# Understanding Vector-Valued Neural Networks and Their Relationship with Real and Hypercomplex-Valued Neural Networks

**Preprint** · September 2023

DOI: 10.48550/arXiv.2309.07716

Marcos Eduardo Valle
University of Campinas

# Understanding Vector-Valued Neural Networks and Their Relationship with Real and Hypercomplex-Valued Neural Networks

Marcos Eduardo Valle

## Abstract

Despite the many successful applications of deep learning models for multidimensional signal and image processing, most traditional neural networks process data represented by (multidimensional) arrays of real numbers. The intercorrelation between feature channels is usually expected to be learned from the training data, requiring numerous parameters and careful training. In contrast, vector-valued neural networks are conceived to process arrays of vectors and naturally consider the intercorrelation between feature channels. Consequently, they usually have fewer parameters and often undergo more robust training than traditional neural networks. This paper aims to present a broad framework for vector-valued neural networks, referred to as V-nets. In this context, hypercomplex-valued neural networks are regarded as vector-valued models with additional algebraic properties. Furthermore, this paper explains the relationship between vector-valued and traditional neural networks. Precisely, a vector-valued neural network can be obtained by placing restrictions on a real-valued model to consider the intercorrelation between feature channels. Finally, we show how V-nets, including hypercomplex-valued neural networks, can be implemented in current deep-learning libraries as real-valued networks.

## Index Terms

Multidimensional signal and image processing, vector-valued neural network, hypercomplex-valued neural network, deep learning.

This work has been partially supported by the National Council for Scientific and Technological Development (CNPq) under grants 315820/2021-7 and São Paulo Research Foundation (FAPESP) under grant 2022/01831-2.

Marcos Eduardo Valle is with the Department Applied Mathematics, Institute of Mathematics, Statistics, and Scientific Computing, Universidade Estadual de Campinas (Unicamp), Campinas – Brazil, e-mail: valle@ime.unicamp.br.

# I. Introduction

Neural networks achieved outstanding performance in many signal and image processing tasks, especially with the advent of the deep learning paradigm [1], [2]. Despite their many successful applications, traditional neural networks are theoretically designed to process real-valued or, at most, complex-valued data. Accordingly, signals and images are represented by (possibly multidimensional) arrays of real or complex numbers [1], [3], [4]. Furthermore, traditional neural networks a priori do not consider possible intercorrelation between feature channels. The relationship between features is expected to be learned from the training data. Consequently, besides relying on appropriate loss functions and effective optimizers, traditional deep learning models usually have too many parameters and demand a long training time.

In contrast, vector-valued neural networks (V-nets) are designed to process arrays of vectors. They naturally take into account the intercorrelation between feature channels. Hence, V-nets are expected to have fewer parameters than traditional neural networks. Furthermore, they should be less susceptible to being trapped in a local minimum of the loss function surface during the training. Hypercomplex-valued neural networks are examples of robust and lightweight V-nets for dealing with vector-valued data [5], [6], [7], [8].

This paper aims to present a detailed framework for V-nets, making plain and understandable their relationship with traditional networks and hypercomplex-valued neural networks. Precisely, we first present the mathematical background for vector-valued neural networks. Then, we address the relationship between real and hypercomplex-valued neural networks, focusing on dense and convolutional layers. On the one hand, hypercomplex-valued neural networks are regarded as vector-valued models with additional algebraic properties. On the other hand, V-nets can be viewed as traditional neural networks with restrictions to take into account the intercorrelation between the feature channels. Using these relationships, we show how to emulate vector-valued (and hypercomplex-valued) neural networks using traditional models, allowing us to implement them using current deep-learning libraries.

The paper is structured as follows. Section II provides the mathematical background for V-nets, including hypercomplex algebras and examples [9], [10], [11]. Basic vector-valued matrix operations and their relationship with traditional linear algebra are briefly reviewed in Section III. Section IV introduces V-nets, with a focus on dense and convolutional layers. This section also addresses the approximation capability of shallow dense networks and explains how to implement V-nets using the current deep-learning libraries designed for real-valued data. The paper finishes with concluding remarks in Section V.

## II. Vector and Hypercomplex Algebras

Despite their many successful applications, traditional neural networks are designed to process arrays of real numbers. However, many image and signal-processing tasks – such as those related to multivariate images and 3D audio signals [8], [12] – are concerned with vector-valued data, which can be better explored by considering models designed to deal with arrays of vectors. Because addition and multiplication are core operations for designing effective neural networks, this section reviews some key concepts of algebra. Broadly speaking, an algebra is a vector space (with component-wise vector addition) enriched with a multiplication of vectors. Such mathematical structure yields the background for developing vector-valued and hypercomplex-valued neural networks.

**Definition 1** (Algebra [11]). *An algebra $\mathbb{V}$ is a vector space over $\mathbb{F}$ with an additional bilinear operation called multiplication.*

As a bilinear operation, the multiplication of $x, y \in \mathbb{V}$, denoted by the juxtaposition $xy$, satisfies

$$(x+y)z = xz + yz \quad \text{and} \quad z(x+y) = zx + zy, \quad \forall x, y, z \in \mathbb{V},$$

and

$$\alpha(xy) = (\alpha x)y = x(\alpha y), \quad \forall \alpha \in \mathbb{F} \quad \text{and} \quad x, y \in \mathbb{V}.$$

**Remark 1.** Because we are mainly concerned with the implementation of models on traditional computers, for the sake of simplicity, we only consider algebras over the field of real numbers, that is, $\mathbb{F} = \mathbb{R}$. Furthermore, we will only be concerned with finite dimensional vector spaces. In other words, we assume that $\mathbb{V}$ is a vector space of dimension $n$, i.e., $dim(\mathbb{V}) = n$.

Let $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$ be an ordered basis for $\mathbb{V}$. Given $x \in \mathbb{V}$, there is an unique $n$-tuple $(x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ such that

$$x = \sum_{i=1}^{n} x_i e_i.$$

The scalars $x_1, \ldots, x_n$ are the coordinates of $x$ relative to the ordered basis $\mathcal{E}$. In computational applications, $x \in \mathbb{V}$ is given by its coordinates relative to the ordered basis $\mathcal{E} = \{e_1, \ldots, e_n\}$. Precisely, $x$ is usually given by a vector in $\mathbb{R}^n$, and the canonical basis is often implicitly considered. In order to further distinguish $x \in \mathbb{V}$ from the $n$-tuple $(x_1, \ldots, x_n) \in \mathbb{R}^n$, we introduce the following isomorphism:

**Definition 2** (Isomorphism between $\mathbb{V}$ and $\mathbb{R}^n$)**.** *Given an ordered basis $\mathcal{E} = \{e_1, \ldots, e_n\}$, the mapping $\varphi : \mathbb{V} \to \mathbb{R}^n$ given by*

$$\varphi(x) = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n, \quad \forall x \in \mathbb{V}, \tag{1}$$

*yields an isomorphism between $\mathbb{V}$ and $\mathbb{R}^n$.*

Using the isomorphism $\varphi$, $\mathbb{V}$ inherits the topology and metric from $\mathbb{R}^n$. For example, we can define the absolute value of $x \in \mathbb{V}$ with respect to the basis $\mathcal{E} = \{e_1, \ldots, e_n\}$ as the Euclidean norm of $\varphi(x)$:

$$|x| := \|\varphi(x)\|_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}. \tag{2}$$

We would like to remark, however, that the absolute value of $x$ given by (2) is not an invariant; it depends on the basis $\mathcal{E} = \{e_1, \ldots, e_n\}$. Like traditional linear algebra, the basis $\mathcal{E}$ plays a crucial role in the algebra $\mathbb{V}$.

