Tessarine and Quaternion-Valued Deep Neural Networks for Image Classification

Fernando Ribeiro de Senna[1], **Marcos Eduardo Valle**[1]

[1]Institute of Mathematics, Statistics, and Scientific Computing
University of Campinas (UNICAMP), Campinas, Brazil

fernandordsenna@gmail.com, valle@ime.unicamp.br

Abstract. *Many image processing and analysis tasks are performed with deep neural networks. Although the vast majority of advances have been made with real numbers, recent works have shown that complex and hypercomplex-valued networks may achieve better results. In this paper, we address quaternion-valued and introduce tessarine-valued deep neural networks, including tessarine-valued 2D convolutions. We also address initialization schemes and hypercomplex batch normalization. Finally, a tessarine-valued ResNet model with hypercomplex batch normalization outperformed the corresponding real and quaternion-valued networks on the CIFAR dataset.*

1. Introduction

Image classification is an important area with many practical applications in which machine learning and deep learning play fundamental roles through particular types of artificial neural networks (ANNs) such as the convolutional neural networks (CNNs) and other deep networks. The CNN proposed by Lecun et al. was one of the pioneers and gave visibility to deep neural architectures [Lecun et al. 1998]. Before the advent of CNNs and other deep neural networks, one of the most common artificial network architectures was the Multilayer Perceptron (MLP) [Haykin 2009]. However, MLP is not well suited for image processing tasks. Images consist of many pixels, so the MLP would need too many parameters. Moreover, MLP is not robust to translations and slight modifications, which are common in images. Also, the structure of images is not considered because they are usually flattened to a vector. CNN models circumvent many of the MLP limitations for image processing tasks by taking advantage of convolutions, which reduce the number of parameters and preserve the input structure [Lecun et al. 1998].

Recent developments are mainly made with real ANNs. Nonetheless, there is evidence that using other algebras such as complex numbers [Aizenberg 2011, Trabelsi et al. 2017] and quaternions [Arena et al. 1997, Gaudet and Maida 2017, Zhu et al. 2019, Kumar and Tripathi 2019, Parcollet et al. 2020, Grassucci et al. 2021] may improve networks performance without increasing (and sometimes reducing) the number of parameters. Reducing the number of parameters is particularly advantageous for applications with memory and other constraints. Despite reducing the number of parameters, using hypercomplex algebras often better treats multi-dimensional information data such as color images.

Tessarines and quaternions are two different four-dimensional algebras. Quaternions have been conceived by Hamilton in 1844 [Hamilton 1844] while the tessarines have been proposed by Cockle in 1848 [Cockle 1848]. Quaternions have been successfully used in deep neural networks models achieving the state of the art performances [Gaudet and Maida 2017, Zhu et al. 2019, Grassucci et al. 2021]. Although there are examples of tessarines usage for signal processing [Navarro-Moreno et al. 2020, Navarro-Moreno and Ruiz-Molina 2021], to the best of our knowledge, no tessarine-valued neural network has been proposed yet. Both quaternions and tessarines are well suited for computer vision applications because a single hypercomplex number can encode the color pixel information. In contrast, real-valued networks treat each channel independently.

In this paper, we address tessarine and quaternion-valued deep neural networks. Precisely, we present 2D tessarine-valued convolutions. We also address hypercomplex weight initialization and batch normalization. Moreover, we discuss activation functions, residual learning, and average pooling for tessarine and quaternion-valued networks. Finally, we compare the performance of real, tessarine, and quaternion-valued ResNet models [He et al. 2015a, He et al. 2016] for image classification tasks using the CIFAR10 database [Krizhevsky 2009].

The paper is structured as follows. Section 2 presents the basic concepts on tessarines and quaternions. Section 3 generalizes some of the main building blocks of deep neural networks for tessarines and quaternions. Computational experiments for an image classification task are detailed in Section 4. In Section 5, we offer concluding remarks and possibilities of future work.

2. Basic Concepts on Tessarines and Quaternions

A tessarine t is represented by $t = t_0 + t_1\mathbf{i} + t_2\mathbf{j} + t_3\mathbf{k}$ where t_0, t_1, t_2, $t_3 \in \mathbb{R}$ and **i**, **j**, and **k** are the hyper-imaginary units. The set of tessarine numbers is denoted by $\mathbb{T}$. To represent each of the tessarine components, we will use the representation $Re(t) = t_0$, $\mathcal{I}(t) = t_1$, $\mathcal{J}(t) = t_2$, and $\mathcal{K}(t) = t_3$. We say that t is a pure tessarine if $Re(t) = 0$.

Similarly, a quaternion q is represented by $q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$ where q_0, q_1, q_2, $q_3 \in \mathbb{R}$. The set of all quaternions is denoted by $\mathbb{Q}$. Moreover, the quaternions components are $Re(q) = q_0$, $\mathcal{I}(q) = q_1$, $\mathcal{J}(q) = q_2$, and $\mathcal{K}(q) = q_3$. A quaternion q is called a pure quaternion if $Re(q) = 0$. Despite the resemblance of quaternions and tessarines, their hyper-imaginary units are not the same and have different properties.

