Low-Latency Neutron-Gamma PSD using Pulse-Coupled and Convolutional Neural Networks

Fabrizio Chinu[1], Marcello Di Costanzo[2], Valerio Pagliarino[3]

[1] MSc. Student in Physics, Università degli Studi di Torino - fabrizio.chinu@edu.unito.it
[2] MSc. Student in Physics, Università degli Studi di Torino - marcello.dicostanzo@edu.unito.it
[3] MSc. Student in Physics, Università degli Studi di Torino - valerio.pagliarino@edu.unito.it

***Abstract*—In this work a deep learning-based technique using Pulse-Coupled Neural Networks (PCNN) and Convolutional Neural Networks (CNN) is applied to a neutron-gamma Pulse Shape Discrimination (PSD) task. The model is provided with a digitized signal from a scintillator coupled with SiPM. After a post-training analysis of the CNN model, the *transfer learning* approach is used to investigate the applicability to other particle detectors. Finally, the CNN model is compressed, quantized and deployed on Field Programmable Gate Array real-time electronics. The final model, on a balanced dataset of 9324 items, obtained an accuracy of 99.98 $\pm$ 0.02 % in binary classification with the same detector. An accuracy of 98.28 % and 99.83 % is obtained after the transfer learning, targeting two different particle detector.**

I. Introduction to real-time neutron-gamma discrimination

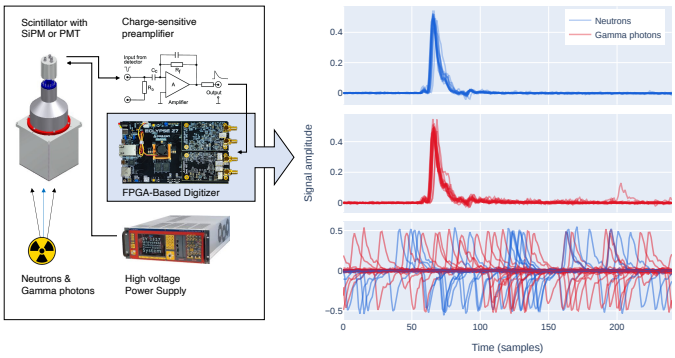

Figure 1. Experimental setup and raw data from the experiment labelled with the Pulse-Coupled Neural Network. In the lower frame: data augmentation for training the CNN introducing time and signal polarity invariance.

The task of discriminating neutron and gamma signals in a single scintillator-based particle detector is very important not only for fundamental research in nuclear physics, but also for a variety of applications ranging from radioactive sources analysis to nuclear reactors monitoring and research in medical physics. Both gamma rays and neutrons are neutral and not easy to shield selectively, therefore this task is considered challenging. A promising technique consists in the analysis of the waveform of the signals produced by plastic scintillators when used to detect such particles: the Pulse Shape Discrimination (PSD). This is feasible since the pulses from neutron and gamma rays are expected to have tails with different shapes, in particular the ratio between the charge integral of the whole signal and the charge integral in the tail is expected to differ. Despite the direct numeric calculation of such integrals is a widely adopted method, it is not considered the most accurate: in this project we investigate the feasibility of a classifier based on deep learning architecture that can be implemented in hardware on the Field Programmable Gate Array (FPGA) electronics already present in a large amount of digitizers for nuclear detectors. In the figure 1 the experimental setup is represented: an organic scintillator is coupled to a SiPM, connected to a charge-sensitive amplifier and to a proper high voltage power supply. The output of the charge-sensitive amplifier is connected to the input of an FPGA-based digitizer. The raw data are augmented introducing an arbitrary time shift and a polarity inversion to improve the generalisation capability of the model.

