Lab 1 Quantization

An Lihua
2022131001
Shanghaitech University, SIST
anlh2022@shanghaitech.edu.cn

***Abstract*—In this lab, we will quantize a convolutional neural network (CNN) using pytorch package. In the notebook, CIFAR10 dataset is downloaded from Internet and a floating-point model is trained first. Using this pre-trained model, the distributions and ranges of the weights and activations are found. Then the quantization is performed according to the results. At last, the biases of the last layer in the CNN are considered. The accuracy loss should be negligible if everything works well by running the notebook (.ipynb) file.**

***Index Terms*—quantization, CNN**

I. Initial Setup

Before beginning the assignment, we import CIFAR10 dataset first, and doing initial setup to prepare for the quatization of weights and activations. The provided network is similar to LeNet-5, and none of the layers have a bias associated with them.

Layer	Type	Input	Output	Activation
conv1	Convolutional	3x32x32	12x28x28	ReLU
pool1	Max pool	12x28x28	12x14x14	None
conv2	Convolutional	12x14x14	32x12x12	ReLU
pool2	Max pool	32x12x12	32x6x6	None
fc1	Fully-connected	1152	256	ReLU
fc2	Fully-connected	256	64	ReLU
fc3	Fully-connected	64	10	None

Then we train and test the dataset, get the test accuracy of the network on the test image is **60.88%**.

II. Visualize Weights

A. Plot Histograms of the Weights of each Layer

The results are shown in Figure 1.

B. 3-sigma Range

- For convolutional Layer 1, actual range is: [-0.6182, 0.5482], 3-sigma range is: [-0.5282, 0.5327].
- For convolutional Layer 2, actual range is: [-0.3643, 0.4440], 3-sigma range is: [-0.3343, 0.3052].
- For fully-connected Layer 1, actual range is: [-0.1489, 0.1432], 3-sigma range is: [-0.0728, 0.0701].
- For fully-connected Layer 2, actual range is: [-0.2099, 0.2348], 3-sigma range is: [-0.1491, 0.1513].
- For fully-connected Layer 3, actual range is: [-0.5104, 0.5138], 3-sigma range is: [-0.4914, 0.4905].

From the results above, we can find the 3-sigma range of **conv2, fc1, fc2, fc3** are smaller than the actual range, 3-sigma range of **conv1** is larger than the actual range.

For quantization, **conv1** is a good choice as it has a larger 3-sigma range.

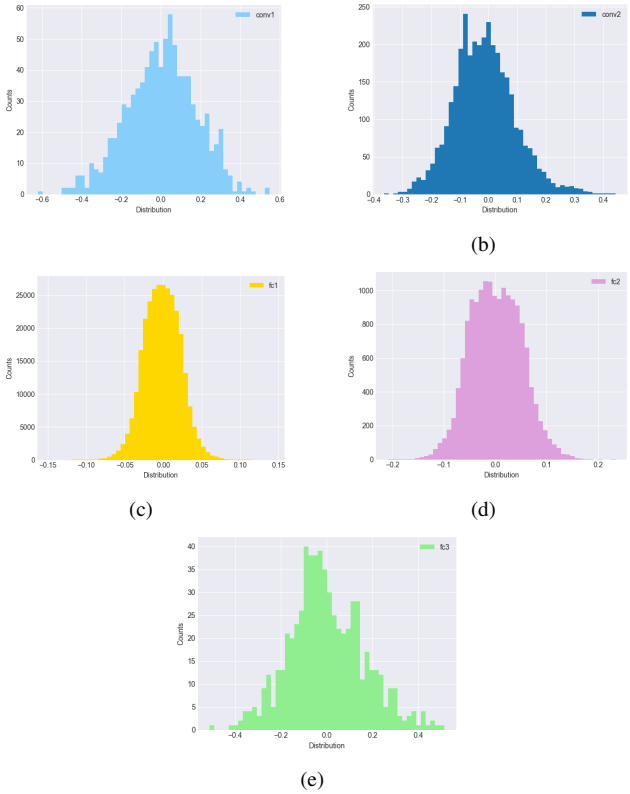


Fig. 1. Distributions of each layer. (a) convolutional layer 1, (b) convolutional layer 2, (c) fully-connected layer 1, (d) fully-connected layer 2, (e) fully-connected layer 3.

