> 知乎地址: https://zhuanlan.zhihu.com/c_1101089619118026752
> 作者: 小哲
> github: https://github.com/lxztju/notes
> 微信公众号: 小哲AI

提供两种pytorch模型的部署方式,一种为web部署,一种是c++部署

1. web部署

web部署就是采用REST API的形式进行接口调用。

web部署的方式采用flask+ redis的方法进行模型部署,pytorch为模型的框架,flask为后端框架,redis是采用键值的形式存储图像的数据库。

各package包的版本:

```
pytorch                   1.2.0
flask                     1.0.2
Redis                     3.0.6
```

1. Redis安装,配置

ubuntu Redis的安装,下载地址:https://redis.io/download

安装教程: https://www.jianshu.com/p/bc84b2b71c1c

```
wget http://download.redis.io/releases/redis-6.0.6.tar.gz
# 拷贝到/usr/local目录下
cp redis-3.0.0.rar.gz /usr/local
# 解压
tar xzf redis-6.0.6.tar.gz

cd /usr/local/redis-6.0.6

# 安装至指定的目录下
make PREFIX=/usr/local/redis install
```

Redis配置:

```
# redis.conf是redis的配置文件，redis.conf在redis源码目录。
# 拷贝配置文件到安装目录下
# 进入源码目录，里面有一份配置文件 redis.conf，然后将其拷贝到安装路径下
cd /usr/local/redis
cp /usr/local/redis-3.0.0/redis.conf  /usr/local/redis/bin
```

此时在/usr/local/redis/bin目录下,有如下文件:

```
redis-benchmark redis性能测试工具
redis-check-aof AOF文件修复工具
redis-check-rdb RDB文件修复工具
redis-cli redis命令行客户端
redis.conf redis配置文件
redis-sentinal redis集群管理工具
redis-server redis服务进程
```

Redis服务开启:

```
# 这是以前端方式启动,关闭终端,服务停止
./redis-server

# 后台方式启动
#修改redis.conf配置文件， daemonize yes 以后端模式启动

cd /usr/local/redis
./bin/redis-server ./redis.conf
```

连接Redis

```
/usr/local/redis/bin/redis-cli
```

关闭Redis

```
cd /usr/local/redis
./bin/redis-cli shutdown
```

强行中止Redis,(可能会丢失持久化数据)

```
pkill redis-server
```

2. server端

```
@app.route('/predict', methods=['POST'])
def predict():

    data = {'Success': False}

    if request.files.get('image'):

        now = time.strftime("%Y-%m-%d-%H_%M_%S",time.localtime(time.time()))

        image = request.files['image'].read()
        image = Image.open(io.BytesIO(image))
        image = image_transform(InputSize)(image).numpy()
        # 将数组以C语言存储顺序存储
        image = image.copy(order="C")
        # 生成图像ID
        k = str(uuid.uuid4())
        d = {"id": k, "image": base64_encode_image(image)}
        # print(d)
        db.rpush(ImageQueue, json.dumps(d))
        # 运行服务
        while True:
            # 获取输出结果
            output = db.get(k)
            # print(output)
            if output is not None:
                output = output.decode("utf-8")
                data["predictions"] = json.loads(output)
                db.delete(k)
                break
            time.sleep(ClientSleep)
        data["success"] = True
    return jsonify(data)

if __name__ == '__main__':

    app.run(host='127.0.0.1', port =5000,debug=True )
```

