FPGA implementation of MLP, 1D-CNN and TTTratio algorithms for neutron/gamma-ray discrimination using plastic scintillator

Ali Hachem, Yoann Moline, Gwenolé Corre, Frédérick Carrel, Imane Belachheb

FPGA Implementation of MLP, 1D-CNN and TTT_{ratio} algorithms for Neutron/Gamma-ray Discrimination using Plastic Scintillator

Ali Hachem
Université Paris-Saclay, CEA-List, Palaiseau, France
ali.hachem@cea.fr

Imane Belalchheb
Université Paris-Saclay, CEA-List, Palaiseau, France
imane.belachheb@cea.fr

Yoann Moline
Université Paris-Saclay, CEA-List, Palaiseau, France
yoann.moline@cea.fr

Frédérick Carrel
Université Paris-Saclay, CEA-List, Palaiseau, France
frederick.carrel@cea.fr

Gwenolé Corre
Université Paris-Saclay, CEA-List, Palaiseau, France
gwenole.corre@cea.fr

***Abstract*—Pulse shape discrimination algorithms, such as Tail-to-Total integral ratio (TTT_{ratio}) have been commonly integrated on edge devices for online neutron/gamma discrimination using organic scintillators. These algorithms have a number of limitations, especially with plastic scintillators which have low intrinsic discriminating ability. Machine learning (ML) models have recently been explored as a way to improve discriminating performance. Most of these methods are proposed for liquid and stilbene scintillators and do not address the embedded implementation. The aim of this study is to compare the FPGA implementation of TTT_{ratio} algorithm, Multi Layer Perceptron Neural Network (MLP), and 1D Convolution Neural Network (1D-CNN) models that are trained for neutron/gamma-ray discrimination using EJ276 plastic scintillator. Therefore, the comparison between the different methods can be done according to the discrimination performance, latency and resource consumption. The objective is to achieve a latency shorter than the signal duration (500 ns) while using minimal resources.**

***Index Terms*—EJ276, Neutron/Gamma Discrimination, Plastic Scintillator, Organic Scintillators, Embedded Machine Learning, MultiLayer Perceptron Model, 1D-CNN, FPGA, AI Accelerator**

I. Introduction

Organic scintillators have been developed to detect neutrons and gamma-rays in many applications such as homeland security. TTT_{ratio} algorithm, which is also called Charge Comparison Method (CCM), has been widely used to discriminate the detected events [1]–[6]. This algorithm relies on the shape difference between the signals to classify them. The interaction of a neutron results in a longer signal than that generated by a gamma-ray (Fig. 1) [7]. In liquid and stilbene scintillators, the difference between the two resulting signals is more significant than the plastic counterpart. Consequently, this discrimination approach offers better performances with these types of detectors [8]–[10]. In contrast, plastic scintillators have several advantages. They can be manufactured in a larger volume, have a lower cost, and are non-toxic [11].

In our recent work we showed that a two hidden layer MLP model can outperform the TTT_{ratio} algorithm for neutron/gamma-ray discrimination in EJ276 plastic scintillator, especially for low energy radiations ([100, 250] keV) [12].

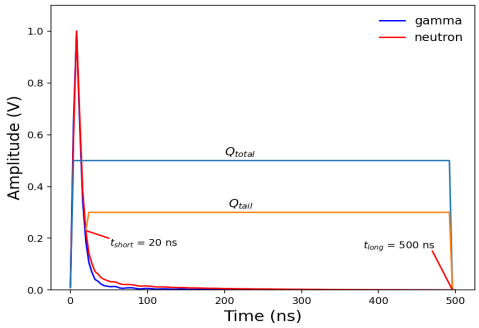


Fig. 1. Average of neutron and gamma-ray signals detected by EJ276 plastic scintillator at 250 MHz. Use of a ^{252}Cf neutron source. Min-max normalization is applied on the average signals.

In some applications, the classification should be achieved online, requiring the implementation of a discrimination approach on an embedded system. Therefore, the comparison between different discrimination approaches should not only be limited to the discrimination performance. Their embedded implementations should also be compared.

Field-Programmable Gate Arrays (FPGA) is a type of integrated circuit that provides a versatile platform for ML acceleration on edge devices due to its reconfigurable nature, parallel processing capability, low latency and high energy efficiency [13]–[15]. By integrating ML algorithms directly into an FPGA, it becomes possible to perform complex computations at high speeds and with low power consumption. The main components of this reconfigurable device are: Look-up tables (LUTs), Flip-Flops (FFs), Block RAM (BRAM) and Digital Signal Processing units (DSPs).

