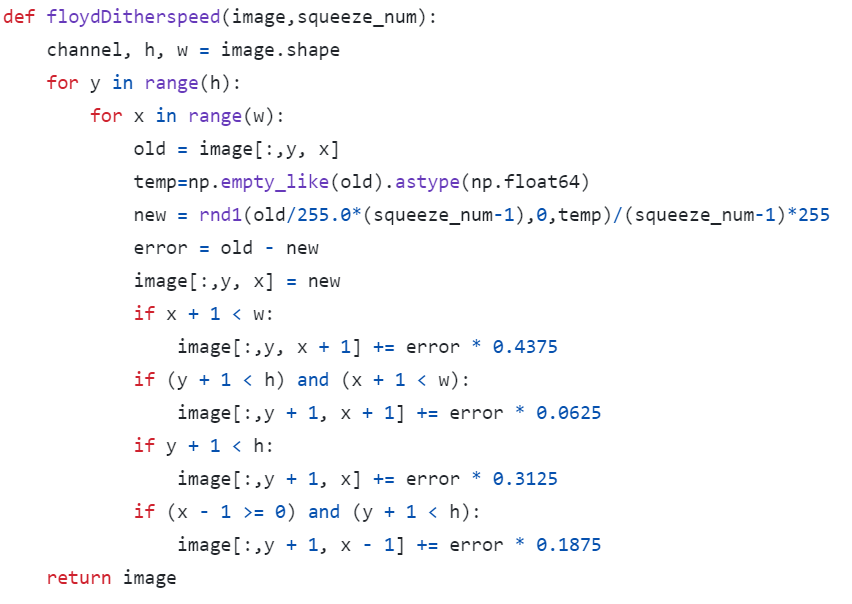
期望和方差都是一样，只是计算过程是完全相反的。div（）函数就是除以括号里面的值，将原本0-255范围的数值标准化到0-1间。

