Deep Cyclic Group Networks

Zhe-Cheng Fan [1,2], Tak-Shing Chan [1], Yi-Hsuan Yang [1], Jyh-Shing Roger Jang [2]
[1]*Research Center for Information Technology Innovation, Academia Sinica*
[2]*Department of Computer Science and Information Engineering, National Taiwan University*
Taipei, Taiwan
{zcfan, takshingchan, yang}@citi.sinica.edu.tw, jang@csie.ntu.edu.tw

***Abstract*—We propose a new network architecture called deep cyclic group network (DCGN) that uses the cyclic group algebra for convolutional vector-neuron learning. The input to DCGN is a three-way tensor, where the mode-3 dimension corresponds to the dimensionality of the input data, e.g., three for RGB images. To handle vector-valued inputs, we replace scalar multiplication with circular convolution for the feedforward and backpropagation processes. As a result, every feature map and kernel map is a three-way tensor with the same mode-3 dimension as the input data. This way, DCGN may capture more of the relations among different data dimensions, especially for regression tasks where the target output has the same dimensionality as the input data. Moreover, DCGN can deal with input data of arbitrary dimensions, a property that existing architectures such as deep complex networks and deep quaternion networks (DQN) lack. Experiments show that DCGN indeed performs better than convolutional neural networks and DQN for two regression tasks, namely color image inpainting and multispectral image denoising.**

***Index Terms*—vector neural learning, deep cyclic group algebra, deep neural networks, convolutional neural networks, deep cyclic group networks**

I. Introduction

Advanced network architectures such as batch normalization [1], highway networks [2], and residual networks [3] have been proposed in recent years to improve the performance or training efficiency of deep learning models. Both highway and residual networks reduce the risk of vanishing gradients via gating-based mechanisms or shortcut paths to regularize information flow. There have also been attempts to make more fundamental changes to convolutional neural networks (CNNs). In particular, vector-valued neural learning in CNNs has received increased attention recently, where the inputs, outputs, weights and biases are all extended from real values to vector values. For instance, deep complex networks (DCN) [4] extend each neuron from a real value to a two-dimensional vector representing complex numbers, and deep quaternion networks (DQN) [5], [6] make each neuron a four-dimensional vector representing the so-called quaternions. Although DCN and DQN show the promise of vector-neuron learning, the dimensionality of their vector-valued inputs are restricted to two or four only. Unfortunately, in EEG-based human emotion recognition [7], the human brain wave is composed of five frequency bands, and for multispectral image denoising [8], the images can comprise of 30 bands or more. Therefore, for these tasks and many others, it is desirable to have a network that can deal with multi-dimensional data with arbitrary dimension N.

Historically, an N-dimensional vector-valued neuron [9]–[11] has previously been proposed to handle N-dimensional vector inputs and outputs, with N-dimensional orthogonal matrix weights. But, this is for a single neuron only and does not support multiple neurons nor backpropagation. A bit more earlier, Clifford algebra has been employed to construct vector-valued neural networks with dimensionality 2^N [12]–[15], but it is not natural to reshape the dimensionality of most datasets into powers-of-two. More recently, matrix-valued neural networks assume that the inputs, outputs, weights, and biases are $N \times N$ square matrices [16]. Yet, again it is not always appropriate to rearrange the inputs as such. More importantly, convolutional architectures of the above works have not been explored at all.

Considering the above, we propose a new CNN architecture named Deep Cyclic Group Network (DCGN), which involves the cyclic group algebra [17]. Our inputs, outputs, kernel maps, and feature maps are three-way tensors [18] with the same mode-3 dimension. The mode-3 dimension corresponds to the prescribed dimensionality N of the input data, where N is an arbitrary positive integer. The proposed architecture is shown in Figure 1. Our contributions are summarized as follows:

- We proposed the DCGN model by using N-dimensional vector-valued neurons with cyclic group algebras, where N is an arbitrary positive integer.
- We derived its backpropagation learning algorithm, weight initialization, and batch normalization.
- We demonstrated the potential of our architecture with image inpainting and denoising experiments.

For reproducible research, we will make the code available at https://github.com/zcfan-tw/vectorNNtoolbox.

II. Proposed Model

In previous work, DCN and DQN extend each neuron and weight to two- and four-dimensions respectively, using complex and quaternion algebras. However, they lack the ability to deal with input data of arbitrary dimensions. The idea of our proposed DCGN is to extend the capacity of each neuron from scalars to vectors with cyclic group algebras. Here, the scalar multiplications in the conventional convolutional procedure are changed into vector multiplications along mode-3 fibers with the help of circular convolutions, which can also be formulated as vector-matrix multiplications with circulant matrices [17],

