DRP-AI Quantizer (INT8 Quantization Tool) Version 1.0.1

User's Manual

Contents

1. Overview

This section describes the operating environment and functions of the DRP-AI Quantizer.

DRP-AI Quantizer is included in the installer of DRP-AI Translator i8.

After DRP-AI Translator i8 is installed, the product will be deployed to the following path:

```
DRP-AI_Translator_i8/drpAI_Quantizer
```

1.1 Product Configuration

Table 1.1 lists the components of this product.

Table 1.1 Product Configuration

Item	Description
r20ut5184ej0102-drp-ai.pdf	This manual
drpAI_Quantizer	DRP-AI Quantizer (product covered by this manual)

1.2 Configuration of Files

Table 1.2 lists the files and modules required for running this tool.

Table 1.2 Configuration of Files

Root Folder	Folder or File Name	Description
drpAI_Quantizer	drpai_quantizer/	Quantization module folder
	onnx_runtime/	ONNX model inference module folder
	inference_resnet.py	Inference script for testing accuracy
	nchw_datareader.py	Sample of channel first calibration data reader(For PyTorch-trained model)
	nhwc_datareader.py	Sample of channel last calibration data reader(For Keras-trained model)
	modA_resnet18.onnx	FLOAT format ONNX file sample 1 (a PyTorch-trained ResNet18 model)
	modB_resnet18.onnx	FLOAT format ONNX file sample 2 (a Keras-trained ResNet18 model)
	licenses-abstract.txt	License information for the modules used in this tool

1.3 Target Device

The target devices of the DRP-AI Quantizer are those of the following series.

- RZ/V2x (next-generation products)

1.4 Operating Environment

Table 1.3 describes the operating environment and software to be installed for the DRP-AI Quantizer.

Table 1.3 Operating Environment

Item	Software Name	Version Number
Operating environment	Ubuntu	20.04 LTS, 64-bit version
Software to be Installed	Python	3.8.10
	ONNX Runtime	1.14.1
	numpy	1.24.3
	pillow	9.5.0
	scipy	1.10.1
	protobuf	4.23.0
	sympy	1.12
	packaging	23.1
	onnxoptimizer	0.3.8
	matplotlib	3.7.1

1.5 Notes on the DRP-AI Quantizer

Development of the DRP-AI Quantizer was based on the quantization module implemented in ONNX Runtime v1.14.1, with some specifications changed and some unique features added. Accordingly, also refer to the related documents for ONNX Runtime.

ONNX Runtime < https://github.com/microsoft/onnxruntime/tree/v1.14.1>

Quantization in ONNX Runtime < https://onnxruntime.ai/docs/performance/quantization.html >

Outline of added and changed specifications relative to ONNX Runtime v1.14.1:

- QuantType.QInt8 is supported as a data type for activation.
- An algorithm for zero-point calculation in INT8 calibration was implemented.
- Debugging of entropy calibration was implemented.
- The setting of the BiasAdd operation for fully connected layers was changed.
- The input scaling for the Add operation was changed.
- Input scale and zero-point for Concat operation was changed.
- Quantization target exclusion function for each activation function was implemented.

1.6 Functional Overview

The DRP-AI Quantizer provides quantization optimized for the DRP-AI for ONNX-format AI models. Quantizing an AI model reduces the size of the model itself, achieving faster inference times. The DRP-AI Quantizer also handles accuracy evaluation for quantized models.

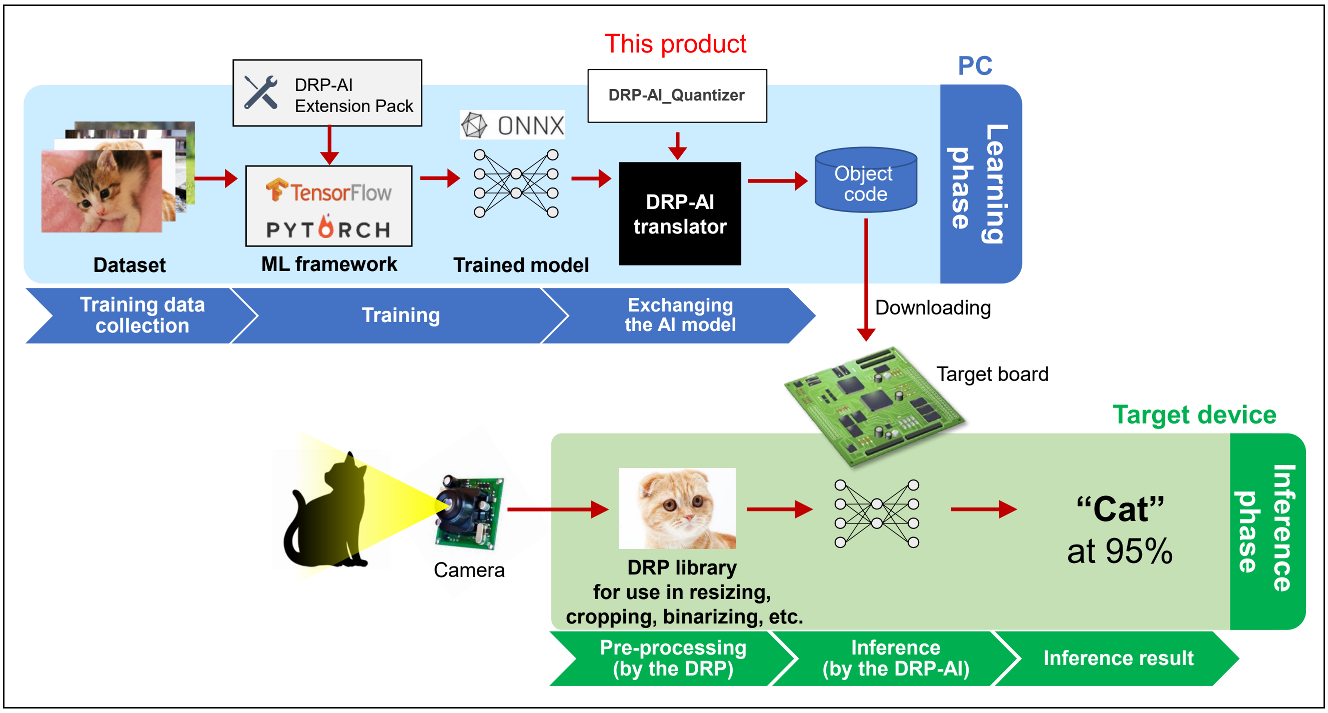


Figure 1.1 This Product's Role in the AI Design Process

Note that AI models to run on the next-generation products of the RZ/V2x series always require quantization.

1.7 Quantization

Quantization is the process of reducing the sizes of models by representing parameters of networks such as weights with a lower bit width.

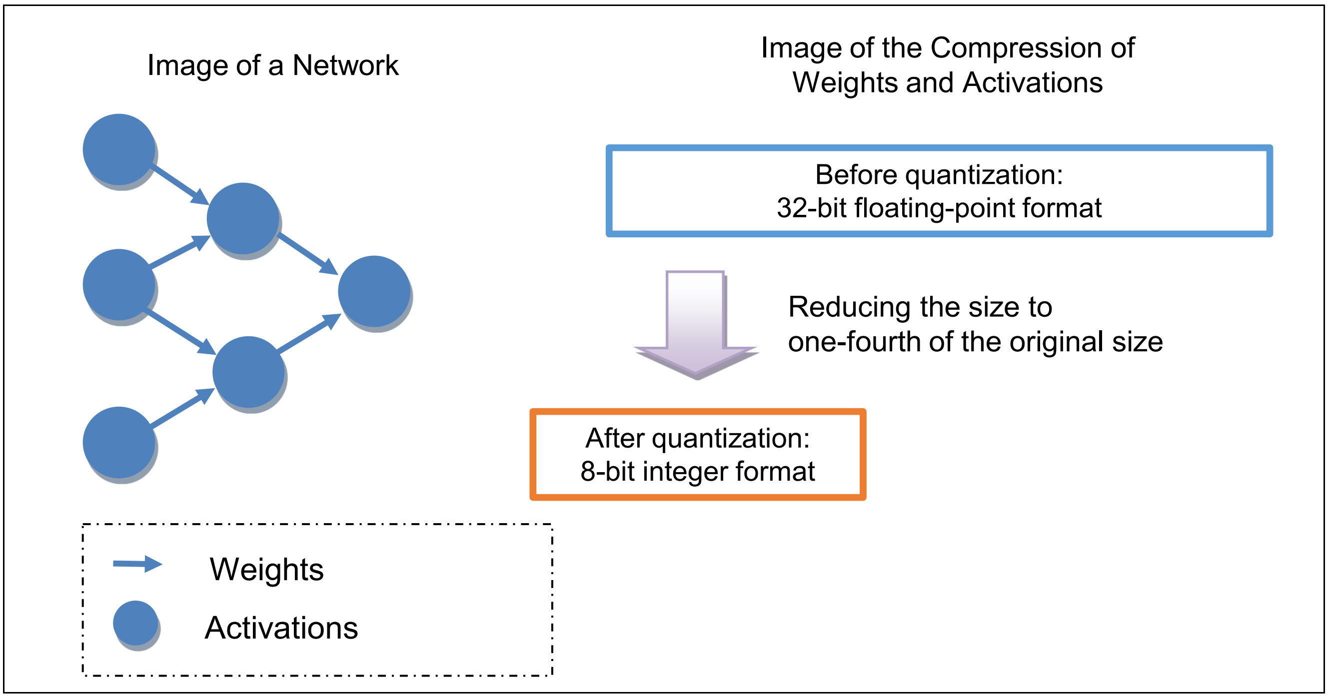


Figure 1.2 Schematic View of the Effect of Quantization with This Tool

This tool converts the weight parameters and the activation values of an AI model from 32-bit single-precision floating-point values into 8-bit integer values, reducing the size of the model to approximately one-fourth of its original size. Note that this tool quantizes activation values as well as weight parameters, because the tool performs static quantization conversion. This requires a dedicated dataset for use in calibration[Note]. See section 3.4.1.1. Furthermore, note that the preprocessing of input data for the target AI model must be reflected in the Calibration data reader before quantization. Again, see section3.4.1.1. Optional settings for quantization can be specified by adding command line options as described in section 3.5.

