A Study on Convolutional Vector-Valued Neural Networks for Color Image Classification

Carolina Rodrigues[1], **Marcos Eduardo Valle**[1]

[1]Instituto de Matemática, Estatística e Computação Científica (IMECC)
Universidade Estadual de Campinas (UNICAMP), Campinas – SP, Brazil

c184197@dac.unicamp.br, valle@ime.unicamp.br

Abstract. *Neural networks have gained significant attention in recent years since the development of deep learning. Vector-valued neural networks (V-nets), including hypercomplex-valued neural networks, are appropriate for processing multidimensional data, such as images or hyperspectral imaging. In this article, we present the mathematical foundation for the development of V-nets. We implement and apply these models for a color image classification task, specifically for detecting acute lymphoblastic leukemia in blood smear images.*

1. Introduction

In recent years, machine learning has influenced how we solve various real-world problems [Géron 2019, Goodfellow et al. 2016]. Indeed, neural networks (NNs) have surpassed many other state-of-the-art approaches in many applications with the development of deep learning (DL). Deep neural networks (DNNs) have become popular partially due to a significant increase in computational power in recent decades. Convolutional neural networks (CNNs) are examples of DNNs applied in image and signal processing [Lecun et al. 2015]. CNNs have a particular type of operation based on the convolution of filters, which are well-adapted to learn and represent local patterns. Thus, CNNs are frequently used as the basis of modern image processing and pattern recognition models. This paper illustrates the application of CNN models for the diagnosis of acute lymphoblastic leukemia [Muhammad et al. 2025, Claro et al. 2020]. Precisely, the CNN models are applied for differentiating healthy white blood cells from lymphoblastic cells in blood smear images [Vogado et al. 2018, Granero et al. 2021, Vieira and Valle 2022b].

This article considers vector-valued neural networks (V-nets), which resemble traditional neural networks but with arithmetic operations defined in non-associative algebras, that is, a vector space equipped with a multiplication [Schafer 1961]. Unlike traditional neural networks, V-nets use vectors instead of real numbers to represent inputs, outputs, and synaptic weights [Valle 2024]. Therefore, V-Nets can process multidimensional data such as color or hyperspectral images. Furthermore, V-nets include hypercomplex-valued NNs [Comminiello et al. 2024], such as NNs based on Clifford Algebra [Buchholz and Sommer 2001, Buchholz and Sommer 2008, Bayro-Corrochano et al. 2005]. Consequently, V-Nets can benefit from the algebra on which it is based. Many works demonstrate the advantages of V-Nets, specifically hypercomplex networks, over equivalent real-valued models, especially in problems involving many channels, such as image processing [Breuils et al. 2022, Parcollet et al. 2020, Vieira and Valle 2022a]. Recent works also show that hypercomplex networks stand out

at reducing computational complexity while offering similar or equal performance as real-valued networks [Grassucci et al. 2023, Grassucci et al. 2022, Vieira and Valle 2022b].

The paper is organized as follows. The next section reviews mathematical concepts, focusing on non-associative and hypercomplex algebras. Section 3 addresses vector-valued matrix computations, while Section 4 demonstrates the implementation of vector-valued dense and convolutional layers using the `keras` deep learning library. In Section 5, we present vector-valued CNNs and their application to the image classification problem of diagnosing acute lymphoblastic leukemia. The paper concludes with final remarks in Section 6.

2. Non-Associative Algebras

A non-associative algebra is a vector space enriched with a multiplication of vectors [Schafer 1961]. Formally, a non-associative algebra $\mathbb{V}$ is a vector space over a field $\mathbb{F}$ with an additional bilinear operation called multiplication or product, which is not necessarily associative. As a bilinear operation, the multiplication of $x, y \in \mathbb{V}$, denoted by xy satisfies

$$\begin{aligned} &(x+y)z = x + yz, \;\; z(x+y) = zx + zy, \;\; \forall x, y, z \in \mathbb{V} \\ &\text{and} \;\; \alpha(xy) = (\alpha x)y = x(\alpha y), \;\; \forall \alpha \in \mathbb{F} \text{ and } \forall x, y \in \mathbb{V}. \end{aligned} \tag{1}$$

To implement these operations on computers, we simplify by considering $\mathbb{F} = \mathbb{R}$. Additionally, we will only consider finite-dimensional vector spaces.

Given an ordered basis $\mathcal{E} = \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\}$ on $\mathbb{V}$ and $x \in \mathbb{V}$, there exists a unique tuple $(x_1, \ldots, x_n) \in \mathbb{R}^n$ such that

$$x = \sum_{i=1}^{n} x_i \boldsymbol{e}_i \tag{2}$$

where $x_1, \ldots x_n$ are the coordinates of x with respect to the ordered basis $\mathcal{E}$. Despite equivalent, we can distinguish $\mathbb{V}$ from $\mathbb{R}^n$ by introducing the map $\varphi : \mathbb{V} \to \mathbb{R}^n$ given by

$$\varphi(x) = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n, \;\; \forall x \in \mathbb{V}. \tag{3}$$

The mapping φ is an isomorphism between $\mathbb{V}$ and $\mathbb{R}^n$. As a result, $\mathbb{V}$ inherits the topology and metric from $\mathbb{R}^n$.

