MACE

Jan 07, 2019

Introduction

Welcome to Mobile AI Compute Engine documentation.

The main documentation is organized into the following sections:

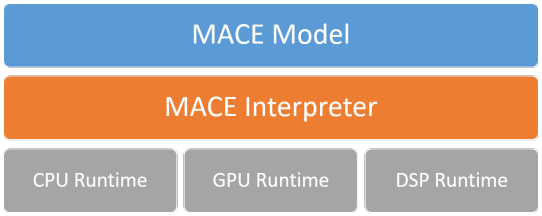
CHAPTER 1

Introduction

MACE (Mobile AI Compute Engine) is a deep learning inference framework optimized for mobile heterogeneous computing platforms. MACE provides tools and documents to help users to deploy deep learning models to mobile phones, tablets, personal computers and IoT devices.

1.1 Architecture

The following figure shows the overall architecture.



1.1.1 MACE Model

MACE defines a customized model format which is similar to Caffe2. The MACE model can be converted from exported models by TensorFlow, Caffe or ONNX Model.

1.1.2 MACE Interpreter

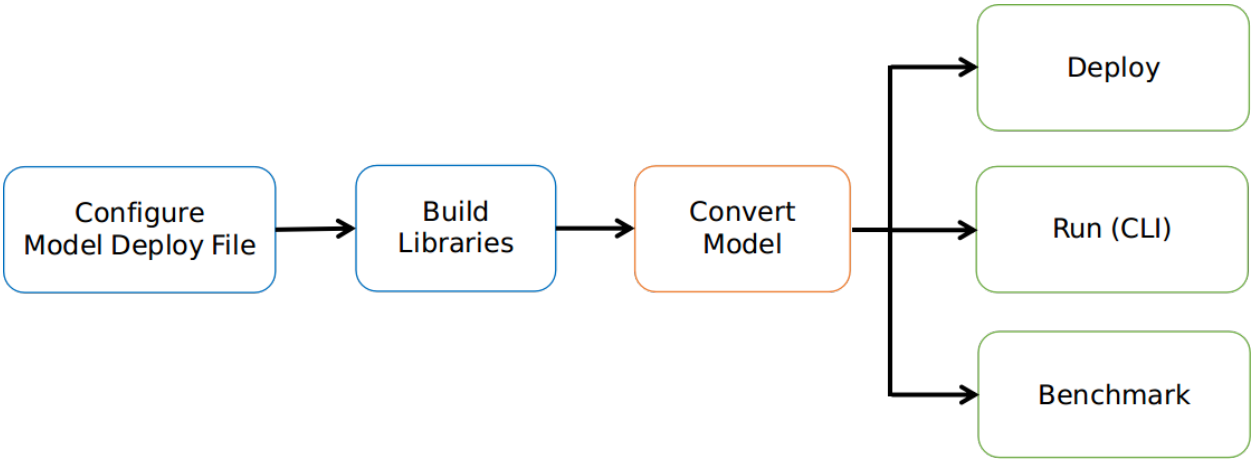
Mace Interpreter mainly parses the NN graph and manages the tensors in the graph.

1.1.3 Runtime

CPU/GPU/DSP runtime correspond to the Ops for different devices.

1.2 Workflow

The following figure shows the basic work flow of MACE.



1.2.1 1. Configure model deployment file

Model deploy configuration file (.yml) describes the information of the model and library, MACE will build the library based on the file.

1.2.2 2. Build libraries

Build MACE dynamic or static libraries.

1.2.3 3. Convert model

Convert TensorFlow, Caffe or ONNX model to MACE model.

1.2.4 4.1. Deploy

Integrate the MACE library into your application and run with MACE API.

1.2.5 4.2. Run (CLI)

MACE provides *mace_run* command line tool, which could be used to run model and validate model correctness against original TensorFlow or Caffe results.

1.2.6 4.3. Benchmark

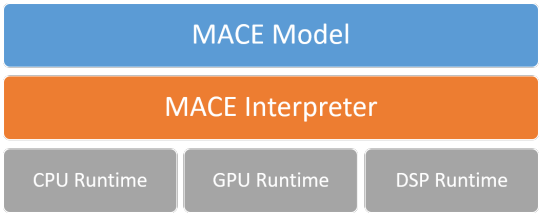
MACE provides benchmark tool to get the Op level profiling result of the model.

1.3 简介

Mobile AI Compute Engine (MACE) 是一个专为移动端异构计算设备优化的深度学习前向预测框架。 MACE覆盖了常见的移动端计算设备（CPU，GPU和DSP），并且提供了完整的工具链和文档，用户借助MACE能够很方便地在移动端部署深度学习模型。MACE已经在小米内部广泛使用并且被充分验证具有业界领先的性能和稳定性。

1.4 框架

下图描述了MACE的基本框架。



1.4.1 MACE Model

MACE定义了自有的模型格式（类似于Caffe2），通过MACE提供的工具可以将Caffe/TensorFlow/ONNX格式的模型 转为MACE模型。

1.4.2 MACE Interpreter

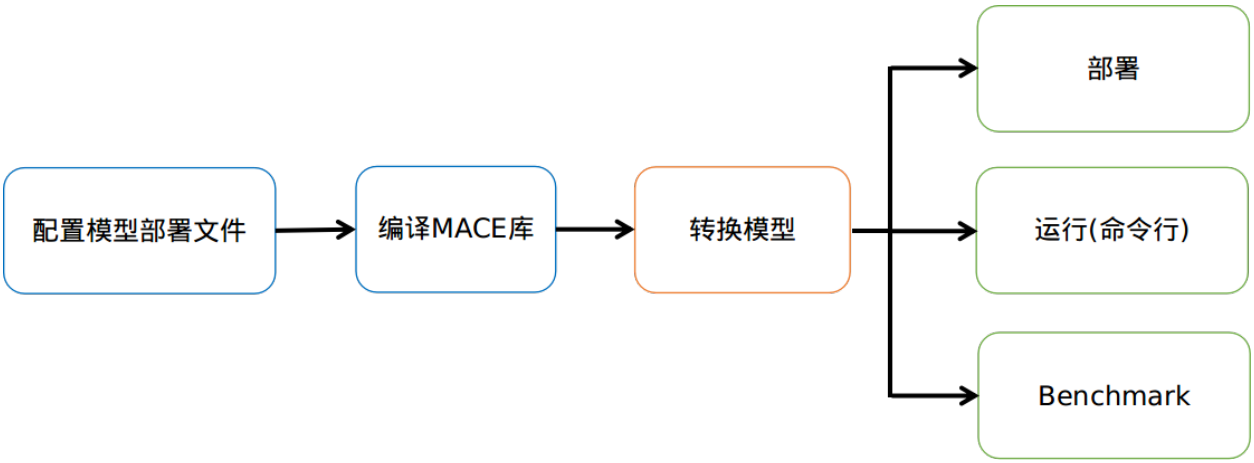
MACE Interpreter主要负责解析运行神经网络图（DAG）并管理网络中的Tensors。

1.4.3 Runtime

CPU/GPU/DSP Runtime对应于各个计算设备的算子实现。

1.5 使用流程

下图描述了MACE使用的基本流程。



1.5.1 1. 配置模型部署文件(.yml)

模型部署文件详细描述了需要部署的模型以及生成库的信息，MACE根据该文件最终生成对应的库文件。

1.5.2 2. 编译MACE库

编译MACE的静态库或者动态库。

1.5.3 3. 转换模型

将TensorFlow或者Caffe或者ONNX的模型转为MACE的模型。

1.5.4 4.1. 部署

根据不同使用目的集成Build阶段生成的库文件，然后调用MACE相应的接口执行模型。

1.5.5 4.2. 命令行运行

MACE提供了命令行工具，可以在命令行运行模型，可以用来测试模型运行时间，内存占用和正确性。

1.5.6 4.3. Benchmark

MACE提供了命令行benchmark工具，可以细粒度的查看模型中所涉及的所有算子的运行时间。

CHAPTER 2

Environment requirement

MACE requires the following dependencies:

2.1 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 compatibility

For Bazel, install it following installation guide. For python dependencies,

```
pip install -U --user setup/requirements.txt
```

2.2 Optional dependencies

Software	Installation command	Remark
Android NDK	NDK installation guide	Required by Android build, r15b, r15c, r16b, r17b
CMake	apt-get install cmake	>= 3.11.3
ADB	Linux:`apt-get install android-tools-adb` Mac:`brew cask install android-platform-tools`	Required by Android run, >= 1.0.32
TensorFlow	pip install tensorflow==1.8.0	Required by TensorFlow model
Docker	docker installation guide	Required by docker mode for Caffe model
Scipy	pip install scipy==1.0.0	Required by model validation
FileLock	pip install filelock==3.0.0	Required by run on Android
ONNX	pip install onnx==1.3.0	Required by ONNX model

For python dependencies,

```
pip install -U --user setup/optionals.txt
```

Note:

- For Android build, *ANDROID_NDK_HOME* must be confifigured by using `export ANDROID_NDK_HOME=/path/to/ndk`
- It will link `libc++` instead of `gnustl` if `NDK version >= r17b` and `bazel version >= 0.13.0`, please refer to NDK cpp-support.
- For Mac, please install Homebrew at first before installing other dependencies. Set ANDROID_NDK_HOME in `/etc/bashrc` and then run `source /etc/bashrc`. This installation was tested with macOS Mojave(10.14).

CHAPTER 3

Using docker

3.1 Pull or build docker image

MACE provides docker images with dependencies installed and also Dockerfiles for images building, you can pull the existing ones directly or build them from the Dockerfiles. In most cases, the `lite edition` image can satisfy developer's basic needs.

Note: It's highly recommended to pull built images.

- `lite edition` docker image.

```
# Pull lite edition docker image
docker pull registry.cn-hangzhou.aliyuncs.com/xiaomimace/mace-dev-lite
# Build lite edition docker image
docker build -t registry.cn-hangzhou.aliyuncs.com/xiaomimace/mace-dev-lite ./docker/
↪mace-dev-lite
```

- `full edition` docker image (which contains multiple NDK versions and other dev tools).

```
# Pull full edition docker image
docker pull registry.cn-hangzhou.aliyuncs.com/xiaomimace/mace-dev
# Build full edition docker image
docker build -t registry.cn-hangzhou.aliyuncs.com/xiaomimace/mace-dev ./docker/mace-
↪dev
```

Note: We will show steps with lite edition later.