Note: Calibration is the process of minimizing the loss of accuracy due to quantization through the input of multiple data from which the neural network is to actually draw inferences.

1.8 Change of Recognition Accuracy Due to Quantization

The table below lists changes in the accuracy of recognition due to INT8 quantization. The changes are negligible.

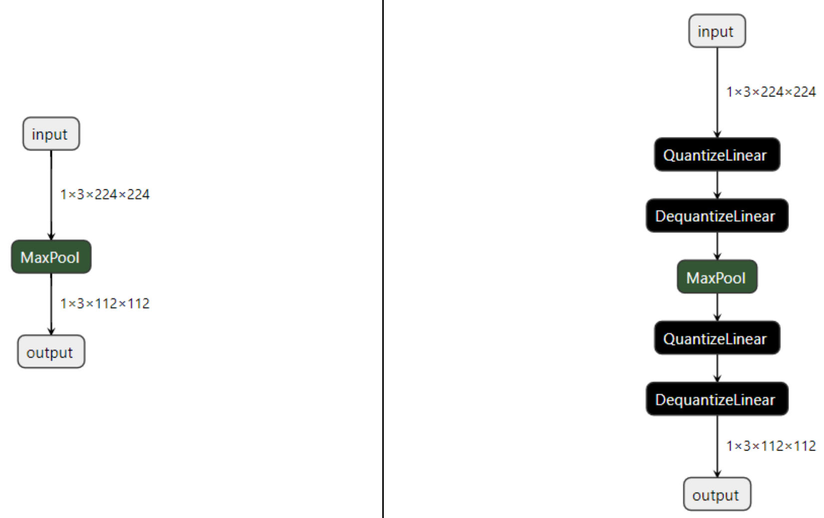
Table 1.4 Accuracy changes before and after INT8 quantization

AI Facility	**Classification**	**Object Recognition**		**Segmentation**	**Pose Estimation**
Model Name	ResNet18	TinyYOLOv2	YOLOv2	DeepLabV3	HRNET
Dataset	ImageNet	VOC	VOC	CityScapes	MMPose
Accuracy before INT8 Quantization	67.40 %	58.20 %	74.85 %	77.14 %	74.60 %
Accuracy after INT8 Quantization	67.00 % (−0.40 %)	57.90 % (−0.30 %)	74.93 % (+0.08 %)	77.01 % (−0.13 %)	74.50 % (−0.10 %)

Note: Quantization is not guaranteed for all models. Also, if the accuracy after INT8 quantization is degraded, please refer to Chapter 5 Suppressing Post-quantization Accuracy Degradation .

1.9 Updates in Version 1.01

In the latest update to version 1.01 of the DRP-AI Quantizer, We have solved the problem of some operators such as `maxpool` or `transpose` nodes can not be quantized when they are initial or final layers.



V1.0.0 quantization result

V1.0.1 quantization result

Figure 1.3 In Version v1.01, When the `maxpool` node is the initial or final layer, It will also be quantized

2. Setting Up the DRP-AI Quantizer

For instructions on setting up the DRP-AI Quantizer, please refer to Chapter 3, ‘Installation’ in the DRP-AI Translator i8 user manual (Document ID: r20ut5336).

3. Using the DRP-AI Quantizer

This chapter describes how to use the DRP-AI Quantizer to perform a post-training quantization(PTQ).

3.1 Overview

The DRP-AI Quantizer performs static quantization conversion for ONNX-format AI models, thus converts the activations and the weight and bias parameters of an AI model from 32-bit single-precision floating-point values into 8-bit integer values. The quantization requires a dataset for use in calibration as well as a trained AI model (an ONNX file). The accuracy of a quantized INT8 ONNX file can be tested by using an inference script.

Chapter 3 is dedicated to the use of the DRP-AI Quantizer, detailing the quantized command line interface and Python API interface, as well as describing how to customize the calibration data reader for different ONNX format models. This chapter also covers advanced options and available parameters.

When using DRP-AI Quantizer, you can use either way to implement static quantization of onnx format models. If you want to use the Command line interface(CLI) for onnx format model quantization, please refer to section 3.2. If you want to use the Python API for onnx format model quantization, please refer to Section 3.3.

After understanding how to quantize a model using DRP-AI Quantizer's CLI or PythonAPI, you can move on to section 3.3 Quantize a model using your own calibration dataset and section 3.5 DRP-AI Quantizer's advanced options.

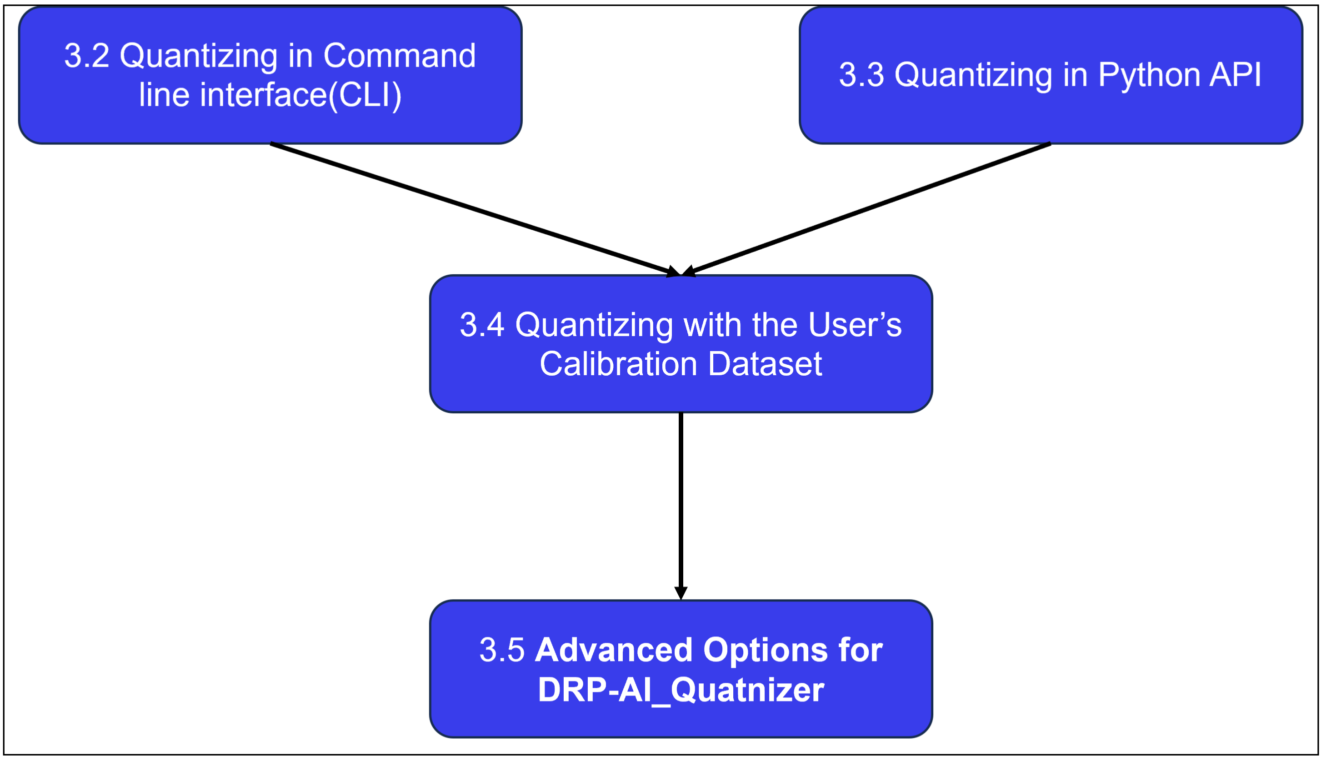


Figure 3.1 Reading index for chapter 3

3.2 Quantizing in Command line interface(CLI)

3.2.1 Quantizing with the Use of Sample Data in CLI

This section describes the procedure for quantization and the testing of accuracy with the included samples. The sample data include the two following FLOAT format ONNX files.