II. Methods

A. *Pulse-Coupled Neural Network and Convolutional Neural Network Architectures*

Pulse-Coupled Neural Networks are a class of neural network models inspired by the neurons in the brain [1] trying to reproduce their timing behavior. Therefore, they exhibit good performances in processing time-varying inputs such as the time series of neutron-gamma discrimination. These neural networks perform better than other discrimination techniques [8] and this is attributed to their ability to recognize and capture the dynamic information of the scintillator signals, which is the most important information for the discrimination process. The model of the PCNN incorporates a dendritic tree, a linking modulation, and a pulse generator. Each of the time samples of the signal (or pixels in case of a 2D image) has a corresponding PCNN neuron. Each neuron receives information from the pixel it corresponds to via the feedback input (FI) and from the adjacent neurons through both the feeding and linking (LI) inputs, which together make up the dendritic tree. Both the FI and the LI contribute to the internal activity of the neuron, which determines its activation state. Since the output of the neighbouring neurons has an influence over the activation state, this network is able to achieve synchronization between its neurons (i.e. if a neuron fires, the adjacent neurons have a greater probability of firing in the next iterations). The PCNN model is also able to reproduce the refractory time of the biological neurons by utilizing a dynamic threshold: once the neuron fires, the threshold is set to a very high value, which prevents the neuron from firing again in the following few iterations. The threshold is then reduced through an exponential decay and the neuron will then fire again once the internal activity of the neuron exceeds its dynamic threshold.

To discriminate neutron signals from γ ones, an ignition map, i.e. a vector with the same dimensions as the signal, containing the number of ignitions of each neuron, is obtained for each input. The pulses are discriminated by summing the number of times that the PCNN neurons corresponding to the sampling points of a time interval of interest has fired. This number will be named R in the following sections; the time interval where the ignition map is integrated was chosen as $(\overline{\mathrm{P}} - 7, \overline{\mathrm{P}} + 123)$ ns, $\overline{\mathrm{P}}$ being the time when a pulse reaches its maximum value. By setting a threshold value on R it is possible to label the signals as neutron's or γ's.

The PCNN is peculiar because it doesn't need a specific training, but only the setting up of a few parameters, that in our case we adapted from the work by Hao-Ran Liu et al [8], and are reported in Appendix V-C, together with the mathematical description of the neuron.

Although PCNNs demonstrate outstanding performances in this task, their hardware implementation faces challenges in terms of overall complexity. To overcome these limitations, a PCNN model is used as a teacher to train a CNN student model. Given the unsupervised nature of the problem, the PCNN is used to provide the labels of the signals to the CNN, which is then trained to predict those labels. The structure of the PCNN and of the CNN used in this work are reported in Figure 2.

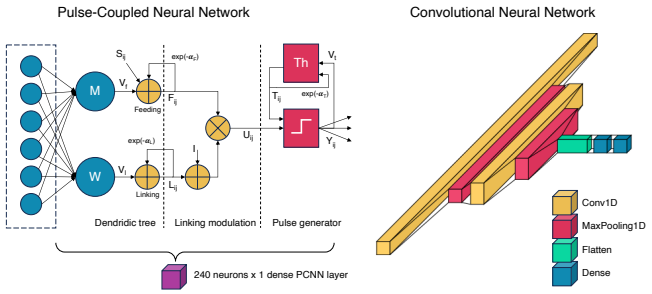

Figure 2. Architecture of the Pulse-Coupled Neural Network and Convolutional Neural Network models.

B. *Transfer learning to other radiation detectors*

The shape of pulses produced by a certain particle stricly depends on detector features, e.g. shaping time and gain. However, since the pulse shape difference between neutrons and gammas are due to their physical properties, similar behaviour for the relative difference of pulses is expected in different scintillators. As a consequence, it is worth investigating whether models optimised in PSD for a specific scintillator can be obtained via transfer learning from one optimised for another detector.

Transfer learning is implemented by fine-tuning the first and second convolutional layers of the net and freezing all other layers. This approach, which differs from the common procedure consisting in fine-tuning the final dense layers, is rewarding for a PSD task. That is because the convolutional layers are in charge of extracting the meaningful features that are related to the physics of the experiment from the detector-dependent raw signal. Moreover, this approach implies an even greater reduction of the number of parameters to retrain than freezing the final dense layers, as shown in appendix V-E.