This article compares the FPGA implementation of TTT_{ratio}, trained MLP and 1D-CNN algorithms to discriminate online neutron/gamma-ray using EJ276 plastic scintillator. The comparison between these discrimination approaches is based on discrimination performance, execution time, and

resources consumption. The main objective is to achieve the embedded implementation without a significant degradation in discrimination performance while ensuring an execution time less than the signal duration and using the minimal amount of resource. Thus, we can avoid the loosing of the next pulse. In the presented case, the signal duration required to perform the discrimination is 500 ns.

Section II describes the settings and process employed for FPGA implementation of the three discrimination methods. In section III, the quantization of the MLP and 1D-CNN models, the dataset used for their training, and the comparison of their performances after quantization are presented. Thereafter, sections IV, V, and VI, respectively explain the implementation details on FPGA of TTT_{ratio}, MLP and 1D-CNN. Finally, in section VII, the article draws the main conclusions resulting from this study and based on the obtained implementation results.

II. Configuration and Implementation Process

In the past, FPGA technology could only be reconfigured by engineers with a deep understanding of digital hardware design using Hardware Description Languages (HDLs) such as VHDL or Verilog. The rise of High Level Synthesis (HLS) tools, however, is changing the rules of FPGA programming. HLS is a design methodology that allows designers to describe digital circuits at a higher level of abstraction using software programming languages, such as C/C++ or SystemC. Then these tools automatically convert these high-level descriptions into synthesizable hardware descriptions, which can be used to generate the bit file for FPGA programming.

The circuit synthesis of each discrimination method can be more optimized directly using HDL code instead of using the HLS approach. Nevertheless, the later can be sufficient to achieve the embedded comparison between the different discrimination approaches. The HLS software used in this work is Vitis HLS (2021.2). Furthermore, we only proceeded as far as the C/RTL co-simulation step, and the reports generated in the synthesis step used as the basis for comparing the different discrimination methods. For each discrimination algorithm, the code was first written and validated in C++ using fixed-point arbitrary precision type instead of floating point types. Fixed point data types (ap_fixed <*m, n*>) represent data as a combination of integer and fraction bits, where *m* is the total number of bits and *n* is the number of bits dedicated to the integer part. By optimizing the number of bits for each variable in the code without significantly degrading the discrimination performance, the area of the synthesized circuit can be reduced. Following the validation and tuning of the number of bits, the synthesis process was optimized to meet the required time constraint while using minimal amount of resources. The final step involved validating the synthesized circuit through the C/RTL co-simulation.

Furthermore, the synthesis of the four discrimination algorithm was performed at a clock frequency of 200 MHz (5 ns per cycle) using the Xilinx part number xc7z020-clg484-3. The input was quantized to 12 bits, where all bits were allocated to the decimal part. This quantization approach was chosen because the data was acquired through a 12-bit resolution Analog Digital Converter (ADC), and the maximum signal amplitude is less than one volt. A test bench created from the dataset used to train the ML models (1000 samples) was used for the validation in the C++ simulation and C/RTL co-simulation steps. Qkeras python library (version 0.9.0) was used to quantize the MLP and 1D-CNN models aware training. It has been proved that this approach was more effective than quantizing the model after training [16].

III. Design and Training of MLP and 1D-CNN

The dataset used to train and assess the proposed MLP and 1D-CNN models are the same as those used in [12]. They were prepared at 250 MHz using EJ276 plastic scintillator, where number of neutron and gamma-ray samples are 40600 and 66800, respectively. The signals are separated into 80% for the training and 20 % for the validation. The signal duration is 500 ns which corresponds to 126 sample points. The MLP model trained in [12] for neutron/gamma-ray discrimination consists of an input, an output and two hidden layers of 32 neurons each. The input layer has *n* neurons, which is the number of points encoding a signal. The output layer is one neuron representing the probability that a signal will be a neutron or a gamma-ray. ReLu and Sigmoid are respectively the activation functions of the hidden and last layers.

The quantization of this MLP model was tuned for both the weights and the outputs of each layer using Qkeras framework. The minimal representation avoiding a significant degradation in the discrimination performance is 8-bit, all of them dedicated to the decimal part (Fig. 2). For an energy range of [100, 250] keV where the discrimination process is the most challenging, the True Positive Rate (TPR) is decreased from 87% to 85%, for a False Positive Rate (FPR) equal to 2% (Table I). More information about the energy calibration of the dataset can be found in [12].