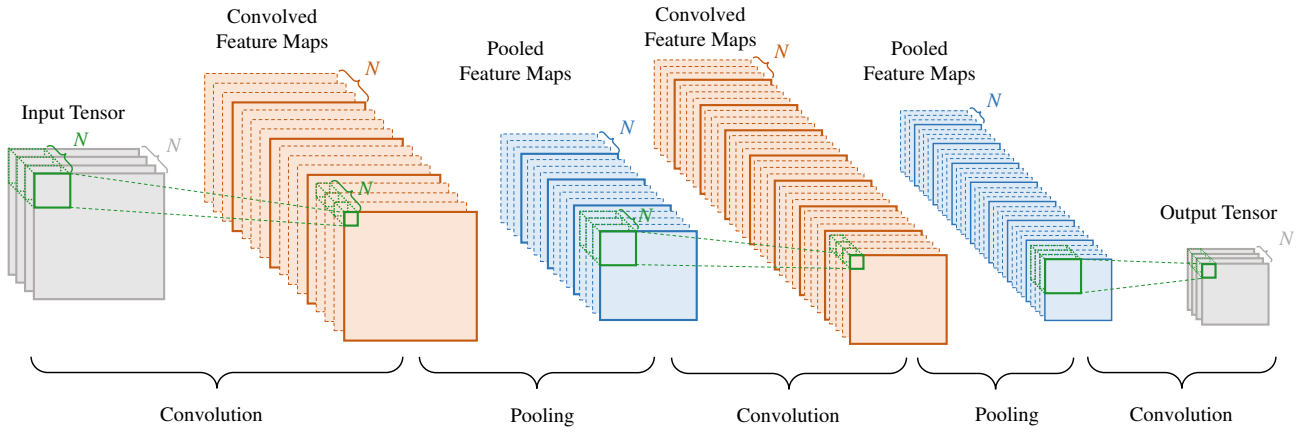


Fig. 1: Illustration of the proposed DCGN model for regression problems (best viewed in color). Orange and blue squares represent the convolved and pooled feature maps, respectively, and green squares represent the kernel maps. All feature maps across all the layers have the same mode-3 dimension (i.e., N). When $N = 1$, the model reduces to the conventional CNNs, and we will not have those dashed squares. The kernel maps perform convolution with scalar multiplications. When $N > 1$, each feature map and kernel map becomes three-way tensors. It is a regression problem since we have tensor input and tensor output. But, the model can be extended to deal with classification problems, by discarding the last convolutional layer and adding fully-connected layers after DCGN. In such a case, DCGN would perform the function of feature learning.

[19]. In other words, our work replaces scalar multiplications with vector multiplications through circulant matrices.

In DCGNs, both the kernel and feature maps are three-way tensors comprising N-dimensional mode-3 fibers. Each fiber in a feature map stands for a N-dimensional vector-valued neuron and its output is also a N-dimensional fiber. The circular convolution is operated between a vector-valued neuron and a fiber coming from a weight kernel map. The convolution procedure is illustrated in Figure 2.

In what follows, we firstly derive the learning algorithm from the point of view of a vector neuron, and then present the application of DCGN for regression and classification problems. Below, matrices are represented by bold uppercase letters, and vectors by bold lowercase letters. Matrix slices of tensors are represented as matrices and vector slices of matrix are represented as vectors. Also, we treat vectors as row vectors.

A. Operation of Circular Convolutions

Given two N-dimensional vectors, $\mathbf{p} = [p_1, p_2, \ldots, p_N] \in \mathbb{R}^N$ and $\mathbf{q} = [q_1, q_2, \ldots, q_N] \in \mathbb{R}^N$, we can compute the circular convolution between them, denoted as $\mathbf{p} * \mathbf{q}$, with vector-matrix multiplication by using the circulant matrix $[\mathbf{q}]_* = \text{circ}(\mathbf{q})$ associated to $\mathbf{q}$ as follows:

$$\mathbf{p} * \mathbf{q} = \mathbf{p}[\mathbf{q}]_* = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ \vdots \\ p_N \end{bmatrix}^T \begin{bmatrix} q_1 & q_2 & q_3 & \cdots & q_N \\ q_N & q_1 & q_2 & \cdots & q_{N-1} \\ q_{N-1} & q_N & q_1 & \cdots & q_{N-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ q_2 & q_3 & q_4 & \cdots & q_1 \end{bmatrix} . \quad (1)$$

We note that $[\mathbf{q}]_* \in \mathbb{R}^{N \times N}$ and $\mathbf{p} * \mathbf{q} \in \mathbb{R}^N$.

B. Feedforward Process for Vector Neurons

In vector neural learning with cyclic group algebras, the inputs, outputs, weights and biases are all N-dimensional vectors. Specifically, for a network with L layers, the relation between the output of a neuron i in layer l ($1 \leq l \leq L$), $\mathbf{a}_i^l \in \mathbb{R}^N$, and the output of a neuron j in the preceding layer $l-1$, $\mathbf{a}_j^{l-1} \in \mathbb{R}^N$, can be written as:

$$\mathbf{a}_i^l \triangleq \phi(\mathbf{z}_i^l) = \phi\left(\sum_{j=1}^{J^{l-1}} \mathbf{w}_{ij}^l * \mathbf{a}_j^{l-1} + \mathbf{b}_i^l\right) , \quad (2)$$

where $\phi(\cdot)$ is a differentiable activation function, $\mathbf{w}_{ij}^l \in \mathbb{R}^N$ the weight vector connecting the two neurons, J^{l-1} the number of neuron in layer $l-1$, and $\mathbf{b}_i^l \in \mathbb{R}^N$ a bias vector. For $l = 1$, we have $J^{l-1} = 1$ and $\mathbf{a}^{l-1} \triangleq \mathbf{x} \in \mathbb{R}^N$, the input data vector of the network. For $l = L$, we have $J^l = 1$, and we define $\mathbf{y} \in \mathbb{R}^N$ as $\mathbf{a}^L$, the output of the network.

C. Backpropagation Learning Algorithm

During the training phase, we are given the target output $\hat{\mathbf{y}} \in \mathbb{R}^N$ for every input data instance, so that we can compare $\hat{\mathbf{y}}$ and $\mathbf{y}$ via some cost function $\epsilon(\mathbf{y}, \hat{\mathbf{y}})$ to know how good the model parameters $\mathbf{\Theta} \triangleq \{\mathbf{w}_{ij}^l, \mathbf{b}_i^l\}$ are. Here, we use $\mathbf{\Theta}$ to denote the collection of all the trainable parameters across the L layers. Given the empirical cost C calculated over all the training data instances, we can use backpropagation to update the model parameters $\mathbf{\Theta}$.

To use backpropagation, we need to take the derivative of C with respect to each parameter $\mathbf{w}_{ij}^l$ or $\mathbf{b}_i^l$. Unlike ordinary CNNs, all the parameters here are vectors. Therefore, in calculating the derivatives we need to compute the Jacobian matrix.

