ONE

Release 1.4.0

May 26, 2020

Contents

1 Overview **1**
- 1.1 Background . . . 1
- 1.2 Roadmap . . . 1
- 1.3 Overall Architecture . . . 3
- 1.4 Supported Operations . . . 3
- 1.5 Workgroup . . . 3

2 How To **5**
- 2.1 How to Add a New Operation . . . 5
- 2.2 How to Build Compiler . . . 5
- 2.3 How to Build Package . . . 5
- 2.4 How to Build Runtime . . . 5
- 2.5 How to Contribute . . . 5
- 2.6 How to Create a Pull Request . . . 7
- 2.7 How to Make an Application with Runtime . . . 7
- 2.8 How to Remote Debugging with Visual Studio Code . . . 7
- 2.9 How to Run Package . . . 7
- 2.10 How to Use API . . . 7
- 2.11 How to Use NNAPI Binding . . . 7

3 Runtime **9**
- 3.1 API . . . 9
- 3.2 Core . . . 9
- 3.3 Compute . . . 9

4 Compiler **11**
- 4.1 Frontend . . . 11
- 4.2 Middleend . . . 15
- 4.3 Backend . . . 19
- 4.4 IR . . . 19
- 4.5 Interpreter . . . 23
- 4.6 Libraries . . . 26
- 4.7 Tools . . . 34

5 Common IR **59**
- 5.1 Introduction to circle . . . 59
- 5.2 What is Common IR . . . 59

6 Package **61**
6.1 Introduction to Package . . . 61
6.2 Design of Package . . . 61

7 Platform **63**
7.1 Ubuntu . . . 63
7.2 Tizen . . . 63
7.3 Android . . . 63

8 Devices **65**
8.1 ODROID-XU3 . . . 65
8.2 ODROID-XU4 . . . 65
8.3 Raspberry Pi 3 . . . 65

9 Test & Benchmarks **67**
9.1 Scripts . . . 67
9.2 Benchmarks . . . 67

10 Release **69**
10.1 1.0 . . . 69
10.2 1.1 . . . 69
10.3 1.2 . . . 69
10.4 1.3 . . . 69
10.5 1.4 . . . 69

11 Indices and tables **71**

CHAPTER 1

Overview

1.1 Background

Artificial intelligence (AI) techniques are getting popular and utilized in various products and services. While the cloud-based AI techniques have been used to perform compute/memory intensive inferences because of the powerful servers on cloud, on-device AI technologies are recently drawing attention from the mobile industry for response time reduction, privacy protection, and connection-less AI service. Big mobile players are investing their research effort on the on-device AI technologies and already announced hardware and software on-device AI solutions. We are not leading this trend currently, but since on-device AI area is just started and still in the initial state, there are still opportunities and possibilities to reduce the gap between pioneers and us. We believe on-device AI will become a key differentiator for mobile phone, TV, and other home appliances, and thus developing on-device AI software stack is of paramount importance in order to take leadership in the on-device AI technology.

Although the vision of on-device AI is promising, enabling on-device AI involves unique technical challenges compared to traditional cloud-based approach. This is because on-device AI tries to conduct inference tasks solely on device without connecting to cloud resources. Specifically, hardware resources on device, such as processor performance, memory capacity, and power budget, are very scarce and limit the compute capability, which is typically required to execute complicated neural network (NN) models. For example, in one product requirement, a mobile device should consume less than 1.2W and could use at most 2W only for 10 minutes due to thermal issue. Next, on-device AI software stack needs to support diverse device environments, since embedded platforms may consist of heterogeneous compute devices, such as CPU, GPU, DSP, or neural processing unit (NPU), and use different OS platforms, such as Tizen, Android, or various Linux.

To tackle the challenges above and to have the leadership on on-device AI technology, this project, as the first step, aims at developing a neural network inference framework specialized and optimized for on-device AI.

1.2 Roadmap

This document describes roadmap of **ONE** project.

This project **ONE** aims at providing a high-performance, on-device neural network (NN) inference framework that performs inference of a given NN model on processors, such as CPU, GPU, DSP, or NPU, in the target platform, such

as Tizen, Android, and Ubuntu.

1.2.1 Progress

Until last year, we already saw significant gains in accelerating with a single CPU or GPU backend. We have seen better performance improvements, not only when using a single backend, but even when mixing CPUs or GPUs considering the characteristics of individual operations. It could give us an opportunity to have a high degree of freedom in terms of operator coverage, and possibly provide better performance compared to single backend acceleration.

On the other hand, we introduced the compiler as a front-end. This will support a variety of deep learning frameworks in relatively spacious host PC environments, while the runtime running on the target device is intended to take a smaller burden. In this approach, the compiler and the runtime will effectively share information among themselves by the Common IR, named *circle*, and a container format which is referred to as the *NN Package*.

1.2.2 Goal

In the meantime, we have been working on improving the acceleration performance based on the vision model. From this year, now we start working on the voice model. The runtime requirements for the voice model will be different from those of the vision model. There will be new requirements that we do not recognize yet, along with some already recognized elements such as control flow and dynamic tensor. In addition, recent studies on voice models require efficient support for specific architectures such as attention, transformer, and BERT. Also, depending on the characteristics of most voice models with large memory bandwidth, we will have to put more effort into optimizing the memory bandwidth at runtime.

1.2.3 Deliverables

- Runtime
  - Control Flow support (IF/WHILE)
  - Dynamic Tensor support
  - High quality kernel development for UINT8 quantized model
  - End-to-end performance optimization for voice models
- Compiler
  - More than 100 operations support
  - Standalone *circle* interpreter
  - Completion and application of *circle2circle* pass
    - *circle-quantizer* for UINT8 and INT16
    - *circle-optimizer*
  - Grphical *circle* model viewer

1.2.4 Milestones

- 2020 Project Milestones

1.2.5 Workgroups (WGs)

- We organize WGs for major topics, and each WG will be working on its own major topic by breaking it into small tasks/issues, performing them inside WG, and collaborating between WGs.
- The WG information can be found here.

1.3 Overall Architecture

1.3.1 Compiler

1.3.2 Package

1.3.3 Runtime

1.4 Supported Operations

1.4.1 Compiler

1.4.2 Common IR (circle)

1.4.3 Runtime

1.5 Workgroup

1.5.1 Runtime WG1

1.5.2 Runtime WG2

1.5.3 Runtime WG3

1.5.4 Compiler Frontend WG

1.5.5 Compiler Backend WG

1.5.6 How to Join?

CHAPTER 2

How To

2.1 How to Add a New Operation

2.2 How to Build Compiler

2.2.1 Build Requires

2.2.2 Build for Ubuntu

2.2.3 Build for windows

2.3 How to Build Package

2.4 How to Build Runtime

2.4.1 Build Requires

2.4.2 Build for Ubunut

2.4.3 Build for Tizen

2.4.4 Build for Android

2.5 How to Contribute

ONE always welcomes your contribution, but there are basic guidelines that you should follow in order to make your contribution be accepted.

This document explains such guidelines for beginners.

2.5.1 General contribution guidelines

If you are not familiar with git or github, please visit here for basic understanding of git and github.

2.5.2 How to create a Pull Request

This section explains the steps to create a pull request (PR).

1. Create an issue

   Maintainers will accept your contribution only when it is well aligned with the *roadmap* and design principles of **ONE**. So, it is optional, but recommended for contributors to create an issue and have a discussion with maintainers before writing code.

2. Create a draft PR

   Maintainers will accept your pull request only when it is **reasonably small** and **focused**. Sometimes, your contribution may require huge and loosely-coupled changes. You **should** split your contribution into multiple small, but focused pull requests in this case. Unfortunately, it is possible that maintainers reject your pull request as it is hard for them to understand the intuition behind these changes. So, it is optional, but recommended for contributors to present the full draft of your contribution and have a discussion with maintainers before creating PR(s).

3. Create a commit

   It is time to create a commit for submission once you are convinced that your contribution is ready to go. Please include signed-off message at the end of commit message. If not, your pull request will be **rejected** by CI.

4. Check code format locally

   ONE has its code formatting rules, and any pull request that violates these rules will be **rejected** by CI. So, it is optional, but recommended for contributor to check code format locally before submission.

5. Create a PR

   It is time to send a pull request. Please explain your intention via description. Maintainers will review your pull request based on that description. Each pull request needs approval from at least two reviewers to be accepted. Note that **description should include at least four words**. If not, your pull request will be **rejected** by CI.

6. Request review

   It is recommended to assign reviewers yourself. Maintainers will honor your review request, and accept your pull request only when

   - Approved by 1+ reviewers
   - 0 rejection(Request Changes)
   - 0 pending review request
   - All the reviewers in the list must approve your pull request

   You can add/remove pending review requests in the middle of the review process. Maintainers (or reviewers) could review your pull request even without explicit review request.

7. Update per feedback

   Sometimes, maintainers (or reviewers) will request changes on your pull request. Please update your pull request upon such feedbacks. These update commits will be squashed into the first commit of your pull request later. Please do **NOT** include a sign-off message or write a full description for update commits.

2.6 How to Create a Pull Request

2.7 How to Make an Application with Runtime

2.8 How to Remote Debugging with Visual Studio Code

2.9 How to Run Package

2.10 How to Use API

2.11 How to Use NNAPI Binding

CHAPTER 3

Runtime

3.1 API

3.2 Core

3.3 Compute

CHAPTER 4

Compiler

4.1 Frontend

4.1.1 caffe2circle

caffe2circle is a Caffe-to-Circle model converter.

4.1.2 circle2circle

circle2circle provides Circle optimizations and quantizations as executable tool

4.1.3 enco

enco is a tool which translates a NN model into a C++ source code that implements the following functions:

```
struct Network;

Network *Network_construct();
void Network_destruct(Network *net);

unsigned Network_input_count(const Network *);
const char *Network_input_name(const Network *, unsigned n);
unsigned Network_input_rank(const Network *, unsigned n);
unsigned Network_input_dim(const Network *, unsigned n, unsigned axis);
void Network_input_bind(Network *net, unsigned n, const void *ptr, unsigned len);

unsigned Network_output_count(const Network *net);
const char *Network_output_name(const Network *, unsigned n);
unsigned Network_output_rank(const Network *, unsigned n);
unsigned Network_output_dim(const Network *, unsigned n, unsigned axis);
void Network_output_bind(Network *net, unsigned n, void *ptr, unsigned len);
```

(continues on next page)

(continued from previous page)

```
void Network_invoke(Network *net);
```

Generated C++ code internally uses Android NN API for acceleration.