3.2 Using the image

Create container with the following command

```
# Create a container named `mace-dev`
docker run -it --privileged -d --name mace-dev \
           -v /dev/bus/usb:/dev/bus/usb --net=host \
           -v /local/path:/container/path \
           -v /usr/bin/docker:/usr/bin/docker \
           -v /var/run/docker.sock:/var/run/docker.sock \
           registry.cn-hangzhou.aliyuncs.com/xiaomimace/mace-dev-lite
# Execute an interactive bash shell on the container
docker exec -it mace-dev /bin/bash
```

CHAPTER 4

Manual setup

The setup steps are based on `Ubuntu`, you can change the commands correspondingly for other systems. For the detailed installation dependencies, please refer to *Environment requirement*.

4.1 Install Bazel

Recommend bazel with version larger than `0.13.0` (Refer to Bazel documentation).

```
export BAZEL_VERSION=0.13.1
mkdir /bazel && \
    cd /bazel && \
    wget https://github.com/bazelbuild/bazel/releases/download/$BAZEL_VERSION/bazel-
↪$BAZEL_VERSION-installer-linux-x86_64.sh && \
    chmod +x bazel-*.sh && \
    ./bazel-$BAZEL_VERSION-installer-linux-x86_64.sh && \
    cd / && \
    rm -f /bazel/bazel-$BAZEL_VERSION-installer-linux-x86_64.sh
```

4.2 Install Android NDK

The recommended Android NDK versions includes r15b, r15c and r16b (Refers to NDK installation guide).

```
# Download NDK r15c
cd /opt/ && \
    wget -q https://dl.google.com/android/repository/android-ndk-r15c-linux-x86_64.
↪zip && \
    unzip -q android-ndk-r15c-linux-x86_64.zip && \
    rm -f android-ndk-r15c-linux-x86_64.zip

export ANDROID_NDK_VERSION=r15c
```

(continues on next page)

(continued from previous page)

```
export ANDROID_NDK=/opt/android-ndk-${ANDROID_NDK_VERSION}
export ANDROID_NDK_HOME=${ANDROID_NDK}

# add to PATH
export PATH=${PATH}:${ANDROID_NDK_HOME}
```

4.3 Install extra tools

```
apt-get install -y --no-install-recommends \
    cmake \
    android-tools-adb
pip install -i http://pypi.douban.com/simple/ --trusted-host pypi.douban.com␣
↪setuptools
pip install -i http://pypi.douban.com/simple/ --trusted-host pypi.douban.com \
    "numpy>=1.14.0" \
    scipy \
    jinja2 \
    pyyaml \
    sh==1.12.14 \
    pycodestyle==2.4.0 \
    filelock
```

4.4 Install TensorFlow (Optional)

```
pip install -i http://pypi.douban.com/simple/ --trusted-host pypi.douban.com␣
↪tensorflow==1.8.0
```

4.5 Install Caffe (Optional)

Please follow the installation instruction of Caffe.

4.6 Install ONNX (Optional)

Please follow the installation instruction of ONNX.

CHAPTER 5

Basic usage

5.1 Build and run an example model

At first, make sure the environment has been set up correctly already (refer to *Environment requirement*).

The followings are instructions about how to quickly build and run a provided model in MACE Model Zoo.

Here we use the mobilenet-v2 model as an example.

Commands

1. Pull MACE project.

```
git clone https://github.com/XiaoMi/mace.git
cd mace/
git fetch --all --tags --prune

# Checkout the latest tag (i.e. release version)
tag_name=`git describe --abbrev=0 --tags`
git checkout tags/${tag_name}
```

Note: It's highly recommended to use a release version instead of master branch.

2. Pull MACE Model Zoo project.

```
git clone https://github.com/XiaoMi/mace-models.git
```

3. Build a generic MACE library.

```
cd path/to/mace
# Build library
# output lib path: builds/lib
bash tools/build-standalone-lib.sh
```

Note:

- Libraries in `builds/lib/armeabi-v7a/cpu_gpu/` means it can run on `cpu` or `gpu` devices.
- The results in `builds/lib/armeabi-v7a/cpu_gpu_dsp/` need HVX supported.

4. Convert the pre-trained mobilenet-v2 model to MACE format model.

```
cd path/to/mace
# Build library
python tools/converter.py convert --config=/path/to/mace-models/mobilenet-v2/
↪mobilenet-v2.yml
```

5. Run the model.

Note: If you want to run on phone, please plug in at least one phone. Or if you want to run on embedded device, please give a *Advanced usage*.

```
# Run example
python tools/converter.py run --config=/path/to/mace-models/mobilenet-v2/
↪mobilenet-v2.yml --example

# Test model run time
python tools/converter.py run --config=/path/to/mace-models/mobilenet-v2/
↪mobilenet-v2.yml --round=100

# Validate the correctness by comparing the results against the
# original model and framework, measured with cosine distance for similarity.
python tools/converter.py run --config=/path/to/mace-models/mobilenet-v2/
↪mobilenet-v2.yml --validate
```

5.2 Build your own model

This part will show you how to use your own pre-trained model in MACE.

5.2.1 1. Prepare your model

MACE now supports models from TensorFlow and Caffe (more frameworks will be supported).

- TensorFlow

  Prepare your pre-trained TensorFlow model.pb file.

- Caffe

  Caffe 1.0+ models are supported in MACE converter tool.

  If your model is from lower version Caffe, you need to upgrade it by using the Caffe built-in tool before converting.

```
# Upgrade prototxt
$CAFFE_ROOT/build/tools/upgrade_net_proto_text MODEL.prototxt MODEL.new.prototxt
```

(continues on next page)

(continued from previous page)

```
# Upgrade caffemodel
$CAFFE_ROOT/build/tools/upgrade_net_proto_binary MODEL.caffemodel MODEL.new.
↪caffemodel
```

- ONNX

  Prepare your ONNX model.onnx file.

  Use ONNX Optimizer Tool to optimize your model for inference. This tool will improve the efficiency of inference like the Graph Transform Tool in TensorFlow.

```
# Optimize your model
$python MACE_ROOT/tools/onnx_optimizer.py model.onnx model_opt.onnx
```

5.2.2 2. Create a deployment file for your model

When converting a model or building a library, MACE needs to read a YAML file which is called model deployment file here.

A model deployment file contains all the information of your model(s) and building options. There are several example deployment files in *MACE Model Zoo* project.

The following shows two basic usage of deployment files for TensorFlow and Caffe models. Modify one of them and use it for your own case.

- TensorFlow

```
# The name of library
library_name: mobilenet
target_abis: [arm64-v8a]
model_graph_format: file
model_data_format: file
models:
  mobilenet_v1: # model tag, which will be used in model loading and must be␣
↪specific.
    platform: tensorflow
    # path to your tensorflow model's pb file. Support local path, http:// and␣
↪https://
    model_file_path: https://cnbj1.fds.api.xiaomi.com/mace/miai-models/mobilenet-
↪v1/mobilenet-v1-1.0.pb
    # sha256_checksum of your model's pb file.
    # use this command to get the sha256_checksum: sha256sum path/to/your/pb/file
    model_sha256_checksum:␣
↪71b10f540ece33c49a7b51f5d4095fc9bd78ce46ebf0300487b2ee23d71294e6
    # define your model's interface
    # if there multiple inputs or outputs, write like blow:
    # subgraphs:
    # - input_tensors:
    #     - input0
    #     - input1
    #   input_shapes:
    #     - 1,224,224,3
    #     - 1,224,224,3
    #   output_tensors:
    #     - output0
    #     - output1
```

(continues on next page)

(continued from previous page)

```
    #    output_shapes:
    #      - 1,1001
    #      - 1,1001
    subgraphs:
      - input_tensors:
          - input
        input_shapes:
          - 1,224,224,3
        output_tensors:
          - MobilenetV1/Predictions/Reshape_1
        output_shapes:
          - 1,1001
    # cpu, gpu or cpu+gpu
    runtime: cpu+gpu
    winograd: 0
```

- Caffe

```
# The name of library
library_name: squeezenet-v10
target_abis: [arm64-v8a]
model_graph_format: file
model_data_format: file
models:
  squeezenet-v10: # model tag, which will be used in model loading and must be␣
↪specific.
    platform: caffe
    # support local path, http:// and https://
    model_file_path: https://cnbj1.fds.api.xiaomi.com/mace/miai-models/squeezenet/
↪squeezenet-v1.0.prototxt
    weight_file_path: https://cnbj1.fds.api.xiaomi.com/mace/miai-models/
↪squeezenet/squeezenet-v1.0.caffemodel
    # sha256_checksum of your model's graph and data files.
    # get the sha256_checksum: sha256sum path/to/your/file
    model_sha256_checksum:␣
↪db680cf18bb0387ded9c8e9401b1bbcf5dc09bf704ef1e3d3dbd1937e772cae0
    weight_sha256_checksum:␣
↪9ff8035aada1f9ffa880b35252680d971434b141ec9fbacbe88309f0f9a675ce
    # define your model's interface
    # if there multiple inputs or outputs, write like blow:
    # subgraphs:
    # - input_tensors:
    #     - input0
    #     - input1
    #   input_shapes:
    #     - 1,224,224,3
    #     - 1,224,224,3
    #    output_tensors:
    #      - output0
    #      - output1
    #    output_shapes:
    #      - 1,1001
    #      - 1,1001
    subgraphs:
      - input_tensors:
          - data
        input_shapes:
```

(continues on next page)

(continued from previous page)

```
        - 1,227,227,3
      output_tensors:
        - prob
      output_shapes:
        - 1,1,1,1000
  runtime: cpu+gpu
  winograd: 0
```

- ONNX

```
# The name of library
library_name: mobilenet
target_abis: [arm64-v8a]
model_graph_format: file
model_data_format: file
models:
  mobilenet_v1: # model tag, which will be used in model loading and must be␣
↪specific.
    platform: onnx
    # path to your onnx model file. Support local path, http:// and https://
    model_file_path: https://cnbj1.fds.api.xiaomi.com/mace/miai-models/mobilenet-
↪v1/mobilenet-v1-1.0.pb
    # sha256_checksum of your model's onnx file.
    # use this command to get the sha256_checksum: sha256sum path/to/your/pb/file
    model_sha256_checksum:␣
↪71b10f540ece33c49a7b51f5d4095fc9bd78ce46ebf0300487b2ee23d71294e6
    # define your model's interface
    # if there multiple inputs or outputs, write like blow:
    # subgraphs:
    # - input_tensors:
    #     - input0
    #     - input1
    #   input_shapes:
    #     - 1,224,224,3
    #     - 1,224,224,3
    #    output_tensors:
    #      - output0
    #      - output1
    #    output_shapes:
    #      - 1,1001
    #      - 1,1001
    subgraphs:
      - input_tensors:
          - input
        input_shapes:
          - 1,224,224,3
        output_tensors:
          - MobilenetV1/Predictions/Reshape_1
        output_shapes:
          - 1,1001
        # onnx backend framwork for validation. Suppport pytorch/caffe/tensorflow.
↪ Default is tensorflow.
        backend: tensorflow
    # cpu, gpu or cpu+gpu
    runtime: cpu+gpu
    winograd: 0
```

More details about model deployment file are in *Advanced usage*.

