Received July 2, 2014, accepted August 2, 2014, date of publication August 26, 2014, date of current version September 10, 2014.

Digital Object Identifier 10.1109/ACCESS.2014.2351832

Neural Implementation of Shape-Invariant Touch Counter Based on Euler Calculus

KEIJI MIURA[1] AND KAZUKI NAKADA[2], (Member, IEEE)
[1]Graduate School of Information Sciences, Tohoku University, Sendai 980-8579, Japan
[2]Graduate School of Informatics and Engineering, University of Electro-Communications, Tokyo 182-8585, Japan

Corresponding author: K. Miura (miura@ecei.tohoku.ac.jp)

This work was supported in part by a Grant-in-Aid for Scientific Research on Innovative Areas, in part by the Neural Creativity for Communication under Grant 4103, in part by the Ministry of Education, Culture, Sports, Science, and Technology, Japan, under Grant 24120701, and in part by a Grants-in-Aid for Scientific Research, Japan Society for the Promotion of Science, under Grant 24700301 and Grant 25820163.

ABSTRACT One of the goals of neuromorphic engineering is to imitate the brain's ability to recognize and count the number of individual objects as entities based on the global consistency of the information from the population of activated tactile (or visual) sensory neurons whatever the objects' shapes are. To achieve this flexibility, it may be worth examining an unconventional algorithm such as topological methods. Here, we propose a fully parallelized algorithm for a shape-invariant touch counter for 2-D pixels. The number of touches is counted by the Euler integral, a generalized integral, in which a connected component counter (Betti number) for the binary image was used as elemental module. Through examples of touches, we demonstrate transparently how the proposed circuit architecture embodies the Euler integral in the form of recurrent neural networks for iterative vector operations. Our parallelization can lead the way to Field-Programmable Gate Array or Digital Signal Processor implementations of topological algorithms with scalability to high resolutions of pixels.

INDEX TERMS Euler calculus, topology, invariance, touch counter, neuromorphic engineering, sensor networks.

I. INTRODUCTION

It has been desired for a long time to design the machine that achieves brains' ability to recognize unity from the global consistency of sensory stimuli [1], [2]. For example, there is a neuron which can count the number of objects in the visual field irrespectively of their shapes [3]–[5]. In the case of tactile system, it can be important to count the number of touches irrespective of their shapes [6].

Conventionally, perceptrons or similar layered networks have been used as a model of pattern recognition in the brain [7]–[10]. Although perceptron-like machines show invariance to, say, object rotations to some extent, their analogue nature of computation makes it difficult to achieve the invariance with the infinite precision, especially in counting [11], [12]. Therefore, it is worth examining an alternative algorithm, which, hopefully, realizes concerted inter-neuronal communications with the perfect invariance guaranteed.

To compute the topological invariants such as the number of distinct objects in an image, the algorithms based on topology, a major area of mathematics [13]–[15], can be more effective than *ad hoc* algorithms. Fortunately, recent advances of the field of computational topology made it possible to compute topological invariants in an accessible way [16]–[21].

Although a couple of latest works by mathematicians tackled the fundamental problems of shape-invariant number count specifically [22]–[25], their implementations on serial system computers can, in principle, suffer from the formidable computational time for the big data with many sensors or pixels. Although there are some works on distributed [25] or gradient-based decentralized algorithms [26], the concrete computations have been actually performed on serial system computers. Therefore it is very important to design a fully parallelized circuit implementation for computing topology. In addition, if the parallelization is biologically plausible in the form of neural networks, which we call "neural implementation," it could give insight into neural information processing as a brain model, as it readily allows to compare its behaviors to those of biological neurons [1].

In this paper, we propose an algorithm of a fully parallelized neural network that can count the number of touches on a two dimensional square lattice of sensors. Based on the Euler calculus, the proposed counter can count the touch number irrespectively of the shapes and positions of the touches. The count is not approximate, but accurate, guaranteed by mathematics. The key idea is to transform the touch number on a 2D plane to the number of connected components of level sets there. Then we implement and demonstrate the connected component counter, which is a key elemental module, in a fully parallelized circuit for iterative vector calculations. Through concrete examples of simple but nontrivial touches, we demonstrate transparently how the proposed circuit architecture embodies the Euler integral.

The background mathematics here is rather advanced, but we try to keep the paper as accessible as possible by engineers and designers. One of the goal of this paper is to import the state-of-the-art idea from mathematics and explain it as plainly as possible. Our substantial contribution resides in the parallelization of the algorithm and its circuit implementation as recursive vector operations, which renders the scalability to high resolutions of pixels.