4.1.4 nnc

Neural Network Compiler

DESCRIPTION

nnc is a neural network compiler that transforms neural networks of various formats into source or machine code.

> At this moment only two NN are supported (MobileNet and InceptionV3) in Tensorflow Lite or Caffe format.

SYNOPSIS

nnc OPTIONS

OPTIONS

```
    --help, -h              -    print usage and exit
    --caffe                 -    treat input file as Caffe model
    --tflite                -    treat input file as Tensor Flow Lite model
    --target                -    select target language to emit for given architecture.
                                 Valid values are 'x86-c++', 'interpreter'
    --nnmodel, -m           -    specify input file with NN model
    --output, -o            -    specify name for output files
    --output-dir, -d        -    specify directory for output files
    --input-model-data      -    interpreter option: specify file with neural network␣
↪input data.
                                 This file contains array of floats in binary form
    --input-node            -    interpreter option: set input node in Computational␣
↪Graph
    --output-node           -    interpreter option: set output node in Computational␣
↪Graph
```

USAGE

Assuming that user has already installed nnc as follows:

```
$ cmake <path_to_nnc_sources> -DCMAKE_INSTALL_PREFIX=<path_to_install>
$ make all && make install
```

Also assuming that we have tflite model (for example inceptionv3.tflite)

1. Running nnc in interpreter mode:

```
<path_to_install>/bin/nnc \
--nnmodel inceptionv3.tflite \
--target interpreter \
--input-model-data data.file \
--input-node input --output-node output
```

2. Running to generate C/C++ source code:

```
<path_to_install>/bin/nnc \
--nnmodel inceptionv3.tflite \
--target x86-c++ \
--output inception \
--output-dir output_dir
```

4.1.5 onnx2circle

onnx2circle is a ONNX-to-Circle model converter.

4.1.6 tf2circle

tf2circle is a TensorFlow-to-Circle model converter.

4.1.7 tf2nnpkg

4.1.8 tf2tflite

tf2tflite is a TensorFlow-to-TensorFlow Lite model converter.

4.1.9 tf2tfliteV2

tf2tfliteV2 is a TensorFlow to TensorFlow Lite model Converter.

Where does V2 come from?

Even though we alreay have *tf2tflite*, we cannot cover all opeartors in TensorFlow. To expand coverage, we introduce *tf2tfliteV2* which uses `TensorFlow Lite Converter`(by Google) internally.

Prerequisite

- Frozen graph from TensorFlow 1.13.1 in binary(`*.pb`) or text(`*.pbtxt`) format
- Desired version of TensorFlow(You can use python virtualenv, docker, etc.)

Example

```
python tf2tfliteV2.py \
> --v1 \
> --input_path=frozen_graph.pb \
> --output_path=converted.tflite \
> --input_arrays=model_inputs \
> --output_arrays=model_outputs
```

```
python tf2tfliteV2.py \
> --v2 \
> --input_path=frozen_graph.pbtxt \
> --output_path=converted.tflite \
> --input_arrays=model_inputs \
> --output_arrays=model_outputs
```

```
python tf2tfliteV2.py \
> --v2 \
> --input_path=multiple_output_graph.pb \
> --output_path=converted.tflite \
> --input_arrays=model_inputs \
> --output_arrays=output,output:1,output:2
```

optional argument

```
  -h, --help            show this help message and exit
  --v1                  Use TensorFlow Lite Converter 1.x
  --v2                  Use TensorFlow Lite Converter 2.x
  --input_path INPUT_PATH
                        Full filepath of the input file.
  --output_path OUTPUT_PATH
                        Full filepath of the output file.
  --input_arrays INPUT_ARRAYS
                        Names of the input arrays, comma-separated.
  --input_shapes INPUT_SHAPES
                        Shapes corresponding to --input_arrays, colon-
                        separated.
  --output_arrays OUTPUT_ARRAYS
                        Names of the output arrays, comma-separated.
```

4.1.10 tflite2circle

tflite2circle is a Tensorflow Lite to Circle model converter.

Usage

Provide *tflite* file input path and *circle* file output path as a parameter to convert.

```
$ tflite2circle in.tflite out.circle
```

4.2 Middleend

4.2.1 exo

exo includes *loco*-to-*T/F Lite* exporter (as a library).

How to add a new TFL node

1. Add a new TFL node into `TFLNodes.lst` and `TFLNodes.h`
2. Define a knob in `Knob.lst` if you need a knob.
3. Add appropriate methods in `TFLShapeInferenceRule.cpp` and `TFLTypeInferenceRule.cpp`
4. Add a new converter under `Conversion` directory
5. Add an appropriate method in `OperationExporter.cpp`
6. Register the converter into `Convert.cpp`

4.2.2 locoex

locoex is an extention of loco. Classes with `COp` prefix enables *Custom Operation*. In this version, a *custom operation* means one of the following:

1. an op that is supported by Tensorflow but not supported both by the moco and the onert
2. an op that is not supported by Tensorflow, moco, and the onert

`COpCall` node will represent IR entity that calls custom operations and kernels.

4.2.3 logo

logo provides *loco* General Graph Passes for Transformation and Optimization

4.2.4 logo-core

logo-core provides *loco* General Graph Pass Core for Transformation and Optimization

4.2.5 mir2loco

4.2.6 moco-tf

moco-tf translates a TensorFlow model into *loco*

Purpose

moco-tf is to convert TensorFlow generated model file to in-memory *loco* IR Graph.

How to use

```
#include <moco/tf/Frontend.h>

...

  ::moco::tf::Frontend moco;

  std::string pb_path = "path_to_pb_file_to_load";

  auto loco_graph = moco.load(sig, pb_path, ::moco::tf::Frontend::FileType::Binary);
```

Dependency

Please refer requires.cmake for dependant modules.

Naming rules

TensorFlow node names

Use `REGISTER_OP` argument used in TensorFlow source `core` folder.

```
cd tensorflow/core
grep -Rn "REGISTER_OP"
```

To see single Op, `Conv2D` for example

```
cd tensorflow/core
grep -Rn "REGISTER_OP" | grep "Conv2D"
```

Names related with TensorFlow nodes

Like `GraphBuilder` and `Canonicalization`, TensorFlow node names can be used as prefix or suffix.

- `Conv2DGraphBuilder`
- `Conv2DCanonicalizier`

TensorFlow Dialect IR

Use `TF` prefix with TensorFlow Dialect node names

- `TFAvgPool`
- `TFConv2D`

This document outlines how to express each TensorFlow operation on top of *loco*

CAUTION All the python examples below are written in Python 3 with TensorFlow v1.13.

DISCLAIMER *loco* does not support named values, but all the below *loco* examples assign "name" to each value to make it easy to read.

4.2.7 Placeholder

Placeholder in *TensorFlow* corresponds to **Pull** in *loco*.

Python:

```
import tensorflow as tf
input = tf.placeholder(dtype=tf.float32, shape=[3, 4], name='input')
print(tf.get_default_graph().as_graph_def())
```

API reference: tf.placeholder

TensorFlow

```
node {
  name: "input"
  op: "Placeholder"
  attr {
    key: "dtype"
    value { type: DT_FLOAT }
  }
  attr {
    key: "shape"
    value {
      shape {
        dim { size: 3 }
        dim { size: 4 }
      }
    }
  }
}
```

loco:

```
%input = Pull(dtype: FLOAT32, shape: [3, 4])
Push(%input)
```

4.2.8 Identity

Identity in *TensorFlow* corresponds to **Forward** in *loco*.

Python:

```
import tensorflow as tf
input = tf.placeholder(dtype=tf.float32, shape=[3, 4])
ident = tf.identity(input)
print(tf.get_default_graph().as_graph_def())
```

API reference: tf.identity

TensorFlow:

```
node {
  name: "Placeholder"
  op: "Placeholder"
  attr {
    key: "dtype"
```

(continues on next page)

(continued from previous page)

```
    value { type: DT_FLOAT }
  }
  attr {
    key: "shape"
    value {
      shape {
        dim { size: 3 }
        dim { size: 4 }
      }
    }
  }
}
node {
  name: "Identity"
  op: "Identity"
  input: "Placeholder"
  attr {
    key: "T"
    value { type: DT_FLOAT }
  }
}
```

loco:

```
%input = Pull(dtype: FLOAT32, shape: [3, 4])
%ident = Forward(%input)
Push(%ident)
```

4.2.9 Const

Const in *TensorFlow* corresponds to **ConstGen** in *loco*.

Python:

```
import tensorflow as tf
constant = tf.constant(value=[1.0], dtype=tf.float32, shape=[3, 4])
tf.get_default_graph().as_graph_def()
```

API reference: tf.constant

TensorFlow:

```
node {
  name: "Const"
  op: "Const"
  attr {
    key: "dtype"
    value { type: DT_FLOAT }
  }
  attr {
    key: "value"
    value {
      tensor {
        dtype: DT_FLOAT
        tensor_shape {
          dim { size: 3 }
```

(continues on next page)

(continued from previous page)

```
          dim { size: 4 }
        }
        float_val: 1.0
      }
    }
  }
}
```

loco:

```
%constant = ConstGen(dtype: FLOAT32, shape: [3, 4], data: ...);
Push(%constant)
```

4.3 Backend

4.4 IR

4.4.1 coco

coco is an experimental coarse-grained intermediate representation (IR) for NN compilers.

4.4.2 loco

loco is a graph-based intermediate representation (IR) for neural network compilers.

4.4.3 Dialect Service

This loco enhancement proposal (*LEP*) discusses how to permit a *loco* graph without canonical dialect.