Let us now show how multiplication is defined on $\mathbb{V}$. Given an ordered basis $\mathcal{E} = \{e_1, \ldots, e_n\}$, the multiplication is completely determined by the $n^3$ parameters $p_{ijk} \in \mathbb{R}$ which appear in the products

$$e_i e_j = \sum_{k=1}^{n} p_{ijk} e_k, \quad \forall i, j = 1, \ldots, n. \tag{3}$$

The products in (3) can be arranged in the so-called multiplication table:

$$\begin{array}{c|ccc} & & e_j & \\ \hline & & \vdots & \\ e_i & \cdots & \sum_{k=1}^{n} p_{ijk} e_k & \cdots \\ & & \vdots & \end{array}$$

The properties of an algebra can be obtained by analyzing the basis elements or the multiplication table. For example, an algebra $\mathbb{V}$ is considered commutative if

$$xy = yx, \quad \forall x, y \in \mathbb{V}.$$

Given an ordered basis $\mathcal{E} = \{e_1, \ldots, e_n\}$, the algebra is commutative if and only if

$$e_i e_j = e_j e_i, \quad \forall i, j = 1, \ldots, n.$$

Equivalently, from the multiplication table, we conclude that an algebra is commutative if and only if

$$p_{ijk} = p_{jik}, \quad \forall i, j, k = 1, \ldots, n.$$

Analogously, an algebra $\mathbb{V}$ is associative if

$$(xy)z = x(yz), \quad \forall x, y, z \in \mathbb{V}.$$

Thus, the algebra is associative if and only if

$$(e_i e_j)e_k = e_i(e_j e_k), \quad \forall i, j, k = 1, \ldots, n.$$

In other words, an algebra is associative if and only if

$$\sum_{\mu=1}^{n} p_{ij\mu} p_{k\mu\ell} = \sum_{\mu=1}^{n} p_{jk\mu} p_{i\mu\ell}, \quad \forall i, j, k, \ell = 1, \ldots, n.$$

The interest in machine learning techniques and neural network models based on hypercomplex algebras, including predominantly complex numbers and quaternions, has a long history [13], [14]. Many researchers (including myself) list the capability to treat multidimensional data as a single entity as one prominent advantage of hypercomplex-valued models. According to Definition 1, however, any algebra provides the mathematical background for dealing with arrays of vectors. Therefore, I suggest defining hypercomplex algebras as algebra with additional (geometric or) algebraic properties. Precisely, I propose the following:

**Definition 3** (Hypercomplex algebra [9], [10])**.** *A hypercomplex algebra, denoted by $\mathbb{H}$, is a finite-dimensional algebra in which the product has a two-sided identity.*

From Definition 3, a hypercomplex algebra $\mathbb{H}$ is equipped with an unique element $e_0$ such that

$$xe_0 = e_0 x = x, \quad \forall x \in \mathbb{V}.$$

The identity is usually the first element of the ordered basis. Thus, $\mathcal{E} = \{e_0, e_1, \ldots, e_n\}$ is an ordered basis of an hypercomplex algebra and $dim(\mathbb{H}) = n + 1$. Moreover, we often consider the canonical basis $\tau = \{1, \boldsymbol{i}_1, \ldots, \boldsymbol{i}_n\}$. Accordingly, a hypercomplex number is given by

$$x = x_0 + x_1 \boldsymbol{i}_1 + \ldots + x_n \boldsymbol{i}_n.$$

The multiplication table of a hypercomplex algebra with respect to the canonical basis $\tau = \{1, \boldsymbol{i}_1, \ldots, \boldsymbol{i}_n\}$ is

|  | $1$ | $\boldsymbol{i}_1$ | $\boldsymbol{i}_j$ | $\boldsymbol{i}_n$ |
|---|---|---|---|---|
| $1$ | $1$ | $\boldsymbol{i}_1$ | $\boldsymbol{i}_j$ | $\boldsymbol{i}_n$ |
|  |  |  | $\vdots$ |  |
| $\boldsymbol{i}_i$ | $\boldsymbol{i}_i$ | $\cdots$ | $p_{ij0} + \sum_{k=1}^{n} p_{ijk} \boldsymbol{i}_k$ | $\cdots$ |
|  |  |  | $\vdots$ |  |

**Remark 2.** We would like to remark that Definition 3 is consistent with the general approach of Kantor and Solodovnik and includes well-known hypercomplex algebras as particular instances [9]. In particular, all Clifford and Cayley-Dickson algebras are examples of hypercomplex algebras.

Let us return our attention to an arbitrary finite-dimensional algebra $\mathbb{V}$. Using the distributive law and the multiplication table, the product of $x = \sum_{i=1}^{n} x_i e_i$ and $y = \sum_{j=1}^{n} y_j e_j$ satisfies

$$xy = \left(\sum_{i=1}^{n} x_i e_i\right)\left(\sum_{j=1}^{n} y_j e_j\right) = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i y_j (e_i e_j) = \sum_{k=1}^{n}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} x_i y_j p_{ijk}\right) e_k.$$

Because the product is bilinear, the function $\mathcal{B}_k : \mathbb{V} \times \mathbb{V} \to \mathbb{R}$ given by

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

is a bilinear form. Therefore, we obtain the following proposition [15]:

**Proposition 1.** *Let $\mathcal{E} = \{e_1, \ldots, e_n\}$ be an ordered basis of an algebra $\mathbb{V}$. The multiplication of $x = \sum_{i=1}^{n} x_i e_i$ and $y = \sum_{j=1}^{n} y_j e_j$ satisfies*

$$xy = \sum_{k=1}^{n} \mathcal{B}_k(x, y) e_k, \tag{4}$$

*where $\mathcal{B}_k : \mathbb{V} \times \mathbb{V} \to \mathbb{R}$ is a bilinear form whose matrix representation in the ordered basis $\mathcal{E}$ is*

$$\boldsymbol{B}_k = \begin{bmatrix} p_{11k} & p_{12k} & \cdots & p_{1nk} \\ p_{21k} & p_{22k} & \cdots & p_{2nk} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1k} & p_{n2k} & \cdots & p_{nnk} \end{bmatrix} \in \mathbb{R}^{n \times n}, \quad \forall k = 1, \ldots, n.$$

*Thus, we have $\mathcal{B}_k(x, y) = \varphi(x)^T \boldsymbol{B}_k \varphi(y)$.*

Using Proposition 1, we introduce the following definition that plays an important role in the approximation capability of V-nets such as the vector-valued multilayer perception (V-MLP) [15]:

**Definition 4** (Non-degenerate algebra)**.** *An algebra $\mathbb{V}$ is non-degenerate if all the bilinear forms $\mathcal{B}_1, \ldots, \mathcal{B}_n$ in* (4) *are non-degenerate*[1]*. Otherwise, we say that the algebra $\mathbb{V}$ is degenerate.*

In addition to expressing the multiplication of two vectors through bilinear forms, it can also be represented as a matrix-vector operation. Precisely, the multiplication to the left by a vector $a = \sum_{i=1}^{n} a_i e_i \in \mathbb{V}$