Given two tessarines, $s = s_0 + s_1\mathbf{i} + s_2\mathbf{j} + s_3\mathbf{k}$ and $t = t_0 + t_1\mathbf{i} + t_2\mathbf{j} + t_3\mathbf{k}$, and two quaternions, $p = p_0 + p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k}$ and $q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$, their sums are defined, respectively, as

$$s + t = (s_0 + t_0) + (s_1 + t_1)\mathbf{i} + (s_2 + t_2)\mathbf{j} + (s_3 + t_3)\mathbf{k}. \tag{1}$$

$$p + q = (p_0 + q_0) + (p_1 + q_1)\mathbf{i} + (p_2 + q_2)\mathbf{j} + (p_3 + q_3)\mathbf{k}. \tag{2}$$

A hypercomplex product is defined using a multiplication table. A multiplication table indicates the product of every two hyper-imaginary units. The tessarine and quaternion multiplication tables are given by Tables 1 and 2, respectively [Cerroni 2017].

Given a multiplication table, the product of two hypercomplex numbers is defined using the distributivity law. Precisely, the product of two tessarines $s = s_0 + s_1\mathbf{i} + s_2\mathbf{j} + s_3\mathbf{k}$

Table 1. Tessarine multiplication table.

$\times$	$\mathbf{i}$	$\mathbf{j}$	$\mathbf{k}$
$\mathbf{i}$	-1	$\mathbf{k}$	$-\mathbf{j}$
$\mathbf{j}$	$\mathbf{k}$	1	$\mathbf{i}$
$\mathbf{k}$	$-\mathbf{j}$	$\mathbf{i}$	-1

Table 2. Quaternion multiplication table.

$\times$	$\mathbf{i}$	$\mathbf{j}$	$\mathbf{k}$
$\mathbf{i}$	-1	$\mathbf{k}$	$-\mathbf{j}$
$\mathbf{j}$	$-\mathbf{k}$	-1	$\mathbf{i}$
$\mathbf{k}$	$\mathbf{j}$	$-\mathbf{i}$	-1

and $t = t_0 + t_1\mathbf{i} + t_2\mathbf{j} + t_3\mathbf{k}$ is defined by

$$\begin{aligned} st &= (s_0 + s_1\mathbf{i} + s_2\mathbf{j} + s_3\mathbf{k})(t_0 + t_1\mathbf{i} + t_2\mathbf{j} + t_3\mathbf{k}) \\ &= (s_0t_0 - s_1t_1 + s_2t_2 - s_3t_3) + (s_0t_1 + s_1t_0 + s_2t_3 + s_3t_2)\mathbf{i} \\ &+ (s_0t_2 - s_1t_3 + s_2t_0 - s_3t_1)\mathbf{j} + (s_0t_3 + s_1t_2 + s_2t_1 + s_3t_0)\mathbf{k}. \end{aligned} \tag{3}$$

Analogously, the Hamilton product of two quaternions $p = p_0 + p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k}$ and $q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$ is defined by

$$\begin{aligned} pq &= (p_0 + p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k})(q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}) \\ &= (p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3) + (p_0q_1 + p_1q_0 + p_2q_3 - p_3q_2)\mathbf{i} \\ &+ (p_0q_2 - p_1q_3 + p_2q_0 + p_3q_1)\mathbf{j} + (p_0q_3 + p_1q_2 - p_2q_1 + p_3q_0)\mathbf{k}. \end{aligned} \tag{4}$$

On the one hand, the quaternion product is noncommutative; that is, there are $p, q \in \mathbb{Q}$ for which $pq \neq qp$. In contrast, the tessarine product is commutative. On the other hand, tessarine numbers have non-zero divisors; that is, there are non-zero tessarines $s, t \in \mathbb{T}$ such that $st = 0$.

3. Tessarine and Quaternion-Valued Deep Neural Networks

Deep neural networks depend on many operations, such as convolution, initialization, batch normalization, and pooling. Analogous operations must be defined for designing valuable hypercomplex-valued deep neural networks. In this section, we first review real-valued operations to address their tessarine and quaternion-valued modifications subsequently.

3.1. Convolution

Given a real-valued image $\mathbf{I}$ and a filter (or kernel) $\mathbf{F}$ of size $(2N + 1) \times (2N + 1)$, the 2D convolution widely used in deep neural networks is defined by

$$(\mathbf{I} * \mathbf{F})^{(x,y)} = \sum_{i=-N}^{N} \sum_{j=-N}^{N} I^{(x+i,y+j)} F^{(i,j)}. \tag{5}$$

In order to define tessarine convolution, it is useful to represent tessarines as vectors. Hence, we identify a tessarine $t = t_0 + t_1\mathbf{i} + t_2\mathbf{j} + t_3\mathbf{k}$ with a four-dimensional vector (t_0, t_1, t_2, t_3). Accordingly, we may represent a tessarine-valued image $\mathbf{I}$ as $\mathbf{I} = (\mathbf{I_0}, \mathbf{I_1}, \mathbf{I_2}, \mathbf{I_3})$.

