Calibration Tool Guide

1 Introduction

This calibration tool would convert the FP32 caffe model to INT8 and generate the calibration table. These files can be later transformed to bmodel via bm_builer.bin.

2 Preparation

2.1 Environment

docker - 18.09.3
python - 2.7

2.2 Build Docker

The Calibration tool should run on our modified caffe framework and depends on several third party packages. So we **strongly recommend** you to use the docker as provided in the 'docker' directory.

Here is an example of how to build with cpu docker file:

docker build -t bmcalibration:2.0 -f docker/Dockerfile_CPU .

Then you can run the docker image as

docker run -v /workspace:/workspace -it bmcalibration:2.0

After running docker, you can try to import caffe in the python console to see it's working or not.

Both calirbation tool and tuning tool shoud use this docker envirement.

If you want to use your local machine(docker environment is not installed), please set the PYTHONPATH and LD_LIBRARY_PATH to the caffe folder in case the shared library files can not be found.

Here is the example:

```
export LD_LIBRARY_PATH=home/test/bm1880-calibration/caffe:$LD_LIBRARY_PATH
export PYTHONPATH=/home/test/compiler/bm1880-calibration/:$PYTHONPATH
```

2.3 GPU Support

For GPU users, we also provide GPU compatible caffe libraries to run calibration faster. Currently we only provide pre-built libraries on CUDA **9.0** environment. **First, copy the prebuilt shared library files in the caffe_gpu folder to caffe folder.**

Then run docker build and run:

nvidia-*docker build -t bmcalibration:2.0_gpu -f docker/Dockerfile_GPU .*
nvidia-docker run -v /workspace:/workspace -it bmcalibration:2.0_gpu

Now you can try nvidia-smi in the docker environment. If success, the GPU infos will print on the console.

if you want to use your local machine, set the PYTHONPATH and LD_LIBRARY_PATH after copy the shared libraries in the caffe_gpu folder to caffe folder.

For running caffe in GPU mode, please use the caffe apis below to set GPU related environment:

caffe.set_mode_gpu()
caffe.set_device(device_id)

device_id means the GPU id on your machine.

Currently we only provide pre-built caffe library based on cuda 9.0. Please make sure your host PC gpu driver is compatible with it .

2.4 Calibration Data

The calibration data is used to calculate the data histogram for KL divergence to find a suitable threshold. An ideal calibration data should have a very similar distribution with real-world data streaming from deep learning based services and cover as much scenario as possible.

We support general caffe data layer like lmdb and customized python data layer. Both kind of data layer should be named by *data* for compiler purpose.

For lmdb, first you need to convert your data to lmdb format and link it to the data layer. Here's an example of how to write the prototxt data layer:

```
layer {
        name: "data"
        type: "Data"
        top: "data"
        include {
                phase: TEST
        }
```

```
    # The preprocess of your data
    transform_param {
        #mirror: false
        #crop_size: 224
        #mean_file: "./imagenet_mean.binaryproto"
    }
    data_param {
        # The lmdb file path
        source: "./calibration/ilsvrc12_val_lmdb"
        batch_size: 1
        backend: LMDB
    }
}
```

Another data layer method is using python layer. Here is an example *“samples/detection/yolov3/yolo_data_layer.py”*. **Please note that the calibration data must have the same preprocess as when testing.**

3 Supported layer

BatchNorm
Concat
Convolution
Crop
Deconvolution
DetectionOutput
Eltwise (sum, max, prod)
InnerProduct
Pooling (max, avg)
PRelu
PriorBox
Relu
Reshape
Scale

4 How to Run calibration tool

4.1 Calibration tool detail explanation

The calibration step is packaged in the *Calibration* class. You can pass the prototxt and caffemodel to it and export outputs. Below is the details in calibration.py:

```python
def run_calibration(args):
    calibration_info = {
        "model_name": args.model_name,
        "in_prototxt": os.path.join(arg.model_path,'deploy.prototxt'),
        "in_caffemodel": os.path.join(arg.model_path,'custom.caffemodel'),
        "iteration": 10,
        "enable_memory_opt": args.memory_opt,
        "enable_calibration_opt": 1,
        "histogram_bin_num": 2048,
        "math_lib_path": './lib/calibration_math.so'
    }
    print(calibration_info)

    calib = Calibration(calibration_info)
    calib.calc_tables()
    calib.export_model('{}/bmnet_{}_int8.caffemodel'.format(args.model_path, args.model_name))
    calib.export_calirabtion_pb2('{}/bmnet_{}_calibration_table'.format(args.model_path, args.model_name))
```

