Shared-Operation Hypercomplex Networks for Handwritten Text Recognition

Giorgos Sfikas[1(✉)], George Retsinas[2], Panagiotis Dimitrakopoulos[3], Basilis Gatos[4], and Christophoros Nikou[3]

[1] Department of Surveying and Geoinformatics Engineering, School of Engineering, University of West Attica, Athens, Greece
gsfikas@uniwa.gr

[2] School of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece
gretsinas@central.ntua.gr

[3] Department of Computer Science and Engineering, University of Ioannina, Ioannina, Greece
p.dimitrakopoulos@uoi.gr, cnikou@cse.uoi.gr

[4] Computational Intelligence Laboratory, Institute of Informatics and Telecommunications, National Center for Scientific Research "Demokritos", Athens, Greece
bgat@iit.demokritos.gr

Abstract. Parameterized hypercomplex layers have recently emerged as very useful alternatives of standard neural network layers. They allow for the construction of extremely lightweight architectures, with little to no sacrifice of accuracy. We propose networks of Shared-Operation Parameterized Hypercomplex layers, where the operation parameterization is co-learned by all layers in tandem. In this manner, we mitigate the computational burden of operation parameterization, which grows cubically with respect to the hypercomplex dimension. We attain good word and character error rate at only a small fraction of the memory footprint of non-hypercomplex models as well as previous non-shared operation hypercomplex ones (up to −96.8% size reduction).

Keywords: Parameterized Hypercomplex Layers · Hypercomplex Algebra · Handwritten Text Recognition · Low memory footprint

1 Introduction and Related Work

Handwritten Text Recognition (HTR) is one of the major pattern recognition tasks in the field of document image processing. The most typical use-case involves segmented lines of text images as inputs, which are to be automatically converted to a string of characters. While the task itself is similar to that of Optical Character Recognition (OCR), which conventionally involves recognition of printed text, handwritten text offers a significantly greater challenge.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
G. A. Fink et al. (Eds.): ICDAR 2023, LNCS 14190, pp. 200–216, 2023.
https://doi.org/10.1007/978-3-031-41685-9_13

The variability of handwriting style, which may be important not only between writers, but within the output of the same writer is one of the major difficulties. Indeed, it is not an uncommon occurence to have a learning system that fails, when trained on one writer or group of writers and tested on another [24,25].

Current state-of-the-art systems for HTR include different variants of Deep Neural Networks. Due to the sequential nature of text, Recurrent Neural Networks (RNN) have been a popular choice of neural network for HTR. Combined with Connectionist Temporal Classification (CTC)-based objectives, which allow for a loss value to be computed during training and without requiring exact alignment between prediction and target, RNNs have been the basis of various excellent-performing systems [6,21]. Sequence-to-sequence models constistute an alternative approach to HTR, which is based on decoupling encoding to a feature vector and decoding to the target string as two separate network components. Compared to Convolutional Neural Networks (CNNs), RNNs have the advantage of capturing information dependencies in a sequential manner, and usually have led to better HTR models w.r.t. to the former. Encoding prior knowledge about the nature of our inputs is primarily enforced through the inherent inductive bias that is represented by each model. Specifically, convolutional layers model statistical dependence of some form for each time frame w.r.t. neighbouring frames, or spatial dependence of line image cues w.r.t. spatially close pixels. The dependence range is hard-coded in the form of the characteristics of each convolution, and primarily as the size of each kernel, along with other hyperparameters (dilation, stride). On this note, there has been important recent work on flexible, adaptive convolutional operations, e.g. [9,27]. With recurrent layers on the other hand, we encode our belief that our data are inherently sequential. Forward and backward dependencies are captured easily through bidirectional recurrent variants. Compared to CNNs however, RNNs are known to be difficult to train and converge to an acceptable solution. To this end, architectures that combine convolutional and recurrent components have been proposed [23,26]. A very much used recipe involves a convolutional backbone that is charged with transforming the input segmented image into a useful feature map. This feature map is then pooled or reshaped into a sequence of features that is fed into a recurrent component [13,23,26]. Regarding convolutional-recurrent model architecture, a technique that has also worked well involves supplying the main recurrent network with an auxiliary CTC-based component [26,36]. This can be understood as a penalty on cross-entropy loss over a recurrent decoder. In practice, this CTC shortcut can be fully convolutional, including 2D and 1D (temporal) convolutions, which translates to less recurrent components and faster convergence.

Transformers have emerged as an antagonist to both convolutional and recurrent architectures, in the sense of the aspiration to replace either one for most important vision tasks. Initially proposed in a Natural Language Processing setting [34], they have been tailored for tasks that can be cast as sequential processing, and can in principle capture complex, far-reaching input interdependencies [20]. The main ingredient in transformers is the self-attention operation. Input sequence vectors are transformed into a set of “keys”, “queries” and

"values" through learnable, shared transformations which create a dictionary of features that are subsequently recombined as a softmax-weighted average and transformed through a fully-connected layer into an output sequence. Multiple transformations for the same inputs have shown to work well in practice, which corresponds to the so-called multi-head variant of self-attention, or simply multi-head self-attention. In turn, cascades of multi-head self-attention can be grouped together to form so-called transformer layers [20]. Use of transformers has been explored in the field of HTR by recent work, where excellent results are reported [8]; interestingly, transformers are however related to inherent shortcomings such as poorly handling text repetitions [35], which in turn are rooted to the indirect manner that they handle sequence positional dependence. Another important disadvantage is that models that fully depend on a transformer structure tend to be orders of magnitude larger than their non-transformer counterparts [36].