Given a tessarine-valued image $\mathbf{I} = (\mathbf{I_0}, \mathbf{I_1}, \mathbf{I_2}, \mathbf{I_3})$ and a tessarine-valued filter $\mathbf{F} = (\mathbf{F_0}, \mathbf{F_1}, \mathbf{F_2}, \mathbf{F_3})$, the tessarine-valued convolution is defined as

$$(\mathbf{I} * \mathbf{F})^{(x,y)} = \sum_{i=-N}^{N} \sum_{j=-N}^{N} I^{(x+i,y+j)} F^{(i,j)}. \tag{6}$$

As (6) depends on the product between the tessarine quantities $I^{(x+i,y+j)}$ and $F^{(i,j)}$, from (3), the tessarine-valued convolution $(\mathbf{I} * \mathbf{F})^{(x,y)}$ is equal to

$$\sum_{i=-N}^{N} \sum_{j=-N}^{N} \begin{bmatrix} I_0^{(x+i,y+j)} F_0^{(i,j)} - I_1^{(x+i,y+j)} F_1^{(i,j)} + I_2^{(x+i,y+j)} F_2^{(i,j)} - I_3^{(x+i,y+j)} F_3^{(i,j)} \\ I_0^{(x+i,y+j)} F_1^{(i,j)} + I_1^{(x+i,y+j)} F_0^{(i,j)} + I_2^{(x+i,y+j)} F_3^{(i,j)} + I_3^{(x+i,y+j)} F_2^{(i,j)} \\ I_0^{(x+i,y+j)} F_2^{(i,j)} - I_1^{(x+i,y+j)} F_3^{(i,j)} + I_2^{(x+i,y+j)} F_0^{(i,j)} - I_3^{(x+i,y+j)} F_1^{(i,j)} \\ I_0^{(x+i,y+j)} F_3^{(i,j)} + I_1^{(x+i,y+j)} F_2^{(i,j)} + I_2^{(x+i,y+j)} F_1^{(i,j)} + I_3^{(x+i,y+j)} F_0^{(i,j)} \end{bmatrix}^t . \tag{7}$$

Since tessarine addition operates independently among its components, (7) can be computed as follows, using real-valued convolutions:

$$\mathbf{I} * \mathbf{F} = \begin{bmatrix} (\mathbf{I} * \mathbf{F})_0 \\ (\mathbf{I} * \mathbf{F})_1 \\ (\mathbf{I} * \mathbf{F})_2 \\ (\mathbf{I} * \mathbf{F})_3 \end{bmatrix}^t = \begin{bmatrix} \mathbf{I}_0 & \mathbf{I}_1 & \mathbf{I}_2 & \mathbf{I}_3 \end{bmatrix} * \begin{bmatrix} \mathbf{F}_0 & \mathbf{F}_1 & \mathbf{F}_2 & \mathbf{F}_3 \\ -\mathbf{F}_1 & \mathbf{F}_0 & -\mathbf{F}_3 & \mathbf{F}_2 \\ \mathbf{F}_2 & \mathbf{F}_3 & \mathbf{F}_0 & \mathbf{F}_1 \\ -\mathbf{F}_3 & \mathbf{F}_2 & -\mathbf{F}_1 & \mathbf{F}_0 \end{bmatrix}. \tag{8}$$

It is important to notice that $\mathbf{I}$ and $\mathbf{F}$ are tessarine-valued image and filter, respectively. Hence, $\mathbf{I} * \mathbf{F}$ is a tessarine convolution. Nonetheless, $\mathbf{I}_0,\ \mathbf{I}_1,\ \mathbf{I}_2$ and $\mathbf{I}_3$ are real images and $\mathbf{F}_0,\ \mathbf{F}_1,\ \mathbf{F}_2$ and $\mathbf{F}_3$ are real filters, so $\mathbf{I}_a * \mathbf{F}_b$, with $a,\ b \in \{0, 1, 2, 3\}$, is a real-valued convolution.