In Sec. 2, we briefly review the Euler calculus for counting the number of touches and propose a concrete and parallelized algorithm to compute it by way of recursive vector operations. For ease of comprehension, we first design a connected component counter as a subunit. Then we construct a whole touch counter as an upper level module. In Sec. 3, we demonstrate the proposed algorithm through the implementation in a fully parallelized circuit architecture. We concretely implement the proposed algorithms for examples of touches in order to demonstrate explicitly how the proposed circuit architecture embodies the Euler integral in the form of recurrent neural networks. We successfully generated HDLs to ensure implementability and accessibility in Field-Programmable Gate Array (FPGA). Finally, Sec. 4 presents a summary and discussions.

II. THEORY AND ALGORITHM OF EULER INTEGRAL

In this section, we briefly review the mathematics of Euler calculus for counting the number of touches [22]–[25] and propose an algorithm to compute it by way of recursive vector operations. The proposed algorithm can be easily implemented in a parallel, distributed neural network architecture as we will see in the next section.

A. PROBLEM SETTING

For ease of comprehension, let us begin with a simple example in which tactile sensors, which are densely aligned on a two dimensional regular square lattice, sense tactile stimulations at each point (Figure 1). The goal here is to count the total number of touches (= three times) irrespective of the shapes and positions of the stimulating objects. Note that each touch has an arbitrary shape of finite size while each point neuron can sense and keep only the total touch number over an entire period at each point. This situation could happen when sensors have no or low temporal resolution (to save resources). The numbers on the figure represents how many times the neurons in each domain were touched. In this example, each neuron was touched from 0 to 2 times. We would like to estimate the total touch number (= 3) only from the local or pointwise touch numbers each neuron senses.

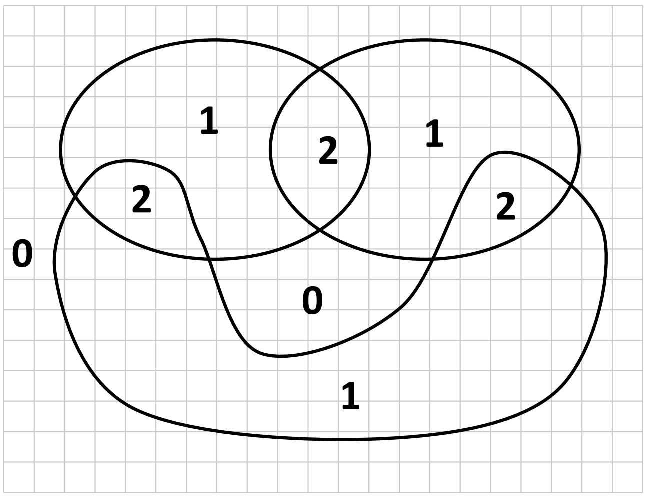

FIGURE 1. Example problem of Euler calculus on two dimensional plane. Each touch has an arbitrary shape of finite size while each point neuron can sense the total touch number at each lattice point. The neurons form a dense, square lattice on the plane. The numbers represent the touch count sensed by neurons in each domain. The total "touch" count, which we want, is three. A toy problem mimicking this example is shown in Eq. 16.

The solution can be achieved by the Euler calculus. In the followings, we first construct a connected component counter as an elemental module that can count the number of connected component in a binary image by way of recursive vector operations. Then, we design a shape-invariant touch counter for a multi-valued image based on the Euler integral as an upper level module.

B. COUNTING THE NUMBER OF CONNECTED COMPONENTS IN A BINARY IMAGE

Here we consider the problem of counting the number of connected component (β_0) in a "binary" image as in Figure 2. To keep the algorithm concrete, throughout this section we consider the 2×3 input image shown in Figure 3 (a) as an example, where each sensor neuron is activated or inactivated depending on binary input to each lattice point ($u_i = 0$ or 1).

We first define an adjacency matrix which takes 1 if the sites i and j are neighbors and the i-th and j-th neurons are both activated, $u_i = u_j = 1$ (Figure 3 (a)):

$$A_{ij} = \begin{cases} u_i u_j & (i \text{ and } j \text{ are neighbors in the 2D square lattice}) \\ 0 & (\text{otherwise}) \end{cases} \tag{1}$$

where diagonal elements A_{ii} are defined to be 0. Note that each neuron has up to four neighbors (up, down, left and right). You can regard the connections between neurons as the Hebbian-like local learning rule as only the (neighboring) neurons which are both active are connected.

