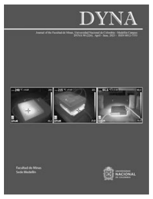
DYNA
http://dyna.medellin.unal.edu.co/



UNIVERSIDAD NACIONAL DE COLOMBIA

Support vector machines implementation over integers modulo-M and Residue Number System

Sergio Andrés Arenas-Hoyos & Álvaro Bernal-Noreña

Escuela de Ingeniería Eléctrica y Electrónica, Universidad del Valle, Santiago de Cali, Colombia. sergio.arenas@correounivalle.edu.co, alvaro.bernal@correounivalle.edu.co

Received: February 3rd, 2023. Received in revised form: April 27th, 2023. Accepted: May 11th, 2023

Abstract
In low-power hardware implementations for classification algorithms, it is often essential to use physical resources efficiently. In this sense, the use of modulo-M integer operations instead of floating-point arithmetic, can lead to better performance, especially when M represents the dynamic range of an arithmetic block of the Residue Number System (RNS) [1,2]. Following this premise, this work is aiming to provide a methodology for implementing a classifier, specifically a Support Vector Machine (SVM) [3], using modulo-M integers and proposing a method for the use of Residue Number System.

Keywords: modular arithmetic; pattern recognition; Residue Number System (RNS); Support Vector Machines (SVN); digital signal processing; radial basis function.

Implementación de máquinas de vectores de soporte sobre enteros módulo-M y en el Sistema Numérico de los Residuos

Resumen
En las implementaciones en hardware de baja potencia para algoritmos de clasificación, a menudo es esencial utilizar los recursos físicos de manera eficiente. En este sentido, la utilización de operaciones con enteros módulo-M en lugar de aritmética de punto flotante puede conducir a un mejor rendimiento, especialmente cuando M representa el rango dinámico de un bloque aritmético del Sistema Numérico de los Residuos (RNS) [1,2]. Siguiendo esta premisa, el objetivo de este trabajo es proporcionar una metodología para implementar un clasificador, en concreto, una Máquina de Vectores de Soporte (SVM) [3], utilizando enteros módulo-M y proponer un método para la utilización del Sistema Número de Residuos.

Palabras clave: aritmética modular; reconocimiento de patrones; Sistema Numérico de los Residuos (SNR); Máquinas de Vectores de Soporte (MVS); procesamiento digital de señales; funciones de base radial.

1 1. Introduction

Supervised learning techniques are commonly used in Machine Learning applications to design specific classifiers that can solve pattern recognition problems. One such classifier is the Support Vector Machine (SVM), which aims to find a decision surface that maximizes the margin between two distinct groups in binary categorization [3,4].

In embedded systems, hardware limitations can pose restrictions on implementing sophisticated algorithms. However, methods such as modular arithmetic can assist in overcoming these limitations by leading to less complex functional units compared to floating-point arithmetic. One specific approach to utilizing modular arithmetic in parallel hardware operations is the Residue Number System (RNS), which enables the representation of large numbers using a set of smaller numbers [5].

Although the RNS exhibits fast operations such as addition, subtraction, and multiplication, it poses challenges when dealing with non-linear operations like comparison and division [6]. Overcoming this limitation requires a specific approach: first, developing a method to represent input

Revista DYNA, 90(226), pp. 17-26, April - June, 2023, ISSN 0012-7353
DOI: https://doi.org/10.15446/dyna.v90n226.107112

features using integers modulo-M; second, mapping non-linear functions such as the Kernel in SVM to integers modulo-M; and finally, utilizing the model to design an equivalent RNS-based implementation of SVM (SVM-RNS).

By leveraging the SVM-RNS equivalent as a foundational component in pattern recognition, it is indeed feasible to explore more complex architectures for multi-class classifiers in subjects such as surface electromyographic (sEMG) signal analysis [7,8].

2 Support Vector Machines (SVM)

In SVM classifiers, the surface decisions aim to the find optimal margin, which involves maximizing the minimum margin between two classes.

This problem can be formulated in terms of optimization using dual problem formulation:

$$maxL(\lambda) = max\sum_{i=1}^{N}\lambda_i - \frac{1}{2}.\sum_{i,j}^{N}\lambda_i.\lambda_j.l_i.l_j.K\langle\vec{x_i},\vec{x_j}\rangle$$

$$s.t: 0 \leq \lambda_i \leq C; \sum_{i=1}^{P}\lambda_i\, l_i = 0 \tag{1}$$

Here, each λ corresponds to Lagrange Multipliers, K represents the Kernel function, C is the maximum bound for λ, x represents an input feature from training set, and l is the class label. Numerical optimization methods such as a simplified version of Sequential Minimal Optimization (SMO) algorithm [9] can be employed to obtain the Lagrange Multipliers and Support Vectors (S.V), which form a hyperplane:

$$g(\vec{x_j}) = \sum_{i\in S.V}^{T}\lambda_i.l_i.K\langle\vec{x_i},\vec{x_j}\rangle + b \tag{2}$$

The Kernel function is a powerful tool that can be used to transform non-separable categories into separable ones. One well-known example is the Gaussian Kernel, which can map finite-dimensional features onto a space characterized by an infinite number of dimensions [10]. This transformation is achieved by calculating the Euclidean distance between the Support Vectors x_i, which involves performing inner product operations:

$$K\langle\vec{x_i},\vec{x_j}\rangle = e^{-\gamma\|\vec{x_i}-\vec{x_j}\|^2} \rightarrow 0 < K\langle\vec{x_i},\vec{x_j}\rangle \leq 1 \tag{3}$$

After training stage, the Lagrange Multipliers lie within the interval:

$$0 \leq \lambda_i \leq \lambda_{MAX} \tag{4}$$

The class labels are defined as 1 for Class 1 and -1 for Class 2, indicating that labels belong to the set {-1,1}:

$$l_i \in \{-1,1\} \tag{5}$$

For classification purposes, the hyperplane g is utilized as the input for the sign function:

$$g(\vec{x_j}) = \begin{cases} \geq 0 \rightarrow \vec{x_j} \in Class1 \\ 0 \rightarrow \vec{x_j} \in Class2 \end{cases} \tag{6}$$

3 Residue Number Systems (RNS)

In hardware implementations, several techniques are available to execute operations in parallel based on modular arithmetic [11]. For instance, considering an integer number X that can be decomposed into a set of residues $\{x_1, x_2, \ldots, x_n\}$ with respect to moduli set $\{m_1, m_2, \ldots, m_n\}$. It is essential to note that each modulus m_i is coprime with each other modulus m_j. Similarly, when this procedure is applied to an integer Y, the following system can be defined:

$$\begin{gathered} X \rightarrow \hat{X} = \langle x_1, x_2, \cdots, x_n\rangle \\ Y \rightarrow \hat{Y} = \langle y_1, y_2, \cdots, y_n\rangle \\ x_i = Xmodm_i \\ y_i = Ymodm_i \\ \hat{X}\circ\hat{Y} = \langle x_1\circ y_1, x_2\circ y_2, \cdots, x_n\circ y_n\rangle \\ M = \prod_{i=1}^{n} m_i \end{gathered} \tag{7}$$

Where ◦ denotes any operator from the set {+, -, *}, and M represents the Dynamic Range of the moduli set, which defines the number of possible representations in the ring [9]. Furthermore, the RNS is specified in this work under the following constraints:

$$\begin{gathered} modset = \{m_3, m_2, m_1\} = \{2^\beta - 1, 2^\alpha, 2^\beta + 1\} \\ M = m_1.m_2.m_3 = 2^{2\beta+\alpha} - 2^\alpha \\ M\ is\ even \end{gathered} \tag{8}$$

Each integer X represented in this finite field must satisfy the following condition:

$$\frac{-M}{2} \leq X \leq \frac{M}{2} - 1 \tag{9}$$

This main constraint is thoroughly analyzed in this work as it significantly influences every aspect of the SVM, as discussed in the following sections.