The multiplication in $\mathbb{V}$ is completely characterized by the product between the elements of the basis $\mathcal{E} = \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\}$. Precisely, let

$$\boldsymbol{e}_i \boldsymbol{e}_j = \sum_{k=1}^{n} p_{ijk} \boldsymbol{e}_k, \;\; \forall i, j = 1, \ldots, n. \tag{4}$$

be the product of two elements of $\mathcal{E}$. Their product can be organized in the so-called multiplication table shown in Table 1. The multiplication table completely characterizes the non-associative algebra. Furthermore, it is possible to infer properties of the non-associative algebra from the multiplication. For example, an algebra is commutative if and only if $\boldsymbol{e}_i \boldsymbol{e}_j = \boldsymbol{e}_j \boldsymbol{e}_i$, for all $i, j = 1, \ldots, n$. Equivalently, we have $p_{ijk} = p_{jik}$ for all $i, j, k = 1, \ldots, n$, which means that the multiplication table is symmetric.

Table 1. Generic multiplication table.

	e_j
	$\vdots$
e_i	$\cdots \quad \sum_{k=1}^{n} p_{ijk} e_k \quad \cdots$
	$\vdots$

Complex Quaternion Octonion

1	i_1	i_2	i_3	i_4	i_5	i_6	i_7
i_1	-1	i_3	$-i_2$	i_5	$-i_4$	$-i_7$	i_6
i_2	$-i_3$	-1	i_1	i_6	i_7	$-i_4$	$-i_5$
i_3	i_2	$-i_1$	-1	i_7	$-i_6$	i_5	$-i_4$
i_4	$-i_5$	$-i_6$	$-i_7$	-1	i_1	i_2	i_3
i_5	i_4	$-i_7$	i_6	$-i_1$	-1	$-i_3$	i_2
i_6	i_7	i_4	$-i_5$	$-i_2$	i_3	-1	$-i_1$
i_7	$-i_6$	i_5	i_4	$-i_3$	$-i_2$	i_3	-1

Figure 1. The multiplication table of complex numbers, quaternions, and octonions with respect to the canonical basis.

2.1. Hypercomplex Algebra

A hypercomplex algebra $\mathbb{H}$ is a finite-dimensional non-associative algebra where the product has a two-sided identity [Kantor and Solodovnikov 1989, Valle 2024]. Recall that $\boldsymbol{e}_0 \in \mathbb{V}$ is a two-sided identity if

$$x\boldsymbol{e}_0 = \boldsymbol{e}_0 x, \quad \forall x \in \mathbb{H}. \tag{5}$$

The identity $\boldsymbol{e}_0$, also denoted by $\mathbf{1}$, is usually the first element of the ordered basis $\mathcal{E}$ of a hypercomplex algebra. Accordingly, the canonical basis of a hypercomplex algebra $\mathbb{H}$ of dimension $dim(\mathbb{H}) = n + 1$ is $\tau = \{\mathbf{1}, \boldsymbol{i}_1, \ldots, \boldsymbol{i}_n\}$, and a hypercomplex number is given by $x = x_0 + x_1\boldsymbol{i}_1 + \ldots + x_n\boldsymbol{i}_n$. Figure 1 shows the multiplication table of the complex numbers, quaternions, and octonions with respect to the the canonical basis $\tau_{\mathbb{C}} = \{\mathbf{1}, \boldsymbol{i_1}\}$, $\tau_{\mathbb{Q}} = \{\mathbf{1}, \boldsymbol{i_1}, \boldsymbol{i_2}, \boldsymbol{i_3}\}$, and $\tau_{\mathbb{O}} = \{\mathbf{1}, \boldsymbol{i_1}, \boldsymbol{i_2}, \boldsymbol{i_3}, \boldsymbol{i_4}, \boldsymbol{i_5}, \boldsymbol{i_6}, \boldsymbol{i_7}\}$, respectively.

2.2. Product Operation

Using the distributive law and the multiplication table, the product of $x, y \in \mathbb{V}$ is given by

$$xy = \left(\sum_{i=1}^{n} x_i \boldsymbol{e}_i\right)\left(\sum_{j=1}^{n} y_j \boldsymbol{e}_j\right) = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i y_i (\boldsymbol{e}_i \boldsymbol{e}_j) = \sum_{k=1}^{n}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} x_i y_j p_{ijk}\right)\boldsymbol{e}_k. \quad (6)$$

