Implementation of convolutional NEURAL network in FPGA for image recognition

Victor Mendonça Aguirre
Fadi Jerji

CITE THIS ARTICLE
Aguirre, Victor Mendonça; Jerji, Fadi; 2022. Implementation of convolutional NEURAL network in FPGA for image recognition. SET INTERNATIONAL JOURNAL OF BROADCAST ENGINEERING. ISSN Print: 2446-9246 ISSN Online: 2446-9432. doi: 10.18580/setijbe.2022.4. Web Link: http://dx.doi.org/10.18580/setijbe.2022.4

Implementation of convolutional NEURAL network in FPGA for image recognition

1st Victor Mendonça Aguirre
Mackenzie Presbyterian University
São Paulo, Brazil
victor.1605@hotmail.com

2nd Fadi Jerji
Mackenzie Presbyterian University
São Paulo, Brazil
fadi.jerji@gmail.com

***Abstract*—Neural Networks (NN) are being researched and improved to a degree that machines can closely resemble the capacity to execute complex tasks that only an intelligent animal is capable of. Vision used in the interpretation and recognition of the environment is one of such tasks that is being researched so future technologies can simulate vision in autonomous vehicles to further improve self-driving capabilities, increase driver convenience, help avoid accidents, and even autonomous delivery. Convolutional neural networks, inspired by the mechanics of animal vision, are utilized for the complex task of image recognition. Field Programmable Gate-Arrays (FPGA) recent developments have given it more parallel processing and processing speed making it a prime candidate for the implementation of NNs efficiently, with more processing capabilities and low response times compared with the alternatives. The objective of this work is to evaluate the performance viability of FPGA implementation of an image classification NN with acceptable accuracy and low response time.**

***Index Terms*—FPGA; CNN; autonomous vehicles; MNIST.**

I. Introduction

Artificial intelligence has attracted attention from the most varied types of industries, with several studies being carried out on the subject and with recent advances in hardware, increasingly complex algorithms are being developed with adequate processing time, making it possible to derive useful and fast information from large amounts of information, but the question of efficient hardware implementation remains to execute these algorithms quickly and efficiently.

Along with the development of techniques and studies, the applications of convolutional neural networks (CNNs) have grown considerably, mainly in activities that require understanding at a level comparable to that of a human being, such as natural language processing and computer vision and it is possible to incorporate CNNs to assist in processing audio, image classification, scenario labeling, and facial recognition [1], [2]. Some networks achieve better results than human performance as evidenced in the work of [3]. The impressive performance of these networks comes at the cost of large memory bandwidth and intensive use of computational logical resources [4].

CNNs have excellent performance when it comes to image classification, but such networks require millions or even billions of operations per second to classify an image, which makes network implementation a challenge in terms of computational power and memory storage capacity. For example, in 2012 Alexnet [5], with a network architecture that required the storage of 60 million parameters to process an image, won the Imagenet contest [6]. In 2014 the VGGNET network [7] wins the same contest, but its design required loading about seven times more parameters, because of this, the network required dedicated hardware to run.

CNNs have enabled the increasing automation of tasks and machines such as autonomous cars, the idea of product delivery via drone is already being discussed [8], but in the same way, as autonomous cars require a large amount of information and an equally large processing power together with a robust algorithm to detect and recognize obstacles [9], drones or autonomous aircraft require an even faster processing and image recognition capacity. To be able to follow the movements and maneuvers performed by these vehicles. The choice of hardware for implementing CNNs is important because it will influence the needs and results of the network, for training the choice mostly adopted is a graphics processing unit (GPU), due to its great capacity for parallelism of calculations, reaching 11 trillion floating point operations per second (TFLOP/s) and due to the need to train the neural network only once, the GPU's energy consumption does not significantly impact the process, as for the implementation of the network in themselves, they can be implemented either on GPU, Field Programmable Gate-Arrays (FPGA) or Central Processing Unit (CPU), being more flexible FPGAs when compared to ASICs, counting on easy and fast implementation in the market and upgradeability even after implementation compared to CPU and it is also worth mentioning its potential for improving the architecture, energy savings compared to GPU and the possibility of using different formats and numerical representations. Microsoft has recently explored the possibility of CNNs on FPGAs as cost-effective network accelerators in a data center [10], [11]. Many studies are being carried out on accelerators for CNNs implemented in FPGAs [12], [13], as well as tools to generate such accelerators automatically [14], [15]. Studies have also been carried out on

CNNs networks with low accuracy, networks using weights and activation function with numbers in binary format [6], [16] and in some cases, the network has an accuracy comparable to networks using 32-bit floating point, these types of implementation are attractive in FPGAs because they take advantage of the efficiency of operations performed on Look Up Tables (LUTs). The implementation in FPGAs has seen more and more attention due to the constant development of tools that help and automate the development of implementations on the board, the Xilinx Vivado tool for high-level synthesis (HLS) allows the user to write code with a reasonable level of abstraction and the tool's algorithm compiles the code for register transfer level (RTL) between registers [17].

