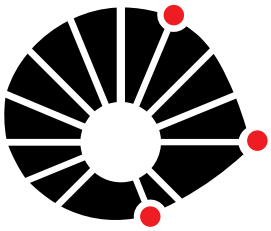

UNICAMP

UNIVERSIDADE ESTADUAL DE CAMPINAS

Instituto de Matemática, Estatística e Computação Científica

GUILHERME VIEIRA NETO

Hypercomplex-Valued Feedforward Neural Networks

Redes Neurais Progressivas com Valores Hipercomplexos

Campinas

2023

Guilherme Vieira Neto

Hypercomplex-Valued Feedforward Neural Networks

Redes Neurais Progressivas com Valores Hipercomplexos

Tese apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Doutor em Matemática Aplicada.

Thesis presented to the Institute of Mathematics, Statistics and Scientific Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Applied Mathematics.

Supervisor: Marcos Eduardo Ribeiro do Valle Mesquita

Este trabalho corresponde à versão final da Tese defendida pelo aluno Guilherme Vieira Neto e orientada pelo Prof. Dr. Marcos Eduardo Ribeiro do Valle Mesquita.

Campinas

2023

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

V673h

Vieira Neto, Guilherme, 1992-
Hypercomplex-valued feedforward neural networks / Guilherme Vieira Neto. – Campinas, SP : [s.n.], 2023.

Orientador: Marcos Eduardo Ribeiro do Valle Mesquita.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1. Redes neurais (Computação). 2. Inteligência artificial. 3. Quatérnios. 4. Álgebra geométrica. 5. Aprendizado de máquina. I. Mesquita, Marcos Eduardo Ribeiro do Valle, 1979-. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. III. Título.

Informações Complementares

Título em outro idioma: Redes neurais progressivas com valores hipercomplexos
Palavras-chave em inglês:
Neural networks (Computer science)
Artificial intelligence
Quaternions
Geometric algebra
Machine learning
Área de concentração: Matemática Aplicada
Titulação: Doutor em Matemática Aplicada
Banca examinadora:
Marcos Eduardo Ribeiro do Valle Mesquita [Orientador]
João Batista Florindo
Carlile Campos Lavor
João Paulo Papa
Leandro Augusto Frata Fernandes
Data de defesa: 24-03-2023
Programa de Pós-Graduação: Matemática Aplicada

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0003-3361-6154
- Currículo Lattes do autor: http://lattes.cnpq.br/4513901014974073

Tese de Doutorado defendida em 24 de março de 2023 e aprovada

pela banca examinadora composta pelos Profs. Drs.

Prof(a). Dr(a). MARCOS EDUARDO RIBEIRO DO VALLE MESQUITA

Prof(a). Dr(a). JOÃO BATISTA FLORINDO

Prof(a). Dr(a). CARLILE CAMPOS LAVOR

Prof(a). Dr(a). JOÃO PAULO PAPA

Prof(a). Dr(a). LEANDRO AUGUSTO FRATA FERNANDES

A Ata da Defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria de Pós-Graduação do Instituto de Matemática, Estatística e Computação Científica.

Acknowledgements

Aos meus pais, Angela e Guilherme, que nunca deixaram faltar carinho, amor, e confiança. Sempre se esforçaram para me ensinar que não é necessário entender para apoiar.

Aos amigos, novos e antigos, que participaram, cada um à sua medida, dessa jornada transformadora. Certamente sou grato a todos e todas de coração. Em especial, exalto aqueles que estavam lá nos momentos de luta mas também nos momentos de colher as recompensas: Danilo Seghese, Juliana Florêncio, Thiago Savoy, Lucca Santos, Mayara Sebinelli, Ana Cláudia Oliveira, e Thais Montanari, que compreendem de maneira visceral os esforços e os louros deste ofício.

Ao meu orientador, Marcos Eduardo Ribeiro do Valle Mesquita, sou muitíssimo grato pela exemplar orientação. Sua prontidão, empenho e compreensão são admiráveis, e me proporcionaram a confiança necessária para realizar os trabalhos aqui compilados.

A todos os professores que contribuíram para a minha formação na Unicamp. Ao Professor Danilo Comminiello e ao grupo de pesquisa ISPAMM da *La Sapienza Università di Roma*, que me receberam de braços abertos e agregaram muito à minha formação.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

"All we have to decide is what to do with the time that is given to us."
Gandalf, *O Senhor dos Anéis*

Resumo

Em décadas recentes redes neurais artificiais (RNAs) ganharam notável popularidade. Este tipo de modelo de inteligência artificial se ramificou através das décadas, alcançando uma grande variedade de áreas desde aplicações na indústria como processamento de sinais digitais, robótica, automação, previsão de séries temporais, assistência a diagnósticos médicos, reconhecimento de padrões e tarefas de classificação, até aplicações diretamente em contato com o usuário final tais como smartphones, sistemas de recomendação, reconhecimento facial e de voz e dispositivos interativos no geral. No entanto, a maioria esmagadora de aplicações de RNAs se baseia em modelos com valores reais. Nesta tese, exploramos dois tipos de RNAs baseadas em álgebras hipercomplexas, a saber as máquinas de aprendizado extremo (ELMs, do inglês *Extreme Learning Machines*) e as redes neurais convolucionais (CNNs, do inglês *Convolutional Neural Networks*). Discutimos detalhadamente seus principais aspectos arquiteturais e analisamos como estes modelos se comparam a modelos reais equivalentes com respeito a três aspectos centrais: armazenamento, desempenho e tempo de processamento. Em particular, implementamos ELMs com número similar de parâmetros treináveis e os modelos baseados em álgebras hipercomplexas mostram resultados expressivamente superiores aos do modelo real tanto numa tarefa de previsão de série temporal quanto em uma tarefa de *auto-encoding* de imagens coloridas. Já com relação às CNNs, desenvolvemos modelos compactos com valores hipercomplexos contendo aproximadamente 33% dos parâmetros treináveis contidos em um modelo real, e mostramos que estes ainda assim tem um desempenho muito superior ao modelo real em duas tarefas de classificação de imagens médicas aqui apresentadas. Ademais, exploramos estes modelos em uma gama variada de álgebras, desde as mais usuais como os quatérnios e os tessarinos até as mais não-usuais como álgebras de Cayley-Dickson e os quatérnios hiperbólicos de MacFarlane. Concluímos que na maioria das vezes uma álgebra não-usual mostra desempenho consistentemente superior aos quatérnios, apesar desta última dominar expressivamente a literatura em RNAs com valores hipercomplexos com respeito ao número de publicações. Ainda, mostramos na prática que a codificação de cores de uma imagem afeta cada modelo diferentemente, e discutimos as implicações deste fato na codificação de dados no geral. Por fim, incluimos uma aplicação de um modelo hipercomplexo baseado na álgebra dos quatérnios duais que demonstra sua equivariância de translação e a maneira com a qual este modelo trata conjuntos de testes transladados sem a necessidade de intervenção de um especialista. Discutimos brevemente o potencial desta álgebra em aplicações do mundo real. Em suma, operações nos modelos com valores hipercomplexos exibem custo computacional mais elevado que nos modelos reais, porém os modelos finais obtidos são mais compactos e tem desempenho consistentemente superior em todas as métricas avaliadas.

Palavras-chave: Redes neurais artificiais, álgebras hipercomplexas, álgebras geométricas, aprendizado de máquinas.

Abstract

Over the past few decades artificial neural networks (ANNs) rose notably in popularity. This type of artificial intelligence model branched out, reaching a wide variety of areas from industry applications such as digital signal processing, robotics and automation, time series forecasting, medical diagnosis assistance, pattern recognition and classification tasks, all the way to end-user applications such as smartphones, recommendation systems, face and voice recognition and smart interactive devices in general. However, the dominant majority of ANN applications feature real-valued models. In this thesis we explore two ANN models based on hypercomplex algebras, namely the extreme learning machines (ELMs) and convolutional neural networks (CNNs). We explore the main architecture features in detail and show how these fair when compared to real-valued equivalent models regarding three key aspects: storage, performance and time. In particular, we build ELMs with similar number of free parameters and the hypercomplex-valued models dominate the real-valued one by a large margin on both a time series prediction task and a color image auto-encoding task. With respect to CNNs, we showcase compact hypercomplex-valued models featuring slightly over 33% of the free parameters in the real-valued model, yet the former heavily outperforms the latter on two classification tasks involving medical images. Moreover, we explore said models on a wide variety of algebras, ranging from usual ones such as quaternions and tessarines to unusual ones such as Cayley-Dickson algebras and MacFarlane's hyperbolic quaternions. We conclude that more often than not these unusual algebras fair consistently better than the quaternions, despite the latter boasting an overwhelmingly larger number of works. Notwithstanding, we also showcase in practice how color encoding of images affects models differently, and briefly discuss the implications of data encoding. Lastly, we include an application of a hypercomplex-valued model based on the dual quaternions that showcases the translation equivariance property and how effortlessly this model predicts translated test sets without specialist intervention. We briefly discuss this potential in real-world applications. In sum, operations in the hypercomplex-valued models exhibit higher computational cost, but the final models obtained are more compact than the real-valued counterparts and perform better according to all evaluated metrics.

Keywords: Artificial neural networks, hypercomplex algebras, geometric algebras, machine learning.

Contents

1 Introduction

Artificial neural networks (ANNs) increasingly rose in popularity since the 1980s. In the 2000s, especially, ANNs skyrocketed in popularity due to the increased access to semiconductors, which greatly enhanced the computational capabilities of computers. Nowadays ANNs figure among the most used artificial intelligence models and are widely applied to many fields. To cite a few, ANNs are widely used in digital signal processing, robotics, automation, time series forecasting, pattern recognition, medical diagnosis assistance, all sorts of classification tasks, computer vision, art, social and demographic sciences, recommendation systems, face and voice recognition and smart interactive devices in general. In sum, ANNs made possible many new functions and systems, and showed that a myriad of tasks can be partially or entirely automated, reducing costs and time requirements while also often increasing performance. From its inception, however, ANNs have been developed both theoretically and practically on the real algebra. By making use of higher dimension objects such as hypercomplex numbers we are able to represent multiple channels of information in a more cohesive manner. Indeed, ANN models based on hypercomplex algebras allow us to consolidate multiple inputs regarding the same object and make use of correlation between said inputs.

Generally speaking a hypercomplex algebra is an algebra of dimension higher than one. The most well known example is the complex numbers algebra, of dimension 2, defined by introducing an imaginary part $\boldsymbol{i}$ to a real number and imposing $\boldsymbol{i}^2 = -1$. Hyperbolic numbers are defined in a similar manner but with $i^2 = 1$. Both these algebras are spanned by a parametric process known as Cayley-Dickson process, which generates algebras of doubling dimension and progressively incurs in loss of properties along the chain of generated algebras. Another notable family is the Clifford algebras, an infinite collection of associative algebras with dimensions in powers of 2. Higher dimensional hypercomplex algebras are usually represented by a table containing the product of each pair of imaginary units; these tables uniquely determine the algebra and express properties of the product such as commutativity, associativity, anti-commutativity, among others. In this work we approach ANNs based on several hypercomplex algebras with varying sets of properties.

Hypercomplex-valued ANNs (Hv-ANNs) are, as the name suggests, ANN models in which parameters are taken from a hypercomplex algebra, as opposed to real numbers. Common examples of Hv-ANNs include complex-valued (AIZENBERG, 2011; HIROSE, 2012), hyperbolic-valued (BUCHHOLZ; SOMMER, 2000; NITTA; BUCHHOLZ, 2008; NITTA; KUROE, 2018), and quaternion-valued neural networks (XIA et al., 2018; PARCOLLET; MORCHID; LINARÈS, 2019). Notably, using hypercomplex algebra inputs

and outputs allows for the compact representation of multiple information channels related to the same object in a single entity, such as phase-angle information for waves or multiple color channels of an image for example. On top of that, product operations in these algebras are more elaborate due to the fact that they consider cross-channel information. In general this leads to longer processing times and yields a much more efficient model in terms of performance and storage. Precisely, we observed this exact phenomenon in (VIEIRA; VALLE, 2020; VIEIRA; VALLE, 2022b).

With these in mind, our main objectives of investigation in this PhD were to

- investigate and propose extensions of neural network models and operations to hypercomplex algebras in a general manner;
- evaluate the processing of multichannel signals by hypercomplex models in comparison to real-valued equivalent models to gain insight on the advantages and identify potential shortcomings;
- strive to better understand the relationship between the underlying algebra of a model and its performance in a given application, investigating whether or not given properties provide an advantage in certain situations.

These objectives steered the course of my PhD studies and all the works I've collaborated with. In sum, the main thesis of this work is that there is an intricate relationship between the triad **model-algebra-problem**, and we are yet at the start of the process of unravelling the full extent of it.

During this PhD program we explored a few feedforward models on hypercomplex algebras. Using extreme learning machines (ELMs), also initially known as random vector functional links (RVFLs), we were able to implement a time series predictor on the Lorenz system as well as auto-encoders on the CIFAR-10 color image dataset (VIEIRA; VALLE, 2020), and rigorously formalized the methodology of such hypercomplex-valued ELMs (HvELMs) in (VIEIRA; VALLE, 2022b). While exploring hypercomplex-valued convolutional neural networks (HvCNNs), we performed a classification task on a real-world medical image dataset using light-weight CNNs in (VIEIRA; VALLE, 2022a) and developed an improved version of this work, presented at the International Conference of Advanced Computational Applications of Geometric Algebra (ICACGA 2022). In both these HvCNN works, the performance achieved by the implemented models rivaled state-of-the-art performances that rely on transfer learning and extremely large models, thus presenting undeniable advantages regarding processing times and storage space. Moreover, the codes used in the aforementioned works considered the multiplication rules of the hypercomplex algebra as an input and thus the implementation is rather general, being easily adaptable to any desired algebra. On top of these works, I participated in an exchange program at

the Department of Information Engineering, Electronics and Telecommunications of the Sapienza Univeristy of Rome during september and october of 2022. In the two months I spent there I collaborated with Prof. Danilo Comminiello and Eleonora Grassucci, as well as my advisor Marcos Eduardo Valle, during which we worked with dual quaternions' ability to model rigid motions to perform human pose forecasting. Lastly, despite not being the main topic of this PhD study we also worked on some recursive models such as the hypercomplex-valued Hopfield neural networks and vector-valued Hopfield neural network (GARIMELLA et al., 2022).

2 Basics Concepts on Hypercomplex Algebras

In this section we introduce the fundamental concepts of hypercomplex algebras. Hypercomplex algebras are broadly defined over an arbitrary field $\mathbb{F}$, but we exclusively consider real numbers as the ground field. The more general approach to hypercomplex algebra concepts can be found in the work by Kantor and Solodovnikov (KANTOR; SOLODOVNIKOV, 1989), which was the main inspiration for this construction.

A hypercomplex number x can be written in the form

$$x = x_0 + x_1\boldsymbol{i}_1 + \cdots + x_n\boldsymbol{i}_n, \tag{2.1}$$

where $x_0, x_1, \ldots, x_n \in \mathbb{R}$. The numbers $\boldsymbol{i}_1, \boldsymbol{i}_2, \ldots, \boldsymbol{i}_n$ are called hyperimaginary units (KANTOR; SOLODOVNIKOV, 1989; CASTRO; VALLE, 2020). We shall denote a set of hypercomplex numbers by $\mathbb{A}$. Then, by enriching a set of hypercomplex numbers $\mathbb{A}$ with addition and multiplication operations one obtains a hypercomplex algebras (KANTOR; SOLODOVNIKOV, 1989; CASTRO; VALLE, 2020).

The addition is carried out component-wise as

$$x + y = (x_0 + y_0) + (x_1 + y_1)\boldsymbol{i}_1 + \ldots + (x_n + y_n)\boldsymbol{i}_n, \tag{2.2}$$

for $x = x_0 + x_1\boldsymbol{i}_1 + \cdots + x_n\boldsymbol{i}_n$ and $y = y_0 + y_1\boldsymbol{i}_1 + \cdots + y_n\boldsymbol{i}_n$ in $\mathbb{A}$.

The multiplication, on the other hand, is performed distributively. By computing the terms we end up with products between hyperimaginary units, which are defined as

$$\boldsymbol{i}_i\boldsymbol{i}_j \equiv \mu_{ij} = (\mu_{ij})_0 + (\mu_{ij})_1\boldsymbol{i}_1 + \cdots + (\mu_{ij})_n\boldsymbol{i}_n \in \mathbb{A}, \tag{2.3}$$

for all $i, j = 1, \ldots, n$. Note that the μ_{ij} are hypercomplex numbers in $\mathbb{A}$ and are usually presented in the form of a multiplication table, as in Table 1. This is due to the fact that the μ_{ij}'s characterize the hypercomplex algebra, i.e., **an algebra $\mathbb{A}$ is uniquely determined by the product of its imaginary units and vice-versa**.

The multiplication of two hypercomplex numbers $x = x_0 + x_1\boldsymbol{i}_1 + \cdots + x_n\boldsymbol{i}_n$ and $y = y_0 + y_1\boldsymbol{i}_1 + \cdots + y_n\boldsymbol{i}_n$ is defined using the distributivity and replacing the products $(x_i\boldsymbol{i}_i)(y_j\boldsymbol{i}_j)$ by $x_iy_j\mu_{ij}$, for all $i, j = 1, \ldots, n$ using (2.3). Formally, we have

$$\begin{aligned} xy = &\left(x_0y_0 + \sum_{i,j=1}^{n} x_iy_j(\mu_{ij})_0\right) \\ &+ \left(x_0y_1 + x_1y_0 + \sum_{i,j=1}^{n} x_iy_j(\mu_{ij})_1\right)\boldsymbol{i}_1 + \ldots \\ &+ \left(x_0y_n + x_ny_0 + \sum_{i,j=1}^{n} x_iy_j(\mu_{ij})_n\right)\boldsymbol{i}_n. \end{aligned} \tag{2.4}$$

Table 1 – Multiplication tables of the complex numbers and quaternions, respectively.

$\mathbb{C}$	1	$\boldsymbol{i}$
1	1	$\boldsymbol{i}$
$\boldsymbol{i}$	$\boldsymbol{i}$	-1

$\mathbb{H}$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	-1	$\boldsymbol{k}$	$-\boldsymbol{j}$
$\boldsymbol{j}$	$-\boldsymbol{k}$	-1	$\boldsymbol{i}$
$\boldsymbol{k}$	$\boldsymbol{j}$	$-\boldsymbol{i}$	-1

As opposed to a field, hypercomplex algebra multiplication needs not to be associative, commutative, nor have any other notable algebraic properties.

We observe that a scalar $\alpha \in \mathbb{R}$ can be identified with the hypercomplex number $\alpha + 0\boldsymbol{i}_1 + \ldots + 0\boldsymbol{i}_n$. Moreover, the scalar product

$$\alpha x = \alpha x_0 + \alpha x_1 \boldsymbol{i}_1 + \cdots + \alpha x_n \boldsymbol{i}_n, \tag{2.5}$$

can be deduced from (2.4). This yields a canonical isomorphism between the hypercomplex algebra $\mathbb{A}$ and the vector space $\mathbb{R}^{n+1}$. Indeed, the canonical isomorphism $\varphi : \mathbb{A} \to \mathbb{R}^{n+1}$ is defined by a simple rearrangement operation as

$$\varphi(x) = \varphi(x_0 + x_1 \boldsymbol{i}_1 + \cdots + x_n \boldsymbol{i}_n) = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \tag{2.6}$$

Clearly φ is a linear operator. The inverse isomorphism φ^{-1} is merely the inverse rearrangement operation. Naturally, the canonical basis of $\mathbb{A}$ is defined as $\beta = \{1, \boldsymbol{i}_1, ..., \boldsymbol{i}_n\}$

Based on this isomorphism between an hypercomplex algebra $\mathbb{A}$ and the vector space $\mathbb{R}^{n+1}$ we can derive an expression of the multiplication of hypercomplex numbers using a real matrix-vector product. Formally, given a hypercomplex number $a \in \mathbb{A}$, the multiplication to the left by a is equivalent to a linear operator $\mathcal{A}_L : \mathbb{A} \to \mathbb{A}$ defined by $\mathcal{A}_L(x) = ax$, for all $x \in \mathbb{A}$. Hence, using the isomorphism φ, we see that

$$\varphi\left(\mathcal{A}_L(x)\right) = \Phi_L(a)\varphi(x), \tag{2.7}$$

where $\Phi_L : \mathbb{A} \to \mathbb{R}^{(n+1)\times(n+1)}$ is the matrix representation of $\mathcal{A}_L$ with respect to the canonical basis β (CATONI et al., 2005). This operator has the form

$$\Phi_L(a) = \begin{bmatrix} | & | & & | \\ \varphi(a) & \varphi(a\boldsymbol{i}_1) & \ldots & \varphi(a\boldsymbol{i}_n) \\ | & | & & | \end{bmatrix}. \tag{2.8}$$

We highlight that the terms $a\boldsymbol{i}_i$ that span the columns are products in $\mathbb{A}$ and, thus, the operator Φ_L carries information regarding the multiplication table of $\mathbb{A}$ and ultimately

regarding the algebra $\mathbb{A}$. Indeed, the identity

$$\Phi_L(a) = \begin{bmatrix} a_0 & \sum_i a_i(\mu_{i1})_0 & \dots & \sum_i a_i(\mu_{in})_0 \\ a_1 & a_0 + \sum_i a_i(\mu_{i1})_1 & \dots & \sum_i a_i(\mu_{in})_1 \\ \vdots & \vdots & \ddots & \vdots \\ a_n & \sum_i a_i(\mu_{i1})_n & \dots & a_0 + \sum_i a_i(\mu_{in})_n \end{bmatrix},$$

is a general formulation, for all $a = a_0 + a_1\boldsymbol{i}_1 + \cdots + a_n\boldsymbol{i}_n$.

Symmetrically, the multiplication to the right by $a \in \mathbb{A}$ yields an operator $\mathcal{A}_R : \mathbb{A} \to \mathbb{A}$ defined by $\mathcal{A}_R(x) = xa$, for all $x \in \mathbb{A}$. Once again by applying the isomorphism φ given by (2.6) we get

$$\varphi\left(\mathcal{A}_R(x)\right) = \Phi_R(a)\varphi(x), \tag{2.9}$$

where $\Phi_R : \mathbb{A} \to \mathbb{R}^{(n+1)\times(n+1)}$ is defined by

$$\Phi_R(a) = \begin{bmatrix} | & | & & | \\ \varphi(a) & \varphi(\boldsymbol{i}_1 a) & \dots & \varphi(\boldsymbol{i}_n a) \\ | & | & & | \end{bmatrix}. \tag{2.10}$$

Lastly, the absolute value (or norm) of a hypercomplex number $x = x_0 + x_1\boldsymbol{i}_1 + \cdots + x_n\boldsymbol{i}_n \in \mathbb{A}$ is defined as

$$|x| = \sqrt{\sum_{i=0}^{n} x_i^2}. \tag{2.11}$$

That means, the absolute value of x corresponds to the Euclidean norm of $\varphi(x)$, that is, $|x| = ||\varphi(x)||_2$.

In the remainder of this work we will often refer to the dimension of an algebra. We define the dimension of the algebra as seen as a vector space, i.e., an algebra with $(n-1)$ hyperimaginary units is said to have dimension n. Moreover, we consider the canonical basis as $\beta = \{1, \boldsymbol{i}_1, \dots, \boldsymbol{i}_{n-1}\}$.

2.1 Notable algebras

In this section we overview a handful of hypercomplex algebras of dimension four, going over their main definition, notable properties and providing their multiplication tables. The examples elected in this section feature some of the most well-known four-dimensional hypercomplex algebras, both from computational and theoretical standpoints. These algebras are also featured in the applications presented in later chapters, 3 and 4. In particular, many algebras used here are also present in the recent work by Takahashi (TAKAHASHI, 2021). Note that in the presentation of the multiplication tables we suppress

the row and column associated with 1, since it is the identity element for the multiplication in every algebra presented, i.e., $1a = a1 = a$, $\forall a$.