Revision

Motivation

One of key design principles behind *loco* is to allow users (= NN compiler writers) to easily define their own intermediate representation (IR) on top of shared infrastructure.

Unfortunately, however, there is a gap between dream and reality. It is currently impossible to create a *loco* graph only with non-canonical dialects; there is no way to express the interaction between graph-level output without *canonical.Push* node.

This proposal aims to remove this restriction in order to bridge the gap between dream and reality.

Design

Each dialect is now allowed to expose its internal to its client (such as transformations and core algorithms) through a so-called "Service" interface.

Although this proposal focuses on `output_nodes` helper in *loco.core*, its coverage is not limited to this helper. Any pass and algorithm can take an advantage of this generic infrastructure.

Let us dive into some details.

What is "service"?

A service declares a collection of APIs that each **client** (not dialect) needs.

Let us consider `output_nodes`. `output_nodes` needs to check whether a node is associated with any graph-level output.

Here is one possible service design that satisfies this need.

```
virtual bool associated(const Node *node) const = 0;
virtual GraphOutputIndex index(const Node *node) const = 0;
```

How to declare a service

All of these service interfaces should inherit `loco::DialectService` interface that *loco.core* defines.

```
struct DialectService
{
  virtual ~DialectService() = default;
};
```

For example, it is possible to declare the service that `output_nodes` needs as follows:

```
struct GraphOutputIndexQueryService : public DialectService
{
  virtual ~GraphOutputIndexQueryService() = default;

  virtual bool associated(const Node *node) const = 0;
  virtual GraphOutputIndex index(const Node *node) const = 0;
};
```

How to access a service

This proposal extends `Dialect` class with `service` method.

Each dialect SHOULD return a valid pointer on `service<Service>` method call if it implements that service. Otherwise, it SHOULD return a null pointer otherwise.

WARNING It is impossible to use `get`. `get` is currently reserved for singleton accessor.

Given a `GraphOutputIndexQueryService`, it is possible to revise `output_nodes` as follows:

```
std::vector<loco::Node *> output_nodes(loco::Graph *g)
{
  std::map<GraphOutputIndex, loco::Node *> table;

  for (uint32_t n = 0; n < g->nodes()->size(); ++n)
  {
    auto node = g->nodes()->at(n);

    if (auto service = node->dialect()->service<GraphOutputIndexQueryService>())
    {
```

(continues on next page)

(continued from previous page)

```
      if (service->associated(node))
      {
        auto output_index = service->index(node);
        assert(table.find(output_index) == table.end());
        table[output_index] = node;
      }
    }
  }

  std::vector<loco::Node *> res;

  for (uint32_t n = 0; n < g->outputs()->size(); ++n)
  {
    auto it = table.find(n);
    // NOTE This behavior originates from the current implementation of output_nodes
    res.emplace_back(it == table.end() ? nullptr : it->second);
  }

  return res;
}
```

PLEASE NOTE THAT `output_nodes` now works with all the dialects that implement `GraphOutputIndexQueryService`.

How to register a service

Each dialect should invoke protected `service` method during its construction.

```
AwesomeDialect::AwesomeDialect()
{
  std::unique_ptr<Impl> impl = ...;
  service<GraphOutputIndexQueryService>(std::move(impl));
}
```

4.4.4 luci

luci provides IR for TFLite/Circle and Graph from FlatBuffer.

4.4.5 luci-export

luci-export provides exporting *loco* graph of Circle IR to Circle model file

4.4.6 luci-import

luci-import provides importing Circle model file to *loco* graph of *luci* Circle Dialect IR

4.4.7 luci-lang

`luci-lang` provides TensorFlow Lite and Circle Dialect IR

4.4.8 luci-logex

luci-logex is a extended logging utility for *luci* compiler framework.

4.4.9 luci-log

luci-log is a logging framework for *luci* compiler framework.

4.4.10 luci-pass

luci-pass provides Circle Dialect transformation passes

4.4.11 luci-service

luci-service provides Circle Dialect Services

4.4.12 Model IR (MIR)

Purpose

This library exposes **NNC**'s model IR to the outer tools (currently `Mirunner`).

Design philosophy

MIR was designed to support a multiple-frontend NN compiler/optimizer.

Function

The high level overview of **MIR** is:

- operations are a composition of their `inputs`, `outputs` and special attributes specific to different operation types.
- operations can have multiple inputs and multiple outputs, each output can be an input to more than one operation (can be used in more than one operation).
- the kernel tensors are represented by `ConstantOp` and are linked to operations via `Input` objects.

Mir has a protobuf serializer/deserializer for shapes and tensors (see `mir.proto` schema).

For list of currently supported operations, see `mir/ops/operations.lst.h`.

How to use

Can be included as a `CMake` target.

TODO

- Expand serialization
- Add More to readme

Dependencies

Mir depends on `adtitas` library, which provides the `small_vector` data type.

4.4.13 moco

moco provides building blocks to load and process TensorFlow models and to produce graph of loco canonical IR

4.4.14 moco-import

moco-import provides importing TensorFlow model file to *moco* TensorFlow Dialect IR

4.4.15 lang

`lang` provides TensorFlow Dialect IR

4.4.16 pass

pass provides *moco* General Graph Passes for Transformation and Optimization

4.4.17 service

`service` provides TensorFlow Dialect Services

4.4.18 support

support privides *moco* support libraries

4.4.19 oneco

4.5 Interpreter

4.5.1 locomotiv

locomotiv is a reference interpreter for *loco* IR.

4.5.2 Purpose

- *locomotiv* would serve as code level specification and reference implementation for loco IR.
- *locomotiv* is required for loco-related tools to be tested.

4.5.3 Sample code to use locomotiv library

This sample code shows how to use locomotiv. Please refer to `src/Session.test.cpp` as well for actual usage.

```
template <typename T> using Buffer = nncc::core::ADT::tensor::Buffer<T>

loco::Graph *graph;
// ... building graph ...

// Open interpreter session
locomotiv::Session sess(graph);

for (uint32_t i = 0; i < s.input_size(); ++i)
{
  Buffer<type> buffer;
  // ... building buffer ...

  locomotiv::NodeData input_data = locomotiv::make_data(buffer);

  sess.set_input(i, input_data);
}

// Run inference
sess.infer();

// Query inferred output
locomotiv::NodeData *output_data = sess.get_output(query_index);

// Get buffer according to data type
switch(output_data->dtype())
{
case loco::DataType::S32:
{
  Buffer<int32_t> output_buffer = output_data->as_s32_bufptr();
  // Do something
  break;
}
case loco::DataType::FLOAT32:
{
  Buffer<float> output_buffer = output_data->as_f32_bufptr();
  // Do something
  break;
}
// ...
}
```

4.5.4 How to support new loco node execution: recommended guide

Steps to support new loco node

1. First of all, understand semantics of the node to newly support, especially on calculation spec and valid use cases.
2. Add the node to `locomotiv/src/Node.lst`. Please keep alphabetical order. This automatically declares `NodeExecution::execute(TheNode *)` and updates `NodeExecution::run()` to deal with the node.

3. Define `execute(loco::TheNode *)` at `locomotiv/src/Node/TheNode.cpp`.
4. Test new node execution at `locomotiv/src/Node/TheNode.test.cpp` if possible.

Note on internal data layout rule

For each domain(see `loco::Domain`), `locomotiv` has fixed layout rule on how to store its data in memory.

- Feature is represented as NHWC layout
  - That is number of batch(N), height(H), width(W) and channel depth(C)
- Filter is represented as NHWC layout
  - That is number of filter(N), height(H), width(W) and input channel depth(C)
- DepthwiseFilter is represented as HWCM layout
  - That is height(H), width(W), input channel depth(C) and depth multiplier(M)
- Matrix is represented as HW layout
  - That is height(H), width(W)

Notes on step 3

- Mocking Tensorflow lite `reference_op.h` might be a good place to start.
- `execute()` can be called multiple time. It just recalculates and updates annotated data. So it should `erase_annot_data()` before newly `annot_data()`.
- Most node execution behaviour would be implemented for each data type.
- `execute()` should throw runtime error on invalid cases. Some of these cases are explained:
  - Invalid argument node
    - e.g.) Pull -> MaxPool2D is invalid as MaxPool2D requires feature map as its argument.
  - Lack of argument data
    - e.g.) Given 'Pull -> Push' graph. On execution of Push, if no NodeData annotated to Pull, it is invalid.
  - Mismatch of argument shapes
    - e.g.) Addition between 2x2 and 3x3 tensor is invalid
    - e.g.) MaxPool2D expects its ifm to be 4D feature, otherwise invalid.
  - Mismatch between node's own information and inferred information
    - Some node already have attributes like shape or data type. If inferred information is different with existing node's, it is invalid.

Recommendation on step 4 (test)

- If the node has no arguments, create a node object and `NodeExecution::run()` on it. Check whether it operates correctly.
- If the node has N(>= 1) arguments, make N pull node inputs, source them to the node to be tested. FeatureEncode or FilterEncode node may be required inbetween depending on the node's argument type. Then annotate N pull nodes with its data, `NodeExecution::run()` on the node to test, and check whether it operates correctly.

4.5.5 mir-interpreter

4.6 Libraries

4.6.1 adtidas

4.6.2 angkor

Purpose

angkor is a `nncc` core library

How to use

angkor implements abstract data type(ADT) for feature, kernel, tensor. There are layout, shape information and enumerator and so on.

To use some of these things, just insert `include`!

```
#include <nncc/core/ADT/feature/WHAT_YOU_WANT>
#include <nncc/core/ADT/kernel/WHAT_YOU_WANT>
#include <nncc/core/ADT/tensor/WHAT_YOU_WANT>
```

Example

- `compiler/coco/core/CMakeLists.txt`

```
target_link_libraries(coco_core PUBLIC angkor)
```

- `compiler/coco/core/src/IR/Arg.cpp`

```
#include "coco/IR/Arg.h"

#include <nncc/core/ADT/tensor/LexicalLayout.h>
#include <nncc/core/ADT/tensor/IndexEnumerator.h>

namespace
{
const nncc::core::ADT::tensor::LexicalLayout l;
}

namespace coco
{

Arg::Arg(const nncc::core::ADT::tensor::Shape &shape) : _shape{shape}, _bag{nullptr}
{
  _map.resize(nncc::core::ADT::tensor::num_elements(shape));
}

// ....

}
```

4.6.3 bino

Let's manipulate std::pair values with UNIX pipe-like syntax.