By letting $\mathcal{B}_k : \mathbb{V} \times \mathbb{V} \to \mathbb{R}$ be given by

$$\mathcal{B}_k(x, y) = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i y_j p_{ijk}, \ \ \forall k = 1, .., n. \quad (7)$$

we have $xy = \sum_{k=1}^{n} \mathcal{B}_k(x, y)\boldsymbol{e}_k$. Moreover, the function $\mathcal{B}_k$ is a bilinear mapping whose matrix representation in the ordered basis $\mathcal{E}$ is

$$B_k = \begin{bmatrix} p_{11k} & p_{12k} & \dots & p_{1nk} \\ p_{21k} & p_{22k} & \dots & p_{2nk} \\ p_{n1k} & p_{n2k} & \dots & p_{nnk} \end{bmatrix} \in \mathbb{R}^{n\times n}, \ \ \forall k = 1, \dots, n. \quad (8)$$

Hence, $\mathcal{B}_k(x, y) = \varphi(x)^T B_k \varphi(y)$.

Alternatively, the product can be obtained through an operation between a matrix and a vector. Given a vector $a = \sum_{i=1}^{n} a_i \boldsymbol{e}_i \in \mathbb{V}$, the multiplication of x to the left by a yields a linear operator $\mathcal{A}_L : \mathbb{V} \to \mathbb{V}$ given by $\mathcal{A}_L = ax$, for all $x \in \mathbb{V}$. Thus, the matrix representation of $\mathcal{A}_L$ relative to the basis $\mathcal{E}$ yields the mapping $M_L : \mathbb{V} \to \mathbb{R}^{n\times n}$ given by

$$\begin{aligned} M_L(a) &= \begin{bmatrix} | & | & & | \\ \varphi(a\boldsymbol{e}_1) & \varphi(a\boldsymbol{e}_2) & \dots & \varphi(a\boldsymbol{e}_n) \\ | & | & & | \end{bmatrix} \\ &= \begin{bmatrix} \sum_{i=1}^{n} a_i p_{i11} & \sum_{i=1}^{n} a_i p_{i11} & \dots & \sum_{i=1}^{n} a_i p_{i11} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} a_i p_{i11} & \sum_{i=1}^{n} a_i p_{i11} & \dots & \sum_{i=1}^{n} a_i p_{i11} \end{bmatrix} \\ &= \sum_{i=1}^{n} a_i P_i^T, \quad \text{where} \quad P_i^T = \begin{bmatrix} p_{i11} & \dots & p_{in1} \\ \vdots & \ddots & \vdots \\ p_{i1n} & \dots & p_{inn} \end{bmatrix}. \end{aligned} \quad (9)$$

In short, $M_L : \mathbb{V} \to \mathbb{R}^{n\times n}$ maps $a \in \mathbb{V}$ to its matrix representation in the multiplication to the left relative to the basis $\mathcal{E} = \{\boldsymbol{e}_1, \dots, \boldsymbol{e}_n\}$. Using the matrix representation, the product ax is given by

$$\varphi(ax) = M_L(a)\varphi(x) = \sum_{i=1}^{n} a_i P_i^T \varphi(x), \ \ \forall a = \sum_{i=1}^{n} a_i \boldsymbol{e}_i, x \in \mathbb{V}. \quad (10)$$

Example 1 *The product between quaternions* $a = a_0 + a_1\boldsymbol{i_1} + a_2\boldsymbol{i_2} + a_3\boldsymbol{i_3}$ *and* $x = x_0 + x_1\boldsymbol{i_1} + x_2\boldsymbol{i_2} + x_3\boldsymbol{i_3}$ *is given by*

$$\begin{aligned} ax =& (a_0x_0 - a_1x_1 - a_2x_2 - a_3x_3) + (a_0x_1 + a_1x_0 + a_2x_3 - a_3x_2)\boldsymbol{i_1} \\ &+ (a_0x_2 - a_1x_3 + a_2x_0 + a_3x_1)\boldsymbol{i_2} + (a_0x_3 + a_1x_2 - a_2x_1 + a_3x_0)\boldsymbol{i_3}. \end{aligned} \quad (11)$$

Using the isomorphism φ given by (3)*, we obtain*

$$\varphi(ax) = \begin{bmatrix} a_0x_0 - a_1x_1 - a_2x_2 - a_3x_3 \\ a_0x_1 + a_1x_0 + a_2x_3 - a_3x_2 \\ a_0x_2 - a_1x_3 + a_2x_0 + a_3x_1 \\ a_0x_3 + a_1x_2 - a_2x_1 + a_3x_0 \end{bmatrix}, \tag{12}$$

which can be rewritten as $\varphi(ax) = \mathcal{M}_L(a)\varphi(x)$ with

$$M_L(a) = \begin{bmatrix} a_0 & -a_1 & -a_2 & -a_3 \\ a_1 & a_0 & -a_3 & a_2 \\ a_2 & a_3 & a_0 & -a_1 \\ a_3 & -a_2 & a_1 & a_0 \end{bmatrix} \quad \textit{and} \quad \varphi(x) = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}. \tag{13}$$