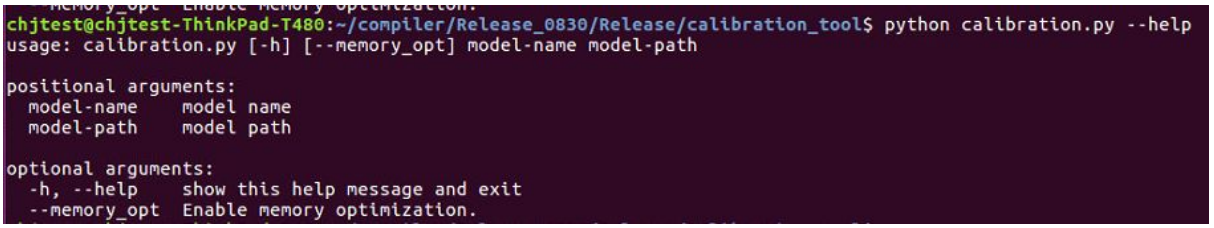
The parameters of *calibration_info* are explained as below. You can choose proper parameters for your model.

Parameter	Description
model_name	The name of your model
in_prototxt	The input caffe prototxt
in_caffemodel	The input caffemodel
iteration	How many images you want to use for calibration.
enable_calibration_opt	Optimize calibration procedure. Some specific models(ssd is one of them) may encounter errors when use this flag .
enable_memory_opt	To open the memory optimization. It reduces memory usage a lot but takes a little more time. Open this option when you are facing the out of memory problem. If there is ‘LRN’ or ‘Normalize’ layer in your model, you should not use this option.
histogram_bin_num	The histogram bin amount of KLD algorithm. Bigger may be more accurate but takes more times. Note that bigger bin number may need more calibration data. Need to be a multiple of 128. If not pass, the default is 2048.
enable_concat_quantize	Enable Concat layer quantization. It may improve accuracy.

The *calc_tables()* function will generate the thresholds and quantize the weight. The *export _model()/export_calibration_pb2()* function exports the quantized model/calibration table to the path you give it.

4.2 Run calibration tool

You can refer to calibration.py help to use it as shown below.

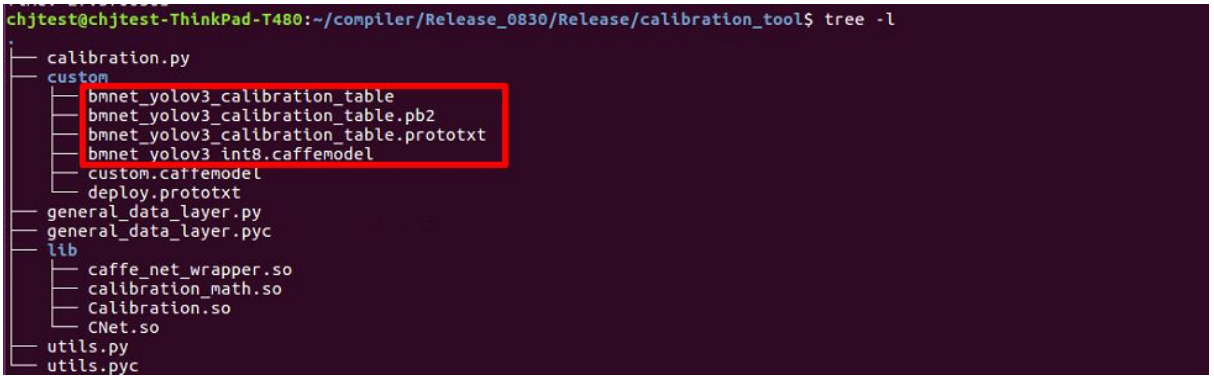


Parameter	Description
model-name	The model type name of your model.
model-path	model path in which you put your model files (deploy.prototxt and custom.caffemodel) and also generated output files will be in it.
--memory_opt	[optional] To open the memory optimization. It reduces memory usage a lot but takes a little more time. Open this option when you are facing the out of memory problem. If there is 'LRN' or 'Normalize' layer in your model, you should not use this option.

