**Tiny-ImageNet 实验报告**

**中国科学技术大学 化学物理系 禤科材 PB20030874**

**实验细节**

1. **single\_gpu.py、training.py包**

Single\_gpu.py是为了实现单GPU训练而从头开始编写的一个程序，它包含数据集的导入与处理（PreprocessDataset）、网络初始化（PrepareNetwork）以及训练（TrainingNetwork）三个部分,只有TrainingNetwork函数保存在single\_gpu.py文件中，其他函数均存在training.py文件中。

1. **dataset.py包与TinyImageNet类**

TinyImageNet是为了读取TinyImage的数据集而写的一个类。其中\_create\_class\_idx\_dict\_train与\_create\_class\_idx\_dict\_val用于读取图片的类别并根据Train变量的状态分别作用于训练集与测试集上；\_make\_dataset则用于创建具有对应关系的数据集；return\_label等函数则用于检查数据集的状态。

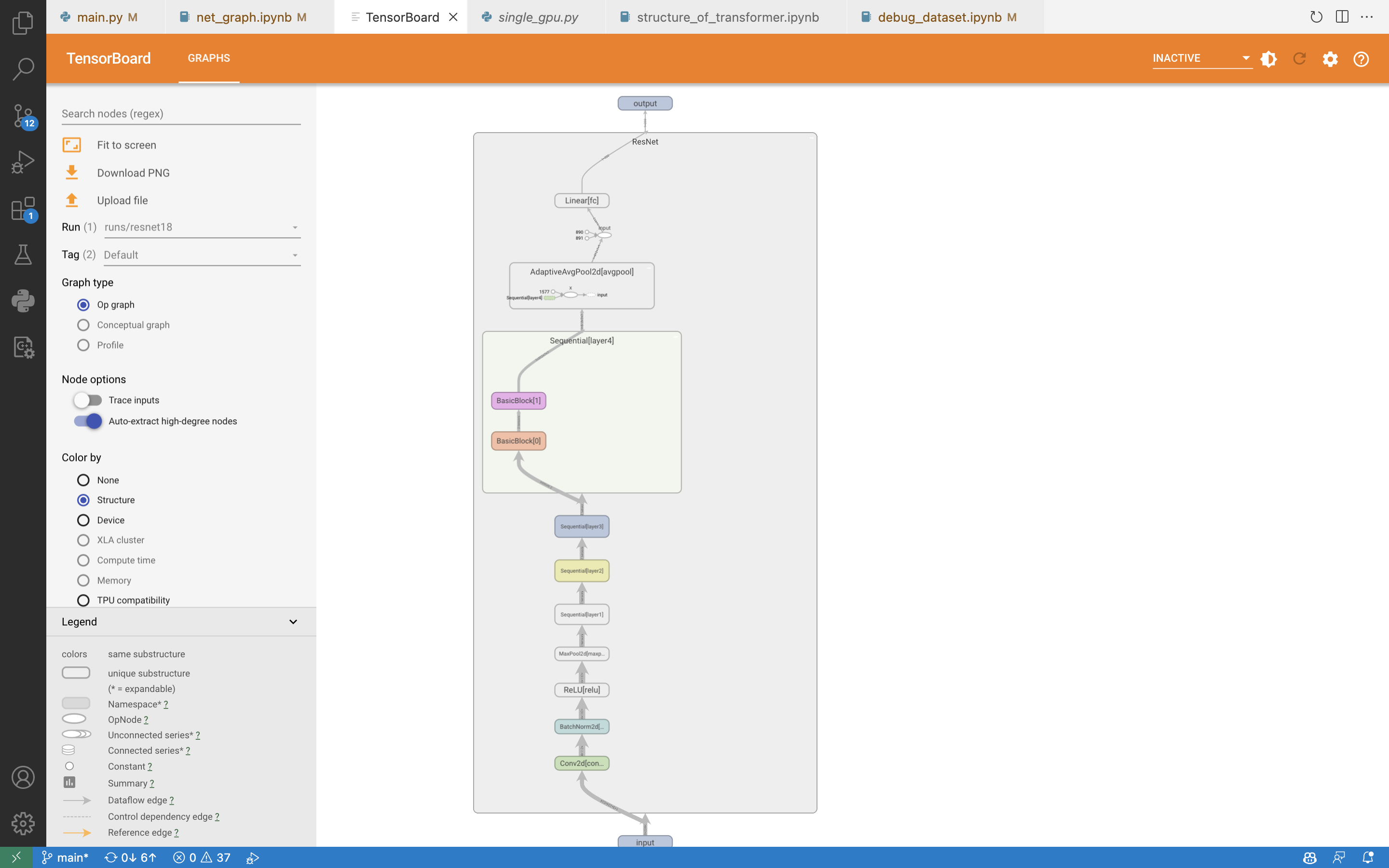
1. **debug\_dataset.ipynb与net\_graph.ipynb**