3. Redis服务器端

```
def classify_process(filepath):
    # 导入模型
    print("* Loading model...")
    model = load_checkpoint(filepath)
    print("* Model loaded")
    while True:
        # 从数据库中创建预测图像队列
```

```
        queue = db.lrange(ImageQueue, 0, BatchSize - 1)
        imageIDs = []
        batch = None
        # 遍历队列
        for q in queue:
            # 获取队列中的图像并反序列化解码
            q = json.loads(q.decode("utf-8"))
            image = base64_decode_image(q["image"], ImageType,
                                        (1, InputSize[0], InputSize[1],
Channel))
            # 检查batch列表是否为空
            if batch is None:
                batch = image
            # 合并batch
            else:
                batch = np.vstack([batch, image])
            # 更新图像ID
            imageIDs.append(q["id"])
            # print(imageIDs)
        if len(imageIDs) > 0:
            print("* Batch size: {}".format(batch.shape))
            preds = model(torch.from_numpy(batch.transpose([0, 3,1,2])))
            results = decode_predictions(preds)
            # 遍历图像ID和预测结果并打印
            for (imageID, resultSet) in zip(imageIDs, results):
                # initialize the list of output predictions
                output = []
                # loop over the results and add them to the list of
                # output predictions
                print(resultSet)
                for label in resultSet:
                    prob = label.item()
                    r = {"label": label.item(), "probability": float(prob)}
                    output.append(r)
                # 保存结果到数据库
                db.set(imageID, json.dumps(output))
            # 从队列中删除已预测过的图像
            db.ltrim(ImageQueue, len(imageIDs), -1)
        time.sleep(ServeSleep)



def load_checkpoint(filepath):
    checkpoint = torch.load(filepath, map_location='cpu')
    model = checkpoint['model']  # 提取网络结构
    model.load_state_dict(checkpoint['model_state_dict'])  # 加载网络权重参数
    for parameter in model.parameters():
        parameter.requires_grad = False
    model.eval()
    return model


if __name__ == '__main__':
    filepath = '../c/resnext101_32x8.pth'
    classify_process(filepath)
```

4. 调用测试

```
curl -X POST -F image=@test.jpg 'http://127.0.0.1:5000/predict'
```

```
from threading import Thread
import requests
import time

# 请求的URL
REST_API_URL = "http://127.0.0.1:5000/predict"
# 测试图片
IMAGE_PATH = "./test.jpg"

# 并发数
NUM_REQUESTS = 500
# 请求间隔
SLEEP_COUNT = 0.05
def call_predict_endpoint(n):

    # 上传图像
    image = open(IMAGE_PATH, "rb").read()
    payload = {"image": image}
    # 提交请求
    r = requests.post(REST_API_URL, files=payload).json()
    # 确认请求是否成功
    if r["success"]:
        print("[INFO] thread {} OK".format(n))
    else:
        print("[INFO] thread {} FAILED".format(n))
# 多线程进行
for i in range(0, NUM_REQUESTS):
    # 创建线程来调用api
    t = Thread(target=call_predict_endpoint, args=(i,))
    t.daemon = True
    t.start()
    time.sleep(SLEEP_COUNT)
time.sleep(300)
```