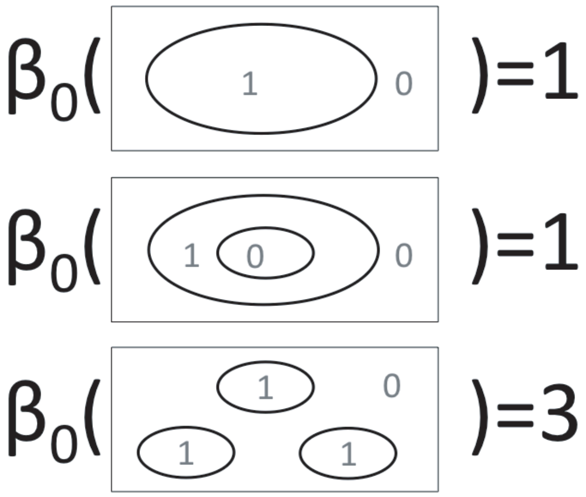

FIGURE 2. Counter of connected components. The numbers 1 and 0 represent the occupied and unoccupied domains, respectively. An ideal counter should count the number of connected components occupied irrespective of their shapes.

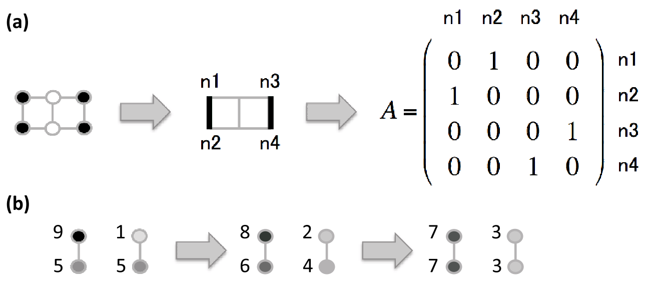

FIGURE 3. (a) Construction of an adjacency matrix of a graph according to the Hebbian-like local learning rule on a 2 × 3 example lattice. (left) The black and white circles represent the activated and inactivated neurons. (middle) Connections are formed when the neurons on the both ends of an edge are activated (Hebbian-like local learning rule). Note that only nearby neurons can form connections. (right) The matrix representation of the graph where the elements represent the existence of connections. (b) Time evolution of neurons. Neurons homogenize their states over time if they are connected. The number of destinations or different final states gives the number of connected components (= 2).

For the current purpose, you can ignore the matrix elements for the inactivated neurons as in A of Figure 3 (a) because they are just all zeros. Notice that we distinguish lattice points by a single index i although the pixels are alined on a two dimensional regular square lattice. That is, we treat the pixels as a vector u_i rather than a matrix u_{xy}, so that the following computations can be implemented in the form of vector operations.

Finally and most importantly, when time evolution of the neurons homogenize their states as in Figure 3 (b), the number of destinations or different final states gives the number of connected components (= 2), which we wanted. This is because only neurons in the same connected component can be homogenized via interactions.

To describe the time evolution in the form of a mathematical equation, let us define the Laplacian matrix Δ [27] by

$$\Delta = D - A, \tag{2}$$

where

$$D_{ii} = \sum_{j=1}^{n} A_{ij} \tag{3}$$

represents the degrees of the nodes. That is, D is a diagonal matrix with D_{ii} = (the number of connections the i-th neuron has). For our concrete example of the 2D input image (Figure 3 (a)),

$$\begin{array}{|ccc|} 1 & 0 & 1 \\ 1 & 0 & 1 \end{array}, \tag{4}$$

the Laplacian matrix Δ_{ex} is given by

$$\begin{aligned} \Delta_{ex} = D - A &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix}. \end{aligned} \tag{5}$$

where -1 represents the connected neurons (n1-n2 and n3-n4) according to the Hebbian-like local learning rule for neighbors and the diagonal elements correspond to the number of connections each neuron formed (one for all the neurons in this case).

Then, the homogenizing interaction in Figure 3 (b) can be mathematically described as

$$v|_{t=n} = (I - \alpha\Delta)^n \, v|_{t=0}, \tag{6}$$

starting from a random vector whose elements (v_i's) are distributed uniformly. Simply, a neuron updates its state toward the average state of the neighboring (active) neurons to the degree of α. The mathematical background is shown in Appendix A. As we take small enough coefficient $\alpha (\approx 1/8)$, the convergence is guaranteed (Appendix B).

Practically, to treat both activated ($u_i = 1$) and inactivated ($u_i = 0$) neurons in a unified way, we multiply u_i to $(v|_{t=0})_i$ before the iteration because v_i for inactivated neuron ($u_i = 0$) is never updated (no connection) and kept to be 0 as will be seen in Figure 8.

With this connected component counter as a subunit, we will construct a touch counter based on the Euler integral in the next subsection.

