HEIG-VD REDS Institut Reconfigurable and Embedded Digital Systems

Prof. Yann Thoma,
Marina Zapater,
Assist. Mehdi Akeddar,
Guillaume Chacun,
Thomas Rieder

Artificial Intelligence for Autonomous Systems
Spring 2024

Lab 4 - Byte-Sized Brains : The Pocket AI Lab

This lab will be graded.

Please answer the questions in a separate text file. You can compile it as a pdf. The questions are marked in blue.

The parts of the code are marked with ADD YOUR CODE HERE in the jupyter notebook. Please read carefully the function declaration to get the necessary information about the meaning of the parameters.

Available time : **2 lab sessions**.

Please upload your code and text file containing your answers on cyberlearn before the next lab session scheduled for **May 21th**.

1 Lab objectives

In this lab, you will get familiar with the concepts of quantization. You will quantize weights and activations, and use NNTool to get a quantization tailored for the drone.

2 Requirements

For this lab, you'll need to create a virtual environment to ensure the correct operation of the jupyter notebook or use a Google Colab. If you choose the first option, use pip to install the following packages :

— torch

— torchvision

— matplotlib

Then run the first part of the notebook to get the data, train your model and get a first test accuracy.

3 Introduction

Machine learning models are commonly trained using 32-bit floating-point data, yet this comes with significant costs in terms of hardware implementation—requiring considerable area, performance, and energy. Moreover, while 32 bits of precision are often necessary during training to capture minute gradient steps, such precision is generally unnecessary during inference. Lowering arithmetic precision can offer savings in circuit area and memory bandwidth, aiding hardware designers in cost reduction across multiple aspects.

Consequently, DSPs, and more recently ML accelerators, often adopt fixed-point arithmetic at significantly lower precisions. While 8-bit unsigned integers are frequently targeted, some accelerators even employ single-bit binary arithmetic. There are two primary approaches to quantization : post-training quantization and quantization-aware training.

4 Stage 1 : Weights quantization

Essentially, weight quantization involves the process of reducing the precision of numerical values representing the parameters (weights) of a neural network, typically from floating-point to lower-bit fixed-point representations. To complete this stage, follow these steps :

— Visualize the weight distribution of every convolutional and fully connected layer. Use histograms.

— *Record the range of the weights, as well as their 3-sigma range (the difference between $\mu - 3\sigma$ and $\mu + 3\sigma$).*

— Implement the function `quantized_weights`.

— *Explain which range you used for your quantization. Does range have an impact on model performance in this case ? Explain your answer.*

— *Do you observe a drop in the general accuracy ? If you did everything right, it should be negligible. Explain your findings.*

— *Compare the memory footprint of the original model and the quantized one. Did the memory footprint change ? Explain your findings. You can use torchinfo or torch-summary to get the memory footprint.*

5 Stage 2 : Activation quantization

Quantization of activations (**input and output of layers**) complements weight quantization by extending the compression benefits to the intermediate values within a neural network. This process involves reducing the precision of activation values, optimizing memory usage, and enhancing computational efficiency without significantly compromising model performance. To complete this stage, follow these steps :

— Visualize the activation distribution of every convolutional and fully connected layer. Use histograms.

— *Record the range of the activations, as well as their 3-sigma range.*

— *Develop a formula for the quantized output of $conv_1$: O_{1q}, as a function of s_{W1}, s_{O1}, s_I, I_q, W_{1q}.*

— *Develop a general formula for the quantized output of $conv_i$: O_{iq}, as a function of s_{Wi}, s_{Oi}, s_I, I_q, W_{iq}, and the scaling factors of the previous layers. Try to do it recursively by starting with $conv_{2q}$ to extract the formula.*

— Complete the functions `quantize_initial_input`, `quantize_activations`, and `forward`.

To guide you toward the development of the formula, let's dive into computations of the activation distribution of every convolutional and fully connected layer. Let O be the output of a layer, I the input, and W the weight matrix. We can note that :

$$O = IW \tag{1}$$

When we introduce quantization, we can consider that :

$$W \approx s_W * W_q \tag{2}$$

With s_W **being the scale factor of the corresponding matrix W** and W_q the quantized matrix. Here a matrix can represent the weights, the inputs, or the outputs. With this information in hand, you can develop a general formula for the quantized output. We also need to modify the forward method of the model to introduce quantization to the outputs of each layer.

6 NNTool quantization

NNTool is a specialized tool for automating the quantization process of neural network models. It streamlines the quantization workflow by analyzing the model structure, identifying opportunities for quantization, and applying quantization techniques to optimize the model for deployment on hardware with limited resources. NNTool is kind of tailored for our drone, which makes it an interesting solution to use.

The objective of this part is to get familiar with the use of NNTool to quantize our model and prepare it to be deployed on the AI deck. The `show` command will let you visualize each layer, its parameters, input sizes, etc. We recommend using it after each command to understand what happened.

Please run the following commands to prepare your environment :

```
$ (source l'env gap SDK)
$  pip install numpy==1.23
$  pip install scikit-learn --upgrade
```

We also provide an onnx version of our model (`model_lab4.onnx`).

Run the `nntool` command to start a nnTool terminal. We will start by opening the model :

```
$ open model/model_lab4.onnx
```

This command will invoke your model inside the nntool bash. Now you can run the `adjust` command.

— *What's the objective of the adjust command ?*

As we saw in the quantization of activations, we don't need to quantize the output of certain layers. So we can consider the fusion of those layers with the one that needs quantization, to compress the model.

— *With the help of the nnTool documentation* [1]*, provide the right command for layer fusions.*

— *Which layers have been fused ?*

Now that our model architecture is ready for quantization, we can start the process. We provide a set of images to be used during this part of the lab, in the folder `img`.

— *Based on the first parts of this lab, explain why we need a set of images for our quantization.*

— *What should be the properties of this set of images ? Think in terms of diversity of images.*

Run the following commands :

```
$ aquant img/* -T -H 32 -W 32 --scheme SQ8 -D 255
$ imageformat input_1 rgb888 offset_int8
$ save_state
```

You can use the following commands to inspect the outputs of the original and quantized models :

```
$ dump 'path_to_image' -S tensors_1
$ dump 'path_to_image' -S tensors_2 -q
$ tensors -t tensors_1 tensors_2 -s 2
```

You can see that the second tensor contains only integers. In the model folder, run the following command :

```
$ nntool -g -M . -m model.c -T tensors -H model.h model_lab4.json
```

— Explore the generated files

In your generated file `model.h`, find the quantization constants (`OUT_SCALE`) and explain why they may be different than the ones you had to compute in stages 1 and 2.

1. `https://github.com/GreenWaves-Technologies/gap_sdk/blob/master/tools/nntool/README.md`