1. 四舍五入处理rnd1()函数



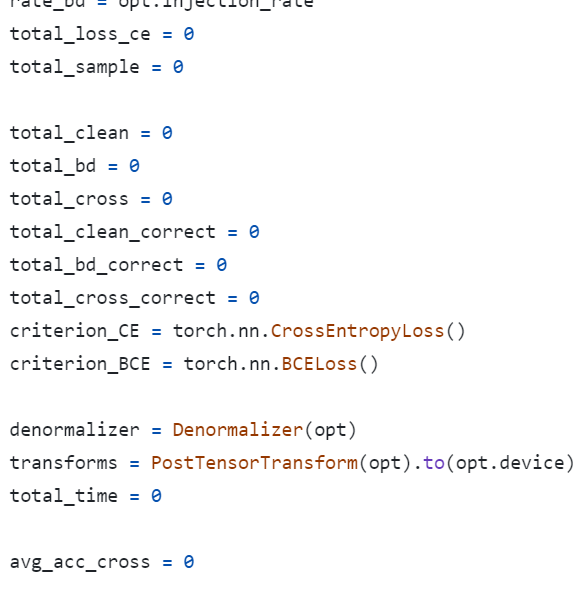
将x进行四舍五入，decimals是要保留的小数点位数，out是结果保存数组。

（6）图像抖动floydDitherspeed()函数



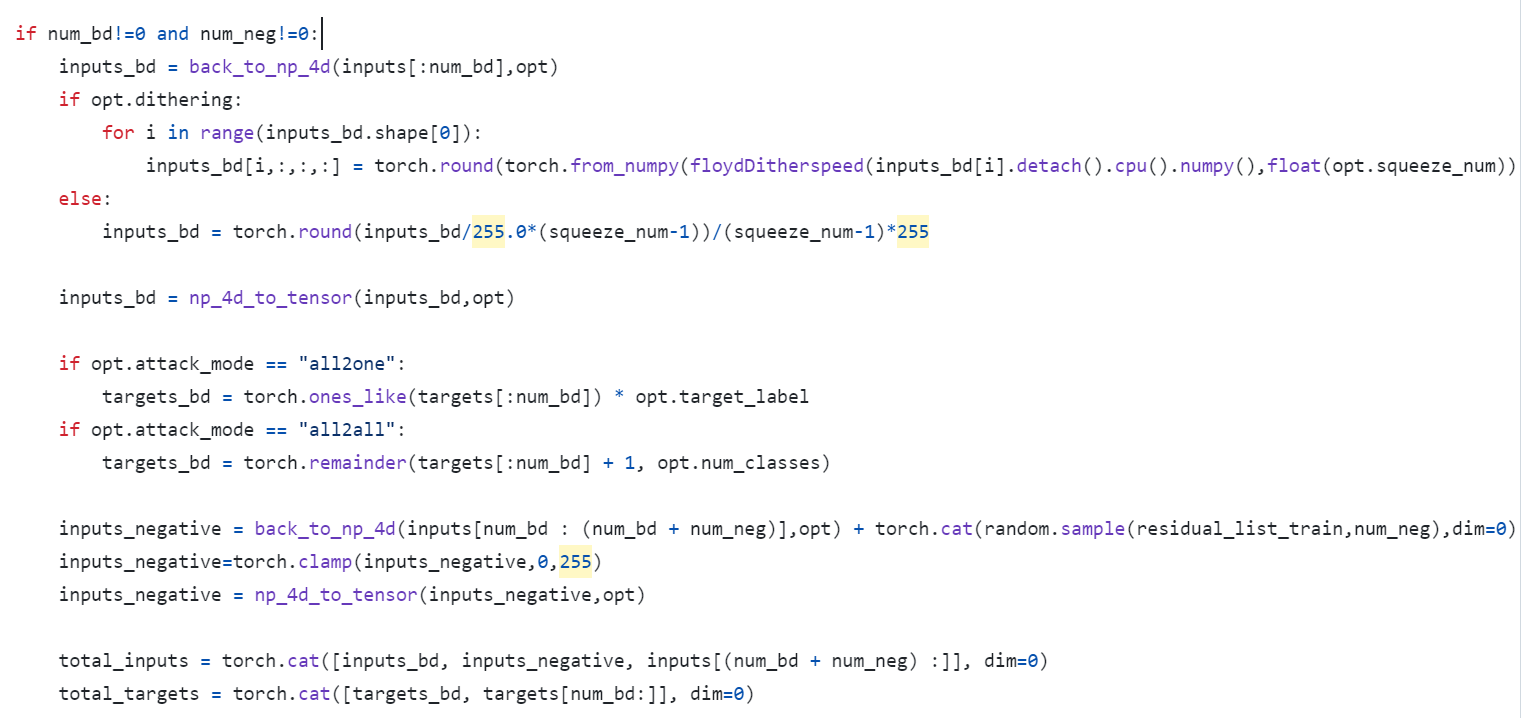
**该算法的计算过程如上。本算法代表着图像抖动过程，是本论文提出的核心之一。其中包括量化操作（根据图片的长和宽），即new这一行，也是核心。“先量化、后抖动”，这是本论文提出的处理方法。**