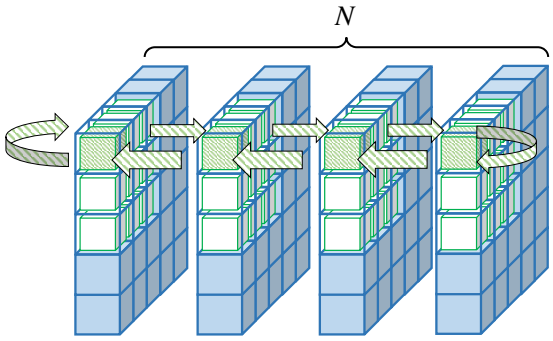


Fig. 2: Illustration of the convolution procedure when using cyclic group algebra (best viewed in color). Each cube represents a scalar. Collectively, the N matrices of cubes represent a three-way tensor of mode-3 dimension N, each matrix is a frontal slice of the tensor, and each mode-3 fiber is an N-dimensional vector. The feature map and the kernel map are composed of blue cubes and green cubes, respectively. In the convolution procedure, each mode-3 fiber of the kernel map performs circular convolution with a mode-3 fiber of the feature map, by rotating itself along the mode-3 dimension, as the green textured cubes represent. Therefore, via the kernel map we can learn the association between the elements across the N dimensions by using a linear kernel and weighted sum. The kernel map moves along the (mode-1, mode-2) plane, each time at a distance of a stride.

For the bias vectors $\mathbf{b}_i^l$, this amounts to calculating the N-dimensional gradient vector, $\frac{\partial C}{\partial \mathbf{b}_i^l} = \left[\frac{\partial C}{\partial b_{i1}^l}, \frac{\partial C}{\partial b_{i2}^l}, \ldots, \frac{\partial C}{\partial b_{iN}^l}\right]$, where $\frac{\partial C}{\partial b_{ik}^l} = \sum_{n=1}^N \frac{\partial C}{\partial z_{in}^l}\frac{\partial z_{in}^l}{\partial b_{ik}^l}$. It can be shown that $\frac{\partial z_{in}^l}{\partial b_{ik}^l} = 1$ only when $n = k$, otherwise it is zero. It will become apparent later that it is more convenient to denote $\frac{\partial C}{\partial z_{in}^l}$ simply as d_{in}^l and accordingly define the *local gradient vector* $\mathbf{d}_i^l = \left[d_{i1}^l, d_{i2}^l, \ldots, d_{iN}^l\right]$ as follows:

$$\mathbf{d}_i^l \triangleq \frac{\partial C}{\partial \mathbf{z}_i^l} = \left[\frac{\partial C}{\partial z_{i1}^l}, \frac{\partial C}{\partial z_{i2}^l}, \ldots, \frac{\partial C}{\partial z_{iN}^l}\right]. \tag{3}$$

It can be shown that $\frac{\partial C}{\partial b_{ik}^l} = d_{ik}^l$ and accordingly,

$$\frac{\partial C}{\partial \mathbf{b}_i^l} = \left[\ldots, \sum_{n=1}^N \frac{\partial C}{\partial z_{in}^l}\frac{\partial z_{in}^l}{\partial b_{ik}^l}, \ldots\right] = \mathbf{d}_i^l. \tag{4}$$

We can calculate the derivative of C w.r.t. the weight vector $\mathbf{w}_{ij}^l$ similarly. With some algebra, it can be shown that

$$\frac{\partial C}{\partial \mathbf{w}_{ij}^l} = \left[\ldots, \sum_{n=1}^N \frac{\partial C}{\partial z_{in}^l}\frac{\partial z_{in}^l}{\partial w_{ijk}^l}, \ldots\right] = \mathbf{d}_i^l \frac{\partial \mathbf{z}_i^l}{\partial \mathbf{w}_{ij}^l} = \mathbf{d}_i^l [\mathbf{a}_j^{l-1}]_*^T. \tag{5}$$

Algorithm 1 Backpropagation Algorithm for Updating Deep Cyclic Group Networks

Input: Training inputs $\mathbf{x}$ and related targets $\hat{\mathbf{y}}$; activation function ϕ; cost function ϵ

Output: Parameters $\mathbf{\Theta}$ of weights and biases

```
while not converged do
    for each minibatch do
        Perform feedforward process with Eq. (2) to get z
        Perform backpropagation with Eqs. (7) and (9) to get
        local gradient vector d per neuron per layer
        Update biases b with Eq. (4)
        Update weights w with Eq. (5)
    end for
end while
```

From Eq. (2), we see that the last equality holds because

$$\frac{\partial \mathbf{z}_i^l}{\partial \mathbf{w}_{ij}^l} = \begin{bmatrix} a_{j1}^{l-1} & a_{jN}^{l-1} & \ldots & a_{j2}^{l-1} \\ a_{j2}^{l-1} & a_{j1}^{l-1} & \ldots & a_{j3}^{l-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{jN}^{l-1} & a_{j(N-1)}^{l-1} & \ldots & a_{j1}^{l-1} \end{bmatrix} = [\mathbf{a}_j^{l-1}]_*^T, \tag{6}$$

where $[\mathbf{a}_j^{l-1}]_*$ is the circulant matrix associated to $\mathbf{a}_j^{l-1}$.