C. *Model compression, quantization and deployment on FPGAs*

In order to run the classifier in real-time obtaining an inference latency in the order of 10-50 μs a specific hardware is needed, consisting in a Field Programmable Gate Array, in particular the Xilinx XC7Z020. FPGAs are integrated circuits containing many logical slices, lookup tables, distributed RAMs, digital signal processors (DSPs) and programmable IOs that can be interconnected in the programming phase in order to obtain a user-designed digital circuit at the logic-gate-level. This is the highest level of hardware optimisation, inferior only to the creation of a custom integrated circuit (ASIC). Furthermore, this allows to deploy the CNN directly inside the digitizer.

In order to implement the CNN on FPGA, the model is compressed using a progressive pruning that allows to obtain a weight sparsity up to 70 %. The pruning is performed while running a fine-tuning training, by increasing the sparsity of the 2.3 % before starting each epoch, on average. The exact number of pruned weights at each epoch is given by the polynomial decay algorithm. The final model is passed to the HLS4ML (High Level Synthesis for Machine Learning) framework [5] that converts the model to low-level C++ code optimized for the Vivado HLS tool, together with ASCII files containing the weights. After being analyzed and manually adjusted, this code is passed to Vivado HLS that produces Verilog code ready to be integrated inside an FPGA firmware. In this phase, the forward propagation though the network is implemented by using the DSP48E blocks [9], that are high performance signed adder-multipliers. Furthermore, the weights and inputs must be converted from floating point to fixed point representation (16 bits). The detailed architecture of the DSP48E block can be found in the appendix. 8

III. Model training, evaluation and discussion of results

A. *Evaluation of the unsupervised PCNN model*

The R distributions generated by the PCNN model for the three different datasets revealed distinct peaks for the two types of signals, with neutrons exhibiting higher values and gammas displaying lower values, as it can be seen in Figure 3. This clear separation in the output distributions demonstrated the PCNN's capability to effectively discriminate between neutron and gamma signals.

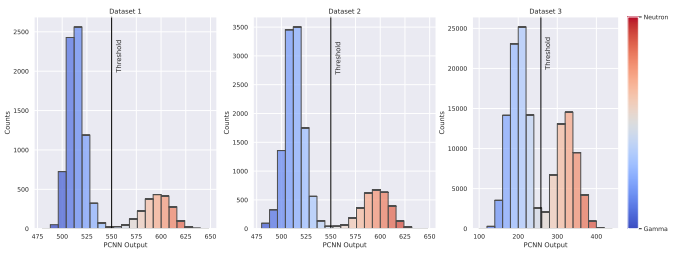

Figure 3. Distributions of the PCNN outputs for the different analysed datasets

In order to quantitatively describe the model performance, the following Figure of Merit (FoM) metric is used:

$$\mathrm{FoM} = \frac{\mathrm{S}}{\mathrm{FWHM_n} + \mathrm{FWHM_\gamma}} \ ,$$

where S is the distance between the neutron and γ peaks, while the FWHMs are the full width at half maximum of the distributions; they are both evaluated though a double-gaussian fit of the model output distribution. A larger FoM-value corresponds to a better discrimination effect. The model obtained a FoM-value of 1.589, 1.536 and 1.028 on the datasets 1, 2 and 3, respectively. Finally, to extract the labels needed for training CNN model, the threshold values on the PCNN outputs reported in Table I are used.

	Dataset 1	Dataset 2	Dataset 3
PCNN Threshold	550	550	258

Table I
THRESHOLD VALUES APPLIED ON THE PCNN OUPUTS

B. Results of the supervised CNN model training with hyperparameters optimization

PCNN validation on various datasets allows a confident supervised training of the CNN *student models* with PCNN-labelled data. The 1D-CNN model is then trained on an equal number of examples for gamma and neutrons taken from Dataset 3 in order to train and evaluate the model on a balanced dataset V-B. The dataset is extended with two types of data augmentation: polarity inversion and temporal translation as shown in figure 1. The data augmentation helps in reducing the generalization error and improving the flexibility of the model to adapt to different operative scenarios. After that, the final dataset is split in training set (54%), validation set (36%) and test set (10%).