（7）对抗训练train()函数



一大堆在生成训练数据和训练中要用到的变量值，用于统计数据量等。

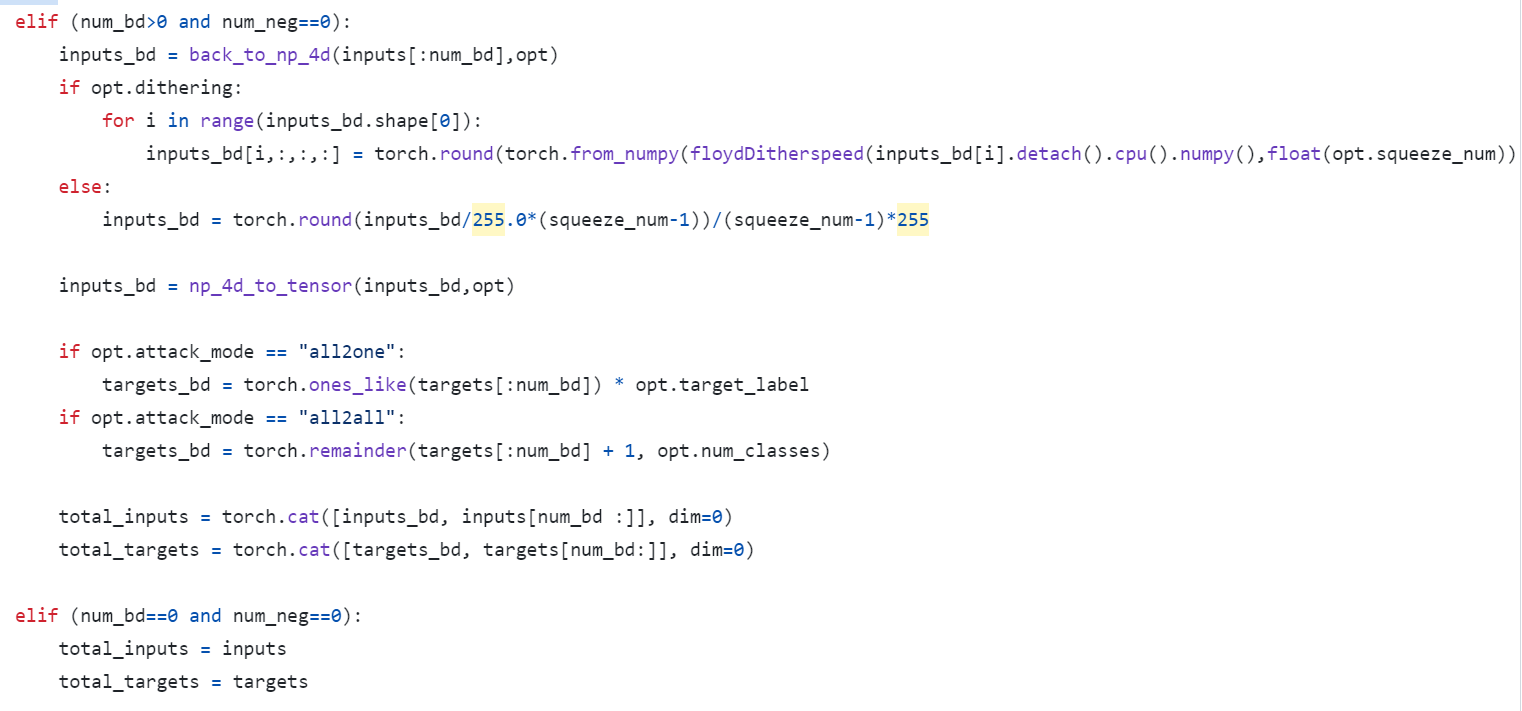
下面就是迭代批次学习，属于重点代码。



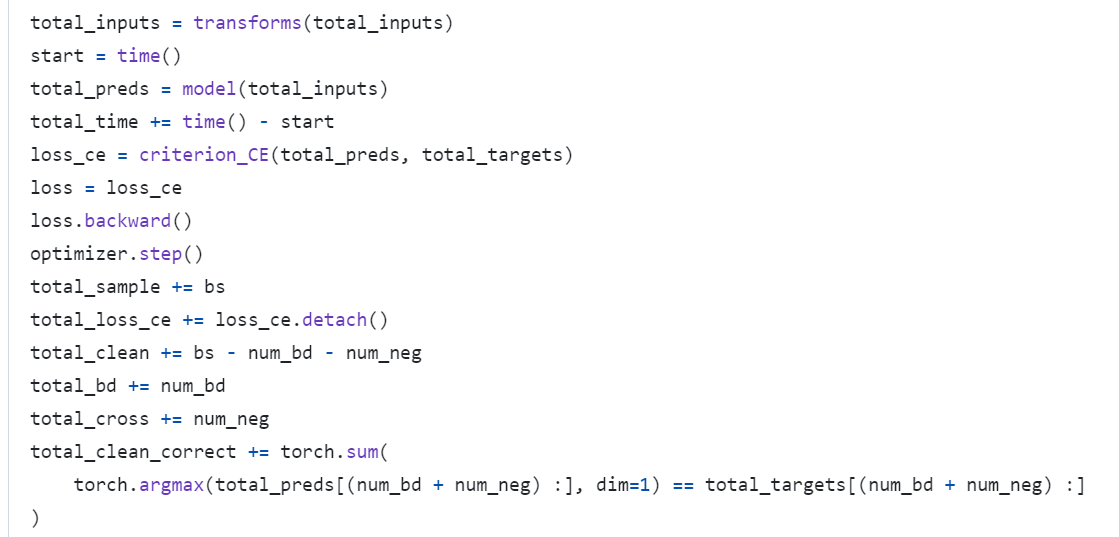
生成后门数据。在后门攻击样本和**负样本（扰动攻击样本，这里就是论文中提出的对比对抗训练的思想）**都不为0时，进行输入数据的处理，包括反标准化（本论文提出的图像量化）、是否进行图像抖动操作，最后对后门攻击样本又进行标准化，将原始数据交付后面的操作进行处理。

根据攻击方式确定后门攻击目标数，后生成负样本。先进行反标准化，在原始像素上进行random.sample（）随机扰动，并用clamp（）函数确保像素值在0-255间，最后再次标准化，交付后面处理。