A direction of research that is orthogonal to optimizing HTR accuracy w.r.t. to NN architectural components and structure, involves creating a network that is as resource-demanding as possible. Network size in terms of numbers of parameters is one such metric. The general trend in learning is to have ever-larger models, with NN sizes reaching billions of parameters in some tasks [20]. The largest models in HTR are Transformer-based and comprise hundreds of millions of parameters [36]. In a real application setting, where budget constraints may be very tight (e.g. on embedded devices), large models are unfortunately inapplicable. To this end, a host of works have explored sparsity in neural networks [4,12,22,39], where, in broad terms, the goal is to train NNs with as many zeroed connections as possible, at as less of an accuracy loss as possible. A family of techniques involves augmenting the training objective with terms that will encourage model sparsity. In [12], a L_0 term is added to the total objective. In [39], a variational inference model is proposed, where model weights follow a zero-mean prior, pushing the posterior towards small magnitudes; values that are under a threshold are pruned. Feature pyramid components are connected with group-wise factors in [4], and in this context variational inference amounts to a neural architecture search scheme. Hypercomplex networks are another group of techniques that follow a very different philosophy in achieving network sparsity. They lead to models which have alternate layers of standard NN layers (fully-connected, convolutional, etc.) but which enforce extensive parameter sharing. Quaternion neural networks were the first type of hypercomplex networks [7,16]. By treating model neurons and weights as quaternionic, which are inherently 4-dimensional, an impressive 75% economy in model size is easily attained. After quaternionic versions of convolutional networks [17,40], proposals for other types of quaternionic layer have followed suit, like recurrent layers, transformers, or extensions to graph neural networks [15,19,32]. Generalizing this paradigm, parameterized hypercomplex networks have recently been proposed as an alternative to quaternionic networks, where the level of parameter sharing is defined as a model hyperparameter. Crucially, in parameterized hypercomplex networks the manner in which weight tuples are multiplied is learnable. This type of multiplication has been named Parameterized Hypercomplex Multiplication (PHM) [38], and has lead to models that were (in practice) downscaled up to 16×.

In this work, we argue that the marginal benefit with respect to increasing network size, structure and training complexity is a critical factor when it comes to choosing an architecture for our HTR model. Is using a model that is $10x$ or $100x$ larger than the previous baseline worth it, only to obtain a decrease in Character Error Rate (CER) by 1–2% as an end result? Also, it is hard to tell whether small accuracy differences generalize well and whether they lead to *any* improvement on out-of-distribution test data. We believe that a trade-off of benchmark accuracy and resource requirements should be considered. In light of the aforementioned considerations, we propose a HTR model that uses a new, even more compact model of Parameterized Hypercomplex Layers that builds on a Convolutional-Recurrent architecture with a CTC shortcut [23,26]. Our model achieves good HTR results at a model size as small as $\sim 500,000$ parameters, which amounts to up to 32-fold compression or 3.1% the size of our baseline model.

The paper is structured as follows. We begin with a brief outline of the prerequisites for hypercomplex algebra and define hypercomplex layers in Sect. 2. In Sect. 3 we present the proposed Shared Hypercomplex architecture for Handwritten Text Recognition. We evaluate the discussed models in Sect. 4, where we show that the proposed model using shared-operation hypercomplex layers performs very adequately on an HTR task while being significantly smaller than competing models; also, we test the model against non-hypercomplex architectures given a tight resource budget. We close with concluding remarks on our contribution and future work in Sect. 5.

2 Hypercomplex Numbers and Hypercomplex Layers

2.1 Quaternions

Hypercomplex algebras are mathematical structures of numbers of "intrinsically high" dimensionality. Historically, quaternions (the set of which is denoted as $\mathbb{H}$ in this text) were the first type of hypercomplex numbers, discovered by Hamilton in the 19^{th} century [10]. Motivated by extending complex numbers $\mathbb{C}$, which are made up of a real and an imaginary part, to an algebra of a multitude of parts, quaternions were defined as numbers q:

$$q = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}, \tag{1}$$

where $a, b, c, d \in \mathbb{R}$ and $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ are imaginary units. The three imaginary units, along with the real unit, are deemed independent and perpendicular to one another. They can also be rewritten as a sum of a scalar part $S(q) = a$ and a vector part $V(q) = b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$, isomorphic to $\mathbb{R}$ and $\mathbb{R}^3$ respectively. All imaginary units admit to being square roots of negative unity:

$$\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = -1, \tag{2}$$

a property which actually extends to infinite elements in $\mathbb{H}$. Additive and multiplication rules are necessary to define an algebra, of which the first is quite

straightforward; corresponding real or imaginary parts are added together, with no operations acting between different number real or imaginary parts. Formally,

$$p + q = (a_p + a_q) + (b_p + b_q)\boldsymbol{i} + (c_p + c_q)\boldsymbol{j} + (d_p + d_q)\boldsymbol{k}, \tag{3}$$

where $p = a_p + b_p\boldsymbol{i} + c_p\boldsymbol{j} + d_p\boldsymbol{k}$ and $q = a_q + b_q\boldsymbol{i} + c_q\boldsymbol{j} + d_q\boldsymbol{k}$. Multiplication of quaternions requires first defining a way to multiply imaginary units. Except Eq. 2, we have

$$\boldsymbol{ij} = \boldsymbol{k}, \boldsymbol{jk} = \boldsymbol{i}, \boldsymbol{ki} = \boldsymbol{j}, \boldsymbol{ji} = -\boldsymbol{k}, \boldsymbol{jk} = -\boldsymbol{i}, \boldsymbol{ki} = -\boldsymbol{j}, \tag{4}$$

and the real unity acts as a multiplicative identity as in $\mathbb{R}$. Note that from Eq. 4 we have the corollary that in general $pq \neq qp$, so multiplication is not commutative in $\mathbb{H}$ (but it still is associative). Combined with a transitive property over our definition of addition, we have the rule of multiplication (also referred to as a "Hamilton" product):