C. COUNTING THE NUMBER OF TOUCHES ON A SENSOR NETWORK

Here we review the Euler integral [22]–[24] and show how to compute it concretely by using the connected component counters. While usually the topological invariants such as connected components, numbers of holes and Euler characteristics are computed for "binary" images, the Euler integral, a generalized Euler characteristics, is computed for the "integer-valued" images. Mathematically, the Euler integral can be regarded as a (generalized) integral.

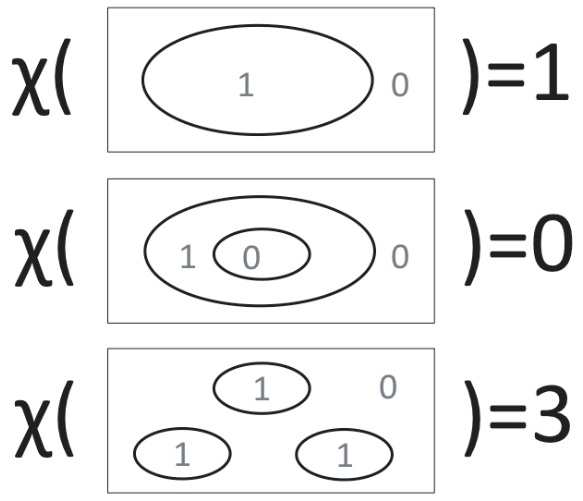

FIGURE 4. Euler characteristics for binary images: $\chi = \beta_0 - \beta_1 =$ (number of connected components) − (number of holes).

For "binary" images, the Euler integral simply returns the Euler characteristics (= the number of connected components - the number of holes). For example, in Figure 4, the first and the third examples have one and three components while the second example has one connected component and one hole, returning zero. Note that the Euler integral does not count and ignores the number of disks with a hole.

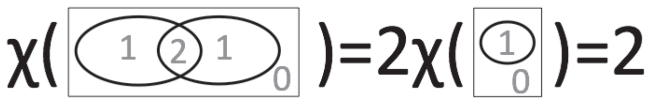

FIGURE 5. Example problem of touch counter. The total touch count is two. Euler calculus allows to decompose the integer-valued integrand into two indicator functions (Eq. 7). This decomposition guarantees the correct touch number in the Euler integral.

For "integer-valued" images $h(= 0, 1, 2, \ldots)$, the integral rule allows to decompose the integral of a sum of integrands into the sum of integrals:

$$\chi(f + g) = \chi(f) + \chi(g). \tag{7}$$

For the case of Figure 5, you can theoretically decompose the original image into the sum of two indicator functions which take 1 on simultaneously touched domains. Then, as each integral counts one in each indicator function, the sum of the integrals returns the correct answer (= 2). In this way, this integral rule results in the correct touch count in general for any touch counts.

However, the computation shown above is theoretical. In reality, you do not necessarily know how to decompose the integrand. Therefore in practice another (automatic) way of decomposition is used. That is, we decompose an integrand or an integer-valued image by levels into binary images:

$$\chi(h) = \sum_{s=0}^{\infty} \chi(h > s), \tag{8}$$

where $h > s$ represents the indicator function that takes 1 in the domain with $h > s$ and 0 elsewhere, for brevity of notation. Thus the integral reduces to the Euler characteristics for the domains whose shapes are represented by binary images. For the case of Figure 5 (left), the integral can be computed by assuming Figure 4 (top) as

$$\chi(h) = \chi(h > 0) + \chi(h > 1) + \ldots = 1 + 1 + 0 = 2. \tag{9}$$

For the case of Figure 1 (or Eq. 23), the integral can be computed by assuming Figure 4 (middle and bottom) as

$$\chi(h) = \chi(h > 0) + \chi(h > 1) + \ldots = 0 + 3 + 0 = 3. \tag{10}$$

In this way, you can compute the total touch number even if you do not know the set of simultaneously touched pixels [22]–[24].

Finally, the computation of Euler characteristics for binary images as in Figure 4 remains to be solved. This computation can be actually reduced to counting the connected components. Remember that (Euler characteristics) = (#connected components) − (#holes). As it is difficult to numerically compute the number of holes without error, we use the Alexander duality [15], [28]–[30]) for 2D planes to compute the Euler characteristics [22]–[24]:

$$\chi(A) = \beta_0(A) - \beta_1(A) = \beta_0(A) - (\beta_0(\mathbb{R}^2 \setminus A) - 1), \tag{11}$$

where A represents a (compact) domain whose pixel values are 1. β_1 is the number of holes and β_0 is the number of connected components. Here we do not prove the above formula, but the formula means that the number of holes of A in $\mathbb{R}^2$ can be counted by counting the number of connected components in the complement domain. The Euler characteristic χ computed through β_0 as above can be numerically more robust than that by the direct computation, say, with the Euler's formula [22]–[24].

