ICAI 2023
12th International Conference on Applied Informatics
Eger, Hungary, 2023

Implementation of Quantized Neural Networks for the Sake of Verification

Dorina Hedvig Kiss, Gergely Kovasznai

Eszterházy Károly Catholic University
k.dorina33@gmail.com, kovasznai.gergely@uni-eszterhazy.hu

Abstract

Neural networks has undergone several changes and their complexity grew as well in the recent years. As a result, examining the vulnerability of such complex systems became an urgent problem. In the last few years several important papers were published in the topic of verifying the robustness of neural networks (e.g. [1], [2]), but the most of these was about so-called Binarized Neural Networks.

Our aim was to implement a Quantized Neural Network (QNN) which operates using only a limited number of values and taking this limit into consideration when training and verifying the QNN. “Some approaches describe the structure of a BNN in terms of sequential composition of blocks of layers rather than individual layers." [3] This statement is true in case of QNNs as well, since they also work with categorized values.

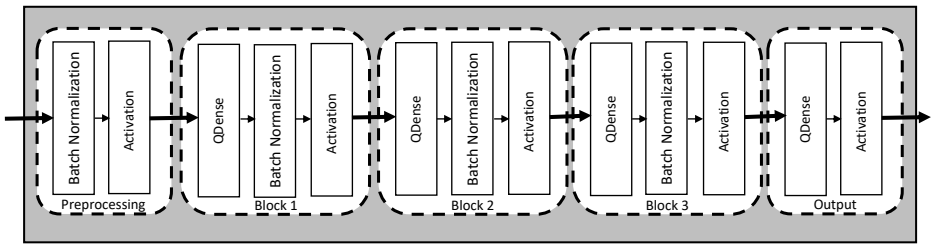

Figure 1. A schematic view of a QNN.

QNNs are also made up of blocks of layers as Figure 1 shows. The reason of the necessity of blocks is that the weights should be quantized before passing to the next linear transformation.

Quantization, or the categorization of values, can be calculated as follows:

$$Q(x, \mathsf{bw}) = \frac{Round(2^{\mathsf{bw}-1}x)}{2^{\mathsf{bw}-1}}, \tag{1}$$

where $\mathsf{bw} \in \mathbb{N}$ is the *quantization bit-width*. Using Equation (1), we implemented a function to categorize the given weights.

Code 1. Quantization function.

```
1 def round_quantize(x, bw):
2     q_pow = 2**(bw-1)
3     numerator = q_pow * x
4     numerator = numerator + K.stop_gradient(K.
          ↪ round(numerator) - numerator)
5     return numerator / q_pow
```

The base for our QNN implementation was the *Dense* class of the *Keras* framework. The method *call* of our *QDense* class uses the function in Code 1 to pass quantized values to the next layer. Therefore, the extracted weights of such layers must be quantized before verification.

In our experiments, the implemented QNN was trained for 10 epochs, using the MNIST dataset, and the test accuracy was approximately 91%, as it can be seen in Figure 2.

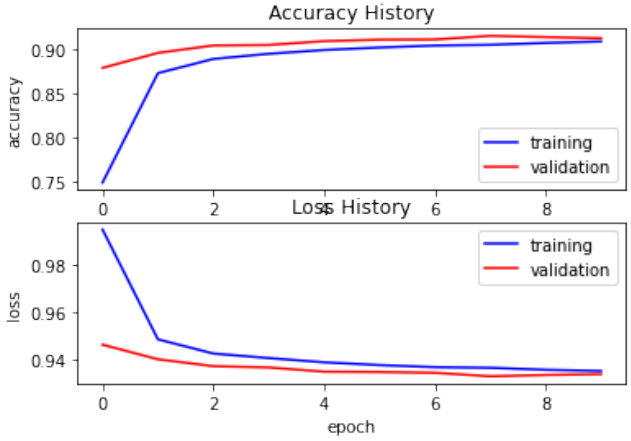

Figure 2. Results of training the QNN using MNIST dataset.

Then the neural network was encoded as a set of Boolean Cardinality Constraints (BCC), as being proposed in [3], which then can be fed into a constraint solver for verifying certain properties over the neural network. The BCC encoding

of QNN blocks are implemented in Python inside our tool VerBiNe, that is able to execute different kind of solvers (SAT, SMT, ILP), including cardinality solvers, via Python packages. For our experiments, we used the cardinality solver MiniCARD.

The use case for our experiments was to verify the adversarial robustness of the trained QNN, meaning that it might have misclassified inputs if we allowed to add perturbation in the range $[-\epsilon, \epsilon]$ to individual input values. For this, we randomly picked 20 images that were correctly classified by the network and then we experimented with three different maximum perturbation values by varying $\epsilon \in \{1, 3, 5\}$. The result can be seen in Figure 3.

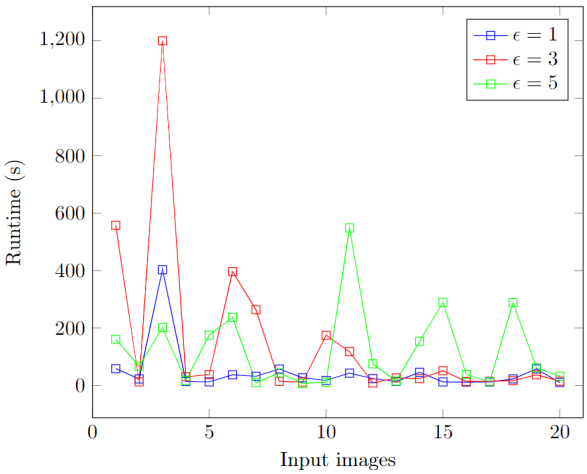

Figure 3. Runtimes of VerBiNe when running MiniCARD on MNIST images with maximum perturbation $\epsilon \in \{1, 3, 5\}$.

To sum up, the implemented QNN works as expected when training over the MNIST dataset. It provides decent accuracy and still operates with quantized values as we wanted. However, our experiments with the CIFAR-10 data were not that promising. Our QNN was able to provide much less accuracy, even when it was trained for more than 100 epochs. As future plan, we are planning to do further investigations in this direction.

References

[1] K. Jia, M. Rinard: *Efficient Exact Verification of Binarized Neural Networks*, Advances in Neural Information Processing Systems (2020), pp. 1782–1795.

[2] G. Kovásznai, K. Gajdár, N. Narodytska: *Portfolio Solver for Verifying Binarized Neural Networks*, Annales Mathematicae et Informaticae 53 (2021), pp. 185–202, DOI: https://doi.org/10.33039/ami.2021.03.007.

[3] G. Kovásznai, D. H. Kiss, P. Mlinkó: *Formal Verification for Quantized Neural Networks*, Annales Mathematicae et Informaticae (2022), under revision.