Compressing deep-quaternion neural networks with targeted regularisation

Riccardo Vecchi ✉, Simone Scardapane, Danilo Comminiello, Aurelio Uncini

Department of Information Engineering, Electronics and Telecommunications (DIET), 'Sapienza' University of Rome, Roma Lazio 00185, Italy
✉ *E-mail: r.vecchi92@gmail.com*

Abstract: In recent years, hyper-complex deep networks (such as complex-valued and quaternion-valued neural networks – QVNNs) have received a renewed interest in the literature. They find applications in multiple fields, ranging from image reconstruction to 3D audio processing. Similar to their real-valued counterparts, quaternion neural networks require custom regularisation strategies to avoid overfitting. In addition, for many real-world applications and embedded implementations, there is the need of designing sufficiently compact networks, with few weights and neurons. However, the problem of regularising and/or sparsifying QVNNs has not been properly addressed in the literature as of now. In this study, the authors show how to address both problems by designing targeted regularisation strategies, which can minimise the number of connections and neurons of the network during training. To this end, they investigate two extensions of ℓ_1 and structured regularisations to the quaternion domain. In the authors' experimental evaluation, they show that these tailored strategies significantly outperform classical (real-valued) regularisation approaches, resulting in small networks especially suitable for low-power and real-time applications.

1 Introduction

Deep neural networks have achieved remarkable results in a variety of tasks and applicative scenarios over the last years [1]. Several breakthroughs have been obtained by designing custom neural modules for exploiting structure in data, such as the spatial organisation of pixels in an image, temporal or sequential information, and so on. The field of quaternion deep learning aims at extending these results to problems for which a hyper-complex representation (as opposed to a real-valued representation) is more adequate [2, 3] (see also [4] for earlier works on the field). Among these approaches, the most common is the use of quaternion values (a straightforward extension of the complex algebra) for representing input values, weights, and output values of the network. The resulting quaternion-valued neural networks (QVNNs) have been successfully applied to, among others, image classification [2, 5, 6], image colouring and forensics [7], natural language processing [8], graph embeddings [9], human motion recognition [10], and 3D audio processing [11]. By exploiting the properties of the quaternion algebra, QVNNs can achieve similar or higher accuracy than their real-valued counterparts, while requiring fewer parameters and computations. In fact, outside of the deep learning field, the use of quaternion representations is well established in, e.g. the robotics community for linear filtering processing [12].

The majority of the literature on QVNNs up to this point has focused on extending standard deep learning operations, such as convolution [2], batch normalisation (BN) [7], or weight initialisation [3], to the quaternion domain. Less attention, however, has been devoted to properly extending other aspects of the training process, including accelerated optimisation algorithms [13] and regularisation strategies. In particular, in many real-world scenarios (e.g. embedded devices), users need to take into careful consideration of the complexity and computational costs, by making the networks as small as possible while maintaining a good degree of accuracy [14, 15].

In the real-valued case, these constraints have been analysed in detail, and several strategies have been developed. Most commonly, compression of neural networks can be achieved while training by applying several regularisation strategies, such as ℓ_2,

ℓ_1, or group sparse norms [16–18], which can target either single weights or entire neurons. A direct extension of these strategies to the case of QVNN, as done in the current literature, applies them independently on the four components of each quaternion weight. However, in this paper, we argue and show experimentally later on that this trivial extension results in highly sub-optimal regularisation procedures, which do not sufficiently exploit the properties of the quaternion algebra. In fact, the problem of sparsifying a quaternion is not restricted to neural networks, but it has received attention from other disciplines, most notably quaternion extensions of matching pursuit [19], and compressive sensing [20]. To the best of our knowledge, however, almost no work has been devoted to the specific problem of targeting quaternion-valued sparsity in QVNNs, possibly hindering their application in certain applicative scenarios.

1.1 Contributions of the paper

In this paper, we leverage on prior works on compression of real-valued networks and quaternion-valued norms to propose two *targeted* regularisation strategies for QVNNs, filling an important gap in the literature.