$$pq = S(p)S(q) - V(p) \cdot V(q) + S(p)V(q) + S(q)V(p) + V(p) \times V(q), \tag{5}$$

where $\cdot$ is the inner product and $\times$ is the cross product. Note from the above, that for "pure" quaternions ($S(p) = S(q) = 0$) that are also perpendicular, the multiplication rule becomes simply a cross product $V(p) \times V(q)$. The multiplication rule can be written in an expanded form as:

$$\begin{aligned} pq = &(a_p a_q - b_p b_q - c_p c_q - d_p d_q) + && (6)\\ &(a_p b_q + b_p a_q + c_p d_q - d_p c_q)\boldsymbol{i} + && (7)\\ &(a_p c_q - b_p d_q + c_p a_q + d_p b_q)\boldsymbol{j} + && (8)\\ &(a_p d_q + b_p c_q - c_p b_q + d_p a_q)\boldsymbol{k}, && (9) \end{aligned}$$

where we replaced $S(\cdot)$ and $V(\cdot)$ with their definitions. Especially interestingly regarding the proposed model in this work, we can rewrite the above in a matrix-vector form, as:

$$\begin{bmatrix} a_{pq} \\ b_{pq} \\ c_{pq} \\ d_{pq} \end{bmatrix} = Pq = \begin{bmatrix} a_p & -b_p & -c_p & -d_p \\ b_p & a_p & -d_p & c_p \\ c_p & d_p & a_p & -b_p \\ d_p & -c_p & b_p & a_p \end{bmatrix} \begin{bmatrix} a_q \\ b_q \\ c_q \\ d_q \end{bmatrix}, \tag{10}$$

where for the resulting quaternion pq we write $pq = a_{pq} + b_{pq}\boldsymbol{i} + c_{pq}\boldsymbol{j} + d_{pq}\boldsymbol{k}$. With a slight abuse of notation we write q also to denote the vector $\in \mathbb{R}^4$ that includes the coefficients of quaternion q, and we write P for the matrix that corresponds to quaternion p. Matrices structured as P form a 4-dimensional subspace of $\mathbb{R}^{4\times4}$ that is isomorphic to $\mathbb{H}$, to which we shall refer to as S_4. It is straightforward to see that a basis for S_4 is formed by the following matrices:

$$A_1^4 = I_4, A_2^4 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, A_3^4 = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}, A_4^4 = \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \tag{11}$$

i.e. any matrix in S_4 can be written as a linear combination of $\{A_i\}_{i=1}^4$, and $\{A_i\}_{i=1}^4$ are linearly independent. Hence, we can rewrite Eq. 10 as:

$$\begin{bmatrix} a_{pq} \\ b_{pq} \\ c_{pq} \\ d_{pq} \end{bmatrix} = (a_p A_1^4 + b_p A_2^4 + c_p A_3^4 + d_p A_4^4) \begin{bmatrix} a_q \\ b_q \\ c_q \\ d_q \end{bmatrix}. \tag{12}$$

Now, supposing that we need to multiply N quaternions $p_1, p_2, \cdots, p_N$ with N quaternions $q_1, q_2, \cdots, q_N$, we can rewrite this process again as a matrix-vector product $q^N = P^N q^N$, where our input and resulting vectors are now of dimensions equal to $4N$, and the transformation matrix is $\in \mathbb{R}^{4N \times 4N}$. Depending on whether we want to multiply only N pairs of quaternions $p_i q_i$ for $\forall i \in [1, N]$ or all possible pairs of p_i $\forall i \in [1, N]$ and q_j $\forall j \in [1, N]$, we'll have a dimensionality equal to $4N$ or $4N^2$ respectively. This is still much less than the containing space of $4N \times 4N$, for which $dim\{\mathbb{R}^{4N \times 4N}\} = 16N^2$. Furthermore, the matrix P^N can be written as a sum of Kronecker products of matrices A_1, A_2, A_3, A_4 with matrices that contain the elements of $p_1, p_2, \cdots, p_N$ [38]. Recall that the Kronecker product of $A \in \mathbb{R}^{k \times l}$ and $B \in \mathbb{R}^{m \times n}$ is defined as:

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1l}B \\ \vdots & \ddots & \vdots \\ a_{k1}B & \dots & a_{kl}B \end{bmatrix}, \tag{13}$$

where $A \otimes B \in \mathbb{R}^{km \times ln}$. Note that in general $A \otimes B \neq B \otimes A$, but the two results are related through a "perfect shuffle" permutation [33]. This means that, depending on whether we stack quaternion elements in input and resulting vectors as $a_p^1, a_p^2, a_p^3, \cdots, d_p^2, d_p^3, d_p^4$ or $a_p^1, b_p^1, c_p^1, \cdots, b_p^N, c_p^N, d_p^N$, we can either use a sum of factors with A_i matrices multiplying from the left or from the right. Writing a set of quaternion products as a single Kronecker-factored product is important with respect to the generalization of the Hamilton product to Parameterized Hypercomplex Multiplication, which we discuss in Subsect. 2.2.

Higher-order Hypercomplex Numbers. Quaternions aside, there exist other types of hypercomplex numbers of dimensionality higher than 4. As a rule of the thumb, the more we progress to higher dimensions, the less "easy-to-use" each hypercomplex algebra becomes; for example, quaternions have a non-commutative but associative multiplication rule but octonions and sedenions, of dimensionalities equal to 8 and 16 respectively, are neither commutative nor associative.