4 Implementation constraints over integers

Recalling the previous analysis, it can be inferred that the SVM hyperplane satisfies the following equation:

$$\begin{gathered} G_1.g(\vec{x_j}) = \begin{cases} \geq 0 \rightarrow \vec{x_j} \in Class1 \\ 0 \rightarrow \vec{x_j} \in Class2 \end{cases} \\ G_1 > 0, G_1 isaconstant \end{gathered} \tag{10}$$

Where G_1 represents a gain that can assists in mapping $g(x_j)$ within a limited scale. In the case of integer implementation, there are more specific restrictions for each part of the hyperplane. Firstly, finite fields only allow the representation of a subset P composed of integers:

$$P = \left\{\frac{-M}{2}, \cdots, \frac{M}{2} - 1\right\} \quad (11)$$
$$P \subseteq Z$$

This implies that an equivalent representation in the subset P must be obtained for every component of the SVM, including Lagrange Multipliers, feature vectors, Kernel function, class labels and comparator function.

5 Proposed Kernel function mapping over integers modulo-M

The Kernel function is an essential component of Support Vector Machines (SVMs) which maps features into a space with higher dimensions. In Radial Basis Functions (RBFs), the distance between two finite-dimensional vectors is measured, forming an inner product between two infinite-dimensional vectors. This approach enhances separability, and by utilizing the *Kernel Trick* [10] in SVM with RBF Kernel, the hyperplane can be calculated without explicitly computing the infinite-dimensional inner product. The calculations can be performed using the following equation, which proposes a mapping over integers modulo-M:

$$K\langle \vec{x_i}, \vec{x_j}\rangle = e^{-\gamma \|\vec{x_i} - \vec{x_j}\|^2} \quad (12)$$

Where γ is a positive constant parameter. The exponential function delimits the range of the function to:

$$0 < K\langle \vec{x_i}, \vec{x_j}\rangle \leq 1 \quad (13)$$

This function can be interpreted as being dependent on the distance d:

$$K\langle \vec{x_i}, \vec{x_j}\rangle = z(d) = e^{-\gamma d^2} \quad (14)$$
$$d^2 = \|\vec{x_i} - \vec{x_j}\|^2$$

Based on this, the behavior of function z is illustrated in Fig.1.

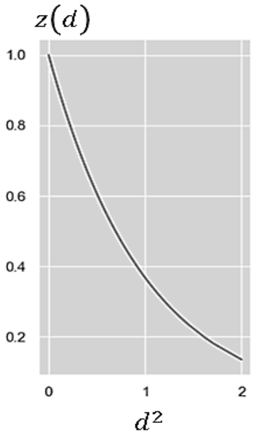


Figure 1. RBF Kernel function
Source: The authors

This function has its domain and range in real numbers, but they can be mapped into integers by determining their respective bounds. The first step is to find the bounds of the domain. Essentially, d^2 represents the Euclidean distance between two k-dimensional vectors x_i and x_j:

$$\vec{x_i} = [x_{i0}x_{i1}x_{i2} \cdots x_{i(k-1)}] \quad (15)$$
$$\vec{x_j} = [x_{j0}x_{j1}x_{j2} \cdots x_{j(k-1)}]$$
$$d^2 = \sum_{m=0}^{k-1} (x_I - x_{jm})^2$$

If the x vectors are normalized, their components must be within in interval [0,1]. Under this assumption, the maximum distance between each pair of components is 1. Therefore, it can be inferred that the maximum distance between vectors is:

$$d^2_{MAX} = \sum_{m=0}^{k-1} (1)^2 = k \quad (16)$$

This implies that:

$$0 \leq d^2 \leq k \quad (17)$$
$$0 \leq \left(\frac{d^2}{k}\right) \leq 1$$

Taking this inequality into account, and defining the mapping function as $\widehat{d^2}$:

$$Data_{max} = \frac{M}{2} - 1 \quad (18)$$
$$\widehat{d^2} = \frac{Data_{max}}{k} . d^2$$

Then:

$$0 \leq \widehat{d^2} \leq Data_{max} \quad (19)$$

This last inequality is useful for achieving the mapping of d^2 to integers, indicating that:

$$\widehat{d^2} = \frac{Data_{max}}{k} . d^2 = \sqrt{\frac{Data_{max}}{k}} . (\vec{x_i} - \vec{x_j})^T . \sqrt{\frac{Data_{max}}{k}} . (\vec{x_i} - \vec{x_j}) \quad (20)$$

By utilizing the floor function to approximate the mapped distance, the following relationship is obtained:

$$\sqrt{\frac{Data_{max}}{k}} . (\vec{x_i} - \vec{x_j}) \approx \left\lfloor \sqrt{\frac{Data_{max}}{k}} . \vec{x_i} \right\rfloor - \left\lfloor \sqrt{\frac{Data_{max}}{k}} . \vec{x_j} \right\rfloor \quad (21)$$

This approximation provides a method for converting a normalized feature vector into its integer equivalent:

$$\hat{\vec{x}} = \left\lfloor \sqrt{\frac{Data_{max}}{k}} . \vec{x} \right\rfloor \quad (22)$$

Where M represents the dynamic range of the RNS and k is the number of dimensions in feature vectors. This analysis provides an integer equivalent for the distance, and the next step is to convert γ and the function z(d) to integers.

To convert γ into an equivalent integer, it is necessary to compensate for the effect of $\frac{Data_{max}}{k}$ in the exponent. Therefore, it is advisable to use the following mapping function:

$$\hat{\gamma} = \left\lfloor \frac{k}{Data_{max}} . \gamma \right\rfloor \tag{23}$$

These approximations lead to the following model for the exponential function:

$$z(d) \approx e^{-\hat{\gamma}\widehat{d^2}} \tag{24}$$

Finally, to adjust the RBF kernel, the following integer-based equivalent is proposed:

$$\hat{z}(\hat{d}) = \left\lfloor \left(\frac{Data_{max}}{k.G_e}\right) . e^{-\hat{\gamma}\widehat{d^2}} \right\rfloor \tag{25}$$
$$G_e \geq 1$$
$$\hat{z}(\hat{d}) \approx \left(\frac{Data_{max}}{k.G_e}\right) z(d)$$

In this case, $\hat{z}(\hat{d})$ represents a function with a range contained in the interval $\left[0, \left(\frac{Data_{max}}{k.G_e}\right)\right]$, which is a valid subset within the finite field. G_e is a positive constant gain that allows adjusting the amplitude of the exponential function according to the subsequent mapping of the SVM hyperplane.