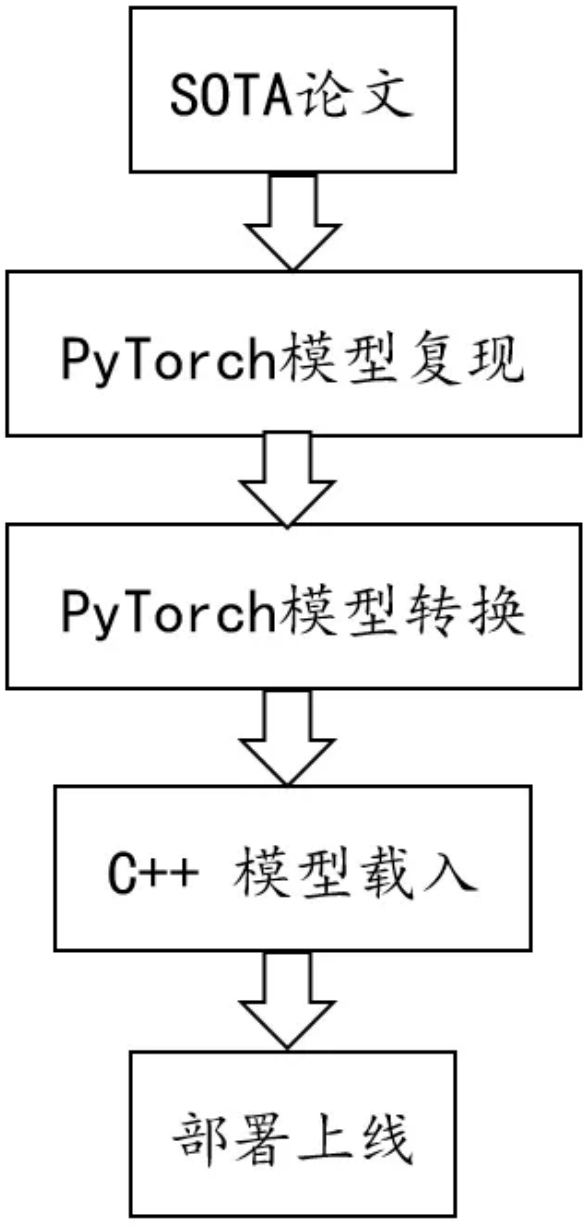
2. c++模型部署

教程:https://pytorch.apachecn.org/docs/1.2/beginner/Intro_to_TorchScript_tutorial.html

利用TorchScript进行模型c++部署,

业界与学术界最大的区别在于工业界的模型需要落地部署，学界更多的是关心模型的精度要求，而不太在意模型的部署性能。一般来说，我们用深度学习框架训练出一个模型之后，使用Python就足以实现一个简单的推理演示了。但在生产环境下，Python的可移植性和速度性能远不如C++。所以对于深度学习算法工程师而言，Python通常用来做idea的快速实现以及模型训练，而用C++作为模型的生产工具。目前PyTorch能够完美的将二者结合在一起。实现PyTorch模型部署的核心技术组件就是TorchScript和libtorch。

所以基于PyTorch的深度学习算法工程化流程大体如下图所示：



1. 安装libtorch

pytorch官网 下载libtorch

解压到指定的位置,我这里直接解压到`/home/xxx/`.

2. 将模型转换为torch脚本

定义一个python文件,载入模型文件pth,然后将其转换为torch脚本.

```
import torch


def load_checkpoint(filepath):
    checkpoint = torch.load(filepath, map_location='cpu')
    model = checkpoint['model']  # 提取网络结构
    model.load_state_dict(checkpoint['model_state_dict'])  # 加载网络权重参数
    for parameter in model.parameters():
        parameter.requires_grad = False
    model.eval()
    return model

model = load_checkpoint('./resnext101_32x8.pth')


# 这里如果保存采用gpu模型,就必须将example转换为cuda类型
example = torch.rand(1, 3, 224, 224)

# 转换为torch脚本
# 这里有两种方式,另一种方式为script,如果模型中存在if的分支结构,使用trace不行的,使用script
# 参考链接:
https://pytorch.apachecn.org/docs/1.2/beginner/Intro_to_TorchScript_tutorial.html
traced_script_module = torch.jit.trace(model, example)

# 测试转换是否正确
output = traced_script_module(torch.ones(1, 3, 224, 224))

print(output)
```

3. torch脚本序列化为文件

将上文转换完成的脚本序列化为pt模型文件

```
traced_script_module.save('./trace_resnext101_32x8.pt')
```

4. 在c++中调用模型

1. CMakeLists.txt

```
# 指定 cmake 最低编译版本
cmake_minimum_required(VERSION 3.0 FATAL_ERROR)

# 指定project的名称
project(c)

#添加需要的库
set(CMAKE_PREFIX_PATH
        /home/lxztju/libtorch_cpu
        /home/lxztju/opencv_3.4.3/build)
```

```
find_package(Torch REQUIRED)
find_package(OpenCV REQUIRED)

#添加可执行文件
add_executable(c main.cpp)

#外部库依赖
target_link_libraries(c ${TORCH_LIBRARIES} ${OpenCV_LIBS})

# 编译语言
set_property(TARGET c PROPERTY CXX_STANDARD 14)
```

opencv安装

由于使用了opencv这里记录opencv的安装(ubuntu)

