Lattice sensAI Neural Networks Quantizer

User Guide

FPGA-UG-02225-1.2

December 2025

Disclaimers

Inclusive Language

Contents

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviation	Definition
CUDA	Compute Unified Device Architecture
CuDNN	CUDA Deep Neural Network library
DNN	Deep Neural Network
FPQ	Fixed Point Quantization
FPGA	Field Programmable Gate Arrays
GEMM	General Matrix Multiplication
GPU	Graphics Processing Unit
LATTE	Lattice SensAI Neural Networks Training Environment
LSQ	Learned Step-size Quantization
LSCQuant	Lattice SensAI Neural Networks Quantizer
QAT	Quantization Aware Training
WSL2	Windows Subsystem for Linux version 2

1. Introduction

This document provides guidelines for installation and use of the Lattice sensAI™ Neural Networks Quantizer (LSCQuant) package. The package provides a set of TensorFlow Keras layers that can be used to train quantized DNN suitable for Lattice Semiconductor FPGAs.

Lattice Semiconductor FPGAs support a subset of Keras layers, which means they do not require a quantized version of the layer. Some other layers require a quantized version, which must be used instead of the original Keras layer. These layers support two different quantization methods:

1. Fixed Step-size quantization, also referred to as FPQ.
2. Quantized learned step-size quantization, also referred to as Learned Step-size Quantization LSQ

Besides quantization methods, there is also the option of performing no quantization at all. This allows the layers to act exactly as their non-quantized Keras counterpart. It is possible to define and train a DNN with or without quantization.

Blocks of layers are provided for model architectures supported by Lattice FPGAs. A block of layers is defined as a class that is instantiated and then applied to input tensor(s). When defining a Keras functional model, a block behaves like a single Keras layer, but when called on an input tensor, it creates multiple layers within the model. Only the individual layers created by the block appear in the model. If a name is specified when instantiating a block, that name is used as a prefix for all layers the block creates, making it easier to identify which block generated each layer in the model.

To understand more about the supported Keras layers and the provided quantized version, refer to documentation included in the LSCQuant package.

2. Prerequisites and Installation

2.1. Prerequisites

This section lists all the prerequisites and system requirements for using the LSCQuant package.

- Python version 3.8 or later
- TensorFlow version 2.7 to 2.15 (2.16 or later is not supported)

Note:

TensorFlow 2.10 is the last release that supports native GPU training on Windows. When training on Windows with a GPU, ensure the installed TensorFlow version does not exceed 2.10. There is also the option to perform training through WSL2. Refer to Install TensorFlow with pip for more information.

2.2. Python Environment

LSCQuant is a Python package that needs to be installed in the Python environment used for training DNN. If an environment has already been created for other SensAI tools such as the Lattice SensAI Neural Networks Training Environment (LATTE), activate that environment and then install LSCQuant in it. If there is no such environment, then it should be created. To create and manage Python environments more easily, especially for dependencies such as CUDA and CuDNN it is recommended to use Miniconda or Anaconda. Miniconda can be downloaded from this link: Installing Miniconda. After installation, you can create a new Python environment by running the following commands in a CLI:

```
conda create -n fpga python~=3.10.0
conda activate fpga
```

In this example, the environment is named *fpga*, but you can choose any name. If you plan to use a GPU for training DNNs, install CUDA and CuDNN in the environment using the *conda* command:

```
conda install -c conda-forge cudatoolkit=11.2 cudnn=8.1.0
```

From this point on, any other Python packages should be installed using the pip command in the environment. The TensorFlow package should be installed as follows:

```
pip install tensorflow~=2.10.0 numpy~=1.24.0
```

2.3. Package Installation

To install LSCQuant in the Python environment that is currently activated, the following command must be run from the folder containing the LSCQuant Wheel (`.whl`) file:

```
pip install -U ./lscquant-x.y.z-py3-none-any.whl
```

where x.y.z is dependent on the exact version of the package. On Windows, replace the character / by \ in the command above.