5.2.3 3. Convert your model

When the deployment file is ready, you can use MACE converter tool to convert your model(s).

```
python tools/converter.py convert --config=/path/to/your/model_deployment_file.yml
```

This command will download or load your pre-trained model and convert it to a MACE model proto file and weights data file. The generated model files will be stored in `builds/${library_name}/model` folder.

Warning: Please set `model_graph_format: file` and `model_data_format: file` in your deployment file before converting. The usage of `model_graph_format: code` will be demonstrated in *Advanced usage*.

5.2.4 4. Build MACE into a library

You could Download the prebuilt MACE Library from Github MACE release page.

Or use bazel to build MACE source code into a library.

```
cd path/to/mace
# Build library
# output lib path: builds/lib
bash tools/build-standalone-lib.sh
```

The above command will generate dynamic library `builds/lib/${ABI}/${DEVICES}/libmace.so` and static library `builds/lib/${ABI}/${DEVICES}/libmace.a`.

Warning: Please verify that the target_abis param in the above command and your deployment file are the same.

5.2.5 5. Run your model

With the converted model, the static or shared library and header files, you can use the following commands to run and validate your model.

Warning: If you want to run on device/phone, please plug in at least one device/phone.

- **run**

  run the model.

```
# Test model run time
python tools/converter.py run --config=/path/to/your/model_deployment_
↪file.yml --round=100

# Validate the correctness by comparing the results against the
# original model and framework, measured with cosine distance for␣
↪similarity.
```

(continues on next page)

(continued from previous page)

```
python tools/converter.py run --config=/path/to/your/model_deployment_
↪file.yml --validate

# If you want to run model on specified arm linux device, you should put␣
↪device config file in the working directory or run with flag `--device_
↪yml`
python tools/converter.py run --config=/path/to/your/model_deployment_
↪file.yml --device_yml=/path/to/devices.yml
```

- **benchmark**

  benchmark and profile the model. the details are in *Benchmark usage*.

```
# Benchmark model, get detailed statistics of each Op.
python tools/converter.py benchmark --config=/path/to/your/model_
↪deployment_file.yml
```

5.2.6 6. Deploy your model into applications

You could run model on CPU, GPU and DSP (based on the *runtime* in your model deployment file). However, there are some differences in different devices.

- **CPU**

  Almost all of mobile SoCs use ARM-based CPU architecture, so your model could run on different SoCs in theory.

- **GPU**

  Although most GPUs use OpenCL standard, but there are some SoCs not fully complying with the standard, or the GPU is too low-level to use. So you should have some fallback strategies when the GPU run failed.

- **DSP**

  MACE only supports Qualcomm DSP. And you need to push the hexagon nn library to the device.

```
# For Android device
adb root; adb remount
adb push third_party/nnlib/v6x/libhexagon_nn_skel.so /system/vendor/lib/
↪rfsa/adsp/
```

In the converting and building steps, you've got the static/shared library, model files and header files.

`${library_name}` is the name you defined in the first line of your deployment YAML file.

Note: When linking generated `libmace.a` into shared library, version script is helpful for reducing a specified set of symbols to local scope.

- The generated `static` library files are organized as follows,

```
builds
├── include
│   └── mace
│       └── public
│           └── mace.h
```

(continues on next page)

(continued from previous page)

```
├── lib
│   ├── arm64-v8a
│   │   └── cpu_gpu
│   │       ├── libmace.a
│   │       └── libmace.so
│   ├── armeabi-v7a
│   │   ├── cpu_gpu
│   │   │   ├── libmace.a
│   │   │   └── libmace.so
│   │   └── cpu_gpu_dsp
│   │       ├── libhexagon_controller.so
│   │       ├── libmace.a
│   │       └── libmace.so
│   └── linux-x86-64
│       ├── libmace.a
│       └── libmace.so
└── mobilenet-v1
    ├── model
    │   ├── mobilenet_v1.data
    │   └── mobilenet_v1.pb
    └── _tmp
        └── arm64-v8a
            └── mace_run_static
```

Please refer to `mace/examples/example.cc`for full usage. The following list the key steps.

```
// Include the headers
#include "mace/public/mace.h"

// 0. Declare the device type (must be same with ``runtime`` in configuration file)
DeviceType device_type = DeviceType::GPU;

// 1. configuration
MaceStatus status;
MaceEngineConfig config(device_type);
std::shared_ptr<GPUContext> gpu_context;
// Set the path to store compiled OpenCL kernel binaries.
// please make sure your application have read/write rights of the directory.
// this is used to reduce the initialization time since the compiling is too slow.
// It's suggested to set this even when pre-compiled OpenCL program file is provided
// because the OpenCL version upgrade may also leads to kernel recompilations.
const std::string storage_path ="path/to/storage";
gpu_context = GPUContextBuilder()
    .SetStoragePath(storage_path)
    .Finalize();
config.SetGPUContext(gpu_context);
config.SetGPUHints(
    static_cast<GPUPerfHint>(GPUPerfHint::PERF_NORMAL),
    static_cast<GPUPriorityHint>(GPUPriorityHint::PRIORITY_LOW));

// 2. Define the input and output tensor names.
std::vector<std::string> input_names = {...};
std::vector<std::string> output_names = {...};

// 3. Create MaceEngine instance
std::shared_ptr<mace::MaceEngine> engine;
MaceStatus create_engine_status;
```

(continues on next page)

(continued from previous page)

```
// Create Engine from model file
create_engine_status =
    CreateMaceEngineFromProto(model_pb_data,
                              model_data_file.c_str(),
                              input_names,
                              output_names,
                              device_type,
                              &engine);
if (create_engine_status != MaceStatus::MACE_SUCCESS) {
  // fall back to other strategy.
}

// 4. Create Input and Output tensor buffers
std::map<std::string, mace::MaceTensor> inputs;
std::map<std::string, mace::MaceTensor> outputs;
for (size_t i = 0; i < input_count; ++i) {
  // Allocate input and output
  int64_t input_size =
      std::accumulate(input_shapes[i].begin(), input_shapes[i].end(), 1,
                      std::multiplies<int64_t>());
  auto buffer_in = std::shared_ptr<float>(new float[input_size],
                                          std::default_delete<float[]>());
  // Load input here
  // ...

  inputs[input_names[i]] = mace::MaceTensor(input_shapes[i], buffer_in);
}

for (size_t i = 0; i < output_count; ++i) {
  int64_t output_size =
      std::accumulate(output_shapes[i].begin(), output_shapes[i].end(), 1,
                      std::multiplies<int64_t>());
  auto buffer_out = std::shared_ptr<float>(new float[output_size],
                                           std::default_delete<float[]>());
  outputs[output_names[i]] = mace::MaceTensor(output_shapes[i], buffer_out);
}

// 5. Run the model
MaceStatus status = engine.Run(inputs, &outputs);
```

More details are in *Advanced usage*.

CHAPTER 6

Advanced usage

This part contains the full usage of MACE.

6.1 Overview

As mentioned in the previous part, a model deployment file defines a case of model deployment. The building process includes parsing model deployment file, converting models, building MACE core library and packing generated model libraries.

6.2 Deployment file

One deployment file will generate one library normally, but if more than one ABIs are specified, one library will be generated for each ABI. A deployment file can also contain multiple models. For example, an AI camera application may contain face recognition, object recognition, and voice recognition models, all of which can be defined in one deployment file.

- **Example**

  Here is an example deployment file with two models.

```
# 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(s) (.pb).
model_graph_format: code
# 'code' for transferring model data(s) into cpp code, 'file' for keeping␣
↪model data(s) in file(s) (.data).
model_data_format: code
```

(continues on next page)

(continued from previous page)

```
# 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-v1-1.0.pb
      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
  squeezenet_v11:
      platform: caffe
      model_file_path: http://cnbj1-inner-fds.api.xiaomi.net/mace/mace-
↪models/squeezenet/SqueezeNet_v1.1/model.prototxt
      weight_file_path: http://cnbj1-inner-fds.api.xiaomi.net/mace/mace-
↪models/squeezenet/SqueezeNet_v1.1/weight.caffemodel
      model_sha256_checksum:␣
↪625c952063da1569e22d2f499dc454952244d42cd8feca61f05502566e70ae1c
      weight_sha256_checksum:␣
↪72b912ace512e8621f8ff168a7d72af55910d3c7c9445af8dfbff4c2ee960142
      subgraphs:
        - input_tensors:
            - data
          input_shapes:
            - 1,227,227,3
          output_tensors:
            - prob
          output_shapes:
            - 1,1,1,1000
      runtime: cpu+gpu
      limit_opencl_kernel_time: 0
      obfuscate: 0
      winograd: 0
```