Before training, model hyperparameters optimization is achieved via the Hyperband algorithm [6] of the *KerasTuner* package [7], an approach which investigates with full training only the hyperparameters configurations that perform better after a user-defined number of epochs, set to 5 in this case. The hyperparameters to be determined are the linear dimension k of the squared kernel in the convolutional layers, the number of filters f for the first convolutional layer (their number for the second layer is set equal to $2f$) the L1 regularization weight L, the number of neurons D in the first dense layer and the learning rate η. The values obtained from the optimization are reported in Table II. The filter size for the pooling layer is set equal to 2. The resulting 1D CNN has 15529 trainable

f	k	L	D	η
4	6	$5{\cdot}10^{-5}$	34	0.001

Table II
MODEL HYPERPARAMETERS OPTIMIZED VALUES

parameters. A detailed representation of the CNN architecture with the number of neurons and trainable parameters for each layer can be found in the appendix V-E. Training is performed with Adam optimizer [4] using the TensorFlow backend [3] for 30 epochs with a minibatch size of 20. The training procedure uses binary cross entropy as loss function and accuracy as metric. Early stopping callback function with patience equal to 5 is also used. After training, the ROC AUC is evaluated on the test set, resulting equal to $1\text{-}3 \cdot 10^{-5}$.

Before the final training on the full dataset, the model accuracy is evaluated via a k-fold cross-validation training in order to have an estimate of its uncertainty. Dataset 3 is split into 12 folds and for each case the model is trained for 7 epochs with Adam optimizer considering a batch size of 20 and early stopping with patience equal to 5. The accuracy obtained is equal to $99.98 \pm 0.02\%$. Figure 4 shows a superposition of the final trained model ROC curve on test set and validation loss and the k-fold cross-validation ROC curve on validation set and loss function on training set.

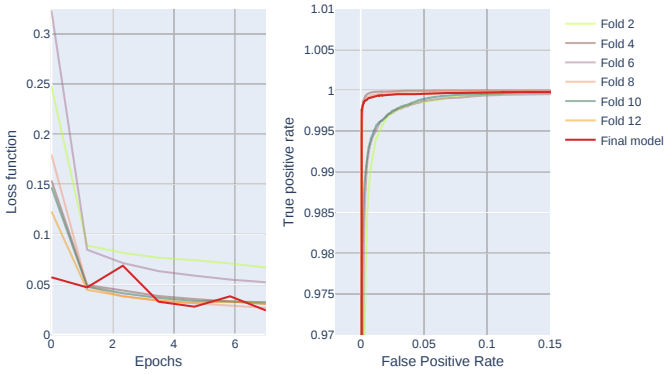

Figure 4. Comparison between training losses and ROC curves on validation set obtained in k-fold cross validation training and final model validation loss and ROC curve on test set.

The figure shows the presence of two trends of the training curve, corresponding to two paths toward different minima of the loss function. While the model trained on all data is able to find the lower minimum, some k-fold trained models remain stuck in the higher minimum, resulting in lower ROC AUCs.

C. Model interpretability

The trained CNN model has been analysed by computing its attention map and by looking at the representation extracted by the convolutional layers. The attention map has been obtained using an occlusion test with a running window covering 10 samples. The output is plotted in figure 5. It is very interesting to observe that the areas of greater attention are the peak and the tail, the same considered by the classical PSD technique based on the direct charge integral ratio computation: the model automatically learned to focus on the right region of the signal. The representation extracted by the convolutional

layers has been investigated by plotting the output, after passing it through the t-SNE algorithm [2] for reducing the dimensionality. The t-SNE algorithm (t-distributed stochastic neighbor embedding) performs a non-linear dimensionality reduction by initially constructing a probability distribution over pairs of high-dimensional object, in such a way that similar object are assigned a higher probability. In a second step, a distribution between points in the low-dimensional output is built. Finally, the Kullback-Leibler divergence between the two distributions is minimised. By looking at the 3D representation it is possible to appreciate the ability of the convolutional layers to extract meaningful features that make the classification problem easily separable, since the points are clustered in 4 clouds. It is interesting to notice that the 2 clouds for each label represents the 2 possible signal polarities introduced by the data augmentation.

