MACE 调试指南

（由于官方文档叙述含糊，实际操作中问题百出，且官方并未指导如何在 C++工程中调用生成的网络，所以我撰写了这篇指南，从环境的准备到最终构建 C++工程，再到顺利调用运行 C++网络）

一、MACE 环境搭建

官方文档：https://mace.readthedocs.io/en/latest/

首先按照官方文档要求把依赖项都装好，转换 Tensorflow 网络模型需要装 Tensorflow，Docker 不是必须的（想用 docker 打包整个 mace 环境也可以，不过我用 docker 的时候环境没调好，编译 mace 的时候报错了）

Required dependencies

Software	Installation command	Tested version
Python		2.7 or 3.6
Bazel	bazel installation guide	0.13.0
CMake	Linux: `apt-get install cmake` Mac: `brew install cmake`	>= 3.11.3
Jinja2	pip install jinja2==2.10	2.10
PyYaml	pip install pyyaml==3.12	3.12.0
sh	pip install sh==1.12.14	1.12.14
Numpy	pip install numpy==1.14.0	Required by model validation
six	pip install six==1.11.0	Required for Python 2 and 3 compat

Optional dependencies

Software	Installation command
Android NDK	NDK installation guide
CMake	apt-get install cmake
ADB	Linux: `apt-get install android-tools-adb` Mac: `brew cask install android-platform-tools`
TensorFlow	pip install tensorflow==1.8.0
Docker	docker installation guide
Scipy	pip install scipy==1.0.0
FileLock	pip install filelock==3.0.0
ONNX	pip install onnx==1.3.0

然后从 Github 直接 clone MACE 工程到本地（如果使用 docker 的话需要按照 https://www.cnblogs.com/missidiot/p/9480033.html 的指南做，我默认不使用 docker），进入 MACE 目录下终端执行$bash tools/build-standalone-lib.sh，MACE 框架即可编译完成。

二、网络模型的转换

（1）首先，你要有网络模型文件和网络参数文件（Tensorflow 生成的分别是.pb 和.data 文件，之后库的调用也要用到，所以必须有），然后创建一个.yml 文件，内容如下（本部分更多详细说明参见官方文档）：

```yaml
# The name of library
library_name: mobile_squeeze
# host, armeabi-v7a or arm64-v8a
target_abis: [arm64-v8a]
# The build mode for model(s).
# 'code' for transferring model(s) into cpp code, 'file' for keeping model(s) in protobuf file(
model_graph_format: code
# 'code' for transferring model data(s) into cpp code, 'file' for keeping model data(s) in file
model_data_format: code
# One yaml config file can contain multi models' deployment info.
models:
  mobilenet_v1:
      platform: tensorflow
      model_file_path: https://cnbj1.fds.api.xiaomi.com/mace/miai-models/mobilenet-v1/mobilenet
      model_sha256_checksum: 71b10f540ece33c49a7b51f5d4095fc9bd78ce46ebf0300487b2ee23d71294e6
      subgraphs:
        - input_tensors:
            - input
          input_shapes:
            - 1,224,224,3
          output_tensors:
            - MobilenetV1/Predictions/Reshape_1
          output_shapes:
            - 1,1001
          validation_inputs_data:
            - https://cnbj1.fds.api.xiaomi.com/mace/inputs/dog.npy
      runtime: cpu+gpu
      limit_opencl_kernel_time: 0
      obfuscate: 0
      winograd: 0
```

（2）然后你需要将.yml 文件中 model_graph_format 和 model_data_format 都改为 code，之后进入 MACE 目录，终端输入命令$ python tools/converter.py convert --config=${PATH_TO_YML}，${PATH_TO_YML}是你的.yml 文件的路径，生成的文件位于 mace/build/{YOUR_MODEL_NAME}/中。

（3）如果你本身没有网络模型文件和网络参数文件，那可以下载官方的文件，Github 上的 mace_models 项目中存放了已经写好的.yml 文件，把其中 model_graph_format 和 model_data_format 都改为 file，然后终端运行命令$ python

