pytorch/serve

**demo**

首先是要把模型封装成包含metadata特定格式的mar，

torch-model-archiver --model-name densenet161 --version 1.0 --model-file ./serve/examples/image\_classifier/densenet\_161/model.py --serialized-file densenet161-8d451a50.pth --export-path model\_store --extra-files ./serve/examples/image\_classifier/index\_to\_name.json --handler image\_classifier

然后用封装好的torchserve脚本就可以启动，

torchserve --start --ncs --model-store model\_store --models densenet161.mar

直接运行HTTP client就可以跑通demo。

**API**

它对用户侧提供了inteface api和management api。Management api 可以调节使用的CPUGPU数量和batch delay,以及设置入口handler等配置。Interface api主要是暴露8081端口提供interface服务。

Torchserve也可以部署到容器中，进而使用K8S进行管理。

**存取模型**

提供从网上下载以及直接在本地Load

（torch.load(model.pth)）

<https://github.com/pytorch/serve/blob/master/docs/custom_service.md>

自建handler的教程

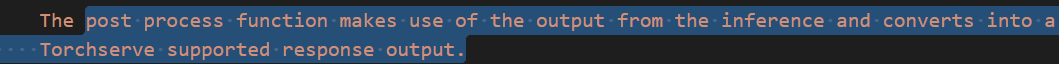
起handler的过程（我还没弄明白handler是worker启动就启还是request来了再起）

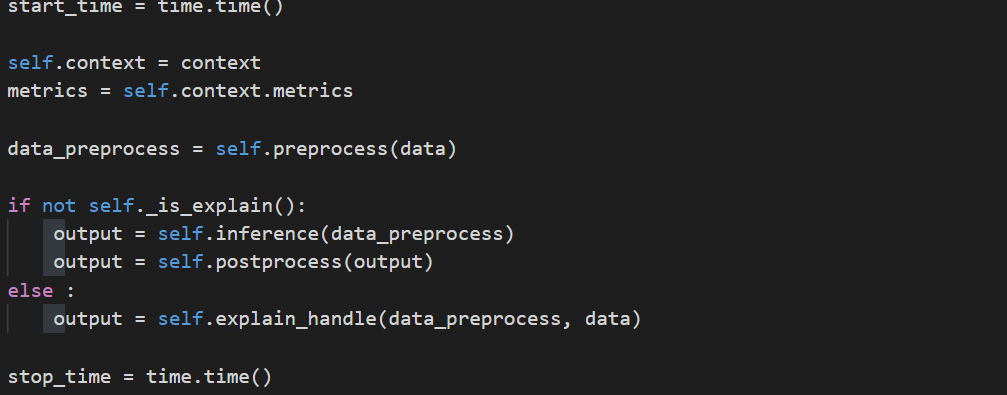
First try to load torchscript else load eager mode state\_dict based model











**实现介绍（在网上收集的资料，不熟悉JAVA以及PY项目的编译调试）**

但是不同模型的输入输出不一样，如果不是跑预置好的mnist其实都是要自己写handler的，handler就是一个Python类要实现inference接口，inference就是自己import pytorch跑一次forward。

PyTorch是C++和Python实现的，而pytorch/serve的HTTP server是使用Java Netty实现的。

start函数（这里在Python VM里），这个函数只是为了拼接一个Java命令然后调用subprocess模块来执行子进程（操作系统系统调用），使用Java命令启动了Java的ModelServer类，这里使用Netty启动HTTP server（这里在JVM里），通过实现各种handler可以处理各种HTTP请求例如在线predict的请求，但这里并**没有直接调用JNI或者py4j去调用底层模型而是把任务放到自己管理的队列中**。

后台线程池处理任务队列居然没有任何代码依赖JNI或者py4j，而是又用Netty连了一个TCP server（也可以是Unix socket），真正的调用不在JVM进程里。而是在**Python侧还有一个TorchModelServiceWorker的实现**（这里在Python VM里），用于加载实际的PyTroch模型逻辑，所以最终其实还是通过PyTorch Python API来做实际模型加载和预估逻辑，只是比直接调用Python预估多了Python -> OS -> Java -> Python这个逻辑而已。

安装依赖时遇到一点问题

我还没有成功运行demo(由于无法正确安装依赖，worker始终起不起来)