- modA_resnet18.onnx: A PyTorch-trained ONNX file in NCHW format (channels first)
- modB_resnet18.onnx: A Keras-trained ONNX file in NHWC format (channels last)

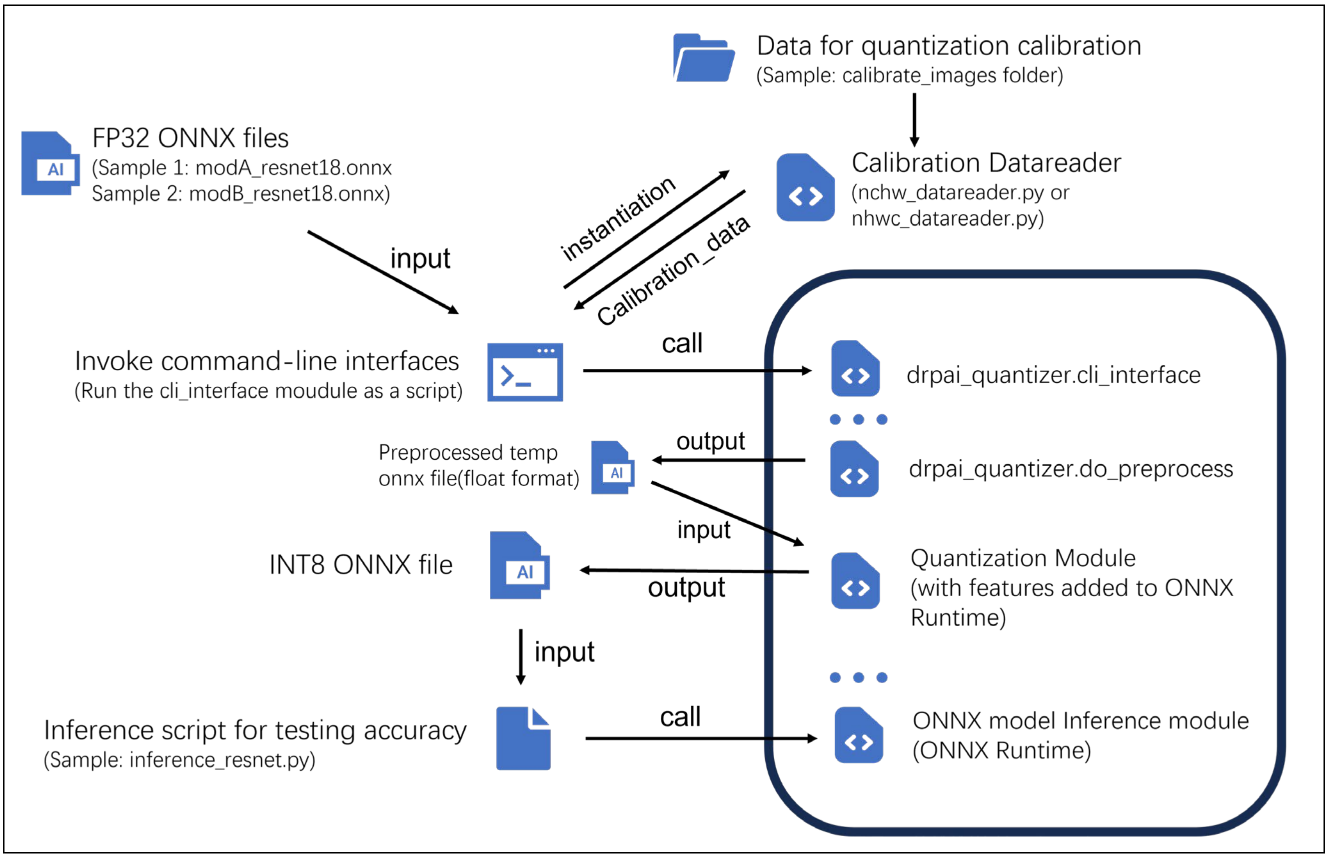


Figure 3.2 Outline of the Procedure for Using CLI

3.2.1.1 Quantizing with the PyTorch-Trained ONNX File

Quantizing with the use of the PyTorch-trained ONNX file in NCHW format (channels first) as the input file is enabled by setting input and output files when run the cli_interface module as a script. In addition, the NCHW format (channels first) calibration data reader can be dynamically importeded as a command line option. In the following sample commandline, There is a class 'NCHWDataReader' class in the file nchw_datareader.py, Refer to Table 3.3.2.3 in Section 3.2.2 for more information of the Class 'NCHWDataReader'. Also, the mean value and standard deviation can be set as command line options.

The sample command line is shown in the listing below.

```
python3 -m drpai_quantizer.cli_interface ¥                # Invoke the cli_interface module
 --input_model_path modA_resnet18.onnx¥                   # Input: FLOAT format ONNX file
 --output_model_path modA_resnet18_q.onnx ¥               # Output: INT8 ONNX file
 --calibrate_dataset ./calibrate_images ¥                 # Dataset for quantization calibration
 --datareader_path ./nchw_datareader.py ¥                 # Dynamic import the calibration datareader
 --norm_mean [0.4914, 0.4822, 0.4465] ¥                   # The normalize mean value
 --norm_std [0.2023, 0.1994, 0.2010]                      # The normalize standard deviation
```

3.2.1.2 Quantizing with the Keras-Trained ONNX File

Quantizing with the use of the Keras-trained ONNX file in NHWC format (channels last) as the input file is enabled by setting input and output files when run the cli_interface module as a script. In addition, the NHWC format (channels last) calibration data reader can be dynamically importeded as a command line option. In the following sample commandline, There is a class 'NHWCDataReader' class in the file nhwc_datareader.py, Refer to Table 3.3.2.4 in Section 3.2.2 for more information of the Class 'NCHWDataReader'. Also, the mean value and standard deviation can be set as command line options.

The sample command line is shown in the listing below.

```
python3 -m drpai_quantizer.cli_interface ¥                # Invoke the cli_interface module
 --input_model_path modB_resnet18.onnx¥                   # Input: FLOAT format ONNX file
 --output_model_path modB_resnet18_q.onnx ¥               # Output: INT8 ONNX file
 --calibrate_dataset ./calibrate_images ¥                 # Dataset for quantization calibration
 --datareader_path ./nhwc_datareader.py ¥                 # Dynamic import the calibration datareader
 --norm_mean [0, 0, 0] ¥                                  # The normalize mean value
 --norm_std [1, 1, 1]                                     # The normalize standard deviation
```

3.2.2 Commandline interface available Options

Table 3.2.2.1 lists the command line options available in invoking the cli_interface module. The default setting is applied when an option is not specified.

Table 3.2.2.1 Command-Line Options for the cli_interface module

Option	Abbreviation	Default Setting	Outline
--input_model_path	—	Explicit setting required	Path to the input ONNX model
--output_model_path	—	Explicit setting required	Path to the quantized output ONNX model
--calibrate_dataset *<path>*	—	./calibrate_images	Dataset path for calibration
--calibrate_method *<value>*	-cm	MinMax	Calibration method: MinMax or Entropy
--datareader_path	—	Explicit setting required	Setting the shape of the input layer as channels last format
--operate_to_exclude *<name,…>*	-ex	—	Non-quantized operation names (comma-separated) Example: --operate_to_exclude Softplus,Tanh
--node_to_exclude *<name,…>*	-exn	(Disabled)	Node names not subject to quantization (comma-separated) Example: --node_to_exclude Concat_264,Concat_285
--norm_mean *<mean value>*	—	Explicit setting required	Average for preprocessing of calibration data Specify the average value of the three input channels to the model in the form [val1, val2, val3] Example: --norm_mean [0.4914,0.4822,0.4465]
--norm_std *<standard deviation>*	—	Explicit setting required	Standard deviation for preprocessing of calibration data Specify the standard deviation value of the three input channels to the model in the form [val1, val2, val3] Example: --norm_std [0.2023,0.1994,0.2010]
--skip_preprocess	—	(Disabled)	Skip the quantization preprocess before quantize the model. If encounter the error message such like `` `Exception: Pre-processing before quantization was Failed.` `` when quantizing a model, try re-quantize the target model with this option. Example: --skip_prerprocess

--preprocess_mode	-ppm	default	Mode for quantization preprocessing. The preprocess_model must be set as default. Example: --preprocess_mode default
--preprocessed_model_output	-pmo	(Disabled)	Save the quantize preprocessed mode. Example: --preprocessed_model_output
--tvm	—	(Disabled)	Quantization for tvm. It will call the TVMDataReader automatically when using this option. So when using this option, do not set the --datareader_path option. Example: --tvm
Non-public advanced options	—	—	Omit "--" when setting option. Example: optimize_model True (It will be described in detail in section 3.5.)

3.3 Quantizing in Python API

3.3.1 Quantizing with the Use of Sample Data in Python API

This section describes the procedure for quantization and the testing of accuracy with the included samples. The sample data include the two following FLOAT format ONNX files.

- modA_resnet18.onnx: A PyTorch-trained ONNX file in NCHW format (channels first)
- modB_resnet18.onnx: A Keras-trained ONNX file in NHWC format (channels last)

