MicroNet Challenge Submission - Qualcomm-M0

Team: Yash Bhalgat, Jinwon Lee, Jangho Kim, Kambiz Yazdi, Hsin-Pai Cheng
Contact: ybhalgat@qti.qualcomm.com, jinwonl@qti.qualcomm.com

1 Description

Starting with the EfficientNet [2] architecture, we use Learned Stepsize Quantization (LSQ) [1] to quantize the model to a bit-width of 6 for both weights and activations of all layers. We design a novel Knowledge Distillation method to increase the accuracy of our quantized model to the required 75% accuracy threshold.

2 Implementation Details

2.1 Quantization

We use LSQ [1] to quantize the weights and activations of our network. The quantization scheme is as follows:

$$\bar{v} = \lfloor clip(v/s, L) \rceil \tag{1}$$

$$\hat{v} = \bar{v} \times s \tag{2}$$

This quantization-dequantizaiton scheme is implemented in `lsq_quantizer/utils/lsq_module.py`.

Symmetric and asymmetric quantization: In our submission, we use 6-bits. Hence, if we use symmetric quantization for a layer, the quantity $\bar{v}$ are constrained to the range:

$$[-31, -30, -29, \ldots, -1, 0, 1, \ldots, 29, 30, 31]$$

And if we use asymmetric quantization for a particular layer, the quantity is constrained to the range:

$$[0, 1, 2 \ldots, 61, 62, 63]$$

This is implemented in the `get_constraint` function in `lsq_quantizer/utils/utilities.py`

Weight quantization: We quantize all the weights in the Conv and Linear layers to 6-bits. The BatchNorm layers are kept unquantized. For weights, symmetric quantization is used.

Activation quantization: All the activations which go into the Conv/Linear layers as inputs are quantized to 6-bits. Hence, all the operations inside the Conv/Linear layers are 6-bit/6-bit operations. For the activations, symmetric and asymmetric quantization are used interchangeable as follows:

1. The ReLU layers are simply replaced with the asymmetric quantization layers. Because, as can be seen above, the asymmetric quantization layer automatically constrains the activations to be greater than 0 (in addition to quantizating them to the corresponding range).

2. There is no ReLU before some of the Conv layers (e.g. `_conv_stem`) and the final `Linear` layer. Hence, we want to preserve both the +ve and -ve activations that go into these layers. So, we use symmetric quantization for these input activations (namely, `*._in_act_quant`, `first_act`, `_head_act_quant0` and `_head_act_quant1`).

For the details on weight and activation quantization, refer to `lsq_quantizer/utils/effnet.py`

2.2 Training

The parameter s is trainable. The gradient update of the parameter s is as follows:

$$\frac{\partial \hat{v}}{\partial s} = \begin{cases} -v/s + \lfloor v/s \rceil & \text{if } |v/s| < L \\ \hat{v}/s & \text{otherwise} \end{cases}$$

For each layer in the network, there is one s-parameter for weights and one s-parameter for activations.

For training, we have 3 learning rates as described in [1]. We modify it and we have 3 learning rate parameters as follows:

1. `learning_rate`: The usual learning rate for the weights of the network
2. `weight_lr_factor`: We need a different learning rate for the s parameter for the weights. We define this learning rate as $weight\_lr\_factor \times learning\_rate$
3. `act_lr_factor`: This is same as above, just for the s parameter for the activations.

2.3 2-step Knowledge Distillation

3 Results

Model	Accuracy	#params	MAC	score
EfficientNet-b0	76.10%	5.3M	0.39G	1.1
+ LSQ (W6A6)	74.2%			0.22
+ 2-step KD	75.1%			0.22

4 Reproducibility

References

[1] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

[2] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.