III. Quantize Weights

A. quantized_weights function

To find a scaling factor n_W for each convolutional and fully connected layer, which would fit inside an 8-bit signed integer, the equation can be described as the following:

$$n_W W * In = n_W Out \tag{1}$$

Since we consider only symmetric quantization, according to the quantization guide provided by NIVIDIA [2], as shown in Figure 2, the scaling factor can be get by:

$$scale = \frac{|max|}{127} \tag{2}$$

Here we use the actual range for calculation.

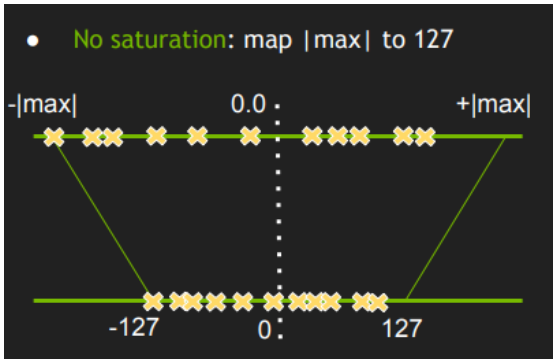


Fig. 2. Symmetric quantization.

Then according to $r = (S \cdot q - Z)$ [3], where r is the real number, q is the quantized number, S is the scaling factor and Z is zero point. Since Z=0 in our case, the quantized result is:

$$q = \frac{r}{S} \tag{3}$$

B. Result after Quantizing Weights

The accuracy of the network after quantizing all weights is **60.9%**, the accuracy change is negligible.

IV. Visualize Activations

Now we have quantized the weights of the CNN, we must also quantize the activations (inputs and outputs to layers) traveling through it. Before doing so, we need to analyze what values the activations take when travelling through the network.

A. Plot Histograms of the Weights of each Layer

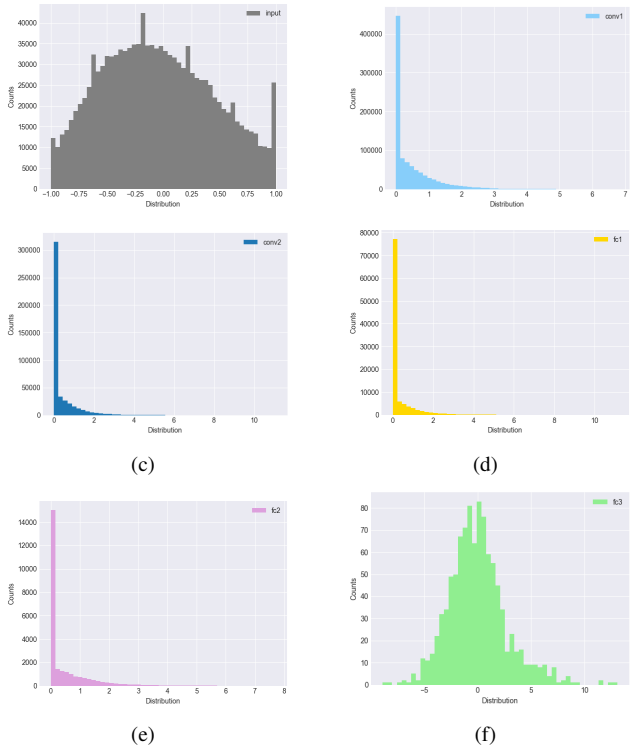
The results are shown in Figure 3.

B. 3-sigma Range

- For Input Layer, actual range is: [-1.0, 1.0], 3-sigma range is: [-1.5247382774508076, 1.430113747019963].
- For convolutional Layer 1, actual range is: [0.0, 6.795027732849121], 3-sigma range is: [-1.545606652766442, 2.4861031223104018].
- For convolutional Layer 2, actual range is: [0.0, 11.123934745788574], 3-sigma range is: [-1.7025899661791586, 2.4097507332214576].
- For fully-connected Layer 1, actual range is: [0.0, 11.145042419433594], 3-sigma range is: [-1.5848371084738495, 2.1411086499930922].
- For fully-connected Layer 2, actual range is: [0.0, 7.7190375328063965], 3-sigma range is: [-1.9397187614118885, 2.8914816223554594].
- For fully-connected Layer 3, actual range is: [-8.853764533996582, 13.063966751098633], 3-sigma range is: [-8.289666489254017, 8.252148309951473].