前者用于找出数据集读取时出现的问题以及检验训练结果，后者用于绘制ResNet18 的结构。

**结果分析**

**网络结构**

使用tensorboard将网络结构打印如下：



**CPU-GPU对比**

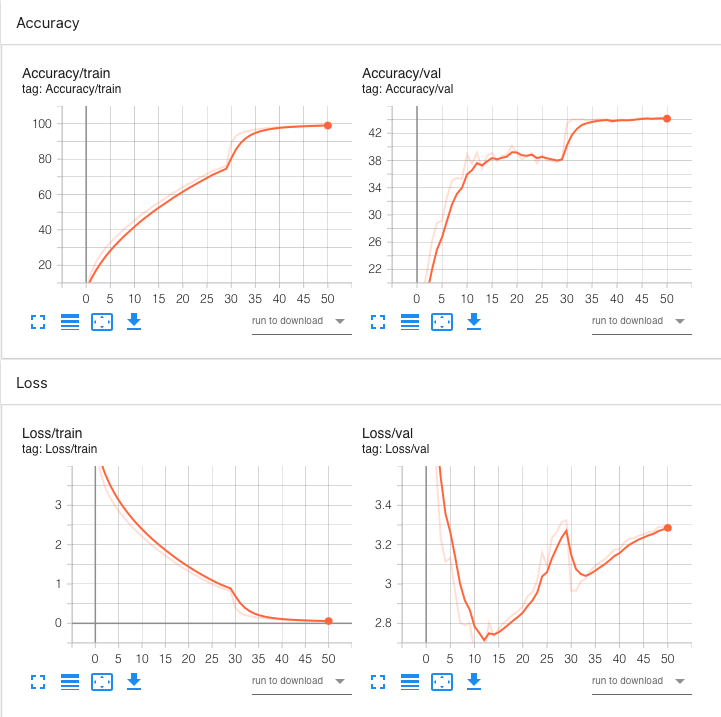
使用CPU训练的时间消耗明显高于GPU。如果仅仅使用CPU训练，2h 21m 57s后仍然在第一个Epoch上停滞不前，而使用1个GPU、2个GPU、4个GPU以及8个GPU都能显著提速，但提速的程度也有差别（控制步数为100 epoch）。



可以看到，每增加一倍的GPU，训练时间的大概可以减半。但6个GPU与8个GPU所消耗的时间差别并不大，这应当归因于多个GPU在相互通信的过程中所消耗的时间也会随着GPU的增加而增长。

**曲线变化情况**

源代码使用的是SGD随机下降的优化方式，从tensorboard的图中可以清晰的看到模型总体而言在训练集、测试集上的准确率均随着训练次数的增加而增长。随着训练次数增加，训练的每一个循环对准确率的贡献逐渐降低，下降也逐渐缓慢，这表明模型逐渐靠近了参数空间中的一个极值点。但在大约30步左右模型准确率出现明显的跃升、交叉熵出现明显的断崖下降，这表明参数空间中的这个极值点实际上并不是局部最值，而是一个鞍点。这个断崖式的下降则说明模型通过随机检索找到了鞍点的下降方向。

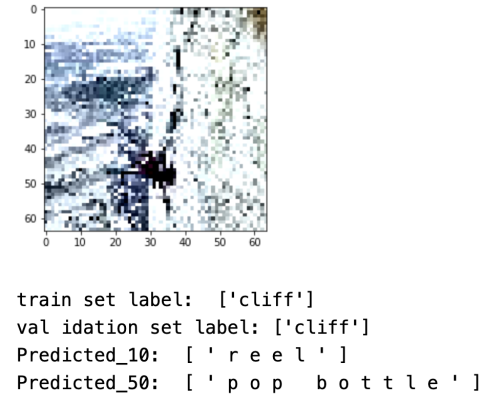
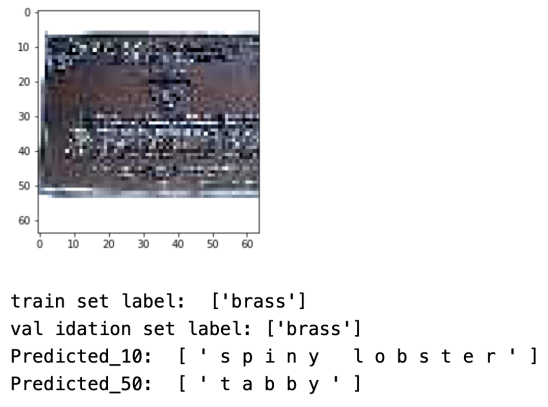


随着训练步数的增长，鞍点前后的曲线走势都基本相似。然而测试集上的loss函数却呈现W型，这表明随着训练次数的增加，模型不可避免地出现了过拟合。

**评估结果**

将模型保存至本地并导入，评估不同步数时的训练结果差异。在这一步出现一些问题，本地测试结果准确率仅在0.008附近徘徊，靠近随机作答的准确率；然而远程的tensorboard却显示有至少40%的准确率。初步判断问题出在数据集的读取上，但直到交作业也没有找出问题所在。

