写在前面

写这份笔记的初衷是记录自己入门科研的过程，对期间遇到的问题及解决做一个记录，仅供个人参考和使用.

Contents

1

服务器使用

组里目前有 2 台 8 卡 RTX 4090 GPU 机架服务器，1 台 2 卡 RTX 2080ti GPU 塔式服务器和 1 台数据存储服务器；服务器彼此间不互联.

1.1 基本使用流程

4090 服务器是通过思腾合力的SCM 人工智能云平台调度使用的，需要在平台上注册账号并通过审核方可登录. 对于一个项目，首先需要将数据上传到自己的空间，将环境打包为 docker 镜像并上传到自己的镜像仓库，然后新建一个项目，为其指定镜像. 有了以上基础，就可以开始愉快地调试代码了.

1.2 文件管理

文件管理很方便，直接上传即可. 超过 1G 的需要使用 FTP ，目前使用的工具是 FileZilla.

1.3 镜像管理

SCM 人工智能云平台采用 Harbor 管理镜像仓库，提供自定义开发环境的功能，用户可以将特定的开发环境封装成 Docker 镜像，上传到服务器并进行管理，待下次使用，直接启动镜像即可，快速而简单，镜像管理功能主要功能有查看项目、设置项目成员、设置项目仓库权限、复制镜像标签下载命令、删除镜像标签以及导入镜像.

具体来说，**容器** (container) 是在主机上运行的沙盒进程，它与该主机上运行的所有其他进程隔离. 而**镜像** (image) 是正在运行的容器使用隔离的文件系统. 要在云平台上建立项目，首先需要上传 docker 镜像到自己的容器中. 常用的环境在 dockerhub 上基本都能找到.

在 windows 下需要安装 docker desktop 以及 wsl2 ，此类安装问题通常都有小坑，折腾一下问题不大. 之后就是找到自己需要的镜像. 例如 pytorch lightning 官方

在 dockerhub 上发布了一系列镜像，点击页面中的”tags”，即镜像的**标签**, 复制右侧的拉取命令，在终端执行即可.

拉取镜像之后就可以执行 save 命令，将其保存在指定的位置. docker 的镜像文件位置可以在 docker desktop>>Settings>>Resources>>Advanced 中找到，是一个硬盘文件，大小将近 30 个 G. 保存过程较长，需耐心等待一下. 查看进度可以通过刷新看硬盘空间少了多少.

需要注意的是，docker save 命令似乎每次都会导致空间的占用，所以尽量不要退出，不然会导致空间一下子变少. 这个问题的解决方案还没有找到.

得到.tar 文件后，将其上传到数据卷，然后通过镜像管理导入镜像文件即可.

2

深度学习代码

2.1 代码基本结构

对于基本的深度学习任务，至少包括了数据 (data)、模型 (model)、训练 (train)、测试 (test)、评估 (evaluation) 等步骤，在代码中以配置文件的方式指定以上路径，能够使代码更清晰．一些工具函数会放在”utils” 文件夹中，方便调用．若有展示需要，可加入 demo 脚本，调用预训练模型和可视化库．使用 requirements.txt 文件来部署 python 环境．下面是一个文件树的示例：

```
project
├── scripts ..... python 脚本
├── configs ..... 训练/测试配置文件
├── checkpoints ..... 训练过程中的权重文件
├── utils ..... 工具函数文件夹
│   ├── preprocess ..... 预处理脚本
│   └── eval.py ..... 评价指标
├── models ..... 模型脚本文件夹
│   ├── main.py ..... 主模型
│   └── components.py ..... 模型组件
├── train.py ..... 训练脚本
├── test.py ..... 测试脚本
└── demo.py ..... 展示脚本
```

实际情况可能会更复杂，例如使用.sh 脚本操作文件，使用 CMakeLists.txt 编译 C++ 脚本等．但深度学习代码的基本架构是类似的，这使我们可以快速找到项目中的训练脚本和模型脚本．

2.2 GPU 显存优化

如果 CUDA error: out of memory 这句话后面跟了 Tried to allocate ...MB 的话，是显存不够，可以尝试缩小 batchsize 等方法来节省显存；但是如果后面没有这句话，

说明是 GPU 被占用或 torch 版本与 pre-trained model 版本不匹配.