For complex tasks that require a large number of calculations, but which, at the same time, demand efficiency and low response time, CNN in FPGA focused on image recognition would be a tool that would drive the development and implementation of such technologies to advance sectors that would benefit from autonomous aerial vehicles and make the autonomous car industry even more robust.

II. NEURAL NETWORKS

A fundamental component of neural networks, in general, are artificial neurons, inspired by biological neurons, which are responsible for most of the processing that occurs in artificial neural networks (ANNs) and can be arranged within a network in various ways.

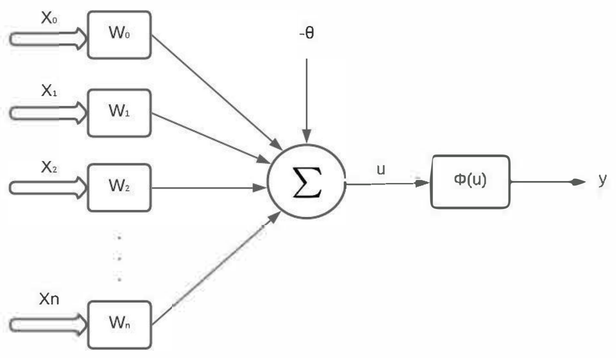

Fig. 1: Perceptron.

It's possible to observe the general structure of a neuron or perceptron in Fig. 1 where from X_0 to X_n representing input data or signals from neurons from another layer, the synaptic weights W_0 to W_n that determine how excitatory or inhibitory the signal is for the neuron. The adder block is responsible for summing the modified input signals with a predetermined value θ called bias, its function is to increase or decrease the net input, in order to translate the activation function on the axis, it can also be used so that, in the network training process, changes in synaptic weights result in less drastic changes in the network as a whole since the bias is independent of the input value in the system, which helps the network to converge on an ideal solution and can also be used to make the value needed to activate the activation function larger or smaller. NNs generally have a forward propagation architecture where signals entering the system propagate towards the output in a single direction. The model represents artificial neurons and can be represented by the following equations 1 and 2

$$u = (\sum_{i=1}^{n} X_n \times W_n) - \theta \quad (1)$$

$$y = \Phi(u) \quad (2)$$

The weight parameters W_n and bias θ are adjusted in the training of neurons in the network according to the final application.

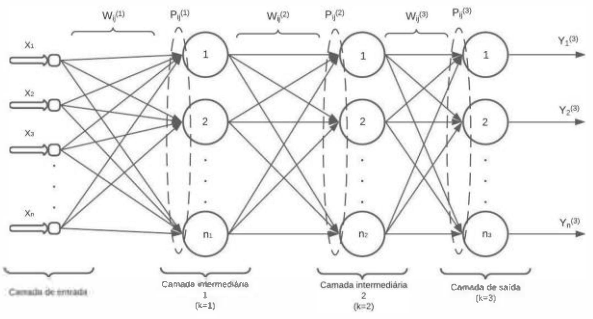

Fig. 2: MLP network.

An ANN can have a different number of layers which can be classified into three categories:

- Input layer: responsible for receiving data, signals, or characteristics from the external environment that are usually normalized in relation to the ranges of dynamic variations produced by the activation functions, which improves the accuracy of the network as a whole.
- Hidden layers: composed of neurons with the function of extracting characteristics associated with the system to be inferred.
- Output layer: a layer of neurons responsible for presenting the final results of the network, from the signals received by the layers that precede it.

The Multiple Layer Perceptron (MLP) network contains at least one intermediate layer, in contrast to the single-layer perceptron network, the MLP has one or more hidden or hidden layers between the input and output layers. MLP networks are more complex in their structure, which allows them to perform more complex work compared to the networks mentioned above and can solve problems that would go beyond binary classification. MLP networks have feed-forward regardless of the number of layers, the first layer captures the signals to be processed, then the intermediate layers extract information about the signals, process and encode through their respective synaptic weights, bias, and activation function and the output layer receives the resulting stimuli from the intermediate layers and produces the network response. Note that in an MLP network it is possible to have multiple neurons in the output layer, resulting in the network having multiple output possibilities.

The adjustment of synaptic weights of the MLP network takes place through the backpropagation process, which consists of an algorithm that calculates the gradient of the error function, starting at the output layer and propagating towards the input layer, partially reusing the calculations of the gradient of the previous layer to carry out the weight adjustments of the next layer [18]. The specific configuration of an MLP network must be determined from a series of factors such as the class of problem to be treated by the network, arrangement of training samples, initial values, and attributes so that the network can be implemented efficiently.

A. Convolutional Neural Networks

CNNs commonly used for pattern recognition in images have an MLP network architecture, but they stand out for the presence of convolutional layers and often, pooling layers observed in Fig. 3 in addition to the concepts already present in MLP networks.