[1] A bilinear form $\mathcal{B}_k : \mathbb{V} \times \mathbb{V} \to \mathbb{R}$ is non-degenerate if its matrix representation $\boldsymbol{B}_k$ is non-singular with respect to any ordered basis $\mathcal{E} = \{e_1, \ldots, e_n\}$.

yields a linear operator $\mathcal{A}_L : \mathbb{V} \to \mathbb{V}$ defined by $\mathcal{A}_L(x) = ax$, for all $x \in \mathbb{V}$. Therefore, the matrix representation of $\mathcal{A}_L$ relative to an ordered basis $\mathcal{E} = \{e_1, \dots, e_n\}$ yields a mapping $\mathcal{M}_L : \mathbb{V} \to \mathbb{R}^{n \times n}$ given by

$$\mathcal{M}_L(a) = \begin{bmatrix} | & | & & | \\ \varphi(ae_1) & \varphi(ae_2) & \dots & \varphi(ae_n) \\ | & | & & | \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n a_i p_{i11} & \sum_{i=1}^n a_i p_{i21} & \dots & \sum_{i=1}^n a_i p_{in1} \\ \sum_{i=1}^n a_i p_{i12} & \sum_{i=1}^n a_i p_{i22} & \dots & \sum_{i=1}^n a_i p_{in2} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^n a_i p_{i1n} & \sum_{i=1}^n a_i p_{i2n} & \dots & \sum_{i=1}^n a_i p_{inn} \end{bmatrix}.$$

In words, $\mathcal{M}_L : \mathbb{V} \to \mathbb{R}^{n \times n}$ maps a vector $a \in \mathbb{V}$ to its matrix representation in the multiplication by the left with respect to the ordered basis $\mathcal{E}$. Alternatively, we can write

$$\mathcal{M}_L(a) = \sum_{i=1}^n a_i \boldsymbol{P}_{i:}^T, \quad \text{with} \quad \boldsymbol{P}_{i:}^T = \begin{bmatrix} p_{i11} & p_{i21} & \cdots & p_{in1} \\ p_{i12} & p_{i22} & \cdots & p_{in2} \\ \vdots & \vdots & \ddots & \vdots \\ p_{i1n} & p_{i2n} & \cdots & p_{inn} \end{bmatrix}. \tag{5}$$

Using the matrix representation, we have

$$\varphi(ax) = \mathcal{M}_L(a)\varphi(x) = \sum_{i=1}^n a_i \boldsymbol{P}_{i:}^T \varphi(x), \tag{6}$$

for all $a = \sum_{i=1}^n a_i e_i \in \mathbb{V}$ and $x \in \mathbb{V}$. Note that (6) provides an efficient formula for computing vector multiplication using traditional matrix operations.

**Example 1** (Quaternions)**.** Consider the quaternions with the canonical basis $\tau = \{1, \boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}\}$. The product of $x = x_0 + x_1\boldsymbol{i} + x_2\boldsymbol{j} + x_3\boldsymbol{k}$ and $y = y_0 + y_1\boldsymbol{i} + y_2\boldsymbol{j} + y_3\boldsymbol{k}$ satisfies

$$\varphi(xy) = \begin{bmatrix} x_0 & -x_1 & -x_2 & -x_3 \\ x_1 & x_0 & -x_3 & x_2 \\ x_2 & x_3 & x_0 & -x_1 \\ x_3 & -x_2 & x_1 & x_0 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \mathcal{M}_L(x)\varphi(y).$$

Note that

$$\mathcal{M}_L(x) = x_0 \boldsymbol{P}_{0:} + x_1 \boldsymbol{P}_{1:} + x_2 \boldsymbol{P}_{2:} + x_n \boldsymbol{P}_{n:},$$

where $\boldsymbol{P}_{0:} = \mathbf{I}_{4\times 4}$ is the identity matrix and

$$\boldsymbol{P}_{1:}^T = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \boldsymbol{P}_{2:}^T = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}, \quad \text{and} \quad \boldsymbol{P}_{3:}^T = \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

**Example 2** (Parametrized “Hypercomplex” Algebras)**.** Recently, Zhang et al. introduced the so-called *parametrized “hypercomplex” algebras* [16], [7]. A parametrized “hypercomplex” algebra is defined as follows using the matrix representation of the multiplication: Given matrices $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_n \in \mathbb{R}^{n \times n}$ and an ordered basis $\mathcal{E} = \{e_1, \ldots, e_n\}$, the product in a parametrized “hypercomplex” algebra is defined by

$$xy = \varphi^{-1}\Big(\sum_{i=1}^{n} x_i \boldsymbol{P}_i \varphi(y)\Big), \tag{7}$$

for all $x = \sum_{i=1}^{n} x_i e_i$ and $y = \sum_{i=1}^{n} y_i e_i$. Note that (7) is equivalent to (6). Therefore, despite being referred to as “hypercomplex”, the multiplication given by (7) does not necessarily have an identity. Thus, a parameterized “hypercomplex” algebras may not meet the criteria to be classified as a hypercomplex algebra as per the Definition 3. Nevertheless, the multiplication defined by (7) has been effectively used to learn the algebra of vector-valued neural networks [7], [16].

## III. Vector-Valued Matrix Computation

Matrix computation is a key concept for developing efficient vector- and hypercomplex-valued network models because some fundamental building blocks, like dense and convolutional layers, compute affine transformations followed by a non-linear activation function. In this section, we present some basic vector-valued matrix computation concepts [5].

As in the traditional matrix algebra, the product of two vector-valued matrices $\boldsymbol{A} \in \mathbb{V}^{M \times L}$ and $\boldsymbol{B} \in \mathbb{V}^{L \times N}$ results in a new matrix $\boldsymbol{C} \in \mathbb{V}^{M \times N}$ with entries defined by

$$c_{ij} = \sum_{\ell=1}^{L} a_{i\ell} b_{\ell j}, \quad \forall i = 1, \ldots, M \quad \text{and} \quad j = 1, \ldots, N.$$

To take advantage of fast scientific computing software, we compute the above operation using real-valued matrix operations as follows. Using the isomorphism $\varphi : \mathbb{V} \to \mathbb{R}^n$ and the mapping $\mathcal{M}_L : \mathbb{V} \to \mathbb{R}^{n \times n}$ defined respectively by (1) and (5), we obtain

$$\varphi(c_{ij}) = \sum_{\ell=1}^{L} \varphi\left(a_{i\ell} b_{\ell j}\right) = \sum_{\ell=1}^{L} \mathcal{M}_L(a_{i\ell}) \varphi(b_{\ell j}).$$

Equivalently, using real-valued matrix operations, we have

$$\varphi(\boldsymbol{C}) = \mathcal{M}_L(\boldsymbol{A}) \varphi(\boldsymbol{B}), \tag{8}$$

where $\mathcal{M}_L$ and $\varphi$ are extended as follows for vector-valued matrices:

$$\mathcal{M}_L(\boldsymbol{A}) = \begin{bmatrix} \mathcal{M}_L(a_{11}) & \mathcal{M}_L(a_{12}) & \ldots & \mathcal{M}_L(a_{1L}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{M}_L(a_{M1}) & \mathcal{M}_L(a_{M2}) & \ldots & \mathcal{M}_L(a_{ML}) \end{bmatrix} \in \mathbb{R}^{nM \times nL}, \tag{9}$$

and

$$\varphi(\boldsymbol{B}) = \begin{bmatrix} \varphi(b_{11}) & \dots & \varphi(b_{1N}) \\ \varphi(b_{21}) & \dots & \varphi(b_{2N}) \\ \vdots & \ddots & \vdots \\ \varphi(b_{L1}) & \dots & \varphi(b_{LN}) \end{bmatrix} \in \mathbb{R}^{nL \times N}. \tag{10}$$

Therefore, reorganizing the elements of $\varphi(\boldsymbol{C})$, we can write

$$\boldsymbol{C} = \varphi^{-1}\left(\mathcal{M}_L(\boldsymbol{A})\varphi(\boldsymbol{B})\right),$$

which allows the computation of vector-valued matrix products using the real-valued linear algebra often available in scientific computing software.