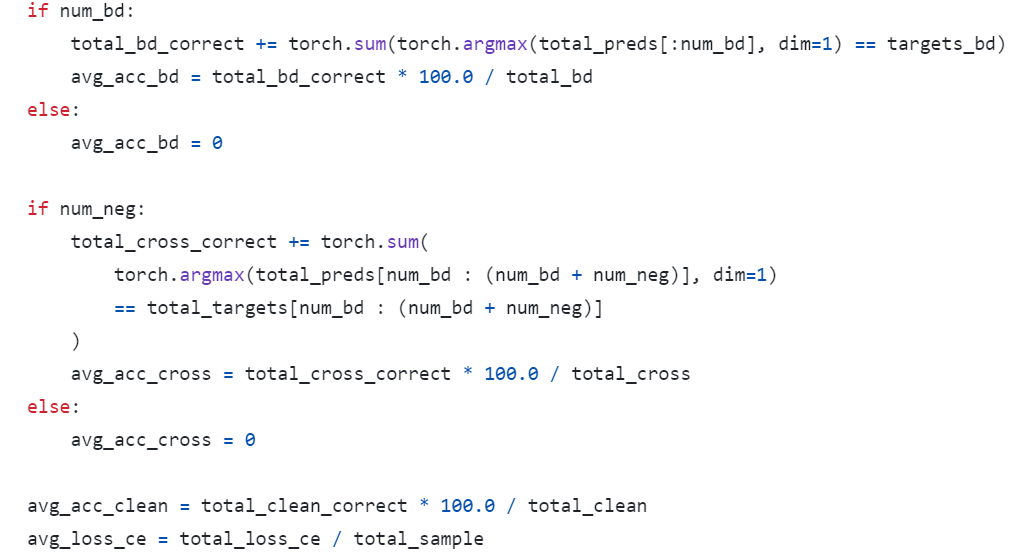
在最后两行代码，就是将处理后的图像和标签合并，形成完整的训练数据。



这两段是根据后门攻击样本和负样本之间的数量（是否存在）进行完整训练数据的生成，处理思想和上面是一样的。

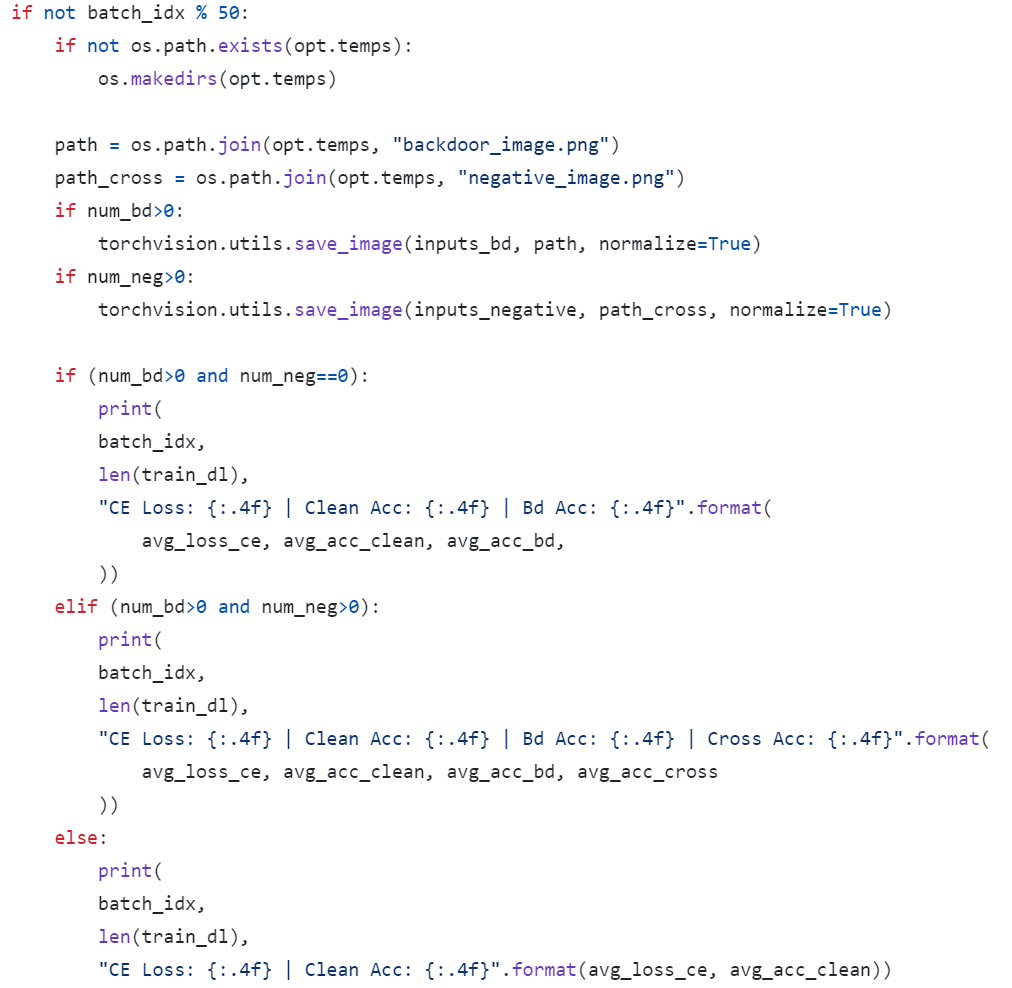


定义训练时涉及到的loss值等变量，在后面会利用其进行计算。

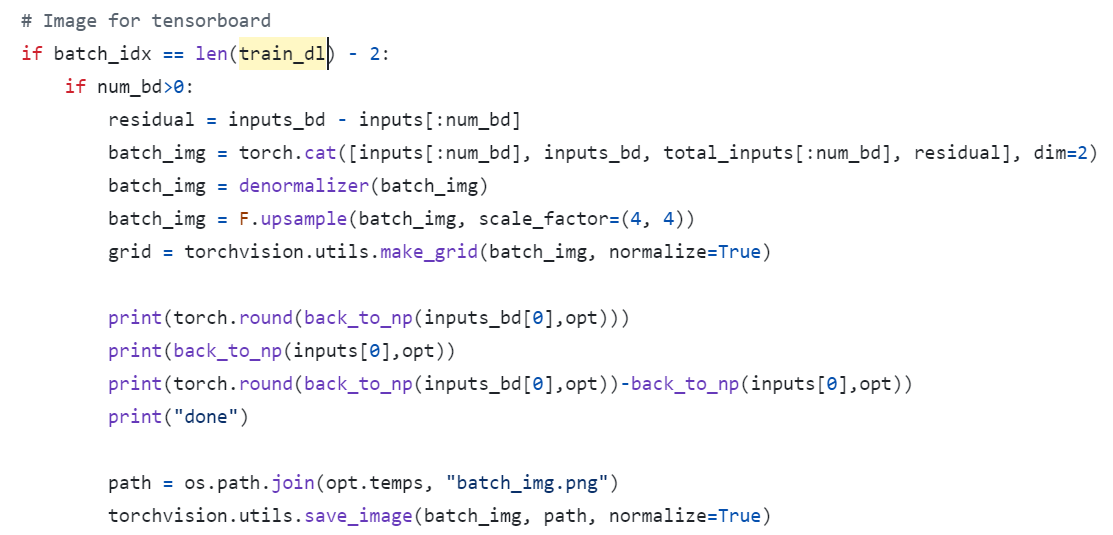


定义后门攻击样本、负样本的成功率。有就通过公式算，没有就是0。