In the quaternions algebra $\mathbb{H}$ the hyperimaginary units are $\boldsymbol{i} \equiv \boldsymbol{i}_1$, $\boldsymbol{j} \equiv \boldsymbol{i}_2$, and $\boldsymbol{k} \equiv \boldsymbol{i}_3$. Since quaternions are so widely known, we will adopt this notation for all algebras in this work, thus the general element of a four-dimensional algebra shall be henceforth represented as

$$x = x_0 + x_1\boldsymbol{i} + x_2\boldsymbol{j} + x_3\boldsymbol{k}. \tag{2.12}$$

Example 1 (Quaternions)**.** *Quaternions ($\mathbb{H}$) are arguably one of the most well-known hypercomplex algebras. Introduced in 1843 by W. R. Hamilton, quaternions are an extension of complex numbers and represent three-dimensional space rotations compactly. Indeed, a quaternion can be expressed in polar form as*

$$q = |q|(\cos\theta + \text{sen}\theta\overrightarrow{u}) \tag{2.13}$$

where $|q|$ is the norm of q, $\theta \in [0, \pi)$ and $\overrightarrow{u} = u_x\boldsymbol{i} + u_y\boldsymbol{j} + u_z\boldsymbol{k}$, where (u_x, u_y, u_z) is a direction in $\mathbb{R}^3$. Then, by taking $q^* = |q|(\cos\theta - \text{sen}\theta\overrightarrow{u})$ the conjugate of q we have that

$$v_r = qvq^* \tag{2.14}$$

is the vector v rotated by 2θ around the axis determined by $\overrightarrow{u}$ and dilated by $|q|^2$. In other words, a quaternion uniquely describes a rotation-dilation operation. In particular, unitary quaternions describe strict rotations.

Quaternions have been extensively applied to control and computer graphics and vision due to this connection to rotations in 3D-space. They are also known to avoid the Gimbal Lock, an effect that incurs a loss of a degree of freedom in certain rotations in space. The product of quaternion hyperimaginary units is anticommutative and is described in Table 2.

Example 2 (Cayley-Dickson Algebras)**.** *In 1919 Dickson introduced a recursive process that generates algebras of doubling dimension (SCHAFER, 1954). Notably, complex numbers are derived from real numbers using this recursive process. Then, quaternions are obtained from complex numbers, and octonions, also known as Cayley numbers, are obtained from quaternions (CULBERT, 2007). For this reason, algebras generated by Dickson's recursive process are called Cayley-Dickson algebras. The Cayley-Dickson process incurs in loss of properties with each increment in dimension. For example, complex numbers are not an ordered set. Quaternions are not ordered, and their product is not commutative. Octonions lack the ordering property as well as the commutativity and associativity of the product.*

In this work, we consider a generalized version of the Cayley-Dickson algebras over $\mathbb{R}$ proposed by Albert in 1942 (ALBERT, 1942). Formally, we take the first Cayley-Dickson algebra as $\mathbb{A}_0 = \mathbb{R}$, the real numbers. Then, given a non-zero scalar $\gamma_k \in \mathbb{R}$ called

Table 2 – Multiplication tables of four-dimensional Cayley-Dickson algebras, including the quaternions ($\mathbb{H} \approx \mathbb{R}[-1,-1]$).

$\mathbb{H}$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	-1	$\boldsymbol{k}$	$-\boldsymbol{j}$
$\boldsymbol{j}$	$-\boldsymbol{k}$	-1	$\boldsymbol{i}$
$\boldsymbol{k}$	$\boldsymbol{j}$	$-\boldsymbol{i}$	-1

$\mathbb{R}[+1,+1]$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	1	$\boldsymbol{k}$	$\boldsymbol{j}$
$\boldsymbol{j}$	$-\boldsymbol{k}$	1	$-\boldsymbol{i}$
$\boldsymbol{k}$	$-\boldsymbol{j}$	$\boldsymbol{i}$	-1

$\mathbb{R}[-1,+1]$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	-1	$\boldsymbol{k}$	$-\boldsymbol{j}$
$\boldsymbol{j}$	$-\boldsymbol{k}$	1	$-\boldsymbol{i}$
$\boldsymbol{k}$	$\boldsymbol{j}$	$\boldsymbol{i}$	1

$\mathbb{R}[+1,-1]$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	1	$\boldsymbol{k}$	$\boldsymbol{j}$
$\boldsymbol{j}$	$-\boldsymbol{k}$	-1	$\boldsymbol{i}$
$\boldsymbol{k}$	$-\boldsymbol{j}$	$-\boldsymbol{i}$	1

generator, the kth Cayley-Dickson algebra is defined recursively as the Cartesian product $\mathbb{A}_{k-1} \times \mathbb{A}_{k-1}$*, that is,*

$$\mathbb{A}_k = \{(x,y) : x,y \in \mathbb{A}_{k-1}\}, \quad k \geqslant 1, \tag{2.15}$$

where addition and multiplication are defined, using the operations from A_{k-1}*, as follows for all* $(x,y),(z,w) \in \mathbb{A}_k$ *and* $\alpha \in \mathbb{R}$*:*

- *Addition:*

$$(x,y) + (z,w) = (x+z, y+w). \tag{2.16}$$

- *Multiplication:*

$$(x,y)(z,w) = (xz + \gamma_k w^* y, wx + yz^*). \tag{2.17}$$

Remark 1. *The conjugate is defined as* $(x,y)^* = (x^*, -y)$*. Note that this uses the conjugate of* $\mathbb{A}_{k-1}$*, thus it is defined recursively. We recall that in* $\mathbb{R}$ *this is the identity, i.e., we have* $x^* = x$ *and hence the conjugate operation is equivalent to changing the sign of each component of the imaginary part.*

The Cayley-Dickson algebra depends on the previous algebra $\mathbb{A}_{k-1}$ *and the generator* γ_k*. Therefore, we can write*

$$\mathbb{A}_k := \mathbb{A}_{k-1}[\gamma_k]. \tag{2.18}$$

We note that $\mathbb{R}[-1] \equiv \mathbb{C}$ *is the complex number field. Moreover, as pointed out previously, quaternions are an extension of* $\mathbb{C}$*, indeed, we have* $\mathbb{C}[-1] \equiv \mathbb{R}[-1,-1] \equiv \mathbb{H}$*. Since the dimension of an algebra doubles with each step of the process obtaining four-dimensional algebras requires 2 steps, and hence 2 generators* γ_1, γ_2*. Without loss of generality we take* $\gamma_1, \gamma_2 \in \{-1,+1\}$*. Therefore we end up with the quaternions and three other notable four-dimensional algebras over* $\mathbb{R}$*, namely,* $\mathbb{R}[+1,+1]$ *and* $\mathbb{R}[+1,-1]$ *are the Clifford algebras* $\mathcal{C}\ell_{2,0}$ *and* $\mathcal{C}\ell_{1,1}$ *respectively, and* $\mathbb{R}[-1,+1]$ *is the coquaternions (or split-quaternions). The multiplication tables for all these algebras are shown in Table 2.*

Example 3 (Clifford Algebras)**.** *Briefly speaking, a Clifford algebra is an associative algebra generated by a vector space endowed with a quadratic form (HESTENES; SOBCZYK, 2012). The main interest in this family of algebras is regarding its geometric properties, which play an important role in many applications, ranging from physics (CRUMEYROLLE, 2013; CHISHOLM; FARWELL, 1996) to digital signal processing (LABUNETS, 2004).*

A Clifford algebra over $\mathbb{R}$ *is denoted by* $\mathcal{C}\ell_{p,q}(\mathbb{R})$*, or simply* $\mathcal{C}\ell_{p,q}$*. Here* p *and* q *are non-negative integers. The dimension of* $\mathcal{C}\ell_{p,q}$ *is* $n = 2^{p+q}$*. Since we are interested in four-dimensional Clifford algebras, we impose the constraint* $p + q = 2$*. This leads to a total of 3 algebras. The algebra* $\mathcal{C}\ell_{0,2}$ *is equivalent to the quaternions, with multiplication table given previously by Table 2. The two remaining configurations are* $\mathcal{C}\ell_{1,1}$ *and* $\mathcal{C}\ell_{2,0}$*, both of which are equivalent to* $M_2(\mathbb{R})$*, i.e., the algebra of square real matrices of order* 2 *(PORTEOUS; others, 1995). Moreover, the multiplication table of* $\mathcal{C}\ell_{1,1}$ *is identical to* $\mathbb{R}[+1, -1]$*, whereas the table for* $\mathcal{C}\ell_{2,0}$ *is identical to that of* $\mathbb{R}[+1, +1]$*.*

Example 4 (MacFarlane's hyperbolic quaternions)**.** *This algebra was formalized in 1900 by MacFarlane (DEMIR; TANIŞLI; CANDEMIR, 2010). Among its particularities is the fact that the MacFarlane's hyperbolic quaternions lacks two desirable properties of the product: associativity and commutativity. Denoted by* $\mathbb{Y}$*, the multiplication table of this algebra is presented in Table 3. We draw attention to the fact that the table is not symmetric, thus corroborating the non-commutativity.*

Commutativity is a key property for algebras. From a computational standpoint it can be used to ensue many simplifications thus reducing operations' computational complexity. We finish this section with two examples addressing commutative four-dimensional algebras.

Example 5 (Tessarines)**.** *The tessarines were introduced in 1849 (ROCHON; SHAPIRO, 2004). Also known as bicomplex numbers, this is a commutative four-dimension algebra that differs slightly from the Cayley-Dickson algebra* $\mathbb{R}[-1, +1]$ *(split-quaternions). For that reason it is also known as commutative quaternions. Table 3 shows the multiplication rules for the tessarines. This algebra has been applied for digital signal processing (PEI; CHANG; DING, 2004; ALFSMANN, 2006).*

Example 6 (Klein Four-Group)**.** *The Klein four-group* $\mathbb{K}_4$ *is a four-dimension algebra whose hyperimaginary unit are self-inverse, i.e.,* $\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = 1$*. Moreover, the product of two hypercomplex units yields the third. Table 3 contains the multiplication rules for the Klein group. This algebra was used extensively in theoretical results in symmetric group theory (HUANG; YU, 2013; CRAVEN et al., 2011), and Hopfield neural networks based on the Klein four-group have also shown outstanding results (KOBAYASHI, 2020).*

Table 3 – Multiplication tables of the hyperbolic quaternions ($\mathbb{Y}$), tessariens ($\mathbb{T}$), and Klein four-group ($\mathbb{K}_4$).

$\mathbb{Y}$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	1	$\boldsymbol{k}$	$-\boldsymbol{j}$
$\boldsymbol{j}$	$-\boldsymbol{k}$	1	$\boldsymbol{i}$
$\boldsymbol{k}$	$\boldsymbol{j}$	$-\boldsymbol{i}$	1

$\mathbb{T}$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	-1	$\boldsymbol{k}$	$-\boldsymbol{j}$
$\boldsymbol{j}$	$\boldsymbol{k}$	1	$\boldsymbol{i}$
$\boldsymbol{k}$	$-\boldsymbol{j}$	$\boldsymbol{i}$	-1

$\mathbb{K}_4$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	1	$\boldsymbol{k}$	$\boldsymbol{j}$
$\boldsymbol{j}$	$\boldsymbol{k}$	1	$\boldsymbol{i}$
$\boldsymbol{k}$	$\boldsymbol{j}$	$\boldsymbol{i}$	1

2.2 Hypercomplex-Valued Matrix Computation

In this section we discuss matrix operations in hypercomplex algebras. Formally, using the operators φ, Φ_L, and Φ_R given respectively by (2.6), (2.7), and (2.9), we present a framework to perform hypercomplex-valued matrix computations using well-known real-valued matrix computation techniques. This not only allows us to work with models without going through the toil of implementing hypercomplex algebra operations from scratch, but also allows us to take advantage of modern computing tools and libraries.

A hypercomplex matrix is simply a matrix with entries in a hypercomplex algebra $\mathbb{A}$. Similar to real-valued linear algebra, the product of matrices $\boldsymbol{A} \in \mathbb{A}^{M\times L}$ and $\boldsymbol{B} \in \mathbb{A}^{L\times N}$ results in a new matrix $\boldsymbol{C} \in \mathbb{A}^{M\times N}$ with entries given by

$$c_{ij} = \sum_{\ell=1}^{L} a_{i\ell} b_{\ell j}, \tag{2.19}$$

where $i = 1, \ldots, M$ and $j = 1, \ldots, N$. Here, $a_{i\ell}$ and $b_{\ell j}$ are entries of the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively, and hence the right hand side of (2.19) is a linear combination of products in $\mathbb{A}$.

Since most modern scientific computing softwares and tools are based on real-valued linear algebra, we use the isomorphism (2.6) and either (2.7) or (2.9) to compute matrix operations of $\mathbb{A}$ using real-valued operations instead. This yields much faster computing than general purpose codes meant to deal with hypercomplex computations. Precisely, the application of φ to both sides of (2.19) along with (2.7) allows us to write

$$\begin{aligned}\varphi(c_{ij}) &= \sum_{\ell=1}^{L} \varphi\left(a_{i\ell} b_{\ell j}\right) = \sum_{\ell=1}^{L} \Phi_L(a_{i\ell})\varphi(b_{\ell j}) \\ &= \begin{bmatrix} \Phi_L(a_{i1}) & \Phi_L(a_{i2}) & \ldots & \Phi_L(a_{iL}) \end{bmatrix} \begin{bmatrix} \varphi(b_{1j}) \\ \varphi(b_{2j}) \\ \vdots \\ \varphi(b_{Lj}) \end{bmatrix}.\end{aligned}$$

Equivalently, using real-valued matrix operations, we have

$$\varphi(\boldsymbol{C}) = \Phi_L(\boldsymbol{A})\varphi(\boldsymbol{B}). \tag{2.20}$$

The operators Φ_L and φ are defined for hypercomplex matrix arguments respectively as:

$$\Phi_L(\boldsymbol{A}) = \begin{bmatrix} \Phi_L(a_{11}) & \Phi_L(a_{12}) & \dots & \Phi_L(a_{1L}) \\ \vdots & \vdots & \ddots & \vdots \\ \Phi_L(a_{M1}) & \Phi_L(a_{M2}) & \dots & \Phi_L(a_{ML}) \end{bmatrix}, \tag{2.21}$$

and

$$\varphi(\boldsymbol{B}) = \begin{bmatrix} \varphi(b_{11}) & \dots & \varphi(b_{1N}) \\ \varphi(b_{21}) & \dots & \varphi(b_{2N}) \\ \vdots & \ddots & \vdots \\ \varphi(b_{L1}) & \dots & \varphi(b_{LN}) \end{bmatrix}. \tag{2.22}$$

Here it is important to note that $\Phi_L(\boldsymbol{A})$ is a real-valued matrix of size $(n+1)M \times (n+1)L$ and $\varphi(\boldsymbol{B})$ is a real-valued matrix of size $(n+1)L \times N$. The real-valued matrix $\varphi(\boldsymbol{C}) \in \mathbb{R}^{(n+1)M \times N}$ yielded by (2.20) is expressed in the form of (2.22). Therefore, one can obtain the hypercomplex-valued matrix $\boldsymbol{C} \in \mathbb{A}^{M \times N}$ simply by rearranging the elements of $\varphi(\boldsymbol{C})$ by means of the inverse mapping $\varphi^{-1} : \mathbb{R}^{(n+1)M \times N} \to \mathbb{A}^{M \times N}$. In sum, this means

$$\boldsymbol{C} = \varphi^{-1}\left(\Phi_L(\boldsymbol{A})\varphi(\boldsymbol{B})\right), \tag{2.23}$$

which provides an effective formula for the computation of hypercomplex-valued matrix product through the real-valued linear algebra operators, which are often available in scientific computing softwares. This is the cornerstone of the implementations in the works discussed in the following chapters.

As a remark, we recall that it is possible to compute $\boldsymbol{C}$ using the same idea for the product to the right by $b_{\ell j}$ in (2.19) instead of product to the left by $a_{i\ell}$. In such case, we have

$$\begin{aligned} \varphi(c_{ij}) &= \sum_{\ell=1}^{L} \Phi_R(b_{\ell j})\varphi(a_{i\ell}) \\ &= \begin{bmatrix} \Phi_R(b_{1j}) & \Phi_R(b_{2j}) & \dots & \Phi_R(b_{Lj}) \end{bmatrix} \begin{bmatrix} \varphi(a_{i1}) \\ \varphi(a_{i2}) \\ \vdots \\ \varphi(a_{iL}) \end{bmatrix}, \end{aligned}$$

for all $i = 1, \dots, M$ and $j = 1, \dots, N$. Using the same logic as the conversion to real-valued matrix operations above, we obtain the identity

$$\phi(\boldsymbol{C}) = \Phi_R(\boldsymbol{B})\phi(\boldsymbol{A}), \tag{2.24}$$

where Φ_R is defined by

$$\Phi_R(\boldsymbol{B}) = \begin{bmatrix} \Phi_R(b_{11}) & \Phi_R(b_{21}) & \dots & \Phi_R(b_{L1}) \\ \vdots & \vdots & \ddots & \vdots \\ \Phi_R(b_{1N}) & \Phi_R(b_{2j}) & \dots & \Phi_R(b_{LN}) \end{bmatrix}. \tag{2.25}$$

Here $\phi(\boldsymbol{A}) = \varphi(\boldsymbol{A}^T)$ since we are using the product to the right. In this case $\Phi_R(\boldsymbol{B})$ and $\varphi(\boldsymbol{A}^T)$ are real-valued matrices of respective sizes $(n+1)N \times (n+1)L$ and $(n+1)L \times M$. Finally, the hypercomplex-valued matrix $\boldsymbol{C} = \boldsymbol{AB}$ can alternatively be computed by means of the equation

$$\boldsymbol{C} = \phi^{-1}\left(\Phi_R(\boldsymbol{B})\phi(\boldsymbol{A})\right). \tag{2.26}$$

Remark 2. *From a computational standpoint, the bottleneck in* (2.23) *and* (2.26) *is the construction of the real-valued matrices* $\Phi_L(\boldsymbol{A})$ *and* $\Phi_R(\boldsymbol{B})$ *of sizes* $(n+1)M \times (n+1)L$ *and* $(n+1)N \times (n+1)L$*, respectively. It can be deduced that* (2.23) *is cheaper than* (2.26) *if the matrix* $\boldsymbol{A}$ *has less entries than* $\boldsymbol{B}$*, and the other way around if* $\boldsymbol{B}$ *has less entries. In practice, we implement both* (2.23) *and* (2.26) *and dynamically check whether to compute the product of two hypercomplex-valued matrices using* (2.23) *or* (2.26) *by a simple comparison of the sizes of matrices.*

3 Hv-ELM: Hypercomplex-valued Extreme Learning Machine

This chapter concerns itself with Extreme Learning Machines (ELMs). Namely, it contains the hypercomplex-valued matrix computation concepts necessary for developing hypercomplex-valued ELMs, which are centered around the hypercomplex-valued least-squares problem. In particular, we show how to solve this type of least-squares problems through real-valued pseudo-inversion (GOLUB, 2007; TREFETHEN; III, 1997).

3.1 Real-valued Extreme Learning Machines

As theoretical baseline we consider the Multilayer Perceptron (MLP) as depicted in Fig. 1. A MLP features multiple dense hidden layers and a dense output layer and training is done by the backpropagation algorithm based on gradient descent. In 1999, Husmeier presented the random vector functional link (RVFL) (HUSMEIER; HUSMEIER, 1999). Years later, the selfsame idea was presented by Huang *et. al* (HUANG et al., 2004) as real-valued Extreme Learning Machine (ELM). This model consists of a feedforward network with multiple dense hidden layers and a dense output layer, i.e., the same layout as the MLP. The main idea behind the ELM is that, by fixating the randomly initialized synaptic weights of the hidden layers (all layers except for the output layer on Fig. 1), the output of the last hidden layer is fixated. Then, the weights connecting the last hidden layer to the output layer can be adjusted by means of a least squares problem (LSP). This allows for a change in the training step from an iterative process (backpropagation) to a direct method, which means the optimal solution is attained in a finite numbers of operations that can also be calculated *a priori*. The most direct implication is that the ELM offers a trade-off: training is way faster when compared to the MLP, but the hidden layer weights are randomly generated and hence not interpretable nor adjustable. Since hidden layers act as feature extractors, the randomly generated hidden layers are akin to black-boxes. On the other hand, since the training cost is lower, one can use an expressively larger number of neurons in the hidden layers.

3.2 Hypercomplex-valued Extreme Learning Machines

Hypercomplex-valued ELMs (Hv-ELMs) are simply ELMs in which the inputs, parameters and outputs are numbers from a hypercomplex algebra. Accordingly, Hv-ELMs are trained by means of hypercomplex-valued least-squares problems. The framework

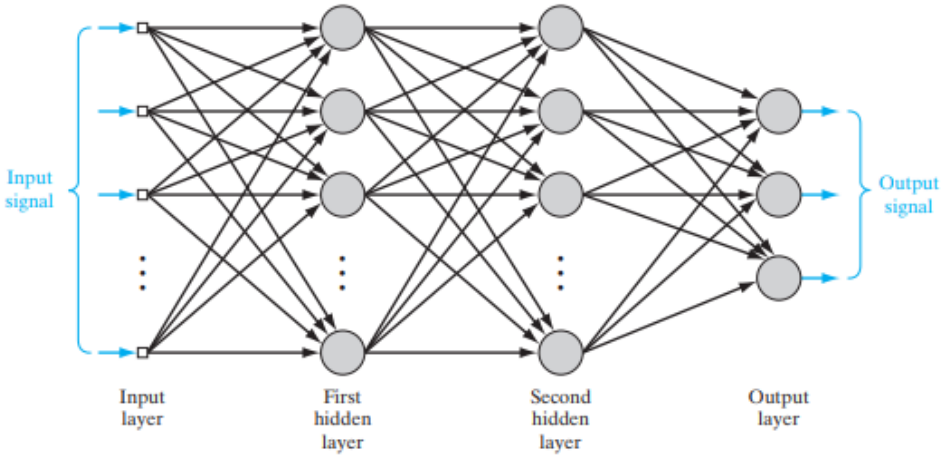

Figure 1 – Feedforward network with dense layers. Figure taken from (HAYKIN et al., 2009).

described below can solve such least-squares problems in hypercomplex algebras. This leads directly to the implementation of hypercomplex-valued ELMs. We begin the discussion regarding the hypercomplex-valued least-squares problem with the Frobenius norm.

Analogously to the real-valued case (GOLUB, 2007), the Frobenius norm of a hypercomplex-valued matrix $\boldsymbol{A} \in \mathbb{A}^{M\times N}$ is defined by

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i=1}^{M}\sum_{j=1}^{N} |a_{ij}|^2}, \tag{3.1}$$

where $|a_{ij}|$ represents the absolute value of $a_{ij} \in \mathbb{A}$. By combining (2.6) and (2.11), we conclude that $|a_{ij}| = \|\varphi(a_{ij})\|_2$, where $\|\cdot\|_2$ stands for the Euclidean norm. Moreover, from (2.22) it can be seen that the Frobenius norm of a hypercomplex-valued matrix is related to the real-valued Frobenius norm by means of the identity:

$$\|\boldsymbol{A}\|_F = \|\varphi(\boldsymbol{A})\|_F. \tag{3.2}$$

As in the real-valued case (GOLUB, 2007), the Frobenius norm (3.1) is required to define the hypercomplex-valued least squares problem as follows:

Definition 1 (Hypercomplex-Valued Least Squares Problem)**.** *Given matrices $\boldsymbol{A} \in \mathbb{A}^{M\times L}$ and $\boldsymbol{B} \in \mathbb{A}^{M\times N}$, the hypercomplex-valued least squares problem consists of finding the minimal Frobenius norm solution to the problem*

$$\min\left\{\|\boldsymbol{AX} - \boldsymbol{B}\|_F : \boldsymbol{X} \in \mathbb{A}^{L\times N}\right\}. \tag{3.3}$$

In practice we solve a hypercomplex-valued least square problem using the real-valued algebra framework detailed previously. A detailed account on real-valued least-squares problems and their solution methods can be found in (GOLUB, 2007;

TREFETHEN; III, 1997). We briefly state that the solution is usually obtained by means of the Moore-Penrose pseudoinverse when the matrix $\boldsymbol{A}$ has full rank. A matrix with randomly generated entries has full rank with probability 1 (FENG; ZHANG, 2007). Formally, for real matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{X}$ we have:

$$\begin{aligned} \boldsymbol{A}\boldsymbol{X} &= \boldsymbol{B} \\ \boldsymbol{A}^T A\boldsymbol{X} &= \boldsymbol{A}^T\boldsymbol{B} \\ \boldsymbol{X} &= (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{B}. \end{aligned}$$

The matrix $(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$ is called the pseudoinverse of $\boldsymbol{A}$ and is denoted by $\boldsymbol{A}^\dagger$. From (2.20) and (3.2), we write

$$\begin{aligned} \|\boldsymbol{A}\boldsymbol{X} - \boldsymbol{B}\|_F &= \|\varphi(\boldsymbol{A}\boldsymbol{X} - \boldsymbol{B})\|_F = \|\varphi(\boldsymbol{A}\boldsymbol{X}) - \varphi(\boldsymbol{B})\|_F \\ &= \|\Phi_L(\boldsymbol{A})\varphi(\boldsymbol{X}) - \varphi(\boldsymbol{B})\|_F. \end{aligned}$$

Hence, the hypercomplex-valued least squares problem is equivalently written as a real-valued problem:

$$\min\{\|\Phi_L(\boldsymbol{A})\boldsymbol{X}^{(r)} - \varphi(\boldsymbol{B})\|_F : \boldsymbol{X}^{(r)} \in \mathbb{R}^{(n+1)L\times N}\}. \tag{3.4}$$

Here $\boldsymbol{X}^{(r)}$ is simply $\varphi(\boldsymbol{X})$. Now, the real-valued least square problem in (3.4) is solved by means of the Moore-Penrose pseudoinverse (GOLUB, 2007), i.e.,

$$\boldsymbol{X}^{(r)} = \Phi_L(\boldsymbol{A})^\dagger\varphi(\boldsymbol{B}), \tag{3.5}$$

where $\Phi_L(\boldsymbol{A})^\dagger$ is the pseudoinverse of $\Phi_L(\boldsymbol{A})$. Finally, making use of the real-valued linear algebra, the solution of the hypercomplex-valued least squares problem (3.3) can be obtained by using the inverse isomorphism φ^{-1} on the solution of the real-valued least squares problem:

$$\boldsymbol{X} = \varphi^{-1}\left(\Phi_L(\boldsymbol{A})^\dagger\varphi(\boldsymbol{B})\right). \tag{3.6}$$

In the following sections we expound two applications featuring Hv-ELMs implemented according to the framework presented above: a time series prediction task on the Lorenz system and one color image auto-encoder in the CIFAR-10 dataset. One of the key aspects of these works is to compare the proposed ELM model based on different hypercomplex algebras of the same dimension. As previously stated, hypercomplex-valued neural network models are well adapted to tasks involving high-dimensional data, that is, when the input consists of many signals related to the same object. In both experiments we include four-dimensional hypercomplex-valued ELMs based on seven of the algebras discussed previously on Section 2.1. We also consider a real-valued ELM of equivalent size for comparison purposes.