By combining Eqs. 8 and 11, finally we have

$$\chi(h) = \sum_{s=0}^{\infty} \chi(h > s) = \sum_{s=0}^{\infty} [\beta_0(h > s) - \beta_0(h \leq s) + 1]. \tag{12}$$

Note that the entire computation reduced to the number of connected components β_0, which we know how to compute concretely and in a fully parallelized manner.

Let us compute examples through Alexander duality in Eq. 12 just for illustration of it. For the case of Figure 5,

$$\begin{aligned} \chi(h) &= [\beta_0(h > 0) - \beta_0(h \leq 0) + 1] \\ &\quad + [\beta_0(h > 1) - \beta_0(h \leq 1) + 1] \\ &\quad + 0 + 0 + \ldots \\ &= [1 - 1 + 1] + [1 - 1 + 1] \\ &= 2. \end{aligned} \tag{13}$$

For the case of Figure 1,

$$\begin{aligned} \chi(h) &= [\beta_0(h > 0) - \beta_0(h \leq 0) + 1] \\ &\quad + [\beta_0(h > 1) - \beta_0(h \leq 1) + 1] \\ &\quad + 0 + 0 + \ldots \\ &= [1 - 2 + 1] + [3 - 1 + 1] \\ &= 3. \end{aligned} \tag{14}$$

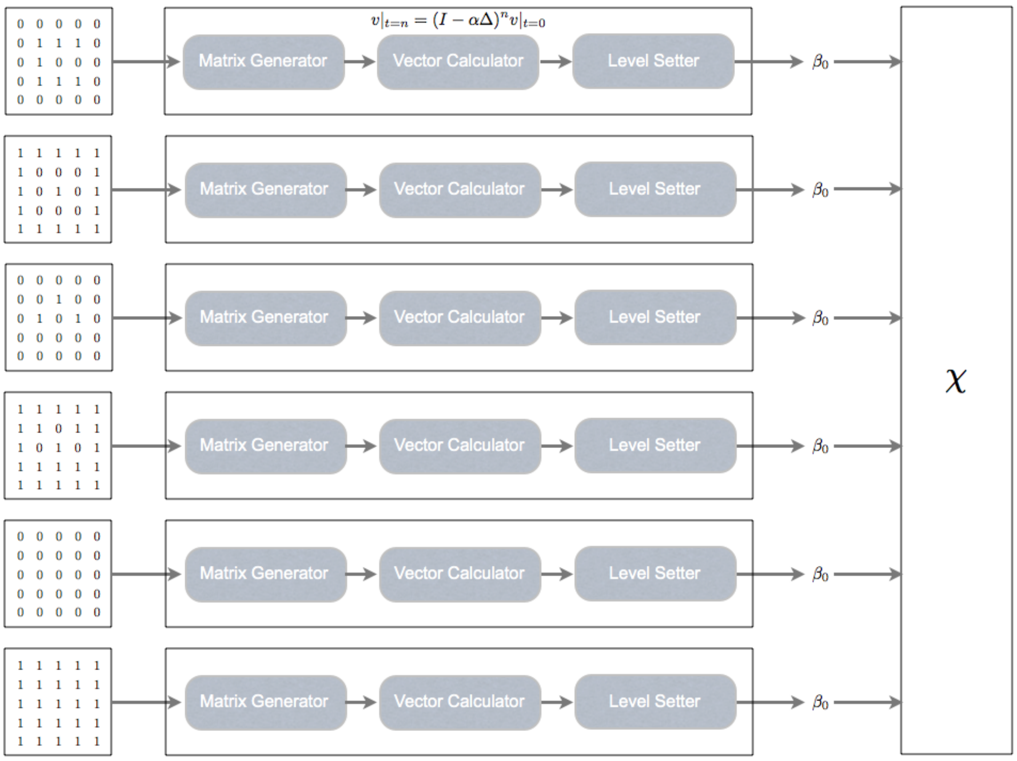

FIGURE 6. Implementation diagram for touch counter. Euler integral χ is computed from connected component counts β_0. The connected component counter β_0 consists of a Laplacian matrix generator, an iterative vector calculator for Eq. 6 and a level setter essentially consisting of sort(), diff() and sum() Matlab functions.

Notice that the Euler integral is unique independently of the ways of decompositions, because it satisfies the additivity axiom of measure [22]–[24],

$$\chi(A \cup B) = \chi(A) + \chi(B) - \chi(A \cap B), \tag{15}$$

and the finest segmentalization guarantees the unique integral as a topological invariance (= #vertices − #edges + #faces) according to the Euler's formula. The underlining assumptions are that each touch forms an open (or closed) set in the 2D plane and pixels are much finer than the touch shapes.