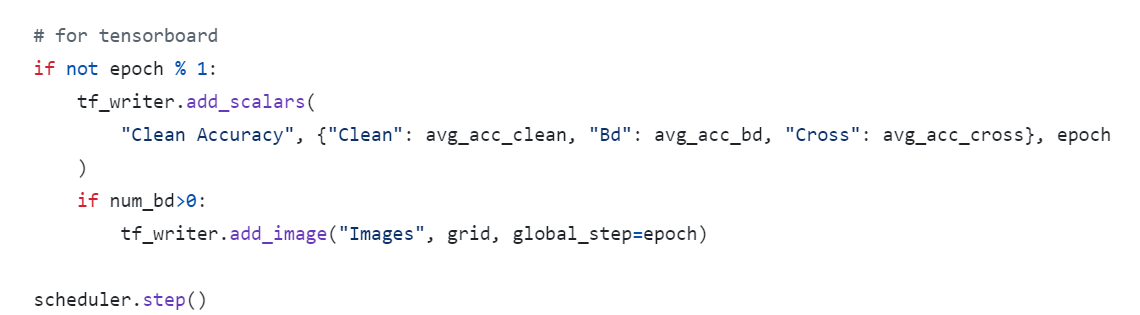
接下来也定义了干净样本的识别准确率和loss值的计算公式。



每 50 个批次执行，定期保存训练中的样本图像，并输出模型的训练状态。在后门攻击样本和负样本数的各种情况下，都要保存，并输出。

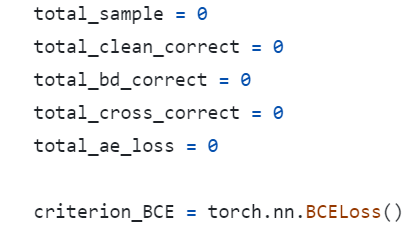


保存一个特定批次的图像，并将其用于 TensorBoard 可视化或其他用途。具体来说，这段代码的功能是展示后门攻击过程中的图像变化，包括原始输入图像、后门图像、带有残差的图像，以及残差本身。