(i) The first regularisation approach that we propose (Section 1) extends the use of ℓ_1 regularisation to consider a single quaternion weight as a unitary component, and it is akin to a structured form of regularisation in the real-valued case. It allows us to remove entire quaternion weights simultaneously, instead of each of their four components independently.
(ii) The second strategy that we propose is instead defined at the level of a single quaternion neuron, extending ideas from [21] to a quaternion domain, thus allowing to remove entire units from the network at once. We consider real-valued regularisation to be applied to quaternion extensions of BN, in which every neuron is scaled by a single real-valued coefficient and is eventually removed by the optimisation process. Thus, we achieve better stability (thanks to the use of BN) and sparsity at the same time.

In our experimental evaluation, we show that these two proposed strategies significantly outperform the naive application of classical regularisation strategies on two standard image recognition benchmarks. The resulting QVNNs are thus smaller (both in terms of neurons and weights) and require a markedly smaller computational footprint when running in inference mode, with up to 5× reductions in the number of connections and 3× speedups in the inference time.

1.2 Organisation of the paper

Section 2 recalls quaternion-algebra and QVNNs. Section 3 describes our two proposed regularisation strategies. We provide an experimental evaluation in Section 4, concluding in Section 5.

1.3 Notation

In the remaining of this paper, the use of a subscript $\{i, j, k\}$ always refers to the respective imaginary component of the corresponding quaternion number, as explained in Section 1. x^* is the conjugate of x. We use bold letters, e.g. $\boldsymbol{x}$, for vectors, and uppercase bold letters, e.g. $\boldsymbol{X}$, for matrices. For indexing, we use a superscript l to refer to the lth layer in a network, while we use bracket notation to refer to an element inside a set (e.g. $\boldsymbol{x}(n)$ is the nth input element of the training set).

2 Preliminaries

2.1 Quaternion algebra

A quaternion-valued number $x \in \mathbb{H}$ can be represented by a tuple of four real-valued numbers $(x_r, x_i, x_j, x_k) \in \mathbb{R}^4$ as [22]

$$x = x_r + ix_i + jx_j + kx_k, \qquad (1)$$

where the three imaginary units i, j, k satisfy the fundamental axiom of quaternion algebra $i^2 = j^2 = k^2 = ijk = -1$. Given two quaternions x and y, we can define their sum as

$$z = (x_r + y_r) + i(x_i + y_i) + j(x_j + y_j) + k(x_k + y_k), \qquad (2)$$

and similarly for multiplication by a real number. More importantly, the (Hamilton) product between the two quaternions is given by

$$
\begin{aligned}
x \otimes y = & \left(x_r y_r - x_i y_i - x_j y_j - x_k y_k \right) \\
& + i \left(x_r y_i + x_i y_r + x_j y_k + x_k y_j \right) \\
& + j \left(x_r y_j - x_i y_k + x_j y_r + x_k y_i \right) \\
& + k \left(x_r y_k + x_i y_j - x_j y_i + x_k y_r \right).
\end{aligned}
\qquad (3)
$$

Note that the product is not commutative, setting apart quaternion algebra from its complex- and real-valued restrictions.

2.2 Quaternion-valued neural networks

QVNNs are flexible models for transforming quaternion-valued vector inputs $\boldsymbol{x} \in \mathbb{H}^d$ to the desired target value $\boldsymbol{y}$, which in the majority of cases are real-valued (e.g. a probability distribution over a certain number of classes). A standard, fully-connected layer of a QVNN is given by

$$g(\boldsymbol{h}^{l+1}) = \sigma(\boldsymbol{W} \otimes \boldsymbol{h}^l + \boldsymbol{b}), \qquad (4)$$

where $\boldsymbol{h}^l$ is the input to the layer, $\boldsymbol{W}$ is a quaternion-valued matrix of adaptable coefficients with components $(\boldsymbol{W}_r, \boldsymbol{W}_i, \boldsymbol{W}_j, \boldsymbol{W}_k)$ (and

similarly for $\boldsymbol{b}$), $\otimes$ performs matrix–vector multiplication according to the Hamilton product in (3), and $\sigma(\cdot)$ is a proper element-wise non-linearity. Similarly to the complex-valued case [23], choosing an activation function is more challenging than for real-valued NNs, and most works adopt a split-wise approach where a real-valued function σ_r is applied component-wise

$$\sigma(s) = \sigma_r(s_r) + i\sigma_r(s_i) + j\sigma_r(s_j) + k\sigma_r(s_k), \qquad (5)$$

where s is a generic activation value. Customarily, the input to the first layer is set to $\boldsymbol{h}^1 = \boldsymbol{x}$, while the output of the last layer is the desired target $\boldsymbol{h}^L = \boldsymbol{y}$. If the target is real-valued, one can transform $\boldsymbol{h}^L$ to a real-valued vector by taking the absolute value element-wise, and eventually apply one or more real-valued layers afterwards. In addition, (4) can be easily extended to consider convolutive layers [2] and recurrent formulations [3].