3. Vector-Valued Matrix Computation

Similarly to traditional matrix algebra, the product between two vector-valued matrices $A \in \mathbb{V}^{M\times L}$ and $B \in \mathbb{V}^{L\times N}$ is a new matrix $C \in \mathbb{V}^{M\times N}$ with entries given by

$$c_{ij} = \sum_{l=1}^{L} a_{il}b_{lj}, \;\; \forall i = 1, \ldots, M \;\; and \;\; j = 1, \ldots, N. \tag{14}$$

In order to use fast scientific computing software, the above operation is computed through real-valued matrix operations. Using the isomorphism $\varphi : \mathbb{V} \to \mathbb{R}^n$ and the mapping $M_L : \mathbb{V} \to \mathbb{R}^{n\times n}$, we obtain

$$\varphi(c_{ij}) = \sum_{l=1}^{L} \varphi(a_{il}b_{lj}) = \sum_{l=1}^{L} M_L(a_{il})\varphi(b_{lj}) \tag{15}$$

which is equivalent to the real-valued matrix operation

$$\varphi(C) = M_L(A)\varphi(B), \tag{16}$$

where

$$M_L(A) = \begin{bmatrix} M_L(a_{11}) & \ldots & M_L(a_{1L}) \\ \vdots & \ddots & \vdots \\ M_L(a_{M1}) & \ldots & M_L(a_{ML}) \end{bmatrix} \in \mathbb{R}^{nM\times nL}, \tag{17}$$

and

$$\varphi(B) = \begin{bmatrix} \varphi(b_{11}) & \ldots & \varphi(b_{1N}) \\ \vdots & \ddots & \vdots \\ \varphi(b_{L1}) & \ldots & \varphi(b_{LN}) \end{bmatrix} \in \mathbb{R}^{nL\times N}. \tag{18}$$

Therefore, we have $C = \varphi^{-1}(M_L(A)\varphi(B))$, which allows the computation of vector-valued matrix operations through real-valued linear algebra available in the current scientific computing software.

3.1. The Kronecker Product

In order to further reduce the computing time, the matrix $M_L(A) \in \mathbb{R}^{nM \times nL}$ can be computed using the Kronecker product [Loan 2000]. The Kronecker product of two real-valued matrices $A = (a_{ij}) \in \mathbb{R}^{N \times M}$ and $B = (b_{ij}) \in \mathbb{R}^{P \times Q}$ is given by

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1M}B \\ \vdots & \ddots & \vdots \\ a_{N1}B & \dots & a_{NM}B \end{bmatrix} \in \mathbb{R}^{NP \times MQ}. \tag{19}$$

Using Kronecker product, we have

$$M_L(A) = \sum_{k=1}^{n} \begin{bmatrix} a_{11k}P_k^T & \dots & a_{1Lk}P_k^T \\ \vdots & \ddots & \vdots \\ a_{M1k}P_k^T & \dots & a_{MLk}P_k^T \end{bmatrix} = \sum_{k=1}^{n} A_k \otimes P_i^T, \tag{20}$$

where $A = \sum_{k=1}^{n} A_k \boldsymbol{e}_k$, with real-valued matrices $A_k \in \mathbb{R}^{M \times L}$ for all $k = 1, \dots, n$. As a consequence, the vector-valued matrix product $C = AB$ can be computed by

$$C = \varphi^{-1}\left(\left[\sum_{k=1}^{n} A_k \otimes P_k^T\right] \varphi(B)\right). \tag{21}$$

4. Vector-Valued Neural Networks

In traditional neural networks, known as real-valued neural networks, data is represented as multidimensional arrays of real numbers. In contrast, vector-valued neural networks (V-nets) represent data as arrays of vectors. This representation allows V-nets to naturally account for the intercorrelation between feature channels using vector algebra, making these models less likely to get stuck in a local minimum during training. As a result, V-nets generally experience more robust training compared to traditional networks and typically have fewer parameters. When working with vector-valued data, such as audio signals or RGB color images, models specifically designed to handle arrays of vectors may perform better than traditional models that deal with arrays of real numbers. The following section reviews the dense and convolutional layers used in vector-valued networks, as described in [Valle 2024].

4.1. Dense Layers

In neural networks, dense layers consist of multiple parallel neurons, each receiving inputs through synaptic connections. Each neuron computes a linear combination of its inputs, weighted by trainable parameters known as synaptic weights. A scalar bias term is added to this combination. Additionally, the output of each neuron can be transformed using a nonlinear activation function. In mathematical terms, the output of the ith vector-valued neuron in a dense layer can be expressed as:

$$y_i = \psi(s_i + b_i), \quad \text{where} \quad s_i = \sum_{j=1}^{N} \omega_{ij} x_j, \quad \forall i = 1, \dots, M, \tag{22}$$

where $x_1, \ldots, x_N \in \mathbb{V}$ represent the vector-valued inputs, ω_{ij} are the synaptic weights connecting input j to neuron i, $b_i \in \mathbb{V}$ is the bias term for neuron i, and $\psi : \mathbb{V} \to \mathbb{V}$ denotes the vector-valued activation function.