在训练过程中，将一些关键指标和图像信息记录到 TensorBoard 以便后续分析和可视化

（8）评估eval（）函数



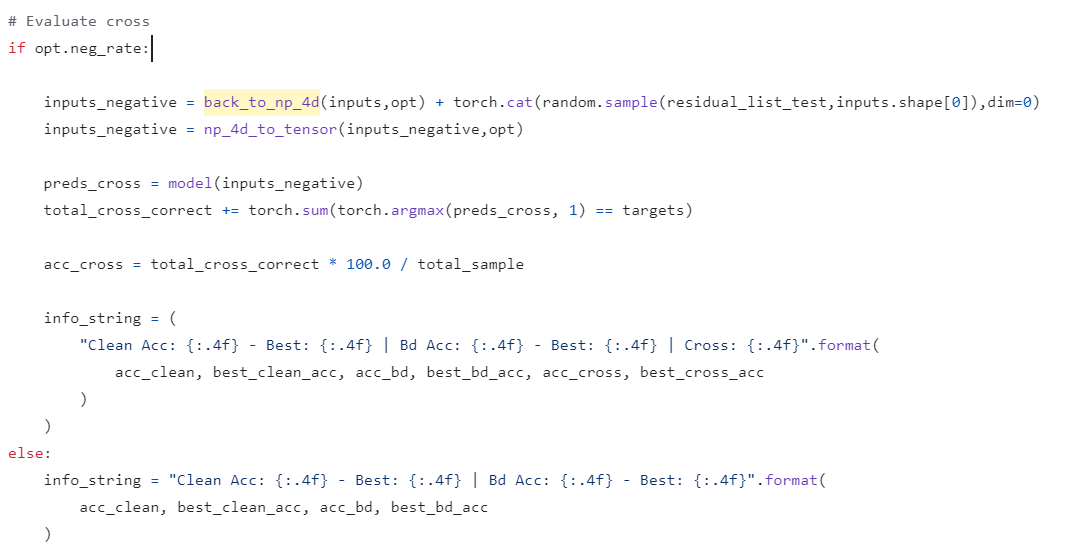
定义各种评估时用的变量，如loss等。

下面就是迭代批次评估，属于重点代码。



估模型在干净样本和后门样本上的性能。具体来说，它计算了模型在这两种样本上的正确预测数量，并输出了这些指标。

该过程大致思想流程其实是和训练过程差不多的，两个过程算的值在本质上差别不大。



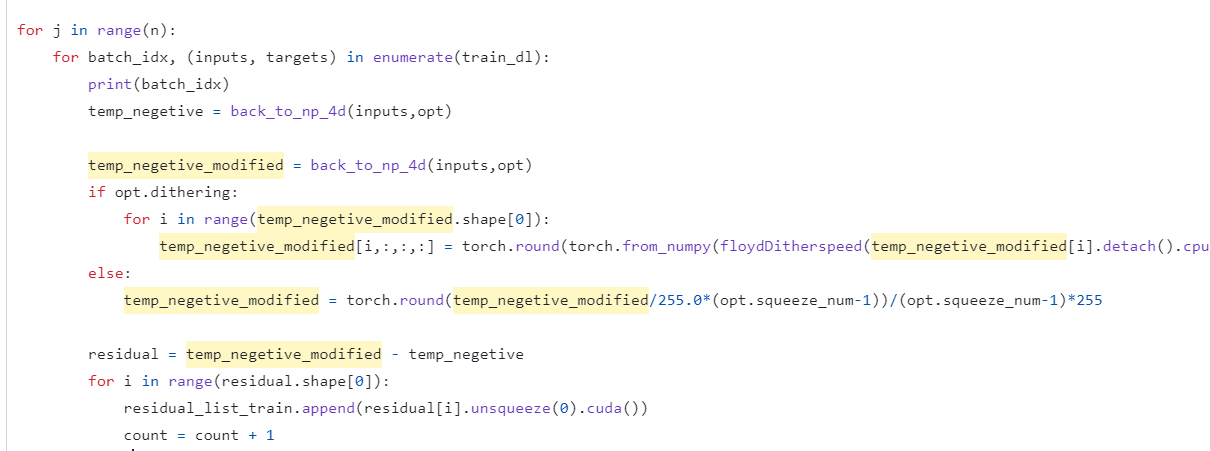
是用来评估模型在交叉扰动样本（即负样本）上的性能，即在原始输入上添加了额外的扰动后的预测准确率，再将干净样本、后门攻击样本、负样本的准确率输出。



这里是进行可视化和保存训练检查点（checkpoints）的代码，非重点。

（9）main()函数

前部分依然是选择数据集、加载数据集、继续训练、从头训练，这个之前已经说明了，就不再说了。



生成扰动并将其添加到 residual\_list\_train 列表中。这些扰动是在干净图像上应用一定的处理（例如，量化和 抖动）后得到的,用于生成训练数据。if判断表示已经进行量化就进行抖动操作，否则先进行量化操作。该量化操作是对图片的反标准化后的张量值。