2.3 Optimisation of QVNNs

Now consider a generic QVNN $f(\boldsymbol{x})$ obtained by composing an arbitrary number of layers in the form of (4) or its extensions. We receive a dataset of N examples $\{\boldsymbol{x}(n), \boldsymbol{y}(n)\}_{n=1}^{N}$, and we train the network by optimising

$$J(\theta) = \frac{1}{N} \sum_{n=1}^{N} l(\boldsymbol{y}(n), f(\boldsymbol{x}(n))) + \lambda \cdot r(\theta), \qquad (6)$$

where θ is the set of all (quaternion-valued) parameters of the network, l is a loss function (e.g. mean-squared error, cross-entropy loss), and r is a regularisation function weighted by a scalar $\lambda \geq 0$. Because the loss function in (6) is non-analytic, one has to resort to the generalised QR-calculus to define proper gradients for optimisation [13]. Luckily, these gradients coincide with the partial derivatives of (6) with respect to all the real-valued components of the quaternions, apart from a scale factor. For this reason, it is possible to optimise (6) using standard tools from stochastic optimisation popular in the deep learning literature, such as Adam or momentum-based optimisers.

While most components described up to now have received considerable attention in the literature, the design of a correct regularisation term $r(\cdot)$ in (6) has been mostly ignored, and it is the focus of the next section.

3 Targeted regularisation for QVNNs

In the real-valued case, a classical choice for the regulariser $r(\cdot)$ in (6) is the ℓ_2 norm. Whenever sparsity is desired, it can be replaced with the ℓ_1 norm, or a proper group version is acting at a neuron-level [16, 17]. In most implementations of QVNNs, these regularisers are applied element-wise on the four components of each quaternion weight. For example, ℓ_1 regularisation in this form can be written as

$$r(\theta) = \sum_{w \in \theta} \left(|w_r| + |w_i| + |w_j| + |w_k| \right). \qquad (7)$$

We argue that, because of the decoupling across the four components, this operation results in far less regularisation and sparsity than one could expect. This is inconvenient both from the generalisation point of view, and from an implementation perspective, where smaller, more compact networks are desired. In this section, we present two *targeted* regularisation strategies, acting on each quaternion as a unitary component, resulting in a more principled form of regularisation for QVNNs.

3.1 ℓ_1 regularisation for quaternion weights

Given any weight $w \in \mathbb{H}$ of the QVNN (i.e. a single element of θ from (6)), the first method we explore is to regularise its norm as

$$r(w) = \frac{1}{Q}\sqrt{w_r^2 + w_i^2 + w_j^2 + w_k^2} = \frac{1}{Q}\sqrt{w^* \otimes w}, \qquad (8)$$

where Q is the number of weights in the network (allowing to split the influence of (8) on each weight with respect to the loss function (6)), and w^* is the conjugate of w, i.e. $w^* = w_r - iw_i - jw_j - kw_k$ (combining the definition of a conjugate with (3), and removing all terms except the first line, shows the second equality in (8)).

This method can be seen as the natural extension of ℓ_1 norm minimisation on a quaternionic signal [20]. It is also equivalent to a structured form of sparsity [17], where we group all the components of the quaternion w together. As a result, minimising (8) will tend to bring the entire quaternion weight to 0, instead of each component independently (similar to how structured sparsity in a real-valued network brings all the incoming or outgoing weights of the network towards zero together [17]).