We now show how to calculate $\mathbf{d}_i^l$, which requires backpropagation from the upper layer $l+1$ and is therefore more complicated. According to the chain rule, we have

$$\mathbf{d}_i^l \triangleq \frac{\partial C}{\partial \mathbf{z}_i^l} = \sum_{j=1}^{J^{l+1}} \frac{\partial C}{\partial \mathbf{z}_j^{l+1}}\frac{\partial \mathbf{z}_j^{l+1}}{\partial \mathbf{a}_i^l}\frac{\partial \mathbf{a}_i^l}{\partial \mathbf{z}_i^l}, \tag{7}$$

which involves vector-matrix-matrix multiplications. From Eq. (4), we see that the first term in the summation is equal to $\mathbf{d}_j^{l+1}$. From Eq. (2), and with some algebra, it can be shown that the middle term is equal to $[\mathbf{w}_{ji}^{l+1}]_* \in \mathbb{R}^{N\times N}$, and the last term is equal to the following $N \times N$ diagonal matrix:

$$\Phi(\mathbf{z}_i^l) \triangleq \frac{\partial \mathbf{a}_i^l}{\partial \mathbf{z}_i^l} = \text{diag}\left(\left[\dot{\phi}(z_{i1}^l),\ \dot{\phi}(z_{i1}^2),\ \ldots,\ \dot{\phi}(z_{i1}^N)\right]\right), \tag{8}$$

where $\dot{\phi}(\cdot)$ denotes the derivative of the activation function. From the above description, we see that the update of $\mathbf{d}_i^l$ depends on $\mathbf{d}_j^{l+1}$ and we can do this layer-by-layer in the backward direction starting from the last layer. In the output layer L, the local gradient vector $\mathbf{d}^L$ can be computed by:

$$\mathbf{d}^L \triangleq \frac{\partial C}{\partial \mathbf{z}^L} = \frac{\partial C}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{z}^L} = \frac{\partial C}{\partial \mathbf{y}}\frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L} = \frac{\partial C}{\partial \mathbf{y}}\Phi(\mathbf{z}^L), \tag{9}$$

where the last term involves computing the derivative of the cost function $\epsilon(\cdot,\cdot)$.

We summarize the aforementioned process in Algorithm 1. We note that the proposed learning algorithm can be applied to vector neurons with arbitrary dimensions N. If the complexity of CNN is $\mathcal{O}(C)$, then the complexity of DCGN is $\mathcal{O}(NC)$. In other words, the time complexity of DCGN is N times the

complexity of CNN. However, if we view N as a constant, the complexity of DCGN and CNN becomes the same.

D. Weight Initialization

Proper weight initialization can reduce the risk of gradient vanishing or exploding when the architecture of neural networks is deep. Below, we propose a weight initialization method for DCGN by extending the method proposed by [20] and [21] to DCGNs.

The main idea is to make the output variance equal to the input variance for each neuron. In doing so, we need to estimate the variance of each kernel map $\mathbf{w}$, which can be calculated as [5], [6]:

$$\mathrm{Var}(\mathbf{w}) = \mathbb{E}\left[|\mathbf{w}|^2\right] . \tag{10}$$

However, such an estimation is ill-defined for DCGN, since $\mathbf{w}$ is a vector not a scalar. That said, if we further assume w_1, w_2, …, $w_N \sim \mathcal{N}\left(0, \sigma^2\right)$ i.i.d., then $\mathbb{E}\left[|\mathbf{w}|^2\right] = \mathbb{E}\left[w_1^2 + w_2^2 + \ldots + w_N^2\right] = \mathbb{E}\left[w_1^2\right] + \mathbb{E}\left[w_2^2\right] + \ldots + \mathbb{E}\left[w_N^2\right] = N\sigma^2$. Thus we can rewrite Eq. (10) as:

$$\mathrm{Var}(\mathbf{w}) = N\sigma^2, \tag{11}$$

where σ is an estimated parameter of the variance of $\mathbf{w}$. Then, if we use the initialization method proposed by [20], we have $\mathrm{Var}(\mathbf{w}) = 2/\left(n_{in} + n_{out}\right)$, where n_{in} and n_{out} denote the number of input and output units of that neuron, respectively. Accordingly, we have $\sigma^2 = \frac{2}{N(n_{in}+n_{out})}$. On the other hand, if we use the initialization method proposed by [21], which is designed for networks that use rectified linear units (ReLUs) [22] as the activation function $\phi(\cdot)$, we would have $\mathrm{Var}(\mathbf{w}) = 2/n_{in}$, and $\sigma^2 = \frac{2}{N\, n_{in}}$.

E. Batch Normalization

Batch normalization [1] is an important technique to stabilize and speed up the training process. The idea is to keep the input of each layer having zero mean and unit variance. The original version is designed for scalar neurons. While [5] have extended the method for DQNs, we present below how to extend it to general vector neurons with arbitrary dimension N. To ensure equal variance in the vector parts, we calculate the variance not only between different training data instances but also between the N dimensions, leading to the following covariance matrix $\mathbf{V}$:

$$\mathbf{V} = \left[\mathrm{Cov}(u_i, v_j)\right]_{\ i=1,\ldots,N\ ,\ j=1,\ldots,N}\ , \tag{12}$$

where $\mathrm{Cov}(\cdot)$ is the covariance function and u_i, v_j denotes entries of two input vectors $\mathbf{u}$ and $\mathbf{v}$ for that layer. Following [5], we use Cholesky decomposition on the covariance matrix to learn the shift vector $\beta \in \mathbb{R}^N$ and the transformation matrix $\mathbf{\Gamma} \in \mathbb{R}^{N\times N}$. If the input vector $\tilde{\mathbf{x}}$ has been normalized to mean 0 and variance 1, then batch normalization is done by

$$\mathrm{BN}(\tilde{\mathbf{x}}) = \mathbf{\Gamma}\tilde{\mathbf{x}} + \beta\,. \tag{13}$$

We note that $\mathbf{\Gamma}$ is a symmetric matrix, and that its diagonal can be initialized to $1/\sqrt{N}$ to obtain modulus of 1 for the variance of the normalized value. The other terms of $\mathbf{\Gamma}$ and all entries of β are initialized to 0.

F. Pooling Layers

As shown in Figure 1, we can optionally use pooling layers after the convolutional layer. A pooling layer is another type of layers simplifying or summarizing the information from convolved feature maps. For max pooling, we propose to calculate the magnitude of each mode-3 fiber in the feature maps, and preserve the one (i.e., a vector) with the largest magnitude. For mean pooling, we calculate the average of each element among the mode-3 fibers to generate new three-way feature maps.