We would like to remark that traditional dense layers and vector-valued dense layers become equivalent when the dimension of the vector space $\mathbb{V}$ equals one, that is, when $dim(\mathbb{V}) = 1$.

4.2. Convolutional Layers

Convolutional layer neurons, unlike dense ones, are only connected to a section of the feature maps of the previous layer through a filter. In addition to significantly reducing the number of parameters, convolutional layers add a level of spatial invariance to the model and are effective for local pattern detection.

Let x be the input image with C vector-valued feature channels. We denote by $x(p, c) \in \mathbb{V}$ the content of the cth feature channel located on the pixel $p \in D_x$, where D_x is the domain of x. The weights of a convolutional layer with K filters are arranged in a vector W such that $W(q, c, k) \in \mathbb{V}$ is the weight of the kth filter in the cth feature channel at $q \in D$, where D is the filter domain. The vector-valued convolution of W and x is given by

$$(W * x)(p, k) = \sum_{c=1}^{C} \sum_{q \in D} W(q, c, k) x(p + S(q), c),\ p \in D_x,\ \forall k = 1, \ldots, K \tag{23}$$

where $p + S(q)$ is a translation that may account for strides. The output of a convolutional layer is obtained by adding a bias term and applying an activation function as follows:

$$y = \psi(W * x + b). \tag{24}$$

Using the isomorphism $\varphi : \mathbb{V} \to \mathbb{R}^n$, the Kronecker product and linearity, we are able to express the convolution operation (23) by means of the equation

$$\varphi(W * x) = \left(\sum_{l=1}^{n} W_l \otimes P_{l:}^T \right) * \varphi(x) = \sum_{l=1}^{n} M_l * \varphi(x), \tag{25}$$

where $W = W_1 \boldsymbol{e}_1 + \ldots + W_n \boldsymbol{e}_n$ is a representation of the filters relative to the ordered basis $\mathcal{E} = \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\}$, $M_l = W_l \otimes P_{i:}^T$ are real-valued filters obtained using the Kronecker product, and $\varphi(x)$ is obtained by concatenating the components of $x = x_1 \boldsymbol{e}_1 + \ldots + x_n \boldsymbol{e}_n$ in the feature axis.

4.3. Combining Traditional and Vector-valued Layers

Traditional CNNs include not only dense and convolutional layers but also pooling layers and others [Lecun et al. 2015]. While there are ongoing efforts to create vector-valued versions of these layers, we can use a method that combines traditional pooling layers with vector-valued convolutional and dense layers. This method effectively applies the pooling layer in a component-wise manner [Valle 2024]. Additionally, V-nets can incorporate a traditional output dense layer for classification tasks. A real-valued dense layer is equivalent to taking the real part of a hypercomplex-valued dense layer [Valle 2024].

5. Computational Experiment

This section outlines the implementation of a vector-valued convolutional neural network (V-CNN) utilizing various hypercomplex algebras, specifically complex numbers, quaternions, and octonions. Additionally, it discusses the application of V-CNNs in a classification problem associated with the diagnosis of acute lymphoblastic leukemia.

5.1. The classification problem

We consider the "ALL-IDB: Acute Lymphoblastic Leukemia Image Database for Image Processing," developed by Fabio Scotti and his collaborators at the Department of Information Technology, Università degli Studi di Milano, Italy. This dataset contains a total of 260 images, with 130 images classified as healthy cells (classification 0) and 130 images classified as lymphoblasts (classification 1). We use one-hot encoding for the class labels. The images have all been resized to 128×128 using the RGB color space. The primary objective of our model is to accurately predict the class of a given image, determining whether it depicts a healthy cell or a lymphoblast.

5.2. Hypercomplex codification: Complex, Quaternion, and Octonion

In order to develop the complex, quaternion, and octonion-valued models, we must first codify the real-valued inputs as hypercomplex-valued ones. By utilizing these hypercomplex algebras, we can leverage their algebraic and geometric properties. The real-valued inputs are represented as an array (or tensor) of size $(128, 128, 3)$, where the first two entries correspond to the width and height in pixels, and the third entry contains the rescaled values of the image's RGB entries.

Let $x_{\mathbb{R}} = [\mathtt{R}, \mathtt{G}, \mathtt{B}]$ denote an RGB color image. The following encoding strategies have been used for encoding the RGB image into complex and quaternion-valued images:

- **Complex Encoding**: The RGB color pixel is encoded into two complex numbers by means of the equation:
$$x_{\mathbb{C}}^{(1)} = [\mathtt{R}\boldsymbol{i}, \mathtt{G} + \mathtt{B}\boldsymbol{i}]. \tag{26}$$
Note that the first component is a pure imaginary complex number; the real part is zero.
- **Quaternion Encoding**: The RGB color pixel is encoded into a quaternion by means of the equation:
$$x_{\mathbb{Q}}^{(1)} = \mathtt{R}\boldsymbol{i} + \mathtt{G}\boldsymbol{j} + \mathtt{B}\boldsymbol{k}. \tag{27}$$