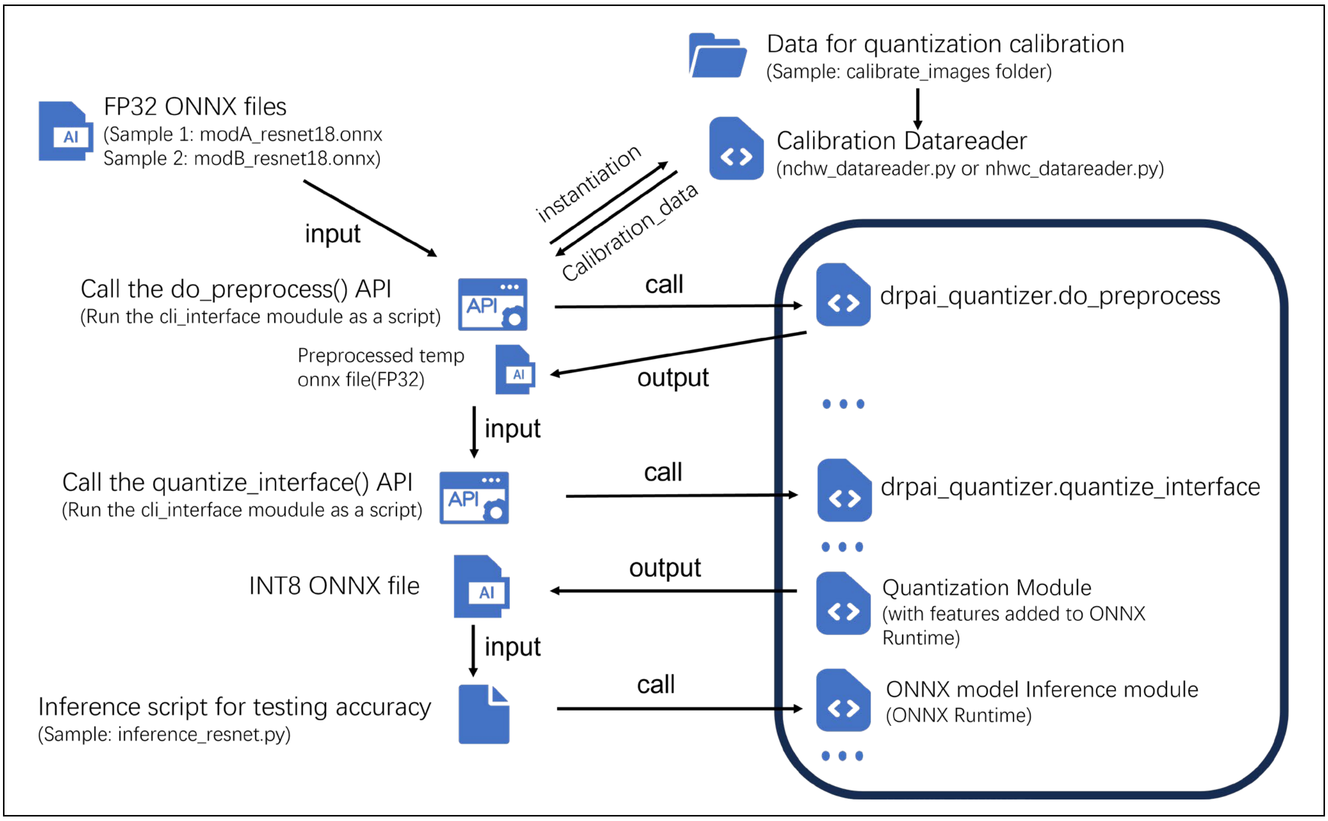


Figure 3.3 Outline of the Procedure for Using Python API

3.3.1.1 Quantizing with the PyTorch-Trained ONNX File

The sample usage code is shown as below:

- Step1. Instantiate the sample NCHW(channel first) calibration data reader.

The `NCHWDataReader` class is utilized to read the calibration data in the NCHW format. It requires the following parameters:
  - The path to the calibration images: `./calibration_data/calibrate_images/`
  - Mean values for normalization: `[0.4914, 0.4822, 0.4465]`
  - Standard deviation for normalization: `[0.2023, 0.1994, 0.2010]`
  - The path to the input ONNX model trained with PyTorch: `./modA_resnet18.onnx`

Refer to Table 3.3.2.3 in Section 3.2.2 for more information of the Class 'NCHWDataReader'.

```
nchw_datareader=NCHWDataReader('./calibration_data/calibrate_images', ¥
                               [0.4914, 0.4822, 0.4465], ¥
                               [0.2023, 0.1994, 0.2010], ¥
                               './modA_resnet18.onnx')
```

- Step2. Call of the do_prerpocess() API to do the quantization preprocess.

The `do_preprocess` function preprocesses the input model. It takes the following parameters:
  - The path to the input ONNX model: `./modA_resnet18.onnx`
  - The path for the preprocessed model's output: `preprocessed_modA_model.onnx`
  - The preprocessing mode, which is set to `default` in this instance.

```
do_preprocess(input_model_path='./modA_resnet18.onnx',¥
              preprocessed_model_output='preprocessed_modA_model.onnx',¥
              preprocess_mode='default')
```

- Step3. Call of the quantize_interface() API to quantize the preprocessed model.

The `quantize_interface` function is used to perform model quantization. The function is provided with:
  - The path to the preprocessed model: `preprocessed_modA_model.onnx`
  - The desired output path for the quantized model: `modA_resnet18_q.onnx`
  - The calibration method, in this case, `CalibrationMethod.MinMax` indicating the minmax-based calibration method.
  - The data reader (`nchw_datareader`), which has been set up earlier to read the calibration data in the NCHW format.

```
quantize_interface(input_model_path='preprocessed_modA_model.onnx', ¥
                   output_model_path='modA_resnet18_q.onnx', ¥
                   calibrate_method=CalibrationMethod.MinMax, ¥
                   datareader=nchw_datareader)
```

3.3.1.2 Quantizing with the Keras-Trained ONNX File

The sample usage code is shown as below:

- Step1. Instantiate the sample NHWC(channel last) calibration data reader.

The `NHWCDataReader` class is utilized to read the calibration data in the NHWC format. It requires the following parameters:
- The path to the calibration images: `./calibration_data/calibrate_images/`
- Mean values for normalization: `[0,0,0]`
- Standard deviation for normalization: `[1,1,1]`
- The path to the input ONNX model trained with Keras: `./modB_resnet18.onnx`

Refer to Table 3.3.2.4 in Section 3.2.2 for more information of the Class 'NHWCDataReader'.

```
nchw_datareader=NHWCDataReader('./calibration_data/calibrate_images', ¥
                               [0, 0, 0], ¥
                               [1, 1, 1], ¥
                               './modB_resnet18.onnx')
```

- Step2. Call of the do_prerpocess() API to do the quantization preprocess.

The `do_preprocess` function preprocesses the input model. It takes the following parameters:
- The path to the input ONNX model: `./modB_resnet18.onnx`
- The path for the preprocessed model's output: `preprocessed_modB_model.onnx`
- The preprocessing mode, which is set to `default` in this instance.

```
do_preprocess(input_model_path='./modB_resnet18.onnx',¥
              preprocessed_model_output='preprocessed_modB_model.onnx',¥
              preprocess_mode='default')
```

- Step3. Call of the quantize_interface() API to quantize the preprocessed model.

The `quantize_interface` function is used to perform model quantization. The function is provided with:
- The path to the preprocessed model: `preprocessed_modB_model.onnx`
- The desired output path for the quantized model: `modB_resnet18_q.onnx`
- The calibration method, in this case, `CalibrationMethod.MinMax` indicating the minmax-based calibration method.
- The data reader (`nhwc_datareader`), which has been set up earlier to read the calibration data in the NHWC format.

```
quantize_interface(input_model_path='preprocessed_modB_model.onnx', ¥
                   output_model_path='modB_resnet18_q.onnx', ¥
                   calibrate_method=CalibrationMethod.MinMax, ¥
                   datareader=nhwc_datareader)
```

3.3.2 Python API available functions table and API reference

■ **Table3.3.2.1 lists of the functions for the PythonAPI**

Class or Function Name	Description
'do_preprocess'	A function to preprocess an ONNX model given a set of arguments and conditions.
'quantize_interface'	A function that interfaces with the quantization process of an ONNX model.
'NCHWDataReader'	A class designed for calibrating ONNX models, specifically those expecting input in the NCHW (Channel-First) format.
'NHWCDataReader'	A class designed for calibrating ONNX models, specifically those expecting input in the NHWC (Channel-Last) format.

The following are the API References for the available functions.

■ **Table 3.3.2.1 Fucntion 'do_preprocess()' API reference**

[Overview]	A function to preprocess an ONNX model given a set of arguments and conditions.	
[Function/Class Name]	do_preprocess()	
[Calling format]	do_preprocess (input_model_path : str, preprocessed_model_output : str, do_symbolic_shape_inference : bool, do_onnx_shape_inference : bool, do_optimization : bool, skip_preprocess : bool, verbose : int, preprocess_mode : str)	
[Argument]	input_model_path : str,	Path to the input target ONNX model.
	preprocessed_model_output : str	Path where the preprocessed model will be saved.
	do_symbolic_shape_inference : bool	Flag to perform symbolic shape inference. Symbolic shape inference is most effective with transformer based models. Skip perform the symbolic shape inferences may reduce the effectiveness of quantization, as a tensor with unknown shape can not be quantized. Default is True.
	do_onnx_shape_inference : bool	Flag to perform ONNX shape inference. Skip perform the onnx shape inferences may reduce the effectiveness of quantization. Default is True.
	do_optimization : bool	Flag to perform optimization on the model. Skip perform this may result in ONNX shape inference failure for some models. Default is True.
	skip_preprocess : bool	Flag to skip the quantization preprocess step.

		Setting this parameter to True will skip all quantization preprocessing (equivalent to setting the 'do_symbolic_shape_inference', 'do_onnx_shape_inference', 'do_optimization' parameters to False at the same time) Default is False.
	verbose : int	Level of verbosity. Logs detailed info of inference, 0: turn off, 1: warnings, 3: detailed. Default is 1.
	preprocess_mode: str	Specifies the mode of preprocessing. The choice of mode must be set as “default” when executing Post training quantization. Default is “default”.
[Returns]	None	
[Remarks]	This function does not necessarily need to be called when performing quantization. Some models may get an error when performing symbolic shape inference. When it is not possible to perform any of the quantization preprocessing step due to model’s structure or choosing not to perform quantization preprocessing, you can choose not to execute this function and perform quantization directly.	

■ **Table 3.3.2.2 Fucntion ‘quantize_interface()’ API reference**

[Overview]	A function that interfaces with the quantization process of an ONNX model.	
[Function/Class Name]	quantize_interface	
[Calling format]	Quantize_interface (input_model_path : str, output_model_path : str, datareader : Instance of CalibrationReader, calibrate_method : str or Calibratemethod, operate_to_exclude : str, node_to_exclude : str)	
[Argument]	input_model_path : str,	Path to the input target ONNX model.
	Output_model_path : str	Path where the quantized model will be saved.
	Datareader : Instance of CalibrationReader	The data reader to use for calibration. An instance of a data reader used for model calibration. Default is None.
	Calibrate_method : str or Calibratemethod	The method of calibration to apply. Can be an instance of CalibrationMethod enum or a string. Default is CalibrationMethod.MinMax.

	operate_to_exclude : str	Operation types to exclude from quantization. Comma-separated string of operation types to exclude from quantization. Default is None.
	Node_to_exclude : str	Specific nodes to exclude from quantization. Comma-separated string of node names to exclude from quantization. Default is None.
[Returns]	None	
[Remarks]	The function allows for per-channel quantization and supports MinMax and Entropy calibration methods. The choice of calibration data and calibration method can significantly impact the model's accuracy and performance.	