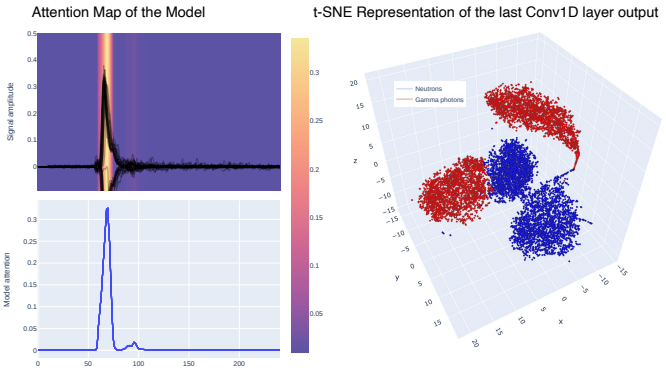

Figure 5. Convolutional model interpretation using an occlusion test and the t-SNE dimensionality reduction algorithm.

D. *Transfer learning*

Transfer learning performance of the trained CNN model is investigated on 2 datasets containing neutron and gamma pulses from scintillators whose material and geometry differ from the one of the training dataset. In both cases the original model has been tested by evaluating its accuracy on the target dataset test set before the transfer fine tuning. The results are reported in Table III. Then, datasets are split in training set (48%), validation set (28%) and test set (30%). The two convolutional layers of the model are fine-tuned with a 20-epochs training having minibatch size set equal to 20. Adam optimizer, learning rate $\eta = 0.0008$, binary cross entropy loss function, accuracy as metric and early stopping with patience equal to 5 are chosen for the training. The accuracy of the transferred models, evaluated on their respective test sets after fine tuning, are reported in Table III, while the training and validation loss function and ROC curves are shown in 6.

	Dataset 1	Dataset 2
Tran. model on target dataset before fine tuning	48.95%	49.34%
Tran. model on target dataset after fine tuning	98.28%	99.83%

Table III
TEST SET ACCURACIES FOR TRANSFER LEARNING ON DATASETS RELATED TO OTHER SCINTILLATORS

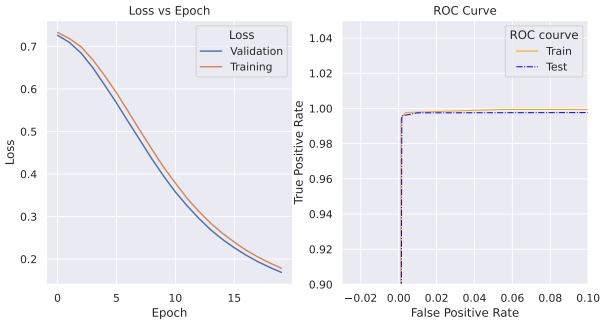

Figure 6. Validation and training loss as a function of the number of epochs, ROC curves on training and test set for the transferred model.

The results show that the scintillator impact on pulse shapes is not negligible. A transfer learning approach with fine-tuning of the 228 convolutional layers parameters leads to a sufficient accuracy for the majority of neutron-gamma PSD applications. This operation has a very low computational cost compared to the standard dense layers fine-tuning, which would lead to retraining 15301 parameters.

E. *Model compression and deployment*

The model compression allowed to reduce the number of parameters and multiplications in the forward propagation, thus reducing the required level of pipelining and making possible a fully parallel architecture, capable of producing a prediction in $\sim$30 μs with a clock of 200 MHz. The accuracy of the final compressed and quantized model is 97.59 %, with the 2.04 % of wrongly classified neutrons and 0.36 % of wrongly classified photons on a balanced test set containing 10361 items. The low level C++ code has been synthesized to Verilog RTL and implemented on the FPGA without hold/setup timing violations.