3.2 Sparse regularisation with quaternion BN

The method described in Section 1 is effective for removing single quaternion weights, but in many real-world scenarios, we also require a principled way to remove entire neurons during the training process [17]. To this end, we investigate a hyper-complex extension of the technique originally proposed in [21]. The basic

Table 1 Average results on MNIST and CIFAR-10 with several weight-level regularisation strategies

Dataset	Measure	No Reg.	ℓ_2	ℓ_1	R_Q	R_{QL}
MNIST	test accuracy, %	98.95	96.69	93.46	96.81	96.29
	component sparsity, %	1.71	54.68	34.08	68.40	75.08
	quaternion sparsity, %	**0.0**	**40.01**	**23.45**	**53.82**	**69.42**
CIFAR-10	test accuracy, %	71.30	72.58	73.43	72.03	73.20
	component sparsity, %	0.77	8.58	44.24	59.29	35.91
	quaternion sparsity, %	**0.0**	**4.77**	**42.61**	**58.73**	**31.89**

With the bold fonts, we highlight the (average) quaternion sparsity.

idea is to compose each layer in (4) with a BN layer [24], and then perform sparse regularisation on the parameters of the BN layer, indirectly removing the original neurons in the network. We briefly recall that BN, originally introduced in [24], allows each neuron to adapt the mean and variance of its activation values, with the general effect of stabilising training and possibly simplifying optimisation.

For implementing the BN model in the quaternion domain, we build on [7] after we consider a single output in (4), i.e. the quaternion-valued output of a single neuron in the network. During training, we observe a mini-batch of B inputs $x(1), \ldots, x(B)$ (a subset of the full dataset) and corresponding outputs of the neuron $h(1), \ldots, h(B)$ (we do not use an index for the neuron for notational simplicity). We can compute the mean and variance of the mini-batch as

$$\hat{\mu} = \frac{1}{B}\sum_{n=1}^{B} \left[h_r(n) + ih_i(n) + jh_j(n) + kh_k(n) \right], \qquad (9)$$

$$\hat{\sigma}^2 = \frac{1}{B}\sum_{n=1}^{B} \left(h(n) - \hat{\mu} \right) \otimes \left(h(n) - \hat{\mu} \right)^*. \qquad (10)$$

These values are computed dynamically during training, while they are set to a fixed (pre-computed) value during inference. The output of the BN layer is defined as [7]

$$h_{\mathrm{BN}}(n) = \left(\frac{h(n) - \hat{\mu}}{\sqrt{\hat{\sigma}^2 + \varepsilon}} \right) \gamma + \beta, \qquad (11)$$

where ε is a small value added to ensure stability, while $\gamma \in \mathbb{R}$ and $\beta \in \mathbb{H}$ are trainable parameters initialised at 1 and 0, respectively. Key for our proposal, the γ parameter in (11) is real-valued, allowing us to apply standard real-valued regularisation. In particular, similar to [21], we apply (real-valued) ℓ_1 regularisation on the γs, since pushing a single γ to zero effectively allows us to remove the entire neuron in the QVNN. Thus, denoting with Γ the set of all γ parameters in the network, we regularise them as

$$r(\Gamma) = \frac{1}{|\Gamma|}\sum_{\gamma \in \Gamma} |\gamma|, \qquad (12)$$

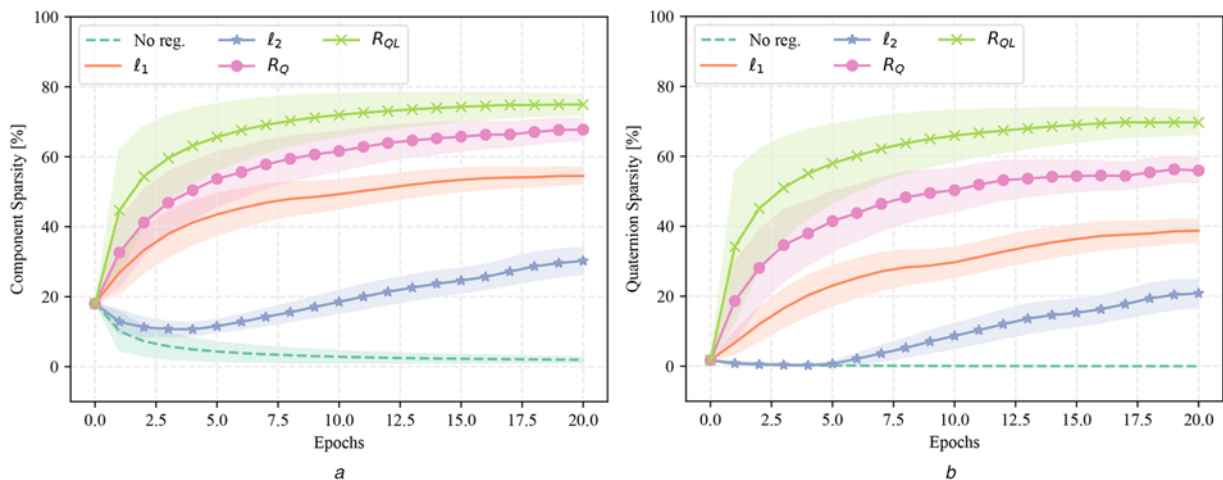where $|\Gamma|$ is the cardinality of the set Γ.