The quaternion-valued convolution is defined analogously. Given a quaternion-valued image $\mathbf{I} = (\mathbf{I_0}, \mathbf{I_1}, \mathbf{I_2}, \mathbf{I_3})$ and a quaternion-valued filter $\mathbf{F} = (\mathbf{F_0}, \mathbf{F_1}, \mathbf{F_2}, \mathbf{F_3})$, their convolution is defined (similarly to the one proposed in [Gaudet and Maida 2017]) by

$$\mathbf{I} * \mathbf{F} = \begin{bmatrix} (\mathbf{I} * \mathbf{F})_0 \\ (\mathbf{I} * \mathbf{F})_1 \\ (\mathbf{I} * \mathbf{F})_2 \\ (\mathbf{I} * \mathbf{F})_3 \end{bmatrix}^t = \begin{bmatrix} \mathbf{I}_0 & \mathbf{I}_1 & \mathbf{I}_2 & \mathbf{I}_3 \end{bmatrix} * \begin{bmatrix} \mathbf{F}_0 & \mathbf{F}_1 & \mathbf{F}_2 & \mathbf{F}_3 \\ -\mathbf{F}_1 & \mathbf{F}_0 & -\mathbf{F}_3 & \mathbf{F}_2 \\ -\mathbf{F}_2 & \mathbf{F}_3 & \mathbf{F}_0 & -\mathbf{F}_1 \\ -\mathbf{F}_3 & -\mathbf{F}_2 & \mathbf{F}_1 & \mathbf{F}_0 \end{bmatrix}. \tag{9}$$

3.2. Activation Function

There are several different activation functions that can be considered for designing hypercomplex-valued neural networks [Trabelsi et al. 2017, Kumar and Tripathi 2019]. In this paper, we consider the so-called split-activation functions [Arena et al. 1997, Gaudet and Maida 2017, Gaudet and Maida 2020]. The main idea is to apply a real activation function to each component of the hypercomplex number. Precisely, given a real-valued activation function $f : \mathbb{R} \to \mathbb{R}$, a split-activation function $Split\_f : \mathbb{H} \to \mathbb{H}$, with $\mathbb{H} \in \{\mathbb{T}, \mathbb{Q}\}$, is defined by

$$Split\_f(z) = f(z_0) + f(z_1)\mathbf{i} + f(z_2)\mathbf{j} + f(z_3)\mathbf{k}, \forall z = z_0 + z_1\mathbf{i} + z_2\mathbf{j} + z_3\mathbf{k} \in \mathbb{H}. \tag{10}$$

3.3. Initialization

Let us now address quaternion and tessarine weight and bias initialization. To this end, we review the expected value and the variance of hypercomplex variables. Given a random variable $Z = Z_0 + Z_1\mathbf{i} + Z_2\mathbf{j} + Z_3\mathbf{k}$ with either quaternion or tessarine values, its expected value is defined by

$$\mathbb{E}[Z] = \mathbb{E}[Z_0] + \mathbb{E}[Z_1]\mathbf{i} + \mathbb{E}[Z_2]\mathbf{j} + \mathbb{E}[Z_3]\mathbf{k}. \tag{11}$$

The definition of variance depends on conjugates. The conjugate of a quaternion $q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$ is $\overline{q} = q_0 - q_1\mathbf{i} - q_2\mathbf{j} - q_3\mathbf{k}$. As a result, $q\overline{q} = q_0^2 + q_1^2 + q_2^2 + q_3^2$. Given a quaternion random variable $Q = Q_0 + Q_1\mathbf{i} + Q_2\mathbf{j} + Q_3\mathbf{k}$ with expected value $\mathbb{E}[Q] = \mu_Q$, its variance [Loots et al. 2012] is defined by

$$Var(Q) = \mathbb{E}\left[(Q - \mu_Q)\overline{(Q - \mu_Q)}\right] = Var(Q_0) + Var(Q_1) + Var(Q_2) + Var(Q_3). \tag{12}$$

In contrast to quaternions, there is not a straightforward definition of tessarine conjugates [Babadağ 2017]. Therefore, it is difficult to define a proper concept of tessarine variance. Nevertheless, this paper is aimed at providing the foundations for tessarine and quaternion-valued deep neural networks. Since initialization is a small part of it, we will focus on developing a quaternion-valued initialization. Then, we shall use the quaternion-valued initialization for tessarine-valued networks as well.

Initialization will be performed based on [He et al. 2015b]. Consider a network with L quaternion layers. Based on a forward propagation, the output of a layer l with input $\mathbf{x}_l$, weight matrix $\mathbf{W}_l$, and bias $\mathbf{b}_l$ is $\mathbf{y}_l = \mathbf{W}_l\mathbf{x}_l + \mathbf{b}_l$. The bias $\mathbf{b}_l$ is initialized as zero. For $l > 1$, we have $\mathbf{x}_l = Split\_f(\mathbf{y}_{l-1})$, where f is an activation function.