Please follow these steps for your caffe model calibration case.

- Modify model prototxt data layer as described in 2.4 (Calibration Data) part. and then rename your prototxt as 'deploy.prototxt'
- Rename weight model as 'custom.caffemodel'
- Put deploy.prototxt and custom.caffemodel in your model folder(e.g. custom)
- Run calibration.py as below (take yolov3 as example).
  python calirbation.py yolov3 custom --memory

Below code tree is just for your reference.



Files in red rectangle are generated.

5 INT8 Inference

The modified caffe provide users to inference in INT8 mode. We can easily check the amount of the accuracy drop via the modified caffe framework.

We will first introduce the APIs we provide and give an example of how to use it.

5.1 int8_init

This api provide an interface to initialize an int8 caffe net. Here is an example of how to use it.

self.net.int8_init(proto_path, cali_model, cali_proto, int8_layer)

self.net is the caffe net object which is initialized by caffe.Net. The parameters of int8_init are as below:

Parameter	Description
proto_path	The caffe prototxt file path
cali_model	The path of int8 caffemodel wich is generated by calibration tool
cali_proto	The path of calibration pb2 file which is generated by calibration tool
int8_layer	The layer name string that should run in int8 mode. You can use ‘,’ to concatenate more than one layer. Note that BatchNorm and the Scale layer after BatchNorm can not use different mode separately.

5.2 yolo

This api only for inference yolo. If you are testing any version of yolo, please remember to call this api ones.

caffe.yolo()

There is no parameter of this function.

5.3 Inference

The input data has to be quantized before pass to the model. We provide the quantization process in the *Input* data layer when *enable_quantize* is opened. So you do not need to do anything. If you are using your own input layer, remember to add the quantization process (x * 128 / threshold). The data layer in prototxt file should be like:

```
layer {
        name: "data"
        type: "Input"
        top: "data"
        input_param {
                shape: { dim: 1 dim: 3 dim: 608 dim: 608 }
                enable_quantize: 1
        }
}
```

All other input format should quantize by yourself.

The flexibility of *int8_layer* parameter makes users to check which layer drop accuracy most. You can add layer by layer from the first layer to the end. Here is an example:

```
param = caffe_pb2.NetParameter()
text_format.Merge(open(proto_path).read(), param)

layer_str = ''
for layer in param.layer:
    layer_str += str(layer.name)

    # BatchNorm and Scale layer must open int8 together.
    if layer.type == 'BatchNorm':
        layer_str += ','
        continue

    net_8 = caffe.Net(proto_path, model_path, caffe.TEST)
    net_8.int8_init(proto_path,
                    calibration_model,
                    calibration_proto,
                    layer_str)

    net_8.blobs['data'].data[...] = image
    out8 = net_8.forward()

    layer_str += ','
```

First, parse all layer name from caffe prototxt. Here we use the *caffe_pb2.NetParameter* object to parse the layer information. Then, loop all layers and append each layer name to *layer_str* string one by one (skip BatchNorm layer because it should use int8 mode with Scale together). The *out8* variable represents the output of int8 mode inference. You can record them and see which layer causes the accuracy drop most.

In our experience, the accuracy drop rate will be higher and higher when the int8 mode layer become more.

6. Tips

Here are some tips to enhance the accuracy when using this tool.

1. **Increase the calibration data number**. By increasing the *iteration* parameter, this tool will use more data to calculate the quantization threshold. In our experiments, more data often gets more accurate. Note that the maximum iteration should be smaller than the size of LMDB dataset or the size of input text file when you use python data layer.
2. **Increase the *histogram_num***. This means that the tool will search the threshold finer and may find a better one.
3. **Using the auto tuning tool to get a better quantization threshold**. Please see the auto tuning tool document for more information.