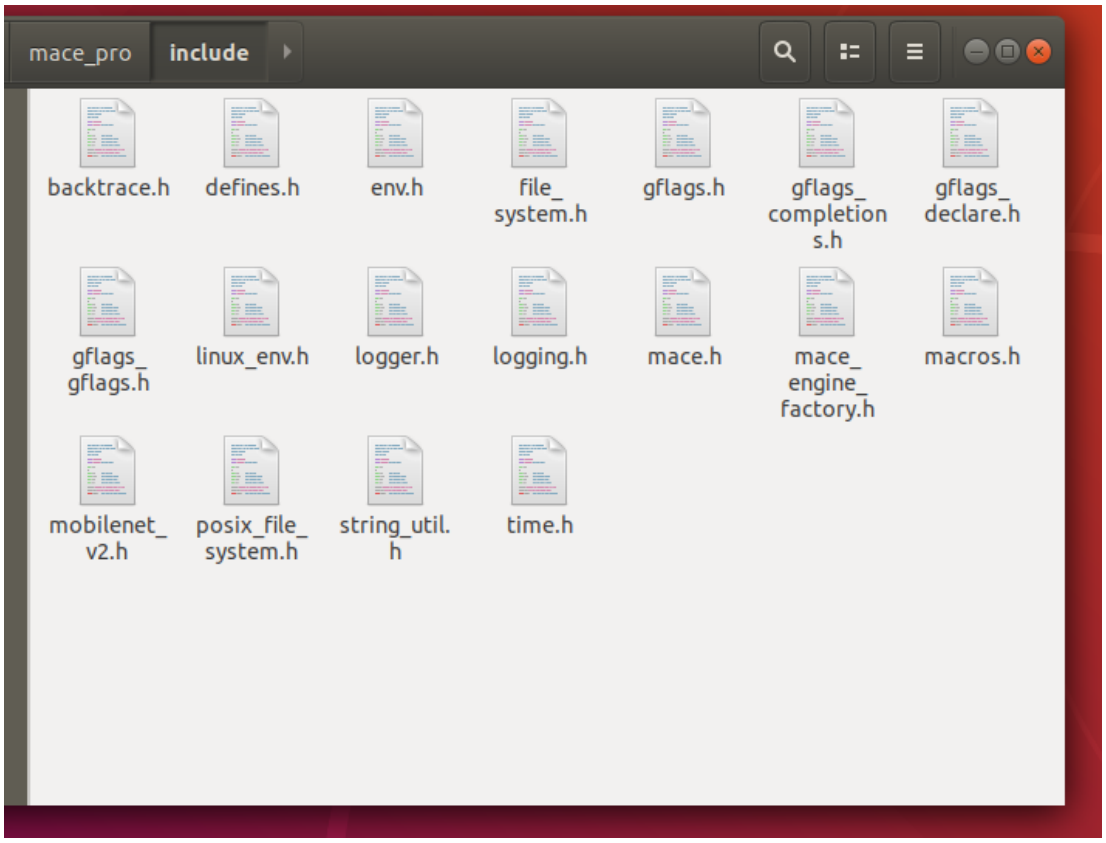
tools/converter.py convert --config=${PATH_TO_YML}即可生成网络模型文件和网络参数文件，将这两个文件移至别处，再重复（2）操作。

三、C++工程的构建

至此，我们已经生成了网络模型的 C++静态库和 MACE 核心库，也生成了对应的头文件，它们都存放于 MACE 目录下的 build 文件夹中。为了演示如何使用 C++网络，我以 mobilenet-v2 网络为例，以 mace/mace/examples/cli/example.cc 这个主文件为例，这也是官方提供的唯一的 C++示例主文件。这个源文件中赘述了太多没有用的东西，也缺少了一些东西的定义，在了解如何调用库之后可以不用它这个，自己重新编写主文件。