后面还有一个用于生成测试数据的代码，一样的，就不说明了。

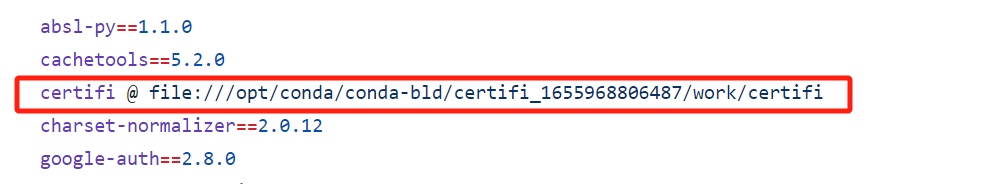
最后就是输出评估过程的各种数据，也不进行说明了。

复现过程：

1.安装环境



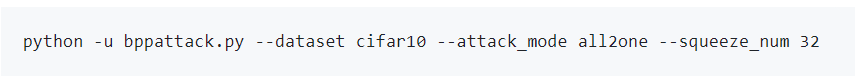
注意：



红框内的文件应该是作者的本地文件，在开源项目里面没有找到，所以我将其替换成最新版的certifi。

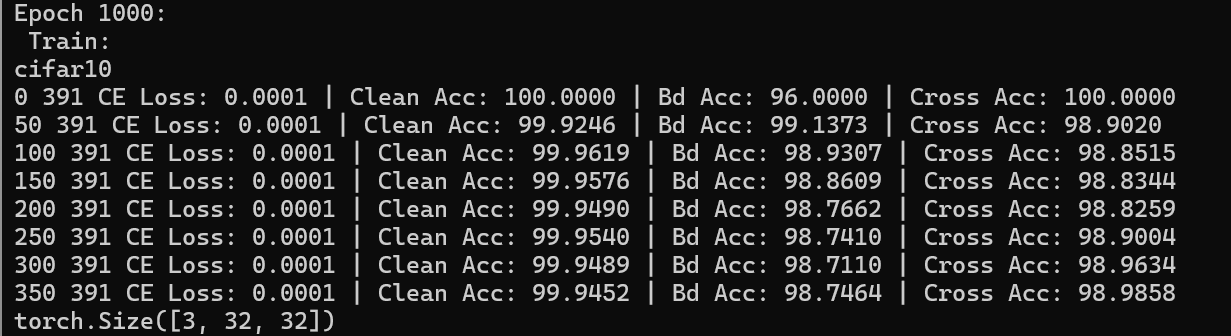
2.训练、评估过程

运行下述命令



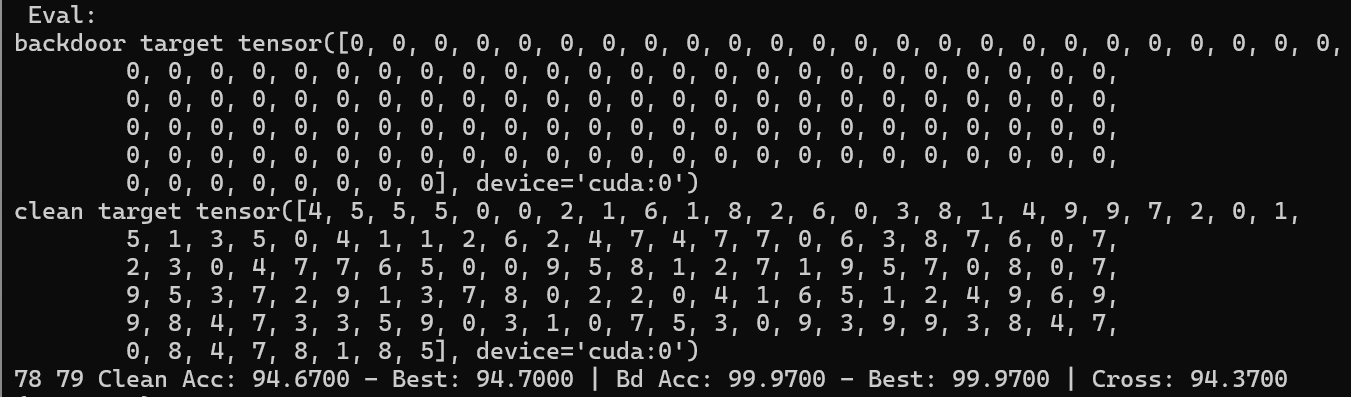
训练过程共一千轮，在每一次的训练后，都会进行一次评估。

训练



训练过程会输出loss以及干净样本、后门攻击样本、负样本准确率。此外，还有在训练过程中的一系列size和张量。

评估

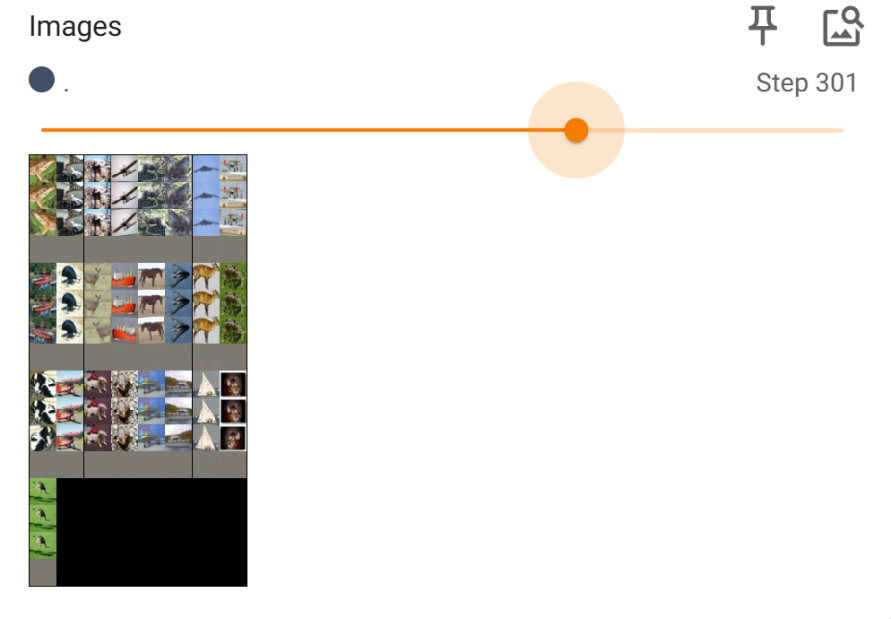