IV. CONCLUSIONS AND OUTLOOKS

In this work, a PCNN-CNN teacher-student based n-γ discrimination method is proposed. The PCNN model effectively separated the neutron signals from the gamma signals in unlabeled datasets, achieving a FoM-value ranging from 1.028 to 1.589. The student CNN, trained to predict the labels assigned by the PCNN, exhibited a remarkable accuracy of 99.98 ± 0.02 %. Moreover, the CNN model demonstrated its versatility by being transferred to two additional datasets, where it achieved high accuracies of 98.28 % and 99.83 % after fine-tuning. The CNN model was then pruned using a polynomial decay algorithm, effectively reducing the model's computational complexity by 70%. Additionally, we employed the HLS4ML package, enabling quantization of the CNN model and its hardware implementation on FPGAs. Future perspectives for this research involve investigating quantization-aware training of the CNN using QKeras. This would allow to achieve better optimization of the hardware implementation, enabling the CNN model to maintain high accuracy while reducing resource requirements and power consumption on FPGA devices, allowing for future advancements in real-time radiation detection systems.

V. APPENDIX

A. Source code

The source code can be downloaded from the following repository: https://github.com/valeriopagliarino/Neutron-Gamma-PSD-2023-UNITO.

B. Size of the different datasets

	# of pulses	# of time samples per pulse
Dataset 1	9414	240
Dataset 2	14404	240
Dataset 3 PCNN	134130	240
Dataset 3 CNN	103606	240
Augmented Dataset 3 CNN	207212	240

Table IV
DATASET FEATURES

C. Mathematical description of the PCNN neuron

The mathematical formulae of the PCNN are given by:

$$F_i[n] = e^{-\alpha_F} F_i[n-1] + V_F \sum_j M_{ij} Y_j[n-1] + S_i \quad (1)$$

$$L_i[n] = e^{-\alpha_L} L_i[n-1] + V_L \sum_j M_{ij} Y_j[n-1] \quad (2)$$

$$U_i[n] = F_i[n]\{1 + \beta L_i[n]\} \quad (3)$$

$$Y_i[n] = \begin{cases} 1 \text{ if } U_i[n] > T_i[n] \\ 0 \text{ otherwise} \end{cases} \quad (4)$$

$$T_i[n] = e^{-\alpha_T} T_i[n-1] + V_T Y_i[n-1] \quad (5)$$

where $\mathbf{F_i}$ and $\mathbf{L_i}$ denote the FI and LI of the i-th sample of the signal, the αs and $\mathbf{V}$s represent the decay times and the amplification coefficients of FI and LI; j represents the coordinate of the neighboring neurons of i, n is the iteration count. $\mathbf{S_i}$ is the amplitude of the i-th sample of the signal, β is the relative influence of the linking input in the internal activity of the neuron $\mathbf{U_i}$; $\mathbf{T_i}$ is the dynamic threshold and $\mathbf{Y}$ is the neuron output. The numerical values assigned to the parameters are reported in Table V

n_max	α_F	α_L	α_T	V_F	V_L	V_T	M
180	0.32	0.356	0.08	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	15	[0.1409, 0, 0.1409]

Table V
PARAMETERS OF THE PCNN

D. Neural activity of a PCNN neuron

Figure 7 shows the the values of the FI, LI, internal activity, dynamic threshold and output of a PCNN neuron as a function of the iterations using a toy input consisting of a signal with 7 samples: [0, 0, 0, 0.8, 0.4, 0.2, 0]. The neuron whose activity is described in the figure corresponds to the sample with the value 0.4. The outputs of the neighbouring neurons are also reported. It is interesting to observe how the neurons fire together at n=0, and then a periodic pattern of the firing is achieved. The time intervals at which each neuron fires depends on the signal: for example, the neuron corresponding to the first sample where a signal is observed has greater firing rate than the neuron studied in the figure. It is also possible to appreciate how the surrounding neurons have a role in increasing the internal activity thus enhancing the probability of firing.

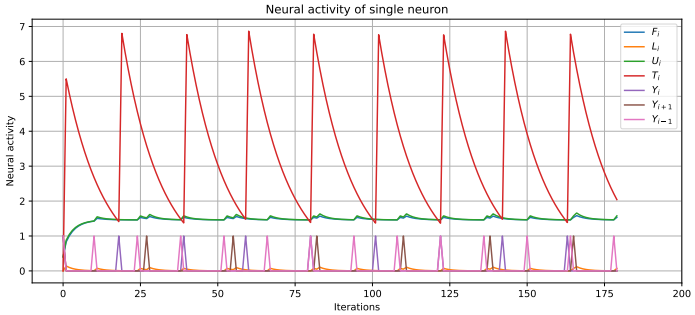

Figure 7. FI (blue), LI (orange), internal activity (green), dynamic threshold (red) and output (purple) of a PCNN neuron as a function of the iterations using a toy input. The outputs of the neighbouring neurons are also shown in brown and pink

E. CNN layers features

The output shape and number of parameters for the CNN model layers is reported in Table VI.