From the results above, we can find the 3-sigma range of **conv1, conv2, fc1, fc2, fc3** are smaller than the actual range, 3-sigma range of **input layer** is larger than the actual range.

For quantization, **input layer** is a good choice as it has a larger 3-sigma range.



(c) (d)

(e) (f)

Fig. 3. Distributions of each layer. (a) input, (b) convolutional layer 1, (c) convolutional layer 2, (d) fully-connected layer 1, (e) fully-connected layer 2, (f) fully-connected layer 3.

V. Quantize Activations

To quantize the activations (inputs and outputs to layers), the equation is:

$$n_W W * n_{In} In * n_{Out} = n_W n_{In} n_{Out} Out \tag{4}$$

where n_{In} is the scaling factor which was applied to the input layer, and n_{Out} is the scaling factor which we decide to apply to the output. n_{Out} must be chosen such that the expected values of the elements of Out can be scaled down to fit within 8 bits.

A. Description of the Output Layers

(a) The equation describing the output of the **conv1** layer with new scaling parameters:

$$\begin{aligned} &n_{W_{conv1}} W_{conv1} * n_{In} In * n_{Out_{conv1}} \\ &= n_{W_{conv1}} n_{In} n_{Out_{conv1}} Out \end{aligned} \tag{5}$$

(b) The equation describing the output of the **conv2** layer in terms of In, W_{conv1}, W_{conv2}, Out_{conv1}, Out_{conv2}, n_{In}, $n_{W_{conv1}}$, $n_{W_{conv2}}$, $n_{Out_{conv1}}$, and $n_{Out_{conv2}}$:

$$\begin{aligned} &\frac{n_{W_{conv2}} W_{conv2}}{n_{W_{conv1}} W_{conv1}} * n_{In} In * \frac{n_{Out_{conv2}}}{n_{Out_{conv1}}} \\ &= n_{W_{conv2}} \frac{n_{In}}{n_{W_{conv1}} n_{Out_{conv1}}} n_{Out_{conv2}} Out \end{aligned} \tag{6}$$

B. *quantized_activations function*

Here we also use actual range for calculation.

The function *quantize_initial_input()* means to get the scaling factor of input images, so the scaling method is the same as quantizing weights.

For function *quantize_activations()*, the new scaling factor depends on previous layer, like the equation given in [3]:

$$q_3^{(i,k)} = Z_3 + M \sum_{j=1}^{N} (q_1^{(i,j)} - Z_1)(q_2^{(j,k)} - Z_2) \tag{7}$$

$$M := \frac{S1S2}{S3} \tag{8}$$

where we take matrix q1 to be the weights, and matrix q2 to be the activations, the multiplier M depending on the quantization scales S1, S2, S3, which is consistant with the equation we wrote in last subsection.

C. *forword function*

For function *forward()* in the *NetQuantized()* class, we need to make sure all the output of each layer are integers between -128 and 127 using *torch.clamp()* and *torch.round()*.

D. *Result after Quantizing Activations*

The accuracy of the network after both weights and activations have been quantized is **60.92%**, still having little accuracy change.

E. *Bonus: Optimization*

Since we did not fully utilize the representation range of a signed 8-bit number ([-128, 127]) for the quantized activation, which may lead to relatively large quantization error. Here propose two methods that can better make use of an 8-bit number and reduce the quantization error.

(a) **KL-Divergence:**

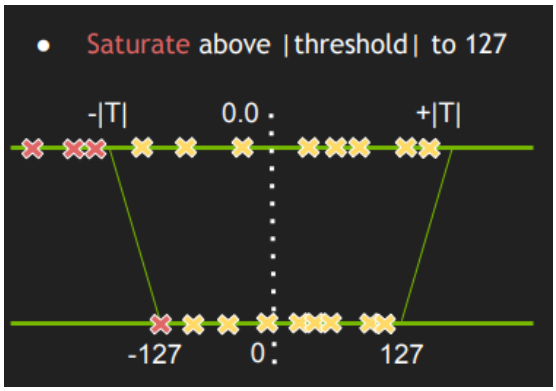


Fig. 4. Quantize using a threshold.