Concerning the architecture of 1D-CNN model, first, the number of layers, number of filters in each layer and size of filter were adjusted to optimize the proposed 1D-CNN model size, which is critical to reduce the inference time. The obtained optimal model has one hidden convolution layer (CL). The number of filters, the filter width and stride are equal to 4, 3 and 3, respectively. The input and output layers are the same compared to the MLP model. The obtained results in Fig. 3 show that the main difference of performances between the implemented 1D-CNN and MLP models lies in the detection of low energy radiations ([100, 250 keV]). In this range, the former model outperforms the latter. For FPR equal to 2%, the TPRs achieved by the two models are 91% and 87%, respectively. For energy levels exceeding 250 keV, the Receiver Operating Characteristic (ROC) curves of both models overlap (Fig. 3). These results validate the previous conclusion, indicating that the discrimination between neutrons and gamma-rays in the presented case is challenging when dealing with low energy radiations ([100, 250 keV]).

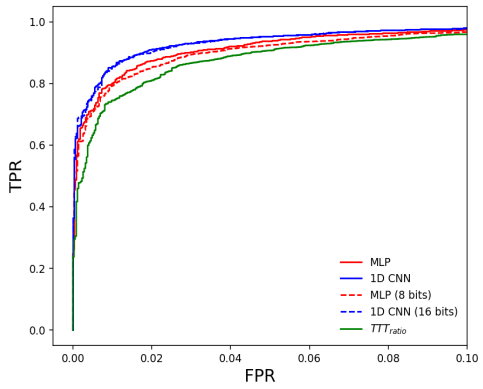


Fig. 2. ROC curves obtained by TTT_{ratio} algorithm, quantized and non quantized MLP and 1D-CNN models on a validation dataset for the energy range [100, 250] keV.

TABLE I
TPR FOR FPR = 2% OBTAINED BY TTT_{ratio} ALGORITHM, QUANTIZED AND NON QUANTIZED MLP AND 1D-CNN MODELS ON A VALIDATION DATASET FOR THE ENERGY RANGE [100, 250] KEV.

Model	TPR for FPR = 2%
MLP	87%
MLP (8 bits)	85%
1D-CNN	91%
1D-CNN (16 bits)	90%
TTT_{ratio}	81%

Thereafter, regarding the MLP model, the quantization of the trained 1D-CNN model was adjusted during training to fine-tune the number of bits used to represent the weights and output of each layer. The obtained optimal representation without significantly impacting the discrimination performance is 16 bits, with 4 bits to the integer part. Furthermore, the obtained results in Fig. 2 and Table I indicate that the quantized 1D-CNN model outperforms the non quantized MLP model.

IV. TTT_{ratio} IMPLEMENTATION

TTT_{ratio} algorithm is based on the decay time difference between neutron and gamma-ray interactions to differentiate them (Fig. 1). It computes the ratio between the tail and total integral of the signals, as shown in equation 1.

$$TTT_{ratio} = \frac{Q_{tail}}{Q_{total}} \tag{1}$$

where $Q_{tail} = \int_{t_{short}}^{t_{long}} f(t)$ and $Q_{total} = \int_{0}^{t_{long}} f(t)$

t_{short} and t_{long} parameters were tuned to optimize the discrimination performance using the optimization algorithm implemented in [17], with the resulting optimum values being 20 ns and 500 ns, respectively. Q_{tail} and Q_{total} integrals in this optimization algorithm are calculated using trapezoidal numerical integration method with delta equal to one.

Q_{tail} and Q_{total} integrals in this optimization algorithm are calculated using trapezoidal numerical integration method with delta equal to one, which is represented by equation 2. The latter indicates that the computing of this numerical method involves one multiplication by a constant (0.5) and k addition operations, where k is the integral length. Thus, the main operations to calculate this ratio and achieve the synthesis for a signal of length n are n and m addition operations for the Q_{total} and Q_{tail} integrals, and one division between both of them.

$$\int_0^k f(t)\,dt \approx \frac{x_0 + x_k}{2} + \sum_{i=1}^{k-1} x_i \tag{2}$$

The algorithm of this method was first coded in C++. Then, the numbers of bits of the parameters responsible for storing the two integral calculation results and their ratio were tuned. ap_fixed *<18, 10>* is the optimal obtained type for the three variables without significant degradation of the discrimination performance. Subsequently, the code was synthesized by paralyzing the calculation of the two integrals. Moreover, experimental results showed that computing the inverse of Q_{tot} integral, then multiplying it by the result of Q_{tail} integral, consumes less time compared to directly dividing the two integrals. The computation of the inverse was done using *hls::recip()* method implemented in Vitis HLS, specifically designed for fixed-point types. ap_fixed *<18, 10>* type was used to store the result of the inverse operation. The obtained synthesis report indicates a total latency of 215 ns using 9 DSPs for *hls::recip()*, 4325 FFs and 3848 LUTs.