Fig. 1 *Evolution of the sparsity (both at the quaternion level and at the level of individual components) for the different strategies under consideration for the first 20 epochs. See the text for a description of the acronyms*
a Weight sparsity
b Quaternion-level sparsity

Table 2 Average training time (in seconds) for the different approaches

Dataset	No Reg.	ℓ_2	ℓ_1	R_Q	R_{QL}
MNIST	179.35	183.17	185.43	184.22	186.22
CIFAR-10	924.77	1146.89	1160.48	1151.50	1168.37

3.3 Mixed regularisation strategies

The strategies described in the previous sections are not exclusive, and we can explore several *mixed* strategies with different regularisation weights. In our experimental section, we consider combining the two strategies, as long as one of the two strategies are combined with a classical ℓ_1 regularisation to be applied independently on each component.

4 Experimental results

4.1 Experimental setup

We evaluate our proposal on two quaternion-valued image recognition benchmarks taken from [2]. Firstly, we use the standard MNIST dataset by converting every image pixel to a quaternion with 0 imaginary components, i.e. we encode one grey-valued pixel g as $x = g + i0 + j0 + k0$. Secondly, we consider the more challenging CIFAR-10 dataset by converting its RGB representation to the three imaginary components of a pure quaternion with 0 the real part, i.e. we encode a single pixel with RGB values (r, g, b) to $x = 0 + ir + jg + kb$.

Similar to previous literature, for MNIST, we use a quaternion convolutional network with two convolutive layers having 16 and 32 filters, respectively, interleaved by (quaternion-valued) max-pooling operations. After the second convolutive layer we apply a dropout operation for regularisation and a final quaternion fully connected layer for obtaining the class probabilities. For CIFAR-10, we increase this to five convolutional layers having, respectively, 32, 64, 128, 256, and 512 filters. In this case, we also apply dropout every two convolutional layers. Overall, the MNIST network has $\simeq 10k$ parameters, while the CIFAR-10 network has $\simeq 500k$ parameters.

All networks use ReLU applied component-wise as in (5). After the last layer, we take the absolute values of each output to obtain a real-valued score (equivalent to the classical logit value in a real-valued network), and we apply a softmax activation function to convert these to probabilities. The networks are trained to minimise the average cost with a cross-entropy loss function using the Adam optimisation algorithm.