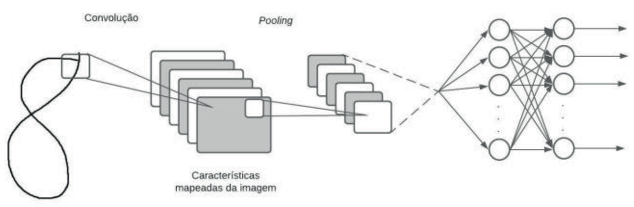

Fig. 3: Convolutional network representation.

Convolutional layers have the function of convoluting a weight matrix sectioned across an image to extract relevant information so that subsequent layers can make use of the extracted information to recognize patterns in specific parts of the image. Thus, a convolutional layer can identify intrinsic characteristics of different parts of the image such as horizontal and vertical lines, and specific angles, among other patterns to be processed by the next layers [19], [20]. Pooling layers accompany the convolution layers and their function is to reduce the dimensions of the data provided by the convolution layer, connecting the output of a group of neurons from the previous layer into a single neuron from the pooling layer. The pooling layer can have different aspects, neurons can extract the maximum value or they can average the values received from the group of the previous layer [19], [20].

III. FPGA

The FPGA has a structure with three main components, LUT is responsible for implementing the logic functions, the input and output blocks allow communication with peripherals and the interconnectors that carry out the communication between the blocks and some other components with more specific functions such as the blocks random access memory (BRAM) and digital signal processing blocks (DSP). All the blocks are configurable so that the user when programming the desired logic into the FPGA, the circuit forms the design structure using the blocks which are essentially a vector or array of combinational logic. Along with LUTs, other resources such as D-flipflops, multiplexers, and transport logic, carry, among others, to implement more complex functions such as boolean functions and multipliers, then LUTs are now called configurable logic blocks (CLB) illustrated in Fig. 4.

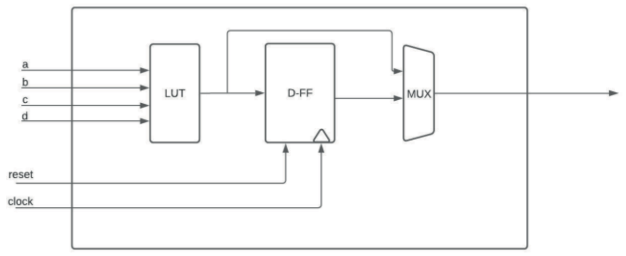

Fig. 4: CLB representation.

Fig. 5 represents a LUT with three inputs (A, B, and C), the possible values of the LUT are stored in a register, because of this, the LUT can be implemented as any function that has the same number of inputs. Once configured the output values are selected according to the inputs. Modern FPGAs have 6-input LUTs and 64-bit registers.

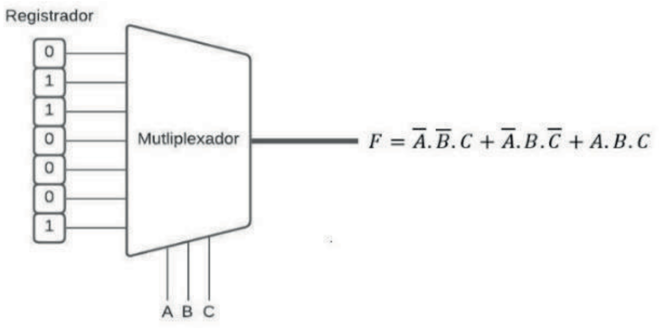
Registrador
0
1
1
0
0
0
1
Mutliplexador
$F = \overline{A}.\overline{B}.C + \overline{A}.B.\overline{C} + A.B.C$
A B C

Fig. 5: 3 input LUT representation.

By itself, a single LUT or CLB is very simple and incapable of performing complex logic functions but connected in large amounts are capable of performing complex functions even if the individual power of each block is limited, there are also FPGAs that have a carry chain which connects the LUTs of CLBs with the LUTs of neighboring CLBs allowing the creation of arithmetic functions as adders, with low-level logic efficiently and quickly.

As the priority of FPGA circuits is efficiency in the use of resources present on the board, the interconnection of the most recent FPGAs has logic circuits to assist in the interconnection and routing of the other blocks as seen in Fig. 6, such as connection blocks (CB) that are responsible for connecting the logic blocks with the interconnection rails with the possibility of using any of the rails to assist in routing, and also the switching blocks, which are configurable blocks that connect the rails themselves to provide more routing possibilities at the time of implementation [21].