To further reduce the computing time, the real-valued matrix $\mathcal{M}_L(\boldsymbol{A}) \in \mathbb{R}^{nM \times nL}$ can be computed using the Kronecker product. The Kronecker product between two real-valued matrices $\boldsymbol{A} = (a_{ij}) \in \mathbb{R}^{N \times M}$ and $\boldsymbol{B} \in \mathbb{R}^{P \times Q}$, denoted by $\boldsymbol{A} \otimes \boldsymbol{B}$, yields the block matrix defined by

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{11}\boldsymbol{B} & a_{12}\boldsymbol{B} & \dots & a_{1M}\boldsymbol{B} \\ a_{21}\boldsymbol{B} & a_{22}\boldsymbol{B} & \dots & a_{2M}\boldsymbol{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1}\boldsymbol{B} & a_{N2}\boldsymbol{B} & \dots & a_{NM}\boldsymbol{B} \end{bmatrix} \in \mathbb{R}^{NP \times MQ}.$$

Basic properties and some applications of the Kronecker product can be found in [17], [18].

As per the references [7], [16], we employ the Kronecker product to calculate $\mathcal{M}_L(\boldsymbol{A})$ in the following manner. From (6), we have

$$\mathcal{M}_L(\boldsymbol{A}) = \sum_{k=1}^{n} \begin{bmatrix} a_{11k}\boldsymbol{P}_{k:}^T & a_{12k}\boldsymbol{P}_{k:}^T & \dots & a_{iLk}\boldsymbol{P}_{k:}^T \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1k}\boldsymbol{P}_{k:}^T & a_{M2k}\boldsymbol{P}_{k:}^T & \dots & a_{MLk}\boldsymbol{P}_{k:}^T \end{bmatrix}.$$

Let $\boldsymbol{A}_k \in \mathbb{R}^{M \times L}$, for $k = 1, \dots, n$, be the real-valued matrices such that

$$\boldsymbol{A} = \sum_{k=1}^{n} \boldsymbol{A}_k e_k.$$

In words, $\boldsymbol{A}_k$ is the “matrix” component associated with the basis element $e_k$ of $\boldsymbol{A}$. Using $\boldsymbol{A}_k \in \mathbb{R}^{M \times L}$, we conclude that

$$\mathcal{M}_L(\boldsymbol{A}) = \sum_{k=1}^{n} \boldsymbol{A}_k \otimes \boldsymbol{P}_{k:}^T. \tag{11}$$

Therefore, $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$ can be efficiently computed by the equation

$$\boldsymbol{C} = \varphi^{-1}\left(\Big(\sum_{k=1}^{n} \boldsymbol{A}_k \otimes \boldsymbol{P}_{k:}^T\Big)\varphi(\boldsymbol{B})\right).$$

**Example 3** (Quaternion matrix product)**.** Consider the quaternion-valued matrix

$$\boldsymbol{A} = \begin{bmatrix} 1+2\boldsymbol{i} & 3\boldsymbol{i}+4\boldsymbol{j} & 5\boldsymbol{j}+6\boldsymbol{k} \\ 7+8\boldsymbol{j} & 9+10\boldsymbol{k} & 11\boldsymbol{i}+12\boldsymbol{k} \end{bmatrix} \in \mathbb{Q}^{2\times 3},$$

and the column vector

$$\boldsymbol{x} = \begin{bmatrix} 1+2\boldsymbol{i}+3\boldsymbol{j}+4\boldsymbol{k} \\ 5+6\boldsymbol{i}+7\boldsymbol{j}+8\boldsymbol{k} \\ 9+10\boldsymbol{i}+11\boldsymbol{j}+12\boldsymbol{k} \end{bmatrix} \in \mathbb{Q}^{3\times 1}.$$

Using quaternion matrix algebra, we obtain

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} -176+45\boldsymbol{i}+96\boldsymbol{j}+11\boldsymbol{k} \\ -306-3\boldsymbol{i}+140\boldsymbol{j}+363\boldsymbol{k} \end{bmatrix} \in \mathbb{Q}^{2\times 1}.$$

To determine the quaternion-valued vector $\boldsymbol{y}$ using real-valued matrix computation, we first compute

$$\begin{aligned} \mathcal{M}_L(\boldsymbol{A}) &= \begin{bmatrix} 1 & 0 & 0 \\ 7 & 9 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \ldots + \begin{bmatrix} 0 & 0 & 6 \\ 0 & 10 & 12 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 1 & -2 & 0 & 0 & 0 & -3 & -4 & 0 & 0 & 0 & -5 & -6 \\ 2 & 1 & 0 & 0 & 3 & 0 & 0 & 4 & 0 & 0 & -6 & 5 \\ 0 & 0 & 1 & -2 & 4 & 0 & 0 & -3 & 5 & 6 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & -4 & 3 & 0 & 6 & -5 & 0 & 0 \\ 7 & 0 & -8 & 0 & 9 & 0 & 0 & -10 & 0 & -11 & 0 & -12 \\ 0 & 7 & 0 & 8 & 0 & 9 & -10 & 0 & 11 & 0 & -12 & 0 \\ 8 & 0 & 7 & 0 & 0 & 10 & 9 & 0 & 0 & 12 & 0 & -11 \\ 0 & -8 & 0 & 7 & 10 & 0 & 0 & 9 & 12 & 0 & 11 & 0 \end{bmatrix}. \end{aligned}$$

Then, we obtain

$$\varphi(\boldsymbol{y}) = \begin{bmatrix} 1 & -2 & 0 & 0 & 0 & -3 & -4 & 0 & 0 & 0 & -5 & -6 \\ 2 & 1 & 0 & 0 & 3 & 0 & 0 & 4 & 0 & 0 & -6 & 5 \\ 0 & 0 & 1 & -2 & 4 & 0 & 0 & -3 & 5 & 6 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & -4 & 3 & 0 & 6 & -5 & 0 & 0 \\ 7 & 0 & -8 & 0 & 9 & 0 & 0 & -10 & 0 & -11 & 0 & -12 \\ 0 & 7 & 0 & 8 & 0 & 9 & -10 & 0 & 11 & 0 & -12 & 0 \\ 8 & 0 & 7 & 0 & 0 & 10 & 9 & 0 & 0 & 12 & 0 & -11 \\ 0 & -8 & 0 & 7 & 10 & 0 & 0 & 9 & 12 & 0 & 11 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \end{bmatrix} = \begin{bmatrix} -176 \\ 45 \\ 96 \\ 11 \\ -306 \\ -3 \\ 140 \\ 363 \end{bmatrix},$$

which corresponds to a row scan of the elements of the quaternion-valued vector $\boldsymbol{y}$.