Besides the encoding given by (26) and (27), we also considered a parameterized encoding strategy defined by the following expressions:

$$\begin{aligned} x_{\mathbb{C}}^{(2)} = [&(\alpha_{1,0}\mathtt{R} + \alpha_{2,0}\mathtt{G} + \alpha_{3,0}\mathtt{B} + \beta_0) + (\alpha_{1,1}\mathtt{R} + \alpha_{2,1}\mathtt{G} + \alpha_{3,1}\mathtt{B} + \beta_1)\boldsymbol{i}, \\ &(\alpha_{1,2}\mathtt{R} + \alpha_{2,2}\mathtt{G} + \alpha_{3,2}\mathtt{B} + \beta_2) + (\alpha_{1,3}\mathtt{R} + \alpha_{2,3}\mathtt{G} + \alpha_{3,3}\mathtt{B} + \beta_3)\boldsymbol{i}], \end{aligned} \tag{28}$$

$$\begin{aligned} x_{\mathbb{Q}}^{(2)} = &(\alpha_{1,0}\mathtt{R} + \alpha_{2,0}\mathtt{G} + \alpha_{3,0}\mathtt{B} + \beta_0) + (\alpha_{1,1}\mathtt{R} + \alpha_{2,1}\mathtt{G} + \alpha_{3,1}\mathtt{B} + \beta_1)\boldsymbol{i} \\ &+ (\alpha_{1,2}\mathtt{R} + \alpha_{2,2}\mathtt{G} + \alpha_{3,2}\mathtt{B} + \beta_2)\boldsymbol{j} + (\alpha_{1,3}\mathtt{R} + \alpha_{2,3}\mathtt{G} + \alpha_{3,3}\mathtt{B} + \beta_3)\boldsymbol{k}. \end{aligned} \tag{29}$$

Note that, like $x_{\mathbb{C}}^{(1)}$, the complex-valued image $x_{\mathbb{C}}^{(2)}$ has two channels. Similarly, the RGB color image is encoded into an octonion-valued image using the equation

$$x_{\mathbb{O}} = (\alpha_{1,0}\mathtt{R} + \alpha_{2,0}\mathtt{G} + \alpha_{3,0}\mathtt{B} + \beta_0) + \sum_{k=1}^{7}(\alpha_{1,k}\mathtt{R} + \alpha_{2,k}\mathtt{G} + \alpha_{3,k}\mathtt{B} + \beta_k)\boldsymbol{i}_k. \tag{30}$$

Table 2. Real- and hypercomplex-valued CNNs architectures.

	Real		Complex		Quaternion		Octonion	
	filters	parameters	filters	parameters	filters	parameters	filters	parameters
Conv#1	32	896	16	608	8	320	4	320
MaxPool#1								
Conv#2	64	18,496	32	9,280	16	4,672	8	2,368
MaxPool#2								
Conv#3	128	73,856	64	36,992	32	18,560	16	9,344
MaxPool#3								
Conv#4	128	147,584	64	73,856	32	36,992	16	18,560
MaxPool#4								
Dense	2	9,218	2	16,386	2	16,386	2	16,386
Total # parameters		250,050		137,138		76,946		47,010

The parameters $\alpha_{i,k}$ and β_k are adjusted during the training of the vector-valued CNN models.

5.3. Convolutional Neural Network Architectures

Table 2 presents the architectures of real-valued and hypercomplex-valued CNN models, omitting the encoding, data augmentation, and dropout layers.

Each convolutional layer consists of filters sized 3×3, with padding set to "SAME", and utilizes the ReLU activation function. For the hypercomplex model, we used the `V_Conv2D` layer available at `https://github.com/mevalle/v-nets`, which supports hypercomplex algebras. The output layer is a real-valued dense layer with two neurons—one for each class—and employs the softmax activation function. As usual, we include a Keras flatten layer before the final dense layer. Finally, we trained 10 times each of our 6 models defined below:

- **Model R:** The real-valued CNN;
- **Models C1 and C2:** The complex-valued CNNs with encodings given by (26) and (28), respectively;
- **Models Q1 and Q2**: The quaternion-valued CNNs with the quaternion encoding given by (27) and (29), respectively;
- **Model O:** The octonion-valued CNN with encoding given by (30).

It is noteworthy that the CNN architectures exhibit a decreasing total number of parameters. Specifically, the real-valued CNN has 250,050 parameters, whereas the octonion-valued model contains only 47,010 parameters (counted as floating-point numbers).

5.4. Training

During training, we utilize data augmentation layers along with dropout regularization techniques. The data augmentation layers include random flips, random rotations (with an angle of 0.3 radians), random zooms (with a zoom factor of 0.2), and random translations (with a shift of (0.1, 0.1)). The dropout rate is set to 0.5.