2.2 Quaternion and Parameterized Hypercomplex Layers

The restatement of the Hamilton product as a matrix-vector product (Eq. 10) has in fact provided the basis for the definition of quaternion layers as part of quaternionic extensions of standard layers in neural networks. As a multiplication by a matrix corresponds to a linear transformation, this construction has

been used as a replacement of standard linear transformation components. Furthermore, a linear transformation matrix sized $M \times N$, which would normally have a dimension equal to MN, when replaced with its quaternionic counterpart only has a dimension equal to four times less, $MN/4$, as we saw in the previous subsection. In practical terms, this means that the layer uses four times less parameters for each quaternionic component. As linear components are ubiquitous in neural networks, whole networks can be revamped to their quaternionic versions [18].

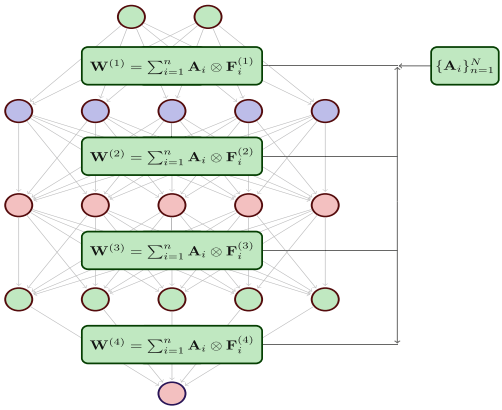


Fig. 1. An illustration of the proposed idea. Standard Parameterized Hypercomplex Networks use multiple layers that are parameterized using two sets of matrices: $\{A_i\}_{i=1}^{n}$ and $\{F_i\}_{i=1}^{n}$, of which the former can be thought of a learnable generalization of the Hamilton product rules. In this work, with "Shared-Operation" Hypercomplex Networks, we argue that learning only one set of $\{A_i\}_{i=1}^{n}$ for the whole network, is enough to construct a useful and very light-weight model.

The major constraint related to quaternionic neural networks is that dimensionality is reduced according to a fixed factor of *four*. Zhang et al. [38] have proposed a generalization of the aforementioned construction from quaternions and four-fold economy to arbitrary-dimension hypercomplex constructions. The matrix W in all cases is defined as a decomposition into matrices $\{A_i\}_{n=1}^{N}$ and $\{F_i\}_{n=1}^{N}$. This is based on writing the transformation matrix W for a given linear component as a sum of Kronecker factors:

$$W = \sum_{i=1}^{n} A_i \otimes F_i, \tag{14}$$

where $A_i \in \mathbb{R}^{n \times n}$ and $F_i \in \mathbb{R}^{f/n \times g/n}$. The resulting matrix W is of size $f \times g$, but the total number of independent parameters is significantly less. Indeed, we have a total of $n^3 + fg/n$ parameters, where supposing that n is much less than either input and output dimensionalities f or g, the second parameter should be

dominant. For a total of L independent linear components in the network, we have $Ln^3 + Lfg/n$ free parameters.

3 Proposed Model for Handwritten Text Recognition

3.1 Shared-Operation Parameterized Hypercomplex Layer

In this work, we propose the use of a new variant of Parameterized Hypercomplex Multiplication Networks (PHM), to which we refer to as *Shared-Operation Hypercomplex Network* (SOHN). "Operation sharing" in SOHN refers to having a shared component that is learned by all hypercomplex layers jointly. Our motivation is related to the three following points:

a) In standard PHM, layer linear components are decomposed as sums of Kronecker products of the form $\sum_i A_i \otimes F_i$. The two sets of matrices A_i, F_i are related to a different intuitive use. Matrices A_i are related to how the shared parameter groups interact with one another. Note for example, that we can understand the quaternionic case as a special case of parameterized hypercomplex multiplication where $n = 4$ and A_i are as in Eq. 11. These matrices control the structure of the resulting Kronecker product, and they are a direct consequence of the definition of the Hamilton product, which in turn stems from the multiplication rules that were set so that $\mathbb{H}$ can form an algebra. In the parameterized hypercomplex case, we no longer have these constraints, and in a sense the equivalent of the Hamilton product is re-learnt for arbitrary n.
b) The number of what we called "independent" linear layers in our network can in effect be significantly larger than the number of layers themselves. For example, for a Hypercomplex LSTM layer, we have 4 or 8 linear transformations assuming unidirectional or bidirectional recurrence; these correspond to each of the related gates (input, output, forget, cell input). For the convolutional case, we can also write a convolution in a matrix-vector form $Wx + b$ with W as a circulant matrix, but it will suffice to deal with a form where parameters are packed in an order-4 tensor as in [5,31]). Nevertheless, the complexity factor related to the size of the A_i matrices can quickly (w.r.t. increasing n) become significant. (In our HTR experiments, we show that choosing $n = 32$ accounts for almost half the network parameter complexity).
c) Also importantly, in practice we have observed that results for $PHM/n = 4$ and quaternionic variants are similar in terms of network accuracy/efficiency (e.g. [31]). This may hint that *learning a separate set of multiplication rules for each linear component is unnecessary.*

For reasons of clarity of presentation, we consider a case where we have only feed-forward connections and all linear operations are hypercomplex; this case can easily be extended to recurrent connections and NNs that include non-hypercomplex layers. Formally, we write our network as a cascade of L layers, i.e.:

$$SOHN(x; \{A_i\}_{i=1}^n, \{F_i^{(1)}\}_{i=1}^n, \{F_i^{(2)}\}_{i=1}^n, \cdots, \{F_i^{(L)}\}_{i=1}^n) =$$

$$l^{(L)}(\{A_i\}_{i=1}^n, \{F_i^{(L)}\}_{i=1}^n) \circ \cdots \circ l^{(1)}(\{A_i\}_{i=1}^n, \{F_i^{(1)}\}_{i=1}^n)(x), \tag{15}$$

each layer uses parameterized hypercomplex operations:

$$y^{(j)} = PHM(y^{(j-1)}; \{A_i\}_{i=1}^n, \{F_i^{(j)}\}_{i=1}^n), \tag{16}$$

where y^j is the output of layer j. In practical terms, a SOHN can be implemented by adding skip connections starting from a single learnable tensor of size $n \times n \times n$ which would represent the A_i matrices, towards all hypercomplex layers. An illustration of this idea can be seen in Fig. 1.