- **Configurations**

Options	Usage
library_name	Library name.
target_abis	The target ABI(s) to build, could be 'host', 'armeabi-v7a' or 'arm64-v8a'. If more than one ABIs will be used, separate them by commas.
target_socs	[optional] Build for specific SoCs.
model_graph_format	model graph format, could be 'file' or 'code'. 'file' for converting model graph to ProtoBuf file(.pb) and 'code' for converting model graph to c++ code.
model_data_format	model data format, could be 'file' or 'code'. 'file' for converting model weight to data file(.data) and 'code' for converting model weight to c++ code.
model_name	model name should be unique if there are more than one models. **LIMIT: if build_type is code, model_name will be used in c++ code so that model_name must comply with c++ name specification.**
platform	The source framework, tensorflow or caffe.
model_file_path	The path of your model file which can be local path or remote URL.
model_sha256_checksum	The SHA256 checksum of the model file.
weight_file_path	[optional] The path of Caffe model weights file.
weight_sha256_checksum	[optional] The SHA256 checksum of Caffe model weights file.
subgraphs	subgraphs key. **DO NOT EDIT**
input_tensors	The input tensor name(s) (tensorflow) or top name(s) of inputs' layer (caffe). If there are more than one tensors, use one line for a tensor.
output_tensors	The output tensor name(s) (tensorflow) or top name(s) of outputs' layer (caffe). If there are more than one tensors, use one line for a tensor.
input_shapes	The shapes of the input tensors, default is NHWC order.
output_shapes	The shapes of the output tensors, default is NHWC order.
input_ranges	The numerical range of the input tensors' data, default [-1, 1]. It is only for test.
validation_inputs_data	[optional] Specify Numpy validation inputs. When not provided, [-1, 1] random values will be used.
validation_threshold	[optional] Specify the similarity threshold for validation. A dict with key in 'CPU', 'GPU' and/or 'HEXAGON' and value <= 1.0.
backend	The onnx backend framework for validation, could be [tensorflow, caffe2, pytorch], default is tensorflow.
runtime	The running device, one of [cpu, gpu, dsp, cpu_gpu]. cpu_gpu contains CPU and GPU model definition so you can run the model on both CPU and GPU.
data_type	[optional] The data type used for specified runtime. [fp16_fp32, fp32_fp32] for GPU, default is fp16_fp32, [fp32] for CPU and [uint8] for DSP.
input_data_types	[optional] The input data type for specific op(eg. gather), which can be [int32, float32], default to float32.
input_data_formats	[optional] The format of the input tensors, one of [NONE, NHWC]. If there is no format of the input, please use NONE. If only one single format is specified, all inputs will use that format, default is NHWC order.
output_data_formats	[optional] The format of the output tensors, one of [NONE, NHWC]. If there is no format of the output, please use NONE. If only one single format is specified, all inputs will use that format, default is NHWC order.
limit_opencl_kernel_time	[optional] Whether splitting the OpenCL kernel within 1 ms to keep UI responsiveness, default is 0.
obfuscate	[optional] Whether to obfuscate the model operator name, default to 0.
winograd	[optional] Which type winograd to use, could be [0, 2, 4]. 0 for disable winograd, 2 and 4 for enable winograd, 4 may be faster than 2 but may take more memory.

Note: Some command tools:

```
# Get device's soc info.
adb shell getprop | grep platform

# command for generating sha256_sum
sha256sum /path/to/your/file
```

6.3 Advanced usage

There are three common advanced use cases:

- run your model on the embedded device(ARM LINUX)
- converting model to C++ code.
- tuning GPU kernels for a specific SoC.

6.4 Run you model on the embedded device(ARM Linux)

The way to run your model on the ARM Linux is nearly same as with android, except you need specify a device config file.

```
python tools/converter.py run --config=/path/to/your/model_deployment_file.yml --
↪device_yml=/path/to/devices.yml
```

There are two steps to do before run:

1. configure login without password

   MACE use ssh to connect embedded device, you should copy your public key to embedded device with the blow command.

   ```
   cat ~/.ssh/id_rsa.pub | ssh -q {user}@{ip} "cat >> ~/.ssh/authorized_keys"
   ```

2. write your own device yaml configuration file.

   - **Example**

     Here is an device yaml config demo.

     ```
     # one yaml config file can contain multi device info
     devices:
       # The name of the device
       nanopi:
       # arm64 or armhf
         target_abis: [arm64, armhf]
       # device soc, you can get it from device manual
         target_socs: RK3399
       # device model full name
         models: FriendlyElec Nanopi M4
       # device ip address
         address: 10.0.0.0
     ```

(continues on next page)

(continued from previous page)

```
  # login username
    username: user
  raspberry:
    target_abis: [armv7l]
    target_socs: BCM2837
    models: Raspberry Pi 3 Model B Plus Rev 1.3
    address: 10.0.0.1
    username: user
```

- **Configuration** The detailed explanation is listed in the blow table.

Options	Usage
target_abis	Device supported abis, you can get it via `dpkg --print-architecture` and `dpkg --print-foreign-architectures` command, if more than one abi is supported, separate them by commas.
target_socs	device soc, you can get it from device manual, we haven't found a way to get it in shell.
models	device models full name, you can get via get `lshw` command (third party package, install it via your package manager). see it's product value.
address	Since we use ssh to connect device, ip address is required.
username	login username, required.

6.5 Convert model(s) to C++ code

- **1. Change the model deployment file(.yml)**

  If you want to protect your model, you can convert model to C++ code. there are also two cases:

  – convert model graph to code and model weight to file with below model configuration.

```
model_graph_format: code
model_data_format: file
```

  – convert both model graph and model weight to code with below model configuration.

```
model_graph_format: code
model_data_format: code
```

Note: Another model protection method is using `obfuscate` to obfuscate names of model's operators.

- **2. Convert model(s) to code**

```
python tools/converter.py convert --config=/path/to/model_deployment_file.
↪yml
```

  The command will generate **${library_name}.a** in **builds/${library_name}/model** directory and ** *.h* * in **builds/${library_name}/include** like the following dir-tree.

```
# model_graph_format: code
# model_data_format: file

builds
├── include
│   └── mace
│       └── public
│           ├── mace_engine_factory.h
│           └── mobilenet_v1.h
└── model
    ├── mobilenet-v1.a
    └── mobilenet_v1.data

# model_graph_format: code
# model_data_format: code

builds
├── include
│   └── mace
│       └── public
│           ├── mace_engine_factory.h
│           └── mobilenet_v1.h
└── model
    └── mobilenet-v1.a
```

- **3. Deployment**
  - Link *libmace.a* and *${library_name}.a* to your target.
  - Refer to `mace/examples/example.cc`for full usage. The following list the key steps.

```
// Include the headers
#include "mace/public/mace.h"
// If the model_graph_format is code
#include "mace/public/${model_name}.h"
#include "mace/public/mace_engine_factory.h"

// ... Same with the code in basic usage

// 4. Create MaceEngine instance
std::shared_ptr<mace::MaceEngine> engine;
MaceStatus create_engine_status;
// Create Engine from compiled code
create_engine_status =
    CreateMaceEngineFromCode(model_name.c_str(),
                             model_data_file, // empty string if model_data_
↪format is code
                             input_names,
                             output_names,
                             device_type,
                             &engine);
if (create_engine_status != MaceStatus::MACE_SUCCESS) {
  // Report error or fallback
}

// ... Same with the code in basic usage
```

6.6 Tuning for specific SoC's GPU

If you want to use the GPU of a specific device, you can just specify the `target_socs` in your YAML file and then tune the MACE lib for it (OpenCL kernels), which may get 1~10% performance improvement.

- **1. Change the model deployment file(.yml)**

  Specify `target_socs` in your model deployment file(.yml):

```
target_socs: [sdm845]
```

  Note: Get device's soc info: *adb shell getprop | grep platform*

- **2. Convert model(s)**

```
python tools/converter.py convert --config=/path/to/model_
↪deployment_file.yml
```

- **3. Tuning**

  The tools/converter.py will enable automatic tuning for GPU kernels. This usually takes some time to finish depending on the complexity of your model.

  Note: You should plug in device(s) with the specific SoC(s).

```
python tools/converter.py run --config=/path/to/model_deployment_
↪file.yml --validate
```

  The command will generate two files in *builds/${library_name}/opencl*, like the following dir-tree.

```
builds
└── mobilenet-v2
    ├── model
    │   ├── mobilenet_v2.data
    │   └── mobilenet_v2.pb
    └── opencl
        └── arm64-v8a
            ├── moblinet-v2_compiled_opencl_kernel.MiNote3.sdm660.
↪bin
            └── moblinet-v2_tuned_opencl_parameter.MiNote3.sdm660.
↪bin
```

  - **mobilenet-v2-gpu_compiled_opencl_kernel.MI6.msm8998.bin** stands for the OpenCL binaries used for your models, which could accelerate the initialization stage. Details please refer to OpenCL Specification.
  - **mobilenet-v2-tuned_opencl_parameter.MI6.msm8998.bin** stands for the tuned OpenCL parameters for the SoC.

- **4. Deployment**

  - Change the names of files generated above for not collision and push them to **your own device's directory**.

  - Use like the previous procedure, below lists the key steps differently.

```
// Include the headers
#include "mace/public/mace.h"
// 0. Declare the device type (must be same with ``runtime`` in␣
↪configuration file)
DeviceType device_type = DeviceType::GPU;

// 1. configuration
MaceStatus status;
MaceEngineConfig config(device_type);
std::shared_ptr<GPUContext> gpu_context;

const std::string storage_path ="path/to/storage";
gpu_context = GPUContextBuilder()
    .SetStoragePath(storage_path)
    .SetOpenCLBinaryPaths(path/to/opencl_binary_paths)
    .SetOpenCLParameterPath(path/to/opencl_parameter_file)
    .Finalize();
config.SetGPUContext(gpu_context);
config.SetGPUHints(
    static_cast<GPUPerfHint>(GPUPerfHint::PERF_NORMAL),
    static_cast<GPUPriorityHint>(GPUPriorityHint::PRIORITY_LOW));

// ... Same with the code in basic usage.
```

6.7 Useful Commands

- **run the model**

```
# Test model run time
python tools/converter.py run --config=/path/to/model_deployment_file.yml --round=100

# Validate the correctness by comparing the results against the
# original model and framework, measured with cosine distance for similarity.
python tools/converter.py run --config=/path/to/model_deployment_file.yml --validate

# Check the memory usage of the model(**Just keep only one model in deployment file**)
python tools/converter.py run --config=/path/to/model_deployment_file.yml --
↪round=10000 &
sleep 5
adb shell dumpsys meminfo | grep mace_run
kill %1
```

Warning: `run` rely on `convert` command, you should `convert` before `run`.

