Hardware acceleration for ultra-fast Neural Network training on FPGA for MRF map reconstruction

Mattia Ricchi[1,2,*]

[1]Department of Computer Science, University of Pisa, Largo Bruno Pontecorvo 3, 56127, Pisa, Italy
[2]National Insitute of Nuclear Physics, Division of Bologna, Viale Carlo Berti Pichat 6/2, 40127, Bologna, Italy

Abstract

Magnetic Resonance Fingerprinting (MRF) is a fast quantitative MR Imaging technique that provides multiparametric maps with a single acquisition. Neural networks (NNs) accelerate reconstruction but require significant resources for training. We propose an FPGA-based NN for real-time brain parameter reconstruction from MRF data. Training the NN takes an estimated 200 seconds, significantly faster than standard CPU-based training, which can be up to 250 times slower. This method could enable real-time brain analysis on mobile devices, revolutionising clinical decision-making and telemedicine.

Keywords

magnetic resonance fingerprinting, neural network, hardware acceleration, FPGA, real-time.

1. Introduction

Magnetic resonance imaging (MRI) improves healthcare through better diagnosis and treatment, supported by artificial intelligence (AI) in data analysis [1, 2]. A key AI application in MRI is the reconstruction of quantitative maps acquired with magnetic resonance fingerprinting (MRF), a significant technique that can produce multiparametric maps in a single scan [3]. Recent work by Barbieri et al. [4, 5] on using neural networks (NN) for the reconstruction of the MRF parameter map reconstruction has shown greater or equal efficiency compared to traditional methods, with better memory efficiency and reduced computational load. However, NN training is resource-intensive and time-consuming. FPGA acceleration for NN offers a potential solution [6], as it exhibits high throughput and low latency. After a traditional software validation, the NN algorithm can be transformed into a hardware version compatible with modern FPGAs, aimed at accelerating processing times by a factor of a few times to hundreds.

The studies by Xiong et al. [7] and Sanaullah et al. [8] highlight the potential of FPGA for accelerating NNs used in medical imaging and diagnosis. Xiong et al. [7] developed an FPGA-accelerated NN for brain tumour segmentation in MRI images, resulting in significant improvements in speed and energy efficiency compared to traditional platforms. Sanaullah et al. [8] created an FPGA-based processor that outperforms CPU and GPU implementations, achieving notable speedups. Despite these advances, NN training is still performed using software before deployment on FPGAs for inference, due to the iterative and offline nature of the training process [7].

However, MRF has a significant disadvantage: it is a non-standardised MR technique. Various factors, such as the scanner manufacturer, the magnetic field strength, and the number of parameters to be retrieved, can affect how the data are acquired and, consequently, how the quantitative maps are reconstructed. Therefore, whenever any single parameter is changed, the NN must be retrained to adapt to the new case. The retraining procedure is both computationally demanding and time-consuming, creating a substantial obstacle to the effective application and standardization of MRF techniques across

Discovery Science - Late Breaking Contributions 2024
*Corresponding author.
mattia.ricchi@phd.unipi.it (M. Ricchi)
0009-0008-4430-3843 (M. Ricchi)
© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

CEUR-WS.org/Vol-3928/paper_183.pdf

various platforms and clinical environments.

The purpose of this work involves hardware programming for an FPGA-accelerated NN training algorithm for the reconstruction of MR parameters (T_1 and T_2) using clinical MRF. To test the ability to accelerate the training process on FPGA, the original NN must first be redesigned, i.e., simplified and quantized, to meet the available resources of the hardware accelerator. This would result in an important reduction in training time and power consumption.

2. Matherials and Methods

The NN model by Barbieri et al. [4, 5] is a feedforward network with nine fully connected layers. It uses ReLU activations for the first eight layers and a linear activation for the output layer. The model inputs are the real and imaginary parts of MRI signals and outputs T_1 and T_2 quantitative maps. Training was supervised using the Mean Squared Error (MSE) loss function, over 500 epochs with 1000 gradient steps each, a learning rate of 10^{-4}, optimized with Adam Optimiser [9], implemented with Keras TensorFlow [10], taking around 16 hours on an AMD Ryzen 9 3900 CPU. To fit FPGA resources, the first two layers were removed and the network was retrained on the original dataset of 250M MRF simulated signals. Performance was evaluated on 5000 new synthetic signals. Quantization Aware Training (QAT) [11] was applied to use lower precision (integer parameters) without degrading performance.