## IV. Vector-Valued Neural Networks (V-Nets)

Vector-valued neural networks (V-nets) are artificial neural networks conceived to process arrays of vectors. Let us begin addressing dense layers of neurons defined on a finite-dimensional algebra $\mathbb{V}$.

### *A. Dense Layers and the Approximation Capability of the Vector-Valued Multilayer Perceptron*

Dense layers, also known as fully connected layers, are the building blocks of several neural network architectures [3]. In particular, the famous multilayer perceptron (MLP) network is given by the composition of a sequence of dense layers [4].

Dense layers are composed of several parallel neurons, each receiving inputs through synaptic connections. Each neuron processes data through a linear combination of its inputs by the synaptic weights (trainable parameters), to which a scalar bias term is added. A non-linear activation function can be applied to yield the neuron's output. In mathematical terms, the output of the $i$th vector-valued neuron in a dense layer is defined by

$$y_i = \psi\left(s_i + b_i\right) \quad \text{with} \quad s_i = \sum_{j=1}^{N} w_{ij} x_j, \quad \forall i = 1, \ldots, M,$$

where $x_1, \ldots, x_N \in \mathbb{V}$ are the vector-valued inputs, $w_{ij} \in \mathbb{V}$ represents the synaptic weighted from input $j$ to neuron $i$, $b_i \in \mathbb{V}$ denotes the bias, and $\psi : \mathbb{V} \to \mathbb{V}$ is a vector-valued activation function.

Because $\mathbb{R}$ is a one-dimensional vector space, traditional and vector-valued dense layers are equivalent when $dim(\mathbb{V}) = 1$.

**Remark 3.** Because an algebra $\mathbb{V}$ may not be commutative, a dense layer can be alternatively defined as follows:

$$y_i = \psi\left(s_i + b_i\right) \quad \text{with} \quad s_i = \sum_{j=1}^{N} x_j w_{ji}, \quad \forall i = 1, \ldots, M.$$

Choosing the appropriate activation function for an image or signal-processing task can prove to be a challenging issue. To keep things simple, this paper focuses only on the so-called *split activation functions* [19], [12]. Briefly, a split activation function is obtained by applying a real-valued function in each coordinate of its vector-valued argument. Formally, we have:

**Definition 5** (Split Activation Functions)**.** *Let $\mathcal{E} = \{e_1, \ldots, e_n\}$ be an ordered basis for a vector space $\mathbb{V}$. A split activation function $\psi : \mathbb{V} \to \mathbb{V}$ is derived from a real-valued function $\psi_{\mathbb{R}} : \mathbb{R} \to \mathbb{R}$ as follows:*

$$\psi(x) = \sum_{i=1}^{n} \psi_{\mathbb{R}}(x_i) e_i, \quad \forall x = \sum_{i=1}^{n} x_i e_i \in \mathbb{V}. \tag{12}$$

**Remark 4.** A split-activation function given by (12) satisfies the identity

$$\varphi\big(\psi(x)\big) = \psi_{\mathbb{R}}\big(\varphi(x)\big), \tag{13}$$

where $\varphi$ is the isomorphism defined by (1) and $\psi_{\mathbb{R}}$ is computed component-wise. Recall that, in practice, $x \in \mathbb{V}$ is identified by $\varphi(x)$. Thus, (13) is useful as it shows that the representation $\varphi(\psi(x))$ of $\psi(x)$ can be found by evaluating $\psi_{\mathbb{R}}$ on $\varphi(x)$, which represents $x$.

The split `relu` and split sigmoid functions are examples of vector-valued activation functions used in V-nets, including hypercomplex-valued neural networks [14].

Despite being computationally expensive due to its numerous parameters, dense layers are widely used because they result in the universal approximation theorem. Briefly, the universal approximation theorem asserts that MLP networks can approximate continuous functions with any desired accuracy on a compact. More specifically, the set of single hidden-layer MLP networks with an appropriate activation function is dense in the set of all continuous functions over a compact. In mathematical terms, the universal approximation theorem for real-valued MLP networks poses conditions on the activation function $\psi : \mathbb{R} \to \mathbb{R}$ such that the set

$$\mathcal{H}_{\mathbb{R}} = \left\{ \sum_{i=1}^{M} \alpha_i \psi \left( \sum_{j=1}^{N} w_{ij} x_j + b_i \right) : M \in \mathbb{N}, \alpha_i, w_{ij}, b_i \in \mathbb{R} \right\},$$

of all single hidden-layer networks is dense on the set $\mathcal{C}(K)$ of all continuous functions over a compact $K \subseteq \mathbb{R}^N$ [20], [21], [22]. Many currently used activation functions like the `relu` and the logistic functions result in the universal approximation capability. A comprehensive review of the approximation capability of traditional MLP networks can be found in [23].

Recently, many researchers addressed the approximation capabilities of neural networks, including deep and shallow models based on piece-wise linear activation functions [24]. Moreover, the approximation capability of hypercomplex-valued neural networks has been investigated by several researchers, including Arena et al., [19], [25], Buchholz and Sommer [26], [27], and more recently [15], [28]. More generally, based on Arena et al. [19] and Vital et al. [15], we can state the following theorem concerning vector-valued multilayer perceptron (V-MLP) networks:

**Theorem 1** (Universal Approximation Theorem for V-MLP Networks)**.** *Let $\psi_{\mathbb{R}} : \mathbb{R} \to \mathbb{R}$ be a real-valued activation function that yields approximation capability to the set $\mathcal{H}_{\mathbb{R}}$ of all single hidden-layer MLP networks and satisfies $\lim_{t \to -\infty} \psi_{\mathbb{R}}(t) = 0$. Consider a finite-dimensional non-degenerate algebra $\mathbb{V}$, let $\psi : \mathbb{V} \to \mathbb{V}$ be the split activation function derived from $\psi_{\mathbb{R}}$, and let $K \subset \mathbb{V}^N$ be a compact set. The class of single hidden-layer V-MLP networks with a real-valued dense output layer given by*

$$\mathcal{H}_{\mathbb{V}} = \left\{ \sum_{i=1}^{M} \alpha_i \psi \left( \sum_{j=1}^{N} w_{ij} x_j + b_i \right) : M \in \mathbb{N}, \alpha_i \in \mathbb{R}, w_{ij}, b_i \in \mathbb{V} \right\}, \tag{14}$$

*is dense in the set $\mathcal{C}(K)$ of all continuous functions from $K$ to $\mathbb{V}$. Furthermore, if $\mathbb{V} \equiv \mathbb{H}$ is a hypercomplex algebra, then the class of single hidden-layer hypercomplex-valued MLP networks given by*

$$\mathcal{H}_{\mathbb{H}} = \left\{ \sum_{i=1}^{M} \alpha_i \psi \left( \sum_{j=1}^{N} w_{ij} x_j + b_i \right) : M \in \mathbb{N}, \alpha_i \in \mathbb{H}, w_{ij}, b_i \in \mathbb{H} \right\}, \tag{15}$$

*is dense in the set $\mathcal{C}(K)$.*

Theorem 1 says that V-MLP networks with split activation function inherit the approximation capability of a real-valued model if the algebra is non-degenerate. Furthermore, the split activation function must be derived from an activation function $\psi_{\mathbb{R}}$ such that $\lim_{t \to -\infty} \psi_{\mathbb{R}}(t) = 0$. In this case, given a continuous function $f : K \to \mathbb{V}$ and $\epsilon > 0$, Theorem 1 ensures that there exists a shallow V-MLP network $\mathcal{N} : \mathbb{V}^N \to \mathbb{V}$, with real-valued dense output layer, such that

$$|f(\boldsymbol{x}) - \mathcal{N}(\boldsymbol{x})| < \epsilon, \quad \forall \boldsymbol{x} \in K. \tag{16}$$

Note that the V-MLP network maps $\mathbb{V}^N$ into $\mathbb{V}$ despite the output layer being a real-valued dense layer. Moreover, the real-valued dense layer can be replaced by a vector-valued one if $\mathbb{V}$ is a hypercomplex algebra. In other words, a V-MLP of exclusively vector-valued dense layers has the universal approximation property if $\mathbb{V}$ is a finite-dimensional non-degenerate algebra with identity.