所以基本每一张图片的判定结果都不相同（这也算形式上完成了作业吧），参见debug\_dataset.ipynb中的结果：



**难点与解决方案**

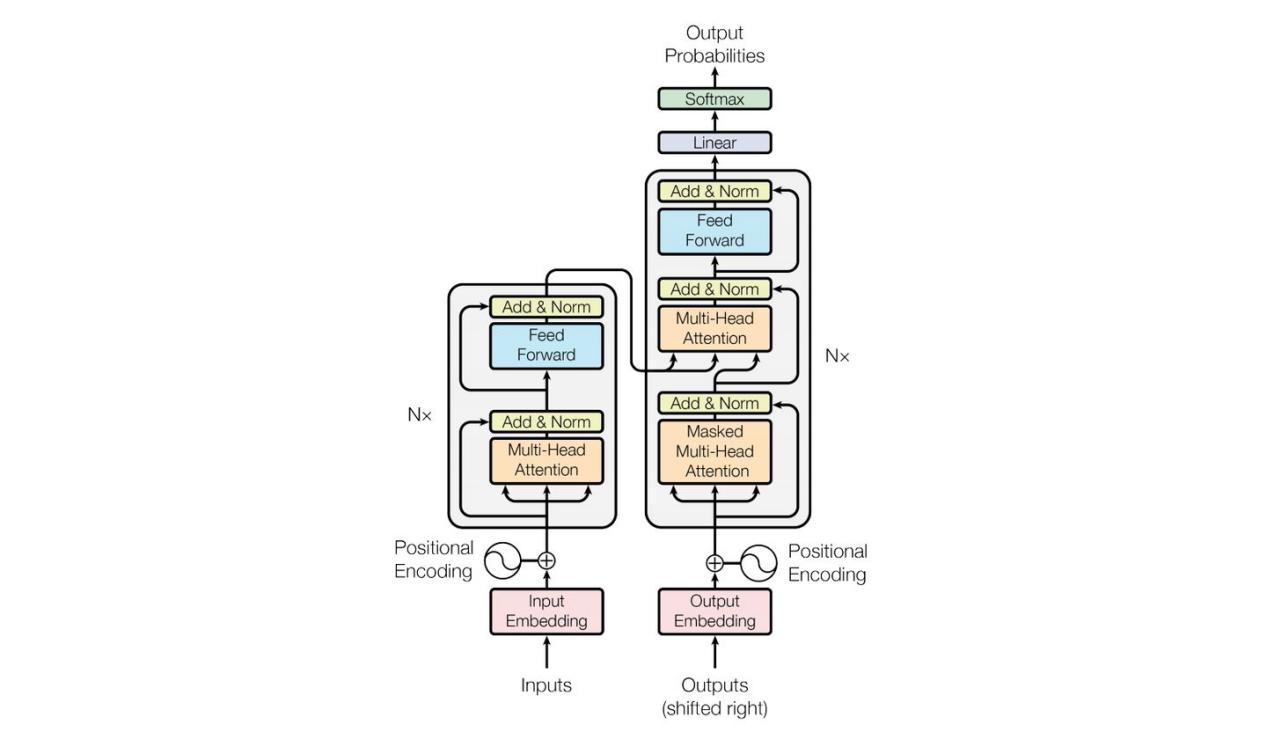
本次实验使用了GitHub最新出品的Copilot辅助编程，非常好用。结合newbing在搜索上的优点，编程中的笔误、符号等小问题基本上在它们的帮助下可以说是完全消失了。

本次实验最初一切顺利，在tensorboard的结果显示也很好（约40%准确率），但是到了本地却出现了一些问题，体现为本地预测准确率在随机回答准确率0.005左右徘徊。

目前已排查：

1. 无论是单GPU（自己写的）还是多GPU（使用pytroch示例修改的）均发现这个问题，这说明是数据集的读取问题
2. train\_set与val\_set中序号与类型的对应关系是相同的
3. 在取消train\_set的transform处理后问题仍然存在
4. 不同checkpoint对应的model对同一组照片的预测结果几乎不同

**复现transformer**



**演示**

参见视频：

**实验总结**

本次实验不太顺利，限于时间，最后并没有找出训练的问题所在。但从写代码的过程中仍然收获不少。在代码书写期间，从什么都不会到稍微会一点点，也是非常有成就感的一件事。

在编程过程中，我大致了解了RESNET的基本原理，也翻阅了一些书籍论文，大致懂得了残差块是如何改善深度网络梯度爆炸的。由于刚刚拿到pytorch的示例代码的时候完全看不懂，所以使用单个GPU的程序single\_gpu.py是自己一步一步写完的。在这个过程中我知道了一些深度学习代码的基本书写方式，例如parser在开头表示从命令行输入参数、main函数检查各项指标，并协调其他函数、main\_worker函数则进一步从parser中获取参数，并启动train、test等核心代码，而更基本的train、validate等函数则随时待命。这样的结构的确有利于程序的开发与调试。

我大致知道了pytorch学习框架的各部分都能够实现什么功能，以及为了完成一个特定的深度学习任务都有哪些步骤、这些步骤分别对应pytorch的那些模组。作为一个3个月前连python都不会的化院同学，在不断bug-debug的过程中也认识了很多计算机、人工智能专业的学长学姐（如果不上这门课我是不可能和他们有任何交流的），这样的经验也是非常宝贵的。