The total number of parameters (TNP), defined as the raw number of trainable parameters of a network, has been used to work with comparable size networks. In the case

of ELMs, all but the last layer's parameters (weights and bias) are randomly initialized and fixed, therefore non-trainable. Thus, a real-valued ELM with an input signal of dimension $D^{(\mathbb{R})}$, $L^{(\mathbb{R})}$ neurons in the hidden layer, and output of dimension $O^{(\mathbb{R})}$, has

$$TNP^{(\mathbb{R})} = (L^{(\mathbb{R})} + 1)O^{(\mathbb{R})}. \tag{3.7}$$

Analogously, a four-dimensional hypercomplex-valued ELM model has 4-times the TNP of the real-valued with the same layout. Formally, we have

$$TNP^{(\mathbb{A})} = 4(L^{(\mathbb{A})} + 1)O^{(\mathbb{A})}, \tag{3.8}$$

where $L^{(\mathbb{A})}$ and $O^{(\mathbb{A})}$ denote respectively the number of hidden neurons and the dimension of the output.

3.3 Application: Times Series Prediction

For the time series prediction task, we considered the well known Lorenz system. The Lorenz system is a chaotic system of ordinary differential equations describing a nonperiodic flow on a three dimensional space. Formally, this system is described by the equations

$$\begin{cases} \dfrac{dx}{dt} = \sigma(y - x), \\ \dfrac{dy}{dt} = x(\rho - z) - y, \\ \dfrac{dz}{dt} = xy - \beta z, \end{cases} \tag{3.9}$$

where the variables (x, y, z) describe a position in space. The constants $\sigma, \rho, \beta > 0$ characterize the system. The Lorenz system is chaotic for some of these constants' values, which means that small perturbations in the initial conditions will often propagate into large changes in the resulting trajectory. We considered $\sigma = 10$, $\beta = 8/3$, and $\rho = 28$ in our computational experiments as these values are known to yield chaotic behavior near the origin.

As usual in time series prediction (DATAR et al., 2002; XU; XIA; MANDIC, 2016), we consider a sliding window of fixed length T. In short: T consecutive positions are used as input for a model that attempts to predict the next position. Here we take $T = 3$. Thus, each training sample contains the positions $\mathbf{p}_{t-2} = (x_{t-2}, y_{t-2}, z_{t-2})$, $\mathbf{p}_{t-1} = (x_{t-1}, y_{t-1}, z_{t-1})$, and $\mathbf{p}_t = (x_t, y_t, z_t))$ as input while the desired output is the position $\mathbf{p}_{t+1} = (x_{t+1}, y_{t+1}, z_{t+1})$, for some t point in time.

Therefore, the real-valued network has 9 real input values consisting of the nine variables concatenated in a vector. Mathematically:

- **Real input:** $(x_{t-2}, y_{t-2}, z_{t-2}, x_{t-1}, y_{t-1}, z_{t-1}, x_t, y_t, z_t)$.

The desired output for the real-valued model is simply the 3-position vector $\mathbf{p}_{t+1}$.

For the four-dimensional hypercomplex-valued models we encode the positions in the hyperimaginary parts of distinct hypercomplex numbers. Formally we have:

- **Hypercomplex input:** (p_{t-2}, p_{t-1}, p_t),

where $p_t = x_t\boldsymbol{i} + y_t\boldsymbol{j} + z_t\boldsymbol{k}$ for discrete time instant t. The output of the hypercomplex-valued network is simply the hypercomplex number representing the position in $t+1$ as $p_{t+1} = x_{t+1}\boldsymbol{i} + y_{t+1}\boldsymbol{j} + z_{t+1}\boldsymbol{k}$.

We generated 4.000 consecutive positions using a fourth-order Runge-Kutta method. The training set consists of the first 300 positions, while the remaining 3.700 positions have been used for testing. Due to the sliding window the training and test sets have 297 and 3697 samples, respectively.

We evaluated the performance of the ELM models according to the prediction gain, following (HAYKIN; LI, 1995). Broadly speaking, this quantity is measured in decibel (dB) and expresses the ratio between reference level (σ_s) and the signal error (σ_e). This metric is defined as

$$R = 10\log_{10}\frac{\sigma_s^2}{\sigma_e^2}, \tag{3.10}$$

where σ_s^2 is the estimated variance of the input signal and σ_e^2 denotes the estimated variance of the prediction error. The reference level for the prediction gain is usually specified for physical quantities, for example. More broadly, the reference level can be determined implicitly from the input signal as the variance σ_s^2. Precisely, we computed the sample variances

$$\sigma_s^2 = \frac{1}{N_S - 1}\sum_{t=1}^{N_S}(\|\mathbf{p}_t\|_2 - \mu_s)^2, \tag{3.11}$$

and

$$\sigma_e^2 = \frac{1}{N_S - 1}\sum_{t=1}^{N_S}(\|\mathbf{p}_{t+1} - \hat{\mathbf{p}}_{t+1}\|_2 - \mu_e)^2, \tag{3.12}$$

where $\hat{\mathbf{p}}_{t+1} = \texttt{ELM}(\mathbf{p}_{t-2}, \mathbf{p}_{t-1}, \mathbf{p}_t)$ is the position predicted by an ELM model, N_S is the number of samples, and

$$\mu_s = \frac{1}{N_S}\sum_{t=1}^{N_S}\|\mathbf{p}_t\|_2 \quad \text{and} \quad \mu_e = \frac{1}{N_S}\sum_{t=1}^{N_S}\|\mathbf{p}_{t+1} - \hat{\mathbf{p}}_{t+1}\|_2. \tag{3.13}$$

are the input and error means, respectively.

According to (3.7), the TNP for the real-valued feedforward network with $L^{(\mathbb{R})}$ hidden neurons is $\text{TNP}^{(\mathbb{R})} = 3L^{(\mathbb{R})} + 3$. Similarly, for a hypercomplex-valued network with $L^{(\mathbb{A})}$ hidden neurons, $\text{TNP}^{(\mathbb{A})} = 4L^{(\mathbb{A})} + 4$. Imposing $\text{TNP}^{(\mathbb{R})} \simeq \text{TNP}^{(\mathbb{A})}$, we obtain the following relationship for the number of hidden neurons:

$$L^{(\mathbb{R})} \simeq \frac{4L^{(\mathbb{A})}}{3}. \tag{3.14}$$

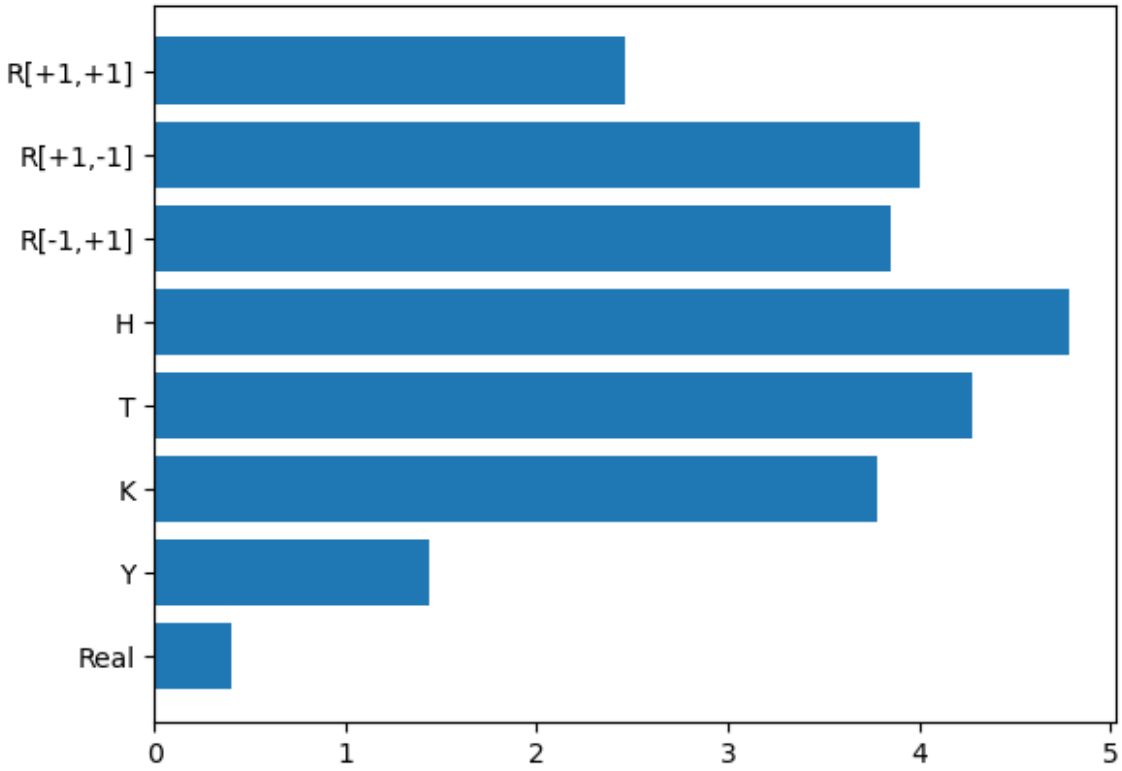

Figure 2 – The probability of an ELM predictor yields the highest prediction gain by the underlying algebra.

We performed a series of simulations with $L^{(\mathbb{A})}$ ranging in $\{11, 12, \ldots, 34, 35\}$ and determining the corresponding number of hidden neurons for the real-valued ELM using (3.14), resulting $L^{(\mathbb{R})} \in \{15, 16, \ldots, 45, 47\}$. For each $L^{(\mathbb{A})}$ and $L^{(\mathbb{R})}$, we trained and tested 100 networks of each, seven hypercomplex-valued, one real-valued, resulting in a total of 20.000 simulations. We annotated the best-performing model for each of these rounds, i.e., the model that yielded the highest prediction gain. Figure 2 shows the normalized frequency with which each of the models outperformed all others. Note that Figure 2 depicts the fraction of the 20.000 simulations won by each model, and can therefore be interpreted as the likelihood of each ELM model yielding the highest prediction gain.

Note that the real-valued model underperformed the hypercomplex-valued ELM networks, seldom showcasing the highest prediction gain. Moreover, among the top-performing models three are based on Cayley-Dickson algebras, namely the quaternions ($\mathbb{H}$), $\mathbb{R}[-1, +1]$, and $\mathbb{R}[+1, -1]$. The tessarines- and Klein-group-based ELMs performed on par with these Cayley-Dickson models, while $\mathbb{R}[+1, +1]$ and the hyperbolic quaternions exhibited noticeably worse results. Nonetheless, the hypercomplex models' advantage over the real-valued becomes clearer by taking the average prediction gain over 100 simulations for each number of hidden neurons. Figure 3 shows the average prediction gain by the total number of parameters for several models, namely the real-valued ELM,

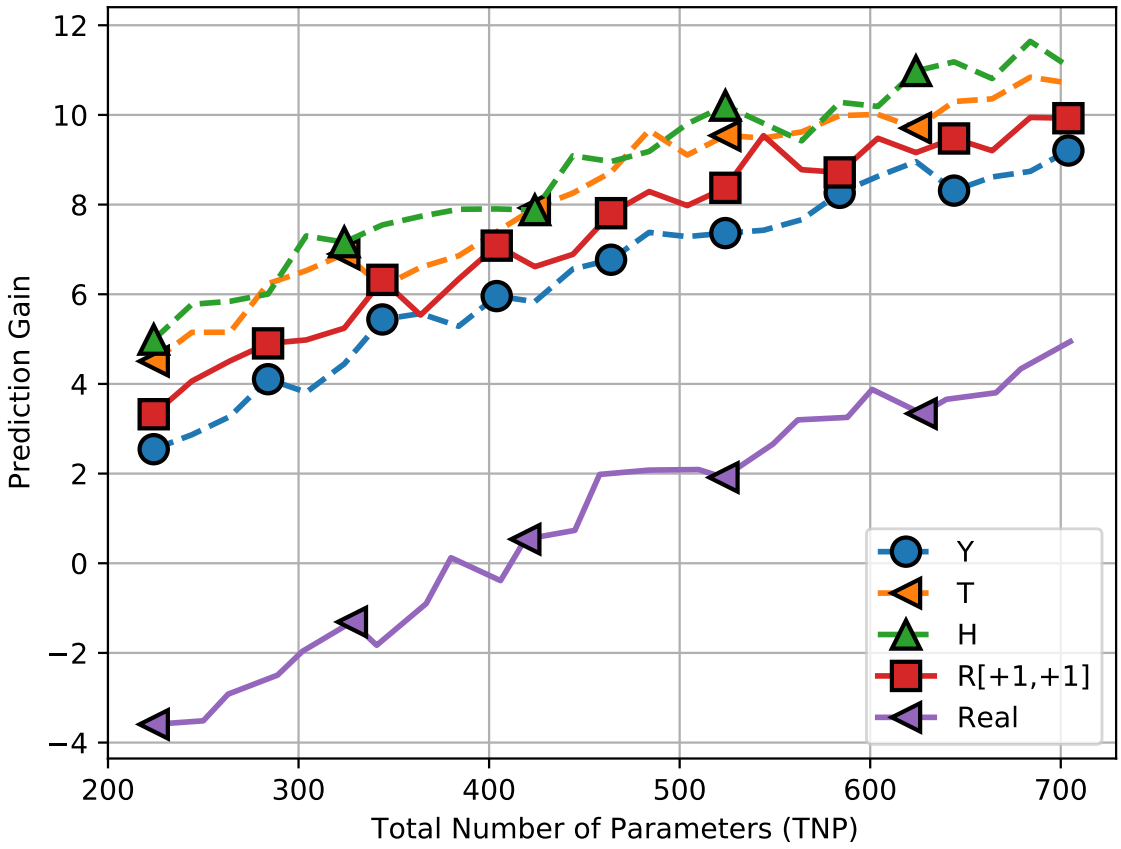

Figure 3 – The average prediction gain by the total number of parameters.

the two top-performing hypercomplex-valued models ($\mathbb{H}$ and $\mathbb{T}$) and the two bottom-performing hypercomplex-valued models ($\mathbb{R}[+1,+1]$ and $\mathbb{Y}$). Figure 3 shows that the top-performing models hold a good margin over the bottom-performing ones, while the three omitted hypercomplex-valued models lie between these two separate groups. Moreover, all hypercomplex-valued models are significantly ahead of the real-valued ELM network.

3.4 Application: Colored Image Auto-encoding

This section presents an application in auto-encoding featuring the CIFAR-10 dataset. This dataset was originally intended as an image database for classification tasks. It consists of five training batches and one testing batch, each one consisting of $10,000$ images, adding up to a total of $60,000$ images divided evenly into 10 classes.

Although the CIFAR-10 was initially aimed at classification tasks, it was also used for training auto-encoders (TANG; DENG; HUANG, 2016; ZHANG; XUE; DANA, 2017; MINEMOTO et al., 2017). Auto-encoding involves obtaining a model capable of compressing data in a way that preserves the maximum amount of information possible. In practice, a high-dimensional object – in this case, an image – is first represented (encoded) in a different space, then reconstructed from the compressed information. This is a vital

task in many fields such as information theory and automated control, for example, where minimal information loss is the key objective. A neural network designed to perform an auto-encoding task, known as an auto-encoder, is trained using a set in which the input and the desired output are the same data. On top of that, the dimension of the intermediate layers must be smaller than that of the data being compressed, thus forcing the network to learn a representation for a high-dimensional object in the space of a smaller dimension in an efficient way. Auto-encoders are known as robust feature detectors for their ability to learn representations that preserve information, and are used in unsupervised pre-training on large datasets. There is a significant similarity between auto-encoders and convolutional layers, which act as feature extractors in deep learning models. Auto-encoders also have applications in modern generative models (GÉRON, 2017).

In this work, a total of 7 Hv-ELM models have been trained with 10.000 images of a training batch of the CIFAR-10 and tested using the original test batch of the same dataset, meaning the test set also consists of $10,000$ images. The hypercomplex-valued architectures evaluated are based on the 7 algebras with multiplication tables given in Tables 2 and 3. Additionally, we include an equivalent real-valued ELM for comparison.

Images in the CIFAR-10 dataset are represented as 32×32 8-bit RGB arrays, i.e., a total of 3072 values per image. An 8-bit RGB image $\mathbf{I}$ has been flattened to a real-valued vector $\boldsymbol{x}^{(\mathbb{R})}$ of length 3072 by concatenating the pixel values in the red, green, and blue channels in that order. The values were also re-scaled to fit the symmetric interval $[-1, +1]$. For the hypercomplex-valued encoding, each image $\mathbf{I}$ was expressed as a hypercomplex-valued vector $\boldsymbol{x}^{(\mathbb{A})}$ of length 1024 with components

$$x_i^{(\mathbb{A})} = \left(\frac{2\mathbf{I}_i^R}{255} - 1\right)\boldsymbol{i} + \left(\frac{2\mathbf{I}_i^G}{255} - 1\right)\boldsymbol{j} + \left(\frac{2\mathbf{I}_i^B}{255} - 1\right)\boldsymbol{k},$$

where $\mathbf{I}_i^R, \mathbf{I}_i^G, \mathbf{I}_i^B$ represent the values of the i-th pixel respectively at the red, green, and blue channels. Hence, each of the components of the input is also scaled to fit the $[-1, 1]$ interval.

In order to make comparable experiments, we considered real- and hypercomplex-valued ELMs with similar total number of **free** parameters (TNP). We recall that input-output pairs are symmetric and that hidden layer parameters are fixed, therefore not free. Thus, the real-valued and hypercomplex-valued ELM have respectively

$$\text{TNP}^{(\mathbb{R})} = D^{(\mathbb{R})}L^{(\mathbb{R})} \quad \text{and} \quad \text{TNP}^{(\mathbb{A})} = 4(D^{(\mathbb{A})}L^{(\mathbb{A})}), \tag{3.15}$$

where $D^{(\mathbb{R})} = 3072$ and $D^{(\mathbb{A})} = 1024$ denote the input dimension of each network, and $L^{(\mathbb{R})}$ and $L^{(\mathbb{A})}$ denote the number of neurons in the hidden layer of the real- and hypercomplex-valued models, respectively. Guided by what was reported previously by Minemoto et al. (MINEMOTO et al., 2017) and further experimentation by us (VIEIRA; VALLE, 2020), the number of hidden layer neurons were taken as $L^{(\mathbb{R})} = 600$ and $L^{(\mathbb{A})} = 450$. This way, models add up to an equal TNP, that is, $\text{TNP}^{(\mathbb{R})} = \text{TNP}^{(\mathbb{A})} = 1,843,200$.

The networks used did not have bias terms in any of the layers. The hidden layer's activation function was the hyperbolic tangent, namely the real-valued one for the real ELM and the split version for the hypercomplex ELMs:

$$\tanh(x_0 + x_1\boldsymbol{i} + x_2\boldsymbol{j} + x_3\boldsymbol{k}) = \tanh(x_0) + \tanh(x_1)\boldsymbol{i} + \tanh(x_2)\boldsymbol{j} + \tanh(x_3)\boldsymbol{k}, \quad (3.16)$$

i.e., the application of $\tanh(\cdot)$ to each component of the hypercomplex number individually. This is common practice in hypercomplex-valued networks as the universal approximation property holds for these split activation functions. The hidden layer parameters were randomly generated according to a standard normal distribution, mean 0 and variance 1. The normally distributed parameters were re-scaled by a constant α based on the network's input length, so as to ensure that the internal state values did not saturate the activation function. The precise values used are $\alpha^{(\mathbb{R})} = 30/3072$ and $\alpha^{(\mathbb{A})} = 10/1024$ for the real and hypercomplex-valued models, respectively.

The first training batch of the CIFAR-10 dataset was used to train the eight ELM auto-encoders. The test batch, containing 10,000 different images, was used for testing. For illustrative purposes, Figure 4 depicts an example from the training set while Figure 5 depicts one from the test set. Each figure consists of nine images, the first one being the original CIFAR dataset image, the remaining ones being the eight decoded images, i.e., the output image reconstruction by each trained auto-encoder.

By visual inspection it can be seen that all auto-encoders performed well in both training and test images. The images were reconstructed with a good resemblance to the original one. However, even to the naked eye it is apparent that the reconstructed images are blurry. This effect is expected since the encoding of the images in a lower-dimensional object is bound to incur some level of information loss. In order to evaluate performances from a quantitative perspective, the peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) were used. These metrics are calculated considering the desired and auto-encoders outputs: on the one hand the PSNR is measured in a logarithmic scale inversely proportional to the mean squared error, thus a higher PSNR value indicates a higher reconstruction quality. On the other hand, the SSIM is a unitary real-valued index ($[-1, +1]$); higher values represent the high similarity between the input and the output images, with 1.0 representing perfect reconstruction. Both metrics are widely used and, together, they cover the concept of similarity in great detail. The metrics attained by the ELM auto-encoders are reported in Table 4 as well as represented by the boxplots shown in Figure 6.

From Table 4 it can be seen that the eight auto-encoders yielded similar PSNR and SSIM rates when comparing training and test results for each model individually. That is, the ELMs learned the auto-encoding task with adequate generalization capability, without overfitting to the training data. This is further backed up by the both sets having the same amount of images, which ensures that the models were thoroughly tested.

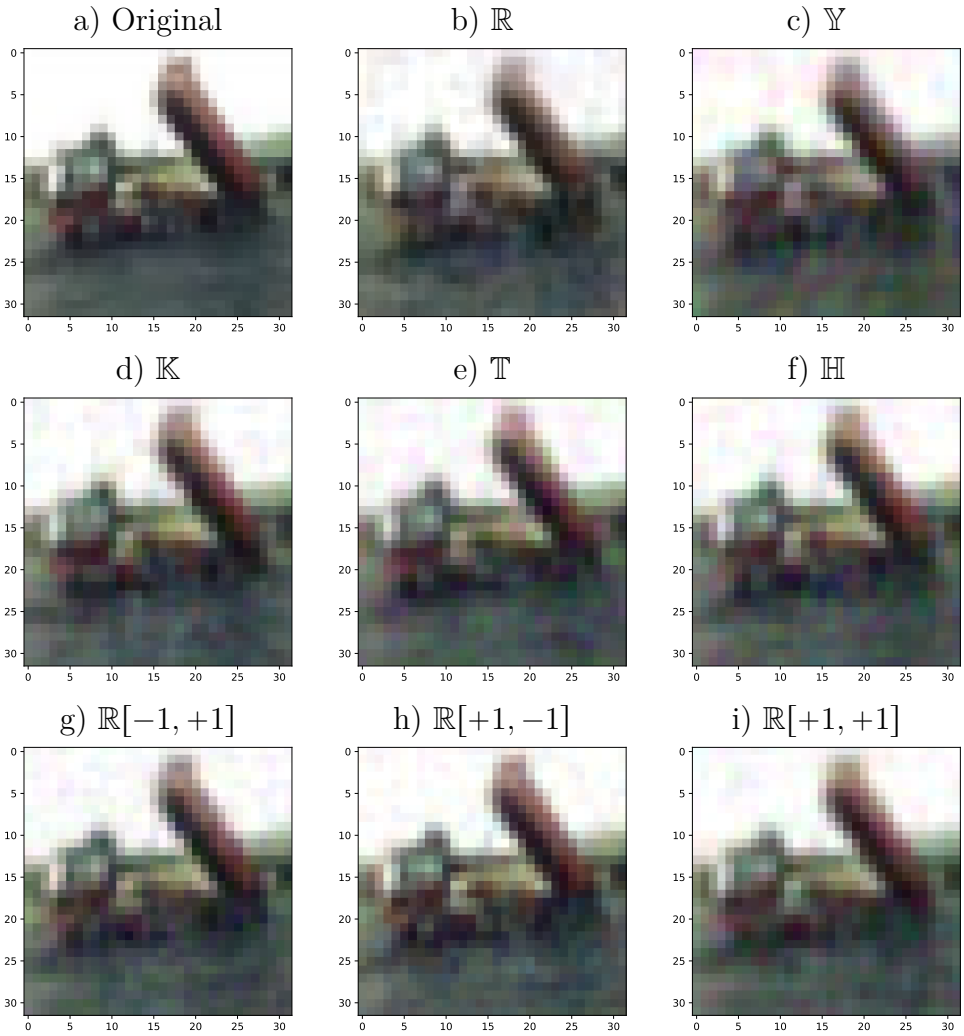

Figure 4 – Training set samples: original color image from the CIFAR dataset and the corresponding images decoded by the real and hypercomplex-valued auto-encoders.