### *B. Relationship Between Real and Vector-Valued Dense Layers*

This subsection discusses the relationship between traditional and vector-valued dense layers. To simplify the exposition, we formulate a dense layer using matrix operations.

From a computational point of view, dense layers are efficiently implemented using matrix and vector operations. Accordingly, the output $\boldsymbol{y} = (y_1, \ldots, y_M) \in \mathbb{V}^M$ of a dense layer with $M$ vector-valued neurons in parallel is determined by the equation

$$\boldsymbol{y} = \boldsymbol{\psi}(\boldsymbol{s} + \boldsymbol{b}) \quad \text{with} \quad \boldsymbol{s} = \boldsymbol{W}\boldsymbol{x}, \tag{17}$$

where $\boldsymbol{x} = (x_1, \ldots, x_N) \in \mathbb{V}^N$ is the input vector, $\boldsymbol{W} = (w_{ij}) \in \mathbb{V}^{M \times N}$ is the matrix containing the synaptic weights, $\boldsymbol{b} = (b_1, \ldots, b_M) \in \mathbb{V}^M$ is the bias vector, and $\boldsymbol{\psi} : \mathbb{V}^M \to \mathbb{V}^M$ is defined in a component-wise manner by means of the following equation for some $\psi : \mathbb{V} \to \mathbb{V}$:

$$[\boldsymbol{\psi}(\boldsymbol{s})]_i = \psi(s_i), \quad \forall i = 1, \ldots, M.$$

In practice, we usually work with the real-valued representation of the inputs and outputs because current deep-learning libraries operate almost exclusively with floating-point numbers. Precisely, using the isomorphism $\varphi$ given by (10), we consider $\varphi(\boldsymbol{x}) \in \mathbb{R}^{nN}$ and $\varphi(\boldsymbol{y}) \in \mathbb{R}^{nM}$ instead of $\boldsymbol{x} \in \mathbb{V}^N$ and $\boldsymbol{y} \in \mathbb{V}^M$, respectively. Now, from (8) and (13), a vector-valued dense layer given by (17) can be emulated by a real-valued dense layer defined by

$$\varphi(\boldsymbol{y}) = \boldsymbol{\psi}_{\mathbb{R}}\big(\varphi(\boldsymbol{s}) + \varphi(\boldsymbol{b})\big) \quad \text{with} \quad \varphi(\boldsymbol{s}) = \mathcal{M}_L(\boldsymbol{W})\varphi(\boldsymbol{x}), \tag{18}$$

where $\varphi(\boldsymbol{x}) \in \mathbb{R}^{nN}$ is a real-valued input vector, $\mathcal{M}_L(\boldsymbol{W}) \in \mathbb{R}^{nM \times nN}$ is a real-valued synaptic weight matrix, $\varphi(\boldsymbol{b}) \in \mathbb{R}^{nM}$ is a real-valued bias vector, and $\varphi(\boldsymbol{y}) \in \mathbb{R}^{nM}$ is the real-valued output. In other words, a vector-valued dense layer can be computed using a real-valued dense layer by appropriately rearranging the elements of the vector-valued arrays. In particular, from (11), we have

$$\mathcal{M}_L(\boldsymbol{W}) = \sum_{k=1}^{n} \boldsymbol{W}_k \otimes \boldsymbol{P}_{k:}^T,$$

where the vector-value matrix is written as $\boldsymbol{W} = \boldsymbol{W}_1 e_2 + \ldots + \boldsymbol{W}_n e_k$ using the basis $\mathcal{E} = \{e_1, \ldots, e_n\}$ and the matrices $\boldsymbol{P}_{1:}^T, \ldots, \boldsymbol{P}_{:n}^T$ depends on the algebra. Therefore, the real-valued matrix $\mathcal{M}_L(\boldsymbol{W})$ associated with a vector-valued dense layer has $n(MN)$ distinct trainable parameters. In contrast, the synaptic weight matrix of a traditional dense layer has $n^2 MN$ trainable parameters. Including the bias vector, we conclude that a vector-valued dense layer has $nM(N+1)$ parameters while an equivalent traditional dense layer has $nM(nN+1)$ parameters. Thus, vector-valued dense layers can be interpreted as constrained versions of traditional dense layers, where the synaptic weights are obtained by imposing a structure that depends on the algebra. Furthermore, the constraints imposed by the algebra follow from supposing that intercorrelations exist between the feature channels.

Finally, in certain applications, the network may require the output of real-valued vectors instead of vector-valued arrays, even though the input is vector-valued because of the intercorrelation between the feature channels. For example, the real part of the output of hypercomplex-valued neural networks has been for acute lymphoblastic leukemia detection through blood smear digital microscopic images in [6]. One can handle such scenarios by focusing on a single component of a vector-valued output. Precisely, the output of a vector-valued dense layer can be written as $\boldsymbol{y} = \sum_{k=1}^{n} \boldsymbol{y}_k e_k \in \mathbb{V}^M$, where $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ are all real-vectors in $\mathbb{R}^M$. Thus, one can consider the component $\boldsymbol{y}_k \in \mathbb{R}^M$, for some $k \in \{1, \ldots, n\}$, as the real-valued output of the network. However, the output component $\boldsymbol{y}_k$ of a vector-valued dense layer with split activation function is equivalent to the output of a real-valued dense layer applied on a flattened version of the input. Indeed, using a split activation function, we conclude from (17) that $\boldsymbol{y}_k = \psi_{\mathbb{R}}(\boldsymbol{s}_k + \boldsymbol{b}_k)$, where $\boldsymbol{s} = \sum_{k=1}^{n} \boldsymbol{s}_k e_k$ and $\boldsymbol{b} = \sum_{k=1}^{n} \boldsymbol{b}_k e_k$. Now, let $s_{ki}$ denote the $i$th entry of $\boldsymbol{s}_k$. From Proposition 1, we have

$$s_{ki} = \sum_{j=1}^{N} \mathcal{B}_k(w_{ij}, x_j) = \sum_{j=1}^{N} \varphi(w_{ij})^T \boldsymbol{B}_k \varphi(x_j) = \sum_{j=1}^{N} \hat{\boldsymbol{w}}_{ij} \varphi(x_j), \quad \forall i = 1, \ldots, M, \tag{19}$$

where $\hat{\boldsymbol{w}}_{ij} = \varphi(w_{ij})^T \boldsymbol{B}_k \in \mathbb{R}^{1 \times n}$ is a row vector for all $i$ and $j$. Using matrix notation, (19) can be written as $\boldsymbol{s}_k = \hat{\boldsymbol{W}} \varphi(\boldsymbol{x})$ where

$$\hat{\boldsymbol{W}} = \begin{bmatrix} \hat{\boldsymbol{w}}_{11} & \ldots & \hat{\boldsymbol{w}}_{1N} \\ \vdots & \ddots & \vdots \\ \hat{\boldsymbol{w}}_{M1} & \ldots & \hat{\boldsymbol{w}}_{MN} \end{bmatrix} \in \mathbb{R}^{M \times (nN)}.$$

Concluding, the $k$th component of the vector-valued output $\boldsymbol{y} \in \mathbb{V}^M$ satisfies $\boldsymbol{y}_k = \psi_{\mathbb{R}}(\hat{\boldsymbol{W}} \varphi(\boldsymbol{x}) + \boldsymbol{b}_k)$, which means $\boldsymbol{y}_k$ is the output of a real-valued dense layer computed on the flattened version $\varphi(\boldsymbol{x}) \in \mathbb{R}^{nN}$ of the input $\boldsymbol{x} \in \mathbb{V}^N$.

