CSE237C Final Project: Quantized Spiking Neural Networks

Jennifer Switzer

December 2020

1 Introduction

Implementing neural networks in hardware can help speed up inference, which is useful for time-sensitive applications. However, neural networks also tend to be energy and resource hungry [5]. This is a problem for many embedded applications, where power and compute resources may be limited.

One potential solution to this problem is spiking neural networks, an energy-efficient class of neural networks that are more closely modeled off of biological systems [7]. Unlike traditional neural networks, which assume that every input needs to be computed on, spiking neural networks only perform computations in response to an event . If nothing changes, no computations are performed and as a result SNNs are often less energy and resource intensive than traditional neural networks [3].

Another common technique for reducing the resource and energy usage of neural networks in hardware is quantization, the use reduced-precision data types (i.e. 8-bit or 16-bit integers rather than floats) [2]. Computing on lower bitwidth data types tends to reduce the throughput and area of the design, but at the cost of reduced precision [6].

For my final project, I explore the effect of quantizing a spiking neural network. I aim to answer the following questions:

1. What is the effect of the chosen bitwidth on accuracy, throughput, area, and energy use? Which bitwidth provides the 'best' trade-off in performance and accuracy?
2. How does the performance of the quantized SNN compare to a non-quantized SNN, in terms of accuracy, throughput, resource, and energy use?

2 Implementation

I started with an existing implementation of a Spiking Neural Network, implemented in PyTorch and trained on the RadioML.16 dataset [4] to perform

RF classification. This provides the baseline, non-quantized implementation. I then modified the SNN (using the Brevitas library [8]) to use fixed-sized integer weights of varying bitwidths.

Testing Methodology: There are two experimental components: (1) Testing in software to get a sense of the accuracy/performance trade-off of different bitwidths, as well as the performance gain achieved in the quantized vs non-quantized network; (2) Testing in hardware (on the PYNQ board) to get real numbers for throughput, resource usage, and energy use. Since testing in hardware requires performing a lengthy conversion from the PyTorch model to an HLS representation, software tests are performed first.

There are five metrics under consideration:

1. **Accuracy (Top-1 and Top-5)**: The fraction of times that the neural network makes the correct prediction on the first try (Top-1), or within the first five tries (Top-5). Measured in software and hardware.

2. **Inference Latency:** The time that it takes to perform inference. Measure in software and hardware.

3. **Area:** The resource usage of the network, including FF, LUT, BRAM, DSP. Measured in hardware.

4. **Estimated Power Use:** The estimated power consumption during inference, based on [1]. Estimated for the hardware implementation.

5. **Actual Power Use:** The actual power draw of the FPGA during inference, measured directly via a USB power monitor.

While the first two metrics can be measured in software, the rest require the neural network to be implemented on the PYNQ board.

3 Results

3.1 Testing in Software

Bit-width testing: The quantized neural network was first tested in software with different bitwidths, in order to determine which precision resulted in the best accuracy-performance trade off. The results of these experiments are summarized in Figure 1.

Observations: The 4-bit model is both less accurate and faster than the 2-bit model. I am not sure why this is the case. Other than this outlier, the general trend is as expected; models with more accurate weights produce more accurate, but slower, results. However, the variance in both accuracy and latency are still relatively low. The least accurate model (4-bit) is only 8% less accurate than the most accurate model (32-bit). The fastest model (4-bit) is only 18% faster than the slowest model (16-bit).

Although the choice of a 'best' model is application dependent, the 8-bit model provides the best accuracy-latency tradeoff; it is only 1.5% less accurate

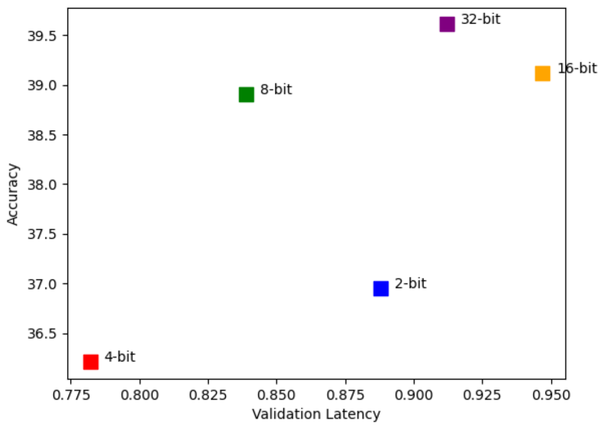

Figure 1: 8-bit weights seem to provide the best latency-accuracy trade-off

than the 32-bit model, but is 9% faster than the same. For this reason, I chose the 8-bit model for my hardware implementation.

Quantized vs Non-Quantized: The performance of the 8-bit model was then compared to the non-quantized model.

The accuracy of both models remained relatively constant across runs, and was extremely similar, with the quantized model being approximately 3% less accurate than the non-quantized model. This was as expected.

The results for inference latency were not quite as expected. The inference latency of the non-quantized model was actually significantly smaller than that of the quantized 8-bit model. However, the inference latency of the 8-bit quantized model was smaller than that of the 32-bit quantized model (as was also seen in Figure 1). This suggests that the better performance of the non-quantized model is likely due to implementation details not related to bitwidth. Furthermore, since these results were collected in software, they likely don't fully reflect how the models will perform in hardware.

Figure 2 summarizes these results as a histogram. The inference latency of all models is approximately Gaussian. The non-quantized model is approximately 2x faster than the 8-bit quantized model, which is approximately 30% faster than the 32-bit quantized model.