■ **Table 3.3.2.3 Class 'NCHWDataReader()' API reference**

[Overview]	A class designed for calibrating ONNX models, specifically those expecting input in the NCHW (Channel-First) format. It extends the functionality of the CalibrationDataReader class. This class is tailored for preprocessing images for model calibration, which includes tasks like normalization, resizing, and converting data into the NCHW format.	
[Function/Class Name]	NCHWDataReader()	
[Calling format]	NCHWDataReader(calibration_image_folder, norm_mean, norm_std, augmented_model_path)	
[Argument]	calibration_image_folder (str):	The path to the directory containing the images used for calibration.
	Norm_mean (list of floats):	Normalization means for each color channel (R, G, B). It should be a list of three float values.
	Norm_std (list of floats):	Standard deviations for normalization of each color channel (R, G, B). This should also be a list of three float values.
	Augmented_model_path (str):	The file path to the augmented ONNX model that will be used for calibration.
[Returns]	None	
[Remarks]	Constraints in model input format: This class is specifically tailored for processing data in the NCHW format (Channels, Height, Width). It's crucial for models that require inputs in this channel-first format. Constraints in File Formats: The class can process images in common file formats like JPG, PNG, and BMP.	

■ **Table 3.3.2.4 Class 'NHWCDataReader()' API reference**

<table>
<tr><td>[Overview]</td><td colspan="2">A class designed for calibrating ONNX models, specifically those expecting input in the NHWC (Channel-Last) format. It extends the functionality of the CalibrationDataReader class. This class is tailored for preprocessing images for model calibration, which includes tasks like normalization, resizing, and converting data into the NHWC format.</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>[Function/Class Name]</td><td colspan="2">NHWCDataReader()</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>[Calling format]</td><td colspan="2">NHWCDataReader(calibration_image_folder, norm_mean, norm_std, augmented_model_path)</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td rowspan="4">[Argument]</td><td>calibration_image_folder (str):</td><td>The path to the directory containing the images used for calibration.</td></tr>
<tr><td>Norm_mean (list of floats):</td><td>Normalization means for each color channel (R, G, B). It should be a list of three float values.</td></tr>
<tr><td>Norm_std (list of floats):</td><td>Standard deviations for normalization of each color channel (R, G, B). This should also be a list of three float values.</td></tr>
<tr><td>Augmented_model_path (str):</td><td>The file path to the augmented ONNX model that will be used for calibration.</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>[Returns]</td><td colspan="2">None</td></tr>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>[Remarks]</td><td colspan="2">Constraints in model input format: This class is specifically tailored for processing data in the NHWC format (Height, Width, Channels). It's crucial for models that require inputs in this channel-last format.

Constraints in File Formats: The class can process images in common file formats like JPG, PNG, and BMP.</td></tr>
</table>

3.3.3 Python API interface available parameters

Table3.3.3.1 ~ Table3.3.3.3 lists the Python API interface available parameter when calling corresponding DRP-AI Quantizer module. The default setting is applied when an parameter is not specified.

■ **Table 3.3.3.1 PythonAPI parameters for the DataReader instantiation**

Parameter	Data_type	Default value	mandatory parameter	Outline
calibration_image_folder	str	None	True	Path to the calibration image folder
norm_mean	list	None	True	List of mean values for normalization, must be in the format [R, G, B].
norm_std	list	None	True	List of standard deviations for normalization, must be in the format [R, G, B].
augmented_model_path	str	None	True	Path to the ONNX model file which needs to be quantized

■ **Table 3.3.3.2 PythonAPI parameters for the call function of 'do_preprocess()' module**

Parameter	Data_type	Default value	mandatory parameter	Outline
input_model_path	str	None	True	Path to the input ONNX model
preprocessed_model_output	str	None	True	Path to the preprocessed output ONNX model
do_symbolic_shape_inference	bool	True	False	Whether do symbolic shape inference during quantization preprocessing
do_onnx_shape_inference	bool	True	False	Whether do onnx shape inference during quantization preprocessing
do_optimization	bool	True	False	Whether do graph optimization during quantization preprocessing
skip_preprocess	bool	False	False	Skip the above three quantization preprocessing
verbose	int	0	False	Level of verbosity. Higher values indicate more detailed logging.
Preprocess_mode	str	None	True	Specifies the type of model (Choice: Set as 'default' for post-training quantization(PTQ) models.
kwargs	—	—	—	Additional keyword arguments for specifying quantization properties.

#particular note: The function of the skip_preprocess parameter in do_preprocess() is not the same as the function of the skip_preprocess option in CLI. CLI's skip_prerpocess option represent skip all the process in do_preprocess(), which means that it will not call the do_preprocess() function.

■ **Table 3.3.3.3 PythonAPI parameters for the call function of 'quantize_interface()' module**

Parameter	**Data_type**	**Default value**	**mandatory parameter**	**Outline**
input_model_path	str	None	True	Path to the ONNX model file which needs to be quantized or the ONNX model file preprocessed by the do_preprocess() module.
output_model_path	str	None	True	Path where the quantized model will be saved.
datareader	A instance of CalibrationDataReader	None	True	A instance of a nchw or nhwc datareader
calibrate_method	CalibrationMethod	CalibrationMethod.MinMax	False	The calibration method used for quantization. Choice can be set as CalibrationMethod.MinMax or CalibrationMethod.Entropy
operate_to_exclude	str	None	False	Operations to exclude during quantization.
node_to_exclude	str	None	False	Nodes to exclude during quantization.
kwargs	—	—	—	Additional keyword arguments for specifying quantization properties.

3.4 Quantizing with the User's Calibration Dataset

Quantizing a model from FP32 to INT8 requires calibration with representative dataset samples to maintain the model's accuracy. To perform this calibration, you need to implement a data reader that feeds data to the model in a format it expects. This part will walk you through the process of creating a custom CalibrationDataReader for ONNX model quantization. We have already produced two files nchw_datareader.py and nhwc_datareader.py which contains the CalibrationDataReader classes corresponds respectively to PyTorch-Trained ONNX files and Keras-Trained ONNX files.

Since the model has a wide variety of data pre-processing processes, the appropriate data pre-processing should also be covered in the calibration data reader to ensure the appropriateness of the calibration data. The following sections will guide you how to create a calibration data reader which can be applied to DRP-AI Quantizer.

3.4.1 Preparation

Quantizing with a user's dataset requires the following four steps to input data to user's AI model and perform calibration during the conversion. The items framed by red round-cornered rectangles in Figure3.3 must be prepared or modified.

1. Prepare user's ONNX model files.
2. Prepare user's data for calibration.
3. Implement the preprocessing of input data in the sample nchw_datareader.py / nchw_datareader.py or user's customized calibration datareader.py file. Specification restrictions on customizing the calibration data reader will be introduces in this section.
4. Modify the inference script for testing accuracy. See section 0.

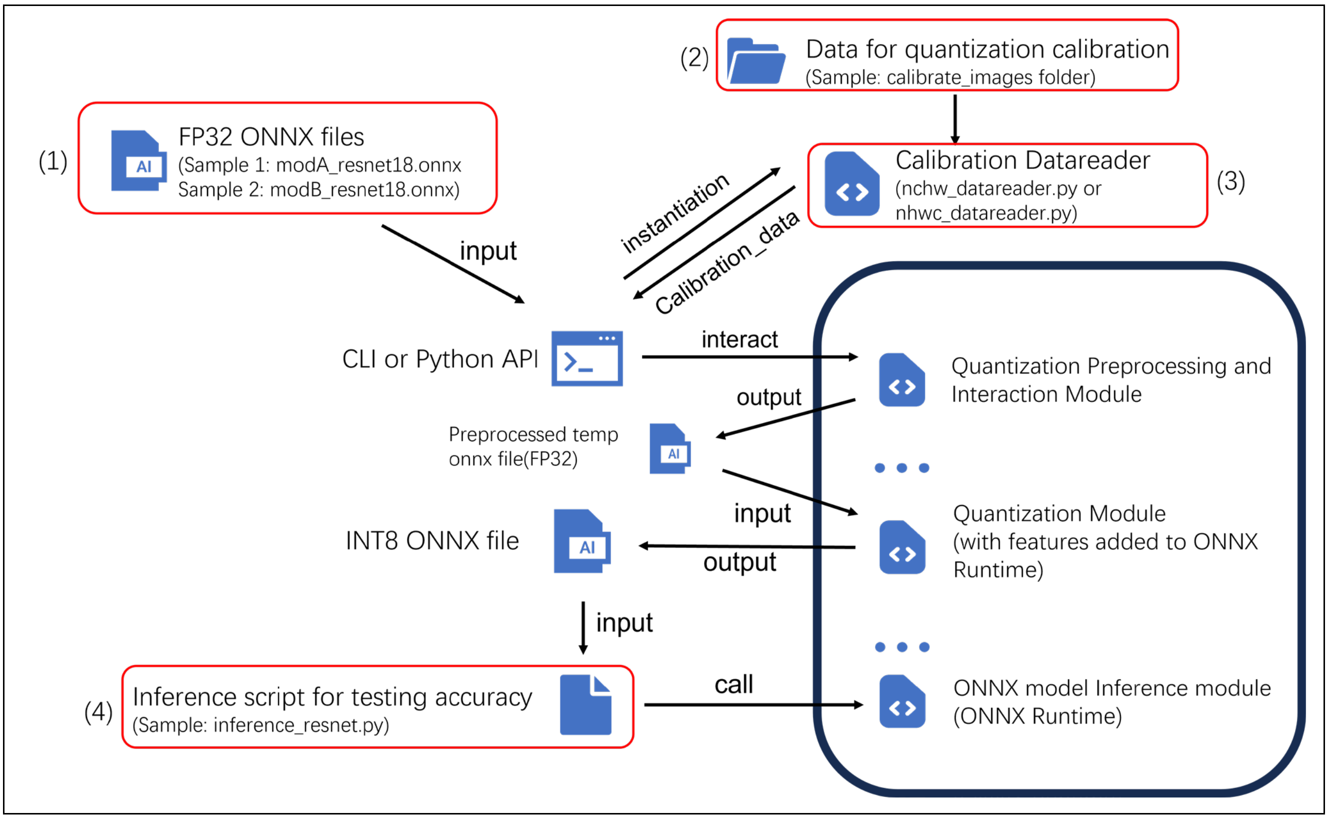


Figure 3.4 Contents in the Sample Data to be Modified with User's Data

3.4.1.1 Detailed Process of Preparing the User's Data for Calibration (2 of Figure3.3)

Calibration requires fewer data than training an AI model because the data are only used to calculate the range of the output values from each layer in the network. The first thing we need to be aware of is Calibration data should come from the same source or have similar characteristics as the training data. For instance, if your model is trained on the COCO dataset, use a subset of COCO for calibration.