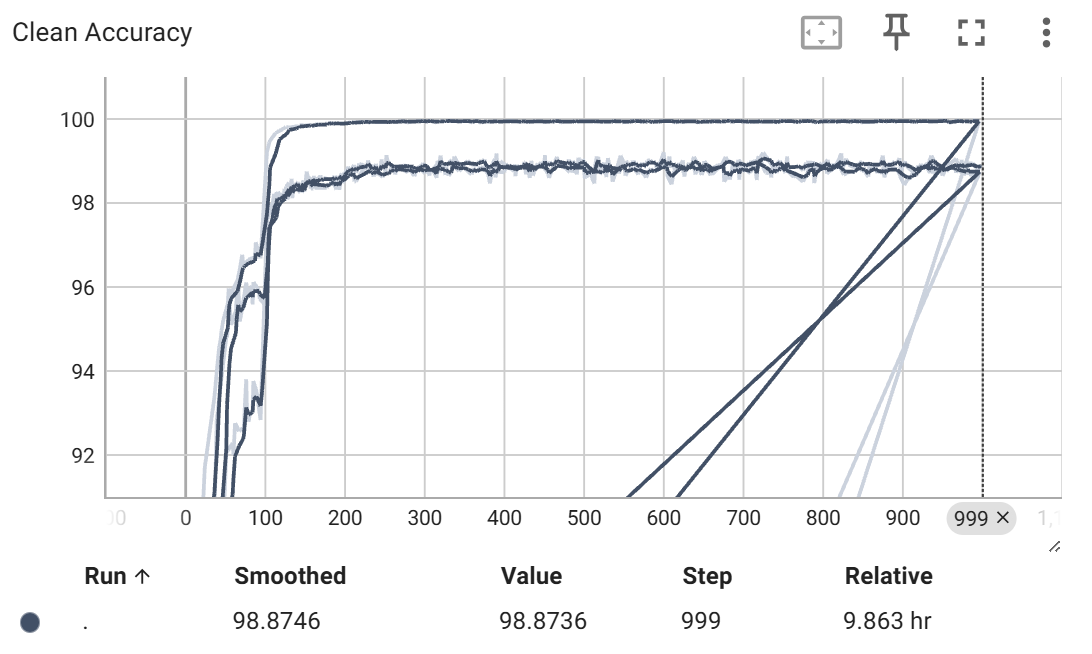
评估过程会输出本次过程和历史最优的准确率。

Backdoor target tensor：全为0，因为本次任务是一个all2one任务，所以目标class就是用一个张量，这里选择0。

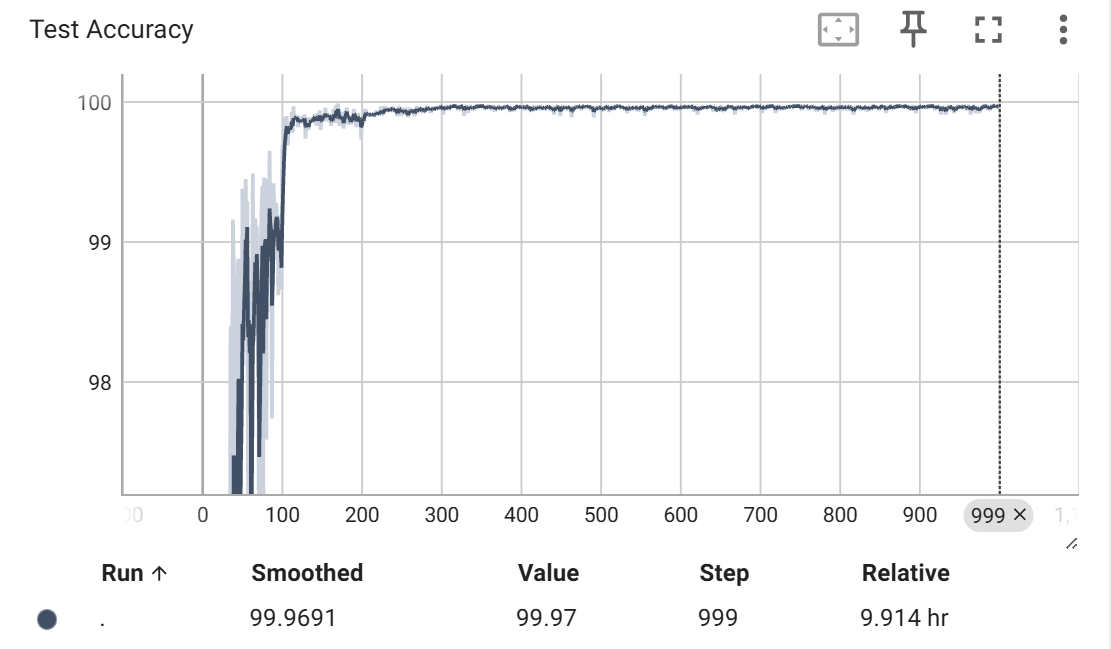
相关图：

攻击时的相关图片





训练过程中，干净样本检测准确率，横轴是训练轮数。



训练过程中，测试样本检测准确率，横轴是训练轮数。

{

"clean\_acc": 94.69999694824219,

"bd\_acc": 99.97000122070312,

"cross\_acc": 94.41999816894531

}

最终准确率。bd\_acc是后门攻击样本攻击成功率；cross\_acc为负样本。

复现难点：

唯一难点就是指定torch版本不适用于服务器，多次尝试更换torch版本后，就OK了。