For example, if M = 510, G_e = 5 and k=2, the integer-based exponential equivalent will have the following bounds:

$$Amp_e = \left(\frac{Data_{max}}{k.G_e}\right) = \frac{254}{2.5} = 25{,}4 \tag{26}$$
$$\lfloor Amp_e \rfloor = 25$$
$$\widehat{d^2}_{MAX} = \lfloor Data_{max} \rfloor = \left\lfloor \frac{510}{2} - 1 \right\rfloor = \lfloor 254 \rfloor$$
$$\widehat{d^2}_{MAX} = 254$$

Thus, the integer exponential function will produce values within the range [0,25], which is within the valid subset of the finite field P.

This relationship is illustrated in Fig. 2.

The kernel function is typically stored in a Look-Up Table (LUT) or an equivalent form to simplify its approximation. Although other methods, such as using Taylor Series for finite fields, exist, they can be challenging to implement. One major issue with the Taylor Series approach is the difficulty of finding a modular multiplicative inverse for the factorial function, which is needed to calculate the series coefficient. If the modulo is not a prime number and is not coprime with all Taylor coefficients, the existence of a modular multiplicative inverse cannot be guaranteed [12]. In the case of a composite number like M, it is not certain that all the Taylor Series coefficients will be available.

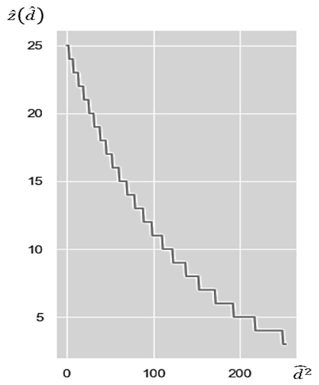


Figure 2. Integer exponential equivalent.
Source: The authors

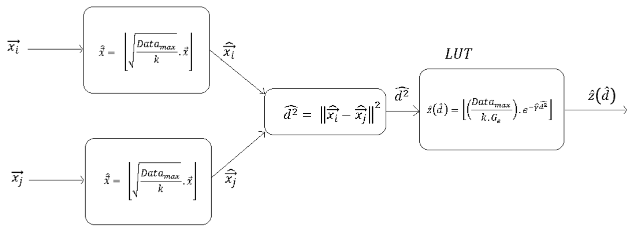

$\overrightarrow{x_i}$
$\hat{\vec{x}} = \left\lfloor \sqrt{\frac{Data_{max}}{k}} . \vec{x} \right\rfloor$
$\widehat{\overrightarrow{x_i}}$
$\overrightarrow{x_j}$
$\hat{\vec{x}} = \left\lfloor \sqrt{\frac{Data_{max}}{k}} . \vec{x} \right\rfloor$
$\widehat{\overrightarrow{x_j}}$
$\widehat{d^2} = \left\| \widehat{\overrightarrow{x_i}} - \widehat{\overrightarrow{x_j}} \right\|^2$
$\widehat{d^2}$
LUT
$\hat{z}(\hat{d}) = \left\lfloor \left(\frac{Data_{max}}{k.G_e}\right) . e^{-\hat{\gamma}\widehat{d^2}} \right\rfloor$
$\hat{z}(\hat{d})$

Figure 3. Block diagram for integer exponential approximation.
Source: The authors

To visualize the integer exponential approximation, a block diagram is provided in Fig. 3.

6 Proposed support vector machine mapping over integers modulo-M

Building on the analysis presented in the previous sections, it is feasible to map the SVM hyperplane into integers by mapping all its components individually. Starting with the Kernel function represented in the modulo-M finite field, similar steps can be taken to convert the other parameters. A summary of these conversions is provided in Table 1.

Table 1.
Conversion functions summary.

Parameter	Range	Conversion Function	Conversion Function Range
Feature vector $\vec{x}$	$0 \leq \vec{x_i} \leq 1$	$\hat{\vec{x}} = \left\lfloor \sqrt{\frac{Data_{max}}{k}} . \vec{x} \right\rfloor$	$0 \leq \hat{\vec{x}}_i \leq \lfloor \sqrt{\frac{Data_{max}}{k}} . \vec{x} \rfloor$
Lagrange Multipliers λ_i	$0 \leq \lambda_i \leq \lambda_{MAX}$	$\hat{\lambda}_i = \left\lfloor G_{lambda} . \frac{\lambda_i}{\lambda_{MAX}} \right\rfloor$	$0 \leq \hat{\lambda}_i \leq G_{lambda}$
Gamma parameter γ	$\gamma > 0$	$\hat{\gamma} = \lfloor \frac{k}{Data_{max}} . \gamma \rfloor$	$\hat{\gamma} > 0$
Class labels l_i	$l_i \in \{-1,1\}$	$\hat{l}_i = l_i$	$\hat{l}_i \in \{-1,1\}$
Bias b	$b > 0$	$\hat{b} = \lfloor G_{bias} . b \rfloor$	$\hat{b} > 0$

Source: The authors.

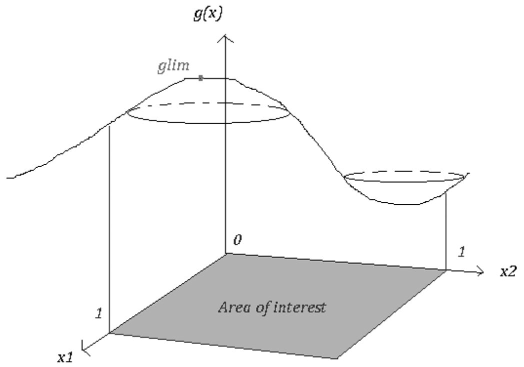


Figure 4. Scheme for two-dimensional feature space and its hyperplane g.
Source: The authors

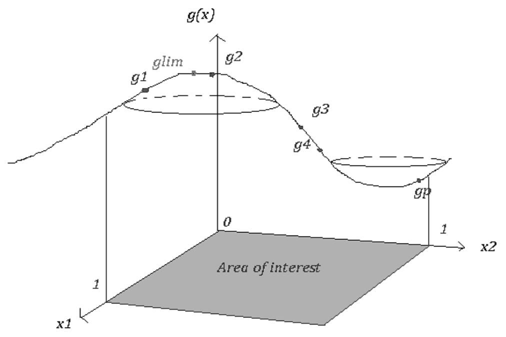


Figure 5. Scheme for g_{lim} estimation.
Source: The authors

By utilizing these conversion functions, it becomes possible to obtain the approximated bound of the hyperplane, referred as g_{lim}. This parameter represents the maximum value of g within a specific area of interest, which corresponds to the vector space encompassing all possible feature vectors. In the case of a two-dimensional normalized feature, for instance, the area of interest is a square defined within the range of [0,1] in both dimensions, as depicted in Fig. 4.

To approximate the value of g_{lim}, the absolute value of g can be evaluated at q randomly selected points within the area of interest. By comparing these evaluations, the maximum value can be determined, yielding the value of g_{lim}:

$$\begin{gathered} g_{lim} = max\{|g_i|\} \\ 0 \leq i \leq q \end{gathered} \tag{27}$$

While using numerical analysis techniques such as SMO (Sequential Minimal Optimization) or genetic algorithms could potentially provide a more precise estimation of g_{lim},for the sake of simplicity, this work utilizes the q random points method, as depicted in Fig. 5. Although it may not guarantee an optimal solution, it offers a practical and straightforward approach for approximating g_{lim}.