3.2 Model Architecture

As a baseline architecture, in the sense of choosing the layout, number of blocks, channels, dimensionality and other component features, we have followed the convolutional-recurrent architecture proposed in [23]. Written as a Python-style data structure –this will come in handy for comparing architectures in Sect. 4– we denote architecture as $[(2, 64), mpool, (4, 128), mpool, (4, 256)], (256, 3)$. A list of tuples corresponds to convolutional backbone of ResNet blocks, and the final tuple corresponds to the setup of the recurrent component before the softmax output. Tuples for the convolutional backbone signify (number of ResNet blocks, number of channels). "mpool" signifies a 2×2 max-pooling operation. Regarding the tuple corresponding to the recurrent component, it signifies (hidden feature dimensionality, number of LSTM layers). After the final convolutional block, a column-wise max-pooling follows, which branches out to two routes. The first, "main" route is fed to the LSTM recurrent component, followed by a fully-connected layer which maps the output number of channels to the number of output classes, so we get a sequence of softmax-activated probability vectors as a result. This result is fed to a CTC loss. The second route that branches after the column-wise max-pooling operation, is transformed to softmax-activated probability vectors directly, i.e. bypassing the recurrent component. This again is fed to a CTC loss that is weighed down ($0.1\times$) with respect to the loss that corresponds to the first, recurrent route; this architectural choice has shown to aid convergence in recurrent architectures [23,26]. In all cases, we have opted for using channel size of convolutional and recurrent layers that are multiples of the highest hypercomplex parameter that we have used, $n = 32$. This choice was made so as to ease comparisons between different hyperparameter choices. Setting channel sizes according to this rule was not possible in a number of cases, namely regarding input and output size. In these cases, we use 1×1 convolution to transform a tensor from or to the desired channel size. Dropout with probability 20% is used between recurrent layers (Fig. 2).

4 Experiments

4.1 Datasets

We have tested our models on three document image datasets: IAM [14], Rimes [1], and Memoirs [28]. Excerpts from the three datasets can be compared in Fig. 3.

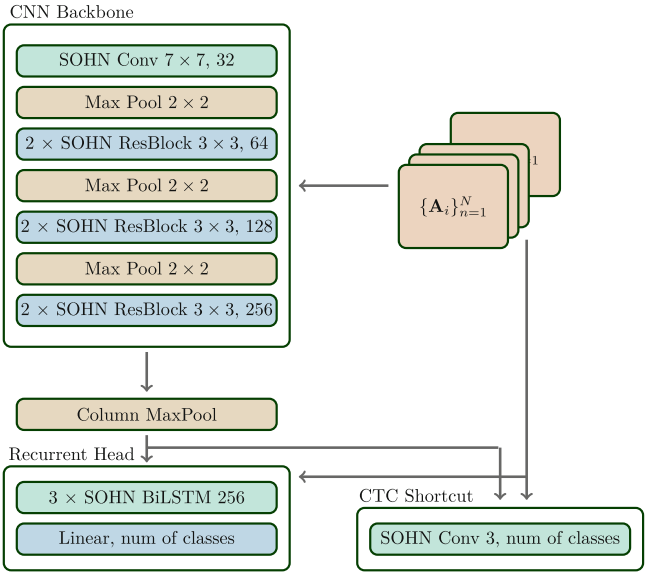


Fig. 2. The proposed shared-operation architecture for HTR.

IAM. The IAM dataset encompasses text produced by 657 different writers, split into writer-independent partitions (we use the same training/validation/set partition as in [21]). There is a total of 6482 lines for the training set, 976 lines for the validation set, and 2915 lines for the test set. IAM is written in a Latin script and in the English language.

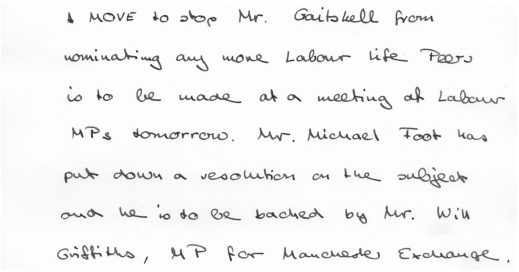
RIMES. RIMES encompasses 11333 lines in the training set and 778 lines in the test set. It is written in a Latin script and in the French language.

Memoirs. The Memoirs dataset [28] comprises 46 manuscripts, written in the 19^{th} as a personal diary of Sophia Trikoupi, sister of a contemporary Greek prime minister. We have a total of $4,941$ words, which correspond to 385 lines for the training set, 129 lines for the validation set and 179 lines of text for the test set. We use the training and test partitions defined in [28].[1] All experiments follow the setting of line-level recognition, with a lexicon-free unconstrained greedy CTC decoding scheme.