G. Applications to Regression Problems

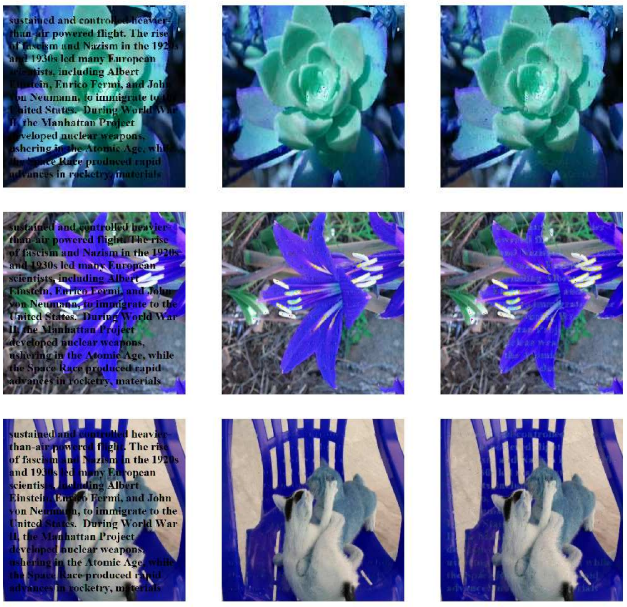
DCGN can be applied to regression problems where we need to learn a nonlinear mapping between tensor inputs and tensor outputs. The inputs and outputs have the same mode-3 dimension, but not necessarily the same mode-1 and mode-2 dimensions. To fit the dimensionality of the output tensor, we need to use only one kernel map for the last convolutional layer of DCGN. For the activation function $\phi(\cdot)$, we can use rectified linear unit (ReLU) or leaky ReLU for all the layers except for the last layer, where we may want to use the sigmoid or tanh function. We can use squared error for the cost function $\epsilon(\mathbf{A}, \mathbf{B}) = \|\mathbf{A} - \mathbf{B}\|_2^2$, where $\mathbf{A}$ and $\mathbf{B}$ denote the groundtruth target and the output tensors of DCGN, respectively. Such a network is shown in Figure 1.

H. Foreseen Applications to Classification Problems

When DCGN is applied to classification problems, the input is a three-way tensor and the corresponding output is a vector for one-hot representation (i.e., the class labels). We can also use only one kernel map for the last convolutional layer, but this is not mandatory. Moreover, we would add fully-connected layers to the end of DCGN for learning the classifier. We usually use the sigmoid function as the activation function for the last layer of the fully-connected layers, so that we can use the cross entropy as the cost function: $\epsilon(\mathbf{a}, \mathbf{b}) = -\sum[a_i \ln b_i + (1 - a_i)\ln(1 - b_i)]$, where $\mathbf{a}$ and $\mathbf{b}$ are the groundtruth label vector and the predicted one, respectively. In such a case, DCGN performs the function of feature learning.

III. Experiments and Results

To evaluate the effectiveness of DCGN, we test it on two regression problems. The first one is *color image inpainting*, which aims to reconstruct the deteriorated or lost parts of the original color image. The second one is *multispectral image denoising*, which aims to recover the original multiband image from a noisy version. For the first problem, the input is a tensor with $N = 3$, the RGB channels. For the second problem, since a multispectral image is composed of multiple greyscale images shooting at different wavelengths (or bands), N is equal to the number of bands. We note that DCGN can deal with multispectral images of arbitrary bands,



(a) Noisy (b) DCGN (c) CNN_2

Fig. 3: Examples of three contaminated images, and the result of DCGN and a conventional CNN model (CNN_2) for image inpainting. DCGN performs better than CNN_2 in removing the imposed text, as also shown in Table I.

but DQN can only deal with 4-band multispectral images each time. Therefore, we intend to vary the value of N from 3 to 6 for the second experiment.

As baseline methods, we consider the conventional CNNs and the DQN model proposed by [5], using the code they shared. The goal is to verify the possible advantages of using the vector neurons over the scalar neurons, instead of pursuing high accuracy. Therefore, we opt for simpler model design and aim to compare the models fairly. For all the baseline models and our model, we use only three fully convolutional layers (without any pooling layers) and employ kernel maps with size 3×3. We used ReLU as the activation function, mean squared error (MSE) as the cost function, and Adam [23] for gradient optimization.

For conventional CNNs, we consider three variants of CNN with different number of kernels per layer. CNN_1 has the same number of kernels per layer (i.e., 24) as DCGN and DQN; CNN_2 has more kernels but the total number of parameters is similar to DCGN, and CNN_3 has even more kernels and roughly three times the total number of parameters as DCGN. DCGN and DQN have more parameters than CNN_1 because their kernel maps are tensors.

The evaluation metrics for both experiments are peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) [24]. SSIM is a measure of the similarity between two images (i.e., the groundtruth clean one and the recovered one) and its range is $[0, 1]$. Both PSNR and SSIM are the higher the better. We calculate them for each output result and report

TABLE I: THE PSNR (IN DB) AND SSIM OBTAINED BY DIFFERENT METHODS FOR COLOR IMAGE INPAINTING. 'KERNELS' STANDS FOR THE NUMBER OF NEURONS PER LAYER, AND 'PARMS' THE TOTAL NUMBER OF PARAMETERS. DQN IS BASED ON [5].