Looking back on the fact that the hyperplane can be multiplied by a positive gain G_1 without altering its sign, one can utilize the following inequality:

$$|\{g(\vec{x_j})\}| \leq g_{\mathrm{lim}} \tag{28}$$

Then:

$$\begin{gathered} g(\hat{\vec{x_j}}) = G_{plane}.\frac{g(\vec{x_j})}{g_{lim}} \\ G_1 = \frac{G_{plane}}{g_{lim}} \\ -G_{plane} \leq g(\hat{\vec{x_j}}) \leq G_{plane} \end{gathered} \tag{29}$$

In principle, G_{plane} can be used to map g into the range $\left[\frac{-Data_{max}}{k}, \frac{Data_{max}}{k}\right]$. However, considering the raw approximation in g_{lim}, a slack of 20% is applied in G_{plane}, which corresponds to the 80% of the total range:

$$G_{plane} = 0{,}8.\frac{Data_{max}}{k} \tag{30}$$

Nevertheless, G_{lambda} is a parameter that scales the Lagrange Multipliers in hyperplane function. If the Lagrange Multipliers are significantly smaller compared to the range $\left[\frac{-Data_{max}}{k}, \frac{Data_{max}}{k}\right]$, G_{lambda} can be defined as a small portion of G_{plane}:

$$G_{lambda} = 0{,}01.G_{plane} \tag{31}$$

Next, combining all the components of the SVM, the integer SVM can be expressed as follows:

$$g(\hat{\vec{x_j}}) = \sum_{i \in S.V}^{T} \hat{\lambda_i}.\hat{l_i}.\hat{z_{ij}} + \hat{b} \tag{32}$$

Then, replacing gains:

$$\begin{gathered} \frac{G_{plane}}{g_{\mathrm{lim}}} \\ g(\vec{x_j}) = \sum_{i \in S.V}^{T} \lambda_i.l_i.K\langle\vec{x_i},\vec{x_j}\rangle.G_{lambda}.\left(\frac{Data_{max}}{\lambda_{MAX.}.k.G_e}\right) \\ + G_b.b \end{gathered} \tag{33}$$

From this model, the system gains can be determined based on the following conditions:

$$\begin{gathered} G_b = \frac{G_{plane}}{g_{\mathrm{lim}}} \\ G_{lambda}.\left(\frac{Data_{max}}{\lambda_{MAX.}.k.G_e}\right) = \frac{G_{plane}}{g_{lim}} \end{gathered} \tag{34}$$

7 Proposed SVM model using Residue Number Systems

Based on the previous approaches, a new objective can be achieved: constructing an SVM classifier using RNS when the SVM is modeled in a modulo-M ring. The initial step in this approach is to select a suitable set of moduli. Using moduli that are powers of two is often a suitable choice to simplify the hardware implementation:

Table 2.
Parameters conversion between modulo-M and RNS residues.

Model	Integer representation	RNS representation
Lagrange multiplier in modulo-M	$\hat{\lambda}_\iota$	$\langle\hat{\lambda}_\iota\rangle_{m_i}$
Class label in modulo-M	$\hat{t}_\iota$	$\langle\hat{t}_\iota\rangle_{m_i}$
Integer exponential in modulo-M	$\hat{z}_{\iota J}$	$\langle\hat{z}_{\iota J}\rangle_{m_i}$

Source: The authors.

$$\begin{gathered} modset = \{m_3, m_2, m_1\} = \{2^\beta - 1, 2^\alpha, 2^\beta + 1\} \\ M = m_1.m_2.m_3 = 2^{2\beta+\alpha} - 2^\alpha \end{gathered} \tag{35}$$

RNS residues for SVM parameters in the modulo-M system can be obtained by applying the conversions outlined in Table 2.

In Table 2, for each parameter p, there is a corresponding 3-tuple where each component is defined as follows:

$$\begin{gathered} \langle p_i\rangle_{m_i} = p_i mod m_i \\ i \in \{1,2,3\} \end{gathered} \tag{36}$$

When using a Look-Up Table with $\lfloor\frac{M}{2}\rfloor x\lfloor\frac{M}{2}\rfloor$ dimensions to represent an integer exponential, there will be a corresponding $\lfloor\frac{M}{2}\rfloor x\lfloor\frac{M}{2}\rfloor x3$ size matrix in RNS representation.

With this parameter conversion, it becomes suitable to create an equivalent hyperplane in RNS. However, it is necessary to define a new function in RNS that can replace the sign function:

$$sign\{\hat{g}\} \rightarrow sign_{RNS}\{\hat{g}\} \tag{37}$$

Implementing the sign function, which involves comparing a number with zero to determine whether it is less than, greater than, or equal to zero, is not a straightforward task in RNS. However, a variant of the LPN algorithm [13] that is specifically designed for the chosen modulus set can be utilized to accomplish the goal of performing the comparison in RNS.

The first step in this process is to represent an integer number in modulo-M, denoted as X, and its equivalent in RNS, denoted as $\langle R_1, R_2, R_3\rangle$.The relationship between them can be expressed as follows:

$$\begin{gathered} X \rightarrow \langle R_1, R_2, R_3\rangle \\ \langle X\rangle_{2^\beta-1} = Xmod(2^\beta - 1) = R_1 \\ \langle X\rangle_{2^\alpha} = Xmod(2^\alpha) = R_2 \\ \langle X\rangle_{2^\beta+1} = Xmod(2^\beta + 1) = R_3 \end{gathered} \tag{38}$$

A periodicity is evident in the set of residues with respect to the m1 and m3 moduli. This can be observed by defining:

$$\begin{gathered} X' = X + m_1.m_3 = X + (2^\beta - 1).(2^\beta + 1) \\ \langle X'\rangle_{2^\beta-1} = X'mod(2^\beta - 1) = R_1'R_1 \\ \langle X'\rangle_{2^\beta+1} = X'mod(2^\beta + 1) = R_3'R_3 \end{gathered} \tag{39}$$

$$\langle X'\rangle_{2^\alpha} = X'mod(2^\alpha) = R_2'$$

The last residue in the set can be defined as:

$$\begin{aligned} \langle X'\rangle_{2^\alpha} &= \langle X + (2^\beta - 1).(2^\beta + 1)\rangle_{2^\alpha} \\ &= \langle X + (-1).(+1)\rangle_{2^\alpha} = \langle X - 1\rangle_{2^\alpha} \\ R_2' &= \langle R_2 - 1\rangle_{2^\alpha} \end{aligned} \tag{40}$$

By generalizing and assuming a period T, where $T = m_1.m_3$, it can be observed that if multiple periods of length kT are added or subtracted to an integer number X, the residue R2 will be affected in k units:

$$\begin{gathered} X' = X \pm k.T \rightarrow \begin{cases} R_1' = R_1 \\ R_2' = \langle R_2 \mp k\rangle_{2^\alpha} \\ R_3' = R_3 \end{cases} \\ T = m_1.m_3 \\ k \in N \end{gathered} \tag{41}$$

To compare numbers in RNS, periodicity can be leveraged by defining the following terms:

- Least Possible Number (LPN): this is the smallest integer whose residues R1 and R3 match those of the number being represented.
- Reference Residue (RR): this is the R2 residue of LPN.