Furthermore, except for the ELM based on MacFarlane's hyperbolic quaternions, the hypercomplex-valued auto-encoders outperformed the real-valued model by a noticeable margin. An outlier to the hypercomplex-valued models, the hyperbolic quaternion-valued ELM was the single one to perform slightly worse than the real-valued network. The auto-encoders based on quaternions, tessarines, Klein four-group, and the Cayley-Dickson algebra $\mathbb{R}[+1,+1]$ yielded similar performance among the hypercomplex-valued models. Finally, the ELM models based on the Cayley-Dickson algebras $\mathbb{R}[-1,+1]$ and $\mathbb{R}[+1,-1]$ pulled ahead by an outstanding margin. These top-performing models boast an almost perfect score in terms of SSIM.

With regards to computational complexity, the hypercomplex-valued ELMs are much more time consuming than their real-valued equivalent. The computational bottlenecks are mainly due to the transformations Φ_L and Φ_R required on (2.23), (2.26), and (3.6). For example, in our experiment, the training step took 3.25s for the real-valued

Figure 5 – Test set samples: original color image from the CIFAR dataset and the corresponding images decoded by the real and hypercomplex-valued auto-encoders.

model and a whopping 283.76s for the hypercomplex-valued models, on average. This amounts to roughly 87 times more training time. As a brief disclaimer, we would like to highlight that we implemented suboptimal codes for the hypercomplex-valued ELM models and strongly believe that this gap in time-wise performance can be greatly reduced with the proper implementation of more optimized hypercomplex-valued operations.

Remark 3. *The experiments above were run on an environment within Google Colaboratory and reported times were given by the platform itself.*

3.5 Concluding Remarks

Extreme learning machines are known for being light-weight models regarding training time. As such, the real-valued ELM is extremely fast in the learning step despite the dataset being so large, with a total of 10.000 images of size 32×32. The hypercomplex-

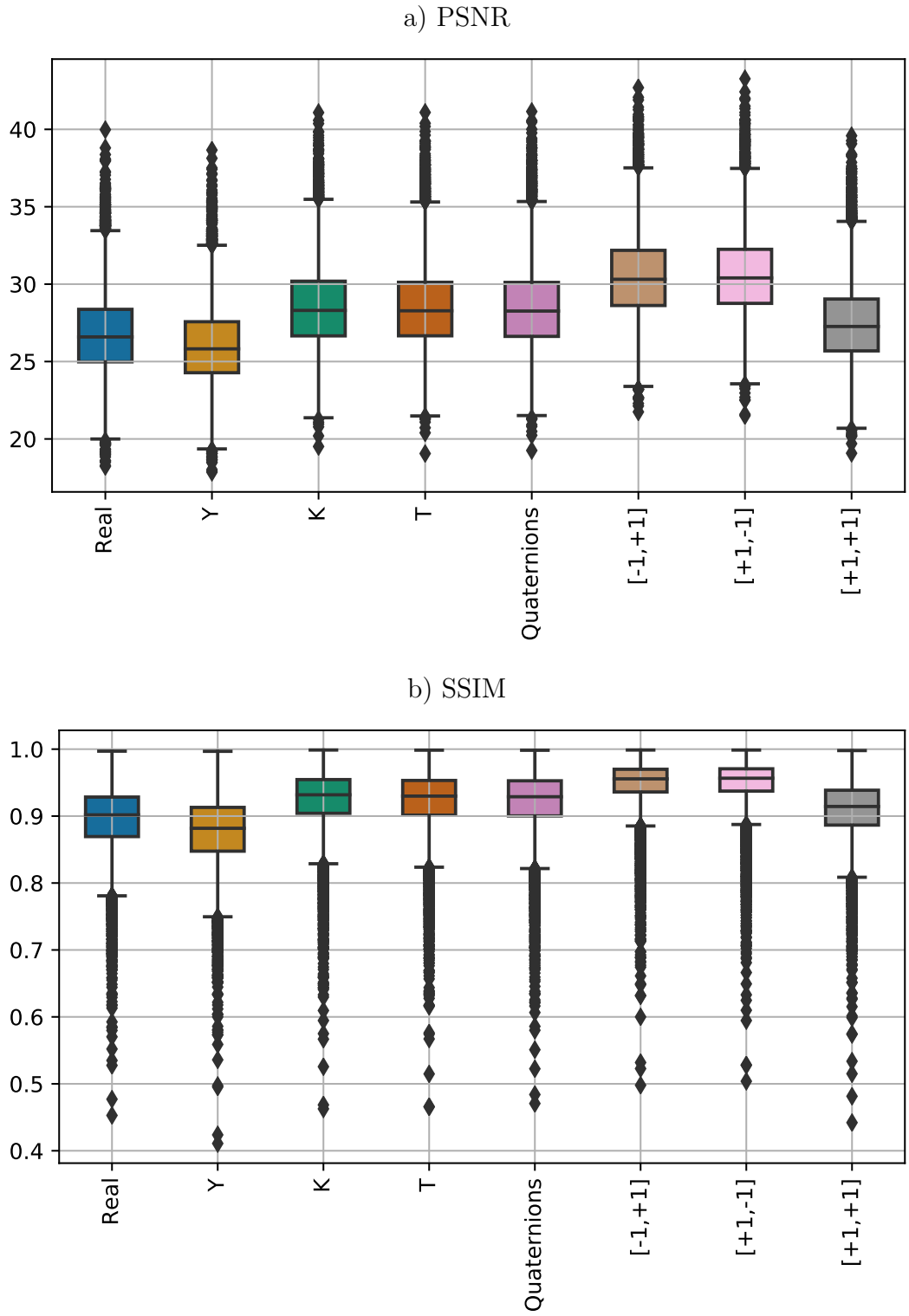

Figure 6 – PSNR and SSIM rates produced by the real and hypercomplex-valued auto-encoders in the test set.

Table 4 – Average PSNR and SSIM rates achieved by real and hypercomplex-valued auto-encoders.

	Train Set		**Test Set**	
Algebra	PSNR	SSIM	PSNR	SSIM
Real	27.3 ± 2.4	0.91 ± 0.05	26.8 ± 2.6	0.89 ± 0.05
$\mathbb{Y}$	26.4 ± 2.4	0.89 ± 0.05	26.0 ± 2.6	0.88 ± 0.05
$\mathbb{K}_4$	28.9 ± 2.5	0.93 ± 0.04	28.5 ± 2.7	0.92 ± 0.05
$\mathbb{T}$	28.9 ± 2.5	0.93 ± 0.04	28.5 ± 2.7	0.92 ± 0.05
$\mathbb{H}$	28.9 ± 2.5	0.93 ± 0.04	28.5 ± 2.7	0.92 ± 0.05
$\mathbb{R}[-1,+1]$	$\mathbf{31.0 \pm 2.5}$	$\mathbf{0.95 \pm 0.03}$	$\mathbf{30.5 \pm 2.7}$	$\mathbf{0.95 \pm 0.04}$
$\mathbb{R}[+1,-1]$	$\mathbf{31.1 \pm 2.5}$	$\mathbf{0.95 \pm 0.03}$	$\mathbf{30.6 \pm 2.7}$	$\mathbf{0.95 \pm 0.04}$
$\mathbb{R}[+1,+1]$	27.9 ± 2.4	0.92 ± 0.04	27.5 ± 2.6	0.91 ± 0.05

valued variants take notably more time to train, showcasing roughly 87 times longer training steps. Nonetheless, Hv-ELMs still stand out as fast models for a task considering such a dataset, thus preserving the core advantage of the original ELM concept.

Performance-wise the hypercomplex-valued models show unparalleled results considering both metrics, PSNR and SSIM. In fact, all hypercomplex models with exception of the one based on the MacFarlane's hyperbolic quaternions outperformed the real-valued baseline by a wide margin. Moreover, our experiments show that a few algebras stand out among the others, as we observe the ELMs based on the Cayley-Dickson algebras $\mathbb{R}[-1,+1]$ and $\mathbb{R}[+1,-1]$ achieve near perfect SSIM scores. We recall that all models have identical total number of parameters (TNP), and thus the storage requirement is the same for all of them.

In sum, the hypercomplex-valued ELMs pose a trade-off: they hold a strict advantage over the real-valued counterpart in terms of performance, while maintaining the same storage requirements and requiring notably longer time on the training steps. However, the main advantage of ELMs is that of having a short training step, thus Hv-ELMs are still fast models despite being much slower than the real-valued version. Moreover, the prediction time for both models is negligible, so once trained the Hv-ELM poses a near-strict advantage over the real-valued ELM.

4 Hv-CNN: Hypercomplex-valued Convolutional Neural Network

To introduce convolution in the context of neural networks we first briefly review the mathematical concept from which it was derived. The convolution is a mathematical operation $(f * g)(t)$ on two real-valued functions f, g that can be interpreted as a weighted average of the value of f around a point t where the weights are given by the function g. The convolution is defined formally as follows, for a continuous variable t:

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(t)g(t-x)dx. \tag{4.1}$$

In case the variable t is discrete in uniform intervals, Eq. (4.1) becomes

$$(f * g)(t) = \sum_{n=-\infty}^{+\infty} f(n)g(t-n). \tag{4.2}$$

Finally, the function g can be null outside of a finite set, so that the above sum is finite. This is often the case in image processing and convolutional layers in particular. It is sometimes useful to use the same weight function g for a handful of different functions f. In these cases it is common to refer to the function g as **kernel** and f as **input**.

Convolutional layers are particular types of layers in which the trainable parameters are arranged in spatial structures called filters (GOODFELLOW; BENGIO; COURVILLE, 2016). The filter structures allow the network to process data in a locally cohesive manner, thus identifying local patterns. Convolutional neural networks are networks equipped with one or more such layers, and are so named because the filters act as the kernels and the data being processed acts as the input in (4.2). During the training step the parameters in the filters are adjusted and thus the network is able to learn local features, acting as trainable encoders. These networks have been widely applied to image processing tasks, taking full advantage of the spatial nature of its learning mechanism and the translation invariance of filters.

4.1 Real-valued Convolutional Layers

Let us consider a convolutional layer being fed by a real-valued image $\mathbf{I}$ containing C channels. We shall express the intensity of the cth channel of the image $\mathbf{I}$ at pixel p as $\mathbf{I}(p, c) \in \mathbb{R}$. By definition, the filters must have the same number C of channels as the image $\mathbf{I}$. Filters have an associated area shape in which they act called domain. The most common domains are square grids. Here we shall denote the domain by D. Then,

the synaptic weights of a convolutional layer with K real-valued filters are compactly expressed in an array $\mathbf{F}$ such that $\mathbf{F}(q, c, k)$ denotes the weight of the cth channel of the kth filter associated to the point $q \in D$, with $c = 1, \ldots, C$ and $k = 1, \ldots, K$. Consequently, the output of a convolutional layer with K filters is a real-valued image $\mathbf{J}$ with K feature channels, one per filter. These are obtained by calculating the convolution of the image patch under D by the respective weights in the filter, adding a bias term (associated to the channel) and lastly applying an activation function. This is similar to the feedforward step of the common neuron structure in MLPs, for example. Visually this operation corresponds to superimposing the filter over the image, centered on a pixel p, and calculating the linear combination of the elements in the domain D in which the weights are given by the filter values and the variables are given by the intensities of the underlying pixels. Formally let us write the convolution of the image $\mathbf{I}$ by the kth filter at pixel p as $(\mathbf{I} * \mathbf{F})(p, k)$. Then, mathematically we have

$$(\mathbf{I} * \mathbf{F})(p, k) = \sum_{c=1}^{C} \sum_{q \in D} \mathbf{I}\big(p + S(q), c\big)\mathbf{F}(q, c, k), \tag{4.3}$$

in which the term $S(q)$, for $q \in D$, represents a translation that takes vertical and horizontal strides into account. Note that (4.3) computes a cross correlation. Finally, the intensity of the output of this channel in this particular pixel is directly obtained as

$$\mathbf{J}(p, k) = \varphi\left(b(k) + (\mathbf{I} * \mathbf{F})(p, k)\right), \tag{4.4}$$

where $\varphi : \mathbb{R} \to \mathbb{R}$ is the activation function and $b(k)$ is the bias. Note that each channel has its own bias term.

4.2 Hypercomplex-valued Convolutional Layers

With the previous section in hand we can extend real-valued convolutional layer concepts in order to obtain the hypercomplex-valued convolutional layer. Indeed, this is done simply by replacing the real numbers and operations with their corresponding hypercomplex versions in (4.3) and (4.4) (TRABELSI et al., 2017; GAUDET; MAIDA, 2017). Moreover, the intensity of the kth channel of the hypercomplex-valued output image $\mathbf{J}^{(h)}$ at pixel p is simply

$$\mathbf{J}^{(h)}(p, k) = \varphi_{\mathbb{A}}\left(b^{(h)}(k) + (\mathbf{I}^{(h)} * \mathbf{F}^{(h)})(p, k)\right), \tag{4.5}$$

with $\varphi_{\mathbb{A}} : \mathbb{A} \to \mathbb{A}$ being a hypercomplex-valued activation function and $b_k^{(h)} \in \mathbb{A}$ is the bias term associated with that channel. Here

$$(\mathbf{I}^{(h)} * \mathbf{F}^{(h)})(p, k) = \sum_{c=1}^{C} \sum_{q \in D} \mathbf{I}^{(h)}(p + S(q), c)\mathbf{F}^{(h)}(q, c, k), \tag{4.6}$$

is the convolution of $\mathbf{I}^{(h)}$ by the kth hypercomplex-valued filter at pixel p. Note that the expression is similar to that in (4.3), except that the product is carried out in $\mathbb{A}$. In order to compute these terms, we emulate these products by means of real-valued matrix-vector products by arranging the filter matrices in a similar fashion to what is described in section 2.2.

Once again, in this work we only consider split-functions $\varphi_{\mathbb{A}}$ defined as in (3.16). That is, based on a real-valued function $\varphi : \mathbb{R} \to \mathbb{R}$ we have

$$\varphi_{\mathbb{A}}(x) = \varphi(x_0) + \varphi(x_1)\boldsymbol{i}_1 + \ldots + \varphi(x_n)\boldsymbol{i}_n, \tag{4.7}$$

for all $x \in x_0 + x_1\boldsymbol{i}_1 + \cdots + x_n\boldsymbol{i}_n \in \mathbb{A}$.

4.3 Emulating Hypercomplex-valued Convolutional Layers

As previously mentioned, the vast majority of deep learning libraries and tools are designed for real-valued inputs using floating point operations. In a way akin to what was done in Section 2.2, we show how to emulate the operations in four-dimensional hypercomplex-valued convolutional layers by means of real-valued convolutions. This is also similar to other approaches reported in the literature for complex- and quaternion-valued deep networks (TRABELSI et al., 2017; GAUDET; MAIDA, 2017).

For this definition let us consider a generalized multiplication table for four-dimensional algebras as

	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	s_{11}	$s_{12}\boldsymbol{k}$	$s_{13}\boldsymbol{j}$
$\boldsymbol{j}$	$s_{21}\boldsymbol{k}$	s_{22}	$s_{23}\boldsymbol{i}$
$\boldsymbol{k}$	$s_{31}\boldsymbol{j}$	$s_{32}\boldsymbol{i}$	s_{33}

. (4.8)

Note that all multiplication tables for the notable algebras presented in Tables 2 and 3 can be obtained by setting the s_{ij}'s in (4.8) equal to ± 1. Thus, the expressions for the emulation of hypercomplex-valued convolutions are valid for the algebras presented previously in this work.

First off, we represent an image $\mathbf{I}^{(h)}$ with C channels on a four-dimensional hypercomplex algebra as

$$\mathbf{I}^{(h)} = \mathbf{I}_0 + \mathbf{I}_1\boldsymbol{i} + \mathbf{I}_2\boldsymbol{j} + \mathbf{I}_3\boldsymbol{k}, \tag{4.9}$$

with $\mathbf{I}_0$, $\mathbf{I}_1$, $\mathbf{I}_2$, and $\mathbf{I}_3$ representing real-valued images also with C channels each. Analogously, a collection of K hypercomplex-valued filters with domain D and C feature channels is represented as

$$\mathbf{F}^{(h)} = \mathbf{F}_0 + \mathbf{F}_1\boldsymbol{i} + \mathbf{F}_2\boldsymbol{j} + \mathbf{F}_3\boldsymbol{k}, \tag{4.10}$$

where $\mathbf{F}_0$, $\mathbf{F}_1$, $\mathbf{F}_2$, and $\mathbf{F}_3$ are real-valued arrays each representing a collection of K filters. Naturally these filter also have domain D and C feature channels. To compute the products on the right hand side of (4.6) we simply apply the distributive law and use the associated multiplication table. Formally, we follow the rules from (4.8) to obtain

$$\begin{aligned}
&\mathbf{I}^{(h)}(p + S(q), c)\mathbf{F}^{(h)}(q, c, k) \\
&= (\mathbf{I}_0 + \mathbf{I}_1\boldsymbol{i} + \mathbf{I}_2\boldsymbol{j} + \mathbf{I}_3\boldsymbol{k})(\mathbf{F}_0 + \mathbf{F}_1\boldsymbol{i} + \mathbf{F}_2\boldsymbol{j} + \mathbf{F}_3\boldsymbol{k}) \\
&= \mathbf{I}_0\mathbf{F}_0 + s_{11}\mathbf{I}_1\mathbf{F}_1 + s_{22}\mathbf{I}_2\mathbf{F}_2 + s_{33}\mathbf{I}_3\mathbf{F}_3 \\
&\quad + (\mathbf{I}_0\mathbf{F}_1 + \mathbf{I}_1\mathbf{F}_0 + s_{23}\mathbf{I}_2\mathbf{F}_3 + s_{32}\mathbf{I}_3\mathbf{F}_2)\boldsymbol{i} \\
&\quad + (\mathbf{I}_0\mathbf{F}_2 + s_{13}\mathbf{I}_1\mathbf{F}_3 + \mathbf{I}_2\mathbf{F}_0 + s_{31}\mathbf{I}_3\mathbf{F}_1)\boldsymbol{j} \\
&\quad + (\mathbf{I}_0\mathbf{F}_3 + s_{12}\mathbf{I}_1\mathbf{F}_2 + s_{21}\mathbf{I}_2\mathbf{F}_1 + \mathbf{I}_3\mathbf{F}_0)\boldsymbol{k},
\end{aligned}$$

in which we omit the arguments $(p + S(q), c)$ and (q, c, k) to help with readability. Now, by isolating the real-part of the convolution in (4.6), we see that

$$\begin{aligned}
\left(\mathbf{I}^{(h)} * \mathbf{F}^{(h)}\right)_0(p, k) = \sum_{c=1}^{C}\sum_{q\in D}\Big[&\mathbf{I}_0(p + S(q), c)\mathbf{F}_0(q, c, k) \\
&+ s_{11}\mathbf{I}_1(p + S(q), c)\mathbf{F}_1(q, c, k) \\
&+ s_{22}\mathbf{I}_2(p + S(q), c)\mathbf{F}_2(q, c, k) \\
&+ s_{33}\mathbf{I}_3(p + S(q), c)\mathbf{F}_3(q, c, k)\Big],
\end{aligned} \tag{4.11}$$

Alternatively, this term can be computed using the real-valued convolution

$$\left(\mathbf{I}^{(h)} * \mathbf{F}^{(h)}\right)_0(p, k) = \left(\mathbf{I}^{(r)} * \mathbf{F}_0^{(r)}\right)(p, k), \tag{4.12}$$

where $\mathbf{I}^{(r)}$ is a real-valued image with $4C$ feature channels obtained simply by concatenating the real and imaginary parts of $\mathbf{I}^{(h)}$. Mathematically, this means

$$\mathbf{I}^{(r)}(p, :) = [\mathbf{I}_0(p, :), \mathbf{I}_1(p, :), \mathbf{I}_2(p, :), \mathbf{I}_3(p, :)], \tag{4.13}$$

for each pixel p. The alternative filter $\mathbf{F}_0^{(r)}$ is the real-valued filter defined by

$$\mathbf{F}_0^{(r)}(q, 1 : C, k) = \mathbf{F}_0(q, 1 : C, k), \tag{4.14}$$

$$\mathbf{F}_0^{(r)}(q, C + 1 : 2C, k) = s_{11}\mathbf{F}_1(q, 1 : C, k), \tag{4.15}$$

$$\mathbf{F}_0^{(r)}(q, 2C + 1 : 3C, k) = s_{22}\mathbf{F}_2(q, 1 : C, k), \tag{4.16}$$

$$\mathbf{F}_0^{(r)}(q, 3C + 1 : 4C, k) = s_{33}\mathbf{F}_3(q, 1 : C, k), \tag{4.17}$$

for all $q \in D$ and $k = 1, \ldots, K$. For short, the notation

$$\mathbf{F}_0^{(r)} = [\mathbf{F}_0, s_{11}\mathbf{F}_1, s_{22}\mathbf{F}_2, s_{33}\mathbf{F}_3], \tag{4.18}$$

means that $\mathbf{F}_0^{(r)}$ is obtained by concatenating $\mathbf{F}_0$, $s_{11}\mathbf{F}_1$, $s_{22}\mathbf{F}_2$, and $s_{33}\mathbf{F}_3$ as prescribed above. Once again we note that this is similar to the way we emulate products of hypercomplex numbers by real-valued matrix-vector products, which is simply a concatenation

of the real-valued components with auxiliary terms (in this case the s_{ij}'s) conveying the information of the multiplication table of the underlying algebra. Note from (4.3) that

$$\begin{aligned}
\big(\mathbf{I}^{(r)} * \mathbf{F}_0^{(r)}\big)(p,k) &= \sum_{c=1}^{4C} \sum_{q\in D} \mathbf{I}^{(r)}\big(p+S(q),c\big)\mathbf{F}_0^{(r)}(q,c,k) \\
&= \sum_{q\in D}\Bigg[\sum_{c=1}^{C} \mathbf{I}^{(r)}\big(p+S(q),c\big)\mathbf{F}_0^{(r)}(q,c,k) + \dots \\
&\quad + \sum_{c=3C+1}^{4C} \mathbf{I}^{(r)}\big(p+S(q),c\big)\mathbf{F}_0^{(r)}(q,c,k)\Bigg] \\
&= \sum_{q\in D}\Bigg[\sum_{c=1}^{C} \mathbf{I}_0\big(p+S(q),c\big)\mathbf{F}_0(q,c,k) + \dots \\
&\quad + \sum_{c'=1}^{C} s_{33}\mathbf{I}_3\big(p+S(q),c'\big)\mathbf{F}_3(q,c',k)\Bigg],
\end{aligned}$$

which corroborates the expression for $\big(\mathbf{I}^{(h)} * \mathbf{F}^{(h)}\big)_0(p,k)$ given by (4.11). Hence, in a network using a split-function $\varphi_{\mathbb{A}}$ based on a real-valued function φ, the real-part $\mathbf{J}_0(p,k)$ of the output $\mathbf{J}^{(h)}(p,k)$ of the hypercomplex-valued convolutional layer can be obtained simply by

$$\mathbf{J}_0(p,k) = \varphi\Big(b_0(k) + \big(\mathbf{I}^{(r)} * \mathbf{F}_0^{(r)}\big)(p,k)\Big). \tag{4.19}$$

Here we denote as $b_0(k)$ the real-part of the bias term $b(k)$, and $\mathbf{I}^{(r)}$ and $\mathbf{F}_0^{(r)}$ are the previously given expressions in (4.13) and (4.18), respectively. The same reasoning yields

$$\mathbf{F}_1^{(r)} = [\mathbf{F}_1, \mathbf{F}_0, s_{23}\mathbf{F}_3, s_{32}\mathbf{F}_2], \tag{4.20}$$

$$\mathbf{F}_2^{(r)} = [\mathbf{F}_2, s_{13}\mathbf{F}_3, \mathbf{F}_0, s_{31}\mathbf{F}_1], \tag{4.21}$$

$$\mathbf{F}_3^{(r)} = [\mathbf{F}_3, s_{12}\mathbf{F}_2, s_{21}\mathbf{F}_1, \mathbf{F}_0], \tag{4.22}$$

as the filters used to calculate the three imaginary parts $\mathbf{J}_1(p,k)$, $\mathbf{J}_2(p,k)$, and $\mathbf{J}_3(p,k)$, respectively. Naturally each imaginary part uses its associated part of the bias term, namely $b_1(k)$, $b_2(k)$, and $b_3(k)$.