Method	N	Kernels	Param.	PSNR	SSIM
Input noise	—	—	—	14.23	0.65
CNN_1	3	24	11.7K	29.50	0.90
CNN_2	3	41	32.6K	30.11	0.91
CNN_3	3	72	97.4K	30.86	0.93
DCGN	3	24	32.4K	**31.72**	0.94
DQN [5]	4	24	43.9K	31.62	0.94

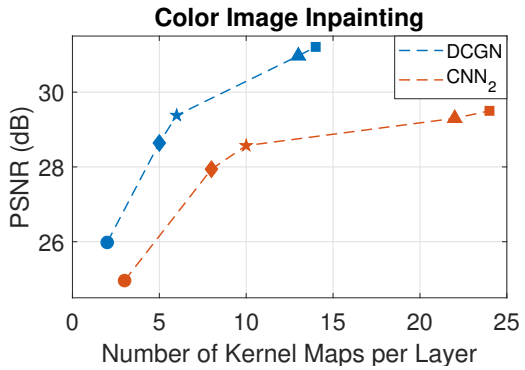


Fig. 4: The PSNR (in dB) of DCGN and CNN for image inpainting, using different number of kernel maps per layer. We use the same mark to indicate the cases where DCGN and CNN have the same total number of parameters.

the average values. All the experiments are performed on a personal computer equipped with an NVIDIA GeForce GTX 1080Ti GPU and 64GB RAM. Our code is written in Python.

A. Color Image Inpainting

Image inpainting is a task that aims to recover corrupted or missing pixels in an image [25]–[28]. Figure 3(a) shows three example images that need to be inpainted. We have to remove the imposed text and fill pixel values that are consistent with the surrounding context.

We use the images from the LabelMe dataset [29] for this evaluation. The dataset is split into two non-overlapping sets: the training set contains 2,920 images, and the test set 1,133 images. To prevent overfitting, we randomly select from the training set 150 images as the validation set. To reduce the training time, we resize all the images from 2,592×1,944 to 512×512. We contaminate all the images by the same text with variant fonts [30], using the contaminated ones as the model input and the clean ones as the target output. For all the models, we set the maximal number of training epochs to 100, and stop the training process when the validation MSE does not decrease for consecutive 10 epochs. Since the color

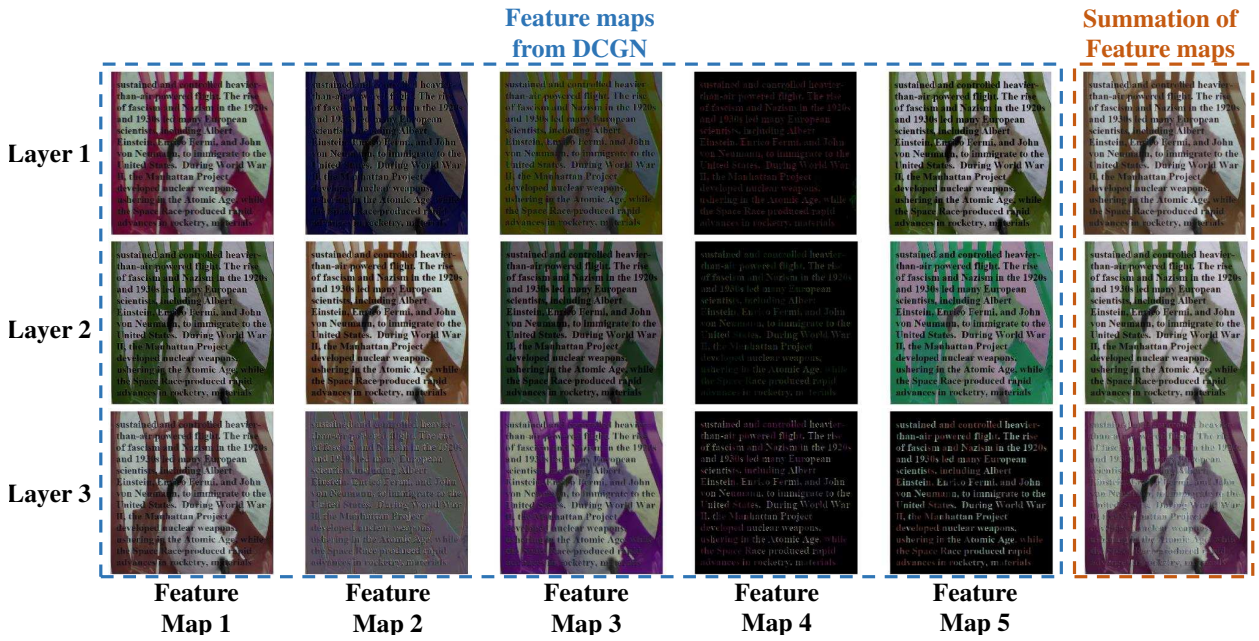


Fig. 5: Illustration of what DCGN learns. For this illustration, we use five kernels for each of the three layers in DCGN. We show the output of each neuron on the left hand side, and the summation of them on the rightmost column. We can see that DCGN learns to separate the image from the occluding text, and that the summation of the feature maps becomes more and more noise-free layer by layer. We note that the final output of DCGN would actually be a weighted sum of the feature maps.

images only have three dimensions (i.e., RGB), we follow the approach of [5] and set the fourth dimension to all zeros for DQN to provide a fair comparison with the quaternion algebra.

Table I shows that DCGN and DQN perform generally better than the conventional CNNs, suggesting the benefit of vector neuron learning. While the results of DCGN and DQN are comparable, the result of DCGN is significantly better ($p-$value < 0.01) than even CNN_3 under the paired t-test for both metrics. Moreover, we note that DCGN has fewer total parameters than DQN, yet DCGN obtains the highest mean PSNR 31.72 dB.

Some inpainting results of DCGN and CNN2 are shown in Figures 3. Both methods can remove the imposed text, but DCGN performs slightly better. Figure 4 further shows that the result of both DCGN and CNN increases as a function of the number of kernels per layer, but DCGN consistently outperforms CNN.

Finally, we show in Figure 5 what DCGN might learn for the inpainting task. For this investigation, we use a three-layer DCGN with only five neurons per layer for simplicity. Figure 5 shows the output (i.e., feature map) of each neuron for a given image, as well as the direct sum of these outputs per layer. Because the feature maps are all tensors with $N = 3$, they can be viewed as color images. We see that DCGN learns to separate the original image from the imposed text, and that the summation of the feature maps becomes more and more noise-free layer by layer.