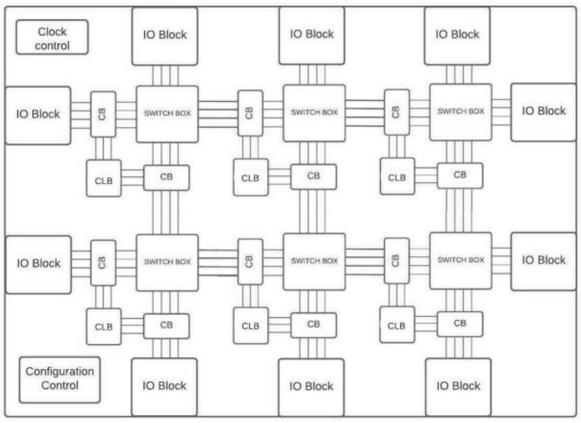

Fig. 6: FPGA structure representation.

BRAMs are memories that allow fast access to data and have the ability to read and write data and are much more efficient for data storage than LUT memory implementations, it is also possible to access two memories in the same block simultaneously, which allows the data preloading or data writing and reading at the same instant. BRAMs can be inferred by the synthesis tool or can be instantiated by the user during design elaboration.

DSP blocks are blocks that allow multiplication operations followed by result accumulation without the use of LUTs, DSPs have a series of configurable functional blocks, the DSP48E1 block present in FPGA Xilinx Series 7 features a 25-by-18-bit two's complement multiplier, a 48-bit accumulator, a pre-adder, a block that can perform the addition, subtraction or accumulating result of multiple data simultaneously, a unit of logical operation with bits like AND, OR, NOT, NAND, NOR, XOR, XNOR, overflow and underflow detectors and configurable pipeline [17].

IV. METODOLOGY

For the implementation of a neural network in an FPGA, it was necessary to generate and train the network externally, for this, R and Python programming languages and the Keras package were used for the elaboration of the neural network and the export of the weights and bias of the trained network with MNIST dataset [22], tests were performed with different network configurations, such as number of layers, number of neurons and activation function to test the implementation in FPGA.

A. Network Weights Preparation

Once the network was trained, the values of the weights and bias of each neuron were exported and manipulated using Excel, with the use of formulas elaborated in the spreadsheets, it was possible to easily process the data for later implementation in the FPGA. For an integer implementation, the weight and bias values were multiplied by a multiple of 10, depending on how accurate the decimal places would be, for example, for three decimal places the values were multiplied by 1000, then rounded using the formula "=round('Nº','Nº of decimal places')". The result was a number with no decimal places, for example, for a weight value of 0.6457, the value implemented in FPGA would be 646.

For training and testing the elaborate network, the MNIST dataset was used, which consists of handmade images of numbers from 0 to 9, widely used for training and testing image processing systems, containing 60,000 images for training and 10,000 for testing, all images are in grayscale with dimensions of 28x28 pixels, so the network input will be 784, a value for each pixel [22].

B. FPGA Implementation

Tests were carried out with different types of networks in the implementation, varying the number of layers, the number of neurons in each layer, except for the output layer, the type of activation function, the precision of decimal places, and the differences between implementation with number integer and with the library for numerical representation with fixed point. For the evaluation of the implementation results, the following parameters were observed: the response time, the network accuracy, the use of board resources, and the energy used for its operation.

1) Block structure: For the implementation in FPGA, using VHDL programming language, different blocks were elaborated that together compose the neural network elaborated previously.

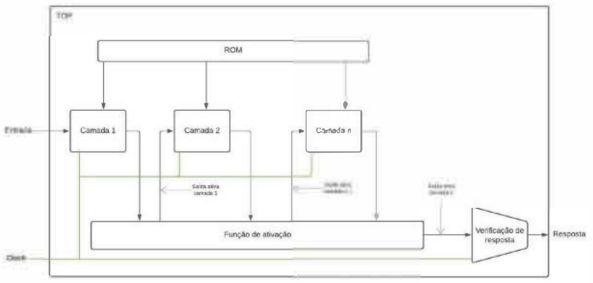

Fig. 7: Representation of the network structure implemented with data storage in LUT.

As illustrated in Fig. 7, the network has six basic blocks in its structure, the TOP block that receives the external data, in the case of this work, receives the values of the pixels of the image to be identified and provides the final response of the network according to the active output of the last layer, the neuron block, illustrated in Fig. 8, is responsible for weighting the inputs by synaptic weights and finally adding the bias, the neurons layer block is responsible for receiving the external input or data from the active output of the previous

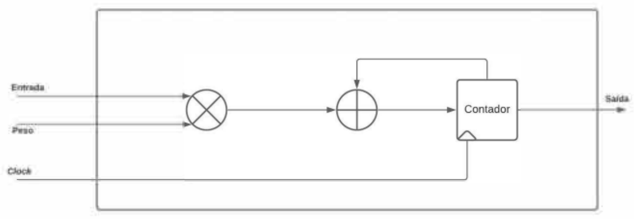

Fig. 8: Single neuron representation.

layer and distribute it to all neurons contained in it, after the completion of the calculations performed by the neurons, the layer sends the accumulated output to the block containing the activation function stored in LUTs or in BRAM, after activating the outputs of the last layer, the data is sent to the block responsible for comparing the activated outputs of the last one and verifying the value most likely to be the correct answer.