3.2 Testing in Hardware (Work in progress)

Two models are being implemented in hardware: (1) The non-quantized SNN, and (2) The quantized SNN with 8-bit weights, which provided the best trade-off in terms of accuracy and performance.

Current state of the hardware implementation: I am working through the conversion of the models from PyTorch to HLS via FINN HLS. I'm documenting

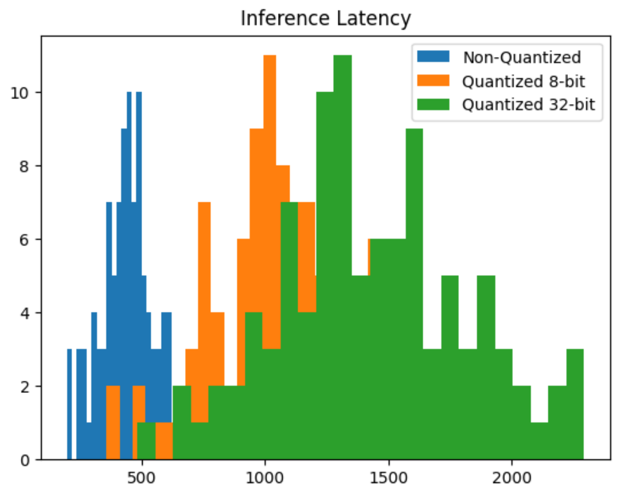

Figure 2: The quantized SNN performed inference faster than the non-quantized SNN

my progress on Google docs. Once this is complete, I will be able to report my results for inference latency, resource use, and energy use.

3.2.1 Resource Use

Waiting on hardware implementation.

3.2.2 Inference Latency

Waiting on hardware implementation.

3.2.3 Estimated Energy Use

Waiting on hardware implementation.

3.3 Energy Use

Methodology: To measure the actual energy usage of the FPGA, a USB power monitor was used. Three measurements were collected: (1) The baseline power consumption of the PYNQ board; (2) The power consumption of the PYNQ board during inference with the non-quantized SNN; (3) The power consumption of the PYNQ board during inference with the quantized SNN.

The experimental setup is shown in Figure 3. The PYNQ board is connected to a power source via USB. The power monitor sits between the PYNQ board and the power source, and reports the instantaneous voltage (V) and current (I) across the connection. The total power delivered to the board is then $P = VI$.

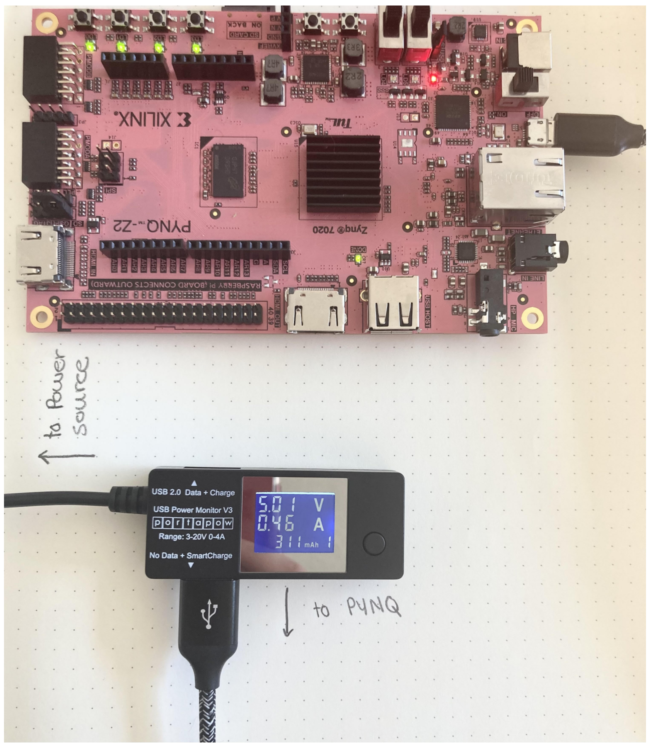

Figure 3: Experimental Setup for Power Consumption

Results: The baseline consumption of the PYNQ board is 2.375 Watts. Once I have the results from the hardware-implemented neural networks I will use this baseline value to calculate the additional power consumed during inference on the quantized and non-quantized networks. (Assuming that the difference is enough to detect with this device).

4 Conclusion

Based on testing in software, quantized SNNs have the potential to improve upon the performance of non-quantized SNNs, at the cost of a relatively small accuracy hit. Testing the model on hardware will confirm whether these trends hold true for a hardware implementation.

References

[1] Xilinx power estimator (xpe).

[2] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[3] Liam P Maguire, T Martin McGinnity, Brendan Glackin, Arfan Ghani, Ammar Belatreche, and Jim Harkin. Challenges for large-scale implementations of spiking neural networks on fpgas. *Neurocomputing*, 71(1-3):13–29, 2007.

[4] Timothy J O'Shea, Johnathan Corgan, and T Charles Clancy. Convolutional radio modulation recognition networks. In *International conference on engineering applications of neural networks*, pages 213–226. Springer, 2016.

[5] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, page 26–35, New York, NY, USA, 2016. Association for Computing Machinery.

[6] Nitin Rathi, Priyadarshini Panda, and Kaushik Roy. STDP Based Pruning of Connections and Weight Quantization in Spiking Neural Networks for Energy Efficient Recognition. *arXiv e-prints*, page arXiv:1710.04734, October 2017.

[7] Stefan Schliebs and Nikola Kasabov. Evolving spiking neural network—a survey. *Evolving Systems*, 4(2):87–98, 2013.

[8] Xilinx. Xilinx/brevitas.