- **benchmark and profile model**

the detailed information is in *Benchmark usage*.

```
# Benchmark model, get detailed statistics of each Op.
python tools/converter.py benchmark --config=/path/to/model_deployment_file.yml
```

Warning: `benchmark` rely on `convert` command, you should `benchmark` after `convert`.

Common arguments

option	type	default	commands	explanation
–omp_num_threads	int	-1	run/benchmark	number of threads
–cpu_affinity_policy	int	1	run/benchmark	0:AFFINITY_NONE/1:AFFINITY_BIG_ONLY/2:AFFINITY_LITTLE_ONLY
–gpu_perf_hint	int	3	run/benchmark	0:DEFAULT/1:LOW/2:NORMAL/3:HIGH
–gpu_perf_hint	int	3	run/benchmark	0:DEFAULT/1:LOW/2:NORMAL/3:HIGH
–gpu_priority_hint	int	3	run/benchmark	0:DEFAULT/1:LOW/2:NORMAL/3:HIGH

Use `-h` to get detailed help.

```
python tools/converter.py -h
python tools/converter.py build -h
python tools/converter.py run -h
python tools/converter.py benchmark -h
```

6.8 Reduce Library Size

- **Build for your own usage purpose.**
  - **dynamic library**
    - If the models don't need to run on device `dsp`, change the build option `--define hexagon=true` to `false`. And the library will be decreased about `100KB`.
    - Futher more, if only `cpu` device needed, change `--define opencl=true` to `false`. This way will reduce half of library size to about `700KB` for `armeabi-v7a` and `1000KB` for `arm64-v8a`
    - About `300KB` can be reduced when add `--config symbol_hidden` building option. It will change the visibility of inner apis in libmace.so and lead to linking error when load model(s) in `code` but no effection for `file` mode.
  - **static library**
    - The methods in dynamic library can be useful for static library too. In additional, the static library may also contain model graph and model datas if the configs `model_graph_format` and `model_data_format` in deployment file are set to `code`.
    - It is recommended to use `version script` and `strip` feature when linking mace static library. The effect is remarkable.
- Remove the unused ops.

Remove the registration of the ops unused for your models in the `mace/ops/ops_register.cc`, which will reduce the library size significantly. the final binary just link the registered ops' code.

```
#include "mace/ops/ops_register.h"

namespace mace {
namespace ops {
// Just leave the ops used in your models

...

}  // namespace ops


OpRegistry::OpRegistry() : OpRegistryBase() {
// Just leave the ops used in your models

  ...

  ops::RegisterMyCustomOp(this);

  ...

}

}  // namespace mace
```

CHAPTER 7

Benchmark usage

This part contains the usage of MACE benchmark tools.

7.1 Overview

As mentioned in the previous part, there are two kinds of benchmark tools, one for operator and the other for model.

7.2 Operator Benchmark

Operator Benchmark is used for test and optimize the performance of specific operator.

7.2.1 Usage

```
python tools/bazel_adb_run.py --target="//mace/ops:ops_benchmark" --run_
↪target=True  --args="--filter=.*BM_CONV.*"
```

7.2.2 Output

```
Benchmark                                                    Time(ns)␣
↪Iterations Input(MB/s)   GMACPS
-------------------------------------------------------------------------------
↪-------------------------
MACE_BM_CONV_2D_1_1024_7_7_K1x1S1D1_SAME_1024_float_CPU       1759129        ␣
↪479     114.09      29.21
MACE_BM_CONV_2D_1_1024_7_7_K1x1S1D1_SAME_1024_float_GPU       4031301        ␣
↪226      49.79      12.75
```

(continues on next page)

(continued from previous page)

```
MACE_BM_CONV_2D_1_1024_7_7_K1x1S1D1_SAME_1024_half_GPU         3996357
 ↪266       25.11       12.86
MACE_BM_CONV_2D_1_1024_7_7_K1x1S1D1_SAME_1024_uint8_t_CPU       914994
 ↪1093       54.84       56.15
```

7.2.3 Explanation

Options	Usage
Benchmark	Benchmark unit name.
Time	Time of one round.
Iterations	the number of iterations to run, which is between 10 and 1000,000,000. the value is calculated based on the strategy total run time does not exceed 1s.
Input	The bandwidth of dealing with input. the unit is MB/s.
GMACPS	The speed of running MACs(multiply-accumulation). the unit is G/s.

7.3 Model Benchmark

Model Benchmark is used for test and optimize the performance of your model. This tool could record the running time of the model and the detailed running information of each operator of your model.

7.3.1 Usage

```
python tools/converter.py benchmark --config=/path/to/your/model_deployment.
 ↪yml
```

7.3.2 Output

```
I benchmark_model.cc:158 ----------------------------------------------------
 ↪-----------------
I benchmark_model.cc:158                                  Warm Up
I benchmark_model.cc:158 ----------------------------------------------------
 ↪-----------------
I benchmark_model.cc:158 | round | first(ms) | curr(ms) | min(ms) | max(ms)
 ↪| avg(ms) |   std |
I benchmark_model.cc:158 ----------------------------------------------------
 ↪-----------------
I benchmark_model.cc:158 |     1 |    51.481 |   51.481 |  51.481 |  51.481
 ↪|  51.481 | 0.000 |
I benchmark_model.cc:158 ----------------------------------------------------
 ↪-----------------
I benchmark_model.cc:158
I benchmark_model.cc:158 ----------------------------------------------------
 ↪-----------------
I benchmark_model.cc:158                           Run without statistics
I benchmark_model.cc:158 ----------------------------------------------------
 ↪-----------------
```

(continues on next page)

(continued from previous page)

```
I benchmark_model.cc:158 | round | first(ms) | curr(ms) | min(ms) | max(ms)
↪| avg(ms) |      std |
I benchmark_model.cc:158 ----------------------------------------------------
↪---------------------
I benchmark_model.cc:158 |   100 |    30.272 |   31.390 |  29.938 |  45.966
↪|  30.913 | 1850.983 |
I benchmark_model.cc:158 ----------------------------------------------------
↪---------------------
I benchmark_model.cc:158
I benchmark_model.cc:158 ----------------------------------------------------
↪---------------------
I benchmark_model.cc:158                          Run with statistics
I benchmark_model.cc:158 ----------------------------------------------------
↪---------------------
I benchmark_model.cc:158 | round | first(ms) | curr(ms) | min(ms) | max(ms)
↪| avg(ms) |      std |
I benchmark_model.cc:158 ----------------------------------------------------
↪---------------------
I benchmark_model.cc:158 |   100 |    32.358 |   33.327 |  32.293 |  33.607
↪|  33.002 | 310.435 |
I benchmark_model.cc:158 ----------------------------------------------------
↪---------------------
I statistics.cc:343 ---------------------------------------------------------
↪---------------------------------------------------------------------------
↪-----------------------------------------------------
I statistics.cc:343
↪                            Sort by Run Order
I statistics.cc:343 ---------------------------------------------------------
↪---------------------------------------------------------------------------
↪-----------------------------------------------------
I statistics.cc:343 |         Op Type |  Start | First | Avg(ms) |     % |
↪ cdf% | GMACPS | Stride |   Pad |    Filter Shape |   Output Shape |
↪Dilation |                                                 name |
I statistics.cc:343 ---------------------------------------------------------
↪---------------------------------------------------------------------------
↪-----------------------------------------------------
I statistics.cc:343 |       Transpose |  0.000 | 0.102 |   0.100 | 0.315 |
↪0.315 |  0.000 |        |       |                 |  [1,3,224,224] |
↪    |                                                input |
I statistics.cc:343 |          Conv2D |  0.107 | 1.541 |   1.570 | 4.943 |
↪5.258 |  6.904 |  [2,2] |  SAME |      [32,3,3,3] | [1,32,112,112] |    [1,
↪1] |             MobilenetV1/MobilenetV1/Conv2d_0/Relu6 |
I statistics.cc:343 | DepthwiseConv2d |  1.724 | 0.936 |   0.944 | 2.972 |
↪8.230 |  3.827 |  [1,1] |  SAME |      [1,32,3,3] | [1,32,112,112] |    [1,
↪1] |   MobilenetV1/MobilenetV1/Conv2d_1_depthwise/Relu6 |
I statistics.cc:343 |         Softmax | 32.835 | 0.039 |   0.042 | 0.131 |
↪99.996 |  0.000 |        |       |                 |       [1,1001] |
↪    |                    MobilenetV1/Predictions/Softmax |
I statistics.cc:343 |        Identity | 32.880 | 0.001 |   0.001 | 0.004 |
↪100.000 |  0.000 |        |       |                 |       [1,1001] |
↪     | mace_output_node_MobilenetV1/Predictions/Reshape_1 |
I statistics.cc:343 ---------------------------------------------------------
↪---------------------------------------------------------------------------
↪-----------------------------------------------------
I statistics.cc:343
I statistics.cc:343 ---------------------------------------------------------
↪---------------------------------------------------------------------------
↪-------------------------------------------
```

(continues on next page)