There are many advantages to the emulation procedure described in this section. First off, the established equivalence makes it possible to implement hypercomplex-valued convolutional layers using fast open-source deep learning libraries such as `Tensorflow` and `PyTorch` for `python` and `Flux` for `Julia Language`. This directly leads to the possibility of implementing hypercomplex-valued versions of well-known deep neural network models in these libraries with ease. Moreover, many machine learning techniques can be incorporated naturally due to being already available in these libraries such as batch normalization and various weight initialization paradigms. Finally, this bridge between hypercomplex-valued models and modern libraries brings immediate benefits such as a handful of state-of-the-art optimizers and multiple types of layers and operators such as flattening and pooling.

To exemplify the introduced concepts we describe a hypercomplex-valued deep neural network used in the classification task of lymphocytes from blood smear images. The implemented model takes inspiration in the LeNet architecture (LECUN et al., 1998).

4.4 Max-pooling Layer

A pooling layer operates a downsampling effect in the input. In addition, this layer structure contains no trainable parameters. The most common pooling layers are the max and average pooling layers. In this work in particular we use the max pooling layer, exclusively. Roughly speaking, a max pooling layer has a kernel shape, usually a rectangular grid G, and it operates by collapsing each set of pixels contained in the grid into the single maximum value present. This operation reduces the dimensionality of the input while also highlighting the "stronger" signal in each window. The max pooling operation is conducted on each filter separately, i.e., it acts as a split maximum function for elements of a hypercomplex algebra. Formally, if the output of a 4-dimensional hypercomplex-valued filter $\mathbf{J}^{(h)}$ is such that

$$\mathbf{J}^{(h)} = \mathbf{J}_0 + \mathbf{J}_1\boldsymbol{i} + \mathbf{J}_2\boldsymbol{j} + \mathbf{J}_3\boldsymbol{k}, \tag{4.23}$$

then $\mathbf{J}^{(h)}$ after the max-pooling operation is simply

$$\mathtt{MP}(\mathbf{J}^{(h)}) = \mathtt{MP}(\mathbf{J}_0) + \mathtt{MP}(\mathbf{J}_1)\boldsymbol{i} + \mathtt{MP}(\mathbf{J}_2)\boldsymbol{j} + \mathtt{MP}(\mathbf{J}_3)\boldsymbol{k}, \tag{4.24}$$

where $\mathtt{MP}(\mathbf{J}_i)$ is the max-pooling operation applied to the i-th real matrix.

4.5 Application: Lymphoblast Image Classification

In this section we describe the experiment conducted to showcase the proposed hypercomplex-valued convolutional neural network (Hv-CNN). It consists in a classification task in a medical-image dataset containing blood smear images.

Acute Lymphoblastic Leukemia (ALL) is a rare type of blood cancer that occurs more frequently in children of ages 2-5 and can be lethal in under a few weeks if left undiagnosed. The main indicator of ALL is the presence of lymphoblasts, a type of malformed lymphocyte, in the blood stream. The most common diagnosis method is the inspection of microscopic blood smear images. The ALL-IDB (LABATI; PIURI; SCOTTI, 2011) is a public benchmark aimed at computer assisted ALL diagnosis and consists of 2 datasets: one directed at a segmentation and classification, and the other directly aimed at the classification task itself. In this work we use the latter dataset which contains 260 images, each containing a single blood element, and perform a binary classification task in which the model decides whether or not the presented image is a lymphoblast. Figure 7 shows examples of a probable lymphoblast and a healthy cell, respectively.

a) Probable lymphoblast b) Healthy cell

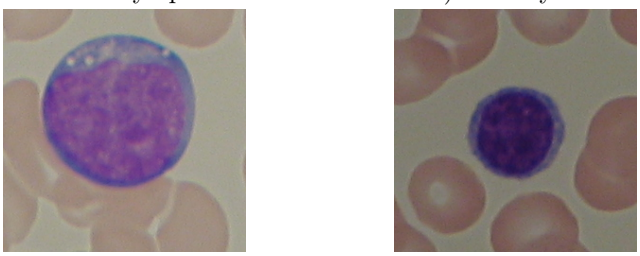

Figure 7 – Example of images from the ALL-IDB dataset used for the classification task.

The original images are encoded in RGB channels. This means each pixel contains 3 values representing the red, green and blue values respectively. An alternate encoding for images is the HSV, which stands for hue, saturation, and value. This color scheme displays colors in a radial slice, and better represents the human eye perception of color elements. In mathematical terms, an HSV encoded color pixel is represented as follows in a four-dimensional hypercomplex algebra:

$$\mathbf{I}(p) = \big(S(p) + V(p)\boldsymbol{i}\big)\big(\cos(H(p)\big) + \operatorname{sen}\big(H(p)\big)\boldsymbol{j}\big), \tag{4.25}$$

where $H(p) \in [0, 2\pi)$ and $S(p), V(p) \in [0, 1]$ denote respectively the hue, saturation, and value, of pixel p.

We conducted experiments in 2 different settings with this dataset. For both experiments we performed a 50%-50% train-test split, i.e., used half of the images for training and half for testing. We also performed data augmentation on the training set using horizontal and vertical flips to increase the number of examples. The next subsections detail the experiments individually.

4.5.0.1 Experiment I: General algebras

This experiment is part of the paper (VIEIRA; VALLE, 2022a). We define four-dimensional algebras in a more general manner for this experiment as follows.

We start by imposing that the product of hypercomplex units is associative. Furthermore, we let the identity

$$\boldsymbol{k} = \boldsymbol{ij}, \tag{4.26}$$

hold true. Finally, we assume the four-dimensional hypercomplex algebra is either commutative or anti-commutative.

- **Anticommutative Algebras:** We obtain an anticommutative four-dimensional hypercomplex algebra by imposing $\boldsymbol{ij} = -\boldsymbol{ji}$. From the associativity and (4.26), we

Table 5 – Multiplication tables of the anticommutative algebras.

Quaternions

$A[-1,-1]$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	-1	$\boldsymbol{k}$	$-\boldsymbol{j}$
$\boldsymbol{j}$	$-\boldsymbol{k}$	-1	$\boldsymbol{i}$
$\boldsymbol{k}$	$\boldsymbol{j}$	$-\boldsymbol{i}$	-1

$C\ell_{2,0}$

$A[+1,+1]$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	1	$\boldsymbol{k}$	$\boldsymbol{j}$
$\boldsymbol{j}$	$-\boldsymbol{k}$	1	$-\boldsymbol{i}$
$\boldsymbol{k}$	$-\boldsymbol{j}$	$\boldsymbol{i}$	-1

Coquaternions

$A[-1,+1]$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	-1	$\boldsymbol{k}$	$-\boldsymbol{j}$
$\boldsymbol{j}$	$-\boldsymbol{k}$	1	$-\boldsymbol{i}$
$\boldsymbol{k}$	$\boldsymbol{j}$	$\boldsymbol{i}$	1

$C\ell_{1,1}$

$A[+1,-1]$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	1	$\boldsymbol{k}$	$\boldsymbol{j}$
$\boldsymbol{j}$	$-\boldsymbol{k}$	-1	$\boldsymbol{i}$
$\boldsymbol{k}$	$-\boldsymbol{j}$	$-\boldsymbol{i}$	1

obtain

$$\boldsymbol{k}^2 = (\boldsymbol{ij})(\boldsymbol{ij}) = \boldsymbol{i}(\boldsymbol{ji})\boldsymbol{j} = \boldsymbol{i}(-\boldsymbol{ij})\boldsymbol{j} = -\boldsymbol{i}^2\boldsymbol{j}^2, \tag{4.27}$$

$$\boldsymbol{ik} = \boldsymbol{i}(\boldsymbol{ij}) = \boldsymbol{i}^2\boldsymbol{j}, \tag{4.28}$$

$$\boldsymbol{jk} = \boldsymbol{j}(\boldsymbol{ij}) = \boldsymbol{j}(-\boldsymbol{ji}) = -\boldsymbol{j}^2\boldsymbol{i}. \tag{4.29}$$

To simplify the exposition, let $\boldsymbol{i}^2 = \gamma_1$ and $\boldsymbol{j}^2 = \gamma_2$. From (4.26)-(4.29), we obtain an associative and anticommutative four-dimensional algebra denoted by $A[\gamma_1, \gamma_2]$, whose multiplication table is then

$A[\gamma_1,\gamma_2]$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	γ_1	$\boldsymbol{k}$	$\gamma_1\boldsymbol{j}$
$\boldsymbol{j}$	$-\boldsymbol{k}$	γ_2	$-\gamma_2\boldsymbol{i}$
$\boldsymbol{k}$	$-\gamma_1\boldsymbol{j}$	$\gamma_2\boldsymbol{i}$	$-\gamma_1\gamma_2$

(4.30)

By considering $\gamma_1, \gamma_2 \in \{-1, +1\}$, we obtain the four-dimensional hypercomplex algebras $A[-1,-1]$, $A[-1,+1]$, $A[+1,-1]$, and $A[+1,+1]$ whose multiplication tables are depicted in Table 5.

Remark 4. *Clearly the algebras $A[-1,-1]$, $A[-1,+1]$, $A[+1,-1]$, and $A[+1,+1]$ can be derived using the generalized Cayley-Dickson process described in Section 2.1 and in (BROWN, 1967; VIEIRA; VALLE, 2020). Notably, the hypercomplex algebra $A[-1,-1]$ is equivalent to the quaternions, which can be easily seen from the identical multiplication tables. On the other hand, the algebra $A[-1,+1]$ corresponds to the co-quaternions or split-quaternions. Finally, $A[+1,+1]$ and $A[+1,-1]$ are identical to the aforementioned Clifford algebras $\mathcal{C}\ell_{2,0}$ and $\mathcal{C}\ell_{1,1}$, respectively. The algebras $A[-1,+1]$, $A[+1,-1]$, and $A[+1,+1]$ are all isomorphic, meaning they can be obtained from one another by a simple change of basis operation, which in this case corresponds to a re-labelling of hyperimaginary units.*

- **Commutative Algebras:** If we instead impose the condition $\boldsymbol{ij} = \boldsymbol{ji}$ we end up with commutative four-dimensional hypercomplex algebras. Once again, using (4.26)

Table 6 – Multiplication tables of commutative algebras.

Bicomplex numbers

$B[-1,-1]$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	-1	$\boldsymbol{k}$	$-\boldsymbol{j}$
$\boldsymbol{j}$	$\boldsymbol{k}$	-1	$-\boldsymbol{i}$
$\boldsymbol{k}$	$-\boldsymbol{j}$	$-\boldsymbol{i}$	1

$B[+1,-1]$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	1	$\boldsymbol{k}$	$\boldsymbol{j}$
$\boldsymbol{j}$	$\boldsymbol{k}$	-1	$-\boldsymbol{i}$
$\boldsymbol{k}$	$\boldsymbol{j}$	$-\boldsymbol{i}$	-1

Tessarines

$B[-1,+1]$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	-1	$\boldsymbol{k}$	$-\boldsymbol{j}$
$\boldsymbol{j}$	$\boldsymbol{k}$	1	$\boldsymbol{i}$
$\boldsymbol{k}$	$-\boldsymbol{j}$	$\boldsymbol{i}$	-1

Klein 4-group

$B[+1,+1]$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	1	$\boldsymbol{k}$	$\boldsymbol{j}$
$\boldsymbol{j}$	$\boldsymbol{k}$	1	$\boldsymbol{i}$
$\boldsymbol{k}$	$\boldsymbol{j}$	$\boldsymbol{i}$	1

and the associativity property, we get the identities

$$\boldsymbol{k}^2 = (\boldsymbol{ij})(\boldsymbol{ij}) = \boldsymbol{i}(\boldsymbol{ji})\boldsymbol{j} = \boldsymbol{i}(\boldsymbol{ij})\boldsymbol{j} = \boldsymbol{i}^2\boldsymbol{j}^2, \tag{4.31}$$

$$\boldsymbol{ik} = \boldsymbol{i}(\boldsymbol{ij}) = \boldsymbol{i}^2\boldsymbol{j}, \tag{4.32}$$

$$\boldsymbol{jk} = \boldsymbol{j}(\boldsymbol{ij}) = \boldsymbol{j}(\boldsymbol{ji}) = \boldsymbol{j}^2\boldsymbol{i}. \tag{4.33}$$

By expressing $\boldsymbol{i}^2 = \gamma_1$ and $\boldsymbol{j}^2 = \gamma_2$, we obtain a commutative hypercomplex algebra $B[\gamma_1, \gamma_2]$ whose multiplication table is

$B[\gamma_1,\gamma_2]$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	γ_1	$\boldsymbol{k}$	$\gamma_1\boldsymbol{j}$
$\boldsymbol{j}$	$\boldsymbol{k}$	γ_2	$\gamma_2\boldsymbol{i}$
$\boldsymbol{k}$	$\gamma_1\boldsymbol{j}$	$\gamma_2\boldsymbol{i}$	$\gamma_1\gamma_2$

(4.34)

Taking $\gamma_1, \gamma_2 \in \{-1, +1\}$ yields the four-dimensional algebras $B[-1,-1]$, $B[-1,+1]$, $B[+1,-1]$ and $B[+1,+1]$, with multiplication tables depicted in Table 6.

Remark 5. *The hypercomplex algebra $B[-1,+1]$ and the tessarines have the same multiplication table, hence they are equivalent. By the same reasoning, the bi-complex numbers are equivalent to the algebra $B[-1,-1]$ while the Klein 4-group is identical to $B[+1,+1]$. Finally, we highlight that $B[-1,-1]$ and $B[+1,-1]$ can be both obtained from $B[-1,+1]$ by a change of bases operation. In other words, the algebras $B[-1,-1]$, $B[+1,-1]$ and $B[-1,+1]$ are all isomorphic.*

In sum, this amounts to eight algebras. As per what was imposed, all of them are associative, half of them are anticommutative while the remaining half are commutative algebras. Furthermore, all the multiplication tables for these algebras follow a pattern, i.e., all can be written as

	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	s_{11}	$s_{12}\boldsymbol{k}$	$s_{13}\boldsymbol{j}$
$\boldsymbol{j}$	$s_{21}\boldsymbol{k}$	s_{22}	$s_{23}\boldsymbol{i}$
$\boldsymbol{k}$	$s_{31}\boldsymbol{j}$	$s_{32}\boldsymbol{i}$	s_{33}

(4.35)

Table 7 – Parameter distribution per layer for each architecture.

		Rv-CNN	Hv-CNN
Conv Layer 1	(3,3) filters	32	8
	Parameters	896	320
Conv Layer 2	(3,3) filters	64	16
	Parameters	18,496	4,672
Conv Layer 3	(3,3) filters	128	32
	Parameters	73,856	18,560
Dense Layer	Units	1	1
	Parameters	12,801	12,801
Total		106,049	36,353

with $s_{ij} \in \{-1, +1\}$, for all $i, j = 1, \ldots, 3$. Note that the s_{ij}'s are uniquely determined by the choice of the parameters γ_1 and γ_2 as well as on the commutativity or anticommutativity of the multiplication. This observation was crucial for the efficient implementation of convolutional neural networks based on the eight hypercomplex algebras presented in this section.

This development leads to a total of 18 configurations featured in this experiment. Namely, each of the 9 models (1 Rv-CNN and 8 Hv-CNNs) was used in the classification task of images encoded in both RGB and HSV color spaces. Following the pipeline detailed by (GRANERO; HERNÁNDEZ; VALLE, 2021) we adopt a similar architecture for real- and hypercomplex-valued models: three convolutional layers with 3×3 filters, each followed by a max-pooling layer with kernels of size 2×2. The output of the third max-pooling layer is flattened and fed to a dense layer featuring a single neuron responsible for outputting the label, i.e., 1 for a lymphoblast image, 0 for a healthy cell. The hypercomplex-valued convolutional layers have a smaller number of filters since each filter in these layers corresponds, in some sense, to four filters in the real-valued convolutional layer. The dense layer is identical in both types of models. The activation functions used are the rectified linear unit (ReLU) and split-ReLU for the real- and hypercomplex-valued models, respectively. A detailed account of the architectures described is comprised in Figure 8 and Table 7. Lastly, all models in this work were implemented using the python libraries `Keras` and `Tensorflow` and were ran using Google Colaboratory. All models used the `'adam'` optimizer with the `BinaryCrossEntropy()` loss function from the `Keras.losses` library. The source codes are available at https://github.com/mevalle/Hypercomplex-valued-Convolutional-Neural-Networks.

The box plots depicted in Fig. 9 and Table 8 summarize the outcome of the computational experiment as measured by the accuracy in the test set. Note that all models showed satisfactory performance for the RGB encoded images, with medians (indicated by the line inside the respective blocks) close to 95% accuracy, with the exception of the Hv-CNN based on the algebra $B[-1, +1]$ boasting a median accuracy rate of 93.8%.

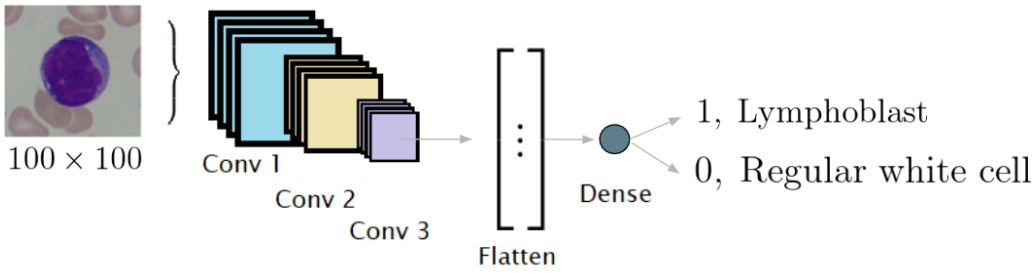

Figure 8 – General architecture of the CNNs used in the experiments.

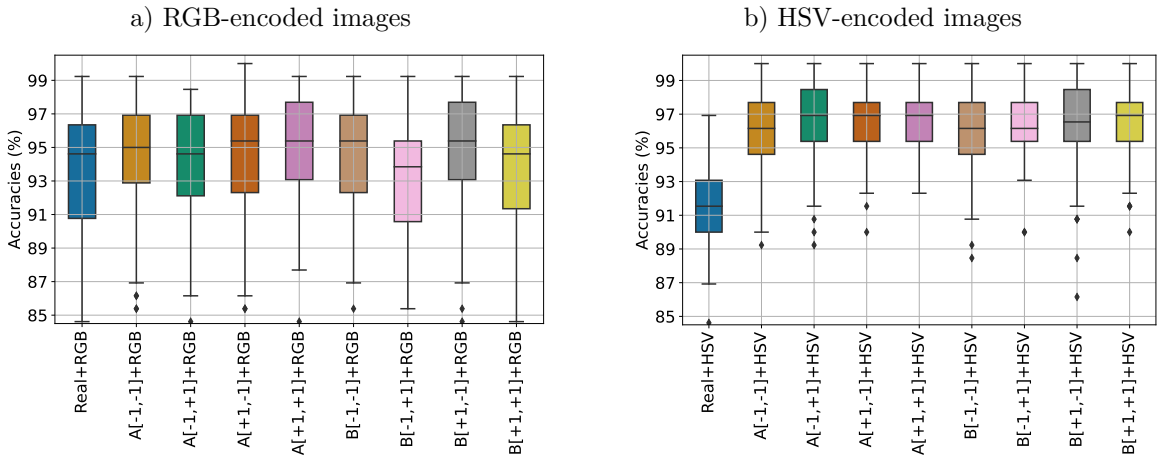

Figure 9 – Boxplot of test set accuracies produced by real-valued and hypercomplex-valued neural networks.

Nevertheless, performances vary more notably in the HSV case. The real-valued model exhibits poor performance when compared to all the hypercomplex-valued ones, showing a very low median accuracy rate of 91.5%. Among the hypercomplex-valued competitors, the Hv-CNNs based on the algebras $A[-1,-1]$, $B[-1,-1]$, and $B[-1,+1]$ yielded a median accuracy score of 96.2%. Above those, the $B[+1,-1]$-based Hv-CNN showcased a median accuracy score of 96.5%. On the top of the charts, the Hv-CNNs based on the isomorphic algebras $A[-1,+1]$, $A[+1,-1]$, and $A[+1,+1]$ as well as the Hv-CNN based on $B[+1,+1]$ showed an impressive median accuracy score of 96.9%. We highlight that there is a significant improvement in the accuracy scores of the Hv-CNN models when changing the image encoding, thus demonstrating that these models work well within the locally cohesive structure of the HSV encoding.

Performances attained in this experiment are summarized visually in the Hasse diagram shown in Fig. 10. We omit multiple algebras of the same isomorphism group to avoid cluttering the image, so each algebra present in the diagram represents all other algebras isomorphic to them. Indeed, only 4 algebras are needed to cover all isomorphism groups, so the chosen representatives are $A[-1,-1]$ (quaternions), $A[-1,+1]$ (coquaternions), $B[-1,+1]$ (tessarines), $B[+1,+1]$ (Klein 4-group), along with the real-

Table 8 – Mean accuracy (%) RGB- and HSV-encoded images of the test set per model. Bold numbers indicate the best performer in each test set.

Model	**RGB-Encoded**	**HSV-Encoded**
Real-valued	94.12 ± 3.78	91.89 ± 2.32
$A[-1,-1]$	93.72 ± 4.48	96.24 ± 2.24
$A[-1,+1]$	93.35 ± 4.12	96.68 ± 2.01
$A[+1,-1]$	94.58 ± 3.14	$\mathbf{96.85 \pm 1.89}$
$A[+1,+1]$	93.71 ± 4.45	96.61 ± 2.03
$B[-1,-1]$	94.12 ± 3.43	95.80 ± 2.80
$B[-1,+1]$	92.93 ± 3.84	96.39 ± 2.36
$B[+1,-1]$	$\mathbf{94.92 \pm 3.51}$	96.18 ± 2.37
$B[+1,+1]$	93.30 ± 4.27	96.22 ± 3.15

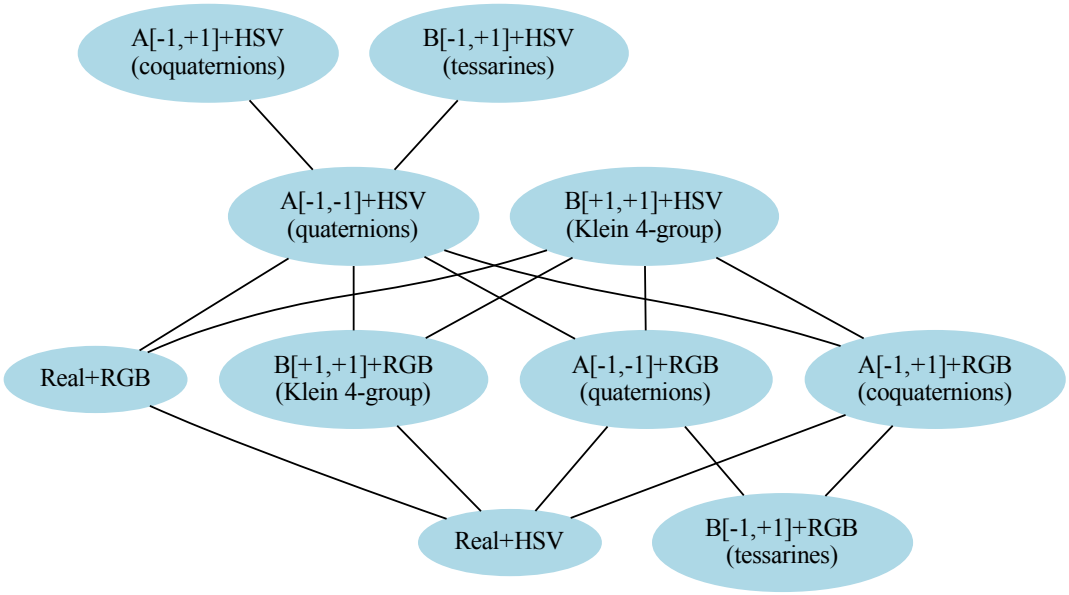

Figure 10 – Hasse diagram of representative experiment configurations.

valued models. Each of the representatives is included with regards to its performance in both RGB and HSV encodings of the image dataset. This diagram is intended as a hierarchical map of the models, in which two models connected by a solid line indicate that the model higher up dominates the one on the lower end of the line, i.e., showcases better performance with confidence level of 0.99 according to a Student's T test. Moreover, this hierarchy relation is transitive so models higher up on the diagram perform better than

all models below, not just immediate ones. The first thing to notice is how all Hv-CNN models on RGB encoded images are dominated by the Hv-CNN models on HSV encoded images, reinforcing our previous claim. Furthermore, all hypercomplex-valued models in HSV encoding outperform the real-valued models regardless of the encoding used on the latter. Indeed, the Rv-CNN on HSV encoded images shows a very poor performance and is outclassed by most models, avoiding being dominated only by the tessarine-valued model on RGB encoded images. Nonetheless, our main takeaway from this diagram is the fact that coquaternion- and tessarine-based Hv-CNNs show dominant performance over the quaternion-based CNN. This is surprising given that the latter is the most commonly used and widespread four-dimensional hypercomplex algebra. It is also important to recall that superior performance of neural networks based on non-usual hypercomplex algebras has been previously reported in the literature (VIEIRA; VALLE, 2020; VIEIRA; VALLE, 2022b). That is not to say that quaternions are underperformers in general, as was shown that a quaternion-based neural network yielded better results in applications like controlling a robot manipulator (TAKAHASHI, 2021). This is merely a piece of evidence that quaternions should not be taken as the automatic choice for any and all sort of task.