Representing an integer in RNS using a modulus set can be thought of as a linear combination of its LPN (LPNx) and RR (RRx). This is illustrated by considering two integers, X and X', and their corresponding RNS residues:

$$\begin{gathered} X \rightarrow \langle R_1, R_2, R_3\rangle \\ X' \rightarrow \langle R_1', R_2', R_3'\rangle \end{gathered} \tag{42}$$

Now, X' can be defined as a linear combination of LPNx, R2 and RRx:

$$\begin{aligned} X' &= m_1.m_3.(RR_x - R_2) + LPN_x \\ &= (2^\beta - 1).(2^\beta + 1).(RR_x - R_2) \\ &+ LPN_x \end{aligned} \tag{43}$$

By means of modular arithmetic, it can be concluded that X is equivalent to X' in RNS representation:

$$\begin{gathered} \langle R_1'\rangle = \langle X'\rangle_{2^\beta-1} = \langle LPN_x\rangle_{2^\beta-1} = R_1 \\ \langle R_2'\rangle = \langle X'\rangle_{2^\alpha} = \langle R_2 - RR_x + LPN_x\rangle_{2^\alpha} \\ = \langle R_2 - RR_x + RR_x\rangle_{2^\alpha} = R_2 \\ \langle R_3'\rangle = \langle X'\rangle_{2^\beta+1} = \langle LPN_x\rangle_{2^\beta+1} = R_3 \\ \rightarrow X' = X \\ X = k.m_1.m_3 + r = m_1.m_3.(RR_x - R_2) + LPN_x \\ \rightarrow \langle X\rangle_{m_1.m_3} = \langle LPN_x\rangle_{m_1.m_3} \end{gathered} \tag{44}$$

Since LPNx is within the first period T = m1.m3, it follows that:

$$\langle X\rangle_{m_1.m_3} = LPN_x \tag{45}$$

Then, an algorithm based on these principles can be utilized to compare two integer numbers, denoted as X and Y, represented in RNS:

$$X = k_x . m_1 . m_3 + r_x \rightarrow X \equiv \langle R_{x1}, R_{x2}, R_{x3} \rangle$$
$$Y = k_y . m_1 . m_3 + r_y \rightarrow X \equiv \langle R_{y1}, R_{y2}, R_{y3} \rangle \tag{46}$$

Then:

$$\begin{gathered} if \langle k_x \rangle_{m_2} > \langle k_y \rangle_{m_2} \rightarrow X > Y \\ if \langle k_x \rangle_{m_2} < \langle k_y \rangle_{m_2} \rightarrow X < Y \\ if \langle k_x \rangle_{m_2} = \langle k_y \rangle_{m_2} \begin{cases} r_x > r_y \rightarrow X > Y \\ r_x < r_y \rightarrow X < Y \\ r_x = r_y \rightarrow X = Y \end{cases} \end{gathered} \tag{47}$$

In terms of LPNs and RRs:

$$\begin{gathered} if \langle (RR_x - R_{x2}) \rangle_{m_2} > \langle (RR_y - R_{y2}) \rangle_{m_2} \rightarrow X > Y \\ if \langle (RR_x - R_{x2}) \rangle_{m_2} < \langle (RR_y - R_{y2}) \rangle_{m_2} \rightarrow X < Y \\ if \langle (RR_x - R_{x2}) \rangle_{m_2} = \langle (RR_y - R_{y2}) \rangle_{m_2} \begin{cases} LPN_x > LPN_y \rightarrow X > Y \\ LPN_x < LPN_y \rightarrow X < Y \\ LPN_x = LPN_y \rightarrow X = Y \end{cases} \end{gathered} \tag{48}$$

To simplify the comparison function in RNS, the subtraction of RRs with their corresponding R_2 residues is carried out in modulo m2 to ensure positive values. In terms of hardware implementation, if LPNs and RRs are obtained from a memory-based LUT, the comparison function can be calculated solely using the LUTs and the residues from X and Y. This approach overcomes the main challenge in RNS, which is the conversion from RNS to Binary using methods such as the Chinese Remainder Theorem. The comparison function can be defined as follows:

$$comp_{RNS}\{X,Y\} = \begin{cases} if \langle (RR_x - R_{x2}) \rangle_{m_2} > \langle (RR_y - R_{y2}) \rangle_{m_2} \rightarrow X > Y \\ if \langle (RR_x - R_{x2}) \rangle_{m_2} < \langle (RR_y - R_{y2}) \rangle_{m_2} \rightarrow X < Y \\ if \langle (RR_x - R_{x2}) \rangle_{m_2} = \langle (RR_y - R_{y2}) \rangle_{m_2} \begin{cases} LPN_x > LPN_y \rightarrow X > Y \\ LPN_x < LPN_y \rightarrow X < Y \\ LPN_x = LPN_y \rightarrow X = Y \end{cases} \end{cases} \tag{49}$$

This last function can be implemented efficiently using adders and comparators. In signed RNS, positives numbers are represented in the range $\left[0, \frac{M}{2} - 1\right]$, while negative numbers are represented in the range $\left[\frac{-M}{2}, -1\right]$, or equivalently in the nominal range $\left[\frac{M}{2}, M - 1\right]$. Using complete nominal range $[0, M - 1]$, the decision boundary between positive and negative numbers is $\frac{M}{2}$. Therefore, sign function in RNS can be defined as follows:

$$sign\{X\} \equiv comp_{RNS}\left\{X, \frac{M}{2}\right\} \tag{50}$$

This property enables the use of RNS for representing SVM-RBF classifiers.

8 Results

The previous methods discussed in the sections were implemented to demonstrate the feasibility of SVMs with Radial Basis Function in both modulo-M and RNS representations. The implementation was carried out using the Python programming language along with various libraries such as Numpy, sklearn, and matplotlib. The SVM training method used was a simplified SMO-algorithm-based approach (SMO-lite).

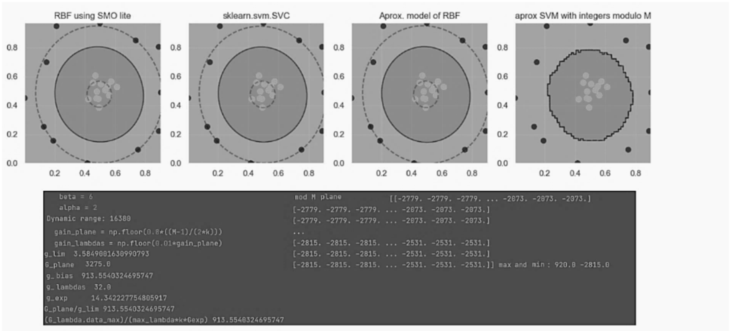
Python proved to be a versatile and widely used platform for implementing SVM-RBF models. The powerful numerical computing library, Numpy, facilitated efficient mathematical operations and array manipulation. The sklearn library provided convenient tools for machine learning, including SVM implementations. Matplotlib was utilized for visualizing the results and analyzing the performance of the models.

As an example, a dataset of 25 two-dimensional feature vectors was analyzed, which were divided into two distinct groups with the main group centered around a cluster. To ensure consistency, all samples were normalized and subjected to Gaussian noise with a standard deviation of 0.1. The Radial Basis Function (RBF) kernel used a γ parameter set to 1.