### *C. Vector-valued Convolutions and Some Remarks on Other Building Blocks*

Convolutional layers are important building blocks in current deep learning models. Broadly speaking, convolutional layers are special layers in which the units are connected to small sections in the feature maps of the preceding layer through a set of weights called a filter bank [1]. Moreover, all units share the same filter banks. Besides reducing significantly the number of parameters, convolutional layers exhibit spatial invariance and are effective for the detection of local patterns.

A vector-valued convolutional layer is defined as follows. Let $\boldsymbol{x}$ be the input (image or signal) with $C$ feature channels. We denote by $\boldsymbol{x}(p,c) \in \mathbb{V}$ the content of $\boldsymbol{x}$ in the $c$th channel at location $p \in D_{\boldsymbol{x}}$, where $\mathcal{D}_{\boldsymbol{x}}$ denotes the domain of $\boldsymbol{x}$. The weights of a convolutional layer with $K$ filters are arranged in an array $\mathbf{W}$ such that $\mathbf{W}(q,c,k) \in \mathbb{V}$ corresponds to the weight of the $k$th filter in the $c$th channel at $q \in D$, where $D$ denotes the filters' domain[2]. The vector-valued convolution of $\mathbf{W}$ and $\boldsymbol{x}$, denoted by $\mathbf{W} * \boldsymbol{x}$, is given by the sum of the cross-correlation of $\mathbf{W}(:,c,k)$ and $\boldsymbol{x}(:,c)$ over all channels $c = 1,\ldots,C$. Precisely, we have

$$(\mathbf{W} * \boldsymbol{x})(p,k) = \sum_{c=1}^{C} \sum_{q \in D} \mathbf{W}(q,c,k)\boldsymbol{x}(p+S(q),c), \quad p \in \mathcal{D}_{\boldsymbol{y}},\ \forall k = 1,\ldots,K, \tag{20}$$

where $p+S(q)$ denotes a translation that can take strides into account and $\mathcal{D}_{\boldsymbol{y}}$ denotes the domain of $\boldsymbol{y}$. The output $\boldsymbol{y}$ of a convolutional layer is obtained by evaluating an activation function on the addition of a bias term $\boldsymbol{b}$ with the convolution $\mathbf{W} * \boldsymbol{x}$. In mathematical terms, we have $\boldsymbol{y} = \psi(\mathbf{W} * \boldsymbol{x} + \boldsymbol{b})$, where the activation function $\psi : \mathbb{V} \to \mathbb{V}$ is applied in an entry-wise manner.

We would like to remark that a traditional convolutional layer is obtained when $\mathbf{W}(q,c,k)$ and $\boldsymbol{x}(p,c)$ are real numbers instead of vectors. In this case, the convolution $\mathbf{W} * \boldsymbol{x}$ is obtained by summing over all the real-valued feature channels. In contrast, the vector-valued convolution operates under the assumption that there is an intercorrelation between $n$ feature channels, which is achieved through the use of vector multiplication in (20).

Like dense layers, vector-valued convolutional layers can be emulated using real-valued convolutional layers, which is particularly useful for implementing convolutional neural networks using current deep learning libraries. Accordingly, using the isomorphism $\varphi : \mathbb{V} \to \mathbb{R}^n$, the Kronecker product, and the linearity of the convolution operation, we have

$$\varphi(\mathbf{W} * \boldsymbol{x}) = \left( \sum_{\ell=1}^{n} \mathbf{W}_\ell \otimes \mathbf{P}_{\ell:}^T \right) * \varphi(\boldsymbol{x}) = \sum_{\ell=1}^{n} \left( \mathbf{M}_\ell * \varphi(\boldsymbol{x}) \right), \tag{21}$$

where $\mathbf{W} = \mathbf{W}_1 e_1 + \ldots + \mathbf{W}_n e_n$ is the representation of the filters with respect to the basis $\mathcal{E} = \{e_1,\ldots,e_n\}$, $\mathbf{M}_\ell = \mathbf{W}_\ell \otimes \mathbf{P}_{\ell:}^T$ are real-valued filters obtained using the Kronecker product, and $\varphi(\boldsymbol{x})$ is obtained concatenating the components $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n$ of $\boldsymbol{x} = \boldsymbol{x}_1 e_1 + \ldots + \boldsymbol{x}_n e_n$ into the feature axis. Examples of the vector-valued convolutions and their implementation can be found in [6], [7], [29]. We would like to remark that split activation functions are particularly helpful in the emulation of vector-valued convolutional layers by real-valued ones. Indeed, from (13), the concatenation $\varphi(\boldsymbol{y})$ of the output

[2] For example, the domain $D$ is usually a rectangular grid in image processing tasks.

$\boldsymbol{y}$ is obtained by evaluating a real-valued activation function $\psi_{\mathbb{R}}$ entry-wise in the sum $\varphi(\mathbf{W} * \boldsymbol{x}) + \varphi(\boldsymbol{b})$, with $\varphi(\mathbf{W} * \boldsymbol{x})$ given by (21).

In addition to convolutional layers, modern deep learning models utilize other structures, such as pooling layers and batch normalization [3]. While vector-valued versions of these structures are a topic of future research, we can currently use a simpler approach. This approach involves combining traditional structures with real-valued emulation of vector-valued convolutional and dense layers, as described in equations (18) and (21). Although this approach may seem overly simplistic, it can still provide valuable insights into the vector-valued blocks. For example, when a max-pooling layer is applied to the real-valued representation of a vector-valued image, the result is equivalent to computing the max-pooling with the maximum given by the so-called marginal or Cartesian product ordering of vectors [30]. Additionally, this pooling layer corresponds to extending the pooling layer in a split manner, as shown in equation (12).

## V. Concluding Remarks

Despite the many successful applications of traditional neural networks for signal and image processing tasks, they do not initially take into account the intercorrelation between feature channels. Such intercorrelation is learned from the dataset, but it demands careful consideration when selecting optimization methods and hyperparameters, as well as using appropriate regularization strategies. In contrast, vector-valued neural networks (V-nets) naturally incorporate the intercorrelation between feature channels through the vector algebra. Furthermore, using hypercomplex algebras can lead to additional geometric and algebraic properties in the network model.