(continued from previous page)

```
I statistics.cc:343                                                                   ␣
↪                    Sort by Computation Time
I statistics.cc:343 ----------------------------------------------------------
↪-----------------------------------------------------------------------
↪----------------------------------------------
I statistics.cc:343 | Op Type |  Start | First | Avg(ms) |      % |    cdf% |␣
↪GMACPS | Stride |  Pad |     Filter Shape |    Output Shape | Dilation |      ␣
↪                                       name |
I statistics.cc:343 ----------------------------------------------------------
↪-----------------------------------------------------------------------
↪----------------------------------------------
I statistics.cc:343 |  Conv2D | 30.093 | 2.102 |   2.198 | 6.922 |  6.922 |␣
↪23.372 |  [1,1] | SAME | [1024,1024,1,1] |   [1,1024,7,7] |     [1,1] |␣
↪MobilenetV1/MobilenetV1/Conv2d_13_pointwise/Relu6 |
I statistics.cc:343 |  Conv2D |  7.823 | 2.115 |   2.164 | 6.813 | 13.735 |␣
↪23.747 |  [1,1] | SAME |   [128,128,1,1] |  [1,128,56,56] |     [1,1] | ␣
↪MobilenetV1/MobilenetV1/Conv2d_3_pointwise/Relu6 |
I statistics.cc:343 |  Conv2D | 15.859 | 2.119 |   2.109 | 6.642 | 20.377 |␣
↪24.358 |  [1,1] | SAME |   [512,512,1,1] |  [1,512,14,14] |     [1,1] | ␣
↪MobilenetV1/MobilenetV1/Conv2d_7_pointwise/Relu6 |
I statistics.cc:343 |  Conv2D | 23.619 | 2.087 |   2.096 | 6.599 | 26.976 |␣
↪24.517 |  [1,1] | SAME |   [512,512,1,1] |  [1,512,14,14] |     [1,1] |␣
↪MobilenetV1/MobilenetV1/Conv2d_10_pointwise/Relu6 |
I statistics.cc:343 |  Conv2D | 26.204 | 2.081 |   2.093 | 6.590 | 33.567 |␣
↪24.549 |  [1,1] | SAME |   [512,512,1,1] |  [1,512,14,14] |     [1,1] |␣
↪MobilenetV1/MobilenetV1/Conv2d_11_pointwise/Relu6 |
I statistics.cc:343 |  Conv2D | 21.038 | 2.036 |   2.091 | 6.585 | 40.152 |␣
↪24.569 |  [1,1] | SAME |   [512,512,1,1] |  [1,512,14,14] |     [1,1] | ␣
↪MobilenetV1/MobilenetV1/Conv2d_9_pointwise/Relu6 |
I statistics.cc:343 |  Conv2D | 18.465 | 2.034 |   2.082 | 6.554 | 46.706 |␣
↪24.684 |  [1,1] | SAME |   [512,512,1,1] |  [1,512,14,14] |     [1,1] | ␣
↪MobilenetV1/MobilenetV1/Conv2d_8_pointwise/Relu6 |
I statistics.cc:343 |  Conv2D |  2.709 | 1.984 |   2.058 | 6.482 | 53.188 |␣
↪12.480 |  [1,1] | SAME |     [64,32,1,1] | [1,64,112,112] |     [1,1] | ␣
↪MobilenetV1/MobilenetV1/Conv2d_1_pointwise/Relu6 |
I statistics.cc:343 |  Conv2D | 12.220 | 1.788 |   1.901 | 5.986 | 59.174 |␣
↪27.027 |  [1,1] | SAME |   [256,256,1,1] |  [1,256,28,28] |     [1,1] | ␣
↪MobilenetV1/MobilenetV1/Conv2d_5_pointwise/Relu6 |
I statistics.cc:343 |  Conv2D |  0.107 | 1.541 |   1.570 | 4.943 | 64.117 | ␣
↪6.904 |  [2,2] | SAME |       [32,3,3,3] | [1,32,112,112] |     [1,1] |      ␣
↪      MobilenetV1/MobilenetV1/Conv2d_0/Relu6 |
I statistics.cc:343 ----------------------------------------------------------
↪-----------------------------------------------------------------------
↪----------------------------------------------
I statistics.cc:343
I statistics.cc:343 ----------------------------------------------------------
↪-------------------------------------
I statistics.cc:343                                          Stat by Op Type
I statistics.cc:343 ----------------------------------------------------------
↪-------------------------------------
I statistics.cc:343 |          Op Type | Count | Avg(ms) |      % |    cdf% |␣
↪         MACs | GMACPS | Called times |
I statistics.cc:343 ----------------------------------------------------------
↪-------------------------------------
I statistics.cc:343 |           Conv2D |    15 |  24.978 | 78.693 |  78.693 |␣
↪551,355,392 | 22.074 |           15 |
I statistics.cc:343 | DepthwiseConv2d |    13 |   6.543 | 20.614 |  99.307 |␣
↪ 17,385,984 |  2.657 |           13 |
```

(continues on next page)

(continued from previous page)

```
I statistics.cc:343 |       Transpose |      1 |   0.100 |  0.315 |  99.622 |␣
↪          0 |  0.000 |            1 |
I statistics.cc:343 |          Pooling |      1 |   0.072 |  0.227 |  99.849 |␣
↪          0 |  0.000 |            1 |
I statistics.cc:343 |          Softmax |      1 |   0.041 |  0.129 |  99.978 |␣
↪          0 |  0.000 |            1 |
I statistics.cc:343 |          Squeeze |      1 |   0.006 |  0.019 |  99.997 |␣
↪          0 |  0.000 |            1 |
I statistics.cc:343 |         Identity |      1 |   0.001 |  0.003 | 100.000 |␣
↪          0 |  0.000 |            1 |
I statistics.cc:343 ------------------------------------------------------------
↪-----------------------------------
I statistics.cc:343
I statistics.cc:343 ------------------------------------------------------------
I statistics.cc:343           Stat by MACs(Multiply-Accumulation)
I statistics.cc:343 ------------------------------------------------------------
I statistics.cc:343 |       total | round | first(G/s) | avg(G/s) |      std |
I statistics.cc:343 ------------------------------------------------------------
I statistics.cc:343 | 568,741,376 |   100 |     18.330 |   17.909 | 301.326 |
I statistics.cc:343 ------------------------------------------------------------
I statistics.cc:343 ------------------------------------------------------------
↪---------------
I statistics.cc:343                            Summary of Ops' Stat
I statistics.cc:343 ------------------------------------------------------------
↪---------------
I statistics.cc:343 | round | first(ms) | curr(ms) | min(ms) | max(ms) |␣
↪avg(ms) |      std |
I statistics.cc:343 ------------------------------------------------------------
↪---------------
I statistics.cc:343 |   100 |    31.028 |   32.093 |  31.028 |  32.346 |  31.
↪758 | 301.326 |
I statistics.cc:343 ------------------------------------------------------------
↪---------------
```

7.3.3 Explanation

There are 8 sections of the output information.

1. **Warm Up**

This section lists the time information of warm-up run. The detailed explanation is list as below.

Key	Explanation
round	the number of round has been run.
first	the run time of first round. unit is millisecond.
curr	the run time of last round. unit is millisecond.
min	the minimal run time of all rounds. unit is millisecond.
max	the maximal run time of all rounds. unit is millisecond.
avg	the average run time of all rounds. unit is millisecond.
std	the standard deviation of all rounds.

2. **Run without statistics**

This section lists the run time information without statistics code. the detailed explanation is the same as the section of Warm Up.

3. **Run with statistics**

This section lists the run time information with statistics code, the time maybe longer compared with the second section. the detailed explanation is the same as the section of Warm Up.

4. **Sort by Run Order**

This section lists the detailed run information of every operator in your model. The operators is listed based on the run order, Every line is an operator of your model. The detailed explanation is list as below.

Key	Explanation
Op Type	the type of operator.
Start	the start time of the operator. unit is millisecond.
First	the run time of first round. unit is millisecond.
Avg	the average run time of all rounds. unit is millisecond.
%	the percentage of total running time.
cdf%	the cumulative percentage of running time.
GMACPS	The number of run MACs(multiply-accumulation) per second. the unit is G/s.
Stride	the stride parameter of the operator if exist.
Pad	the pad parameter of the operator if exist.
Filter Shape	the filter shape of the operator if exist.
Output Shape	the output shape of the operator.
Dilation	the dilation parameter of the operator if exist.
Name	the name of the operator.

5. **Sort by Computation time**

This section lists the top-10 most time-consuming operators. The operators is listed based on the computation time, the detailed explanation is the same as previous section.

6. **Stat by Op Type**

This section stats the run information about operators based on operator type.

Op Type	the type of operator.
Count	the number of operators with the type.
Avg	the average run time of the operator. unit is millisecond.
%	the percentage of total running time.
cdf%	the cumulative percentage of running time.
MACs	The number of MACs(multiply-accumulation).
GMACPS	The number of MACs(multiply-accumulation) runs per second. the unit is G/s.
Called times	the number of called times in all rounds.

7. **Stat by MACs**

This section stats the MACs information of your model.

total	the number of MACs of your model.
round	the number of round has been run.
First	the GMAPS of first round. unit is G/s.
Avg	the average GMAPS of all rounds. unit is G/s.
std	the standard deviation of all rounds.

8. **Summary of Ops' Stat**

This section lists the run time information which is summation of every operator's run time. which may be shorter than the model's run time with statistics. the detailed explanation is the same as the section of Warm Up.

CHAPTER 8

Operator lists

Operator	Supported	Remark
AVERAGE_POOL_2D	Y	
ARGMAX	Y	Only CPU and TensorFlow is supported.
BATCH_NORM	Y	Fusion with activation is supported.
BATCH_TO_SPACE_ND	Y	
BIAS_ADD	Y	
CAST	Y	Only CPU and TensorFlow model is supported.
CHANNEL_SHUFFLE	Y	
CONCATENATION	Y	For GPU only support channel axis concatenation.
CONV_2D	Y	Fusion with BN and activation layer is supported.
CROP	Y	Only Caffe's crop layer is supported (in GPU, offset on channel-dim sho
DECONV_2D	Y	Supports Caffe's Deconvolution and TensorFlow's tf.layers.conv2d_tran
DEPTHWISE_DECONV2D	Y	Supports Caffe's Group and Depthwise Deconvolution. For GPU only su
DEPTHWISE_CONV_2D	Y	Only multiplier = 1 is supported; Fusion is supported.
DEPTH_TO_SPACE	Y	
DEQUANTIZE	Y	Model quantization will be supported later.
ELEMENT_WISE	Y	ADD/MUL/DIV/MIN/MAX/NEG/ABS/SQR_DIFF/POW/RSQRT/EQU
EMBEDDING_LOOKUP	Y	
EXPANDDIMS	Y	Only CPU and TensorFlow is supported.
FILL	Y	Only CPU and TensorFlow is supported.
FULLY_CONNECTED	Y	
GROUP_CONV_2D		Caffe model with group count = channel count is supported.
IDENTITY	Y	Only TensorFlow model is supported.
LOCAL_RESPONSE_NORMALIZATION	Y	
LOGISTIC	Y	
LSTM		
MATMUL	Y	Only CPU is supported.
MAX_POOL_2D	Y	
PAD	Y	

Table 1 – continued from previous page

Operator	Supported	Remark
PSROI_ALIGN	Y	
PRELU	Y	Only Caffe model is supported
REDUCE_MEAN	Y	Only TensorFlow model is supported. For GPU only H + W axis reduce
RELU	Y	
RELU1	Y	
RELU6	Y	
RELUX	Y	
RESHAPE	Y	Limited support: GPU only supports softmax-like usage, CPU only supp
RESIZE_BILINEAR	Y	
REVERSE	Y	Only CPU and Tensorflow is supported
RNN		
RPN_PROPOSAL_LAYER	Y	
SHAPE	Y	Only CPU and TensorFlow is supported.
STACK	Y	Only CPU and TensorFlow is supported.
STRIDEDSLICE	Y	Only CPU and TensorFlow is supported.
SPLIT	Y	In Caffe, this op is equivalent to SLICE; For GPU only support channel
SOFTMAX	Y	
SPACE_TO_BATCH_ND	Y	
SPACE_TO_DEPTH	Y	
SQUEEZE	Y	Only CPU and TensorFlow is supported.
TANH	Y	
TRANSPOSE	Y	Only CPU and TensorFlow is supported.
UNSTACK	Y	Only CPU and TensorFlow is supported.