The VHDL code was designed in such a way that it is possible to modify the number of neurons and layers with minimal configuration of the code itself, which facilitates testing and adapting the code to different needs such as greater accuracy, fewer used resources, faster response time, etc. after any necessary modifications to the code, the Xilinx Vivado HLS tool is used to synthesize the code for RTL and implement it on the FPGA board.

2) Numerical Representation: As it is not possible to synthesize real numbers in VHDL, tests were carried out with two types of numerical representation. The first form of representation used as integers, with the arithmetic operations already implemented in VHDL through the numeric_std library, to represent decimal places the input and weight values were multiplied by multiples of ten, depending on the determined precision. For example, for the representation of two decimal places the values were multiplied by one hundred, and the remainder was rounded, the values were then transferred to the FPGA.

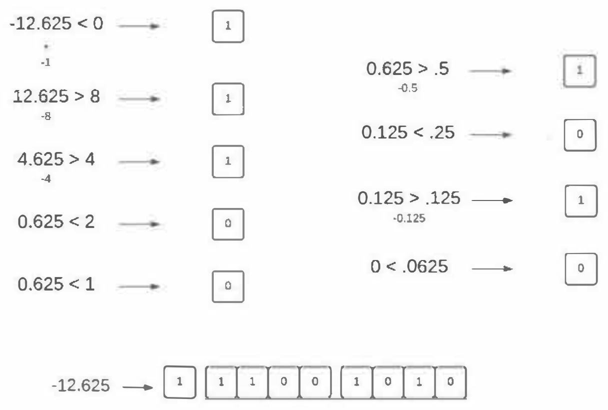

Fig. 9: Fixed point conversion.

The other form of numerical representation used for tests was binary with fixed point, for that, a package was elaborated and implemented that includes conversion from real number to binary with fixed point, addition and multiplication so that it was possible to implement the network using such numerical representation. with ten bits for the decimal part, 14 bits for the integer part, and one bit for the sign. The conversion was done by an algorithm elaborated in Octave and is done in software before the implementation, it consists of comparisons, between the number you want to convert and each of the numbers that are represented by the bits at a fixed point, as illustrated in Fig. 9.

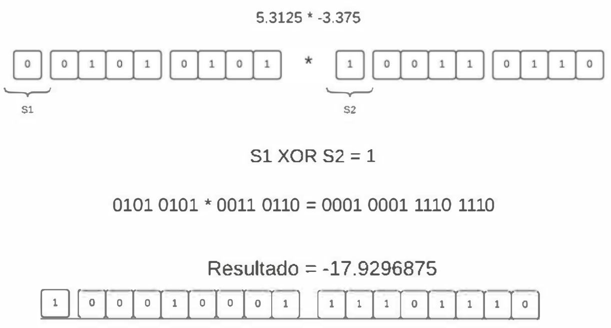

Fig. 10: Fixed point multiplication.

The implemented fixed-point number multiplication is represented in Fig. 10, the operation is relatively simple, performing a standard binary number multiplication and then dividing between the vector that represents the integer number and the vector that represents the decimal. In the example, it can be seen that for multiplication between two numbers with four bits in the decimal part, the result will have the eight least significant bits as the decimal part.

3) Activation Functions: Tests were performed with different activation functions to evaluate results, mainly observing the accuracy and resource requirements of the FPGA used to implement the activation function.

For better response times and simplicity of implementation, the activation functions were calculated in software with a programming language with a high level of Octave abstraction, used primarily for mathematical computation, the values were tabulated and stored in BRAM on the board.

Due to optimizations made by the Xilinx Vivado HLS tool during synthesis and implementation, the algorithm can conclude that the activation function can be implemented in a combination of LUTs, multiplexers, and registers for better use of the final design area.

The tests highlighted that implementing the activation function more efficiently is through LUTs elaborated outside the FPGA and stored in RAM for quick access, saving DSP blocks and processing time due to calculations of the implemented model. The test implementation of the activation function determined that the calculations necessary for the sigmoid activation function would be performed in at least four clock cycles for each value, however with the values in LUTs or RAM only one cycle is needed for each value, decreasing response time without sacrificing accuracy, it is also possible to utilize more resources on the board to calculate the response of the activation function of all output values of a layer at the same time.

4) Weight Storage: Two network models were developed in terms of the way of storing the weights of the network, one of the ways was implementation directly in LUTs of the FPGA, once implemented the weights were stored as functions in LUTs, registers, and multiplexers while the other way is initial storage external value of the weights that are then stored in the FPGA's BRAM and during the calculations, the BRAM supplies the weights to the neurons.

The process of storing the values in RAM requires instantiating the blocks according to the number of neurons so that there is one block per neuron, so there will be no difference in the network response time, the process requires an initialization period for the data to be provided by an external source to RAM, however, it is only necessary to perform the process once.