We employ the Adam optimizer and use the cross-entropy loss function, with a batch size of 32 and a total of 200 epochs. The original dataset is divided into training (64%), validation (16%), and test (20%) sets, with the "`random_state`" argument set to 42. The training process for each model is repeated 10 times, and the resulting loss and accuracy scores are saved for a comparative analysis.

Table 3. The mean and standard deviation of the accuracy on the test set.

	Model R	Modelo C1	Model C2	Model Q1	Model Q2	Model O
mean	0.967308	0.863462	0.934615	0.973077	0.926923	0.971154
std	0.015832	0.196629	0.090291	0.013446	0.055736	0.018689

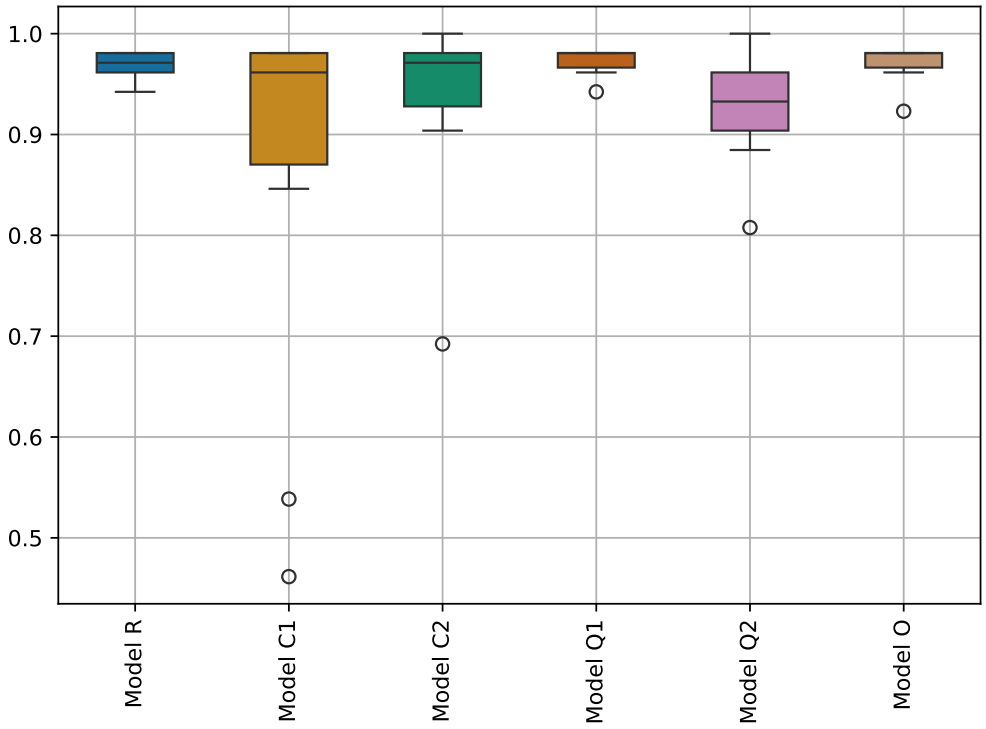
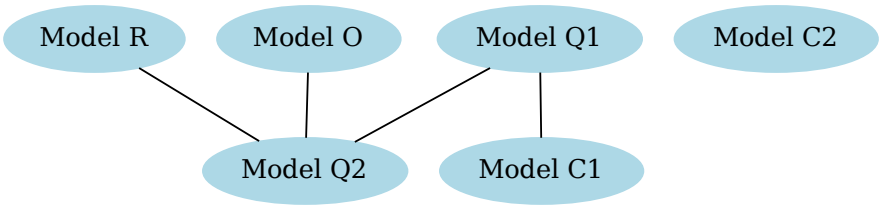


Figure 2. Boxplot with the accuracy of the CNN models on the test set.

5.5. Results and Analysis

Table 3 displays the mean and standard deviation for the accuracy of each model on the test set. To enhance the interpretation of the results from the computational experiment, Figure 2 illustrates the boxplot of the accuracy achieved by each model in the test set. Additionally, Figure 3 presents a Hasse diagram, where the best-performing models are positioned at the top. In this diagram, an edge indicates that the model above outperformed the one below, as determined by a pairwise hypothesis test at a 95% significance level.

The models C1, C2, and Q2 exhibited the largest variations, with models C1 and Q2 also showing the lowest mean accuracy scores. Conversely, models R, C2, Q1, and O achieved the highest accuracy scores and demonstrated statistical equivalence. We can conclude that the V-Nets, particularly the hypercomplex-valued models discussed in this paper, perform nearly as well as, or even slightly better than, their real-valued counterparts. This is accomplished while significantly reducing the number of parameters and, consequently, the computational resources required to define the deep learning models.



Hasse diagram of Wilcoxon signed-rank test
(confidence level at 95.0%)

Figure 3. Hasse diagram comparing the CNN models on the test set.