Let $\mathcal{Y}_l$, $\mathcal{W}_l$, and $\mathcal{X}_l$ be the quaternion random variables from which the components of $\mathbf{y}_l$, $\mathbf{W}_l$, and $\mathbf{x}_l$ are sampled, respectively. Assuming that n_l is the length of $\mathbf{y}_l$ (after eventual reshapes), because $\mathcal{Y}_l = \mathcal{W}_l\mathcal{X}_l$, we have

$$Var(\mathcal{Y}_l) = n_l Var(\mathcal{W}_l\mathcal{X}_l). \tag{13}$$

If $\mathcal{W}_l$ is symmetrically distributed around zero, so is $\mathcal{Y}_l$. Assuming that $\mathcal{W}_l$ and $\mathcal{X}_l$ are independent, we get

$$Var(\mathcal{Y}_l) = n_l Var(\mathcal{W}_l)\mathbb{E}[\mathcal{X}_l\overline{\mathcal{X}_l}]. \tag{14}$$

Analogously to [He et al. 2015b], for the $Split\_ReLU$ activation function, we obtain

$$\mathcal{X}_l\overline{\mathcal{X}_l} = ReLU(Re(\mathcal{Y}_{l-1}))^2 + ReLU(\mathcal{I}(\mathcal{Y}_{l-1}))^2 + ReLU(\mathcal{J}(\mathcal{Y}_{l-1}))^2 + ReLU(\mathcal{K}(\mathcal{Y}_{l-1}))^2. \tag{15}$$

Assuming that the quaternion components of $\mathcal{W}_{l-1}$ are independent and identically distributed with mean zero and variance $\sigma^2_{W_{l-1}}$, due to the structure of quaternion product, the quaternion components of $\mathcal{Y}_{l-1}$ are also identically distributed with mean zero and variance $\sigma^2_{Y_{l-1}}$. Thus, we have

$$\mathbb{E}[\mathcal{X}_l\overline{\mathcal{X}_l}] = 4\frac{\sigma^2_{Y_{l-1}}}{2} = 2\sigma^2_{Y_{l-1}}. \tag{16}$$

Therefore,

$$Var(\mathcal{Y}_l) = 2n_l Var(\mathcal{W}_l)\sigma^2_{Y_{l-1}}, \tag{17}$$

or, equivalently,

$$\sigma^2_{Y_l} = 2n_l\sigma^2_{W_{l-1}}\sigma^2_{Y_{l-1}}. \tag{18}$$

As proposed in [He et al. 2015b], accumulating over L layers:

$$\sigma^2_{Y_L} = \sigma^2_{Y_1}\prod_{l=1}^{L-1} 2n_l\sigma^2_{W_l}. \tag{19}$$

Concluding, the identity $2n_l\sigma^2_{W_l} = 1$ is a sufficient condition to have a scalar $\sigma^2_{Y_L}$. We should initialize each layer having each component with mean zero and variance $\sigma^2_{W_l} = \frac{1}{2n_l}$. It is also possible to deduce an initialization based on backpropagation. However, the one presented above guarantees that variance will not grow exponentially in backpropagation [He et al. 2015b].

Following a similar procedure, it is possible to make initialization based on [Glorot and Bengio 2010]. In this case,

$$\sigma^2_{W_l} = \frac{1}{2(n_l^{in} + n_l^{out})}. \tag{20}$$

3.4. Hypercomplex Batch Normalization

Ioffe and Szegedy originally proposed real-valued batch normalization (BN) [Ioffe and Szegedy 2015]. Batch normalization consists in centering and normalizing each input and then scaling and shifting it. BN prevents exploding and vanishing gradients, allowing the use of more effective learning rates and reducing overfitting. To implement hypercomplex batch normalization (HBN), we relied on whitening and decorrelation of each of the hypercomplex components [Trabelsi et al. 2017, Gaudet and Maida 2017]. Thus, instead of using hypercomplex-based operations, we used real-valued statistics.

Consider a real-valued random vector $\mathcal{X}$ with mean μ and covariance matrix Σ. Whitening [Kessy et al. 2018] consists in transforming $\mathcal{X}$ into another real-valued random vector $\mathcal{Y} = \mathcal{X}\mathbf{W}$ such that the convariance matrix of $\mathcal{Y}$ is identity matrix I.

In HBN, $\mathcal{X}$ is a four-dimensional vector whose entries represent the components of the tessarine or the quaternion-valued random variable to be whitened (e.g., batch normalization inputs). Furthermore, we apply whitening to $\mathcal{X} - \mu$ because we also want to center the inputs.

$$\begin{aligned} Var(\mathcal{Y}) &= \mathbb{E}[\mathcal{Y}^t\mathcal{Y}] && (21)\\ &= \mathbb{E}[\mathbf{W}^t(\mathcal{X}-\mu)^t(\mathcal{X}-\mu)\mathbf{W}] && (22)\\ &= \mathbf{W}^t\mathbb{E}[(\mathcal{X}-\mu)^t(\mathcal{X}-\mu)]\mathbf{W} && (23)\\ &= \mathbf{W}^t\Sigma\mathbf{W} && (24)\\ &= I && (25) \end{aligned}$$