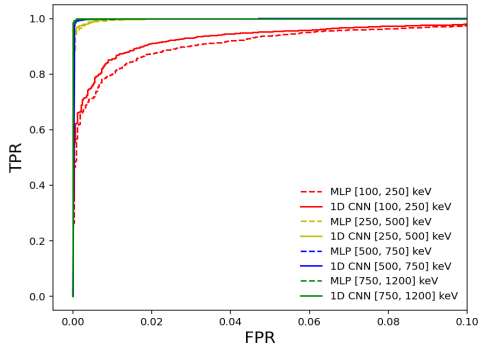


Fig. 3. ROC curves obtained by MLP and 1D CNN models on validation dataset at 250 MHz.

V. MLP IMPLEMENTATION

Different approaches have been proposed to optimize the inference of MLP on FPGA [13], [18], [19]. The works by [18] and [19] explain the implementation of two MLP models trained for medical diagnosis and digital recognition tasks, respectively. The authors in [13] developed and evaluated a general MLP flow that can take arbitrary datasets as input and automatically produce optimized neural network architectures and hardware designs based on a set of constraints and fitness functions such as the accuracy, latency, number of operation per second and throughput. They demonstrated that executing a MLP trained model on a FPGA is faster than on

a GPU. Their framework is based on evolutionary optimization algorithms, OpenCL framework and 2D systolic array configuration. In the presented case we worked to optimize the embedded implementation of the trained MLP model in section III for the neutron/gamma-ray discrimination task, while respecting the time constraint. Building an automatic framework is not an objective of this study.

The trained MLP model in the presented case has three main components: dense layer, ReLu and Sigmoid activation functions (Fig. 4). The equation of Sigmoid is $1/(1+exp^{-x})$, where *x* is the output of the last dense layer. Calculating the exponential component of this function can be computationally challenging for hardware implementation. Therefore, to speed up the hardware computation, Sigmoid function can be approximated using piecewise linear approach and a lookup table. First, a lookup table is created, dividing the input range into *N* segments and calculating Sigmoid values for each segment using the standard Sigmoid function. This created table is stored in BRAM during the synthesis. Then, the segment index of *x* is calculated based on the input range, where the obtained index corresponds to the position of the element in the created lookup table representing the Sigmoid value of *x*. The input range and the number of segments *N* should be tuned to preserve a similar precision of the standard Sigmoid function. Concerning the ReLu function, its computation is fully unrolled during the synthesis for all layers.

Dense layer is the fundamental component of a MLP model. The computation of this layer is done via two nested *for* loops. The first one iterates over the number of neurons (*m*) in the dense layer, and the second one iterates over the input length (*n*), as shown in algorithm 1. In other words we have *n*m* Multiply–Accumulate (MAC) operations for one layer. The two loops of each layer can be unrolled to optimize the synthesis, where the unrolling factor of a loop is a divisor of its number of iterations. The parallel execution of the first loop requires to copy the data *k* times, where *k* is the unrolling factor of the loop, as shown in Fig. 5. Therefore, the parallel execution of the second loop of all the layers has the first priority. However, there is a memory dependency problem when the second loop is unrolled. In algorithm 1, the next iteration reads the variable *output[i]*, while the previous iteration writes data to *output[i]*. Consequently, the subsequent iteration cannot start until the prior one is completed. An adder tree structure can be a solution to this problem (Fig. 6). In this approach, all *n* multiplication operations from the second loop are first computed and loaded into a *temp* array of length *n*. Subsequently, the sum of the elements in *temp* is calculated using the adder tree function, which significantly reduces the calculation complexity from O(n) to O(log(n)).