To test if the package is correctly installed, the following command can be run to display the installed version:

```
python -c "import lscquant; print(lscquant.__version__)"
```

3. Usage

The package provides some convenience functions along with the quantized version of certain Keras layers. These features simplify the process of defining a quantized DNN model by converting an existing Keras model. This is performed by automatically replacing layers that require quantization with the quantized layer provided in LSCQuant. They also allow you to specify a quantization scheme to apply to all the quantization layers in the model.

The built-in schemes include no-quantization, fpq-default, and lsq-default. There are also other variations of the LSQ default schemes such as using 4-bit weights for some layers instead of 8-bit weights. The complete list can be found in the LSCQuant documentation that is provided with the package.

It is important to note that only Keras functional models can be quantized with LSCQuant. Custom models, defined as a subclass of the Keras Model class, cannot be quantized. For more information about Keras functional models, visit this link: The Functional API.

3.1. Quantization Aware Training with Keras

Performing QAT with Keras requires you to use functions from the package to build the quantization model and save it to a format that can be used by sensAI Compiler.

The example below is based on the standard TensorFlow Keras example available here: Training and evaluation with the built-in methods.

The first step involves loading a dataset for training, which in this case consists of images, and ensuring the image value ranges are normalized for use on FPGA:

```
from tensorflow import keras
from lscquant.model import build_quantization_model, save_model, load_model
from lscquant.layers import set_post_training_calibration_phase
from lscquant.processing import normalize_image

# Load train datasets
# Data type should be float32
train_x, train_y = ...

# Make sure that the model input values are normalized between in the range [0, 2).
# For unsigned 8 bits images (pixel values in the range [0, 255]), this can be
# achieved using the normalize_image function; for data with a different input range,
# the function normalize_values can be used
train_x = normalize_image(train_x)
```

The next step is to create a Keras model, build a quantized model from it, and train it:

```
# An instance of a pure Keras sequential or functional model
model = keras.Model(...)

# Build quantization model from the Keras model
quant_model = build_quantization_model(model, scheme="lsq-default")

# Compile the model with the optimizer, losses, metrics, etc.
quant_model.compile(optimizer="adam", loss=...)

# Train the model
quant_model.fit(x=train_x, y=train_y, epochs=100, ...)
```

```
# Post-training calibration (for LSQ-based quantization schemes)
set_post_training_calibration_phase(True)
quant_model.predict(train_x)
set_post_training_calibration_phase(False)
```

Note that in case of LSQ-based quantization, a post-training calibration phase is required.

The final step is to save the model and possibly reload it later in another session:

```
# Save model
save_model(quant_model, "my_quantized_model.h5", include_optimizer=False)

# (in another session) reload model
quant_model_loaded = load_model("my_quantized_model.h5", compile=False)
```

For more information, refer to documentation included in the LSCQuant package.

3.2. QAT with Lattice sensAI Neural Networks Training Environment

The simplest way to perform QAT with LSCQuant is by using it through LATTE. This is achieved by adding a callback declaration to the list of callbacks in a LATTE experiment configuration file:

```
callbacks:
    - class_name: "lscquant"
      config:
          scheme: "lsq-default"
```

When using LATTE for training, testing, and converting DNN models, the proper LSCQuant step will be performed automatically. The Keras model provided in the configuration file will be automatically quantized, and a calibration phase will be performed after the last epoch of training using the training dataset.

Note that the dataset handler should ensure that the model input is normalized properly, as shown in the previous section example.

For more information about LATTE usage and LSCQuant callback configuration parameters, refer to Lattice sensAI Neural Networks Training Environment User Guide (FPGA-UG-02226).

Reference

- Lattice sensAI Neural Networks Training Environment User Guide (FPGA-UG-02226)
- Lattice sensAI Neural Networks Quantizer Version 1.3.0
- Lattice Insights for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.2, December 2025

Section	Change Summary
Abbreviations in This Document	Added *GEMM*.
Introduction	Added description about blocks.

Revision 1.1, May 2025

Section	Change Summary
Prerequisites and Installation	Updated the Miniconda installation link in the Python Environment section.
References	Replaced *LSCQuant web page (variation 1.3.0)* with *Lattice sensAI Neural Networks Quantizer Version 1.3.0*.

Revision 1.0, November 2024

Section	Change Summary
All	Initial release.