NOTE The *bino* originates from a binocular telescope.

4.6.4 cli

`cli` is a CLI (Command Line Interface) application framework.

4.6.5 Background

Many tools in `nncc` are command-line interface (CLI) applications. They generally need to handle command line parameters. `cli` was written to reduce code duplication across such applications.

4.6.6 How to use

Please refer to `cli/src/App.test.cpp` for an example.

4.6.7 cwrap

cwrap is a collection of C++ wrappers for POSIX C API.

How to use

Currently it supports only file descriptor.

Example

- File Descriptor

```
cwrap::Fildes fildes{open(path.c_str(), O_RDONLY)};

if (fildes.get() < 0)
{
    std::ostringstream ostr;
    ostr << "Error: " << path << " not found" << std::endl;
    throw std::runtime_error{ostr.str()};
}

google::protobuf::io::FileInputStream fis(fildes.get());
```

4.6.8 enco-intf

4.6.9 fipe

4.6.10 hermes

An **extensible** logging framework

4.6.11 hermes-std

hermes-std is a collection of **primitive** *hermes* extensions.

4.6.12 kuma

kuma is a collection of offline memory allocators.

What does "kuma" mean?

kuma originates from *cooma* which is an abbreviation of **C**ollection **O**f **O**ffline **M**emory **A**lloators.

4.6.13 locop

locop is a collection of *loco* pretty printers.

4.6.14 mio-circle

Let's make it easy to read and write Circle models.

4.6.15 mio-tf

mio-tf provides a library to access TensorFlow model files

4.6.16 mio-tflite

mio-tflite provides a library to access TensorFlow lite model files

4.6.17 moco-log

moco-log is a logging framework for *moco* compiler framework.

4.6.18 morph

`morph` is a collection of shape conversion routines for various NN frameworks, such as Caffe.

4.6.19 nest

nest is a lightweight nested loop generation library, which makes it easy to generate complex, optimized nested loops (such as loops in conv2d).

References

- Halide
- Tensor Comprehension

4.6.20 nike

nike is a collection of **numeric** value comparison routines.

- *nike* is a combination of two words: *numeric* and *dike*. FYI, *dike* is the goddess of justice in ancient Greek culture.

4.6.21 nnop

4.6.22 nnsuite

4.6.23 oops

4.6.24 pepper-assert

4.6.25 pepper-env

pepper-env makes it easy to access "process environment variables".

4.6.26 pepper-str

Let us simulate string interpolation in C++!

HOW TO USE

```
#include <pepper/str.h>

int main(int argc, char **argv)
{
  std::cout << pepper::str("There are ", argc, " arguments") << std::endl;
  return 0;
}
```

4.6.27 pepper-strcast

pepper-strcast is a collection of string-to-value casting functions.

4.6.28 plier-tf

plier-tf is a collection of small tools to handle TensorFlow model.

4.6.29 pp

`pp` is a library which provides various helper functions and classes for pretty-printing. This was originted while writing C/C++ code generator.

4.6.30 Function (Feature)

With `pp`, the following can be built:

- multi-line structure with easy indentation, where each line can be accessed by index
- indented string
- concating `string`, `int`, etc., without user's explicit type conversion
- multi-line string an so on.

4.6.31 How to use

- Some of examples are listed below:
  - `pp::fmt`

```
std::cout << pp::fmt("Hello ", 2) << "\n"; // "Hello 2"
std::cout << pp::fmt("Hello ", "Good ", "World") << "\n"; // ""Hello Good␣
↪World"
```

  - `pp::IndentedStringBuilder`

```
pp::IndentedStringBuilder builder{};

std::cout << builder.build("A") << "\n"; // "A"
builder.increase();
std::cout << builder.build("B") << "\n"; // "  B"
builder.decrease();
std::cout << builder.build("C") << "\n"; // "C"
```

  - For more usage and examples, please refer to `*.test.cpp` under `pp/src`.

4.6.32 safemain

4.6.33 stdex

`stdex` is an extension over standard C++ libraries.

4.6.34 How to use

Please read each header files.

One example of `stdex::make_unique(..)` in `compiler/stdex/Memory.h` is as follows:

```
#include <stdex/Memory.h>

using stdex::make_unique;

class A { ... };

...

std::unique_ptr<A> a = make_unique<A>(); // Note: std::make_unique is not supported␣
↪in C++ 11
```

4.6.35 v4tf

What is this?

v4tf is is a wrapper interface to use TensorFlow by its C API. The name was originated from the movie, *V for Vendetta*, where the main character *V* hides his face by wearing a mask.

Why do we need this?

In *nncc*, some tests use TensorFlow, which uses Protocol Buffers. For example, TensorFlow 1.12 uses Protocol Buffers 3.5.2.

Some of *nncc* modules use different version Protocol Buffers for internal purpose. If such modules also try to use TensorFlow API, errors were thrown due to resolution of wrong symbols of different versions of Protocol Buffers.

To prevent these errors, *v4tf* loads TensorFlow dynamically with all of its symbols resolved.

4.6.36 ann-api

4.6.37 ann-ref

ann-ref is a reference Android NN API implementation for Linux.

DISCLAIMER

ann-ref is incomplete in terms of its functionalities.

4.6.38 dredd-rule-lib

dredd-rule-lib is a library that defines functions to run *dredd* tests, which checks non-functional aspect of compiled files.

Terms

Assume that we want to check the size of generated tflite file to be less than 1024 Bytes. In such case, we'd like to use the following terms:

- "metric" : *file size*
- "rule" : *file size < 1024*
- "metric function": `file_size` that returns size of a compiled tflite file

Models (input of test) exist in *model repo*, where

- "model repo" : directory where models exist. For *tf2tflite-dredd-pbtxt-test*, model repo is `res/TensorFlowTests`.

Metrics supported

The following metric functions are provided:

- `all_op_count` : the count of operations inside a compiled tflite file
- `file_size` : the size of compiled tflite file

- In addition, `op_count`, `conv2d_weight_not_constant`, etc.
- Please , refer to `rule-lib.sh` for metric functions

Related projects - *dredd* tests

Four *dredd* test projects use *dredd-rule-lib*:

- *tf2tflite-dredd-pbtxt-test*
  - Models in `pbtxt`, text file, are compiled into `tflite` file.
  - Then `rule` file that each model has is checked against the `tflite` file.
- *tf2tflite-dredd-pb-test*
  - Models in `pb`, binary file, are compiled into `tflite` file.
  - Then `rule` file that each model has is checked against the `tflite` file.
- *tf2circle-dredd-pbtxt-test*
  - Models in `pbtxt`, text file, are compiled into `circle` file.
  - Then `rule` file that each model has is checked against the `circle` file.
- *tf2circle-dredd-pb-test*
  - Models in `pb`, binary file, are compiled into `circle` file.
  - Then `rule` file that each model has is checked against the `circle` file.

Rule file

To be a target of *dredd*-tests, a `.rule` file **must** exist in a model directory. Please refer to `res/TensorFlowTests/NET_0025/tflite_1.0_rel_requirement.rule` for an example.

Naming convention of rule file

Note that the file name `tflite_1.0_rel_requirement.rule` is our convention containing the information below:

- Generated file type (`tflite`)
- SDK version (`1.0_rel`)
- Purpose (`requirement`)

How do all these work?

For *tf2tflite-dredd-pbtxt-test*, (*tf2circle-dredd-pbtxt-test* works similarly)

```
model repo                                    tf2tflite-dredd-pbtxt-test
-------------------------------------------------------------------------------------
↪---------
   NET_0025
    ├── test.pbtxt  ---------------------->  converted to NET_0025.pb, and then NET_
↪0025.tflite
    |                                       /|\
```

(continues on next page)

(continued from previous page)

```
    ├── test.info --------------------------+
    |   (input/output info of model)
    |
    └── tflite_1.0_rel_requirement.rule -->  running rule file against tflite -->␣
↪pass or fail
                                                          /|\
                          dredd-rule-lib                   | (using)
                      ----------------------               |
                          rule-lib.sh                      |
                            - defining rule function --+
```

For *tf2tflite-dredd-pb-test*, (*tf2circle-dredd-pb-test* works similarly)

```
model repo                                    tf2tflite-dredd-pb-test
-------------------------------------------------------------------------------------
↪---------
   Inception_v3
    ├── model.pb  ------------------------>  converted to Inception_v3.tflite
    |                                      /|\
    ├── model.info --------------------------+
    |   (input/output info of model)
    |
    └── tflite_1.0_rel_requirement.rule -->  running rule file against tflite -->␣
↪pass or fail
                                                          /|\
                          dredd-rule-lib                   | (using)
                      ----------------------               |
                          rule-lib.sh                      |
                            - defining rule function --+
```

Model repo and How to add a model as a target of a *dredd*-test.

For *tf2tflite-dredd-pbtxt-test* and *tf2circle-dredd-pbtxt-test*, model repo is `res/TensorFlowTests`.

To add a model into these tests, the model directory name should be added into one of the following files:

- `test.lst` : This file resides in git
- `test.local.lst` : This file is ignored by git. Use this for personal purpose.

For *tf2tflite-dredd-pb-test* and *tf2circle-dredd-pb-test*, model repo is `tf2tflite-dredd-pb-test/contrib` and `.tf2circle-dredd-pb-test/contrib` respectively.

Use these tests for binary models in large size.

To add a model into these tests, the model directory name should be added into the following file:

- `contrib.lst` : This file is ignored by git.

4.6.39 gen-core

gen-core is a common library used by *gen-tf-input*, *gen-tf-output*, and *gen-tflite-output*.

4.6.40 nnkit-caffe

4.6.41 nnkit-intf

nnkit-intf provides **basic** interface classes for *nnkit* backend/action.