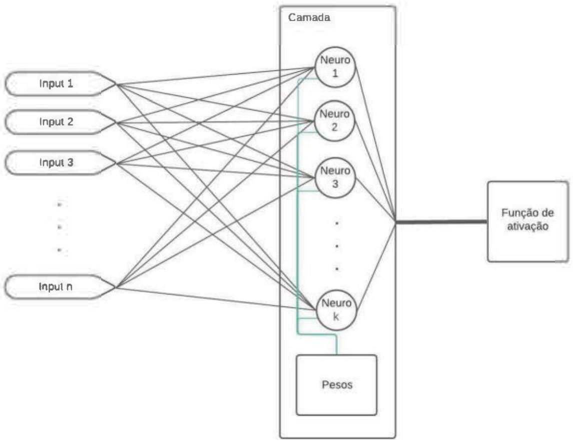

Fig. 11: Representation of a layer with data storage in BRAM.

Fig. 11 illustrates a layer when the method of implementing weights in BRAM is used, the layer block distributes the weights stored in RAM to the respective neurons according to the input, so the layer block has a vector of BRAMs, and each BRAM stores the weight value of each neuron in vector form, so that during the weighting calculation the neuron receives the weight corresponding to the input for each clock period.

5) FPGA Board Used in Tests: To perform the tests, the FPGA Basys 3 Artix-7 board was used, which has 33,280 logic cells in 5200 slices, each containing four LUTs of six inputs and eight flip-flops, 50 RAM blocks of 1,800 Kbits each, 90 DSP48E1 slices. For the implementation, 100MHz was used as clock time [17].

V. RESULTS

A. Parallelism

The initial implementation had total parallelism, with a completely asynchronous network it was possible to perform all the necessary calculations in parallel and obtain the result in the picoseconds scale, however for a small network of ten neurons in the hidden layer and ten in the layer of output were used around 103800 LUTs and more than 10000 DSP slices making the implementation unfeasible for the board available for testing.

Based on this experience, it was decided to perform the calculations synchronously, but many of the calculations are still performed in parallel. The network was then modified to perform the multiplication and accumulation calculations for each neuron synchronously, so that each multiplication and addition calculation for a neuron takes three clock cycles at 100MHz to perform, conserving board resources in exchange for a longer response time, but this allows all neurons to perform calculations in parallel so that the number of neurons in the hidden layer minimally changes the total response time of the layer, but increases the number of input values of the next layer which increases the response time on the order of approximately 20ns per neuron. With the reduction of parallelism, it was possible to reduce the resources used by the implemented network considerably.

B. Decimal Place Precision and Numeric Representation

The variation of the precision of decimal places allows both savings in the use of resources on the board with the use of less precision, while greater precision guarantees greater accuracy of the network in exchange for a greater number of resources used, both in the implementation with integers and in the implementation with fixed point. tests were carried out with a different number of decimal places to verify the use of resources. Table I and II demonstrates the test results for the implementation with two and three decimal points, respectively, while Table III test results for the fixed point implementation.

Varying decimal places and comparing with the final accuracy of the network, it was determined that even using more resources, the implementation with greater precision of decimal places, both in the case of the use of integers, as in the use of 25 bits at a fixed point, offered better results, since better network accuracy is expected due to higher numerical precision.

C. Network Design

The VHDL code was designed in such a way that it allows flexibility both in the number of neurons and in the number of intermediate layers and also in the number of input values in the network, to be able to adapt the implementation according to the need, without being limited for the MNIST dataset [22], to only one model or the board used for the tests performed in this work.

Tests were carried out with different numbers of layers and neurons, which varies the number of resources used, the

TABLE I: Integer implementation results with two decimal places of precision.

Integer – 2 decimal places	784x10x10	784x15x10	784x30x10	784x45x10	784x100x10	784x10x10x10
LUTs	6273	7704	11979	13960	33117	7811
DSP	10	25	40	55	100	30
BRAM(18KBits)	10	15	30	45	50	30
Registers	2023	2320	3240	4372	8377	2483
F7 MUX	323	546	973	2330	3229	682
F8 MUX	50	101	150	846	648	168
Original accuracy	92,75%	94,92%	96,70%	97,55%	97,88%	92,42%
Energy consumption	0,133W	0,184W	0,190W	0,197W	*	0,211W
Response time	23.77µs	23.97µs	24.57 µs	25.17µs	27.37µs	47.85µs

TABLE II: Integer implementation results with three decimal places of precision.

Integer - 3 decimal places	784x10x10	784x15x10	784x30x10	784x45x10	784x100x10	784x10x10x10
LUTs	9655	11300	13518	17559	27854	14132
DSP	20	25	40	55	110	30
BRAM(18KBits)	0	15	30	45	50	20
Registers	2955	3026	3423	4460	9652	3358
F7 MUX	1357	1187	1333	1689	3152	1493
F8 MUX	220	228	249	263	740	304
Original accuracy	92,75%	94,92%	96,70%	97,55%	97,88%	92,42%
Energy consumption	0,259W	0,274W	0,307W	0,312W	*	0,288W
Response time	23.77µs	23.97µs	24.57 µs	25.17µs	27.37µs	47.85µs

TABLE III: Implementation results with fixed-point binary.