2.3 任务

1. 生成低分辨率深度图，低分辨率点云，高分辨率深度图，作为训练数据. 其中低分辨率深度图输送到 U-Net 中，得到粗略的深度重建图；低分辨率点云输送到 FCGF 中，得到特征图；最后通过一个融合模块得到精细的深度重建图.
2. 在 dataloader 中加载低分辨率深度图，低分辨率点云和高分辨率深度图，分别表示为 $3 \times N, 1 \times L_1 \times R, 1 \times L_2 \times R$ 的张量.
3. 评估指标需要 MAE、MSE、CD、EMD、F1-Score 等

3

Linux

3.1 常用命令

3.1.1 Linux 系统命令

查看系统版本	**lsb_release -a**
当前文件夹内容	**ls 或 ll**
返回上级文件夹	**cd ..**
进入文件夹	**cd xxx**
安装 Python 依赖	**pip install xxx**
root 用户跳过警告	**pip install –root-user-action=ignore xxx**
查找文件名	**find xx/xx -name xxx 或 locate xxx**
查看显卡信息	**nvdia-smi**
创建文件夹	**makedir xxx**
创建文件	**touch xxx**
显示占用显存的进程	**fuser -v /dev/nvidia***
释放进程	**kill -9 xxx**
查看文件个数	**ls -p | grep -v / | wc -l**

3.1.2 Conda 命令

创建虚拟环境	**conda create -n envname python=x.x**
查看已安装虚拟环境	**conda env list**

3.1.3 pip 命令

指定镜像源	**pip install xxx -i https://xxxxx**

4

问题及解决

4.1 Pip 相关

Example 4.1

安装 Hydra 时，使用命令

> pip install -i http://pypi.douban.com/simple/ hydra-core==1.3.2

返回报错

> ERROR: Cannot determine archive format of /tmp/pip-req-build-qmx413h5

Solution. 添加对镜像源的信任，将命令改为

> *pip install -i http://pypi.douban.com/simple/ –trusted-host pypi.douban.com hydra-core==1.3.2*

■

Example 4.2

安装时遇到版本不兼容的警告

> ERROR: After October 2020 you may experience errors when installing or updating packages. This is because pip will change the way that it resolves dependency conflicts.
>
> We recommend you use –use-feature=2020-resolver to test your packages with the new resolver before it becomes the default.

Solution. 按照指示，在安装命令后面添加 *–use-feature=2020-resolver* 即可

> *pip install -i http://pypi.douban.com/simple/ –trusted-host pypi.douban.com pytest –use-feature=2020-resolver*

■

Example 4.3

安装 Pytorch-Lightning 就算降低版本，也会自动安装最新的 torch，而且是 CPU 版本.

> Installing collected packages: torch

> Attempting uninstall: torch

> Found existing installation: torch 1.9.1+cu111

Solution. 卸载掉新安装的 *torch*，重新按照官网的命令安装一遍 *torch* 和 *cuda* 即可，之后 *import* 一下 *pytorch_lightning* 看看是否安装成功. ■

5

CMake

5.1 C/C++ 编译过程

Definition 5.1

编译器 (compiler) 是一种计算机程序，它会将某种编程语言写成的**源代码**转换成**可执行文件**. 源代码是便于人编写、阅读、维护的高级计算机语言，而可执行文件是计算机能解读、运行的低阶机器语言.

一个现代编译器的主要工作流程如下：源代码 (source code) → 预处理器 (preprocessor) → 编译器 (compiler) → 汇编程序 (assembler) → 目标代码 (object code) → 链接器 (linker) → 可执行文件 (executables)，最后打包好的文件就可以给电脑去判读执行了.