4.6.42 nnkit-misc

nnkit-misc includes various helpers that make it easy to implement *nnkit* extensions and tools.

4.6.43 nnkit-mocotf

4.6.44 nnkit-onnxrt

4.6.45 nnkit-tf

4.6.46 nnkit-tflite

4.6.47 tfinfo

This dir contains a helper classes to handle `test.info` files under `res/TensorFlowTests`.

Format of 'test.info' file

Each line should contain the following fields:

- `input` or `output`
- node_name:digits
- type (see enum TF_DataType in tensorflow/c/c_api.h)
- [shapes]
  - In case of scalar, use '[ ]' as shape

4.6.48 tfinfo-v2

4.7 Tools

4.7.1 Utils

Caffe model generation helpers

REQUIRES:

- caffe
- h5py
- lmdb
- numpy

- caffegen in `$PATH`

`GenerateCaffeModels.py` creates `*.prototxt` files for 1 and 2 layer caffe models The generator can create multiple examples of any layer, assuming you add a `how_many` field into the layer's dict. You will also need to replace the constants in said dict with `PH(type, param)` values, where `type` is the type of the placeholder variable and `params` is a list (or tuple) of paramenters for generating the mock.

For an example of generating multiple instances of a layer see the `Log` layer.

`Filler.sh` fills a single model with random weights by using `caffegen` and creates a dir with a filled `prototxt` and a `caffemodel` binary file. The result directory is located in the same directory as the `prototxt` file

`AllFill.sh` fills all `*.prototxt` files in the current directory or in provided directory (-d)

These scripts can be useful for developing/testing nnc. Usage and purpose of the scripts can be found in comments in their source code.

Note that these scripts are just development artifacts and are not supposed to go into production in any form.

infer_testcases.py: run inference with `nnkit` on testcases res2bin.py: used by infer_testcases.py to convert resulting hdf5 to binary format

'testcases' folder structure: At the moment we use the following structure: a folder for a model contains 'models' and 'testcases' subfolders. The 'models' subfolder contains model that we run inference on, 'testcases' subfolder contains a 'testcase*' folder for each different testcase. Each of those folders in turn contain 'input' with a '.JPEG' file (and '.hdf5' and '.dat' files after running `jpeg2hdf5` script), and 'output' folder where inference results are stored.

4.7.2 here I write how I run model on my computer

sections: a) goal of this script b) examples of code running in author's local machine c) parametrs and short comment

---

goal of this script

Here the author has attempted to implement a program capable of running any of the 4 models (caffe, caffe2, tflite, onnx) in a simple and user-friendly manner. The goal of the program is to get the file containing the output of the computation graph at the program output.

---

examples of code running in author's local machine

The purpose of the examples below is to demonstrate which arguments and in which order you should use to run this script correctly.

caffe:

```
$ python3 model_runner.py -m  caffe1_runer/inception-v3_ref.caffemodel  caffe1_runer/
↪inception-v3_ref.prototxt  -i caffe1_runer/ILSVRC2012_val_00000002.JPEG.tfl.hdf5
```

caffe2:

```
$ python model_runner.py -m  caffe2_runer_and_photo/caffe2_models/init_net.pb  caffe2_
↪runer_and_photo/caffe2_models/predict_net.pb -i randomInput.hdf5
```

tflite:

```
$ python model_runner.py -m  tflite_runer_and_photo/TST-1-2\ AVARAGE_POOP_2D.tflite -
↪i  tflite_runer_and_photo/in.hdf5
```

onnx:

```
$ python model_runner.py  -m onnx_runer/model.onnx -i RANDOM.hdf5
```

---

parametrs and short comment

-m mean pre learned model which you run -i mean model's input

These scripts can be useful for developing/testing nnc. Usage and purpose of the scripts can be found in comments in their source code.

Note that these scripts are just development artifacts and are not supposed to go into production in any form.

jpeg2hdf5.py: prepare '.hdf5' files from '.JPEG' to be used by nnkit. Can also convert those '.JPEG's to binary format along the way.

'testcases' folder structure: At the moment we use the following structure: a folder for a model contains 'models' and 'testcases' subfolders. The 'models' subfolder contains model that we run inference on, 'testcases' subfolder contains a 'testcase*' folder for each different testcase. Each of those folders in turn contain 'input' with a '.JPEG' file (and '.hdf5' and '.dat' files after running `jpeg2hdf5` script), and 'output' folder where inference results are stored.

4.7.3 caffegen

`caffegen` is a tool for generating caffe model and decoding binary file of caffe model

How caffegen works

Some of commands in `caffegen` use standard input for reading data and standard output for exporting result. In this case, we strongly recommand you to use pipe, not copy & paste the content of file itself.

Otherwise, `caffegen` use arguments to pass some directories.

Supported command

Basically, caffgen command is used as `caffegen [COMMAND]` and there are four `COMMAND` types.

- init : initialize parameters using prototxt.
- encode : make a binary file(caffemodel) using initialized data
- decode : decode a binary file(caffemodel) and reproduce the initialized data
- merge : copy the trained weights from a caffemodel into a prototxt file

How to use each command

1. Init (Using stdin and stdout)

- `./build/compiler/caffegen/caffegen init`
  - Type the prototxt by yourself
  - Then you can get the result on the shell.
- `cat ./res/BVLCCaffeTests/Convolution_000/test.prototxt | ./build/compiler/caffegen/caffegen init`
  - Prototxt will be automatically passed
  - Then you can get the result on the shell.

1. Encode (Using stdin and stdout)

- `./build/compiler/caffegen/caffegen encode`
  - Type the initialized data by yourself
  - Then you can get the result on the shell.
- `cat ./res/BVLCCaffeTests/Convolution_000/test.prototxt | ./build/compiler/caffegen/caffegen init | ./build/compiler/caffegen/caffegen encode > Convolution_000.caffemodel`
  - The initialized data will be automatically passed
  - The encoded result will be automatically saved in caffemodel file

1. Decode (Using stdin and stdout)

- `cat Convolution_000.caffemodel | ./build/compiler/caffegen/caffegen decode`
  - Caffemodel file will be automatically passed
  - Then you can get the result on the shell

1. Merge (Using arguments)

- `./build/compiler/caffegen/caffegen merge ./res/BVLCCaffeTests/Convolution_000/test.prototxt Convolution_000.caffemodel`
- `./build/compiler/caffegen/caffegen merge ./res/BVLCCaffeTests/Convolution_000/test.prototxt Convolution_000.caffemodel > Convolution_000.merged`

4.7.4 circle-inspect

circle-inspect allows users to retrieve various information from a Circle model file

Information to inspect

Operators with `--operators`

- show operator codes one line at a time in execution order

Example

```
$ circle-inspect --operators model.circle
```

Result

```
RESHAPE
DEPTHWISE_CONV_2D
ADD
```

To get the count of specific operator, use other tools like sort, uniq, etc.

4.7.5 circle-verify

circle-verify allows users to verify Circle models.

Usage

Provide *circle* file as a parameter to verify validity.

```
$ circle-verify circlefile.circle
```

Result for valid file

```
[ RUN       ] Check circlefile.circle
[      PASS ] Check circlefile.circle
```

Result for invalid file

```
[ RUN       ] Check circlefile.circle
[      FAIL ] Check circlefile.circle
```

4.7.6 circlechef

What is circlechef?

Do you need a circle model for testing? Ask it to *circlechef*. Given a recipe, *circlechef* will cook a circle model for you.

NOTE *circlechef* covers only what *tflchef* does not cover. This is to support ops that exist only in circle shema, and other things can be made using *tflchef* and *tflite2circle*.

4.7.7 circledump

What is this?

circledump is a tool that dumps binary circle file into human readable text to console.

circledump is implemented with C++ not python. We can do the same thing much easier with python but this tool doesn't need to install TensorFlow python package.

Schema for FlatBuffer used is from TensorFlow v1.13.1 release.

Design philosophy

Make the code simple.

To do

- Print weight values other than uint8_t
- Add more operators

How to use

Command argument format:

```
circledump circle_file
```

Example output of dump `readme.circle` file

```
Dump: readme.circle

Data Format:
CHANNEL_LAST (NHWC for 2d, NDHWC for 3d data)

Operator Codes: [order] OpCodeName (OpCode Enum)
[0] CONV_2D (code: 3)

Buffers: B(index) (length) values, if any
B(0) (0)
B(1) (8) 0x94 0x5b 0x95 0xbf 0x42 0xa4 0x52 0xbf ...
B(2) (4) 0xcd 0xcc 0x8c 0x3f

Operands: T(tensor index) TYPE (shape) B(buffer index) OperandName
T(0) FLOAT32 (1, 3, 3, 2) B(0) ifm
T(1) FLOAT32 (1, 1, 1, 2) B(1) ker
T(2) FLOAT32 (1) B(2) bias
T(3) FLOAT32 (1, 3, 3, 1) B(0) ofm

Operators: O(operator index) OpCodeName
    Option(values) ... <-- depending on OpCode
    I T(tensor index) OperandName <-- as input
    O T(tensor index) OperandName <-- as output
O(0) CONV_2D
    Padding(1) Stride.W(1) Stride.H(1) Activation(0)
    I T(0) ifm
    I T(1) ker
    I T(2) bias
    O T(3) ofm

Inputs/Outputs: I(input)/O(output) T(tensor index) OperandName
I T(0) ifm
I T(1) ker
O T(3) ofm
```

Dependency

- mio-circle
- safemain
- stdex
- FlatBuffers

4.7.8 encodump

encodump is a dumper for coco IR generated by enco

How to use

Sources for *encodump* are:

1. enco frontend library `*.so` file
2. model description file for matching to enco frontend

```
$ path/to/encodump \
         --frontend [enco frontend library .so file]
         --frontend-arg [model file] ...
```

Currently supported enco frontends are Caffe and tensorflow lite. For Caffe, both `*.prototxt` and `*.caffemodel` are required, and for TFlite, `*.tflite` flatbuffers file is required.

Output is dumped into terminal.