Figs. 6-8 compare the decision boundaries of the SVM-RBF approximation in modulo-M to the original SMO-lite algorithm, sklearn SVM, and a Maclaurin Series approximation with a ninth-degree polynomial. The effect of changing the dynamic range (M) is also depicted, with α and β parameters influencing the shape of the decision boundaries.

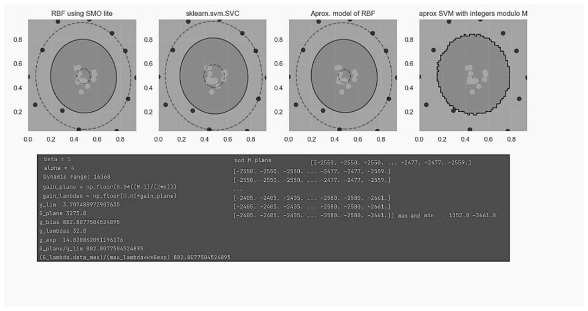
Fig. 9 shows the plot of the sign function in Python using the LPN-method. The plot labels the first half of the nominal range as positive 1, the second half of the nominal range as negative 1, and zero value in the input represents a 0 sign according to the Dirichlet Conditions.

Finally, Fig. 10 compares four previous methods to a fifth SVM approach, known as SVM-RNS, which maps the modulo-M equivalent to the RNS representation. The figure provides a visual comparison of the performance and accuracy of these different methods. The implementation was carried out using the Python programming language and libraries such as Numpy, Scikit-learn, and Matplotlib, which facilitated efficient mathematical operations, machine learning tools, and data visualization.



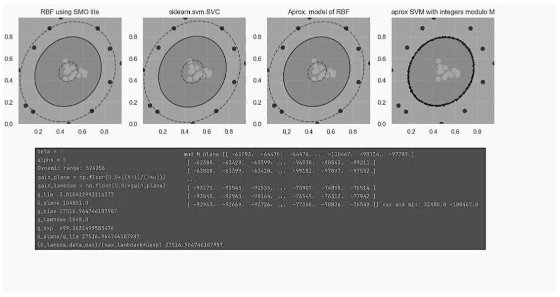
```
   beta = 6                                  mod M plane        [[-2779. -2779. -2779. ... -2073. -2073. -2073.]
   alpha = 2                                 [-2779. -2779. -2779. ... -2073. -2073. -2073.]
 Dynamic range: 16380                        [-2779. -2779. -2779. ... -2073. -2073. -2073.]
  gain_plane = np.floor(0.8*((M-1)/(2*k)))   ...
  gain_lambdas = np.floor(0.01*gain_plane)   [-2815. -2815. -2815. ... -2531. -2531. -2531.]
g_lim  3.5849001630990793                    [-2815. -2815. -2815. ... -2531. -2531. -2531.]
G_plane   3275.0                             [-2815. -2815. -2815. ... -2531. -2531. -2531.]] max and min : 920.0 -2815.0
g_bias  913.5540324695747
g_lambdas  32.0
g_exp      14.342227754805917
G_plane/g_lim 913.5540324695747
(G_lambda.data_max)/(max_lambda*k*Gexp) 913.5540324695747
```

Figure 6. SVM in integers modulo-M using α= 2, β= 6, M = 16380.
Source: The authors.



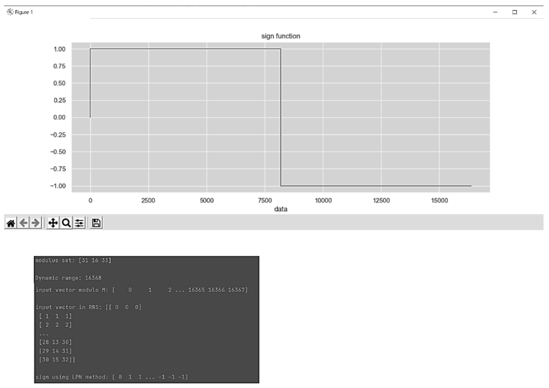
```
 beta = 5                                        mod M plane         [[-2550. -2550. -2550. ... -2477. -2477. -2559.]
 alpha = 4                                       [-2550. -2550. -2550. ... -2477. -2477. -2559.]
 Dynamic range: 16368                            [-2550. -2550. -2550. ... -2477. -2477. -2559.]
 gain_plane = np.floor(0.8*((M-1)/(2*k)))        ...
 gain_lambdas = np.floor(0.01*gain_plane)        [-2405. -2405. -2405. ... -2580. -2580. -2661.]
g_lim  3.707488972907635                         [-2405. -2405. -2405. ... -2580. -2580. -2661.]
G_plane 3273.0                                   [-2405. -2405. -2405. ... -2580. -2580. -2661.]] max and min  : 1152.0 -2661.0
g_bias 882.8077504524895
g_lambdas 32.0
g_exp  14.830862091196176
G_plane/g_lim 882.8077504524895
(G_lambda.data_max)/(max_lambda*k*Gexp) 882.8077504524895
```

Figure 7. SVM in integers modulo-M using α= 4, β= 5, M = 16368.
Source: The authors.



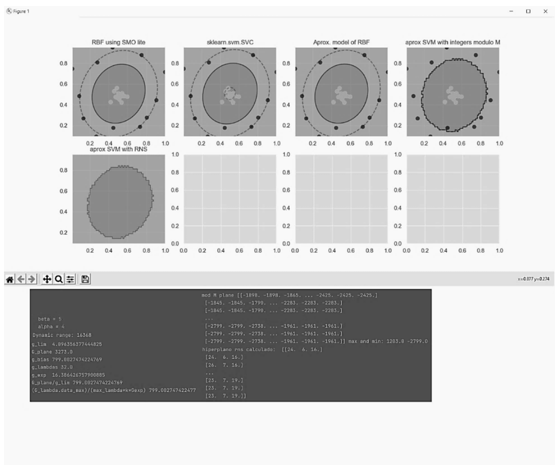
```
beta = 7                                   mod M plane [[ -65093.  -64476.  -64476. ... -100467.  -98134.  -97789.]
alpha = 5                                   [ -62380.  -63428.  -63399. ...  -96038.  -98543.  -99211.]
Dynamic range: 524256                       [ -63808.  -63399.  -63428. ...  -99182.  -97897.  -97552.]
gain_plane = np.floor(0.8*((M-1)/(2*k)))    ...
gain_lambdas = np.floor(0.01*gain_plane)    [ -93171.  -93545.  -91535. ...  -73807.  -74855.  -76514.]
g_lim  3.810412993116377                    [ -93545.  -92963.  -90164. ...  -76549.  -76312.  -77942.]
G_plane 104851.0                            [ -92963.  -92669.  -92726. ...  -77360.  -78006.  -76549.]] max and min: 31480.0 -100467.0
g_bias 27516.964746187987
g_lambdas 1048.0
g_exp  499.1631499583476
G_plane/g_lim 27516.964746187987
(G_lambda.data_max)/(max_lambda*k*Gexp) 27516.964746187987
```

Figure 8. SVM in integers modulo-M using α= 5, β= 7, M = 524256.
Source: The authors.



```
modulus set: [31 16 33]

Dynamic range: 16368

input vector modulo M: [    0     1     2 ... 16365 16366 16367]

input vector in RNS: [[ 0  0  0]
 [ 1  1  1]
 [ 2  2  2]
 ...
 [28 13 30]
 [29 14 31]
 [30 15 32]]

sign using LPN method: [ 0  1  1 ... -1 -1 -1]
```