All experiments are implemented in the PyTorch framework extending the QVNN library from [3] [https://github.com/Orkis-Research/Pytorch-Quaternion-Neural-Networks.]. For replicability, we release our demo files on a separate repository online [https://github.com/Riccardo-Vecchi/Pytorch-Quaternion-Neural-Networks.]. All hyper-parameters are fine-tuned independently for each network and dataset using the corresponding validation data. Importantly, this means that all regularisation coefficients are optimised separately for every method.
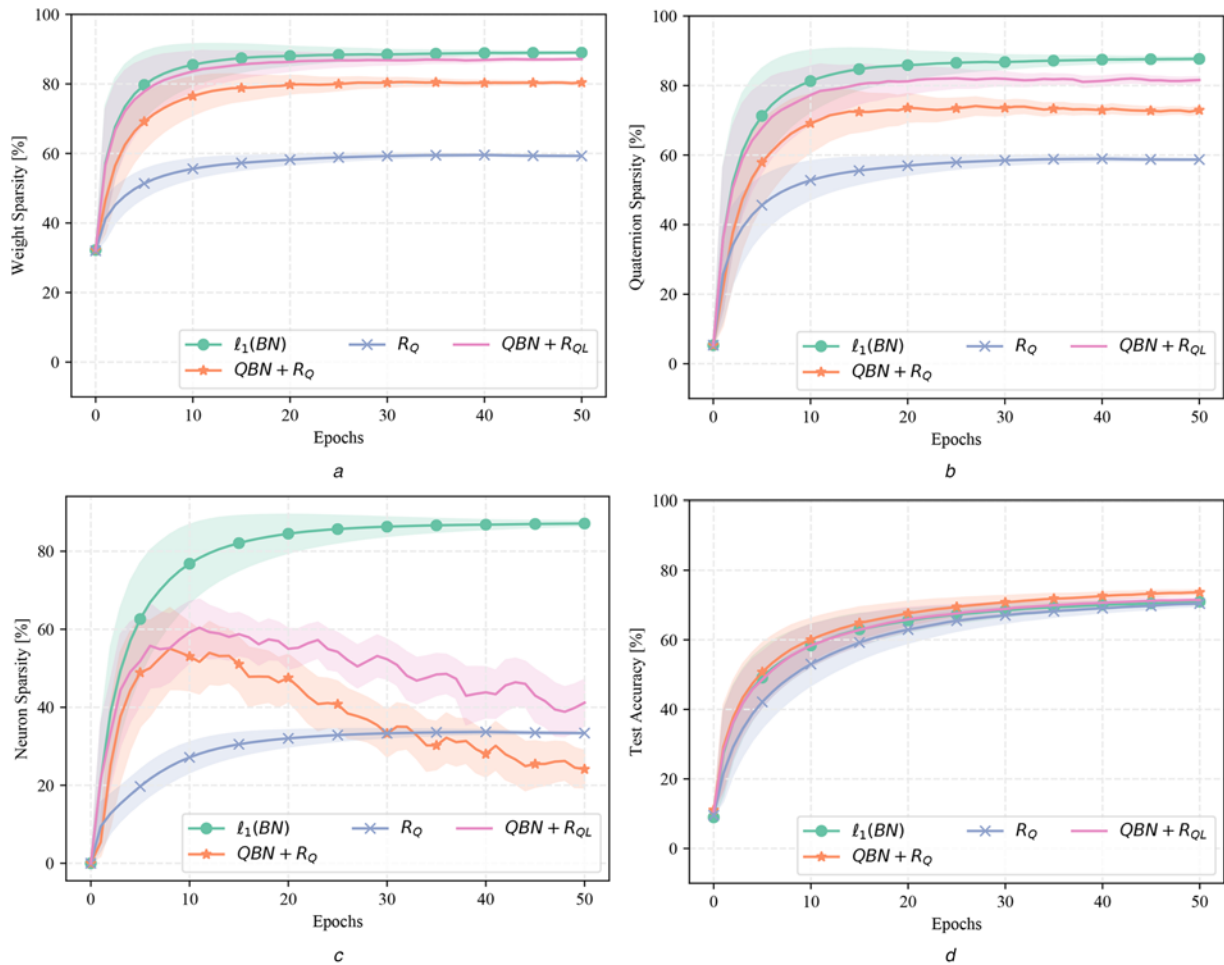
Fig. 2 Evolution of sparsity and accuracy for the CIFAR-10 dataset, zoomed on the first epochs. See the text for a description of the different acronyms
a Weight sparsity
b Quaternion-level sparsity
c Neuron sparsity
d Accuracy

4.2 Results for the quaternion-level sparsity

We start by evaluating the quaternion-level regularisation strategy described in Section 3.1, denoted as R_Q in the experiments. We compared classical ℓ_2 and ℓ_1 regularisations, which are applied independently on every component. In addition, we evaluate a mixed regularisation strategy combining our proposed R_Q method with an additional ℓ_1 regularisation on the components, denoted as R_{QL}, which is similar to the *sparse group sparse* technique in [17]. For this case, we consider a single, shared regularisation factor to be optimised to make comparisons fair.