CHAPTER 9

Quantization

MACE supports two kinds of quantization mechanisms, i.e.,

- **Quantization-aware training (Recommend)**

After pre-training model using float point, insert simulated quantization operations into the model. Fine tune the new model. Refer to Tensorflow quantization-aware training.

- **Post training quantization**

After pre-training model using float point, estimate output range of each activation layer using sample inputs.

9.1 Quantization-aware training

It is recommended that developers fine tune the fixed-point model, as experiments show that by this way accuracy could be improved, especially for lightweight models, e.g., MobileNet. The only thing you need to make it run using MACE is to add the following config to model yaml file:

1. *input_ranges*: the ranges of model's inputs, e.g., -1.0,1.0.
2. *quantize*: set *quantize* to be 1.

9.2 Post training quantization

MACE supports post-training quantization if you want to take a chance to quantize model directly without fine tuning. This method requires developer to calculate tensor range of each activation layer statistically using sample inputs. MACE provides tools to do statistics with following steps:

1. Convert original model to run on CPU host without obfuscation (by setting *target_abis* to *host*, *runtime* to *cpu*, and *obfuscate* to *0*, appending *:0* to *output_tensors* if missing in yaml config). E.g.,

```
python tools/converter.py convert --config ../mace-models/inception-v3/
↪inception-v3.yml
```

2. Log tensor range of each activation layer by inferring several samples.

```
python tools/converter.py run --config ../mace-models/inception-v3/inception-
↪v3.yml --example --quantize_stat --input_dir samples > range_log
```

3. Calculate overall range of each activation layer by specifying percentage cutoff.

```
python mace/python/tools/quantization/quantize_stat.py --log_file range_log -
↪-percentile 5 > overall_range
```

4. Convert quantized model (by setting *quantize* to *1* and *quantize_range_file* to the overall_range file path in yaml config).

Note: *quantize_weights* and *quantize_nodes* should not be specified when using *TransformGraph* tool if using MACE quantization.

CHAPTER 10

Contributing guide

10.1 License

The source file should contain a license header. See the existing files as the example.

10.2 Python coding style

Changes to Python code should conform to PEP8 Style Guide for Python Code.

You can use pycodestyle to check the style.

10.3 C++ coding style

Changes to C++ code should conform to Google C++ Style Guide.

You can use cpplint to check the style and use clang-format to format the code:

```
clang-format -style="{BasedOnStyle: google,           \
                      DerivePointerAlignment: false,  \
                      PointerAlignment: Right,        \
                      BinPackParameters: false}" $file
```

10.4 C++ logging guideline

VLOG is used for verbose logging, which is configured by environment variable `MACE_CPP_MIN_VLOG_LEVEL`. The guideline of VLOG level is as follows:

```
0. Ad hoc debug logging, should only be added in test or temporary ad hoc
   debugging
1. Important network level Debug/Latency trace log (Op run should never
   generate level 1 vlog)
2. Important op level Latency trace log
3. Unimportant Debug/Latency trace log
4. Verbose Debug/Latency trace log
```

10.5 C++ marco

C++ macros should start with `MACE_`, except for most common ones like `LOG` and `VLOG`.

CHAPTER 11

Adding a new Op

You can create a custom op if it is not supported yet.

To add a custom op, you need to follow these steps:

11.1 Implement the Operation

The Best way is to refer to the implementation of other operator(e.g. `/mace/ops/activation.cc`)

Define the new Op class in `mace/ops/my_custom_op.cc`.

1. ARM kernels: Kernel about NEON is located at `mace/ops/arm/my_custom_op.cc`
2. GPU kernels: OpenCL kernel API is defined in `mace/ops/opencl/my_custom_op.h`,
   - Kernel based on Image is realized in `mace/ops/opencl/image/my_custom_op.cc`,
   - Kernel based on Buffer is realized in `mace/ops/opencl/buffer/my_custom_op.cc`.
   - OpenCL kernel file is realized in `mace/ops/opencl/cl/my_custom_op.cl`.
   - Add the path of opencl kernel file in file `mace/repository/opencl-kernel/opencl_kernel_configure.bzl`

The structure of Op is like the following code.

```
#include "mace/core/operator.h"

namespace mace {
namespace ops {

template <DeviceType D, class T>
class MyCustomOp;

template <>
class MyCustomOp<DeviceType::CPU, float> : public Operation {
```

(continues on next page)

(continued from previous page)

```
...
}

#ifdef MACE_ENABLE_OPENCL
template <typename T>
class MyCustomOp<DeviceType::GPU, T> : public Operation {
...
};
#endif  // MACE_ENABLE_OPENCL

void RegisterMyCustomOp(OpRegistryBase *op_registry) {
  MACE_REGISTER_OP(op_registry, "MyCustomOp", MyCustomOp,
                   DeviceType::CPU, float);

#ifdef MACE_ENABLE_OPENCL
  MACE_REGISTER_OP(op_registry, "MyCustomOp", MyCustomOp,
                   DeviceType::GPU, float);

  MACE_REGISTER_OP(op_registry, "MyCustomOp", MyCustomOp,
                   DeviceType::GPU, half);
#endif  // MACE_ENABLE_OPENCL
}

}  // namespace ops
}  // namespace mace
```

11.2 Register the Operation

Register the new Op in `mace/ops/ops_register.cc`.

```
#include "mace/ops/ops_register.h"

namespace mace {
namespace ops {
// Keep in lexicographical order

...

extern void RegisterMyCustomOp(OpRegistryBase *op_registry);

...

}  // namespace ops


OpRegistry::OpRegistry() : OpRegistryBase() {
  // Keep in lexicographical order

  ...

  ops::RegisterMyCustomOp(this);

  ...
```

(continues on next page)

(continued from previous page)

```
}

}  // namespace mace
```

11.3 Add UTs

Add operation unit tests in `mace/ops/my_custom_op_test.cc`

11.4 Add benchmark

Add operation benchmark in `mace/ops/my_custom_op_benchmark.cc` It's strongly recommended to add unit tests and micro benchmarks for your new Op. If you wish to contribute back, it's required.

11.5 Add Op in model converter

You need to add this new Op in the model converter.

11.6 Document the new Op

Finally, add an entry in operator table in the document.

CHAPTER 12

How to run tests

To run tests, you need to first cross compile the code, push the binary into the device and then execute the binary. To automate this process, MACE provides `tools/bazel_adb_run.py` tool.

You need to make sure your device has been connected to your dev pc before running tests.

12.1 Run unit tests

MACE use gtest for unit tests.

- Run all unit tests defined in a Bazel target, for example, run `ops_test`:

```
python tools/bazel_adb_run.py --target="//mace/ops:ops_test" \
                              --run_target=True
```

- Run unit tests with gtest filter, for example, run `Conv2dOpTest` unit tests:

```
python tools/bazel_adb_run.py --target="//mace/ops:ops_test" \
                              --run_target=True \
                              --args="--gtest_filter=Conv2dOpTest*"
```

12.2 Run micro benchmarks

MACE provides a micro benchmark framework for performance tuning.

- Run all micro benchmarks defined in a Bazel target, for example, run all `ops_benchmark` micro benchmarks:

```
python tools/bazel_adb_run.py --target="//mace/ops:ops_benchmark" \
                              --run_target=True
```

- Run micro benchmarks with regex filter, for example, run all `CONV_2D` GPU micro benchmarks:

```
python tools/bazel_adb_run.py --target="//mace/ops:ops_benchmark" \
                              --run_target=True \
                              --args="--filter=MACE_BM_CONV_2D_.*_GPU"
```

CHAPTER 13

How to debug

13.1 Debug correctness

MACE provides tools to examine correctness of model execution by comparing model's output of MACE with output of training platform (e.g., Tensorflow, Caffe). Three metrics are used as comparison results:

- **Cosine Similarity**:

$$Cosine\ Similarity = \frac{X \cdot X'}{\|X\|\|X'\|}$$

This metric will be approximately equal to 1 if output is correct.

- **SQNR** (Signal-to-Quantization-Noise Ratio):

$$SQNR = \frac{P_{signal}}{P_{noise}} = \frac{\|X\|^2}{\|X - X'\|^2}$$

It is usually used to measure quantization accuracy. The higher SQNR is, the better accuracy will be.

- **Pixel Accuracy**:

$$Pixel\ Accuracy = \frac{\sum_{b=1}^{batch} equal(\text{argmax}X_b, \text{argmax}X'_b)}{batch}$$

It is usually used to measure classification accuracy. The higher the better.

where X is expected output (from training platform) whereas X' is actual output (from MACE) .

You can validate it by specifying *–validate* while running the model.

```
# Validate the correctness by comparing the results against the
# original model and framework
python tools/converter.py run --config=/path/to/your/model_deployment_file.
↪yml --validate
```

MACE automatically validate these metrics by running models with synthetic inputs. If you want to specify input data to use, you can add an option in yaml config under 'subgraphs', e.g.,

```
models:
  mobilenet_v1:
    platform: tensorflow
    model_file_path: https://cnbj1.fds.api.xiaomi.com/mace/miai-models/mobilenet-v1/
↪mobilenet-v1-1.0.pb
    model_sha256_checksum:␣
↪71b10f540ece33c49a7b51f5d4095fc9bd78ce46ebf0300487b2ee23d71294e6
    subgraphs:
      - input_tensors:
          - input
        input_shapes:
          - 1,224,224,3
        output_tensors:
          - MobilenetV1/Predictions/Reshape_1
        output_shapes:
          - 1,1001
        check_tensors:
          - MobilenetV1/Logits/Conv2d_1c_1x1/BiasAdd:0
        check_shapes:
          - 1,1,1,1001
        validation_inputs_data:
          - https://cnbj1.fds.api.xiaomi.com/mace/inputs/dog.npy
```

If model's output is suspected to be incorrect, it might be useful to debug your model layer by layer by specifying an intermediate layer as output, or use binary search method until suspicious layer is found.