B. Multispectral Image Denoising

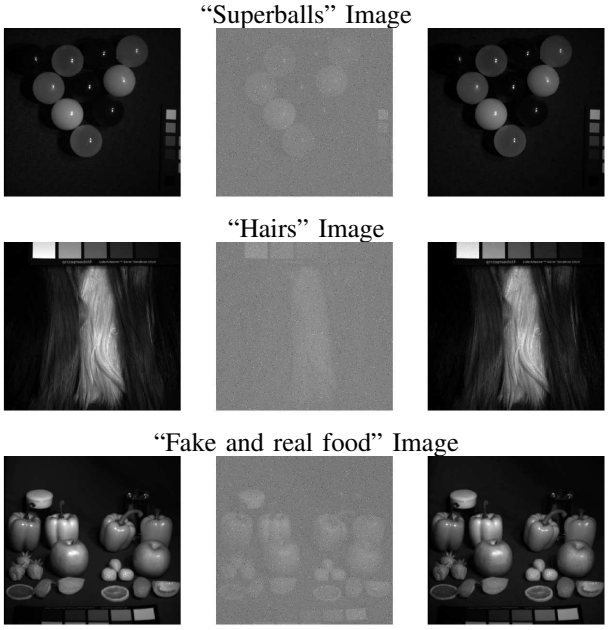
A multispectral (or hyperspectral) image is composed of a collection of monospectral (greyscale) images, each captured with a specific wavelength [31]. These monospectral images can be thought of as in different *bands* of the multispectral image. Since some associations may exist among different bands, we can improve the performance of image processing applications by leveraging such associations [32].

In multispectral image denoising [8], [33], we are given a noisy version of a multispectral image with N bands, and are asked to restore the clean version (also N bands). We use the Columbia multispectral image database [34] for this evaluation. It contains 32 real-world scenes and each scene is made up of 31 monochrome images of size 512×512, captured by varying the wavelength of a camera from 400nm to 700nm with a step size of 10nm. We randomly select 16 scenes for the training set, and the rest for the test set. We pick images of the last 3–6 bands to make $N \in [3, 6]$. Moreover, we divide each image into $32\times32\times N$ overlapping tensor patches with hop size being one. Then, we randomly select approximate 20,000 tensor patches as the training data, 1,000 of which are held out as the validation data to prevent overfitting. For each tensor patch, all the N bands are processed at the same time. We set the maximal number of training epochs to 30 and stop the training process if the validation MSE does not decrease for 5 consecutive epochs.

To create a noisy image, we randomly pick a certain ratio of pixels from each band to add Gaussian noise with

TABLE II: PSNR (IN DB) AND SSIM OBTAINED BY DIFFERENT METHODS FOR MULTISPECTRAL IMAGE DENOISING UNDER DIFFERENT SPARSITY (IN %) AND SIGMA VALUES. WE CAN USE DIFFERENT NUMBER OF BANDS FOR DENOISING, LEADING TO DIFFERENT DATA DIMENSIONS N.

Method	N	**Kernels**	**Param.**	**(5%, 100)**		**(10%, 100)**		**(15%, 100)**		**(10%, 150)**		**(10%, 200)**	
				PSNR	**SSIM**	**PSNR**	**SSIM**	**PSNR**	**SSIM**	**PSNR**	**SSIM**	**PSNR**	**SSIM**
Input noise	—	—	—	21.23	0.72	18.35	0.59	16.68	0.50	14.81	0.53	12.30	0.52
CNN_1	4	24	12.2K	42.07	0.97	40.62	0.95	39.06	0.94	38.97	0.95	39.24	0.94
CNN_2	4	48	45.1K	45.07	0.98	42.18	0.97	41.02	0.96	42.07	0.97	42.01	0.97
CNN_3	4	83	130.2K	43.82	0.97	41.71	0.97	41.57	0.97	43.04	0.97	41.49	0.96
DCGN ($N = 3$)	3	24	32.4K	48.24	0.99	46.00	0.98	45.02	0.98	46.97	0.98	44.93	0.98
DCGN ($N = 4$)	4	24	43.3K	48.61	0.99	46.58	0.98	45.42	0.98	46.02	0.98	43.62	0.97
DCGN ($N = 5$)	5	24	54.0K	48.96	0.99	46.97	0.98	45.55	0.98	44.60	0.98	45.03	0.98
DCGN ($N = 6$)	6	24	64.8K	50.26	0.99	47.19	0.98	45.63	0.98	45.45	0.98	46.31	0.98
DQN [5]	4	24	43.5K	45.15	0.98	43.20	0.97	42.29	0.97	42.61	0.97	41.57	0.97



(a) Original (b) Noisy (c) Denoised

Fig. 6: Examples of three noisy images (at 700nm band) and the multispectral image denoising result by DCGN with $N = 4$. The sparsity of the noisy pixels is 10% and the sigma value of the additive Gaussian noise is set to 200.

specific *sigma* value. We refer to this ratio as the *sparsity* below. We vary the sparsity from 5% to 15% and sigma from 100 to 200 to simulate different degrees of corruption. In model training, the MSE is computed between the restored and the original versions of the patches across all the N bands. We vary N from 3 to 6 for DCGN, and set $N = 4$ for CNNs and DQN.

Table II shows the result of multispectral image denoising. The following observations can be made. First, comparing the result of the two CNNs and DCGN ($N = 4$), we see again DCGN outperforms CNNs for a regression task. The performance difference between DCGN and CNN_2 is significant (p-value$<$0.01) under the t-test for the first four evaluation scenarios. The fifth evaluation scenario (i.e., 10% sparsity with a sigma of 200) is the most noisy scenario but DCGN still performs better. Figure 6 demonstrates the denoised result of DCGN.

Second, Table II indicates that the performance of DQN [5] is also much better than CNN_1. However, it is only comparable to CNN_2, which uses more kernels per layer but the same overall total number of parameters. Comparing the result of DCGN ($N = 4$) and DQN, it seems DCGN is more effective, possibly owing to the use of cyclic group algebras.