Layer name	Output shape	# parameters
Conv1D_1	(235,4)	28
MaxPooling1D_1	(117,4)	0
Conv1D_2	(112,8)	200
MaxPooling1D_2	(56,8)	0
Flatten_1	448	0
Dense_1	34	15266
Dense_2	1	35

Table VI
CNN LAYERS STRUCTURE

The convolutional layers output is a matrix whose first dimension is equal to the feature map dimension obtained from a filter with size k, given by $n-k+1$ with n being the dimension of the input in the case of zero padding and stride equal to one. For the first convolutional layer results $240-6+1=235$, for the second one $117-6+1=112$. The second dimension is equal to the number of filters considered for the layer, so 4 and 8 for the first and second layers respectively. Pooling layers with filter size m and stride s are used to reduce the output width d to $(d-m+1)/s$, leading to an output size of $(235-2+1)/2=117$ for the first layer and $(112-2+1)/2=56$ for the second layer. The second dimension is left unchanged. The flatten layer conveys a $a \times b$ matrix into a 1D column vector with $a \cdot b$ entries, in this case $56 \cdot 8 = 448$. Finally, dense layers return a vector of dimension equal to the number of neurons in the layer.

The number of parameters in the i-th convolutional layer is calculated multiplying the number of filters in the layer f_i by the total entries of the filter matrix incremented by one to include the bias, $f_i \cdot (k_i \cdot h_{i-1} + 1)$ with k_i filters dimension and h_i second output dimension for the i-th layer. In this case

for the first layer $4 \cdot (6 \cdot 1 + 1) = 28$, for the second layer $8 \cdot (6 \cdot 4 + 1) = 200$. The number of parameters for the i-th dense layer is obtained multiplying $l_i \cdot (l_{i-1} + 1)$, with l_i being the number of neurons of the i-th layer. In this case for the first layer $34 \cdot (448 + 1) = 15266$, for the second layer $1 \cdot (34 + 1) = 35$.

F. Evaluation of computational cost for complete CNN model

In this section the computational cost to obtain a PSD discrimination is evaluated for the complete CNN model by calculating the number of operations executed.

The i-th convolutional layer performs $f_i \cdot (n_i - k_i + 1)$ matrix multiplications each one consisting in $k_i + 1$ operations, with n_i being the dimension of the input, k_i the filter size and f_i the number of filters for the i-th layer. Moreover, an activation function is applied to each entry of the obtained feature map, increasing the number of operations by a factor $f_i \cdot (n_i - k_i + 1)$ The total number of operations executed by a convolutional layer is then $f_i \cdot (n_i - k_i + 1) \cdot (k_i + 1) + f_i \cdot (n_i + k_i - 1) = f_i \cdot (n_i - k_i + 1) \cdot (k_i + 2)$.

The j-th neuron in the i-th dense layer performs $l_{i-1} + 3$ operations with l_i being the number of neurons in the i-th layer. l_{i-1} represents the number of multiplications between input vector and neuron weight vector entries. Moreover, the neuron has to perform 3 operations: sum all products, add the bias and evaluate its activation function. The total operations executed in the i-th dense layer is then $\sum_j (l_{i-1} + 3) = l_i \cdot (l_{i-1} + 3)$.

A max pooling layer with filter size m and moving with stride s receiving an input vector of dimension d executes $[(d - m)/s] + 1$ operations.