III. DEMONSTRATIONS BY PARALLELIZED CIRCUIT ARCHITECTURE

A. A SIMPLEST EXAMPLE WITH 5 × 5 PIXELS

In order to demonstrate the proposed algorithm, we concretely designed a fully parallelized circuit architecture (Figures 6 and 7) to automatically solve problems on the two dimensional pixels. As an illustrative example, we first solve the following problem mimicking Figure 1:

$$\begin{array}{|ccccc|} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 2 & 0 & 2 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\ . \tag{16}$$

Note that we do not get the correct answer (= three times) unless we put zeros outside. For example, if we consider another problem

$$\begin{array}{|ccc|} 1 & 2 & 1 \\ 2 & 0 & 2 \\ 1 & 1 & 1 \end{array}\ , \tag{17}$$

we get a wrong touch number due to wrong β_0 such as

$$\beta_0(h \le 0) = \beta_0 \left(\begin{array}{|ccc|} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{array} \right) = 1.\ \text{(wrong! see Eqs. 21)} \tag{18}$$

Thus we have to be careful about the boundary condition in modeling 2D plane, especially when there are holes inside.

To save users' efforts to manually insert 0's outside the input touch image, we implemented the automatic insertion as a preprocessing of Matrix Generator in Figure 6. As the peripheral pixels can flip to 1 after the operations such as $h \leqq 0$, it is appropriate to implement this insertion initially.

For ease of comprehension, first we elaborate on the connected component counter β_0("*"), which works as an elemental module. The proposed circuit generates the 25×25 Laplacian matrix Δ from the input pixel values u according to Eq. 2. For example, for the binary image with full activations

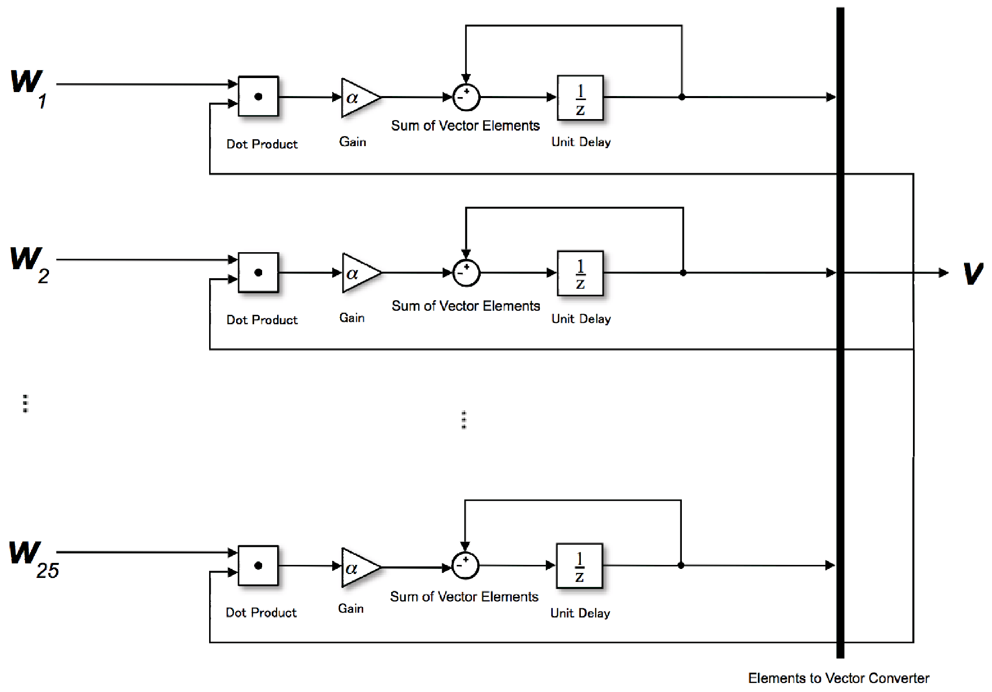

FIGURE 7. **Implementation diagram for iterative vector calculator for Eq. 6 inside connected component counter.**

(corresponding to Eq. 21 (bottom right)),

$$u = \boxed{\begin{matrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{matrix}}\,, \tag{19}$$

Δ is given by Eq. 20, as shown at the bottom of this page, where -1 represents the connected edges (in this specific case, all the neighboring neurons) while diagonal elements represent the number of edges for each neuron (four for inner neurons).