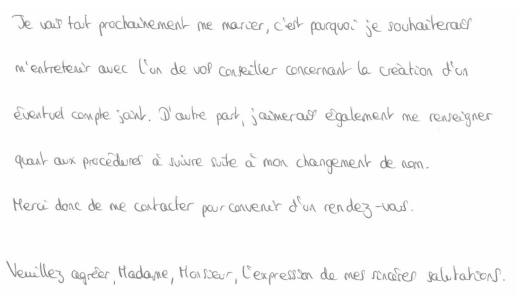
4.2 Varying the Hypercomplex Dimension and PHM vs SOHN

In Table 1 we report results of HTR on the three aforementioned datasets. We have run different variants of the proposed hypercomplex model, using either standard Parameterized Hypercomplex Multiplication [38] (PHM), or the proposed Shared-Operation Hypercomplex variant (SOHN). In both cases we report

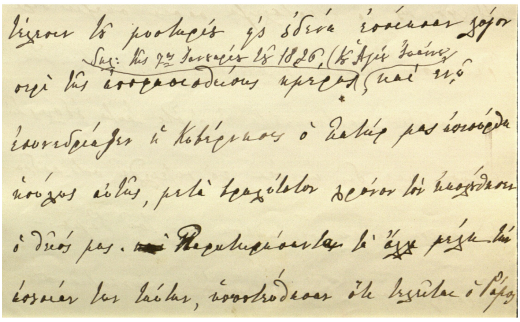
[1] The dataset is publicly available at https://github.com/sfikas/sophia-trikoupi-handwritten-dataset/.



(a) IAM



(b) RIMES



(c) Memoirs

Fig. 3. Excerpts from the datasets used for our experiments.

the employed hypercomplex parameter (n). Different values for n correspond to different structure of the A_i and F_i Kronecker factors, and in general a higher value for n leads to more compression. The exception to this rule comes when the cubic factor of complexity $O(n^3)$ for the A_i matrices surpasses that of the F_i matrices, i.e. $O(fg/n)$. In the SOHN variant the former is significantly mitigated, as we only require a single n^3-sized tensor for the whole network, hence the savings in parameter size. (Note that the difference in accuracy between [23] stems from the custom implementation of LSTM that was written to extend to the hypercomplex variants – the architecture of the "Retsinas et al." and

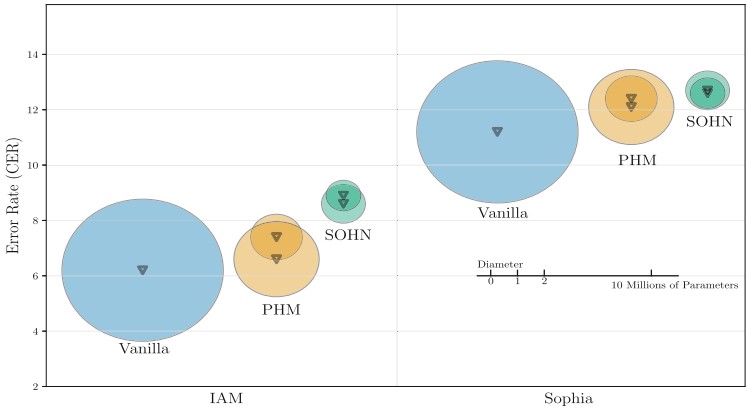


Fig. 4. Comparison of HTR models in terms of model size and test set error rate. The area of each circle is proportional to the number of trainable parameters of a variant in a model family. Different circle sizes for SOHN and PHM correpond to different hyperparameter values ($n = 16$ and $n = 32$ for the larger and smaller radii respectively).

"Real-valued" models as referenced in Table 1 is otherwise identical). A graphical illustration can also be examined in Fig. 4, where we present a comparison of both error rate and model size.

4.3 PHM Model vs Real-Valued Model on a Resource Budget

We have compared against a variants of CNN-RNN with a constrained, fixed budget of trainable parameters. We have tested models on two different budgets, namely 500 thousand and 750 thousand parameters, which we can examine in Tables 2 and 3 respectively. In these cases, we used the proposed SOHN models for hypercomplex dimensions $n = 16$ and $n = 32$, and compared against non-hypercomplex models that use approximately the same number of parameters. As there is naturally more than one way to build a model given a set budget, we built different non-hypercomplex models in order to have as much an unbiased and fair comparison as possible. The setups for these model variants are:

1. $[(4, 64)], (64, 2)$ ($500k$ parameters)
2. $[(1, 64)], (100, 2)$ ($513k$ parameters)
3. $[(1, 64)], mpool, (1, 128)], (64, 2)$ ($576k$ parameters)
4. $[(2, 32), mpool, (2, 64), mpool, (2, 64)], (96, 1)$ ($521k$ parameters)
5. $[(2, 8), mpool, (4, 8), mpool, (4, 8)], (112, 2)$ ($494k$ parameters)
6. $[(1, 64), mpool, (2, 112)], (64, 2)$ ($746k$ parameters)
7. $[(2, 96)], (100, 2)$ ($768k$ parameters)
8. $[(2, 16), mpool, (4, 18), mpool, (4, 32)], (128, 2)$ ($744k$ parameters)

where we have used the same convention to represent architectural choices as in Sect. 3.

Table 1. Comparison of HTR models in terms of model size and test set error rate. Model size is evaluated in terms of model number of parameters, and error rate is evaluated in terms of CER over the test set of three different datasets (IAM [14], RIMES [1], Memoirs [28]). For both considered metrics, a lower value is more desirable. Both works of Li et al. [11] and Wick et al. (+syn) [36] use additional synthetic data in training, while for the rest only the predefined training set is used to train the model.