The trained MLP model in section III consists of three dense, two ReLu and one Sigmoid operations (Fig. 4). The dimensions of the weight matrices (loop parameters) for these layers are (32, 126), (32, 32), and (1, 32) respectively. The structure of this model was first implemented in C++. Then, the synthesis was optimized by tuning the unrolling factors for each layer. The obtained optimal *unrolling factors* for the loop iterations in the three layers are (1, 126), (1, 32), and (1, 32) respectively, with corresponding latency of 39, 37, and 6 cycles. The Sigmoid function uses 1 BRAM for lookup table and takes 2 cycles for execution, with input range (-8, 8) and 1024 segments. The first layer consumes 126 DSPs for MAC operations, while the MAC operations of the second and third layers are computed by LUTs. This feature in Vitis HLS, can only be applied when the multiplication operation involves numbers represented by a maximum of 8 bits each. Weight storage in FFs and LUTs is preferred over BRAMs to avoid latency increase due to read and write operations. The obtained synthesis report in Table II indicates a total latency of 490 ns (98 cycles) using 126 DSP units, 46,059 LUTs, 17,657 FFs, and 1 BRAM.

Algorithm 1 Dense algorithm

Input: $input$ array with length n and weight matrix (w) with dimensions $m \times n$
Output: $output$ array with length n

```
function Dense(input, weight)
    for i ← 0 to m do
        output[i] ← b[i]
        for j ← 0 to n do
            output[i] += input[j] · w[i][j]
        end for
    end for
return output
```

Another synthesis approach can yield a latency value of 350 ns. In this proposed solution, the unrolling factors of the three layers are (32, 3), (32, 4) and (1, 32). This solution enables the parallel computation of 256 multiplications. It should be noticed that only 96 DSP units are employed for the first layer, with the remaining 160 multiplications being executed by LUTs. However, in this solution the weights of first and second layers and 32 copies of the input signal are stored in BRAMs. Therefore, the number of BRAMs is increased to 220 (Table II). In summary, a trade-off exists between the latency and resources of both solutions. The choice between them depends on the specific requirements of the target application.

VI. 1D CNN IMPLEMENTATION

Different methods for incorporating machine learning into FPGA devices primarily focus on 2D-CNNs for image recognition tasks [20]–[28]. Different studies have addressed the implementation of 1D-CNN on FPGA for 1D signal applications [14], [29]. The implementation of a trained 1D-CNN designed for underwater target spectrum recognition is optimized in [29]. The authors used three main techniques: quantization, loop unrolling and tilling. Moreover, they assessed the impact of these techniques on the resource consumption of their 1D-CNN model implementation. The authors in [14] propose a pyramid layer-folding pipeline structure to implement a 1D-CNN model trained for speaker recognition task.

The trained 1D-CNN model has four components: 1D CL, dense layer, ReLu and Sigmoid activation functions. The

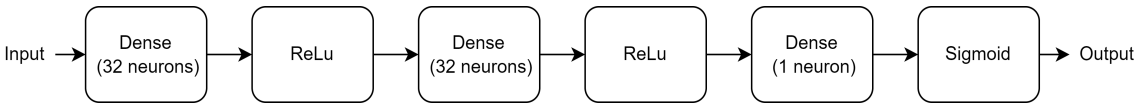


Fig. 4. The inference flow of the trained MLP model.

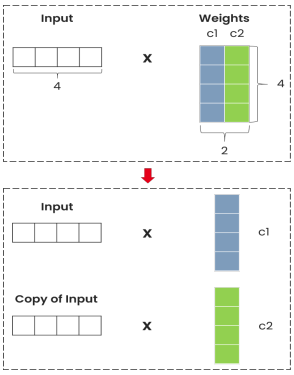


Fig. 5. Illustration of the first loop unrolling in dense layer.

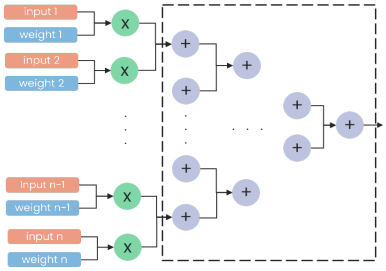


Fig. 6. Adder tree structure

calculation in the 1D CL goes through four nested *for* loops. They iterate over the output length (*out_size*), number of filters (*nf*), number of input channels (*nc*), and kernel width (*k*), respectively (algorithm 2). Thus, the total number of MAC operations for a layer is $out\_size * nf * nc * k$.