Multiplying both sides of $\mathbf{W}^t\Sigma\mathbf{W} = I$ by $\mathbf{W}$, we get $\mathbf{W}\mathbf{W}^t\Sigma\mathbf{W} = \mathbf{W}$, or equivalently, $\mathbf{W}\mathbf{W}^t = \Sigma^{-1}$. There are several ways to calculate the matrix $\mathbf{W}$, as discussed in [Kessy et al. 2018]. [Gaudet and Maida 2017] propose choosing $\mathbf{W}$ as the Cholesky factor of Σ^{-1}. Hence, one needs to compute the inverse of a 4×4 square matrix and its Cholesky decomposition. We propose a slight modification to decrease computational cost. First, we compute the Cholesky factor $\mathbf{R}$ of Σ and then we define $\mathbf{W} = \mathbf{R}^{-t}$. It is easy to see that $\mathbf{W}\mathbf{W}^t = \Sigma^{-1} = (\mathbf{R}\mathbf{R}^t)^{-1}$. Using this alternative procedure, we compute the inverse of an upper triangular matrix instead of computing the inverse of a square matrix. Despite its small improvement, as this computation occurs many times in different layers, it reduces the time required for training and prediction.

There is only a slight problem to the approach described above: Σ is positive semidefinite since it is a covariance matrix, and Cholesky decomposition is well defined only for positive definite matrices. To circumvent this problem, we add $\epsilon\mathbf{I}$, with $\epsilon > 0$ a small number. Hence, $\Sigma + \epsilon\mathbf{I}$ is positive definite.

Moreover, like the real-valued BN, the input is scaled and shifted after centering and normalizing within the HBN. In [Ioffe and Szegedy 2015], the authors propose that the network must be able to return to the original variance and mean of the input during learning whenever it is necessary. Hence, we should define the scaling parameter γ as a matrix that can learn $\mathbf{W}^{-1}$. As $\mathbf{W}$ is an upper triangular matrix, its inverse is an upper triangular matrix, and so must be γ. This is another difference from what is proposed by [Gaudet and Maida 2017], as they suggest a symmetric γ matrix. Finally, shifting is performed with a vector β. Thus, given an input $\mathbf{x}$, the output of the hypercomplex-value batch normalization layer is

$$\text{HBN}(\mathbf{x}) = (\mathbf{x} - \mu)\mathbf{W}\gamma + \beta. \tag{26}$$

The parameters μ and Σ are updated during training with an exponential moving mean. Thus, they are not learned, as its correlates in real-valued BN. γ and β are learned during training with the only condition that γ is an upper triangular matrix. We followed the initialization proposed in [Gaudet and Maida 2017] for HBN parameters. β and μ are initialized as zero, and so are the off-diagonal elements of Σ and γ. The diagonals of Σ and γ are initialized as 0.5. HBN considers each tessarine or quaternion feature map independently. There is no relationship between different hypercomplex feature maps.

Finally, we would like to point out that HBN usually depends on heavy computations due to the Cholesky decomposition and the matrix inversion. Alternatively, the real-valued BN could be applied independently on each hypercomplex component like the split-activation functions.

3.5. Residual Learning, Average Pooling and Backpropagation

A problem that arises from training in deep learning is degradation because it has many layers and learning of each of them depends on the learning of the next one in backpropagation. Therefore, layers that have not started learning block the process for those before it. Residual learning [He et al. 2015a], [He et al. 2016] is aimed at overcoming this issue.

Suppose a deep neural network receives an input $\mathbf{x}$ and models a function $h(\mathbf{x})$. If $\mathbf{x}$ is added to its output (skip connection), the network will now model a function $g(\mathbf{x}) = h(\mathbf{x}) - \mathbf{x}$. Skip connection is the basic concept of residual learning. Input signal skips some layers of the network and is added to their output. The main advantage occurs when the network has several aligned residual units (RUs), which are blocks of few layers with a skip connection. The RUs reduce the impact of layers preventing learning of previous ones because there is little dependence between different RUs. Furthermore, the hypercomplex residual learning is as simple as its real-valued counterpart because skip connections consist of adding the input to the output of some intermediate layers, and the addition of hypercomplex numbers operates independently at each component.

Besides, average pooling is based on addition (and division by a real number). Hence, its transformation to tessarines and quaternions is straightforward as well.

Finally, backpropagation depends on derivatives and the chain rule. Working with hypercomplex derivatives is complicated because they depend on the algebra structure and Cauchy-Riemann conditions. Designing a proper and efficient hypercomplex backpropagation algorithm would be out of the scope of this study. Our aim with this paper is to focus on the basic foundations of tessarine and quaternion-valued deep neural networks and to see whether these algebraic systems achieve satisfactory results or not. Thus, like [Trabelsi et al. 2017, Gaudet and Maida 2017], we used the isomorphism between hypercomplex numbers and four-dimensional vectors to train tessarine and quaternion-valued neural networks within the real-valued backpropagation.