Finally, with regards to state-of-the-art comparison, Genovese et al. obtained a reported accuracy rate of 97.92% using a ResNet18 paired with histopathological transfer learning (GENOVESE et al., 2021b) in the same experimental settings of train-test split as above. Genovese's model builds upon the ResNet18 backbone by adding a dense layer after the convolutional layers; this dense layer is fine tuned while the convolutional layers are initialized by transfer learning (weights pre-trained on an histopathological dataset). Our top-performing Hv-CNNs achieved average accuracy scores of 96.51% and 96.39%. That being said, Genovese's ResNet18-based network contains approximately 11.4M parameters while the Hv-CNNs here described have only 36k parameters. In particular, the coquaternion-valued Hv-CNN with HSV encoding achieved 98% of the accuracy score of the ResNet18 with transfer learning while using only 0.3% of its total number of parameters, thus representing massive savings in terms of storage and training time. This will be further discussed in Section 4.6.

4.5.0.2 Experiment II: Clifford algebras

This experiment is part of a work entitled *"Clifford Convolutional Neural Networks for Lymphoblast Image Classification"* that has been presented at the *International Conference of Advanced Computational Applications of Geometric Algebra* (ICACGA 2022), held in October 2022.

This work features similar architectures to that of Experiment I. The algebras used here are only 3, namely the Clifford algebras $\mathcal{C}\ell_{2,0}$, $\mathcal{C}\ell_{1,1}$, $\mathcal{C}\ell_{0,2}$, with a real-valued model for comparison. The baseline Clifford CNN model ($\mathcal{C}\ell$CNN) used in this work is

Table 9 – Product of vectors and multivectors in four-dimensional Clifford algebras.

$\mathcal{C}\ell(2,0)$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	1	$\boldsymbol{k}$	$\boldsymbol{j}$
$\boldsymbol{j}$	$-\boldsymbol{k}$	1	$-\boldsymbol{i}$
$\boldsymbol{k}$	$-\boldsymbol{j}$	$\boldsymbol{i}$	-1

$\mathcal{C}\ell(0,2)$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	-1	$\boldsymbol{k}$	$-\boldsymbol{j}$
$\boldsymbol{j}$	$-\boldsymbol{k}$	-1	$\boldsymbol{i}$
$\boldsymbol{k}$	$\boldsymbol{j}$	$-\boldsymbol{i}$	-1

$\mathcal{C}\ell(1,1)$	$\boldsymbol{i}$	$\boldsymbol{j}$	$\boldsymbol{k}$
$\boldsymbol{i}$	1	$\boldsymbol{k}$	$\boldsymbol{j}$
$\boldsymbol{j}$	$-\boldsymbol{k}$	-1	$\boldsymbol{i}$
$\boldsymbol{k}$	$-\boldsymbol{j}$	$-\boldsymbol{i}$	1

composed by a convolutional layer with 4 filters, followed by two consecutive convolutional layers with 8 filters each and a convolutional layer with 16 filters. All layers use the split-rectified linear unit (split-ReLU) activation, i.e., the real ReLU applied separately to each channel of a Clifford number, and filters of size 3×3. Each of these layers is followed immediately by a max pooling layer with 2×2 kernels. The output of the final max pooling layer is then flattened and fed to a real-valued dense layer containing a single unit whose output is the label, 1 for lymphoblast, 0 otherwise. This defines a total of 3 $\mathcal{C}\ell$CNNs, one based on each algebra with multiplication table presented in Table 9.

For comparison, we propose a real-valued network with similar number of free trainable parameters and, hence, we shall refer to the $\mathcal{C}\ell$CNNs defined above as "**equivalent**". Since each hypercomplex-valued channel is roughly equivalent to four real-valued channels, we take the real-valued architecture with a larger number of filters per layer. Precisely, the real-valued CNN (Rv-CNN) is composed of the same four convolutional layers with 3×3 filters, each followed by a max pooling operator with 2×2 kernel. The number of filters per layer is 8, 16, 16 and 32, respectively, i.e., twice the number of filters in the corresponding equivalent hypercomplex-valued layer. The activation function used is the ReLU. The output of the fourth max pooling operation is then fed to a real-valued dense layer with a single neuron which outputs the calculated label. Lastly, to illustrate the vast learning capabilities of the $\mathcal{C}\ell$CNNs we take much smaller versions of the equivalent $\mathcal{C}\ell$CNNs and use these to perform the same task. These henceforth called "**small**" models use the same architecture of four convolutional layers followed by max pooling layers and a dense layer with a single neuron for labeling, yet each convolutional layer is taken with half the number of filters in the equivalent model. This leads to 3 small models with considerably less parameters than the real-valued and equivalent Clifford models. Thus, we end up with a total of 9 networks, namely, a real-valued model, 3 **equivalent** Clifford models with similar size to that of the real-valued model, and 3 **small** Clifford networks. All the architectures include a dropout layer before the dense layer with rate 0.5. This layer acts on a random behavior of setting inputs of the layer to the value 0 with a 0.5 rate. This layer helps avoiding overfitting the network to the training examples. Overfitting causes the network to learn noisy patterns present in the training set, thus making it

Table 10 – Sequential architecture outline and the number of trainable parameters.

		Rv-CNN	$\mathcal{C}\ell$CNN (equivalent)	$\mathcal{C}\ell$CNN (small)
Conv Layer 1	(3,3) filters	8	4	2
	Parameters	224	160	80
Max Pooling	2×2	-	-	-
Conv Layer 2	(3,3) filters	16	8	4
	Parameters	1,168	1,184	304
Max Pooling	2×2	-	-	-
Conv Layer 3	(3,3) filters	16	8	4
	Parameters	2,320	2,336	592
Max Pooling	2×2	-	-	-
Conv Layer 4	(3,3) filters	32	16	8
	Parameters	4,640	4,672	1,184
Max Pooling	2×2	-	-	-
Dense Layer	Neurons	1	1	1
	Parameters	1,153	2,305	1,153
Total		9,505	10,657	3,313

worse at generalizing. Table 10 outlines the architectures and shows a comparison of the total number of parameters. Despite the architectural similarity, the equivalent and small networks proposed in this paper have respectively 29% and 9% of the trainable parameters of the hypercomplex-valued CNNs considered by us in (VIEIRA; VALLE, 2022a). Again, all models were trained using the `'adam'` optimizer with the `BinaryCrossEntropy()` loss function from the `Keras.losses` library.

The dataset contains 260 images evenly divided between the two classes. We resize images to 126×126 upon loading. Next, we perform 100 experiments with each of the 7 networks, a total of 700 experiments. We adopted the same 50% train-test split used in (GENOVESE et al., 2021a) and performed vertical/horizontal flips to augment the training set. To showcase the proposed model's ability to learn on scarce datasets, we prioritized the use of compact (i.e. lower total number of parameters) networks, which help reduce the risk of overfitting on small training sets, a frequent issue with deep-learning applications in the medical field due to the inherent data scarcity. The proposed neural networks were implemented using `Tensorflow v2.9` and `Keras`. We trained for 300 epochs, using the Adam optimizer, with learning rate of 0.001, batch size of 32, and binary cross-entropy loss function. Performance is gauged using the accuracy in the test set [1].

Remark 6. *We tested the 9 networks on both RGB- and HSV-encoded images and the results reported here are the best for each network. Namely, the Rv-CNN uses RGB-encoded images while all $\mathcal{C}\ell$CNNs use the HSV-encoded images.*

[1] The complete code is available at https://github.com/mevalle/Hypercomplex-valued-Convolutional-Neural-Networks.

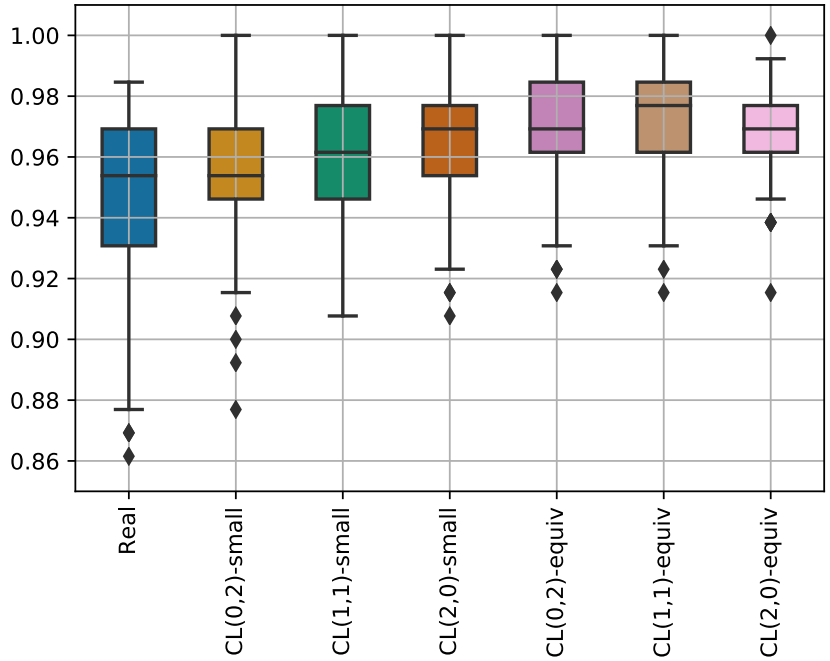

Figure 11 – Boxplot of test set accuracy performance by model.

Results for the 100 experiments of each model are depicted in Figure 11. The real-valued model shows a larger range and a wider interquartile range (IQR) when compared to the Clifford models. Furthermore, the maximum and minimum values attained by the Rv-CNN are lower than the maximum and minimum for the remaining models respectively. This clearly indicates that the Clifford models outperformed the Rv-CNN.

When comparing the 6 $\mathcal{C}\ell$CNNs we observe that the equivalent models show smaller IQRs and ranges than the small models. Also, the equivalent models achieve a superior mean accuracy in the test set than their respective small counterparts. Thus, the equivalent networks perform better and are statistically more reliably than the respective small models. Nonetheless, the accuracy showcased by the small models is impressive in the light of their reduced number of parameters. Precisely, Genovese et al. (GENOVESE et al., 2021b) attained average accuracy of 97.92% using the same dataset on a ResNet18 architecture combined with histopathological transfer learning, a network with approximately 11.4M free parameters. The top performing small $\mathcal{C}\ell$CNN attained 96.50% accuracy using only $3,313$ trainable parameters. This amounts to 98.55% of ResNet18's accuracy using roughly 0.0029% of the total parameters. Table 11 shows the detailed metrics for all models, including the accuracy achieved by the ResNet18 combined with histopathological transfer learning (GENOVESE et al., 2021b).

Finally, Figure 12 presents a Hasse diagram of the seven models used. This diagram represents a hypothesis test with 95% significance level. Models higher up in the hierarchy perform better than the ones to which they're linked and also better than the ones below those models. As expected, at the top are located the equivalent $\mathcal{C}\ell$CNN

Table 11 – Average accuracy (%) in train and test sets per model. Bold numbers are used to indicate the best performing small and equivalent models.

Model	**Train Set**	**Test Set**
Real-valued	99.71 ± 0.89	94.60 ± 2.76
$\mathcal{C}\ell(0,2)$-small	99.72 ± 1.45	95.55 ± 2.30
$\mathcal{C}\ell(1,1)$-small	99.81 ± 0.75	96.05 ± 1.88
$\mathcal{C}\ell(2,0)$-small	99.91 ± 0.35	$\mathbf{96.40 \pm 1.92}$
$\mathcal{C}\ell(0,2)$-equiv	99.97 ± 0.24	96.96 ± 1.69
$\mathcal{C}\ell(1,1)$-equiv	99.92 ± 0.52	$\mathbf{97.02 \pm 2.22}$
$\mathcal{C}\ell(2,0)$-equiv	99.96 ± 0.25	96.94 ± 1.54

State of the art model (GENOVESE et al., 2021b):

Model	**Test Set**
ResNet18	97.92 ± 1.62

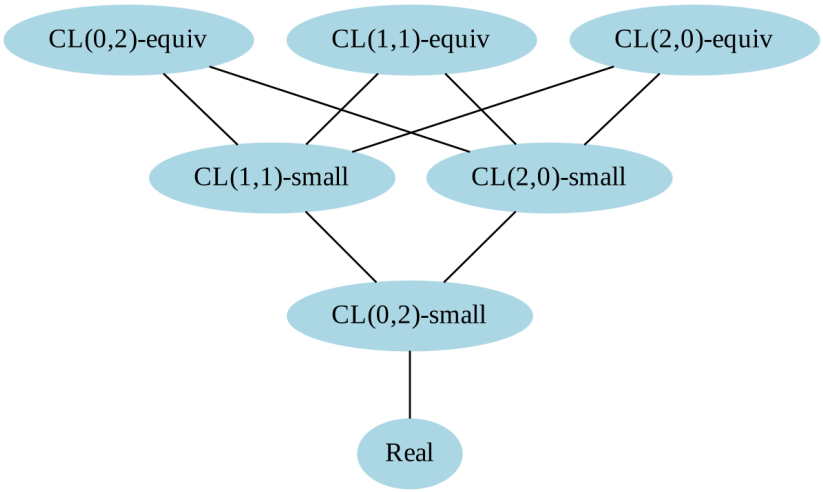

Figure 12 – Hasse diagram of the nine models present. A solid line linking two models indicates that the one on top performs better than the one below on a hypothesis test of 95% significance.

models. The Rv-CNN model is the poorest performer, being at the bottom of the diagram. The small $\mathcal{C}\ell$CNNs lie in the middle, with a special mention to the $\mathcal{C}\ell(0,2)$ (quaternion-valued) model showcasing the worst performance of the three. Notably, the small $\mathcal{C}\ell$CNNs outperform the Rv-CNN despite having significantly less parameters.

4.6 Concluding Remarks

Convolutional neural networks are among the most common deep learning (DL) models. As many other DL methods, CNNs usually boast a massive number of parameters.

Model	**TNP**	**Training Time**	**Accuracy (%)**
Real-valued CNN	$106k$	9s (100 epochs)	94.60 ± 2.76
$\mathbb{R}[+1, -1]$	$36.3k$	16s (100 epochs)	$\mathbf{96.85 \pm 1.89}$
$\mathbb{R}[-1, +1]$	$36.3k$	16s (100 epochs)	96.68 ± 2.21
Real-valued CNN **(in Experiment II)**	$9.5k$	30s (300 epochs)	94.60 ± 2.76
$\mathcal{C}\ell(2,0)$-small **(in Experiment II)**	$3.3k$	42s (300 epochs)	96.40 ± 1.92
$\mathcal{C}\ell(1,1)$-equiv **(in Experiment II)**	$10.6k$	56s (300 epochs)	$\mathbf{97.02 \pm 2.22}$
ResNet18 **(State-of-the-art)**	$11.4M$	Not reported *	$\mathbf{97.92 \pm 1.62}$

Table 12 – Comparison of attained results to the literature. The first three rows are taken from (VIEIRA; VALLE, 2022a). State-of-the-art model is taken from (GENOVESE et al., 2021b). Top performing model in each study has results in bold.
*Training time not reported. This work is based on transfer learning techniques and large CNNs.

In this chapter we showcased experiments carried out with compact CNNs. The real-valued models considered by us are small compared to the state-of-the-art model by Genovese *et al.* (GENOVESE et al., 2021a) and, on top of that, the Hv-CNNs further reduce the number of required parameters. This results in an outstandingly compact Hv-CNN model which is still capable of performing well in the classification task at hand.

As can be seen in Tab. 12, the attained results are not far from the state-of-the-art performance given in (GENOVESE et al., 2021a). We would like to highlight that the latter is a network based on a ResNet18 backbone and uses transfer learning, i.e., the weights are already pre-trained on a histopathological dataset. These thoroughly trained parameters work as the feature extractors of this state-of-the-art model, upon which Genovese *et al.* builds by attaching dense layers to perform the classification. In sum, it is a massive model that requires a hefty amount of storage space as well as potent processing hardware such as multiple modern GPUs to be executed. Our top performer, on the other hand, is an ultra light-weight model that requires negligible storage space and can be trained from scratch in less than a minute on a single GPU, yet it achieves 99.08% of Genovese's reported accuracy. Thus, our model is well adapted to being docked onto portable devices or embedded into a microscope, for example, retaining excellent performance.

Remark 7. *The GPU used in our work is the one provided by the Google Colaboratory platform. According to Google, the GPUs provided are of two types, NVidia K80 and T4, with 12 and 16GB of memory as well as 4.1 and 8.1 TFLOPS respectively.*

Regarding the comparisons both in (VIEIRA; VALLE, 2022a) and Experiment II, the hypercomplex-valued models outperform the respective equivalent real-valued model by a large margin. Not only the HvCNNs showcase higher accuracy while also work with a notably smaller number of parameters. This is due to each individual parameter

appearing multiple times in the convolution operation and the cross-channel information usage. Concerning training time, as opposed to the 87 times more that was seen in Section 3.5 for the ELM, the HvCNNs take only slightly more than the real-valued counterparts. This can be explained by the expressive reduction in number of parameters achieved by hypercomplex convolution. Overall, HvCNNs show a notably higher performance with only a fraction of the storage space requirement of the real-valued models, while requiring less than twice as much time to train.

5 Dual Quaternion Neural Network for Rigid Motion Modelling

In the previous chapters we discussed hypercomplex-valued models and frameworks in a general manner. Nonetheless, applications were restricted to four-dimensional algebras despite this generality. In this chapter we present a work using models in an eight-dimensional algebra, namely the dual quaternions $\mathbb{D}$, to predict 3D rigid motions. This was developed as part of the exchange program I participated in collaboration with the Sapienza University of Rome, Prof. Danilo Comminiello and Eleonora Grassucci.

5.1 Definition of Dual Quaternions

We recall that a quaternion given by $q = q_W + q_X\boldsymbol{i} + q_Y\boldsymbol{j} + q_Z\boldsymbol{k}$ efficiently describes a rotation in 3D space. Formally, q_W is called the real part and $\overline{\mathbf{q}} = q_X\boldsymbol{i}+q_Y\boldsymbol{j}+q_Z\boldsymbol{k}$ is referred to as the vector part of q. Hence, a quaternion can be written as $q = q_W + \overline{\mathbf{q}}$. We say that q is a pure quaternion if $q_W = 0$. The conjugate of $q = q_W + \overline{\mathbf{q}}$ is easily expressed in this notation as $q^* = q_W - \overline{\mathbf{q}}$. Moreover, we can write a quaternion in polar form

$$q_\theta = \|q\| \left(\cos\left(\frac{\theta}{2}\right) + \operatorname{sen}\left(\frac{\theta}{2}\right) \overline{\mathbf{u}} \right), \tag{5.1}$$

for $\theta \in [0, 2\pi)$ and a pure quaternion $\overline{\mathbf{u}} = u_X\boldsymbol{i}+u_Y\boldsymbol{j}+u_Z\boldsymbol{k}$, where $\|q\| = \sqrt{q_W^2 + q_X^2 + q_Y^2 + q_Z^2}$ is the absolute value of q. Finally, the rotation of a 3D vector (p_X, p_Y, p_Z) by an angle θ around the axis determined by (u_X, u_Y, u_Z) can be efficiently determined by

$$p_{rot} = qpq^*, \tag{5.2}$$

where $p = p_X\boldsymbol{i} + p_Y\boldsymbol{j} + p_Z\boldsymbol{k}$ and $u = u_X\boldsymbol{i} + u_Y\boldsymbol{j} + u_Z\boldsymbol{k}$ are pure quaternions encoding the position and axis vectors, and q is the quaternion given by (5.1). Moreover, if q is not a unitary quaternion p_{rot} will also be scaled by a factor of $\|q\|^2$. Therefore, it is common to normalize q simply as $q \leftarrow \dfrac{q}{\|q\|}$. Note that the representation in (5.1) is unique due to the domain of θ, and thus a quaternion uniquely defines a rotation-dilation and vice-versa. We highlight that this is equivalent to what was presented in (2.13) in Chapter 2.

Dual numbers are a hypercomplex algebra of dimension 2 over $\mathbb{R}$. A dual number has the form $\widehat{a} = a_0 + \varepsilon a_\varepsilon,\ a_0, a_\varepsilon \in \mathbb{R}$, where ε is called the dual unit and $\varepsilon^2 = 0$. Due to this, the product obeys $(a_0 + \varepsilon a_\varepsilon)(b_0 + \varepsilon b_\varepsilon) = a_0 b_0 + \varepsilon(a_0 b_\varepsilon + a_\varepsilon b_0)$, and thus the dual numbers are a degenerate algebra since the product of two non-zero elements may be null, e.g. $(\varepsilon a_\varepsilon)(\varepsilon b_\varepsilon) = 0$ with $a_\varepsilon, b_\varepsilon \neq 0$.

$\mathbb{D}$	1	$\mathbf{i}$	$\mathbf{j}$	$\mathbf{k}$	ε	$\varepsilon\mathbf{i}$	$\varepsilon\mathbf{j}$	$\varepsilon\mathbf{k}$
1	1	$\mathbf{i}$	$\mathbf{j}$	$\mathbf{k}$	ε	$\varepsilon\mathbf{i}$	$\varepsilon\mathbf{j}$	$\varepsilon\mathbf{k}$
$\mathbf{i}$	$\mathbf{i}$	-1	$\mathbf{k}$	$-\mathbf{j}$	$\varepsilon\mathbf{i}$	$-\varepsilon$	$\varepsilon\mathbf{k}$	$-\varepsilon\mathbf{j}$
$\mathbf{j}$	$\mathbf{j}$	$-\mathbf{k}$	-1	$\mathbf{i}$	$\varepsilon\mathbf{j}$	$-\varepsilon\mathbf{k}$	$-\varepsilon$	$\varepsilon\mathbf{i}$
$\mathbf{k}$	$\mathbf{k}$	$\mathbf{j}$	$-\mathbf{i}$	-1	$\varepsilon\mathbf{k}$	$\varepsilon\mathbf{j}$	$-\varepsilon\mathbf{i}$	$-\varepsilon$
ε	ε	$\varepsilon\mathbf{i}$	$\varepsilon\mathbf{j}$	$\varepsilon\mathbf{k}$	0	0	0	0
$\varepsilon\mathbf{i}$	$\varepsilon\mathbf{i}$	$-\varepsilon$	$\varepsilon\mathbf{k}$	$-\varepsilon\mathbf{j}$	0	0	0	0
$\varepsilon\mathbf{j}$	$\varepsilon\mathbf{j}$	$-\varepsilon\mathbf{k}$	$-\varepsilon$	$\varepsilon\mathbf{i}$	0	0	0	0
$\varepsilon\mathbf{k}$	$\varepsilon\mathbf{k}$	$\varepsilon\mathbf{j}$	$-\varepsilon\mathbf{i}$	$-\varepsilon$	0	0	0	0

Table 13 – Multiplication table of the dual quaternions.