The preparation of a calibration dataset for quantization is a balance between the amount of data and the distribution of data.

It is commonly recommended to use between 100 and 500 images for calibration data. However, the actual number should be determined based on the number of classes in your dataset. Please follow the examples below for reference when preparing your calibration data:

- If dataset has class number, pick up each class's 20-50 representative data. For the dataset has over 100 classes, pick up fewer images as following examples.
  - For datasets with 1000 classes, prepare 1 image for each class, resulting in a total of 1000 images.
  - For datasets with 100 classes, prepare 2 to 5 images for each class, resulting in a total of 200 to 500 images.
  - For datasets with 10 classes, prepare 20 to 50 images for each class, resulting in a total of 200 to 500 images.
  - For datasets with a single class, prepare 20 to 50 images for the class, resulting in a total of 20 to 50 images.

Figure3.5 calibration data preparation when dataset has class concept

- If dataset does not have class number such as those are used for pose estimation, prepare a random selection of 100 to 500 images.

Figure3.6 calibration data preparation when dataset does not have class concept

Use the prepared data for calibration according to either of the following procedures:

1. Store the data in the calibrate_images folder.
2. For CLI, Specify the folder including the data by using the "--calibrate_dataset" option described in section 3.2.1.1 and section 3.2.1.2.

For Python API, Specify the folder including the data by setting the “calibrateion_image_folder” parameters when instantiate the corresponding calibration data reader option described in the section 3.3.1 and section 3.3.2’s step1.

3.4.1.2 Detailed Process of Implementing the CalibrationDataReader Script (3 of Figure3.3))

The CalibrationDataReader scripts(nchw_datareader.py and nhwc_datareader.py) in the DRP-AI Quantizer can be used directly. If the input-data pre-processing of the target model is different from the pre-processing in the script, you need to edit the pre-processing part of the script or make a model-specific CalibrationDataReader.

In the produced nchw_datareader.py and nhwc-datareader.py, there is already defined a preprocess_func() function which can cope with most of the input data preprocessing, but when there are some special preprocessing needs, It can be realized by modifying this function or defining new functions to implement the appropriate preprocessing.

Implement Data Preprocessing

The Quantization and testing of accuracy in the produced CalibrationDataReader scripts(The nchw_datareader.py and nhwc_datareader.py) are performed with a CIFAR-10 dataset. Using a different dataset requires editing of the CalibrationDataReader script. Note that using a dataset preprocessed in training particularly requires preprocessing of the quantization and inference scripts. In the included CalibrationDataReader script, normalization of input data levels is performed using the values specified in the norm_mean and norm_std options as preprocessing for the data set.

The code snippet within the red frame in Figure 3.5 is the input data pre-processing. If padding or other processing other than normalization is required, please add additional processing in the code snippet within the red frame or customize your own CalibrationDataReader script.

```python
    def preprocess_func(self, height, width, size_limit=0):
        if size_limit > 0 and len(self.image_names) >= size_limit:
            batch_filenames = [self.image_names[i] for i in range(size_limit)]
        else:
            batch_filenames = self.image_names
        unconcatenated_batch_data = []

        for image_filepath in batch_filenames:
            pillow_img = Image.new("RGB", (width, height))
            pillow_img.paste(Image.open(image_filepath).resize((width, height)))
            input_data = np.float32(pillow_img) / 255
            input_data = np.float32(input_data)
            if self.norm_mean is not None:
                input_data = input_data - self.norm_mean
            if self.norm_std is not None:
                input_data = input_data / self.norm_std
            input_data = input_data.astype(np.float32)
            nhwc_data = np.expand_dims(input_data, axis=0)
            nchw_data = nhwc_data.transpose(0, 3, 1, 2)
            unconcatenated_batch_data.append(nchw_data)
        batch_data = np.concatenate(np.expand_dims(unconcatenated_batch_data, axis=0), axis=0)
        return batch_data
```

Figure 3.5 Input data pre-processing code snippet in nchw_datareader.py

After completing the customization of CalibrationDataReader, Continue using DRP-AI Quantizer for model quantization, please refer to section3.2 or section3.3 for quantization using CLI or Python API.

3.5 Advanced Options for DRP-AI Quatnizer

This section of the manual is dedicated to the advanced, non-public options available in the DRP-AI Quantizer command-line interface and PythonAPI. These options provide more granular control and customization for users who require specific functionalities beyond the standard toolset. It's important to note that these options are intended for advanced users who have a thorough understanding of the tool and its workings.

Non-public options are additional arguments that can be passed to the DRP-AI Quantizer command-line interface. These options are not listed in the standard help output (--help) and are meant for specific, advanced use cases. Non-public options are passed after specifying all public options. They are key-value pairs provided in a sequence. It's essential to ensure that each option key is followed by its corresponding value.

3.5.1 Use Cases of Advanced Options

The following is an example of using one of the advanced option exclude_act_func_dir:

The exclude_act_func_dir non-public option allows users to specify a csv file containing activation functions that should not be quantized. This is particularly useful when certain custom or specific activation functions need to be excluded from the quantization process.

The csv file should be written in the following format.

・Encoding format

Character encoding: UTF-8

Newline code: LF

・Format of activation functions to be excluded from quantization target

Specify one pattern of activation functions to be excluded from quantization per line. The pattern lists the ONNX operators that make up the activation function in sequential order from the input side. In addition, by adding the special character "^" to the beginning of the ONNX operator, the ONNX operator can be quantized.

Example:

```
Mish
Softplus,Tanh,^Mul
Exp,Add,Log,Tanh,^Mul
```

In this example, the following three activation functions are not quantized. However, since "^" is added to the Mul operator included in the patterns (2) and (3), it is subject to quantization (See Figure3.5, Figure3.6.)

(1) Mish

(2) Softplus → Thanh → Mul

(3) Exp → Add → Log → Tanh → Mul

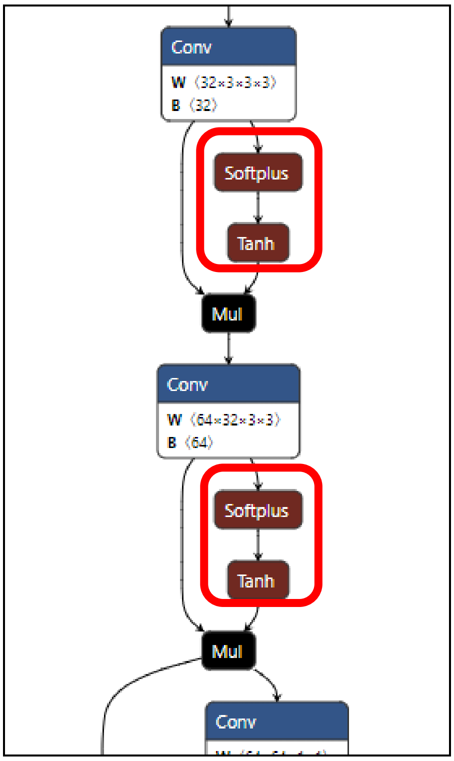


Figure3.5 Nodes that are not subject to quantization in pattern (2) (Surrounded by Red Round-Cornered Rectangles)

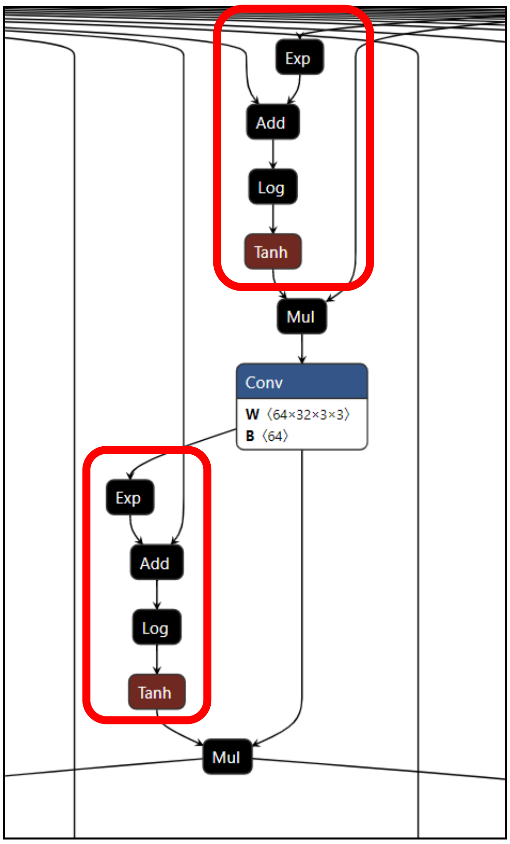


Figure 3.1 Nodes that are not subject to quantization in pattern (3) (Surrounded by Red Round-Cornered Rectangles)