A low-level HDL design approach, in which every firmware component is written in VHDL, without any high-level synthesis support, has been selected, ensuring full control and data protection through an on-FPGA firewall security algorithm [12]. The ALVEO U250 FPGA board, with 1.7M LUTs, 3.4M FFs, 12k DSPs, and 2.6k BRAMs, was selected for implementation. Firstly, the behaviour function of a single node was implemented as given in Eq. (1).

$$y = \sigma\left(\sum_{i=1}^{N_{inputs}} x_i \cdot w_i + b\right) \tag{1}$$

This function was generically implemented once and then used all the necessary times to cover all the node operations present in the NN. Proper functioning was verified by deploying 16 nodes in the FPGA and comparing their output with those of Python. Secondly, the backpropagation algorithm was implemented. As a starting point, the simple stochastic gradient descent was chosen, which describes how the parameters of the NN, i.e. weights and biases, are updated at each iteration during training, following Eq. (2).

$$\begin{aligned} \delta^l &= \left(\left(\mathbf{w}^{l+1}\right)^T \delta^{l+1}\right) \circ \sigma'(z^l) \\ \frac{\partial \mathscr{L}}{\partial \mathbf{w}^l} &= y^{l-1}\delta^l \quad \text{and} \quad \frac{\partial \mathscr{L}}{\partial b^l} = \delta^l \end{aligned} \tag{2}$$

Finally, an assessment was conducted to evaluate the resource requirements for node operations, backpropagation, and memory storage on the FPGA, involving the necessary LUTs, DSPs, and FFs.

3. Results and Discussion

Tab. 1 reports the error metrics both in the cases of the original and the quantized NN, showing that the quantization process did not affect the NN performance but simply reduced its precision, allowing it to work with full integer numbers.

The synthesis tool provided by Vivado enables the transformation of high-level design descriptions into gate-level models, aiding in the development of efficient and optimised digital circuits. By analysing synthesis results, one can estimate the maximum achievable clock frequency for the implemented design. In our scenario, based on the synthesis outcomes of a single node and the backpropagation

	T_1		T_2	
	Original	Quantized	Original	Quantized
MAPE (%)	2.15	2.36	8.89	11.07
MPE (%)	−0.66	0.12	0.02	−3.12
RMSE (ms)	75	78	145	148

Table 1
Comparison between the error metrics in the case of the original and quantized NN. Results show good performance of the quantized NN.

process, a clock frequency of 200 MHz is totally feasible, with the possibility of increasing it to 250 MHz. Based on the estimation of the required resources for executing all the necessary operations on the FPGA, the whole network and backpropagation algorithm cannot be implemented on the FPGA but, it is feasible to implement 16 nodes of the second layer on the FPGA, as well as the backpropagation between the layers containing 16 and 32 nodes. Thus, by iterating these two blocks multiple times in a semiparallelised way, all the operational requirements of the network can be covered.

To verify the correct implementation of the single node function in VHDL, the outputs produced on both the FPGA and in software were compared after providing identical inputs, weights, and biases. The comparison produced promising results, as there was no difference between the Python outputs and those of the FPGA, indicating that the mathematical operations were correctly translated into VHDL.

The estimate of the necessary FPGA resources was 145k LUTs, 5k DSPs, and 146k FFs. This implies that the entire NN and backpropagation use 8% of the available LUTs and 40% of the available DSPs, demonstrating that the algorithm's implementation is entirely viable from the resource point of view. PCI Express technology was chosen to communicate from the PC's CPU to the FPGA and back resulting in additional resources of 83k LUTs, 148kn FFs and 150 BRAMs, the internal RAM memories of the FPGA.