You can also specify *–validate_all_layers* to validate all the layers of the model(excluding some layers changed by MACE, e.g., BatchToSpaceND), it only supports TensorFlow now. You can find validation results in *builds/your_model/model/runtime_in_yaml/log.csv*.

For quantized model, if you want to check one layer, you can add *check_tensors* and *check_shapes* like in the yaml above. You can only specify MACE op's output.

13.2 Debug memory usage

The simplest way to debug process memory usage is to use `top` command. With `-H` option, it can also show thread info. For android, if you need more memory info, e.g., memory used of all categories, `adb shell dumpsys meminfo` will help. By watching memory usage, you can check if memory usage meets expectations or if any leak happens.

13.3 Debug performance

Using MACE, you can benchmark a model by examining each layer's duration as well as total duration. Or you can benchmark a single op. The detailed information is in *Benchmark usage*.

13.4 Debug model conversion

After model is converted to MACE model, a literal model graph is generated in directory *mace/codegen/models/your_model*. You can refer to it when debugging model conversion.

13.5 Debug engine using log

Mace defines two sorts of logs: one is for users (LOG), the other is for developers (VLOG).

LOG includes four levels, i.e, `INFO`, `WARNING`, `ERROR`, `FATAL`; Environment variable `MACE_CPP_MIN_LOG_LEVEL` can be set to specify log level of users, e.g., `set MACE_CPP_MIN_LOG_LEVEL=0` will enable `INFO` log level, while `set MACE_CPP_MIN_LOG_LEVEL=4` will enable `FATAL` log level.

VLOG level is specified by numbers, e.g., 0, 1, 2. Environment variable `MACE_CPP_MIN_VLOG_LEVEL` can be set to specify vlog level. Logs with higher levels than which is specified will be printed. So simply specifying a very large level number will make all logs printed.

By using Mace run tool, vlog level can be easily set by option, e.g.,

```
python tools/converter.py run --config /path/to/model.yml --vlog_level=2
```

If models are run on android, you might need to use `adb logcat` to view logs.

13.6 Debug engine using GDB

GDB can be used as the last resort, as it is powerful that it can trace stacks of your process. If you run models on android, things may be a little bit complicated.

```
# push gdbserver to your phone
adb push $ANDROID_NDK_HOME/prebuilt/android-arm64/gdbserver/gdbserver /data/
↪local/tmp/


# set system env, pull system libs and bins to host
export SYSTEM_LIB=/path/to/android/system_lib
export SYSTEM_BIN=/path/to/android/system_bin
mkdir -p $SYSTEM_LIB
adb pull /system/lib/. $SYSTEM_LIB
mkdir -p $SYSTEM_BIN
adb pull /system/bin/. $SYSTEM_BIN


# Suppose ndk compiler used to compile Mace is of android-21
export PLATFORMS_21_LIB=$ANDROID_NDK_HOME/platforms/android-21/arch-arm/usr/
↪lib/


# start gdbserver, make gdb listen to port 6000
# adb shell /data/local/tmp/gdbserver :6000 /path/to/binary/on/phone/example_
↪bin
adb shell LD_LIBRARY_PATH=/dir/to/dynamic/library/on/phone/ /data/local/tmp/
↪gdbserver :6000 /data/local/tmp/mace_run/example_bin
# or attach a running process
adb shell /data/local/tmp/gdbserver :6000 --attach 8700
# forward tcp port
adb forward tcp:6000 tcp:6000


# use gdb on host to execute binary
```

(continues on next page)

(continued from previous page)

```
$ANDROID_NDK_HOME/prebuilt/linux-x86_64/bin/gdb [/path/to/binary/on/host/
↪example_bin]


# connect remote port after starting gdb command
target remote :6000


# set lib path
set solib-search-path $SYSTEM_LIB:$SYSTEM_BIN:$PLATFORMS_21_LIB

# then you can use it as host gdb, e.g.,
bt
```

CHAPTER 14

Memory layout

14.1 CPU runtime memory layout

The CPU tensor buffer is organized in the following order:

Tensor type	Buffer
Intermediate input/output	NCHW
Convolution Filter	OIHW
Depthwise Convolution Filter	MIHW
1-D Argument, length = W	W

14.2 GPU runtime memory layout

GPU runtime implementation base on OpenCL, which uses 2D image with CL_RGBA channel order as the tensor storage. This requires OpenCL 1.2 and above.

The way of mapping the Tensor data to OpenCL 2D image (RGBA) is critical for kernel performance.

In CL_RGBA channel order, each 2D image pixel contains 4 data items. The following tables describe the mapping from different type of tensors to 2D RGBA Image.

14.2.1 Input/Output Tensor

The Input/Output Tensor is stored in NHWC format:

Tensor type	Buffer	Image size [width, height]	Explanation
Channel-Major Input/Output	NHWC	[W * (C+3)/4, N * H]	Default Input/Output format
Height-Major Input/Output	NHWC	[W * C, N * (H+3)/4]	WinogradTransform and MatMul output format
Width-Major Input/Output	NHWC	[(W+3)/4 * C, N * H]	Unused now

Each Pixel of **Image** contains 4 elements. The below table list the coordination relation between **Image** and **Buffer**.

Tensor type	Pixel coordinate relationship	Explanation
Channel-Major Input/Output	P[i, j] = {E[n, h, w, c] | (n=j/H, h=j%H, w=i%W, c=[i/W * 4 + k])}	k=[0, 4)
Height-Major Input/Output	P[i, j] = {E[n, h, w, c] | (n=j%N, h=[j/H*4 + k], w=i%W, c=i/W)}	k=[0, 4)
Width-Major Input/Output	P[i, j] = {E[n, h, w, c] | (n=j/H, h=j%H, w=[i%W*4 + k], c=i/W)}	k=[0, 4)

14.2.2 Filter Tensor

Tensor	Buffer	Image size [width, height]	Explanation
Convolution Filter	OIHW	[I, (O+3)/4 * W * H]	Convolution filter format, There is no difference compared to [H*W*I, (O+3)/4]
Depthwise Convlution Filter	MIHW	[H * W * M, (I+3)/4]	Depthwise-Convolution filter format

Each Pixel of **Image** contains 4 elements. The below table list the coordination relation between **Image** and **Buffer**.

Tensor type	Pixel coordinate relationship	Explanation
Convolution Filter	P[m, n] = {E[o, i, h, w] | (o=[n/HW*4+k], i=m, h=T/W, w=T%W)}	HW= H * W, T=n%HW, k=[0, 4)
Depthwise Convlution Filter	P[m, n] = {E[0, i, h, w] | (i=[n*4+k], h=m/W, w=m%W)}	only support multiplier == 1, k=[0, 4)

14.2.3 1-D Argument Tensor

Tensor type	Buffer	Image size [width, height]	Explanation
1-D Argument	W	[(W+3)/4, 1]	1D argument format, e.g. Bias

Each Pixel of **Image** contains 4 elements. The below table list the coordination relation between **Image** and **Buffer**.

Tensor type	Pixel coordinate relationship	Explanation
1-D Argument	P[i, 0] = {E[w] | w=i*4+k}	k=[0, 4)

CHAPTER 15

Frequently asked questions

15.1 Does the tensor data consume extra memory when compiled into C++ code?

When compiled into C++ code, the tensor data will be mmaped by the system loader. For the CPU runtime, the tensor data are used without memory copy. For the GPU and DSP runtime, the tensor data are used once during model initialization. The operating system is free to swap the pages out, however, it still consumes virtual memory addresses. So generally speaking, it takes no extra physical memory. If you are short of virtual memory space (this should be very rare), you can use the option to load the tensor data from data file (can be manually unmapped after initialization) instead of compiled code.

15.2 Why is the generated static library file size so huge?

The static library is simply an archive of a set of object files which are intermediate and contain much extra information, please check whether the final binary file size is as expected.

15.3 Why is the generated binary file (including shared library) size so huge?

When compiling the model into C++ code, the final binary may contains extra debug symbols, they usually take a lot of space. Try to strip the shared library or binary and make sure you are following best practices to reduce the size of an ELF binary, including disabling C++ exception, disabling RTTI, avoiding C++ iostream, hidden internal functions etc. In most cases, the expected overhead should be less than {model weights size in float32}/2 + 3MB.

15.4 How to set the input shape in your model deployment file(.yml) when your model support multiple input shape?

Set the largest input shape of your model. The input shape is used for memory optimization.

15.5 OpenCL allocator failed with CL_OUT_OF_RESOURCES

OpenCL runtime usually requires continuous virtual memory for its image buffer, the error will occur when the OpenCL driver can't find the continuous space due to high memory usage or fragmentation. Several solutions can be tried:

- Change the model by reducing its memory usage
- Split the Op with the biggest single memory buffer
- Change from armeabi-v7a to arm64-v8a to expand the virtual address space
- Reduce the memory consumption of other modules of the same process

15.6 Why is the performance worse than the official result for the same model?

The power options may not set properly, see `mace/public/mace.h` for details.

15.7 Why is the UI getting poor responsiveness when running model with GPU runtime?

Try to set `limit_opencl_kernel_time` to `1`. If still not resolved, try to modify the source code to use even smaller time intervals or changed to CPU or DSP runtime.

15.8 Why is MACE not working on DSP?

Running models on Hexagon DSP need a few prerequisites for DSP developers:

- You need to make sure SOCs of your phone is manufactured by Qualcomm and has HVX supported.
- You need a phone that disables secure boot (once enabled, cannot be reversed, so you probably can only get that type phones from manufacturers)
- You need to sign your phone by using testsig provided by Qualcomm. (Download Qualcomm Hexagon SDK first, plugin your phone to PC, run scripts/testsig.py)
- You need to push `third_party/nnlib/v6x/libhexagon_nn_skel.so` to `/system/vendor/lib/rfsa/adsp/`.

Then, there you go. You can run Mace on Hexagon DSP.