The so called dual quaternions $\mathbb{D}$ can then be simply defined as quaternions in which each component is a dual number, $\hat{\mathbf{q}} = \hat{q}_W + \hat{q}_X\boldsymbol{i} + \hat{q}_Y\boldsymbol{j} + \hat{q}_Z\boldsymbol{k}$. In this case the dual unit obeys $\varepsilon^2 = 0$ and additionally ε commutes with the quaternion hyperimaginary units $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$. Thus this is an eight-dimensional algebra over $\mathbb{R}$. We note that $\mathbb{D}$ can be defined equivalently as dual numbers in which each part is a quaternion, i.e., $\hat{\mathbf{q}} = (q_0 + \varepsilon q_\varepsilon)$, $q_0, q_\varepsilon \in \mathbb{H}$. The multiplication table of this algebra is shown in Table 13.

We represent dual quaternions as bold-face letters with a hat $\hat{\mathbf{q}}$, while dual numbers will be regular letters with hat $\hat{q}$.

5.2 Translation-Equivariant Rigid Motion Representation by Dual Quaternions

It can be seen from the multiple null entries in the multiplication Tab. 13 that the dual quaternions are a degenerate algebra. Nonetheless, this algebra has interesting properties regarding modelling rigid motions, i.e., movements that do not incur in a deformation of the body. Any rigid motion in 3D space can be reduced to a rotation-translation with respect to a screw axis $\overrightarrow{h}$. Using a proper dual quaternion representation, we can characterize a full rigid motion as a unique entity and leverage algebra properties to build a more robust model.

Let us take a unit dual quaternion $\hat{\mathbf{q}} = \hat{q}_W + \hat{q}_X\boldsymbol{i} + \hat{q}_Y\boldsymbol{j} + \hat{q}_Z\boldsymbol{k}$. In a direct manner, the real part of $\hat{\mathbf{q}}$ is decomposed as $\widehat{\theta} = \arccos(\hat{q}_W) = \frac{\theta}{2} + \varepsilon\frac{s}{2}$ for $\theta \in [0, 2\pi)$, where $\frac{\theta}{2}$ is the rotation angle around the screw axis $\overrightarrow{h}$ and $\frac{s}{2}$ is the translation distance along that axis. The imaginary part of $\hat{\mathbf{q}}$, on the other hand, contains information regarding the direction of $\overrightarrow{h}$, which consists of a vector through the origin $\overrightarrow{u}$ translated by a vector $\overrightarrow{d}$ (PENNESTRÌ; VALENTINI, 2010). Formally, we have

$$\hat{\mathbf{q}} = \hat{q}_W + \hat{q}_X\boldsymbol{i} + \hat{q}_Y\boldsymbol{j} + \hat{q}_Z\boldsymbol{k}$$

where

$$\begin{aligned}
\widehat{q}_W &= \cos\frac{\theta}{2} - \frac{\varepsilon}{2}\overrightarrow{u}\cdot\overrightarrow{d}\,\mathrm{sen}\frac{\theta}{2} \\
\widehat{q}_X &= u_x\mathrm{sen}\frac{\theta}{2} + \frac{\varepsilon}{2}\left[d_x\cos\frac{\theta}{2} - \mathrm{sen}\frac{\theta}{2}(u_yd_z - u_zd_y)\right] \\
\widehat{q}_Y &= u_y\mathrm{sen}\frac{\theta}{2} + \frac{\varepsilon}{2}\left[d_y\cos\frac{\theta}{2} - \mathrm{sen}\frac{\theta}{2}(u_zd_x - u_xd_z)\right] \\
\widehat{q}_Z &= u_z\mathrm{sen}\frac{\theta}{2} + \frac{\varepsilon}{2}\left[d_z\cos\frac{\theta}{2} - \mathrm{sen}\frac{\theta}{2}(u_xd_y - u_yd_x)\right]
\end{aligned} \tag{5.3}$$

where $u = u_x\boldsymbol{i} + u_y\boldsymbol{j} + u_z\boldsymbol{k}$ and $d = d_x\boldsymbol{i} + d_y\boldsymbol{j} + d_z\boldsymbol{k}$ are purely imaginary quaternions and $\cdot$ on the first equation indicates the dot product, $\overrightarrow{u}\cdot\overrightarrow{d} = u_xd_x + u_yd_y + u_zd_z$.

Equation (5.3) highlights an interesting property: the dual part is responsible for the translation of the rigid motion. Indeed, if there is no translation the vector representing the translation of the screw axis is $\overrightarrow{d} = 0\boldsymbol{i} + 0\boldsymbol{j} + 0\boldsymbol{k}$. By setting $d_x = d_y = d_z = 0$ in the above equations we find that the dual part becomes null, and the resulting equation is equivalent to (5.1). This is a straightforward fact: a rigid motion with no translation is simply a rotation around an axis that contains the origin, a motion fully described by a quaternion.

On the other hand, if we take a rigid motion without rotation, i.e., $\theta = 0$ then (5.3) yields

$$\widehat{\mathbf{q}}_{\mathbf{t}} = 1 + \frac{\varepsilon}{2}\overrightarrow{d}\,. \tag{5.4}$$

This further reinforces that the dual part is responsible for the translation and shows that the imaginary part of the non-dual part is associated to the rotation operation.

This elegant formulation puts in evidence the role played by the dual part and formally shows why dual quaternions can encapsulate both operations while quaternions are restricted to representing rotations. Precisely, we note that the rotation q_θ and translation $\widehat{\mathbf{q}}_{\mathbf{t}}$ operations in (5.1) and (5.4) combine naturally into the full rigid motion by means of the Hamilton product

$$\widehat{\mathbf{q}} = \widehat{\mathbf{q}}_{\mathbf{t}}q_\theta. \tag{5.5}$$

Once again, if one is working with unit quaternions for rotations then $q_\theta^* = q_\theta^{-1}$ and the translation can be extracted from the rigid motion simply by calculating $\widehat{\mathbf{q}}_{\mathbf{t}} = \widehat{\mathbf{q}}q_\theta^*$. In other words, a unit quaternion rotation q_θ is enriched to a full rigid motion $\widehat{\mathbf{q}}$ simply by multiplication by a dual quaternion $\widehat{\mathbf{q}}_{\mathbf{t}}$ responsible for the translation.

Lastly, a unit dual quaternion $\widehat{\mathbf{q}}$ can be written as

$$\widehat{\mathbf{q}} = \cos\left(\frac{\widehat{\theta}}{2}\right) + \widehat{\mathbf{h}}\mathrm{sen}\left(\frac{\widehat{\theta}}{2}\right), \tag{5.6}$$

where $\widehat{\mathbf{h}}$ is a unit dual quaternion with zero scalar part. Note that this equation is similar to the quaternion polar form (5.1), except that $\widehat{\theta} = \theta_0 + \varepsilon\theta_\varepsilon$ is a dual number and

$\widehat{\mathbf{h}} = h_0 + \varepsilon h_\varepsilon$ is a dual quaternion. Indeed, this representation yields the screw motion parameters directly: $\theta_0/2$ is the rotation angle around the axis defined by h_0 and $\theta_\varepsilon/2$ is the translation along that same axis. h_ε is the so-called moment of the axis and provides an unambiguous representation of the axis in space. It is defined as $h_\varepsilon = \overrightarrow{p} \times h_0$, where $\overrightarrow{p}$ is a vector from the origin pointing to any point of the axis h_0. We note that any choice of vector yields the same moment, since $(\overrightarrow{p} + \alpha h_0) \times h_0 = \overrightarrow{p} \times h_0 = h_\varepsilon$, $\forall \alpha \in \mathbb{R}$. This illustrates one very desirable property: dual quaternions express rigid motions based on an unambiguous representation of the screw axis (KAVAN et al., 2006). Thus, dual quaternion rigid motions are independent of the coordinate system, hence being translation-equivariant, as opposed to quaternions whose rotations around axes containing the origin are deformed by translations.

5.3 Applications

The previous section ties dual quaternions to rigid motions. However, these concepts are presented in an abstract manner and evidenced formally but the translation of such aspects to practice is not straightforward. With the aim of clarifying this usage, the following subsections present two experiments featuring dual quaternion-valued models and 3D rigid motions: one exploring the translation equivariance in the Lorenz system and one showcasing an application to human pose forecasting on a real-world dataset.

The following sections compare the performance of a real-valued model, a quaternion-valued model and a dual-quaternion-valued model. As a disclaimer, the adaptation of the base real-valued models to hypercomplex values consists of a simple substitution of real-valued products and sums for the equivalent operations in the respective algebras, following Table 1 for the quaternions and Table 13 for the dual quaternions.

5.3.0.1 Experiment I: Lorenz System and Translational Equivariance

The Lorenz system is an ordinary differential equations (ODE) system that describes the movement of a free particle in atmospheric domain effects. This movement is characterized by rigid motions since the particle is under effects of translation at all times with its direction constantly rotating. The equations system is

$$\begin{cases} \dfrac{dx}{dt} = \sigma(y - x) \\ \dfrac{dy}{dt} = x(\rho - z) - y \\ \dfrac{dz}{dt} = xy - \beta z \end{cases} \tag{5.7}$$

where σ, β, ρ are constants. This system exhibits chaotic behavior for certain values of these constants such as $\sigma = 10$, $\beta = 8/3$, $\rho = 28$, meaning that a slight deviation in the

initial position results in a large deviation in the particle trajectory. Using these constant values we generate a time series of $10k$ consecutive positions, 10% of which are used for training, and the remaining 90% are used for testing. By strongly limiting the size of the training set we ensure that the performance of the network is more closely related to how well it learns the rigid motions rather than overfitting.

We employ single-hidden-layer MLPs with ReLU activation in the hidden layer and identity in the output layer in this example. The MLP neuron units simply perform a linear combination of inputs by the respective weights, followed by the activation function. The quaternion- and dual quaternion-valued MLP neurons perform the same operation, except the product follows the Hamilton rules and uses the commutativity of ε when needed; the activation function is applied entry-wise to each real component of the hypercomplex number. The networks were trained for a one-step-ahead prediction task using a 2-step sliding window, i.e., they received as inputs two consecutive positions and the desired output is the immediate third one. The inputs and outputs for the real-, quaternion-, and dual quaternion-valued models are formatted as follows:

- **Real-Valued Model:** 6 input values containing the 2 positions as $(x_{t-1}, y_{t-1}, z_{t-1}, x_t, y_t, z_t)$, 3 output values as $(x_{t+1}, y_{t+1}, z_{t+1})$.
- **Quaternion-Valued Model:** 2 quaternion inputs, one for each position, as $(0 + x_{t-1}\boldsymbol{i} + y_{t-1}\boldsymbol{j} + z_{t-1}\boldsymbol{k}, 0 + x_t\boldsymbol{i} + y_t\boldsymbol{j} + z_t\boldsymbol{k})$, 1 output quaternion as $(0 + x_{t+1}\boldsymbol{i} + y_{t+1}\boldsymbol{j} + z_{t+1}\boldsymbol{k})$.
- **Dual Quaternion-Valued Model:** 1 dual quaternion input containing the 2 positions as $(0 + x_{t-1}\boldsymbol{i} + y_{t-1}\boldsymbol{j} + z_{t-1}\boldsymbol{k}) + \varepsilon(0 + x_t\boldsymbol{i} + y_t\boldsymbol{j} + z_t\boldsymbol{k})$, one dual quaternion output $(0 + x_{t+1}\boldsymbol{i} + y_{t+1}\boldsymbol{j} + z_{t+1}\boldsymbol{k}) + \varepsilon(0 + 0\boldsymbol{i} + 0\boldsymbol{j} + 0\boldsymbol{k})$.

After training, the networks are tested twofold: first on the original test set and then on a translated test set, obtained by generating a (single) random translation vector and applying it to each point of the test set. Figure 13 shows these results. From visual inspection of the left column it is clear the dual quaternion model outputs a trajectory extremely close to the actual one (Fig. 13**(c)**), whereas real and quaternion models show clear flaws, especially on the lower left and upper right corners (Fig. 13 **(a)(b)**). Moreover, on the translated test set (on the second column) these two models slightly distort the shape and fail to translate the trajectory (Fig. 13 **(d)(e)**), while the dual quaternion model barely suffers any performance loss (Fig. 13 **(f)**). These visual observations are confirmed by the mean squared error (MSE) comparison present in Tab. 14. This showcases how dual quaternions are robust to translation transformations, and provides strong evidence that the model indeed learns the rigid motions of the system rather than a local behavior. The possibility to train a network on a dataset and achieve the same performance on

Table 14 – Mean squared error (MSE) for the Lorenz system prediction task with the original and the translated test set.

Model	**Number of parameters**	**Test MSE**	
		Original	**Translated**
Real-valued	1283	0.439	5.551
Quaternion-valued	1284	0.872	11.914
Dual Quaternion-valued	972	**0.119**	**0.183**

translated objects is invaluable and is a direct consequence of the translation equivariance of the underlying algebra.

5.3.0.2 Experiment II: Human Pose Forecasting

We carry out a validation inspection on human pose forecasting (HPF) using the 3D Poses in the Wild (3DPW) dataset. This experiment aims at predicting short-term immediate future poses for human bodies based on joint information. The 3DPW dataset contains over $51k$ registered frames from 60 short videos portraying humans performing basic activities such as hugging, arguing, playing basketball, and dancing, among others. Data is provided in 2 forms: RGB images and structured data. We focus on the latter, which comprises the 3D coordinates of the center of mass and of 13 key joints of individuals. This application is very modern and extremely important for self driving cars, elderly assistance systems, security surveillance, among others. However, it remains extremely daunting for most models, since it involves dealing with high dimensional data such as images and video as well as very noisy data due to the key points of the skeleton being mostly inferred.

We adapt the CoRPoF model in this experiment (PARSAEIFARD et al., 2021). The proposed Dual Quaternion CoRPoF comprises a single-layer DQLSTM encoder with a hidden dimension of 64, and two dual quaternion fully connected (DQFC) layers with latent dimension 32 to encode the mean and the variance statistics of the latent vector. The decoder is composed of two DQLSTM layers with the same hidden dimension with final DQFC layers to output the predicted pose. The model is trained for 1000 epochs by the Adam optimizer with a learning rate 0.01 and an adaptive scheduler, as suggested in (PARSAEIFARD et al., 2021). It is worth mentioning that due to the properties of the Hamilton product, the quaternion and dual quaternion model reduces the number of parameters of the network with respect to the real-valued baseline CoRPoF by 75% and 88%, respectively.

To assess the performance of our model and to be consistent with the previous literature, we compute the visibility ignored metric (VIM), which is the mean Euclidean distance between the predicted joints and the points in the ground truth, and the final displacement error (FDE), which is instead an L2 distance. Table 15 shows the objective

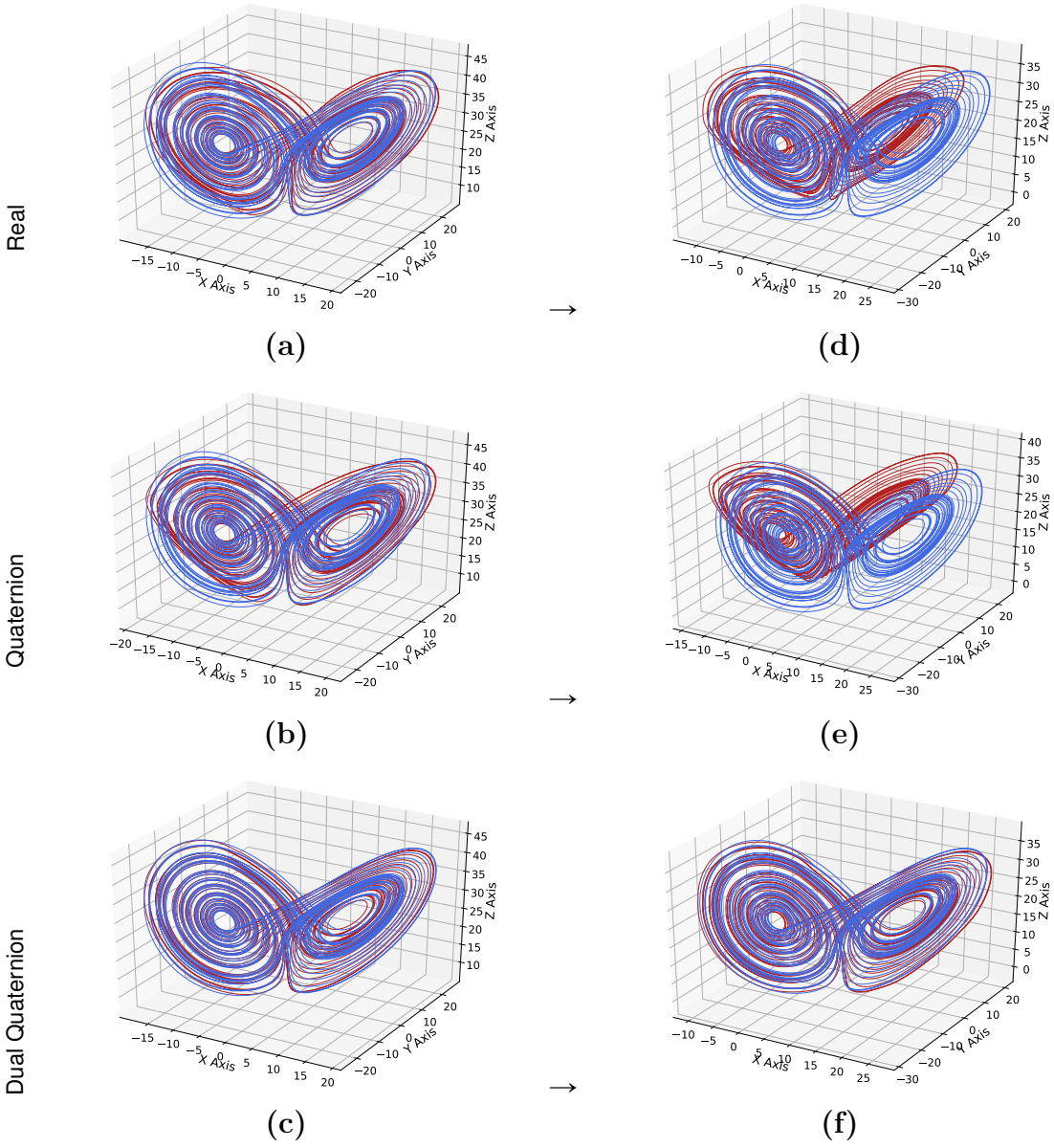

Figure 13 – Predicted trajectories in red, expected in blue. Rows contain real-, quaternion- and dual quaternion-valued outputs, respectively. First column shows the original test set output, while second column has the translated test set output. When translating the test set, the dual quaternion model barely shows any loss in performance, as opposed to the other two models failing to properly model the system.

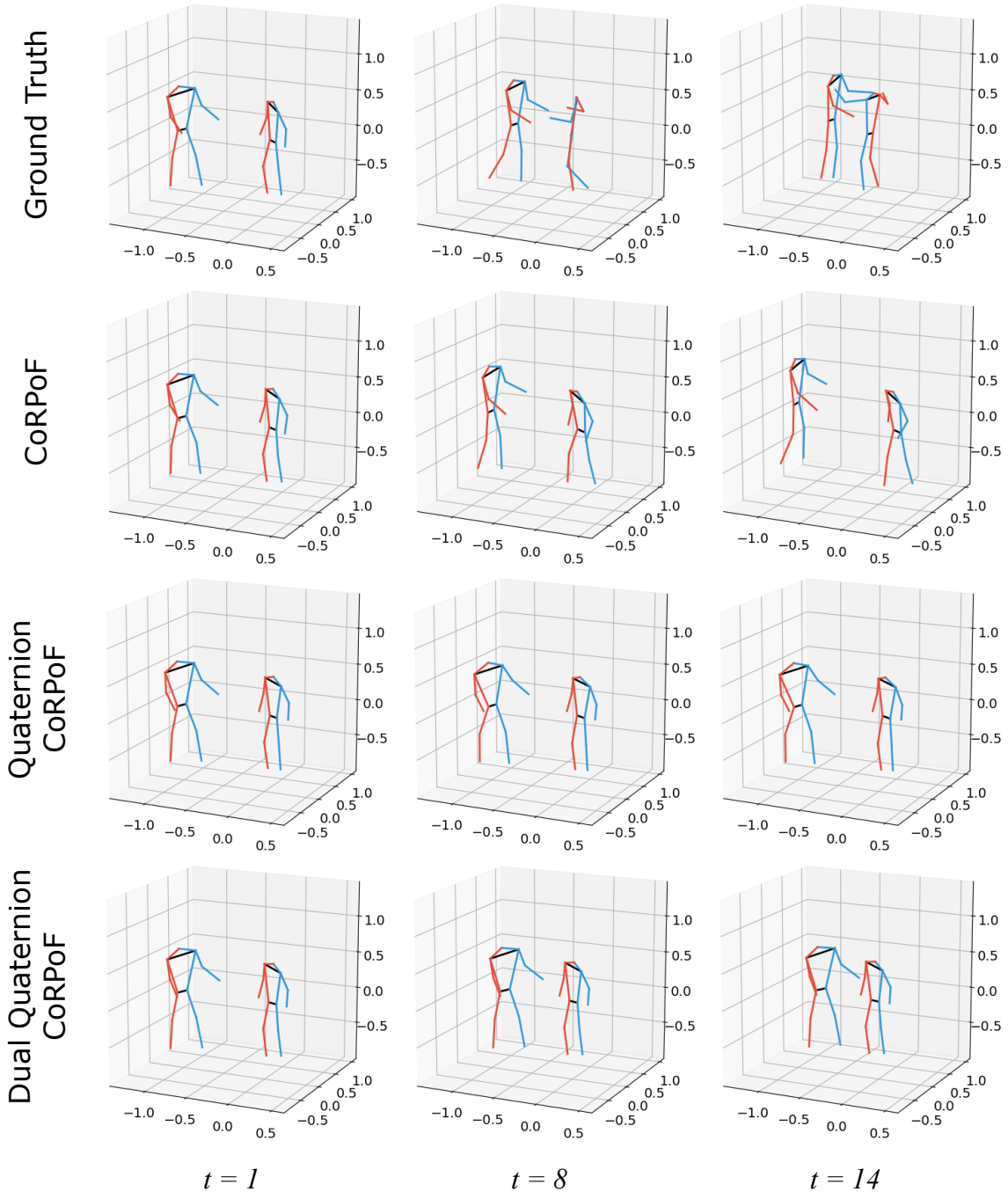

Figure 14 – Estimated poses at different time steps. The dual quaternion CoRPoF well models the trajectory of the skeletons, while the real- and quaternion-valued networks fail to properly learn such translations.

metrics evaluation and the improvement that our dual quaternion formulation brings for this task, proving the effectiveness of higher-dimensional representations for movements in the 3D space. We run the training multiple times and we report average scores and standard deviations over the runs. The comparisons with SC-MPF, Nearest Neighbours, and Zero velocity are performed with the scores reported in their papers, as their code is

Table 15 – Results for 3D human pose estimation on the 3DPW dataset. VIM scores are in centimeters, as suggested in (PARSAEIFARD et al., 2021). SC-MPF is taken from (ADELI et al., 2020), Nearest Neighbour is taken from (ZHANG et al., 2019), Zero Velocity is taken from (MARTINEZ; BLACK; ROMERO, 2017), and both DeRPoF and CoRPoF are taken from (PARSAEIFARD et al., 2021).

Model	**VIM↓**	**FDE↓**	**Val Loss↓**
SC-MPF	46.28	-	-
Nearest Neighbour	27.34	-	-
Zero Velocity	29.35	-	-
DeRPoF	19.07 ± .005	0.360 ± .007	-
CoRPoF	16.76 ± .003	0.317 ± .001	0.118 ± .004
Quaternion CoRPoF	16.35 ± .009	0.271 ± .002	0.105 ± .010
Dual Quaternion CoRPoF	**15.23** ± .002	**0.266** ± .001	**0.103** ± .006

not available. Additionally, Figure 14 displays a ground truth sample and the corresponding predicted skeletons by the Dual Quaternion CoRPoF and its quaternion- and real-valued counterparts. It is evident that, although CoRPoF in the real domain tries to model local poses, it completely fails to learn the right movement trajectory with the skeletons stuck in the same coordinates, as is the case of the quaternion model too. On the contrary, our dual quaternion approach well learns the correct trajectory of both skeletons, proving the ability of the proposed dual representation to model translations in 3D space.

5.4 Concluding Remarks

This work is particularly intended to showcase the application of dual quaternion's translation equivariance. This property is invaluable for most applications that can take advantage of it, as is the case of very modern applications such as Human Pose Forecasting. This challenging task has proved to be very daunting to even the most well-established models in the literature. Thus we also intend this work to open new paths of investigation in an effort to satisfactorily solve this problem.