Finally, a fairly accurate estimate of the training time can be made. Each node needs 4 clock cycles to perform its operations. The 16 nodes implemented on the FPGA work in a semi-parallel way, resulting in 56 clock cycles required for all levels. Similarly, the single backpropagation module requires 3 clock cycles, iterating through the entire process for a total of 104 clock cycles. With a clock frequency of 200 MHz, the clock period is 5 ns, considering 250M training data the total training time results in:

$$(5 \cdot (250'000'000 \cdot (56 + 104))) = 200\,\text{s} \tag{3}$$

This result highlights how the NN can be trained on FPGA in less than 5 minutes, which is 200 times faster than the corresponding training on CPU. The proposed method poses a big step in the direction of real-time and personalized healthcare, opening the possibility of having an integrated NN hardware accelerator for map reconstruction inside the MRI scanner.

Acknowledgments

The author would like to thank all the people who contributed to this work: Fabrizio Alfonsi (INFN Bologna), Camilla Marella (University of Bologna), Marco Barbieri (Stanford University), Alessandra Retico (INFN Pisa), Leonardo Brizi (University of Bologna), Alessandro Gabrielli (University & INFN Bologna), Claudia Testa (University & INFN Bologna).

References

[1] J. C. Gore, Artificial intelligence in medical imaging, Magnetic Resonance Imaging 68 (2020) A1–A4. URL: https://www.sciencedirect.com/science/article/pii/S0730725X19307556. doi:https://doi.org/10.1016/j.mri.2019.12.006.

[2] D. Shen, G. Wu, H.-I. Suk, Deep learning in medical image analysis, Annual Review of Biomedical Engineering 19 (2017) 221–248. URL: https://www.annualreviews.org/content/journals/10.1146/annurev-bioeng-071516-044442. doi:https://doi.org/10.1146/annurev-bioeng-071516-044442.

[3] D. Ma, V. Gulani, N. Seiberlich, K. Liu, J. Sunshine, J. Duerk, M. Griswold, Magnetic resonance fingerprinting, Nature 495 (2013) 187–92. doi:10.1038/nature11971.

[4] M. Barbieri, L. Brizi, E. Giampieri, F. Solera, G. Castellani, C. Testa, D. Remondini, Circumventing the curse of dimensionality in magnetic resonance fingerprinting through a deep learning approach, 2018.

[5] M. Barbieri, L. Brizi, E. Giampieri, F. Solera, D. Manners, G. Castellani, C. Testa, D. Remondini, A deep learning approach for magnetic resonance fingerprinting: Scaling capabilities and good training practices investigated by simulations, Physica Medica 89 (2021) 80–92. doi:10.1016/j.ejmp.2021.07.013.

[6] A. Sanaullah, C. Yang, Y. Alexeev, K. Yoshii, M. Herbordt, Real-time data analysis for medical diagnosis using fpga-accelerated neural networks, BMC Bioinformatics 19 (2018). doi:10.1186/s12859-018-2505-7.

[7] S. Xiong, G. Wu, X. Fan, X. Feng, Z. Huang, W. Cao, X. Zhou, S. Ding, J. Yu, L. Wang, Z. Shi, Mri-based brain tumor segmentation using fpga-accelerated neural network, BMC bioinformatics 22 (2021) 421. doi:10.1186/s12859-021-04347-6.

[8] A. Sanaullah, C. Yang, Y. Alexeev, K. Yoshii, M. Herbordt, Real-time data analysis for medical diagnosis using fpga-accelerated neural networks, BMC Bioinformatics 19 (2018). doi:10.1186/s12859-018-2505-7.

[9] D. Kingma, J. Ba, Adam: A method for stochastic optimization, International Conference on Learning Representations (2014).

[10] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, X. Zheng, Tensorflow : Large-scale machine learning on heterogeneous distributed systems, 2015.

[11] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko, Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017. URL: https://arxiv.org/abs/1712.05877. arXiv:1712.05877.

[12] M. Grossi, F. Alfonsi, M. Prandini, A. Gabrielli, A high throughput intrusion detection system (ids) to enhance the security of data transmission among research centers, Journal of Instrumentation 18 (2023) C12017. URL: https://dx.doi.org/10.1088/1748-0221/18/12/C12017. doi:10.1088/1748-0221/18/12/C12017.