The parallel execution of the first loop involves copying *(k-s)* * *nc* columns of the input data and the kernel matrix of the layer *m* times, where *m* and *s* are unrolling factor of the loop and the stride, respectively (Fig. 7(a)). The parallel execution of the second loop requires duplicating the input data *m* times (Fig. 7(b)), while the unrolling of third and fourth loops does not impact the data size (Fig. 7(c)). In our approach, the primary focus is on the parallel execution of the third and fourth loops for all layers. Subsequently, if these unrolling steps do not help to meet the time constraint, we tune the unrolling factor of first and second loop. The dense layer is the output layer including a single neuron. The number of multiplications performed in this layer is determined by multiplying the output length of the last CL by its number of channels. To optimize the implementation, it is necessary to tune the unrolling factor, which should be a divisor of this multiplication result, along with other unrolling parameters of the CLs. The Sigmoid and ReLu activation functions can be synthesized in the same manner than in the MLP model.

Algorithm 2 1D Convolution layer algorithm

Input: $input$ and weight (w) matrices with dimensions $nc \times n$ and $nc \times k \times nf$

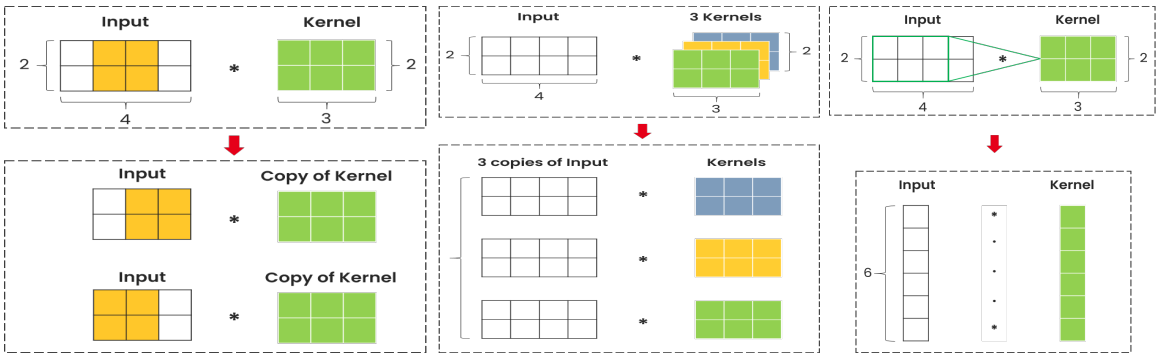
Output: $output$ matrix with dimensions $out_size \times nf$

function $1DCNN\_Layer(input, weight)$
  for $i \leftarrow 0$ to out_size **do**
    for $j \leftarrow 0$ to nf **do**
      $output[j,i] \leftarrow b[i]$
      for $c \leftarrow 0$ to nc **do**
        for $p \leftarrow 0$ to k **do**
          $output[j,i] \mathrel{+}= input[m, s*(p-1)+k] \cdot w[m][p][j]$
        end for
      end for
    end for
  end for
$output$

First, the structure of the obtained quantized 1D-CNN model in section III was implemented in C++ using *ap_fixed* type. The results obtained from the Vitis HLS simulation showed that the performance of the model was significantly degraded, even when a higher number of bits (higher than 16) was used. This may be due to the difference between the implementation of the quantization methods in Qkeras and Vitis HLS, leading to variations in quantization errors. Some information is lost during quantization by Vitis HLS due to rounding errors, which can impact the model performance as observed in the presented case. To address this problem, increasing the size of the model can be a potential solution. A larger model might be more resilient to quantization errors as it can distribute the errors across more parameters and absorb the quantization noise, leading to less performance drop.

Therefore, the size of the model was tuned by quantizing various model sizes aware training and then assessing their discrimination performances through C++ simulation in Vitis HLS. The obtained optimal model consists of two CLs, two ReLu, one dense and one Sigmoid operations (Fig. 8). The optimal number of bits for quantization is 16, with 4 of them dedicated to the integer part. Both CLs have 4 filters, a stride (*S*) of 3, and a filter width of 3. In other words, the dimensions of the four loops for the CL layers are (42, 4, 1, 3) and (14, 4, 4, 3) respectively. The input length of the dense layer is 56, obtained by multiplying the output length of the second CL (14) by its number of filters.