Example

```
nncc$ ./build/compiler/encodump/encodump \
         --frontend ./build/compiler/enco/frontend/tflite/libenco_tflite_frontend.so \
         --frontend-arg ./build/compiler/enco/test/tflite/Conv2D_000.tflite
```

Output:

```
<Module>
  <Block> (index: 0)
    <Inst>:
      Eval (0x10cfa90)
        out: 0x10cf960
        <op>:
          Load(0x10cf600, obj: 0x10cd670)
          Conv2D(0x10cf8a0, ker obj: 0x10cf2d0, padding [T/B/L/R=0,0,0,0], stride [V/
↪H = 1,1])
    <Inst>:
      Eval (0x10cff80)
        out: 0x10cfb20
        <op>:
          Load(0x10cfe70, obj: 0x10cfcc0)
          Load(0x10cfdd0, obj: 0x10cf960)
          Add
    <Inst>:
```

(continues on next page)

(continued from previous page)

```
      Copy (0x10d0120)
        from: 0x10cfb20
        into: 0x10cfff0
    <Inst>:
      Copy (0x10d01f0)
        from: 0x10cfff0
        into: 0x10cf210
  <Input>: bag 0x10ce650, name=ifm
  <Output>: bag 0x10ce9c0, name=ofm
  <Bag>:
    0x10ce650, obj: [0x10cd670], size: 18, input, const, reader: [x], updater: [x],
    0x10ce770, obj: [0x10cf2d0], size: 2, const, reader: [x], updater: [x],
    0x10ce890, obj: [0x10cfcc0], size: 1, const, reader: [x], updater: [x],
    0x10ce9c0, obj: [0x10cf210], size: 9, output, const, reader: [x], updater: [x],
    0x10cf9d0, obj: [0x10cf960], size: 9, const, reader: [x], updater: [x],
    0x10cfbe0, obj: [0x10cfb20], size: 9, const, reader: [x], updater: [x],
    0x10d0060, obj: [0x10cfff0], size: 9, const, reader: [x], updater: [x],
  <Object>:
    0x10cd670, bag: 0x10ce650, kind: Feature, Shape [H/W/D=3,3,2], producer: x,␣
↪comsumer: [op: 0x10cf600]
    0x10cf210, bag: 0x10ce9c0, kind: Feature, Shape [H/W/D=3,3,1], producer: instr:␣
↪0x10d01f0, comsumer: [x]
    0x10cf2d0, bag: 0x10ce770, kind: Kernel, Shape [N/H/W/D=1,1,1,2], producer: x,␣
↪comsumer: [op: 0x10cf8a0]
    0x10cf960, bag: 0x10cf9d0, kind: Feature, Shape [H/W/D=3,3,1], producer: instr:␣
↪0x10cfa90, comsumer: [op: 0x10cfdd0]
    0x10cfb20, bag: 0x10cfbe0, kind: Feature, Shape [H/W/D=3,3,1], producer: instr:␣
↪0x10cff80, comsumer: [inst: 0x10d0120]
    0x10cfcc0, bag: 0x10ce890, kind: Feature, Shape [H/W/D=3,3,1], producer: x,␣
↪comsumer: [op: 0x10cfe70]
    0x10cfff0, bag: 0x10d0060, kind: Feature, Shape [H/W/D=3,3,1], producer: instr:␣
↪0x10d0120, comsumer: [inst: 0x10d01f0]
```

4.7.9 gen-tf-input

gen-tf-input generates random input data for testing in HDF5 format.

4.7.10 How to use

Use the following to generate a file that contains random values of input tensors:

```
$ gen-tf-input <info_v2_path> <pb_path> <file_path_to_generate>
```

4.7.11 gen-tf-output

gen-tf-output generates a file containing the result of running TensorFlow in HDF5 format.

4.7.12 How to use

Use the following:

```
$ gen-tf-output <info_v2_path> <pb_path> <input_of_TensorFlow_path> <output_path_to_
↪generate>
```

Use *gen_tf_input* to generate `<input_of_TensorFlow_path>` file.

4.7.13 gen-tflite-output

gen-tflite-output generates a file containing the result of running TensorFlow Lite interpreter in HDF5 format.

4.7.14 How to use

Use the following:

```
$ gen-tflite-output <tflite_file_path> <input_file_path> <output_path_to_generate>
```

Use *gen_tf_input* to generate `<input_file_path>` file.

4.7.15 i5diff

i5diff compares two HDF5 files that *nnkit* HDF5 export action generates.

DISCLAIMER *i5diff* is not designed as a general diff tool. It works only for HDF5 files that *nnkit* HDF5 export action generates.

Yet Another Diff?

i5diff is able to detect *shape mismatch* that *h5diff* cannot detect.

To be precise, *h5diff* is also able to detect *shape mismatch*. Unfortunately, however, *h5diff* ends with 0 exitcode in the presence of *shape mismatch*, and thus it is impossible to use *h5diff* for continuous integration.

How to use

```
$ /path/to/i5diff -d 0.001 /path/to/fst.h5 /path/to/snd.h5
```

4.7.16 nnkit

`nnkit` is collection of neural networks tools for our *nncc* project. This tool is mostly used for testing.

4.7.17 Purpose

For testing, we need to have

- a tool to run existing framework such as Tensorflow for expected tensor result — (1)
- a tool to run our implementation for actual tensor result — (2)

`nnkit` provides a flexible framework to get expected and actual result.

4.7.18 Design

Requirements to address:

- Input
  - Same randomized input is used for both of (1) and (2)
  - Expect tensor layout (e.g., NHWC) could be different for (1) and (2)
- Input and output format
  - Results of (1) and (2) have same file format and data format

For (1), `nnkit` designed to enable the following:

- Input of `nnkit` is randomized and saved into a file in a specific format
- Existing framework such as Tensorflow can run with input tensors that is properly translated
- Result is written into a file in a specific format

For (2), `nnkit` designed to enable the following:

- Data of `nnkit` in a file by (1) is used as input
- Our implementation can run with input tensors that is properly translated
- Result is written into a file in a specific format

`nnkit-run`

`nnkit-run` is a command line interface to interact with existing inference engines or compiled artifacts.

How `nnkit-run` works

`nnkit-run` first dynamically loads `backend` and multiple pre/post `action` specified by command-line. After loading backend and actions, `nnkit-run` requests `backend` to prepare itself. When backend is prepared, `backend` exposes its internal state to `nnkit-run` (as `nnkit::TensorContext`). `nnkit-run` takes this state, and passes it to registered pre `action`(s). Each action may read tensor(s) (e.g. dump the content into a file), or manipulate their value (e.g. fill random values). `nnkit-run` then invokes `backend` through `run()` method. After successful running the `backend`, post `action`(s) are called same like pre `action`(s) as a teardown step.

Backends

In 2019 there will be the following backends as of writing this document

- Backends for the existing framework:
  - Caffe as `libnnkit_caffe_backend.so`
  - Tensorflow Lite as `libnnkit_tflite_backend.so`
  - Tensorflow as `libnnkit_tf_backend.so`
  - Onnx as `libnnkit_onnx_backend.so`
- Backends for our implementation:
  - Moco Tensorflow (TBD)
  - Moco Onnx (TBD)

4.7.19 How to use

How to run inference with nnkit-run

To run `nnkit-run`, we need to provide a backend module and argument(s) if required and optional `pre-` or `post-` action module(s)

How to pass arguments

Syntax is `--argument` with `value` form. Existing arguments are as follows.

- `--backend` [Backend module path]. Only one is needed.
- `--backend-arg` [Backend argument]. Argument(s) for the backend.
- `--pre` [Pre-Action module path]. Multiple Pre-Action can be given.
- `--pre-arg` [Pre-Action argument]. Set argument(s) for the pre-action just before.
- `--post` [Post-Action module path]. Multiple Post-Action can be given.
- `--post-arg` [Post-Action argument]. Set argument(s) for the post-action just before.

For example,

```
nnkit-run \
--backend ./path/to/backend --backend-arg arg1 --backend-arg arg2 \
--pre ./path/to/preA --pre-arg arg1preA --pre-arg arg2preA \
--pre ./path/to/preB --pre-arg arg1preB --pre-arg arg2preB \
--post ./path/to/postA --post-arg arg1postA
```

This will run

- backend `./path/to/backend` with arguments `arg1 arg2` with
  - pre-action `./path/to/preA` with arguments `arg1preA arg2preA`,
  - pre-action `./path/to/preB` with arguments `arg1preB arg2preB` and
  - post-action `./path/to/postA` with an argument `arg1postA`

Example : Running with Tensorflow backend

To run Tensorflow backend, you need two parameters: model file in protobuf format (`pb` file) and input/output tensor information such as tensor name, data type, shape. Please refer to `test.info` files under `moco/test/tf`.

```
cd build

compiler/nnkit/tools/run/nnkit-run \
--backend ./compiler/nnkit-tf/backend/libnnkit_tf_backend.so \
--backend-arg inceptionv3_non_slim_2015.pb \
--backend-arg inceptionv3_non_slim_2015.info
```

Example: Running with Onnx backend

TBD

Example : Running with tflite backend

```
cd build

compiler/nnkit/tools/run/nnkit-run \
--backend ./compiler/nnkit-tflite/backend/libnnkit_tflite_backend.so \
--backend-arg inceptionv3_non_slim_2015.tflite
```

Example: Running with Caffe backend

Running with caffe backend is similar to running with tflite, except that you need to provide `prototxt` file, `caffemodel` is not necessary, unless you want to use specific weights (weights are random if `caffemodel` is not provided and `prototxt` is not filled with specific weights):

```
cd build

compiler/nnkit/tools/run/nnkit-run \
--backend ./compiler/nnkit-caffe/backend/libnnkit_caffe_backend.so \
--backend-arg inception_v3.prototxt
```

Running with pre & post actions

The above command for the tflite backend shows nothing except `nnapi error: unable to open library libneuralnetworks.so` warning even though running correctly. The following command displays inferenced values.

```
cd build

compiler/nnkit/tools/run/nnkit-run \
--backend ./compiler/nnkit-tflite/backend/libnnkit_tflite_backend.so \
--backend-arg inceptionv3_non_slim_2015.tflite \
--post ./compiler/nnkit/actions/builtin/libnnkit_show_action.so
```