As shown in Figure 7, the Laplacian matrix Δ and a random vector v are input to the module, where the operation of Δv is realized as an inner product $(w_i \cdot v)$ between v and the rows w_i of the Laplacian matrix $\Delta = (w_1; w_2; \ldots; w_{25})$. The iterative vector calculation of Eq. 6, which essentially homogenizes the states within each connected component, is implemented recursively with unit time delay to save space. Thus we realized a connected

$$\begin{pmatrix}
2 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 3 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 3 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 3 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 3 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 2
\end{pmatrix} \tag{20}$$

$$\beta_0(h>0)=\beta_0\left(\begin{array}{|ccccc|}\hline 0&0&0&0&0\\0&1&1&1&0\\0&1&0&1&0\\0&1&1&1&0\\0&0&0&0&0\\\hline\end{array}\right)=1,\quad \beta_0(h\le 0)=\beta_0\left(\begin{array}{|ccccc|}\hline 1&1&1&1&1\\1&0&0&0&1\\1&0&1&0&1\\1&0&0&0&1\\1&1&1&1&1\\\hline\end{array}\right)=2,$$
$$\beta_0(h>1)=\beta_0\left(\begin{array}{|ccccc|}\hline 0&0&0&0&0\\0&0&1&0&0\\0&1&0&1&0\\0&0&0&0&0\\0&0&0&0&0\\\hline\end{array}\right)=3,\quad \beta_0(h\le 1)=\beta_0\left(\begin{array}{|ccccc|}\hline 1&1&1&1&1\\1&1&0&1&1\\1&0&1&0&1\\1&1&1&1&1\\1&1&1&1&1\\\hline\end{array}\right)=1,$$
$$\beta_0(h>2)=\beta_0\left(\begin{array}{|ccccc|}\hline 0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\\hline\end{array}\right)=0,\quad \beta_0(h\le 2)=\beta_0\left(\begin{array}{|ccccc|}\hline 1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\\hline\end{array}\right)=1. \tag{21}$$

component counter $\beta_0(``*'')$ in a fully parallelized manner.

Figure 8 shows the recursive time evolutions of v implemented on the circuit simulator for the six level sets in Eqs. 21, as shown at the top of this page, for our example in Eq. 16. Note that the number of destinations gives the number of connected components β_0 in each input figure. For example, the top left figure and the middle left figure show one and three connected components respectively. Note that in the middle left figure, v stays at the initial values under the iteration of Eq. 6 as there is no connection in this case. As can be expected from the converging nature of Figure 8, the result is robust in the sense that any noise on v halfway does not change the final result at all.

Finally, the upper module that computes the Euler integral χ just calls the connected component counters, β_0 (Eq. 12), and it can be implemented straightforwardly (Figure 6). For our example in Eq.16 the touch number can be correctly computed as "three times" from the connected components of level sets (Eqs. 21) as

$$\begin{aligned}\chi(h) &= \chi(h>0)+\chi(h>1)+\chi(h>2)+\ldots\\ &= [\beta_0(h>0)-\beta_0(h\le 0)+1]\\ &\quad+[\beta_0(h>1)-\beta_0(h\le 1)+1]\\ &\quad+[0]+\ldots\\ &= [1-2+1]+[3-1+1]+[0]+\ldots\\ &= 0+3+0+\ldots\\ &= 3. \end{aligned}\tag{22}$$

Note that our level-set decomposition into characteristic functions, $h=\mathbb{1}(h>0)+\mathbb{1}(h>1)$, or

$$\begin{array}{|ccccc|}\hline 0&0&0&0&0\\0&1&2&1&0\\0&2&0&2&0\\0&1&1&1&0\\0&0&0&0&0\\\hline\end{array}=\begin{array}{|ccccc|}\hline 0&0&0&0&0\\0&1&1&1&0\\0&1&0&1&0\\0&1&1&1&0\\0&0&0&0&0\\\hline\end{array}+\begin{array}{|ccccc|}\hline 0&0&0&0&0\\0&0&1&0&0\\0&1&0&1&0\\0&0&0&0&0\\0&0&0&0&0\\\hline\end{array}, \tag{23}$$

differs from the one into original touch shapes in Figure 1, which should be,

$$\begin{array}{|ccccc|}\hline 0&0&0&0&0\\0&1&2&1&0\\0&2&0&2&0\\0&1&1&1&0\\0&0&0&0&0\\\hline\end{array}=\begin{array}{|ccccc|}\hline 0&0&0&0&0\\0&1&1&0&0\\0&1&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\\hline\end{array}+\begin{array}{|ccccc|}\hline 0&0&0&0&0\\0&0&1&1&0\\0&0&0&1&0\\0&0&0&0&0\\0&0&0&0&0\\\hline\end{array}+\begin{array}{|ccccc|}\hline 0&0&0&0&0\\0&0&0&0&0\\0&1&0&1&0\\0&1&1&1&0\\0&0&0&0&0\\\hline\end{array}. \tag{24}$$

Thus we can compute the correct touch number as a topological invariant independent of the decomposition even if we do not know the original decomposition.