4. Computational Experiments

The experiments performed consisted in training real, tessarine and quaternion-valued ResNet models [He et al. 2015a, He et al. 2016] with similar architectures on CIFAR10 database [Krizhevsky 2009]. Recall that the CIFAR10 dataset consists of a training set with 50 thousand images and a test set with 10 thousand divided into 10 classes. The deep neural networks were implemented in `python` with `TensorFlow`. Moreover, the training was performed with *Google Colab Pro* using GPU[1].

The ResNet models are based on stacking residual units (RUs). The RUs are structured as follows: convolution, BN, activation, convolution, BN, activation, and skip connection. Default convolution uses 3×3 kernels, stride of 1, same padding, and regularization L2 with regularizer factor of 10^{-3}. Every few RUs, the number of filters doubles, and the size of the resulting feature maps halves. This is achieved by applying the first convolution of the RU with stride of 2 every time the number of filters doubles. Consequently, a problem with the skip connection arises because it would try to add different numbers of feature maps with mismatched shapes. To overcome this, we apply a convolution with the same number of 1×1 kernels and stride of 2 in the skip connection, followed by batch normalization.

We worked on six neural networks: two real-valued, two tessarine-valued, and two quaternion-valued models. All of them have 16 layers. Apart from the differences created by the algebraic systems, the main difference is the number of filters in each convolution. Let n represent the number of filters in the first convolutional layer. Then, there are seven stacked residual units (RUs): 3 using convolutions with n filters, 2 with $2n$ filters, and 2 with $4n$ filters. Finally, a global average pooling is applied, the result is flattened, and a dense layer with ten units is used for classification purposes with *softmax* activation.

[1]Complete code is available at `https://github.com/FernandoRSenna/Tessarine-and-Quaternion-Valued-Convolutional-Neural-Networks`

For the tessarine and quaternion-valued networks, we adopted $n = 6$ (considering that each kernel is hypercomplex-valued). They use the corresponding hypercomplex convolution, quaternion-based He uniform initialization exposed in Section 3.3, and the $Split\_ELU$ activation function. The dense layer was emulated as a real-valued layer and also used the *softmax* function. Despite having derived initialization based on $Split\_ReLU$, we used $Split\_ELU$ because it yielded higher accuracy than the $Split\_ReLU$. The real-valued equivalent is detailed in [Clevert et al. 2016]. Furthermore, for each hypercomplex number system, there is a network that uses Hypercomplex Batch Normalization (HBN) proposed in Section 3.4 and another with real-valued BN applied to each of the hypercomplex-valued components.

It is also important to contrast the hypercomplex-valued networks with the corresponding real-valued models. Because of the isomorphism between the hypercomplex algebras and four-dimensional vectors, some researchers compare the hypercomplex-valued model with a real-valued network with similar architecture (SA) obtained by multiplying n by 4 [Trabelsi et al. 2017, Gaudet and Maida 2017]. In our case, the real network has $n = 6 \cdot 4 = 24$ filters in the first convolutional layer. However, the real-valued network with similar architecture has more parameters than the hypercomplex-valued models. As an alternative and fairer comparison, we also analysed a real-valued network with similar number of parameters (SP) to the hypercomplex-valued models ($n = 12$). Both real-valued networks use ELU as activation function [Clevert et al. 2016], real batch normalization [Ioffe and Szegedy 2015], and He uniform initialization [He et al. 2015b].

The optimization was performed using stochastic gradient descent within 250 epochs using a mini-batch size of 128 images. The loss function is the categorical cross-entropy. From epoch 1 to 10, the learning rate was 10^{-2}, then it was 10^{-1} until epoch 150, 50 more epochs used 10^{-2}, and the last 50 considered 10^{-3} as the learning rate. We used data augmentation in training with horizontal flip and horizontal and vertical shift of 0.125. We also subtract each pixel mean before training.

Finally, data needs to be preprocessed in the hypercomplex-valued networks because tessarines and quaternions are formed by four real numbers, and the images have three color channels. We encoded each (color) pixel as a pure tessarine or a pure quaternion (zero as real-part), with the RGB values associated with the hyper-imaginary units.

Table 3 presents the results for each network. The total number of parameters is presented. The number of convolutions of each layer may be accessed by the value of n. Test set accuracy is defined by the average accuracy after training each network three times.

Table 3. Networks results comparison.

Network	n	Parameters	Accuracy
Real (SA)	24	404818	87.8%
Real (SP)	12	102478	84.5%
Tessarine (HBN)	6	107170	85.3%
Tessarine (Real BN)	6	104578	84.0%
Quaternion (HBN)	6	107170	84.2%
Quaternion (Real BN)	6	104578	83.9%

The real-valued network with similar architecture to the hypercomplex-valued model presents the best result. However, the tessarine-valued ResNet with HBN yielded considerably high accuracy scores, with around one-fourth of the parameters. It also outperformed the real-valued network with a similar number of parameters (SP). The other hypercomplex-valued networks presented worse accuracy scores than the real-valued network with a similar number of parameters, with the accuracy difference ranging between 0.3% and 0.6%.