首先为了方便和整洁，我把构成 C++工程的所有必要文件都从 MACE 目录中提取出来，然后新建了文件夹重新调整结构如下：

```
Mace_pro
├── example.cc
├── env.cc
├── include（头文件，具体包含内容见图片）
│   ├── mace.h
│   ├── . . .
│   └── mace_engine_factory.h
├── libs（库文件）
│   ├── linux-x86-64
│   ├── (arm_linux_gnueabihf)
│   ├── (arm64-v8a)
│   ├── (armeabi-v7a)
│   ├── (aarch64_linux_gnu)
│   ├── netmodels
│   │   └──mobilenet-v2
│   │       ├── host
│   │       ├── (armeabi-v7a)
│   │       └── (arm64-v8a)
│   ├── libgflags.a
│   └── libgflags_nothreads.a
└── mobilenet（内放.pb 和.data）
    ├── mobilenet_v2.data
    └── mobilenet_v2.pb
```



说明：

example.cc 为主文件;

env.cc 文件位于 mace/mace/port 目录下；

头文件包括三部分：**所有 mace/build 目录下的头文件，example.cc 和 env.cc 中 include 的头文件及这些头文件依赖的头文件**（有 2 个头文件有重名文件，所以需要改个名：mace/mace/port/posix/file_system.h 改为 posix_file_system.h，mace/mace/port/linux/env.h 改为 linux_env.h，对应在各文件 include 这两个文件的时候也需要改一下名），自行下载编译**第三方库 gflags 生成的头文件**；

库文件包括三部分：**MACE 核心库**（红字标出，只需选择对应运行平台的架构即可，原始位置位于 mace/build 下），**网络模型的 C++静态库**（蓝字标出，只需选择对应运行平台的架构即可，原始位置位于 mace/build/mobilenet-v2 下），自行下载编译**第三方库 gflags 生成的库文件**（紫字标出）；

mobilenet 文件夹中存放网络模型文件和网络参数文件。

至此，一个完整有序的 C++工程就构建起来了，下面还需要对主文件

example.cc 进行修改。

四、主文件的修改

example.cc 本身十分混乱，不能编译，且加入了很多不必要的东西，比如 gflags。因此，我们需要对其进行整理，我修改后的 example.cc 文件可以前往 https://github.com/tk42635/MACE-Pro.git 下载，各处修改都已注释标明。

五、工程的编译

现在我们只需要把库文件和源文件链接起来编译即可，由于 example 中调用了 openmp，我还调用了 opencv 用来读数据，所以这两个库也要链接上。用 g++ 编译的终端命令如下（也可以使用 CMake 来编译）：

```
$g++ -fopenmp -o output env.cc example.cc `pkg-config --cflags --libs opencv` libs/net_models/mobilenet-v2/host/mobilenet-v2.a libs/linux-x86-64/libmace.a libs/libgflags.a libs/libgflags_nothreads.a
```

另外如果想移植 ARM 上的话所有的第三方库都需要在 ARM 上编译生成，最后编译 C++工程也需要在 ARM 上完成（交叉编译太麻烦），可能还需要链接-ldl 库和第三方 protobuf 库，protobuf 的编译参考 Github 官方指南，使用的版本必须和 mace.pb.h 和 mace.pb.cc 中要求的版本相同（目前为 3.6.1），这两个文件是生成 C++网络时自动生成的，存放于寄存器.cache 中，可以用 VScode 打开 mace/mace/codegen 中的 model.cc 文件，里面 include 了 mace.pb.h，这样就可以跳转过去了查看了。要注意的是在用$./configure 命令编译时需要在后面加上-D_GLIBCXX_USE_CXX11_ABI=0，即：

```
$./configure -D_GLIBCXX_USE_CXX11_ABI=0
```

不然的话会在最后编译 C++工程时，链接 protobuf 库报错 fixed_address_empty_string 未定义的引用。