As highlighted in Experiment I, the Lorenz system, the MLP based on dual quaternions virtually performs the same regardless of the test set used. In other words, the model was trained on a given data set but it captures the time series nature in an essential level, beyond the particular coordinate system in which the time series is described. In practice, this goes to show that, without retraining or any kind of specialist intervention, the dual quaternion model is not affected by translation of the origin as it instead learns the pattern regardlessly. The same cannot be said about the real- and quaternion-valued models. Even for a simple task as the Lorenz system one-step-ahead prediction these models are exceedingly thrown off by any relevant translation. Indeed, Figure 13 shows that the original test set, a simple continuation of the time series used in training, already proves a bit of a challenge to these models and the translated test set yields downright

bad results. This is confirmed by the egregious raise in the MSE when passing from one test set to another, as seen in Table 14.

In Experiment II, an encoder-decoder model of the literature (CoRPof (PARSAEI-FARD et al., 2021)) was modified in order to encapsulate quaternion and dual quaternion operations. As seen in Figure 14 the CoRPoF fails to completely model the movement of the skeletons in the immediate future, showcasing a basic limb movement understanding but entirely lacking the translation of the bodies. The Quaternion CoRPoF is not any better in this task, providing a mix of slight joint movement with small translation of the bodies, while not modelling notably well either of them. Lastly, the Dual Quaternion CoRPoF shows barely noticeably better results. The skeletons perform the translation almost to perfection but they are stiff, with little limb and joint movement. In a quantitative evaluation shown in Table 15 this model outperforms the real- and quaternion-valued CoRPoFs by a slim margin. This experiment represents but an initial step in the direction of applying dual quaternions to state-of-the-art models as means of improving translation modelling. As mentioned before, however, this problem is still very challenging, but our work intends to show that the translation equivariance of dual quaternions may prove invaluable for it, as well as the ease of adaptability of this algebra to modelling translations in general.

6 Conclusions

The core of this PhD study was to explore the usage of hypercomplex algebras in ANN models. Broadly speaking, this ranged from implementing entire frameworks for models in general hypercomplex algebras to simply replacing operations in state-of-the-art models by their respective hypercomplex counterparts. The main advantage of this stems from the ability to compactly represent many channels of information regarding the same object in a single entity, and processing these channels at once compactly. Therefore, all carried out applications and experiments contain data with multiple channels of information, such as images, videos, and positions in 3D space.

The first notable point of this work is the generalization of the concept of hypercomplex algebras in the context of ANNs. Indeed, a simple search on Scopus by combining the terms "neural network" and the name of notable hypercomplex algebras of the literature returns the values reported in Table 16. It is clear that the overwhelming majority of publications in this area focuses on complex- or quaternion-valued ANNs. This disregards an enormous family of algebras with huge potential, as explored in our works in the form of (VIEIRA; VALLE, 2020; VIEIRA; VALLE, 2022b). In fact, in all of the experiments showcased in this thesis, with exception of the one on Section 3.3, the quaternions were outperformed by at least one unusual hypercomplex algebra. Recognizing the potential of these alternatives and implementing general models are a main contribution of our previously mentioned works. By expressing an algebra through its multiplication table and including these tables as modular pieces of a model we are able to explore a broad family of algebras at once with ease.

Some recent works interpret the multiplication table with more freedom regard-

Table 16 – Number of publications on Scopus featuring the respective terms. Date: February 2022.

	Search Terms	**# of publications**
“neural network”	“complex number” OR “complex valued”	1.855
	“quaternion”	704
	“Clifford"	121
	“hypercomplex”	80
	“octonion”	22
	“hyperbolic number” OR “hyperbolic valued”	18
	“Bicomplex”	8
	“Cayley-Dickson”	5
	“tessarines”	1
	“coquaternion”	1
	“Klein four-group”	1

ing the values. In our approaches here described there are some unwritten rules that are consequence of fundamental assumptions that we make, such as for example the closure of an algebra. In all the algebras presented here the product of hyperimaginary units $\mu_{ij} = (\mu_{ij})_0 + (\mu_{ij})_1\boldsymbol{i} + (\mu_{ij})_2\boldsymbol{j} + (\mu_{ij})_3\boldsymbol{k}$ satisfies $(\mu_{ij})_m = \pm 1$ for some $m \in \{0, 1, 2, 3\}$ and $(\mu_{ij})_n = 0$ for $n \neq m$. In other words, one of the components of each element μ_{ij} is a signed unit while the rest is null, hence the product of two basis elements of the hypercomplex algebra always yields a third one with potential minus sign. In the recent work by Grassucci *et al.* (GRASSUCCI; ZHANG; COMMINIELLO, 2022) the algebras can be of arbitrary integer dimension and the product of units needs not follow this rule, thus $(\mu_{ij})_m \in \mathbb{R}$, $\forall m \in \{0, 1, 2, 3\}$. In that work the authors argue that the multiplication table can itself be trained. However, there are still important questions regarding that concept and any implications the training of the multiplication table has on the algebra.

Another core point of this PhD study is the development of the emulation techniques. Indeed, by constructing isomorphisms we formalized in detailed manner how to carry out multiplication and convolution operations in any hypercomplex algebra using real-valued linear algebra. This allows for the implementation of Hv-ANNs with ease using top tier libraries and tools. These advanced tools provide much faster processing and avoid the hassle of having to implement custom-made tools such as pooling layers and optimizers to the hypercomplex case. This was an important step to avoid spending too much time in implementing well-known functions, makes it easier to integrate our work with the majority of available models and allows us to better converse with researchers of the field.

A main aspect of hypercomplex-valued ANNs is data encoding. In essence, the main strength of this type of model lies in representing all information regarding the same object compactly. For example, a real-valued network processes a 3×3 window of RGB pixels of an image as 27 different values treated equally, disregarding the primal connection between the R, G and B channels of a same pixel among themselves. As in the experiments presented above, hypercomplex-valued ANNs process all channels of a same pixel as one entity. Moreover, the data itself can be encoded in a multitude of ways, as exposed in Chapter 4 with the RGB vs HSV encoded images. These objects both carry the same information but the encoding used to represent this information plays a key, yet not fully understood, role in the performance of models. Indeed, conversion of images from one color space to another affected models in different ways and many of the Hv-CNNs experienced significant raise in performance while the real-valued model suffered a notable loss. Color encoding spaces are numerous and yet are barely explored in ANN-related works; instead many of these works explore non-trivial contrivances that often result in marginal increase in performance. Furthermore, different data encodings are fairly easy to test and are ultimately similar to the effect of a hidden layer. Indeed, a hidden layer with the correct weights and activation function can operate the change from RGB to HSV

encodings, yet the increase seen due to the conversion of the images is far more expressive than the inclusion of a simple hidden layer before the model itself. Every type of data has multiple possible representations and exploring alternative candidates is a good practice.

One key question that underlies all the research done in this PhD program is: "How are the algebra properties or lack thereof connected to the performance of models based on it?". This is not a straightforward question with a simple answer. The expected answer is that an algebra with more desirable properties yields better results. This can be backed up by the experiment in Section 3.4 in which the MacFarlane's hyperbolic quaternions are the only hypercomplex algebra to be outperformed by the baseline real model. This algebra lacks two core properties: associativity and commutativity. Additionally, we have shown that degeneracy of the product of the algebra leads to absence of the universal approximation capability of MLPs (VITAL; VIEIRA; VALLE, 2022), or in other words, MLPs based on degenerate algebras lack the universal approximation property, initially proposed by Cybenko and a cornerstone of ensuing results in machine learning (CYBENKO, 1989). Nonetheless, we see on the same experiment of Section 3.4 that two Cayley-Dickson algebras ($\mathbb{R}[-1,+1]$ and $\mathbb{R}[+1,-1]$) lacking commutativity are top performers, notably above two commutative algebras ($\mathbb{T}$ and $\mathbb{K}_4$) and also above two other Cayley-Dickson algebras, which possess the same set of properties. The same can be seen in Experiment I of Chapter 4, where the top performer is again non-commutative. Lastly, Chapter 5 shows an experiment in which a degenerate algebra, namely the dual quaternions, outperforms both real- and quaternion-valued models by a decent margin. Moreover, there appears to be significant contribution of the translation equivariance property, which is not easily identifiable nor a standard mathematical property. This all shows that there is much depth to the question than simply expecting the most "robust" algebra with most desirable properties to yield better results.

On a final note, there is much to be said and weighted about the trade-offs of using hypercomplex-valued models in place of real-valued ones. As we strived to show during this program, there is a general trend of hypercomplex-valued models being able to "learn more" on fewer parameters and deliver better results in overall performance at the expense of complexity in the operations. Hypercomplex-valued pseudoinversion in the case of the ELMs and convolution in the case of CNNs are notably slower than operating the same computations in real-valued algebra. Even with the emulation techniques and the top tier libraries there is still a decent gap time-wise between these models and their real-valued baselines. Notwithstanding, this can be mitigated. For example, ELMs are known to be lightweight models with very little computational cost, and once the training is done the prediction step is extremely cheap. On the other hand, CNNs usually require a massive number of trainable parameters to extract desired features of the training data but the drastic reduction in total number of parameters operated by the Hv-CNNs can balance that out by learning the complex features in fewer parameters due to parameter reusing

and cross-channel information processing. At the end of the day, hypercomplex-valued models pose a trade-off involving performance, storage, and time-cost, and by clever choices of models, data encodings and architecture in general one can work this trade-off in their advantage, taking slightly more time on training in exchange for a final model with low storage requirements and similar-or-better performance, for example.

This PhD study played an important part in broadening my research horizons as well as strengthening my comprehension of concepts regarding multi-channel data, hypercomplex algebras, neural network models and more. A few collaborations were initiated, especially with the research group in La Sapienza University of Rome due to the exchange program that had me spend 2 months there. In addition, through my advisor Marcos Valle and the Mathematical Imaging Laboratory (*MiLab - UNICAMP*) there is great potential in collaborations with renown researchers in the field such as Danilo Mandic (Imperial College London, UK), Igor Aizenberg (Manhattan College, USA), Akira Hirose (University of Tokyo, Japan), Ramamurthy Garimella (Mahindra University, India), Danilo Comminiello (Sapienza University of Rome, Italy) as well as with organizations such as University of Warmia and Mazury in Olsztyn (Poland) and Universidad de Jaén (Spain). As this thesis strived to show, many avenues of research are clearer now and many promising more await thorough investigation. I look forward for the perspective of contributing with these investigations.

Bibliography

ADELI, V.; ADELI, E.; REID, I. D.; NIEBLES, J. C.; REZATOFIGHI, H. Socially and contextually aware human motion and pose forecasting. *IEEE Robotics and Automation Letters*, v. 5, p. 6033–6040, 2020. Citado na página 63.

AIZENBERG, I. N. *Complex-Valued Neural Networks with Multi-Valued Neurons*. [S.l.]: Springer, 2011. v. 353. (Studies in Computational Intelligence, v. 353). Citado na página 11.

ALBERT, A. A. Quadratic Forms Permitting Composition. *Annals of Mathematics*, Annals of Mathematics, v. 43, n. 1, p. 161–177, 1942. Citado na página 17.

ALFSMANN, D. On families of 2 N-dimensional hypercomplex algebras suitable for digital signal processing. In: IEEE. *2006 14th European Signal Processing Conference*. [S.l.], 2006. p. 1–4. Citado na página 19.

BROWN, R. B. On generalized Cayley-Dickson algebras. *Pacific Journal of Mathematics*, Pacific Journal of Mathematics, v. 20, n. 3, p. 415–422, 1967. ISSN 0030-8730. Disponível em: https://projecteuclid.org/euclid.pjm/1102992693. Citado na página 43.

BUCHHOLZ, S.; SOMMER, G. Hyperbolic Multilayer Perceptron. In: *Proceedings of the International Joint Conference on Neural Networks*. [S.l.: s.n.], 2000. v. 2, p. 129–133. Citado na página 11.

CASTRO, F. Z. de; VALLE, M. E. A broad class of discrete-time hypercomplex-valued Hopfield neural networks. *Neural Networks*, v. 122, p. 54–67, 2020. Citado na página 14.

CATONI, F.; CANNATA, R.; NICHELATTI, E.; ZAMPETTI, P. Commutative hypercomplex numbers and functions of hypercomplex variable: a matrix study. *Advances in Applied Clifford Algebras*, v. 15, n. 2, p. 183–212, 2005. Citado na página 15.

CHISHOLM, J. S. R.; FARWELL, R. S. Properties of Clifford algebras for fundamental particles. In: *Clifford (Geometric) Algebras*. [S.l.]: Springer, 1996. p. 365–388. Citado na página 19.

CRAVEN, D. A.; EATON, C. W.; KESSAR, R.; LINCKELMANN, M. The structure of blocks with a Klein four defect group. *Mathematische Zeitschrift*, Springer, v. 268, n. 1-2, p. 441–476, 2011. Citado na página 19.

CRUMEYROLLE, A. *Orthogonal and Symplectic Clifford Algebras: Spinor Structures*. [S.l.]: Springer Science & Business Media, 2013. v. 57. Citado na página 19.

CULBERT, C. Cayley-Dickson algebras and loops. *J. Gen. Lie Theory Appl*, v. 1, n. 1, p. 1–17, 2007. Citado na página 17.

CYBENKO, G. Approximation by superposition of sigmoidal functions. *Mathematics of Control, Signals and Systems*, v. 2, n. 4, p. 303–314, 1989. Citado na página 67.

DATAR, M.; GIONIS, A.; INDYK, P.; MOTWANI, R. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, Society for Industrial and Applied Mathematics, v. 31, n. 6, p. 1794–1813, 9 2002. ISSN 00975397. Citado na página 26.

DEMIR, S.; TANIŞLI, M.; CANDEMIR, N. Hyperbolic quaternion formulation of electromagnetism. *Advances in Applied Clifford Algebras*, Springer, v. 20, n. 3, p. 547–563, 2010. Citado na página 19.

FENG, X.; ZHANG, Z. The rank of a random matrix. *Applied mathematics and computation*, Elsevier, v. 185, n. 1, p. 689–694, 2007. Citado na página 25.

GARIMELLA, R. M.; VALLE, M. E.; VIEIRA, G.; RAYALA, A.; MUNUGOTI, D. Vector-valued hopfield neural networks and distributed synapse based convolutional and linear time-variant associative memories. *Neural Processing Letters*, Springer, p. 1–20, 2022. Citado na página 13.

GAUDET, C.; MAIDA, A. Deep Quaternion Networks. *arXiv*, arXiv, 12 2017. Disponível em: http://arxiv.org/abs/1712.04604. Citado 2 vezes nas páginas 37 and 38.

GENOVESE, A.; HOSSEINI, M. S.; PIURI, V.; PLATANIOTIS, K. N.; SCOTTI, F. Acute lymphoblastic leukemia detection based on adaptive unsharpening and deep learning. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, Institute of Electrical and Electronics Engineers Inc., v. 2021-June, p. 1205–1209, 2021. ISSN 15206149. Citado 2 vezes nas páginas 50 and 53.

_____. Histopathological transfer learning for acute lymphoblastic leukemia detection. *CIVEMSA 2021 - IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications, Proceedings*, Institute of Electrical and Electronics Engineers Inc., 6 2021. Citado 4 vezes nas páginas 48, 51, 52, and 53.

GÉRON, A. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. [S.l.]: O'Reilly Media, Incorporated, 2017. ISBN 9781491962299. Citado na página 30.

GOLUB, G. H. *Numerical methods for solving linear least squares problems*. 2007. Citado 3 vezes nas páginas 23, 24, and 25.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. Citado na página 36.

GRANERO, M. A.; HERNÁNDEZ, C. X.; VALLE, M. E. Quaternion-valued convolutional neural network applied for acute lymphoblastic leukemia diagnosis. In: BRITTO, A.; DELGADO, K. V. (Ed.). *Intelligent Systems*. Cham: Springer International Publishing, 2021. p. 280–293. Citado na página 45.

GRASSUCCI, E.; ZHANG, A.; COMMINIELLO, D. Phnns: Lightweight neural networks via parameterized hypercomplex convolutions. *IEEE Transactions on Neural Networks and Learning Systems*, IEEE, 2022. Citado na página 66.

HAYKIN, S.; LI, L. Nonlinear adaptive prediction of nonstationary signals. *IEEE Transactions on signal processing*, IEEE, v. 43, n. 2, p. 526–535, 1995. Citado na página 27.

HAYKIN, S. S.; HAYKIN, S. S.; HAYKIN, S. S.; HAYKIN, S. S. *Neural networks and learning machines*. [S.l.]: Pearson Upper Saddle River, NJ, USA:, 2009. v. 3. Citado na página 24.

HESTENES, D.; SOBCZYK, G. *Clifford algebra to geometric calculus: a unified language for mathematics and physics*. [S.l.]: Springer Science & Business Media, 2012. v. 5. Citado na página 19.

HIROSE, A. *Complex-Valued Neural Networks*. 2nd editio. ed. Heidelberg, Germany: Springer, 2012. (Studies in Computational Intelligence). Citado na página 11.

HUANG, G.-B.; ZHU, Q.-Y.; SIEW, C.-K.; others. Extreme learning machine: a new learning scheme of feedforward neural networks. *Neural networks*, v. 2, p. 985–990, 2004. Citado na página 23.

HUANG, J.-S.; YU, J. Klein four-subgroups of Lie algebra automorphisms. *Pacific Journal of Mathematics*, Mathematical Sciences Publishers, v. 262, n. 2, p. 397–420, 2013. Citado na página 19.

HUSMEIER, D.; HUSMEIER, D. Random vector functional link (rvfl) networks. *Neural Networks for Conditional Probability Estimation: Forecasting Beyond Point Predictions*, Springer, p. 87–97, 1999. Citado na página 23.

KANTOR, I. L.; SOLODOVNIKOV, A. S. *Hypercomplex Numbers: An Elementary Introduction to Algebras*. [S.l.]: Springer New York, 1989. Citado na página 14.

KAVAN, L.; COLLINS, S.; O'SULLIVAN, C.; ZARA, J. Dual quaternions for rigid transformation blending. *Trinity College Dublin, Tech. Rep.*, v. 46, 2006. Citado na página 58.

KOBAYASHI, M. Hopfield neural networks using Klein four-group. *Neurocomputing*, Elsevier B.V., v. 387, p. 123–128, 4 2020. ISSN 18728286. Citado na página 19.

LABATI, R. D.; PIURI, V.; SCOTTI, F. All-idb: The acute lymphoblastic leukemia image database for image processing. In: IEEE. *2011 18th IEEE international conference on image processing*. [S.l.], 2011. p. 2045–2048. Citado na página 41.

LABUNETS, V. Clifford algebras as unified language for image processing and pattern recognition. In: *Computational Noncommutative Algebra and Applications*. [S.l.]: Springer, 2004. p. 197–225. Citado na página 19.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2323, 1998. ISSN 00189219. Citado na página 41.

MARTINEZ, J.; BLACK, M. J.; ROMERO, J. On human motion prediction using recurrent neural networks. *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, p. 4674–4683, 2017. Citado na página 63.

MINEMOTO, T.; ISOKAWA, T.; NISHIMURA, H.; MATSUI, N. Feed forward neural network with random quaternionic neurons. *Signal Processing*, v. 136, p. 59–68, 2017. Citado 2 vezes nas páginas 29 and 30.

NITTA, T.; BUCHHOLZ, S. On the decision boundaries of hyperbolic neurons. In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. [S.l.: s.n.], 2008. p. 2974–2980. Citado na página 11.

NITTA, T.; KUROE, Y. Hyperbolic gradient operator and hyperbolic back-propagation learning algorithms. *IEEE Transactions on Neural Networks and Learning Systems*, v. 29, n. 5, p. 1689–1702, 2018. Citado na página 11.

PARCOLLET, T.; MORCHID, M.; LINARÈS, G. Quaternion Convolutional Neural Networks for Heterogeneous Image Processing. In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.: s.n.], 2019. p. 8514–8518. Citado na página 11.

PARSAEIFARD, B.; SAADATNEJAD, S.; LIU, Y.; MORDAN, T.; ALAHI, A. Learning decoupled representations for human pose forecasting. *IEEE/CVF Int. Conf. on Computer Vision Workshops (ICCVW)*, p. 2294–2303, 2021. Citado 3 vezes nas páginas 60, 63, and 64.

PEI, S.-C.; CHANG, J.-H.; DING, J.-J. Commutative reduced biquaternions and their Fourier transform for signal and image processing applications. *IEEE Transactions on Signal Processing*, IEEE, v. 52, n. 7, p. 2012–2031, 2004. Citado na página 19.

PENNESTRÌ, E.; VALENTINI, P. P. Dual quaternions as a tool for rigid body motion analysis: A tutorial with an application to biomechanics. *Archive of Mechanical Engineering*, v. 57, n. 2, p. 187–205, 2010. Citado na página 56.

PORTEOUS, I. R.; others. *Clifford algebras and the classical groups*. [S.l.]: Cambridge University Press, 1995. v. 50. Citado na página 19.

ROCHON, D.; SHAPIRO, M. On algebraic properties of bicomplex and hyperbolic numbers. *Anal. Univ. Oradea, fasc. math*, v. 11, n. 71, p. 110, 2004. Citado na página 19.

SCHAFER, R. D. On the Algebras Formed by the Cayley-Dickson Process. *American Journal of Mathematics*, Johns Hopkins University Press, v. 76, n. 2, p. 435–446, 1954. Citado na página 17.

TAKAHASHI, K. Comparison of high-dimensional neural networks using hypercomplex numbers in a robot manipulator control. *Artificial Life and Robotics*, Springer, v. 26, n. 3, p. 367–377, 2021. Citado 2 vezes nas páginas 16 and 48.

TANG, J.; DENG, C.; HUANG, G. Extreme Learning Machine for Multilayer Perceptron. *IEEE Transactions on Neural Networks and Learning Systems*, v. 27, n. 4, p. 809–821, 2016. Citado na página 29.

TRABELSI, C.; BILANIUK, O.; ZHANG, Y.; SERDYUK, D.; SUBRAMANIAN, S.; SANTOS, J. F.; MEHRI, S.; ROSTAMZADEH, N.; BENGIO, Y.; PAL, C. J. *Deep complex networks*. [S.l.]: arXiv, 2017. Citado 2 vezes nas páginas 37 and 38.

TREFETHEN, L. N.; III, D. B. *Numerical Linear Algebra*. Philadelphia, PA: SIAM Publications, 1997. Citado 3 vezes nas páginas 23, 24, and 25.

VIEIRA, G.; VALLE, M. E. Extreme Learning Machines on Cayley-Dickson Algebra Applied for Color Image Auto-Encoding. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020. p. 1–8. ISBN 978-1-7281-6926-2. Disponível em: https://ieeexplore.ieee.org/document/9207495/. Citado 5 vezes nas páginas 12, 30, 43, 48, and 65.

VIEIRA, G.; VALLE, M. E. Acute lymphoblastic leukemia detection using hypercomplex-valued convolutional neural networks. In: IEEE. *2022 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2022. p. 1–8. Citado 4 vezes nas páginas 12, 42, 50, and 53.

_____. A general framework for hypercomplex-valued extreme learning machines. *Journal of Computational Mathematics and Data Science*, Elsevier, v. 3, p. 100032, 2022. Citado 3 vezes nas páginas 12, 48, and 65.

VITAL, W. L.; VIEIRA, G.; VALLE, M. E. Extending the universal approximation theorem for a broad class of hypercomplex-valued neural networks. In: SPRINGER. *Intelligent Systems: 11th Brazilian Conference, BRACIS 2022, Campinas, Brazil, November 28–December 1, 2022, Proceedings, Part II*. [S.l.], 2022. p. 646–660. Citado na página 67.

XIA, Y.; XIANG, M.; LI, Z.; MANDIC, D. P. Echo State Networks for Multidimensional Data: Exploiting Noncircularity and Widely Linear Models. In: COMMINIELLO, D.; PRÍNCIPE, J. C. (Ed.). *Adaptive Learning Methods for Nonlinear System Modeling*. [S.l.]: Butterworth-Heinemann, 2018. p. 267–288. ISBN 978-0-12-812976-0. Citado na página 11.

XU, D.; XIA, Y.; MANDIC, D. P. Optimization in Quaternion Dynamic Systems: Gradient, Hessian, and Learning Algorithms. *IEEE Transactions on Neural Networks and Learning Systems*, v. 27, n. 2, p. 249–261, 2016. Citado na página 26.

ZHANG, H.; XUE, J.; DANA, K. Deep ten: Texture encoding network. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 708–717. Citado na página 29.

ZHANG, J. Y.; FELSEN, P.; KANAZAWA, A.; MALIK, J. Predicting 3d human dynamics from video. *IEEE/CVF Int. Conf. on Computer Vision (ICCV)*, p. 7113–7122, 2019. Citado na página 63.