When comparing the accuracy of all hypercomplex-valued networks, the tessarine-valued with HBN outperformed the other ones. It may seem that there is little difference between tessarine and quaternion networks when analyzing the similarity of their performance with real-valued BN. However, their behavior with HBN shows that this is not the case. HBN increases tessarine-valued ResNet performance by 1.3%, while the quaternion-valued network yields a minor difference.

Finally, it is important to mention that we did not achieve state of the art performance [He et al. 2015a, He et al. 2016, Trabelsi et al. 2017, Gaudet and Maida 2020]. The main reason is lack of powerful computing tools, which led us to decide to work with networks that had fewer layers and parameters than those proposed in the papers mentioned above. We believe that the smaller accuracy scores are not a major problem because the main focus was on the foundations of tessarine and quaternion-valued networks and comparing their performances with real-valued networks.

5. Conclusions and Future Work

This paper proposed a 2D tessarine-valued convolution, a quaternion-based weight initialization, and improvements on hypercomplex-valued batch normalization. We also discussed how to adapt other operations commonly used in deep learning to tessarine and quaternion-valued neural networks.

Results show that tessarine-valued ResNet model with HBN outperformed its real and quaternion-valued corresponding models with a similar number of parameters. Besides, hypercomplex batch normalization has increased the tessarine-valued network accuracy score.

Currently, many real-based operations are merged with hypercomplex ones to create hypercomplex-valued networks. We do not take complete advantage of the structure of these algebraic systems, as we work with split-activations and mainly the real-based backpropagation. Nevertheless, this paper has presented robust evidence that further studies and developments in this field might improve network performance without increasing their number of parameters.

Future work may focus on the following: tessarine-valued initialization, further reduction in hypercomplex-valued batch normalization computing cost, different activation functions, larger networks to achieve better performance, comparison of diverse ways to represent images as tessarine and quaternion-valued images, hypercomplex dense layers, other neural networks applications, and neural networks based on alternative hypercomplex algebras.

References

Aizenberg, I. (2011). *Complex-valued neural networks with multi-valued neurons*, volume 353. Springer.

Arena, P., Fortuna, L., Muscato, G., and Xibilia, M. G. (1997). Multilayer perceptrons to approximate quaternion valued functions. *Neural Networks*, 10(2):335–342.

Babadağ, F. (2017). A new approach to homothetic motions and surfaces with tessarines. *International Journal of New Technology and Research (IJNTR)*, Volume-3:45–48.

Cerroni, C. (2017). From the theory of "congeneric surd equations" to "segre's bicomplex numbers". *Historia Mathematica*, 44(3):232–251.

Clevert, D., Unterthiner, T., and Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (elus).

Cockle, J. (1848). On certain functions resembling quaternions, and on a new imaginary in algebra. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 33(224):435–439.

Gaudet, C. and Maida, A. (2017). Deep quaternion networks.

Gaudet, C. and Maida, A. (2020). Generalizing complex/hyper-complex convolutions to vector map convolutions.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256. PMLR.

Grassucci, E., Cicero, E., and Comminiello, D. (2021). Quaternion generative adversarial networks.

Géron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 1st edition.

Hamilton, W. R. (1844). On quaternions; or on a new system of imaginaries in algebra. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 25(163):10–13.

Haykin, S. (2009). *Neural Networks and Learning Machines*. Pearson International Edition. Pearson.

He, K., Zhang, X., Ren, S., and Sun, J. (2015a). Deep residual learning for image recognition.

He, K., Zhang, X., Ren, S., and Sun, J. (2015b). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Identity mappings in deep residual networks.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Kessy, A., Lewin, A., and Strimmer, K. (2018). Optimal whitening and decorrelation. *The American Statistician*, 72(4):309–314.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report.

Kumar, S. and Tripathi, B. K. (2019). On the learning machine with compensatory aggregation based neurons in quaternionic domain. *Journal of Computational Design and Engineering*, 6(1):33–48.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Loots, M., Bekker, A., Arashi, M., and Roux, J. (2012). On the real representation of quaternion random variables. *Statistics*, 47:1–17.

Navarro-Moreno, J., Fernández-Alcalá, R. M., Jiménez-López, J. D., and Ruiz-Molina, J. C. (2020). Tessarine signal processing under the t-properness condition. *Journal of the Franklin Institute*, 357(14):10100–10126.

Navarro-Moreno, J. and Ruiz-Molina, J. C. (2021). Wide-sense markov signals on the tessarine domain. a study under properness conditions. *Signal Processing*, 183:108022.

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

Trabelsi, C., Bilaniuk, O., Serdyuk, D., Subramanian, S., Santos, J. F., Mehri, S., Rostamzadeh, N., Bengio, Y., and Pal, C. J. (2017). Deep complex networks. *CoRR*.

Zhu, X., Xu, Y., Xu, H., and Chen, C. (2019). Quaternion convolutional neural networks.