This paper provided the basic concepts of V-nets. We began reviewing the concept of algebra, which is obtained by enriching a vector space with a multiplication. We defined a hypercomplex algebra as an algebra with an identity. Therefore, hypercomplex-valued neural networks are V-nets with additional geometric or algebraic properties. In particular, from Example 2, we conclude that the recently introduced parameterized “hypercomplex” neural networks are, in fact, V-nets [7], [16]. In addition, this paper establishes the relationship between V-nets and traditional neural networks. Precisely, we showed how vector-valued dense and convolutional layers can be emulated using real-valued ones. Such emulation is particularly helpful for the implementation of V-nets using current deep-learning libraries like `tensorflow` and `pytorch`. Besides making the implementation of V-nets straightforward, we can utilize automatic differentiating features to train V-nets without having to deal with complicated vector or hypercomplex calculus for the gradients. Moreover, other traditional building blocks can be combined with vector-valued convolutional and dense. Such a straightforward approach yielded promising results in image and signal processing applications [6], [7], [12]. Besides the practical considerations, this paper also addressed important theoretical issues. Namely, Theorem 1 concerns the approximation capability of vector-valued multilayer perceptron defined on finite-dimensional non-degenerate algebras.

Concluding, V-nets are obtained by imposing certain intercorrelation between the features. As a consequence, they provide a graceful approach to the bias-variance trade-off by incorporating into the neural network's operation additional characteristics of the data. Future research should focus on selecting the appropriate algebra for image and signal processing applications. Efficient techniques for learning the most suited algebra for a task also require further research. Furthermore, vector-valued activation functions beyond the split functions, as well as deep-learning building blocks and tools like batch normalization and dropout strategies, should be further investigated in the future.

## Acknowledgment

I would like to acknowledge that this paper is partially based on a minicourse I held at the Institute for Research and Applications of Fuzzy Modeling, University of Ostrava in the Czech Republic in February 2023. This minicourse was part of the Strategic Research Partnership called ”Mathematical Aspects of Complex, Hypercomplex, and Fuzzy Neural Networks,” which was supported by the Polish National Agency for Academic Exchange. I am grateful to Agnieszka Niemczynowicz, the partnership coordinator, and the project collaborators for their valuable insights and discussions.

## References

[1] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 5 2015.

[2] J. Schmidhuber, “Deep Learning in neural networks: An overview,” pp. 85–117, 1 2015.

[3] Aurelien Géron, *Hands–On Machine Learning with Scikit–Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed. Sebastopol, California, USA.: O Reilly, 10 2019.

[4] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2009.

[5] G. Vieira and M. E. Valle, “A general framework for hypercomplex-valued extreme learning machines,” *Journal of Computational Mathematics and Data Science*, vol. 3, p. 100032, 6 2022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2772415822000062

[6] ——, “Acute Lymphoblastic Leukemia Detection Using Hypercomplex-Valued Convolutional Neural Networks,” in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 7 2022, pp. 1–8.

[7] E. Grassucci, A. Zhang, and D. Comminiello, “PHNNs: Lightweight Neural Networks via Parameterized Hypercomplex Convolutions,” *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13, 10 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9983846/

[8] E. Grassucci, G. Mancini, C. Brignone, A. Uncini, and D. Comminiello, “Dual quaternion ambisonics array for six-degree-of-freedom acoustic representation,” *Pattern Recognition Letters*, vol. 166, pp. 24–30, 2 2023.

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

[10] F. Catoni, D. Boccaletti, R. Cannata, V. Catoni, E. Nichelatti, and P. Zampetti, *The Mathematics of Minkowski Space-Time*. Birkhäuser Basel, 2008.

[11] R. Schafer, *An Introduction to Nonassociative Algebras*. Project Gutenberg, 1961. [Online]. Available: https://www.gutenberg.org/ebooks/25156

[12] T. Parcollet, M. Morchid, and G. Linarès, "A survey of quaternion neural networks," *Artificial Intelligence Review*, vol. 53, no. 4, pp. 2957–2982, 4 2020.

[13] I. N. Aizenberg, *Complex-Valued Neural Networks with Multi-Valued Neurons*, 1st ed., ser. Studies in Computational Intelligence, J. Kacprzyk, Ed. Berlin Heidelberg: Springer, 2011, vol. 353.

[14] A. Hirose, *Complex-Valued Neural Networks*, 2nd ed., ser. Studies in Computational Intelligence. Heidelberg, Germany: Springer, 2012.

[15] W. L. Vital, G. Vieira, and M. E. Valle, "Extending the universal approximation theorem for a broad class of hypercomplex-valued neural networks," in *Intelligent Systems*, J. C. Xavier-Junior and R. A. Rios, Eds. Cham: Springer International Publishing, 2022, pp. 646–660.

[16] A. Zhang, Y. Tay, S. Zhang, A. Chan, A. T. Luu, S. C. Hui, and J. Fu, "Beyond fully-connected layers with quaternions: Parameterization of hypercomplex multiplications with 1/n parameters," *CoRR*, vol. abs/2102.08597, 2021. [Online]. Available: https://arxiv.org/abs/2102.08597

[17] F. Stenger, "Kronecker Product Extensions of Linear Operators," *SIAM Journal on Numerical Analysis*, vol. 5, no. 2, pp. 422–435, 6 1968.

[18] C. F. Loan, "The ubiquitous Kronecker product," *Journal of Computational and Applied Mathematics*, vol. 123, no. 1-2, pp. 85–100, 11 2000.

[19] P. Arena, L. Fortuna, G. Muscato, and M. G. Xibilia, "Multilayer perceptrons to approximate quaternion valued functions," *Neural Networks*, vol. 10, no. 2, pp. 335–342, 3 1997.

[20] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems 1989 2:4*, vol. 2, no. 4, pp. 303–314, 12 1989. [Online]. Available: https://link.springer.com/article/10.1007/BF02551274

[21] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1 1991.

[22] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, no. 6, pp. 861–867, 1 1993.

[23] A. Pinkus, "Approximation theory of the MLP model in neural networks," *Acta Numerica*, vol. 8, pp. 143–195, 1 1999.

[24] P. Petersen and F. Voigtlaender, "Optimal approximation of piecewise smooth functions using deep ReLU neural networks," *Neural Networks*, vol. 108, pp. 296–330, 12 2018.

[25] P. Arena, L. Fortuna, G. Muscato, and M. G. Xibilia, "Quaternion algebra," in *Neural Networks in Multidimensional Domains*, ser. Lecture Notes in Control and Information Sciences. Springer London, 1998, vol. 234, pp. 43–47.

[26] S. Buchholz and G. Sommer, "Hyperbolic Multilayer Perceptron," *Proceedings of the International Joint Conference on Neural Networks*, vol. 2, pp. 129–133, 2000.

[27] ——, "Clifford Algebra Multilayer Perceptrons," in *Geometric Computing with Clifford Algebras*. Springer Berlin Heidelberg, 2001, pp. 315–334.

[28] F. Voigtlaender, "The universal approximation theorem for complex-valued neural networks," *Applied and Computational Harmonic Analysis*, vol. 64, pp. 33–61, 5 2023.

[29] C. J. Gaudet and A. S. Maida, "Deep Quaternion Networks," *Proceedings of the International Joint Conference on Neural Networks*, vol. 2018-July, 10 2018.

[30] E. Aptoula and S. Lefèvre, “On the morphological processing of hue,” *Image and Vision Computing*, vol. 27, no. 9, pp. 1394–1401, 2009.