The total number of operations O computed by the CNN model in order to predict a label for an input vector having q entries is then:

$$\begin{aligned} O = & [f_1 \cdot (q - k_1 + 1) \cdot (k_1 + 2)] + \\ & + [f_1 \cdot (\frac{d_2 - m_2}{s_2} + 1)] + \\ & + [f_3 \cdot (n_3 - k_3 + 1) \cdot (k_3 + 2)] + \\ & + [f_3 \cdot (\frac{d_4 - m_4}{s_4} + 1)] + \\ & + [l_6 \cdot (l_5 + 3)] + \\ & + [l_7 \cdot (l_6 + 3)] \end{aligned} \tag{6}$$

Layers are labelled with numeric indexes ranging from 1 to 7. For our case $q = 240$, $k_1 = k_3 = 6$, $f_1 = 4$, $f_3 = 8$, $s_2 = s_4 = 2$, $m_2 = m_4 = 2$, $d_2 = 235$, $d_4 = 112$ $l_5 = 448$, $l_6 = 34$, $l_7 = 1$. The total number of operations performed by the network to produce an output is:

$$O = 7520 + 470 + 7168 + 448 + 15334 + 37 = 30977 \tag{7}$$

The formula for O applies to all the networks with the same layer structure as the 1D CNN model considered in this work. To evaluate the impact of pruning on the number of operations, one can use the same formula given in this section considering only non-zero weights, individuated and then ignored by the Vivado HLS package.

G. Internal architecture of the DSP48E Digital Signal Processor

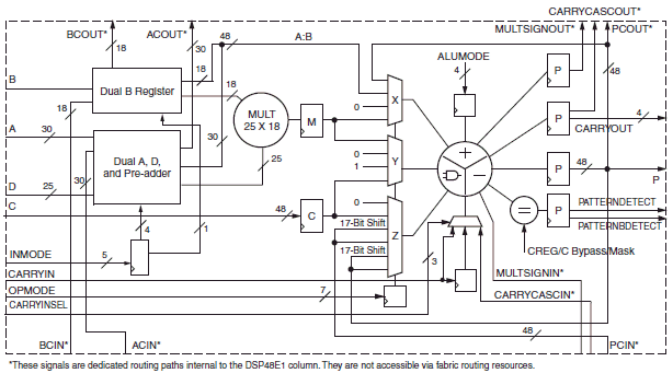

Figure 8. Internal architecture of the DSP48E Digital Signal Processor. The Zynq XC7Z020 contains 220 DSPs. The most important elements are the pre-adder, the 25 bit x 18 bit fixed-point multiplier and the programmable operation that can perform addition, subtraction or pattern recognition. This block is highly efficient in computing the dot product involved in forward propagation through the Neural Network.

VI. REFERENCES

REFERENCES

[1] R. Eckhorn et al. "Feature Linking via Synchronization among Distributed Assemblies: Simulations of Results from Cat Visual Cortex". In: *Neural Computation* 2.3 (Sept. 1990), pp. 293–307. DOI: 10.1162/neco.1990.2.3.293. URL: https://doi.org/10.1162/neco.1990.2.3.293.

[2] Laurens van der Maaten and Geoffrey Hinton. "Viualizing data using t-SNE". In: *Journal of Machine Learning Research* 9 (Nov. 2008), pp. 2579–2605.

[3] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[4] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[5] Javier Duarte et al. "Fast inference of deep neural networks in FPGAs for particle physics". In: *JINST* 13.07 (2018), P07027. DOI: 10.1088/1748-0221/13/07/P07027. arXiv: 1804.06913 [physics.ins-det].

[6] Lisha Li et al. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*. 2018. arXiv: 1603.06560 [cs.LG].

[7] Tom O'Malley et al. *KerasTuner*. https://github.com/keras-team/keras-tuner. 2019.

[8] Hao-Ran Liu et al. "Discrimination of neutrons and gamma rays in plastic scintillator based on pulse-coupled neural network". In: *Nuclear Science and Techniques* 32.8 (Aug. 2021). DOI: 10.1007/s41365-021-00915-w. URL: https://doi.org/10.1007/s41365-021-00915-w.

[9] *AMD Adaptive Computing Documentation Portal*. URL: https://docs.xilinx.com/r/2021.2-English/ug1483-model-composer-sys-gen-user-guide/DSP48E (visited on 07/04/2023).