下载地址:https://opencv.org/releases/

然后解压在指定文件夹下,这里解压在`/home/lxztju`下.

```
cd /home/lxztju/opencv-3.4.3
mkdir build
cd build
```

```
sudo cmake -D CMAKE_BUILD_TYPE=Release -D CMAKE_INSTALL_PREFIX=/usr/local ..
sudo make -j8
sudo make install
```

配置环境

```
sudo gedit /etc/ld.so.conf
# 添加一行 include /usr/loacal/lib
# 其中/usr/local是makefile中指定的路径

sudo gedit /etc/bash.bashrc
```

在末尾添加如下内容

```
PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
export PKG_CONFIG_PATH
```

```
source /etc/bash.bashrc
# 查看是否安装成功
pkg-config opencv --modversion
```

2. cpp文件

```cpp
//头文件
#include <torch/script.h>
#include <opencv2/opencv.hpp>

#include <iostream>
#include <memory>
#include <string>
#include <vector>

# include <ctime>
#include <dirent.h>


using namespace std;
//https://pytorch.org/tutorials/advanced/cpp_export.html

//存储测试图像的文件夹
string image_path ( "/home/lxztju/git/model_deployment/c/image");

//获取一个文件夹下的所有图像,存入files着发饿vector中.
void getFiles( string path, vector<string>& files )
{
    struct dirent *ptr;
    DIR *dir;
    dir = opendir(path.c_str());
    while ((ptr = readdir(dir)) != NULL)
    {
        //跳过'.'和'..'两个目录
        if(ptr->d_name[0] == '.')
            continue;
        files.push_back(ptr->d_name);
    }
}



int main(int argc, const char* argv[])
{
    //载入模型
    torch::jit::script::Module module =
torch::jit::load("/home/luxiangzhe/git/model_deployment/c/trace_resnext101_32x8.
pt");

    cout << "ok\n";


    vector<string> files;
    char * filePath = "/home/luxiangzhe/git/model_deployment/c/image";

////获取该路径下的所有文件
    getFiles(filePath, files );
    int size = files.size();
//    for (int i = 0;i < size;i++)
//    {
//        cout<<files[i]<<endl;
//    }

    clock_t start, end;
```

```
    double totle_time;


    for (int i = 0; i < files.size(); i++) {
        // 输入图像
        auto image = cv::imread(image_path + '/' + files[i],
cv::ImreadModes::IMREAD_COLOR);
        cv::Mat image_transformed;
        cv::resize(image, image_transformed, cv::Size(224, 224));
        cv::cvtColor(image_transformed, image_transformed, cv::COLOR_BGR2RGB);

        //图像转换为tensor
        torch::Tensor image_tensor = torch::from_blob(image_transformed.data,
                                                      {image_transformed.rows,
image_transformed.cols, 3},
                                                      torch::kByte);
        image_tensor = image_tensor.permute({2, 0, 1});
        image_tensor = image_tensor.toType(torch::kFloat);
        image_tensor = image_tensor.div(255);
        image_tensor = image_tensor.unsqueeze(0);
        //这里如果采用gpu版本的libtorch模型,需要将测试图像转换为cuda
        //image_tensor = image_tensor.to(at::kCUDA);
        //start = clock();
        //前向传播
        at::Tensor output = module.forward({image_tensor}).toTensor();
        //end = clock();
        //totle_time = (double)(end-start) /CLOCKS_PER_SEC;
        //cout << "totle time: " << totle_time <<endl;
        auto max_result = output.max(1, true);
        auto max_index = std::get<1>(max_result).item<float>();
        cout <<"label: " <<max_index<<endl;

    }

    return 0;
}
```

3. 编译链接

首先在c++ 项目中建立文件夹

```
mkdir build
cd build
```

编译链接

```
cmake ..
make
```

执行生成的可执行文件

```
./project_name
```

一个问题

这里我对比了pytorch与libtorch模型的推理速度,发现采用c++与libtorch的结合速度要慢很多,这个我看了github上的一些回答,也没有发现合理的解释,不知道怎么回事,可能采用c++并发会很大程度上加快整体的运行速度,也不清楚怎么回事.还希望看到的大佬能够解答一下.