To optimize the synthesis process, fine tuning is made to unrolling factors for each CL layer, number of segments, and input range of the Sigmoid function, resulting in optimal



(a) Unrolling of first loop, s=1, k=3, and nc=2 (b) Unrolling of second loop, nc=2, nf=3 (c) Unrolling of third and fourth loops

Fig. 7. Illustration of 1D CL loops unrolling

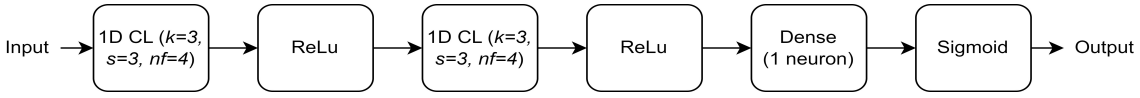


Fig. 8. The inference flow of the trained 1D-CNN model.

TABLE II
SUMMARY REPORT OF SYNTHESIS FOR 1D CNN, MLP, AND TTT_{ratio} NEUTRON/GAMMA-RAY DISCRIMINATION METHODS.

Model	Latency	BRAM	DSP	FF	LUT
MLP 1	490 (ns)	1	126	17657	46059
MLP 2	350 (ns)	220	96	20580	51177
1D CNN	490 (ns)	2	228	193625	156833
TTTratio	215 (ns)	0	9	4325	3848

values of (42, 1, 1, 3), (1, 4, 4, 3), 2048, and (-32, 32) respectively. The dense layer was fully unrolled. As a result, the total latency achieved is 490 ns (Table II). The 2 consumed BRAMs are dedicated for the lookup table of the Sigmoid. The 228 consumed DSPs are distributed as follows: 128 for the first CL, 44 for the second CL, and 55 for the dense layer.

These results demonstrate that, in the presented case, the MLP model can be effectively implemented on a FPGA for online classification with fewer resources compared to the 1D-CNN model, while achieving the same latency (Table II). However, it is important to note that the discrimination performance of the 1D-CNN model is higher than that of the MLP model. Furthermore, Table II indicates that both the latency and resource usage of TTT_{ratio} algorithm are considerably lower than those of ML models. Nevertheless, for relatively low energy range ([100, 250] keV), this discrimination method is significantly less efficient than ML models [12]. Therefore, in applications involving classification of low energy radiations, the higher resources consumed by MLP and 1D-CNN models can be justified. In contrast, for the discrimination of higher energy radiations, this algorithm is more advantageous as it achieves the same discrimination performance as ML models with lower latency and resource consumption.

VII. CONCLUSION

This study compared the embedded FPGA implementations of the TTT_{ratio} discrimination algorithm, MLP, and 1D-CNN models for neutron/gamma-ray discrimination using EJ276 plastic scintillator. The main objective was to optimize the implementation of each method, aiming for a latency lower than the signal duration while minimizing resource consumption. Based on discrimination performance as the evaluation metric, the 1D-CNN model outperforms the MLP model, which, in turn, outperforms the TTT_{ratio} algorithm, especially for low energy radiations ([100, 250] keV). However, when the comparison is based on latency and resource consumption, the order of the three methods is completely inverted. Therefore, the choice of the method depends on the target application and the available resources for implementation. For a more accurate comparison between the three methods, a hardware implementation should be considered for future work.

REFERENCES

[1] J. Adams and G. White, "A versatile pulse shape discriminator for charged particle separation and its application to fast neutron time-of-flight spectroscopy," *Nuclear Instruments and Methods*, vol. 156, no. 3, pp. 459–476, 1978.

[2] O. McCormack, L. Giacomelli, G. Croci, A. Muraro, G. Gorini, G. Grosso, R. Pasqualotto, E. P. Cippo, M. Rebai, D. Rigamonti, *et al.*, "Characterization and operational stability of ej276 plastic scintillator-based detector for neutron spectroscopy," *Journal of Instrumentation*, vol. 16, no. 10, p. P10002, 2021.

[3] A. Tomanin, J. Paepen, P. Schillebeeckx, R. Wynants, R. Nolte, and A. Lavietes, "Characterization of a cubic ej-309 liquid scintillator detector," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 756, pp. 45–54, 2014.

[4] M. Grodzicka-Kobylka, T. Szczesniak, M. Moszyński, K. Brylew, L. Swiderski, J. Valiente-Dobón, P. Schotanus, K. Grodzicki, and H. Trzaskowska, "Fast neutron and gamma ray pulse shape discrimination in EJ-276 and EJ-276G plastic scintillators," *Journal of Instrumentation*, vol. 15, no. 03, p. P03030, 2020.