	#Millions of Params	IAM	RIMES	Memoirs
Li et al. (+syn) [11]	334	3.42	–	–
Li et al. (+syn) [11]	558	2.89	–	–
Wick et al. (+syn) [36]	4.8	3.96	–	–
Wick et al. [36]	4.8	5.09	–	–
Yousef et al. [37]	3.4	4.9	–	–
Diaz et al. [3]	12	2.75	–	–
Retsinas et al. [23]	10	4.62	2.75	–
Real-valued model	10	6.2	3.9	11.2
PHM/$n = 2$ model	5.1	6.5	6.5	11.4
Quaternion model	2.6	6.9	4.3	11.8
PHM/$n = 4$ model	2.6	6.9	7.0	11.4
PHM/$n = 8$ model	1.4	7.5	4.7	11.8
PHM/$n = 16$ model	1.03	7.4	4.5	12.4
SOHN/$n = 16$ model	0.74	8.6	5.5	12.7
PHM/$n = 32$ model	2.8	6.6	3.9	12.1
SOHN/$n = 32$ model	0.46	8.9	5.7	12.6

As a remark on the results of Tables 2, 3, we can state that it seems that trimming down a network to less parameters leads to error rates that vary widely; some of the compared variants are much worse than the proposed SOHN models, while others come quite close. We can however deduce that while applying a Shared-Operation architecture invariably leads to a slight detriment of accuracy, attempting to enforce a constrained budget on a real-valued architecture is very much dependent on the way the baseline network is "pruned-down".

Table 2. Peformance comparison of HTR models on a budget of $500k$ parameters. Loss, CER and WER over the IAM test set are reported. See text for details.

Model type	Number of Parameters	Test Loss	Best CER	Best WER
Ours (SOHN/n=32)	$459k$	25.3	8.9	29.6
Non-PHM variant #1	$500k$	35.4	16.6	50.9
Non-PHM variant #2	$513k$	72.4	37.2	82.7
Non-PHM variant #3	$576k$	44.8	17.3	52.9
Non-PHM variant #4	$521k$	27.3	9.0	30.6
Non-PHM variant #5	$494k$	61.7	26.3	65.4

Table 3. Peformance comparison of HTR models on a budget of $750k$ parameters. Loss, CER and WER over the IAM test set are reported. See text for details.

Model type	Number of Parameters	Test Loss	Best CER	Best WER
Ours (SOHN/n=16)	$742k$	26.4	8.6	28.9
Non-PHM variant #6	$746k$	34.2	13.1	42.2
Non-PHM variant #7	$768k$	53.7	23.6	64.5
Non-PHM variant #8	$744k$	31.2	10.9	34.9

5 Conclusion and Future Work

We have presented a new type of hypercomplex architecture called Shared-Operation Hypercomplex Networks. This scheme is based on the idea that the multiplication operation matrices used in hypercomplex layers can be shared across the network. As more parameters are shared, the computational complexity alongside the trainable parameters is significantly reduced. Our claims are *to an extent* corroborated by the reported experimental results, which test SOHN based networks and PHM variants of our method against the current state of the art in handwritten text recognition. While cutting down the number of matrices to a single one does not throw off the model mechanics and adequate error rates are still achieved, our experiments suggest that learning multiple A_i matrices (as in standard PHM) is still beneficial, and by no means the difference in accuracy between PHM and SOHN is insignificant. In general however, integration of the proposed method results in high network compression with small accuracy drop. For future work, we plan to extend the model with other ways of constraining the expenses related to the two Kronecker factors, like low-rank approximations or other types of factorizations. Another direction of research could involve imposing contraints in the form of a probabilistic prior over the A_i matrices, which could in this context help regulate between choosing a global, shared representation versus a local representation [29,30]. In terms of applications, interesting use cases include segmentation-free HTR frameworks [2] or word-level recognition systems [26].

Acknowledgments. This research has been partially co - financed by the EU and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call "OPEN INNOVATION IN CULTURE", project *Bessarion* (T6YBΠ - 00214).

References

1. Augustin, E., Carré, M., Grosicki, E., Brodin, J.M., Geoffrois, E., Prêteux, F.: Rimes evaluation campaign for handwritten mail processing. In: International Workshop on Frontiers in Handwriting Recognition (IWFHR 2006), pp. 231–235 (2006)
2. Coquenet, D., Chatelain, C., Paquet, T.: DAN: a segmentation-free document attention network for handwritten document recognition. IEEE Trans. Pattern Anal. Mach. Intell. **45**, 8227–8243 (2023)
3. Diaz, D.H., Qin, S., Ingle, R., Fujii, Y., Bissacco, A.: Rethinking text line recognition models. arXiv preprint arXiv:2104.07787 (2021)
4. Dimitrakopoulos, P., Sfikas, G., Nikou, C.: Variational feature pyramid networks. In: International Conference on Machine Learning, pp. 5142–5152. PMLR (2022)
5. Grassucci, E., Zhang, A., Comminiello, D.: Lightweight convolutional neural networks by hypercomplex parameterization. arXiv preprint arXiv:2110.04176 (2021)
6. Graves, A., Fernández, S., Gomez, F., Schmidhuber, J.: Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 369–376. ACM (2006)
7. Isokawa, T., Kusakabe, T., Matsui, N., Peper, F.: Quaternion neural network and its application. In: Palade, V., Howlett, R.J., Jain, L. (eds.) KES 2003. LNCS (LNAI), vol. 2774, pp. 318–324. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45226-3_44
8. Kang, L., Riba, P., Rusiñol, M., Fornés, A., Villegas, M.: Pay attention to what you read: non-recurrent handwritten text-line recognition. Pattern Recogn. **129**, 108766 (2022)
9. Knigge, D.M., et al.: Modelling long range dependencies in ND: from task-specific to a general purpose CNN. arXiv preprint arXiv:2301.10540 (2023)
10. Kuipers, J.B.: Quaternions and Rotation Sequences: A Primer with Application to Orbits, Aerospace and Virtual Reality. Princeton University Press, Princeton (1999)
11. Li, M., et al.: TROCR: transformer-based optical character recognition with pre-trained models. arXiv preprint arXiv:2109.10282 (2021)
12. Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through l_0 regularization. arXiv preprint arXiv:1712.01312 (2017)
13. Markou, K., et al.: A convolutional recurrent neural network for the handwritten text recognition of historical greek manuscripts. In: Del Bimbo, A., et al. (eds.) ICPR 2021. LNCS, vol. 12667, pp. 249–262. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-68787-8_18
14. Marti, U.V., Bunke, H.: The IAM-database: an English sentence database for offline handwriting recognition. Int. J. Doc. Anal. Recogn. **5**(1), 39–46 (2002)
15. Nguyen, T.D., Phung, D., et al.: Quaternion graph neural networks. In: Asian Conference on Machine Learning, pp. 236–251. PMLR (2021)
16. Nitta, T.: A quaternary version of the backpropagation algorithm. In: Proceedings of ICNN 1995 - International Conference on Neural Networks, pp. 2753–2756 (1995)