Results, averaged over 5 different repetitions of the experiments, are presented in Table 1. We see that applying a regularisation has only a marginal effect on accuracy in the MNIST test accuracy, while it improves the accuracy in the more challenging CIFAR-10 case, possibly counter-acting any overfitting effect. In terms of sparsification effects, we show both the component sparsity (i.e. ratio of zero-valued quaternion components) and quaternion sparsity (i.e. ratio of quaternions where all components have been set to 0). We can see that the proposed R_Q strategy results in significantly sparser architectures in both cases, with corresponding gains when considering computational power and inference speed. The mixed strategy R_{QL} performs very well on MNIST and poorer on CIFAR-10, possibly because we are using only a single shared regularisation factor. For a clearer visualisation, in Fig. 1 we show the corresponding sparsity levels during training (for the first 20 epochs of training).

In addition, in Table 2 we report the average training time on different experiments. As expected, adding a regularisation term adds only a small overhead in terms of computational time, while we see no statistical difference between different approaches, further validating the use of a targeted strategy.

4.3 Results for the neuron-level sparsity

Next, we evaluate the inclusion of the neuron-level sparsity strategy described in Section 3.2. We consider the R_Q strategy from the previous section, and compare a network where we add BN layers after every convolutional layer, penalising the γ coefficients with an ℓ_1 strategy. For fairness, we also compared two additional baselines, where we add the BN layers, but regularise only with the R_Q or R_{QL} strategies. For space constraints, we only consider the CIFAR-10 dataset, which was the most challenging in the previous section.

The averaged results are presented in Fig. 2. We see that, when considering only quaternion-sparsity, the proposed neuron-level strategy (denoted as $L_1(\text{BN})$) is marginally superior to the proposed R_{QL} applied on the network having BN layers. However, when evaluating the level of structure in this sparsity, we see that the proposed neuron-level technique in Fig. 2c vastly outperforms all other strategies, leading to a network having $< 17\%$ of neurons than the original one, as long as having $< 15k$ remaining parameters. As a result, the final network has a memory footprint of only 1/5 of the original one, with an inference time speedup of $\sim 3\times$. From Fig. 2d, we also see that this is achieved with no loss in terms of test accuracy of the final networks and, similarly to Table 1, with no significant increase in computational training time.

5 Conclusions

The field of quaternion neural networks explores the extensions of deep learning to handle quaternion-valued data processing. This has shown to be especially promising in the image domain and similar fields with highly structured data that lend itself to representation in a hyper-complex domain. While several models and training algorithms have been already extended to this new challenging domain, less attention has been provided to the tasks of regularising and compressing the networks, which is essential in time-critical and embedded applications.

In this paper, we proposed two regularisation techniques that are specific to quaternion-valued networks. In the first case, we apply some results from quaternion compressive sensing, regularising each quaternion weight with an ℓ_1-level norm. In the second case, we consider the problem of removing entire neurons from the network, by regularising appropriately inserted batch normalisation layers. Our experimental results on two image classification benchmarks show that these two techniques vastly outperform standard regularisation methods when they are applied to quaternion networks, allowing to obtain networks that are extremely smaller (and cheaper to implement) with no loss in testing accuracy at inference time and in computational cost at training time.

For future work, we plan on extending these results to other applications of quaternion-based deep networks (e.g. text processing), as long as evaluating the benefits of sparsifying the networks on custom hardware.