Figure 9. Sign function developed in Python for RNS using α= 4, β= 5, M = 16368
Source: The authors.



```
                                          mod M plane [[-1898. -1898. -1865. ... -2425. -2425. -2425.]
                                           [-1845. -1845. -1790. ... -2283. -2283. -2283.]
                                           [-1845. -1845. -1790. ... -2283. -2283. -2283.]
  beta = 5                                 ...
  alpha = 4                                [-2799. -2799. -2738. ... -1961. -1961. -1961.]
 Dynamic range: 16368                      [-2799. -2799. -2738. ... -1961. -1961. -1961.]
g_lim  4.096356377444825                   [-2799. -2799. -2738. ... -1961. -1961. -1961.]] max and min: 1203.0 -2799.0
G_plane 3273.0                            hiperplano rns calculado:  [[24.  6. 16.]
g_bias 799.0027474224769                   [24.  6. 16.]
g_lambdas 32.0                             [26.  7. 16.]
g_exp  16.386426757900885                  ...
G_plane/g_lim 799.0027474224769            [23.  7. 19.]
(G_lambda.data_max)/(max_lambda*k*Gexp) 799.002747422477   [23.  7. 19.]
                                           [23.  7. 19.]]
```

Figure 10. SVM in RNS using α= 4, β= 5, M = 16368
Source: The authors.

Table 3.
Literature review

Autor	Year	Main objective	Comments
[14]	2020	To reduce the resources required to build and operate CNN based on RNS.	RNS proves to be advantageous for enhancing the efficiency of arithmetic operations in CNNs.
[15]	2021	To design a CNN accelerator that operates entirely in RNS, handling data, stored weights, and communication/computation tasks.	The results demonstrate that integrating RNS into a CNN based on a Processing-in-Memory architecture leads to enhanced computational performance and reduced energy consumption.
[16]	2022	To propose a systematic methodology called ReFACE for the efficient implementation of computing-in-memory (CIM) accelerators for FIR filters. ReFACE leverages the benefits of the residue number system (RNS) to enhance the performance of digital filters.	The utilization of RNS in FIR filters resulted in significant reductions in power consumption and latency.
[17]	2023	To present a novel design approach for ultralow-power arithmetic circuits, which leverages the Residue Number System (RNS) in combination with deterministic bit-streams	The proposed method offers a cost-efficient design in terms of area occupancy and power consumption. These features make it an ideal choice for edge devices, particularly energy harvesting and bio-implantable devices.

Source: The authors.

The results obtained indicate that the SVM in RNS and SVM in modulo-M demonstrate similar performance to the other methods, with decision boundaries that are almost identical across different modulo sets. In contrast, the LPN method for implementing the sign function has been shown to be effective for classification purposes in SVM-RNS. This suggests that the LPN-based approach provides a viable solution for implementing the sign function in RNS-based SVM classifiers.

9 Literature review

Table 3 provides a summary of recent research studies that have focused on utilizing of RNS in machine learning applications to enhance computing performance. The literature review reveals that incorporating RNS in signal processing systems can lead to substantial benefits such as improved power efficiency and reduced latency.

10 Conclusions

The results show that it is possible to implement SVMs using integer numbers, even with nonlinear kernel functions such as Radial Basis Functions. The modulo-M approximation provides a better fit to the original SVM when the dynamic range is increased, allowing for more possible represented values. However, this approach requires a larger Look-Up Table (LUT) size, which may demand more memory resources.

Furthermore, the implementation of SVM-RNS demonstrates that Residue Number System (RNS) is suitable for SVMs with RBF kernels. This opens possibilities for future research in developing hardware-based methods for implementing classifiers. RNS-based SVMs provide a promising avenue for efficient and robust hardware implementations of SVM classifiers, which can be explored in future studies.

Both Look-Up Tables (LUTs) and Residue Number System (RNS) have been effective techniques for feature extraction, as in the case of Wavelet coefficients [18], and for training Artificial Neural Networks (ANNs) [19]. In addition, modular arithmetic has been used in various deep learning acceleration techniques, such as employing Number Theoretical Transforms (NTTs) in convolutional layers [20].

It is important to note that approaches like Convolutional Neural Network (CNN) require more resources compared to an SVM and may offer similar performance in certain applications, such as EMG pattern recognition [21]. Thus, the choice between SVMs and CNNs should depend on the specific requirements and constraints of a given application. Depending on the situation, SVMs can provide a simpler and more resource-efficient solution while still achieving satisfactory performance.

Nonlinear activation functions play a crucial role in introducing complex decision boundaries in the stages of a classifier. Commonly used nonlinear activation functions include softmax, logistic, hyperbolic, and ReLU [22]. In the context of neural network development with the RNS system, advantages have been observed in performing addition, subtraction, and multiplication operations [15].

However, when it comes to using nonlinear functions such as divisions, comparisons, and exponentials in the RNS, there is a high computational cost involved [23]. Previous research has explored various implementations, including representations in Mixed Radix Representation and the Chinese Remainder Theorem [24]. However, these methods can be complex to implement. In the case of the comparison operation, the LPN method has been recently implemented in a memristor array (PIM) [13]. In other recent research, a comparable approach has been demonstrated utilizing the Mixed Radix representation with dynamic range partitioning [25]. It is noteworthy to emphasize that one of the major contributions of this research is the proposal of a method to utilize SVM with RNS. This method involves the use of the LPN-based comparison operation with a modulus set distinct from the one utilized in the implementation with PIM.

Additionally, the research presents a methodology for performing the exponential operation in RNS. Both of these objectives hold fundamental importance in advancing the practical application of unconventional numerical systems, such as RNS, in pattern recognition systems and digital signal processing [11,15,16,24,26].

References

[1] Cardarilli, G.C., Nannarelli, A. and Re, M., Residue number system for low-power DSP applications, in: 2007 Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers, 2007, pp. 1412-1416. DOI: https://doi.org/10.1109/ACSSC.2007.4487461

[2] Albicocco, P., Cardarilli, G. C., Nannarelli A. and Re, M., Twenty years of research on RNS for DSP: Lessons learned and future perspectives, 2014 International Symposium on Integrated Circuits (ISIC), pp. 436-439, 2014. DOI: https://doi.org/10.1109/ISICIR.2014.7029575

[3] Boser, B.E., Guyon, I.M. and Vapnik, V.N., A training algorithm for optimal margin classifiers, in: Proceedings of the fifth annual workshop on Computational learning theory, 1992, pp. 144-152.

[4] Cortes, C. and Vapnik, V., Support-vector networks, Machine learning, 1995, pp. 273-297.