6. Concluding Remarks

In this article, we reviewed the algebraic foundations of vector-valued and hypercomplex-valued neural networks. We demonstrated that models utilizing complex numbers, quaternions, and octonions can effectively perform image classification on color images. Specifically, we evaluated the performance of hypercomplex-valued convolutional neural networks using the ALL-IDB dataset [Labati et al. 2011]. Our findings indicate that vector-valued models can reduce the number of parameters without compromising accuracy. This reduction is particularly significant in the context of lightweight deep learning, as it requires less computational resources to store the trained model.

Acknowledgments

Carolina Rodrigues acknowledges support from the National Council for Scientific and Technological Development (CNPq), Brazil, through the Institutional Undergraduate Research Scholarship Program (PIBIC) at UNICAMP. Marcos Eduardo Valle acknowledges support from CNPq, Brazil, under grant number 315820/2021-7, and from the São Paulo Research Foundation (FAPESP), Brazil, under grant number 2023/03368-0.

References

Bayro-Corrochano, E., Vallejo, R., and Arana-Daniel, N. (2005). Geometric preprocessing, geometric feedforward neural networks and Clifford support vector machines for visual learning. *Neurocomputing*, 67(1-4 SUPPL.):54–105.

Breuils, S., Tachibana, K., and Hitzer, E. (2022). New Applications of Clifford's Geometric Algebra. *Advances in Applied Clifford Algebras 2022 32:2*, 32(2):1–39.

Buchholz, S. and Sommer, G. (2001). Clifford Algebra Multilayer Perceptrons. In *Geometric Computing with Clifford Algebras*, pages 315–334. Springer Berlin Heidelberg.

Buchholz, S. and Sommer, G. (2008). On Clifford neurons and Clifford multi-layer perceptrons. *Neural Networks*, 21(7):925–935.

Claro, M., Vogado, L., Veras, R., Santana, A., Tavares, J., Santos, J., and MacHado, V. (2020). Convolution Neural Network Models for Acute Leukemia Diagnosis. *International Conference on Systems, Signals, and Image Processing*, 2020-July:63–68.

Comminiello, D., Grassucci, E., Mandic, D. P., and Uncini, A. (2024). Demystifying the hypercomplex: Inductive biases in hypercomplex deep learning. *IEEE Signal Processing Magazine*, 41(3):59–71.

Géron, A. (2019). *Hands–On Machine Learning with Scikit–Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O Reilly, Sebastopol, California, USA., 2nd edition.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Granero, M. A., Hernández, C. X., and Valle, M. E. (2021). Quaternion-Valued Convolutional Neural Network Applied for Acute Lymphoblastic Leukemia Diagnosis. In *Brazilian Conference on Intelligent Systems. BRACIS 2021. Lecture Notes in Computer Science*, pages 280–293. Springer, Cham.

Grassucci, E., Mancini, G., Brignone, C., Uncini, A., and Comminiello, D. (2023). Dual quaternion ambisonics array for six-degree-of-freedom acoustic representation. *Pattern Recognition Letters*, 166:24–30.

Grassucci, E., Zhang, A., and Comminiello, D. (2022). PHNNs: Lightweight Neural Networks via Parameterized Hypercomplex Convolutions. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–13.

Kantor, I. L. and Solodovnikov, A. S. (1989). *Hypercomplex Numbers: An Elementary Introduction to Algebras*. Springer New York.

Labati, R. D., Piuri, V., and Scotti, F. (2011). All-idb: The acute lymphoblastic leukemia image database for image processing. In *2011 18th IEEE International Conference on Image Processing*, pages 2045–2048.

Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Loan, C. F. (2000). The ubiquitous Kronecker product. *Journal of Computational and Applied Mathematics*, 123(1-2):85–100.

Muhammad, D., Salman, M., Keles, A., and Bendechache, M. (2025). ALL diagnosis: can efficiency and transparency coexist? An explainble deep learning approach. *Scientific Reports*, 15(1):1–20.

Parcollet, T., Morchid, M., and Linarès, G. (2020). A survey of quaternion neural networks. *Artificial Intelligence Review*, 53(4):2957–2982.

Schafer, R. (1961). *An Introduction to Nonassociative Algebras*. Project Gutenberg.

Valle, M. E. (2024). Understanding vector-valued neural networks and their relationship with real and hypercomplex-valued neural networks: Incorporating intercorrelation between features into neural networks. *IEEE Signal Processing Magazine*, 41(3):49–58.

Vieira, G. and Valle, M. E. (2022a). A general framework for hypercomplex-valued extreme learning machines. *Journal of Computational Mathematics and Data Science*, 3:100032.

Vieira, G. and Valle, M. E. (2022b). Acute Lymphoblastic Leukemia Detection Using Hypercomplex-Valued Convolutional Neural Networks. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Vogado, L. H., Veras, R. M., Araujo, F. H., Silva, R. R., and Aires, K. R. (2018). Leukemia diagnosis in blood slides using transfer learning in CNNs and SVM for classification. *Engineering Applications of Artificial Intelligence*, 72:415–422.