[5] N. Zaitseva, A. Glenn, A. Mabe, M. Carman, C. Hurlbut, J. Inman, and S. Payne, "Recent developments in plastic scintillators with pulse shape discrimination," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 889, pp. 97–104, 2018.

[6] E. Ryabeva, I. Urupa, E. Lupar, V. Kadilin, A. Skotnikova, Y. Kokorev, and R. Ibragimov, "Calibration of ej-276 plastic scintillator for neutron–gamma pulse shape discrimination experiments," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 1010, p. 165495, 2021.

[7] F. Brooks, "Development of organic scintillators," *Nuclear Instruments and Methods*, vol. 162, no. 1-3, pp. 477–505, 1979.

[8] T. Laplace *et al.*, "Comparative scintillation performance of EJ-309, EJ-276, and a novel organic glass," *Journal of Instrumentation*, vol. 15, no. 11, p. P11020, 2020.

[9] F. Ferrulli, N. Dinar, L. G. Manzano, M. Lablme, and M. Silari, "Characterisation of stilbene and EJ-276 scintillators coupled with a large area sipm array for a fast neutron dose rate detector," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, p. 165566, 2021.

[10] M. Grodzicka-Kobylka, T. Szczesniak, M. Moszyński, K. Brylew, L. Swiderski, J. Valiente-Dobón, P. Schotanus, K. Grodzicki, and H. Trzaskowska, "Fast neutron and gamma ray pulse shape discrimination in EJ-276 and EJ-276G plastic scintillators," *Journal of Instrumentation*, vol. 15, no. 03, p. P03030, 2020.

[11] G. F. Knoll, *Radiation detection and measurement*. John Wiley & Sons, 2010.

[12] A. Hachem, Y. Moline, G. Corre, J. Gauthier, and F. Carrel, "Multilayer perceptron model vs charge comparison method for neutron/gamma discrimination in plastic scintillator according to sampling frequency and energy radiation," *IEEE Transactions on Nuclear Science*, pp. 1–1, 2023.

[13] P. Colangelo, O. Segal, A. Speicher, and M. Margala, "Automl for multilayer perceptron and fpga co-design," in *2020 IEEE 33rd International System-on-Chip Conference (SOCC)*, pp. 265–266, IEEE, 2020.

[14] J. Xu, S. Li, J. Jiang, and Y. Dou, "A simplified speaker recognition system based on fpga platform," *IEEE Access*, vol. 8, pp. 1507–1516, 2019.

[15] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–39, 2018.

[16] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[17] C. Lynde, E. Montbarbon, M. Hamel, A. Grabowski, C. Frangville, G. H. Bertrand, G. Galli, F. Carrel, V. Schoepff, and Z. El Bitar, "Optimization of the charge comparison method for multiradiation field using various measurement systems," *IEEE Transactions on Nuclear Science*, vol. 67, no. 4, pp. 679–687, 2020.

[18] A. Sanaullah, C. Yang, Y. Alexeev, K. Yoshii, and M. C. Herbordt, "Real-time data analysis for medical diagnosis using fpga-accelerated neural networks," *BMC bioinformatics*, vol. 19, pp. 19–31, 2018.

[19] I. Westby, X. Yang, T. Liu, and H. Xu, "Fpga acceleration on a multilayer perceptron neural network for digit recognition," *The Journal of Supercomputing*, vol. 77, no. 12, pp. 14356–14373, 2021.

[20] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 2072–2085, 2018.

[21] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 37, no. 1, pp. 35–47, 2017.

[22] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *Proceedings of the 54th Annual Design Automation Conference 2017*, pp. 1–6, 2017.

[23] Y. Shen, M. Ferdman, and P. Milder, "Maximizing cnn accelerator efficiency through resource partitioning," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 535–547, 2017.

[24] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I.-A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S.-C. Liu, *et al.*, "Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 3, pp. 644–656, 2018.

[25] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnbuilder: An automated tool for building high-performance dnn hardware accelerators for fpgas," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2018.

[26] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 152–159, IEEE, 2017.

[27] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, and D. Chen, "Fpga/dnn co-design: An efficient design methodology for iot intelligence on the edge," in *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.

[28] Y. Liang, L. Lu, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on fpgas," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 4, pp. 857–870, 2019.

[29] W. Wang, X. Zhao, and D. Liu, "Design and optimization of 1d-cnn for spectrum recognition of underwater targets," *Integrated Ferroelectrics*, vol. 218, no. 1, pp. 164–179, 2021.