Finally, comparing the result of different DCGN models in Table II, we see that the performance in PSNR improves as a function of N, except for the two noisier scenarios. This nicely demonstrates the importance of having a vector neural learning model that can accommodate arbitrary data dimensions. The dependency across different data dimensions may increase when we have more dimensions, and DCGN can exploit such dependency to improve the result of a machine learning task.

IV. CONCLUSION

In this paper, we have presented a new type of CNN architecture consisting of N-dimensional vector-valued neurons endowed with the cyclic group algebra, where N can be an arbitrary positive integer. In the convolution procedure, scalar multiplications in conventional CNNs are changed to vector multiplications through circulant matrices. The learning process is performed with a backpropagation algorithm. Experimental results on color image inpainting and multispectral denoising show that the proposed model exhibit better performance than conventional CNNs and deep quaternion networks. In the future, we would like to test it on classification problems, and datasets with even larger dimensions.

REFERENCES

[1] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[2] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Proc. Advances in Neural Information Processing Systems*, 2015, pp. 2377–2385.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Computer Vision Pattern Recognition*, 2016, pp. 770–778.

[4] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal, "Deep complex networks," *Proc. Int. Conf. Learning Representations*, 2017.

[5] C. Gaudet and A. Maida, "Deep quaternion networks," *arXiv preprint arXiv:1712.04604*, 2017, [Online]. Source code available at: https://github.com/gaudetcj/DeepQuaternionNetworks.

[6] T. Parcollet, Y. Zhang, M. Morchid, C. Trabelsi, G. Linarès, R. De Mori, and Y. Bengio, "Quaternion convolutional neural networks for end-to-end automatic speech recognition," *arXiv preprint arXiv:1806.07789*, 2018.

[7] S. Jirayucharoensak, S. Pan-Ngum, and P. Israsena, "EEG-based emotion recognition using deep learning network with principal component based covariate shift adaptation," *The Scientific World J.*, vol. 2014, 2014.

[8] Z. Zhang and S. Aeron, "Denoising and completion of 3D data via multidimensional dictionary learning," in *Proc. Int. Joint Conf. Artificial Intelligence*, 2016, pp. 2371–2377.

[9] T. Nitta, "N-dimensional vector neuron," in *Proc. Int. Joint Conf. Artificial Intelligence (IJCAI)*, 2007, pp. 2–7.

[10] ——, "N-dimensional vector neuron and its application to the N-bit parity problem," in *Complex-valued neural networks: Advances and applications*. New York: John Wiley and Sons, 2013, pp. 59–74.

[11] B. K. Tripathi, *High Dimensional Neurocomputing*. India: Springer, New Delhi, 2015.

[12] J. K. Pearson and D. L. Bisset, "Back propagation in a Clifford algebra." in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 2, 1992, pp. 413–416.

[13] ——, "Neural networks in the Clifford domain," in *Proc. IEEE Int. Conf. Neural Networks*, vol. 3, 1994, pp. 1465–1469.

[14] E. J. Bayro-Corrochano, "Geometric neural computing," *IEEE Trans. Neural Networks*, vol. 12, no. 5, pp. 968–986, 2001.

[15] S. Buchholz and G. Sommer, "Clifford algebra multilayer perceptrons," in *Geometric Computing with Clifford Algebras: Theoretical Foundations and Applications in Computer Vision and Robotics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 315–334.

[16] C.-A. Popa, "Matrix-valued neural networks," in *Mendel 2015*. Cham: Springer International Publishing, 2015, pp. 245–255.

[17] D. R. Farenick, *Algebras of Linear Transformations*. Springer-Verlag, 2001.

[18] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[19] P. J. Davis, *Circulant matrices*. American Mathematical Soc., 2012.

[20] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. Int. Conf. Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proc. IEEE Int. Conf. Computer Vision*, 2015, pp. 1026–1034.

[22] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. Int. Conf. Machine Learning*, 2010, pp. 807–814.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[25] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, "Image inpainting," in *Proc. Annual Conf. Computer Graphics and Interactive Techniques*, 2000, pp. 417–424.

[26] J. Xie, L. Xu, and E. Chen, "Image denoising and inpainting with deep neural networks," in *Proc. Advances in Neural Information Processing Systems*, 2012, pp. 341–349.

[27] C. Guillemot and O. L. Meur, "Image inpainting: Overview and recent advances," *IEEE Signal Processing Magazine*, vol. 31, no. 1, pp. 127–144, 2014.

[28] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Globally and locally consistent image completion," *ACM Trans. Graphics*, vol. 36, no. 4, p. 107, 2017.

[29] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, "Labelme: a database and web-based tool for image annotation," *Int. J. Computer Vision*, vol. 77, no. 1-3, pp. 157–173, 2008.

[30] J. S. Ren, L. Xu, Q. Yan, and W. Sun, "Shepard convolutional neural networks," in *Proc. Advances in Neural Information Processing Systems*, 2015, pp. 901–909.

[31] P. D. Burns and R. S. Berns, "Analysis multispectral image capture," in *Proc. Color and Imaging Conf.*, 1996, pp. 19–22.

[32] Y. Peng, D. Meng, Z. Xu, C. Gao, Y. Yang, and B. Zhang, "Decomposable nonlocal tensor dictionary learning for multispectral image denoising," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2014, pp. 2949–2956.

[33] W. Xie and Y. Li, "Hyperspectral imagery denoising by deep learning with trainable nonlinearity function," *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 11, pp. 1963–1967, 2017.

[34] F. Yasuma, T. Mitsunaga, D. Iso, and S. K. Nayar, "Generalized assorted pixel camera: postcapture control of resolution, dynamic range, and spectrum," *IEEE Trans. Image Processing*, vol. 19, no. 9, pp. 2241–2253, 2010, [Online]. Dataset available at: http://www1.cs.columbia.edu/CAVE/databases/multispectral/.