After understanding how to determine nodes that are not subject to quantization and write nodes to a csv file, we will use CLI with PYTHONAPI to quantize the model using this advanced option. The following are the example command line and sample code snippet:

CLI example:

```
python3 -m drpai_quantizer.cli_interface ¥                  # Invoke the cli_interface module
 --input_model_path <path_to_target_onnx_file>¥             # Input: FLOAT format ONNX file
 --output_model_path <path_to_output_onnx_file> ¥           # Output: INT8 ONNX file
 --calibrate_dataset <path_to_calibration_data>¥            # Dataset for quantization calibration
 --datareader_path <path_to_calibration_datareader> ¥       # Dynamic import the calibration datareader
 --norm_mean <mean_value> ¥                                 # The normalize mean value
 --norm_std <standard_deviation> ¥                          # The normalize standard deviation
 exclude_act_func_dir <path_to_created_csv_file>            # Use the csv file to labeling nodes that do not need to be quantized
```

PythonAPI example:

```
nchw_datareader=NCHWDataReader('<path_to_calibration_dataset>', ¥
                               <mean_value>, ¥
                               <standard_deviation>, ¥
                               '<path_to_target_onnx_file>')

do_preprocess(input_model_path='<path_to_target_onnx_file>',¥
              preprocessed_model_output='<path_to_preprocessed_onnx_file>',¥
              preprocess_mode='<preprocess_mode>')

quantize_interface(input_model_path='<path_to_preprocessed_onnx_file>', ¥
                   output_model_path='<path_to_output_onnx_file>', ¥
                   calibrate_method=CalibrationMethod.<quantization_method>, ¥
                   datareader=nchw_datareader, ¥

                   exclude_act_func_dir='<path_to_created_csv_file>')
```

4. Testing Accuracy Obtained through Inference

4.1 Testing Accuracy with Sample Data

4.1.1 Testing Accuracy with the PyTorch-Quantized ONNX File

Testing the accuracy by using the PyTorch-quantized ONNX file in NCHW format (channels first) is enabled by setting the --nchw option as well as name of the input model file in the sample inference script as shown below.

```
python3 inference_resnet.py --nchw modA_resnet18_q.onnx
```

4.1.2 Testing Accuracy with the Keras-Quantized ONNX File

Testing the accuracy by using the Keras-quantized ONNX file in NHWC format (channels last) requires setting the --nhwc option as well as name of the input model file in the sample inference script as shown below.

```
python3 inference_resnet.py --nhwc modB_resnet18_q.onnx
```

4.2 Testing Accuracy with the User's Dataset

(Detailed Process of Modifying the Inference Script for Testing Accuracy (4 of 1.1.1)

Quantization and testing of accuracy in the included inference_resnet.py script is performed with a CIFAR-10 dataset. Using a different dataset requires editing of the inference script. Note that using a preprocessed dataset in a trained script particularly requires preprocessing of both the inference script and the quantization script. The input data levels in the included inference script are quantized by using a PyTorch-trained ONNX file as a preprocessing example for the dataset as shown in the listing below.

```
A preprocessing example for the dataset in the inference_resnet.py file:
transform_test = transforms.Compose([
   transforms.ToTensor(),
   transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])
```

Furthermore, CIFAR-10 images are loaded in the inference script as shown in the listing below.

```
  trainset = torchvision.datasets.CIFAR10(root='./', train=True,
                                          download=True, transform=transform_train)
  trainloader = torch.utils.data.DataLoader(trainset, batch_size=1,
                                           shuffle=True, num_workers=2)

  testset = torchvision.datasets.CIFAR10(root='./', train=False,
                                         download=True, transform=transform_test)

  testloader = torch.utils.data.DataLoader(testset, batch_size=1,
                                           shuffle=False, num_workers=2)
```

The examples of quantizing and loading shown above are for reference for preprocessing in testing the accuracy with the user's dataset.

4.3 Command-Line Options for the Sample Inference Script

lists the command-line options available in running the inference_resnet.py file as the sample inference script. The default setting is applied when an option is not specified.

Table 4.1 Command-Line Options for the inference_resnet.py File

Option	Abbreviation	Default Setting	Outline
--nhwc	—	(Disabled)	Setting the shape of the input layer as channels last format
--nchw	—	(Disabled)	Setting the shape of the input layer as channels first format

5. Suppressing Post-quantization Accuracy Degradation

This chapter describes some examples of methods to prevent accuracy degradation after quantization. If the accuracy is significantly lower than before quantization, please refer to the following for confirmation.

Note: The recommended measures are not guaranteed to always suppress unacceptable deterioration of the accuracy.

5.1 Basic Institutional Deterioration Control Methods

The following is a basic list of items to be checked.

- Confirming that the model is quantizable
  Note: Quantization is not guaranteed for all models
- Using the --calibrate_method option to switch between the MinMax and Entropy calibration methods
- Confirming that the data used in training or testing is quantized
- Confirming that the number or composition of data for use in calibration is appropriate.
  - An error may occur during quantization process with the Entropy specified to the --calibrate_method option. Please reduce the number of calibration data and perform quantization again.
  - Try to increase or decrease the number of calibration data.
  - Try to replace the composition of calibration data.
- Confirming that the implementation of preprocessing in the corresponding training or testing with the use of the quantization and inference scripts.

5.2 How to Suppress Accuracy Degradation in Specific Models Such as YOLOv4 and YOLOv5

5.2.1 Excluding Specific Activation Functions from Quantization

The --exclude_operate option can be used to exclude operations that comprise the activation function from quantization, thereby minimizing precision degradation.

Example: In the case of YOLOv4

If the operations that make up the activation function, the Mish function, are exp,add,log,tanh, specify exp,add,log,tanh in the --exclude_operate option (in no particular order) as in the following command to exclude them from quantization.

```
python3 -m drpai_quantizer.cli_interface ¥
        --input_model_path yolov4.onnx ¥
        --output_model_path yolov4_q.onnx ¥
        --calibrate_dataset ./calibrate_images ¥
        --exclude_operate exp,add,log,tanh
```

If the operations that make up the Mish function are softplus,tanh, specify softplus,tanh in the --exclude_operate option.

A more advanced method of specification is to use the exclude_act_func_dir option. For details, see 3.5.1 the advanced options in DRP-AI Quantizer.

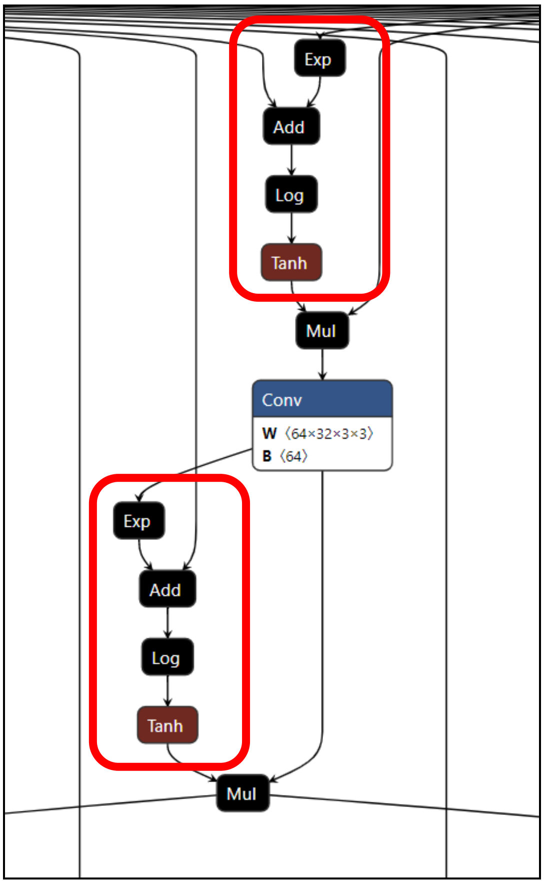


Figure 5.1 Operations exp,add,log,tanh Nodes that Make Up the Mish Function in YOLOv4 (Surrounded by Red Round-Cornered Rectangles)

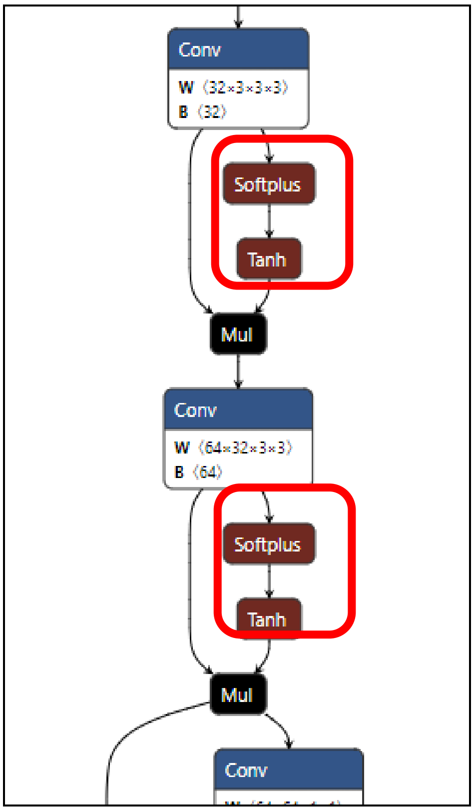


Figure 5.2 Operation softplus,tanh Node that Constitutes the Mish Function in YOLOv4 (Surrounded by Red Round-Cornered Rectangles)

Table 5.1 Actual Accuracy When YOLOv4 is Quantized Using the Methods Described in This Chapter

Conditions	Baseline	mAP IoU 0.5
Number of calibration data: 10 Calibration method: MinMax	49.9%	49.8%(0.1% degradation)

Example: In the case of MobileNetV2 or DeepLabV3

For networks that use RELU6 as the activation function, such as MobileNetV2 and DeepLabV3, RELU6 is excluded from quantization. However, since "ReLU6" is replaced by the "Clip" operation in the ONNX file, "Clip" is actually excluded from quantization as in the following command.

```
python3 -m drpai_quantizer.cli_interface ¥
        --input_model_path model.onnx ¥
        --output_model_path model _q.onnx ¥
        --calibrate_dataset ./calibrate_images ¥
        --exclude_operate clip
```

5.2.2 Exclude a Post Processing of a Neural Network from Quantization

Specifically, the following procedure can be used to exclude post-processing from the target to suppress the degradation of accuracy.