6 References

[1] Basavegowda, H.S., Dagnew, G.: 'Deep learning approach for microarray cancer data classification', *CAAI Trans. Intell. Technol.*, 2020, **5**, (1), pp. 22–33
[2] Zhu, X., Xu, Y., Xu, H., *et al.*: 'Quaternion convolutional neural networks'. Proc. 2018 European Conf. on Computer Vision (ECCV), Munich, Germany, 2018, pp. 631–647
[3] Parcollet, T., Ravanelli, M., Morchid, M., *et al.*: 'Quaternion recurrent neural networks', arXiv preprint arXiv:180604418, 2018
[4] Arena, P., Fortuna, L., Muscato, G., *et al.*: 'Multilayer perceptrons to approximate quaternion valued functions', *Neural Netw.*, 1997, **10**, (2), pp. 335–342
[5] Parcollet, T., Morchid, M., Linarès, G.: 'Quaternion convolutional neural networks for heterogeneous image processing'. Proc. 2019 IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), Brighton, United Kingdom, 2019, pp. 8514–8518
[6] Chen, Y., Xiao, X., Zhou, Y.: 'Low-rank quaternion approximation for color image processing', *IEEE Trans. Image Process.*, 2019, **29**, pp. 1426–1439
[7] Yin, Q., Wang, J., Luo, X., *et al.*: 'Quaternion convolutional neural network for color image classification and forensics', *IEEE Access*, 2019, **7**, pp. 20293–20301
[8] Tay, Y., Zhang, A., Tuan, L.A., *et al.*: 'Lightweight and efficient neural natural language processing with quaternion networks', arXiv preprint arXiv:190604393, 2019
[9] Zhang, S., Tay, Y., Yao, L., *et al.*: 'Quaternion knowledge graph embedding', arXiv preprint arXiv:190410281, 2019
[10] Pavllo, D., Feichtenhofer, C., Auli, M., *et al.*: 'Modeling human motion with quaternion-based neural networks', *Int. J. Comput. Vis.*, 2020, **128**, pp. 855–872
[11] Comminiello, D., Lella, M., Scardapane, S., *et al.*: 'Quaternion convolutional neural networks for detection and localization of 3d sound events'. Proc. 2019 IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), Brighton, United Kingdom, 2019, pp. 8533–8537
[12] Cheng, L., Song, B., Dai, Y., *et al.*: 'Mobile robot indoor dual Kalman filter localisation based on inertial measurement and stereo vision', *CAAI Trans. Intell. Technol.*, 2019, **2**, (4), pp. 173–181
[13] Xu, D., Xia, Y., Mandic, D.P.: 'Optimization in quaternion dynamic systems: gradient, hessian, and learning algorithms', *IEEE Trans. Neural Netw. Learn. Syst.*, 2015, **27**, (2), pp. 249–261
[14] Alippi, C.: 'Selecting accurate, robust, and minimal feedforward neural networks', *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, 2002, **49**, (12), pp. 1799–1810
[15] Koneru, B.N.G., Vasudevan, V.: 'Sparse artificial neural networks using a novel smoothed lasso penalization', *IEEE Trans. Circuits Syst. II, Express Briefs*, 2019, **66**, (5), pp. 848–852
[16] Wen, W., Wu, C., Wang, Y., *et al.*: 'Learning structured sparsity in deep neural networks'. Advances in Neural Information Processing Systems, Barcelona, Spain, 2016, pp. 2074–2082
[17] Scardapane, S., Comminiello, D., Hussain, A., *et al.*: 'Group sparse regularization for deep neural networks', *Neurocomputing*, 2017, **241**, pp. 81–89
[18] Moini, S., Alizadeh, B., Emad, M., *et al.*: 'A resource-limited hardware accelerator for convolutional neural networks in embedded vision applications', *IEEE Trans. Circuits Syst. II, Express Briefs*, 2017, **64**, (10), pp. 1217–1221
[19] Barthélemy, Q., Larue, A., Mars, J.I.: 'Sparse approximations for quaternionic signals', *Adv. Appl. Clifford Algebras*, 2014, **24**, (2), pp. 383–402
[20] Badeńska, A., Błaszczyk, Ł.: 'Compressed sensing in the quaternion algebra', arXiv preprint arXiv:170408202, 2017
[21] Gordon, A., Eban, E., Nachum, O., *et al.*: 'Morphnet: fast & simple resource-constrained structure learning of deep networks'. Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, Utah, USA, 2018, pp. 1586–1595
[22] Ward, J.P.: 'Quaternions and Cayley numbers: algebra and applications' (Springer Science & Business Media, Berlin, Germany, 2012), vol. 403
[23] Scardapane, S., Van Vaerenbergh, S., Hussain, A., *et al.*: 'Complex-valued neural networks with nonparametric activation functions', *IEEE Trans. Emerg. Top. Comput. Intell.*, 2020, **4**, pp. 140–150
[24] Ioffe, S., Szegedy, C.: 'Batch normalization: accelerating deep network training by reducing internal covariate shift', arXiv preprint arXiv:150203167, 2015

CAAI Trans. Intell. Technol., 2020, Vol. 5, Iss. 3, pp. 172–176

176