Fixed point – 25 bits	784x10x10	784x15x10	784x30x10	784x45x10	784x100x10	784x10x10x10
LUTs	7863	8344	10573	12792	22570	9532
DSP	20	25	40	55	110	30
BRAM(18KBits)	10	15	40	45	50	20
Registers	2506	2729	3693	4888	8762	3058
F7 MUX	1146	917	1242	1799	3420	1493
F8 MUX	323	128	157	217	613	304
Original accuracy	0,9275	0,9492	0,967	0,9755	0,9788	0,9242
Energy consumption	.179W	.186W	.26W	.198W	*	.192W
Response time	23.77µs	23.97µs	24.57µs	25.17µs	27.37µs	47.85µs

accuracy of the model, and the response time of the network, after different tests it was concluded that networks with more than one layer did not only consumed more resources and more time, as they did not guarantee better results in the accuracy of the network in general, the network response time being independent of the number of neurons, since all neurons in a layer perform calculations in parallel, the network is more effective, both in response time and accuracy, by increasing the number of neurons in the hidden layer.

D. Activation Function

Tests were performed with different activation functions, taking into account the accuracy of the elaborated network and the consumption of resources on the board, as the output layer needs the Sigmoid or SoftMax activation function, tests were carried out with the two functions that presented the best accuracy during training was Sigmoid and as the number of resources consumed when implementing any of the functions is similar, around 2000 LUTs or five BRAMs for implementation with integers with three places of precision and binary with fixed point and around 1000 LUTs for precision with two decimal places the most used features in the implementations, the tests were carried out with Sigmoid, while the activation function of the middle layer were carried out tests with different activation functions as mentioned above and hyperbolic tangent and RELU, the tests carried out demonstrated that the hyperbolic tangent activation function has lower accuracy and would consume more board resources than the function RELU which also had better accuracy compared to Sigmoid.

Observing the results of test implementations, it can be concluded that the number of neurons directly affects the resources used by the network and the expected accuracy, while the number of layers also affects the accuracy, consumes more energy and more resources, and increases the response time without a significant increase in.

Comparing the results of integers with representation using binary with a fixed point we can see that there is significant conservation of resources and less consumption of energy, the only obstacle is the conversion of the input data to binary with a fixed point since the algorithm runs externally, the data needs treatment before processing, which would make the network response time larger.

E. Comparison with CPU and GPU Acceleration

The response time in the different tests performed with the hardware implementation was measured and compared with the results of the software implementation using different tools. For testing with CPU, a test was carried out with Intel i7-6700K 4.00 GHz and 16 GB of RAM, it was also carried out in a Kaggle virtual environment, and the results showed an average response time of 20ms for each image, compared with the response time of one-layer network with 45 neurons implemented in 25.17μs FPGA, it is possible to notice that the response time is considerably lower. It was also compared to GPU acceleration on both an Nvidia GTX 980TI GPU with 6Gb of dedicated video RAM and a virtual environment so the time reduction was not very significant, reducing the average response time for individual images to 18ms.

VI. DISSCUSION

Regarding the network accuracy, it is possible to notice that the consumption of resources is directly related to the precision of decimal places of the network, which affects the final accuracy of the implementation, with the calculated error we can estimate the accuracy of the network according to the expected results.

Taking the results of tests carried out with the network implemented with a response time of 25.17us and considering the average speed of a commercial aircraft as 300 km/h, we can estimate that, with the network response time, the plane would travel 0.0020975 meters or 2.0975 millimeters approximately, between the input of the image and the decision making, considering that the response time depends directly on the number of inputs in the network, even if the number of inputs was ten times greater, the plane would still travel less than one meter between image input and decision making.

There is the possibility of using boards with a greater number of resources to implement networks with a greater number of inputs, neurons, and layers.

The greatest demand of convolutional networks is the memory to store parameters and processing power to perform arithmetic and logic operations the main components of FPGA for neural networks are LUTs, BRAM, and DSPs.

VII. FINAL CONSIDERATIONS

Considering the results acquired through the tests of the research carried out, it can be concluded that the implementation of neural networks for image recognition in FPGA has enormous potential for reducing response time, in addition to being economical in terms of energy consumed.

Initial comparisons with CPUs and GPUs show a significant reduction in response time with the potential for optimizations in the implementation to obtain even better results, in addition, the FPGA implementation allows flexibility in updating the network if necessary, complemented by the fact that the code is itself flexible.

A. OPTIMIZATION PROPOSALS

As previously mentioned, there are several ways to implement neural networks for image recognition in FPGAs, being important factors: the number of input values (pixels), the number of dense layers, the number of neurons in each layer, the representation number and its precision. All these factors influence the complexity of the calculations performed, the consumption of logical resources, memory and power on the board, and the total time required for image processing.