The following command initializes input tensors with random values generated by `RandomizeAction` pre-action.

```
compiler/nnkit/tools/run/nnkit-run \
--backend ./compiler/nnkit-tflite/backend/libnnkit_tflite_backend.so \
--backend-arg inceptionv3_non_slim_2015.tflite \
--pre ./compiler/nnkit/actions/builtin/libnnkit_randomize_action.so \
--post ./compiler/nnkit/actions/builtin/libnnkit_show_action.so
```

Example: Dump HDF5

You can drop a HDF5 file of inputs and outputs with `HDF5_export_action` action module.

```
cd build

compiler/nnkit/tools/run/nnkit-run \
--backend ./compiler/nnkit-tflite/backend/libnnkit_tflite_backend.so \
--backend-arg inceptionv3_non_slim_2015.tflite \
--pre ./compiler/nnkit/actions/builtin/libnnkit_randomize_action.so  \ # randomize␣
↪first
--pre ./compiler/nnkit/actions/HDF5/libnnkit_HDF5_export_action.so \   # then drop␣
↪input in HDF5 format
```

(continues on next page)

(continued from previous page)

```
--pre-arg ./pre.hdf5 \
--post ./compiler/nnkit/actions/HDF5/libnnkit_HDF5_export_action.so \  # drop output␣
↪in HDF5 format
--post-arg ./post.hdf5
```

This will drop `pre.hdf5` and `post.hdf5` files containing input and output tensor of inceptionv3_non_slim_2015.tflite model.

4.7.20 To do

- `nnkit` backend for `moco` Tensorflow frontend
- `nnkit` backend for `moco` Onnx frontend
- `nnkit` backend for Onnx frontend

4.7.21 onnxkit

Purpose

onnxkit allows users to encode/decode ONNX model files.

How to use

Currently it supports two operations, *decode* and *encode*.

```
nncc$ path_to_onnxkit/onnxkit
ERROR: COMMAND is not provided

USAGE: path_to_onnxkit/onnxkit [COMMAND] ...

SUPPORTED COMMANDS:
  decode
  encode
```

`decode` reads a binary graphproto file and shows its textual form.

`encode` is the reverse of decode, it reads a textual graphproto file and prints its binary form.

Each command can read from or print to the console or from/to a file if given through the argument. First argument is used as an input file path and second as a output file path. If second argument is omitted, output is the console. To give the first argument as a console, please use `-`.

Examples

Example to `decode`

```
nncc$ cat my_awesome_model.pb | path_to_onnxkit/onnxkit decode > decoded.pbtxt
```

```
nncc$ cat my_awesome_model.pb | path_to_onnxkit/onnxkit decode - decoded.pbtxt
```

```
nncc$ path_to_onnxkit/onnxkit decode my_awesome_model.pb > decoded.pbtxt
```

```
nncc$ path_to_onnxkit/onnxkit decode my_awesome_model.pb decoded.pbtxt
```

Above four examples for `decode` command gives the same result. This applies to other commands.

Example to `encode`

```
nncc$ cat decoded.pbtxt | path_to_onnxkit/onnxkit encode > encoded.pb
```

Dependency

- onnx
- Protobuf
- cli
- stdex

4.7.22 tfgraph-xform

Let's build TensorFlow "transform-graph" tool without Bazel.

DISCLAIMER Not every transformation is supported.

4.7.23 tfkit

What is tfkit?

`tfkit` is a tool for manipulating TensorFlow model files.

Tutorial: How to use?

Currently it supports two operations, *decode* and *encode*.

```
nncc$ path_to_tfkit/tfkit
ERROR: COMMAND is not provided

USAGE: path_to_tfkit/tfkit [COMMAND] ...

SUPPORTED COMMANDS:
  decode
  encode
  unpack
  pack
```

`decode` reads a binary graphdef file and shows its textual form.

`encode` is the reverse of decode, it reads a textual graphdef file and prints its binary form.

`unpack` decodes tensor value in byte encoded string in `tensor_content` field to human readable list of float values. currently only supports textual graphdef files.

`pack` is the reverse of unpack. this can be used to change the values for debugging. also currently only supports textual graphdef files.

Each command can read from or print to the console or from/to a file if given through the argument. First argument is used as an input file path and second as a output file path. If second argument is omitted, output is the console. To give the first argument as a console, please use `-`.

Examples

Example to `decode`

```
nncc$ cat my_awesome_model.pb | path_to_tfkit/tfkit decode > decoded.pbtxt
```

```
nncc$ cat my_awesome_model.pb | path_to_tfkit/tfkit decode - decoded.pbtxt
```

```
nncc$ path_to_tfkit/tfkit decode my_awesome_model.pb > decoded.pbtxt
```

```
nncc$ path_to_tfkit/tfkit decode my_awesome_model.pb decoded.pbtxt
```

Above four examples for `decode` command gives the same result. This applies to other commands.

Example to `encode`

```
nncc$ cat decoded.pbtxt | path_to_tfkit/tfkit encode > encoded.pb
```

Example to `unpack`

```
nncc$ cat packed.pbtxt | path_to_tfkit/tfkit unpack > unpacked.pbtxt
```

Example to `pack`

```
nncc$ cat unpacked.pbtxt | path_to_tfkit/tfkit pack > packed.pbtxt
```

4.7.24 tfl-inspect

tfl-inspect allows users to retrieve various information from a TensorFlow Lite model files

Information to inspect

–operators

Operators with `--operators`

- show operator codes one line at a time in execution order

Example

```
$ tfl_inspect --operators model.tflite
```

Result

```
RESHAPE
DEPTHWISE_CONV_2D
ADD
```

To get the count of specific operator, use other tools like sort, uniq, etc.

Example

```
$ tfl-inspect --operators inception_v3.tflite | sort | uniq -c
```

Result

```
     10 AVERAGE_POOL_2D
     15 CONCATENATION
     95 CONV_2D
      4 MAX_POOL_2D
      1 RESHAPE
      1 SOFTMAX
```

–conv2d_weight

Conv2D series weight input node type with `--conv2d_weight`

- shows Conv2D series node weight input node type
- Conv2D series: CONV2D, DEPTHWISE_CONV_2D

Example result

```
CONV2D,CONST
DEPTHWISE_CONV_2D,RELU
CONV2D,CONST
```

4.7.25 tfl-verify

tfl-verify allows users to verify TF Lite models.

Usage

Provide *tflite* file as a parameter to verify validity.

```
$ tfl-verify tflitefile.tflite
```

Result for valid file

```
[ RUN       ] Check tflitefile.tflite
[      PASS ] Check tflitefile.tflite
```

Result for invalid file

```
[ RUN       ] Check tflitefile.tflite
[      FAIL ] Check tflitefile.tflite
```

4.7.26 tflchef

What is tflchef?

Do you need a tensorflow lite model for testing? Ask it to *tflchef*. Given a recipe, *tflchef* will cook a tensorflow lite model for you.

NOTE A model that *tflchef* generates is compatible with TensorFlow Lite in TensorFlow v1.12.0 release

Tutorial: How to use?

This example explains how to generate a tensorflow lite model with a single Conv2D operation with a kernel filled with random values generated according to normal (or gaussian) distribution (mean = 0.0f / stddev = 1.0f) and bias with constant values (1.1f) with *tflchef*.

The first step is to write a recipe! Type the following command, and then you may get `sample.recipe`:

```
$ cat > sample.recipe <<END
operand {
  name: "ifm"
  type: FLOAT32
  shape { dim: 1 dim: 3 dim: 3 dim: 2 }
}
operand {
  name: "ker"
  type: FLOAT32
  shape { dim: 1 dim: 1 dim: 1 dim: 2 }
  filler {
    tag: "gaussian"
    arg: "0.0"
    arg: "1.0"
  }
}
operand {
  name: "bias"
  type: FLOAT32
  shape { dim: 1 }
  filler {
    tag: "constant"
    arg: "1.1"
  }
}
operand {
  name: "ofm"
  type: FLOAT32
  shape { dim: 1 dim: 3 dim: 3 dim: 1 }
}
operation {
  type: "Conv2D"
  conv2d_options {
    padding: VALID
    stride_w: 1
    stride_h: 1
  }
  input: "ifm"
  input: "ker"
  input: "bias"
  output: "ofm"
}
```

(continues on next page)

(continued from previous page)

```
input: "ifm"
input: "ker"
output: "ofm"
END
```

Generate `sample.tflite` from `sample.recipe` with one of the following commands:

- With redirection

```
$ cat sample.recipe | tflchef > sample.tflite
```

- Without redirection

```
$ tflchef-file sample.recipe sample.tflite
```

Done :)

4.7.27 tfldump

What is this?

tfldump is a tool that dumps binary tflite file into human readable text to console.

tfldump is implemented with C++ not python. We can do the same thing much easier with python but this tool doesn't need to install TensorFlow python package.

Schema for FlatBuffer used is from TensorFlow v1.12.0 release.

Design philosophy

Make the code simple.

To do

- Print weight values other than uint8_t
- Add more operators

How to use

Command argument format:

```
tfldump tflite_file
```

Example output of dump `readme.tflite` file

```
Dump: readme.tflite

Operator Codes: [order] OpCodeName (OpCode Enum)
[0] CONV_2D (code: 3)

Buffers: B(index) (length) values, if any
B(0) (0)
```

(continues on next page)

(continued from previous page)

```
B(1) (8) 0x94 0x5b 0x95 0xbf 0x42 0xa4 0x52 0xbf ...
B(2) (4) 0xcd 0xcc 0x8c 0x3f

Operands: T(tensor index) TYPE (shape) B(buffer index) OperandName
T(0) FLOAT32 (1, 3, 3, 2) B(0) ifm
T(1) FLOAT32 (1, 1, 1, 2) B(1) ker
T(2) FLOAT32 (1) B(2) bias
T(3) FLOAT32 (1, 3, 3, 1) B(0) ofm

Operators: O(operator index) OpCodeName
    Option(values) ... <-- depending on OpCode
    I T(tensor index) OperandName <-- as input
    O T(tensor index) OperandName <-- as output
O(0) CONV_2D
    Padding(1) Stride.W(1) Stride.H(1) Activation(0)
    I T(0) ifm
    I T(1) ker
    I T(2) bias
    O T(3) ofm

Inputs/Outputs: I(input)/O(output) T(tensor index) OperandName
I T(0) ifm
I T(1) ker
O T(3) ofm
```

Dependency

- safemain
- stdex
- FlatBuffers

4.7.28 tfts

TensorFlow Testcase Service provides various services on the TensorFlow testcases commited in this repo.