17. 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), pp. 8514–8518. IEEE (2019)
18. Parcollet, T., Morchid, M., Linarès, G.: A survey of quaternion neural networks. Artif. Intell. Rev. **53**(4), 2957–2982 (2020)
19. Parcollet, T., et al.: Quaternion recurrent neural networks. arXiv preprint arXiv:1806.04418 (2018)
20. Prince, S.J.: Understanding Deep Learning. MIT Press (2023). https://udlbook.github.io/udlbook/
21. Puigcerver, J.: Are multidimensional recurrent layers really necessary for handwritten text recognition? In: 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), vol. 1, pp. 67–72. IEEE (2017)
22. Retsinas, G., Elafrou, A., Goumas, G., Maragos, P.: Online weight pruning via adaptive sparsity loss. In: 2021 IEEE International Conference on Image Processing (ICIP), pp. 3517–3521. IEEE (2021)
23. Retsinas, G., Sfikas, G., Gatos, B., Nikou, C.: Best practices for a handwritten text recognition system. In: Uchida, S., Barney, E., Eglin, V. (eds.) DAS 2022. LNCS, vol. 13237, pp. 247–259. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-06555-2_17
24. Retsinas, G., Sfikas, G., Louloudis, G., Stamatopoulos, N., Gatos, B.: Compact deep descriptors for keyword spotting. In: 2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR), pp. 315–320. IEEE (2018)
25. Retsinas, G., Sfikas, G., Nikou, C.: Iterative weighted transductive learning for handwriting recognition. In: Lladós, J., Lopresti, D., Uchida, S. (eds.) ICDAR 2021. LNCS, vol. 12824, pp. 587–601. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86337-1_39
26. Retsinas, G., Sfikas, G., Nikou, C., Maragos, P.: From Seq2Seq to handwritten word embeddings. In: British Machine Vision Conference (BMVC) (2021)
27. Romero, D.W., Bruintjes, R.J., Tomczak, J.M., Bekkers, E.J., Hoogendoorn, M., van Gemert, J.C.: Flexconv: continuous kernel convolutions with differentiable kernel sizes. arXiv preprint arXiv:2110.08059 (2021)
28. Sfikas, G., Giotis, A.P., Louloudis, G., Gatos, B.: Using attributes for word spotting and recognition in polytonic greek documents. In: 2015 13th International Conference on Document Analysis and Recognition (ICDAR), pp. 686–690. IEEE (2015)
29. Sfikas, G., Nikou, C., Galatsanos, N., Heinrich, C.: MR brain tissue classification using an edge-preserving spatially variant Bayesian mixture model. In: Metaxas, D., Axel, L., Fichtinger, G., Székely, G. (eds.) MICCAI 2008. LNCS, vol. 5241, pp. 43–50. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85988-8_6
30. Sfikas, G., Nikou, C., Galatsanos, N., Heinrich, C.: Majorization-minimization mixture model determination in image segmentation. In: CVPR 2011, pp. 2169–2176. IEEE (2011)
31. Sfikas, G., Retsinas, G., Gatos, B., Nikou, C.: Hypercomplex generative adversarial networks for lightweight semantic labeling. In: El Yacoubi, M., Granger, E., Yuen, P.C., Pal, U., Vincent, N. (eds.) ICPRAI 2022, Part I. LNCS, vol. 13363, pp. 251–262. Springer, Cham (2022)
32. Tay, Y., et al.: Lightweight and efficient neural natural language processing with quaternion networks. arXiv preprint arXiv:1906.04393 (2019)
33. Van Loan, C.F.: The ubiquitous kronecker product. J. Comput. Appl. Math. **123**(1–2), 85–100 (2000)

34. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
35. Wick, C., Zöllner, J., Grüning, T.: Transformer for handwritten text recognition using bidirectional post-decoding. In: Lladós, J., Lopresti, D., Uchida, S. (eds.) ICDAR 2021. LNCS, vol. 12823, pp. 112–126. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86334-0_8
36. Wick, C., Zöllner, J., Grüning, T.: Rescoring sequence-to-sequence models for text line recognition with CTC-prefixes. In: Uchida, S., Barney, E., Eglin, V. (eds.) DAS 2022. LNCS, vol. 13237, pp. 260–274. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-06555-2_18
37. Yousef, M., Hussain, K.F., Mohammed, U.S.: Accurate, data-efficient, unconstrained text recognition with convolutional neural networks. Pattern Recogn. **108**, 107482 (2020)
38. Zhang, A., et al.: Beyond fully-connected layers with quaternions: Parameterization of hypercomplex multiplications with $1/n$ parameters. In: International Conference on Learning Representations (ICLR 2021) (2021)
39. Zhao, C., Ni, B., Zhang, J., Zhao, Q., Zhang, W., Tian, Q.: Variational convolutional neural network pruning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2780–2789 (2019)
40. Zhu, X., Xu, Y., Xu, H., Chen, C.: Quaternion convolutional neural networks. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) ECCV 2018. LNCS, vol. 11212, pp. 645–661. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01237-3_39