There are a series of optimizations that are possible to perform in the model elaborated, in future works, which could further reduce the response time and improve the consumption of resources by the model, some of these improvements are:

- Develop a controller unit for the arithmetic operations performed by the network, thus improving the use of DSPs, using fewer LUTs, and enabling greater parallelism of the calculations performed, limited only by the amount of DSPs on the FPGA board.
- Greater parallelization of the calculations performed, since the calculations consume more time in image processing, using techniques such as parallel reduction could considerably reduce the response time in exchange for greater consumption of resources and energy per image.

B. FUTURE WORKS

The focus of this work was to evaluate the response time for calculations performed by a neural network developed for image recognition, also considering the use of resources and the energy consumed during tests performed with different parameters, but due to the time required, in addition to limitations of resources on the board used for tests for studies carried out on neural networks, FPGA, VHDL, and implementation techniques. It was not possible to implement different types of layers such as the convolution layer and the max pooling layer to test convolutional neural networks completely implemented in FPGAs, considering that the code elaborated is flexible in terms of the number of input values, number of layers and amount of neurons per layer, it is possible to continue the work in the future with the development of convolution and max pooling layers and integrate them into the work already done. Due to the difficulties of testing large amounts of images to verify the accuracy of the network, it would be necessary to devise a way to send images quickly to the board to verify the accuracy of the implementation.

REFERENCES

[1] S. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. J. Weiss, and K. W. Wilson, "CNN architectures for large-scale audio classification," *CoRR*, vol. abs/1609.09430, 2016. [Online]. Available: http://arxiv.org/abs/1609.09430

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec 2015. [Online]. Available: https://doi.org/10.1007/s11263-015-0816-y

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[4] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[6] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 525–542.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[8] R. Kellermann, T. Biehle, and L. Fischer, "Drones for parcel and passenger transportation: A literature review," *Transportation Research Interdisciplinary Perspectives*, vol. 4, p. 100088, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590198219300879

[9] A. Gupta, A. Anpalagan, L. Guan, and A. S. Khwaja, "Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues," *Array*, vol. 10, p. 100057, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590005621000059

[10] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," 2015.

[11] A. Jahanshahi, M. K. Taram, and N. Eskandari, "Blokus duo game on fpga," in *The 17th CSI International Symposium on Computer Architecture Digital Systems (CADS 2013)*, 2013, pp. 149–152.

[12] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 161–170. [Online]. Available: https://doi.org/10.1145/2684746.2689060

[13] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. 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*, ser. FPGA '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 26–35. [Online]. Available: https://doi.org/10.1145/2847263.2847265

[14] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 16–25. [Online]. Available: https://doi.org/10.1145/2847263.2847276

[15] S. Liang, C. Liu, Y. Wang, H. Li, and X. Li, "Deepburning-gl: An automated framework for generating graph neural network accelerators," in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3400302.3415645

[16] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[17] Artix-7 fpgas data sheet: Dc and ac switching characteristics. Accessed: 2022-08-20. [Online]. Available: xilinx.com/content/dam/xilinx/support/documents/data_sheets/ds181_Artix_7_Data_Sheet.pdf

[18] I. N. d. Silva, D. H. Spatti, and R. A. Flauzino, *Redes neurais artificiais para engenharia e ciências aplicadas* . Artliber Editora, 2010.

[19] Y. L. Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson, *Handwritten Digit Recognition with a Back-Propagation Network*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, p. 396–404.

[20] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, *Object Recognition with Gradient-Based Learning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 319–345. [Online]. Available: https://doi.org/10.1007/3-540-46805-6_19

[21] "Acknowledgments," in *The Design* Warrior's *Guide to FPGAs*, C. ldMaxrd Maxfield, Ed. Burlington: Newnes, 2004, pp. xv–xvi. [Online]. Available: www.sciencedirect.com/science/article/pii/B9780750676045500015

[22] "Mnist handwritten digit database," http://yann.lecun.com/exdb/mnist/, accessed: 2022-08-20.

VICTOR M. AGUIRRE received a B.S. degree in electrical and electronic engineering in 2021 from Mackenzie Presbyterian University, São Paulo, Brazil. His main research interests are Neural Networks, Field Programmable Gate Arrays (FPGA), Industrial Automation, and Programmable Logic Controllers (PLC)..

FADI JERJI (S'18) received a B.S. degree in computer engineering in 2010 and an M.S. degree in electrical and computation engineering from Mackenzie Presbyterian University, São Paulo, Brazil, in 2019. He is currently pursuing a Ph.D.degree in electrical and computation engineering at Mackenzie Presbyterian University. Since 2017 he has been a post-grad Researcher with the Digital TV Research Laboratory at Mackenzie Presbyterian University.

Received in 2022-07-15 | Approved in 2022-08-20