4.7.29 circle2circle-dredd-pbtxt-test

It tests the non-functional conditions of the optimized circle binary resulting from circle2circle.

This test basically refers to the *TensorFlowLiteRecipes* resource. So you should add what you want to test to both of the resource and `test.lst`.

Example

```
# TensorFlowLiteRecipes
res/TensorFlowLiteRecipes/BatchMatMulV2_000
├── test.recipe     # What you want to test
└── test.rule       # Non-functional conditions to be satisfied

# test.lst
```

(continues on next page)

(continued from previous page)

```
...
Add(BatchMatMulV2_000 PASS resolve_customop_batchmatmul)
...
```

For more information on the rules, see *dredd-rule-lib* module.

4.7.30 moco-value-pbtxt-test

4.7.31 oneco-value-pbtxt-test

4.7.32 onnx2tflite-integration-test

4.7.33 tf2circle-conversion-test

Run `tf2circle` to `test.lst` and check whether given TF model is able to be converted into Circle model. Write `test.local.lst` for local test list.

4.7.34 tf2circle-dredd-pb-test

TODO write content

4.7.35 tf2circle-dredd-pbtxt-test

TODO write content.

4.7.36 tf2circle-model-test

4.7.37 tf2circle-ui-check

tf2circle-ui-check makes it easy to check what `tf2circle` shows for selected TensorFlow testcases.

HOW TO USE

First of all, create "test.lst" file and add tests of interest. Here is an example of "test.lst"

```
Add(NET_0000)
Add(NET_0001)
```

Run "nncc configure". You may find the below messages if `tf2circle-ui-check` is configured properly:

```
-- Configure TF2CIRCLE-UI-CHECK
-- Build tf2circle-ui-check: TRUE
-- Configure TF2CIRCLE-UI-CHECK - Done
```

Finally, build `tf2circle_ui_check` target and see what happens! If CMake uses "make" as a generator, you may build `tf2circle_ui_check` target via running `./nncc build tf2circle_ui_check`.

4.7.38 tf2circle-value-pbtxt-remote-test

`tf2circle-value-pbtxt-remote-test` does random value test for `.circle` file using remote machine, normally Odroid, which `nnfw` runs on.

Prerequisites

1. Tensorflow library
   - Make sure that Tensorflow library could be found at `nncc configure` step. If there is no Tensorflow library, this test will not be created.
   - If CMake reports TensorFlow library is not found in configure step, even when the library exists, set `TENSORFLOW_PREFIX` to include Tensorflow library like below.

```
$ ./nncc configure -DTENSORFLOW_PREFIX=/path/to/Tensorflow/library
```

   - `TENSORFLOW_PREFIX` should contain Tensorflow library as shown below.

```
TENSORFLOW_PREFIX
    ├ include
    |   ├ tensorflow
    |   |     └ c
    |   |       ├ c_api.h
    |   ├ ...
    |
    ├ lib
    |   ├ libtensorflow.so
    |   ├ ...
    ├ ...
```

2. Runtime Library and Binary files
   - Detailed information is located in here
   - If you build runtime, related files will be produced in `Product/out`. Do not rename or move it.
   - (TBD) Support native build option
3. Remote machine information and test list
   - You should create `test.lst` file first as shown below.
     - Set IP address and username of remote machine using `set` command.
     - Add Tensorflow models which you want to verify, which are in `/res/TensorflowTests/`

```
#--------------- Remote Machine Setting ---------------#
set(REMOTE_IP "xxx.xxx.xxx.xxx")
set(REMOTE_USER "remote_username")

#--------------------- Tests list ----------------------#
add(UNIT_Add_000)
add(UNIT_Add_001)
...
```

   - If any Tensorflow model is added, or if `REMOTE_IP` and `REMOTE_USER` is not given, `tf2circle-value-pbtxt-remote-test` will not be created.
4. (Optional) ssh authentication

- This test uses `ssh` and `scp` commands, and those commands require a password of remote machine whenever they are called. This means that you should enter the password everytime when `ssh` and `scp` require.
- This test resolves the problem by using `ssh-copy-id`, which copies the public key of host machine to `authorized_keys` of remote machine. Because of that, this test will ask the password of remote machine only once, at the first time. This is the only user interaction while running this test.
- If you do not want to interact with system, just do `ssh-copy-id ${REMOTE_USER}@${REMOTE_IP}` in advance, before running this test. Once `ssh-copy-id` is done, there will be no user-interaction action while running the test.

Running

- If you finished prerequisites properly, configuring -> building -> testing steps create cmake test automatically.
- All the related materials will be sent to `REMOTE_WORKDIR` in remote machine. Default value of `REMOTE_WORKDIR` is `CVT_YYMMDD_hhmmss`, which means Circle Value Test done on YY/MM/DD at hh:mm:ss.
- `REMOTE_WORKDIR` will not be removed automatically after this test finish.

```
$ ./nncc configure && ./nncc build

# Default REMOTE_WORKDIR is CVT_YYMMDD_hhmmss folder
$ ./nncc test -R tf2circle_value_pbtxt_remote_test

# You can set REMOTE_WORKDIR where you have write privilege
$ REMOTE_WORKDIR=/path/you/want/ ./nncc test -R tf2circle_value_pbtxt_remote_test
```

Generated Files While Running

- All related files(`pb`, `circle`, `h5` ... etc.) are created in `build/compiler/tf2circle-value-pbtxt-remote-test` folder.

```
build/compiler/tf2circle-value-pbtxt-remote-test
    ├ Result_latest -> Result_YYMMDD_hhmmss.csv
    ├ Result_YYMMDD_hhmmss.csv
    ├ ...
    |
    ├ UNIT_Add_000
    |     ├ metadata
    |     |     ├ MANIFEST
    |     |     └ tc
    |     |        ├ expected.h5
    |     |        └ input.h5
    |     └ UNIT_Add_000.circle
    |
    ├ UNIT_Add_000.circle
    ├ UNIT_Add_000.expected.h5
    ├ UNIT_Add_000.info
    ├ UNIT_Add_000.input.h5
    ├ UNIT_Add_000.log
    ├ UNIT_Add_000.passed
    ├ UNIT_Add_000.pb
    ├ UNIT_Add_000.pbtxt
```

(continues on next page)

(continued from previous page)

```
    |
    ├ ...
```

- `nnpkg_test.sh`, runtime products and each nnpackage are sent to `REMOTE_WORKDIR` in remote machine.
- (TBD) Modify script not to remove obtained h5 file.

```
REMOTE_WORKDIR
    ├ nnpkg_test.sh
    |
    ├ Product
    |     └ out
    |        ├ bin
    |        ├ lib
    |        ├ ...
    |
    ├ UNIT_Add_000
    |     ├ metadata
    |     |     ├ MANIFEST
    |     |     └ tc
    |     |        ├ expected.h5
    |     |        ├ input.h5
    |     |        └ UNIT_Add_000.out.h5
    |     |          (Only when comparing with expected.h5 fails)
    |     |
    |     └ UNIT_Add_000.circle
    ├ ...
```

Check Test Result

- Summary of test result will be created as csv file in host.

```
# Result_latest is symbolic link to the latest csv result file
# Print the latest test result
$ cat build/compiler/tf2circle-value-pbtxt-remote-test/Result_latest
TEST_NAME, TF2CIRCLE, CIRCLE_VALUE_TEST
UNIT_Add_000, TRUE, TRUE
...

# List all result csv files
$ ls build/compiler/tf2circle-value-pbtxt-remote-test/Result_*.csv
Result_20191119_212521.csv
...
```

- Detailed log file for each test cases is also created.

```
$ cat build/compiler/tf2circle-value-pbtxt-remote-test/*.log
```

4.7.39 tf2tflite-dredd-pb-test

tf2tflite-dredd-pb-test validates non-functional aspects of `.tflite` files, which are compiled from `.pb` files.

For more information, please refer to `README.md` in *dredd-rule-lib*.

4.7.40 tf2tflite-dredd-pbtxt-test

4.7.41 tf2tflite-value-pb-test

4.7.42 tf2tflite-value-pbtxt-test

Run `tf2tflite` to `test.lst` and do random value test using `nnkit`. Write `test.local.lst` for local test list.

4.7.43 tf2tfliteV2-conversion-test

4.7.44 tflite2circle-conversion-test

Run `tflite2circle` to check whether *tflite* model is able to be converted into *circle* model.

CHAPTER 5

Common IR

5.1 Introduction to circle

5.2 What is Common IR

CHAPTER 6

Package

6.1 Introduction to Package

6.2 Design of Package

6.2.1 Manifest

6.2.2 Supported Models

6.2.3 User Defined Operation

CHAPTER 7

Platform

7.1 Ubuntu

7.2 Tizen

7.3 Android

CHAPTER 8

Devices

8.1 ODROID-XU3

8.2 ODROID-XU4

8.3 Raspberry Pi 3

CHAPTER 9

Test & Benchmarks

9.1 Scripts

9.2 Benchmarks

CHAPTER 10

Release

10.1 1.0

10.1.1 Release Note 1.0.0

10.2 1.1

10.2.1 Release Note 1.1.0

10.3 1.2

10.3.1 Release Note 1.2.0

10.4 1.3

10.4.1 Release Note 1.3.0

10.5 1.4

10.5.1 Release Note 1.4.0

CHAPTER 11

Indices and tables

- genindex
- modindex
- search