[5] Chang, C.H., Molahosseini, A.S., Zarandi, A.A.E. and Tay, T.F., Residue number systems: a new paradigm to datapath optimization for low-power and high-performance digital signal processing applications., IEEE Circuits and Systems Magazine, 15, pp. 26-44, 2015. DOI: https://doi.org/10.1109/MCAS.2015.2484118

[6] Arthireena, S. and Shanmugavadivel, G., Efficient sign detection using parallel prefix adder, in: IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE), 2017, pp. 1-5. DOI: https://doi.org/10.1109/ICEICE.2017.8191852

[7] Hakonen, M., Piitulainen, H. and Visala, A., Current state of digital signal processing in myoelectric interfaces and related applications, Biomedical Signal Processing and Control, 18, pp. 334-359, 2015. DOI: https://doi.org/10.1016/j.bspc.2015.02.009

[8] Liao, L.Z., Tseng, Y.L., Chiang, H.H. and Wang, W.Y., EMG-based control scheme with SVM classifier for assistive robot arm, in: 2018 International Automatic Control Conference, 2018, pp. 1-5. DOI: https://doi.org/10.1109/CACS.2018.8606762

[9] Tymchyshyn, V. and Khlevniuk A., Yet more simple SMO algorithm, 2020.

[10] Shashua, A., Introduction to machine learning: class notes, in Introduction to machine learning: class notes, 2009, pp. 30-39.

[11] Jenkins, W.K., Soderstrand, M.A. and Radhakrishnan, C., Historical patterns of emerging residue number system technologies during the evolution of computer engineering and digital signal processing, in: IEEE International Symposium on Circuits and Systems (ISCAS), pp. 2018, pp. 1-5. DOI: https://doi.org/10.1109/ISCAS.2018.8351066

[12] Shoup, V., Congruences, in a computational introduction to number theory and algebra, Cambridge University Press, U.K., 2009, 15 P. DOI: https://doi.org/10.1017/CBO9781139165464

[13] Salamat, S., Imani, M., Gupta, S. and Rosing, T., Rnsnet: in-memory neural network acceleration using residue number system, in: 2018 IEEE International Conference on Rebooting Computing (ICRC), 2018, pp. 1-12. DOI: https://doi.org/10.1109/ICRC.2018.8638592

[14] Chervyakov, N.I., Lyakhov, P.A., Deryabin, M.A., Nagornov, N.N., Valueva, M.V. and Valuev, G.V., Residue number system-based solution for reducing the hardware cost of a convolutional neural network, Neurocomputing, 407, pp. 439-453, 2020.

[15] Roohi, A., Taheri, M., Angizi, S. and Fan, D., Rnsim: efficient deep neural network accelerator using residue number systems. In: 2021 IEEE/ACM International Conference on Computer Aided Design, ICCAD, 2021, pp. 1-9, DOI: https://doi.org/10.1016/j.neucom.2020.04.018

[16] Roohi, A., Angizi, S., Navaeilavasani, P, and Taheri, M., ReFACE: efficient design methodology for acceleration of digital filter implementations, in: 2022 23rd International Symposium on Quality Electronic Design (ISQED), 2022, pp. 1-6. DOI: https://doi.org/10.1109/ISQED54688.2022.9806144

[17] Givaki, K., Khonsari, A., Gholamrezaei, M.H., Gorgin, S. and Najafi, M.H., A generalized residue number system design approach for ultra-low power arithmetic circuits based on deterministic bit-streams, in: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2023, 14 P. DOI: https://doi.org/10.1109/TCAD.2023.3250603

[18] Ramírez, J., García, A., Meyer-Bäse, U., Taylor F. and Lloris, A., Implementation of RNS-based distributed arithmetic discrete wavelet

transform architectures using field-programmable logic., Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology, 33(1), pp. 171-190, 2003.

[19] Nakahara, H. and Sasao, T., A deep convolutional neural network based on nested residue number system, in: 2015 25th International Conference on Field Programmable Logic and Applications (FPL), 2015, pp. 1-6. DOI: https://doi.org/10.1109/FPL.2015.7293933

[20] Xu, W., You, X. and Zhang, C., Using Fermat number transform to accelerate convolutional neural network, in: 2017 IEEE 12th International Conference on ASIC (ASICON), 2017, pp. 1033-1036. DOI: https://doi.org/10.1109/ASICON.2017.8252655

[21] Park, K.H. and Lee, S.W., Movement intention decoding based on deep learning for multiuser myoelectric interfaces, in: 2016 4th international winter conference on brain-computer Interface (BCI), 2016, pp. 1-2. DOI: https://doi.org/10.1109/IWW-BCI.2016.7457459

[22] Lin, L.Y., Schroff, J., Lin, T P. and Huang, T.C., Residue number system design automation for neural network acceleration, in: IEEE International Conference on Consumer Electronics-Taiwan (ICCE-Taiwan), 2020, pp. 1-2. DOI: https://doi.org/10.1109/ICCE-Taiwan49838.2020.9258020

[23] Sakellariou, V., Paliourasy, V., Kouretasy, I., Saleh, H. and Stouraitis, T., A High-performance RNS LSTM block, in: IEEE International Symposium on Circuits and Systems (ISCAS), 2022, pp. 1264-1268. DOI: https://doi.org/10.1109/ISCAS48785.2022.9937633

[24] Sousa, L., Nonconventional computer arithmetic circuits, systems and applications, IEEE Circuits and Systems Magazine, 21(1), pp. 6-40, 2021. DOI: https://doi.org/10.1109/MCAS.2020.3027425

[25] Samimi, N., Kamal, M., Afzali-Kusha, A. and Pedram, M., Res-DNN: a residue number system-based DNN accelerator unit., IEEE Transactions on Circuits and Systems I: regular papers, 67(2), pp. 658-671, 2019. DOI: https://doi.org/10.1109/TCSI.2019.2951083

[26] Soloviev, R., Telpukhov, D., Mkrtchan, I., Kustov, A. and Stempkovskiy, A., Hardware implementation of convolutional neural networks based on residue number system, in: Moscow Workshop on Electronic and Networking Technologies (MWENT), 2020, pp. 1-7. DOI: https://doi.org/10.1109/MWENT47943.2020.9067498

[27] Ran, S., Zhao, B., Dai, X., Cheng, C. and Zhang,Y., Software-hardware co-design for accelerating large-scale graph convolutional network inference on FPGA., Neurocomputing, 532, pp. 129-140, 2023. DOI: https://doi.org/10.1016/j.neucom.2023.02.032.

S. Arenas-Hoyos, received the BSc. in Electronic Eng. from the Universidad del Valle, Colombia in 2015. In September 2011, he joined the Digital Architectures and Microelectronic Group at Universidad del Valle as a researcher. Currently, he is pursuing a PhD degree in Engineering with a focus on Electrical and Electronics Engineering from the same university. Alongside his studies, he holds the position of professor at the Engineering Faculty of Unidad Central del Valle and Universidad del Valle in Tuluá, Colombia. His research interests include programmable architectures, digital systems design, hardware description languages, embedded systems, and digital signal processing.
ORCID: 0000-0002-5396-241X

A. Bernal-Noreña, received the BSc. in Electrical Eng. in 1987 from the Universidad del Valle, Cali, Colombia, the MSc. degree in Science in Electrical Engineering majoring in VLSI circuit design from Escola Politécnica da Universidade de São Paulo, São Paulo, Brazil in 1997, and the PhD degree in Microelectronics from Institute National Polytechnique de Grenoble, Grenoble, France in 1999. Currently, is full professor at the Engineering Faculty of Universidad del Valle, Cali, Colombia and leader of the Group of Digital Architectures and Microelectronic.
ORCID: 0000-0003-4766-8086