1. Check all the node names of the post-processing to be excluded from quantization on the ONNX file.
   It is convenient to check the node names using an external tool*.
   *Netron (https://github.com/lutzroeder/netron) is one example
2. Add the names of the non-quantizing nodes in a list then concatenate into a Single String, Finally use the PythonAPI interface to execute quantization.

```
postprocessing_nodes = ["Reshape_261",…<Enumerate node names to be excluded from quantization>]
postprocessing_nodes_exclude = ",".join(postprocessing_nodes)

nchw_datareader = NCHWDataReader('<path_to_calibration_dataset>', ¥
                                <mean_value>, ¥
                                <standard_deviation>, ¥
                                '<path_to_target_onnx_file>')

do_preprocess(input_model_path ='<path_to_target_onnx_file>',¥
              preprocessed_model_output ='<path_to_preprocessed_onnx_file>',¥
              preprocess_mode ='<preprocess_mode>')

quantize_interface(input_model_path='<path_to_preprocessed_onnx_file>', ¥
                   output_model_path ='<path_to_output_onnx_file>', ¥
                   calibrate_method = CalibrationMethod.<quantization_method>, ¥
                   datareader = nchw_datareader
                   node_to_exclude = postprocessing_nodes_exclude
```

Example: In case of YOLOv5n6

The nodes circled in red in the figure below are the nodes that perform post-processing of the neural network.

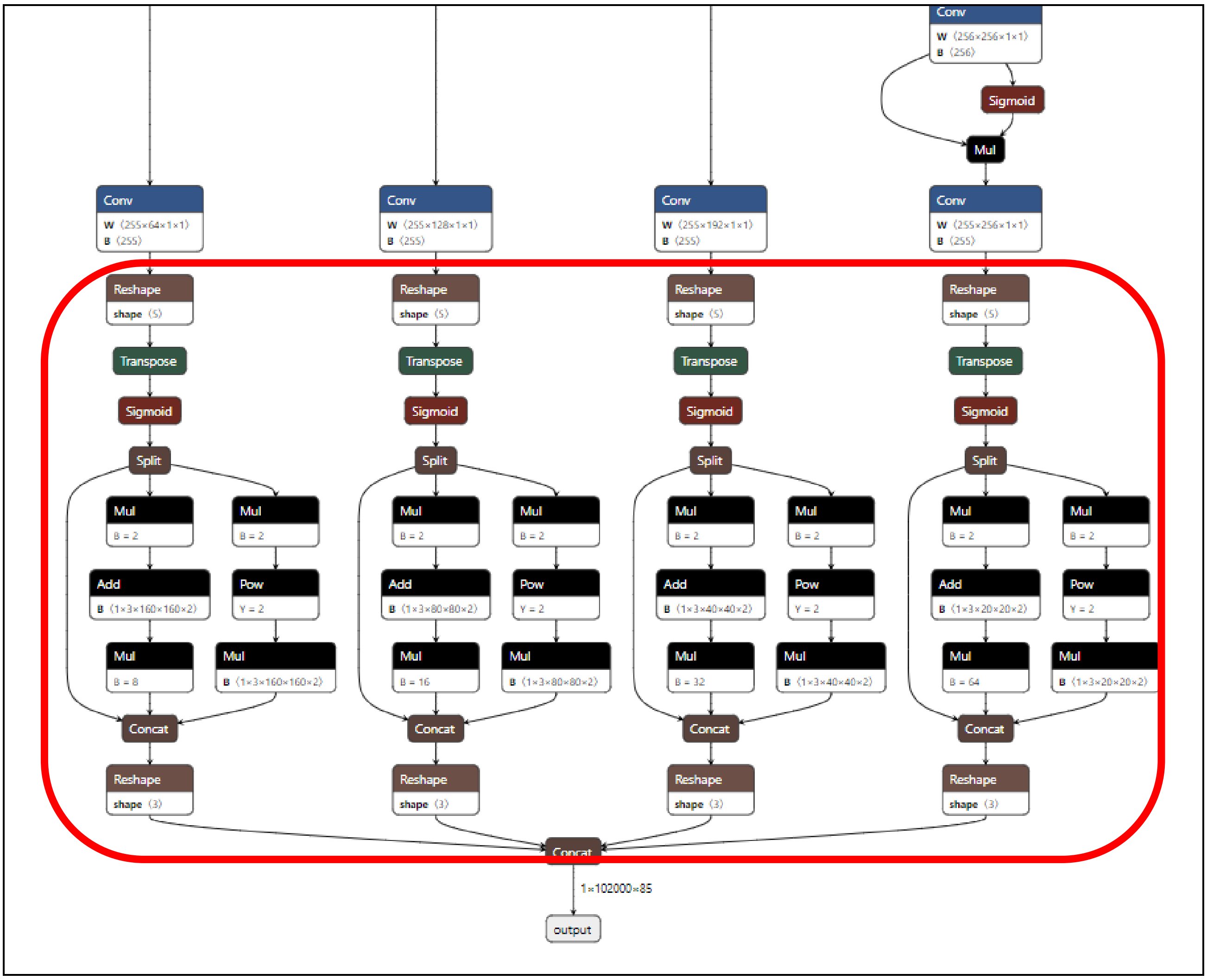


Figure 5.3 YOLOv5n6 Post-processing Node (Surrounded by Red Round-Cornered Rectangles)

The specific names of the nodes for post-processing are as follows.

```
"Reshape_261", "Reshape_280", "Reshape_299", "Reshape_318", "Concat_277", "Concat_296", "Concat_315",
"Concat_334", "Reshape_278", "Reshape_297", "Reshape_316", "Reshape_335", "Concat_336", "Split_264",
"Split_283", "Split_302", "Split_321", "Mul_266", "Mul_272", "Mul_285", "Mul_291", "Mul_304", "Mul_310",
"Mul_323", "Mul_329", "Add_268", "Pow_274", "Add_287", "Pow_293", "Add_306", "Pow_312", "Add_325",
"Pow_331", "Mul_270", "Mul_276", "Mul_289", "Mul_295", "Mul_308", "Mul_314", "Mul_327", "Mul_333",
"Transpose_262", "Transpose_281", "Transpose_300", "Transpose_319", "Sigmoid_263", "Sigmoid_282",
"Sigmoid_301", "Sigmoid_320"
```

Table 5.2 Actual Accuracy When YOLOv5n6 is Quantized Using the Methods Described in This Chapter.

Conditions	Baseline	mAP IoU 0.5
Number of calibration data: 30 Calibration method: Entropy	54.2%	53.1(1.1% degradation)

6. Usage Notes

This section gives notes on the usage of the DRP-AI Quantizer.

- Installing the software in the required version of this tool will overwrite the existing software if any is present. Therefore, confirm that the installation will not affect other software that is in use.
- The development of this tool was based on ONNX Runtime v1.14.1, so the tool provides features in accord with the quantization in ONNX Runtime along with the changed or added features.
- The opset version available for input ONNX files is 12.
- The sample inference script includes processing to download a CIFAR-10 dataset from the torchvision.datasets. This requires additional time for downloading in the first round of inference.
- When Error occurs, refer to the following solutions:
  - Error Messages 1: `` `Exception: Pre-processing before quantization was Failed.` ``

    Try re-quantizing the target model using the command line option "`-skip_preprocess`" or skip calling the "`do_preprocess()`" API when quantizing a model in PYTHONAPI.
  - Error Messages 2: `` `abort - cause abnormal process termination` `` or `` `killed` ``(the task been killed) when executing entropy quantization.

    Reduce the amount of calibration data.

    Entropy quantization typically involves calculating histograms or distributions of data, which can be memory-intensive, especially if the dataset is large or the histograms are high resolution.

    Take the provided “nchw_datareader.py” as an example, the following Figure6.1 is the code snippet of NCHWDataReader class, The `` `size_limit` `` attribute in the class determines how many images from the calibration dataset are used. You can adjust the `` `size_limit` `` attribute of the NCHWDataReader class to control the amount of calibration data used during the quantization process.

```
def __init__(self, calibration_image_folder, norm_mean, norm_std, augmented_model_path):
    self.augmented_model_path = augmented_model_path
    self.preprocess_flag = True
    self.enum_data_dicts = []
    self.datasize = 0
    self.size_limit = 0
    self.norm_mean = self.check_list_data("norm_mean", norm_mean)
    self.norm_std = self.check_list_data("norm_std", norm_std)
```

Figure 6.1 Code snippet of nchw_datareader.py

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Dec.15.23	—	First edition issued
1.01	Jan.26.24	3, 8	To adapt to DRP-AI Translator i8, delete the content in Chapter 2 regarding the setting up of DRP-AI Quantizer.
1.02	Feb.13.24	7	Describes what is updated in DRP-AI Quantizer's version 1.01

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.