Different from MIN-MAX quantization, KL method is to find a threshold $|T| < max(|max|, |min|)$ and reflect it to [-127, 127], as shown in Figure 4 [2]. A proper threshold value do not affect the precise. KL-Divergence is to measure the similarity of two distributions. If the KL divergence value is smaller, it means that the two distributions are more similar, which means that the threshold is the better choice. For symmetric quantization, scaling factor can be calculated according to this threshold. The related algorithm is given in Figure 5 [2]:

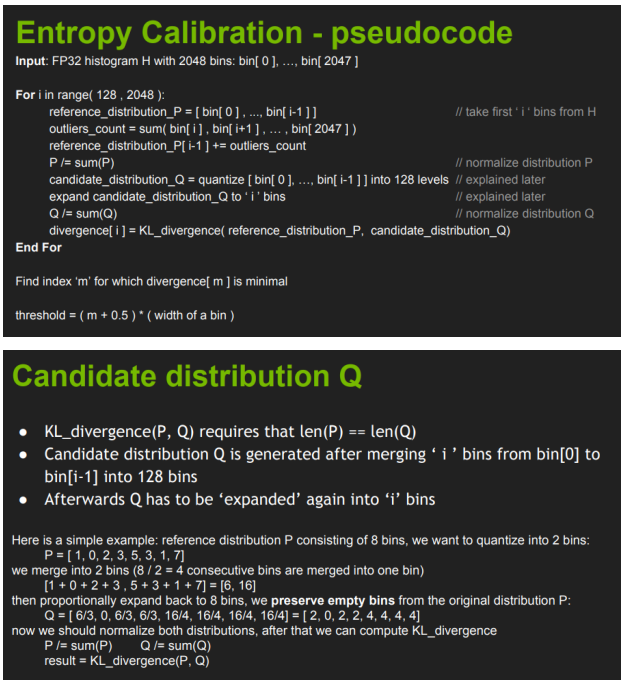


Fig. 5. KL Algorithm.

(b) **Learning Quantization Ranges [3]:**

Quantization ranges are treated differently for weight quantization and activation quantization.

For weights, the basic idea is simply to set a := min(W), b := max(W).

For activations, ranges depend on the inputs to the network. Collect [a, b] ranges on activations during training and then aggregate them via exponential moving averages (EMA) with the smoothing parameter being close to 1. Given significant delay in the EMA updating activation ranges when the ranges shift rapidly.

As a result, the learned quantization parameters map to the scaling factor, the workflow is given is Algorithm 1 [3].

Algorithm 1 Quantized graph training and inference

1: Create a training graph of the floating-point model.
2: Insert *fake quantization* TensorFlow operations in locations where tensors will be downcasted to fewer bits during inference according to equation 12.
3: Train in simulated quantized mode until convergence.
4: Create and optimize the inference graph for running in a low bit inference engine.
5: Run inference using the quantized inference graph.

VI. Quantize Bias

Now we need to update CNN to include a bias in the final layer, fc3. Consider how a bias affects the equation for an unquantized layer:

$$W * In + \beta = Out \tag{9}$$

where β is the bias.

A. Description of the Quantized Output Layers with a Bias

Suppose that we again quantized a biased layer with the same scaling factors used in previous questions, then we have equation:

$$n_W W * n_{In} In * n_{Out} + n_{Out}\beta = n_W n_{In} n_{Out} Out \tag{10}$$

The new accuracy of the network with a bias on the test images is **60.88%**. And the accuracy of the network on the test images after all the weights are quantized but the bias isn't is **57.07%**, which has a obvious reduction on the accuracy.

B. quantized_bias function

Since the bias we get is already scaled by the final layer's scale, we need to find a bias scale to modify the result bias to match the equation we give in Eq (10). As shown is [3], quantization scale should be the product of the scales of the weights and of the input activations:

$$S_{bias} = S_1 S_2 \tag{11}$$

C. Result after Quantizing the Bias

The accuracy before quantizing with the bias is **60.88%**, while after quantizing the bias is **60.9%**, the accuracy change is partially improved (or sometimes negligible).

References

[1] EE219 ICS 2022-Fall

[2] Szymon Migacz. 8-bit Inference with TensorRT. GTC San Jose, 2017

[3] Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. Benoit Jacob. arXiv:1712.05877