We successfully computed the touch count as the Euler integral $\chi(=\ 3)$ for our example in Eq. 16 by executing our Simulink model/library on a PC platform and generated the corresponding HDL code for verification in large-scale through SW/HW co-simulations on FPGA platforms. All the digital computations were done to the 16-bit precision with 8-bit after the decimal point. All the imple-

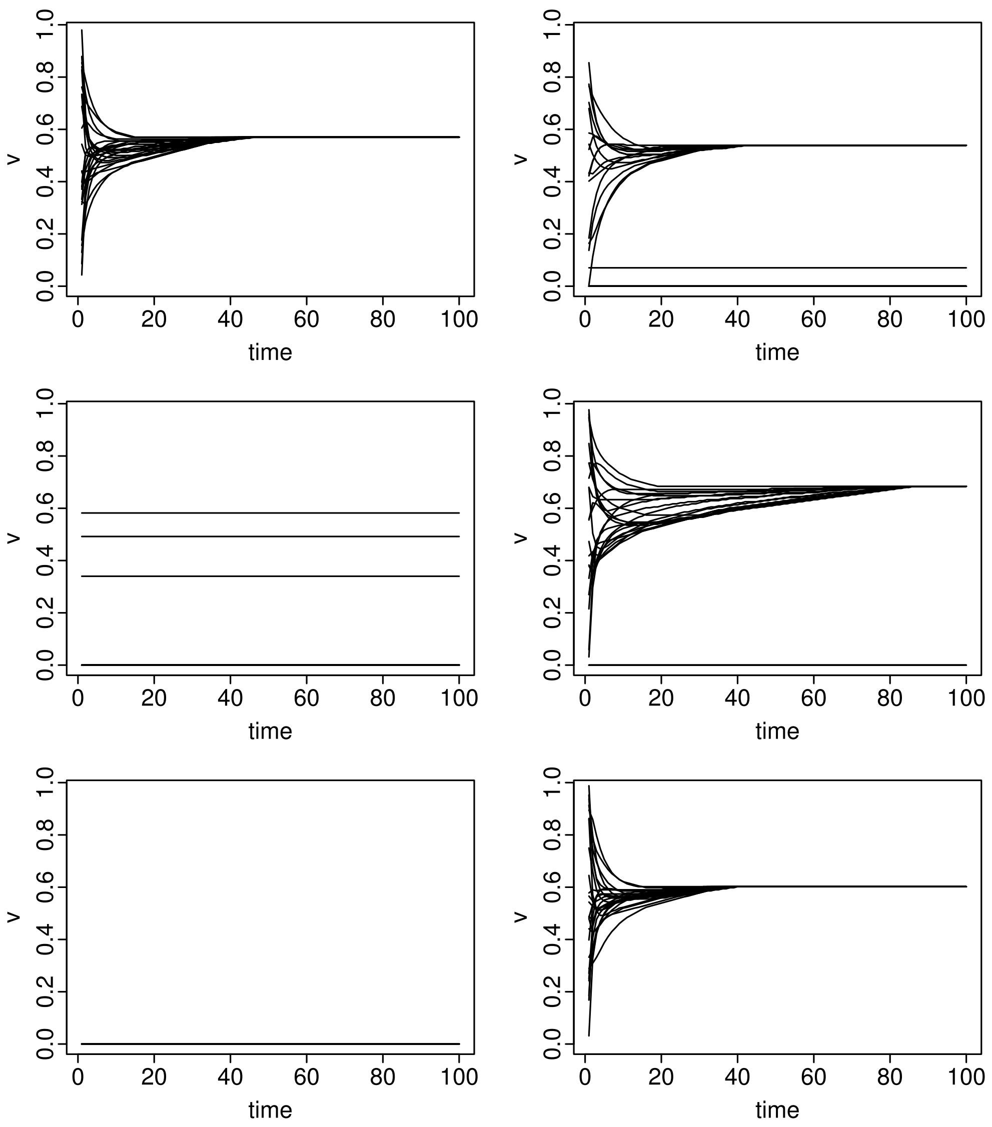

FIGURE 8. Time evolution of v by Eq. 6 for six cases in Eqs. 21. The number of destinations gives the number of connected components β_0 in each input figure. 25 lines in each figure represent the time evolutions of 5 × 5 neurons, $v = (v_1, v_2, \ldots, v_{25})$.

mentation including the level set operation, which is essentially "sum(diff(sort(v)))," are designed as custom Simulink model/library employing vector operations. In particular, the iterative vector calculator is implemented using recursive vector operations as described as the block diagram in Figure 7. In this way, the proposed circuit has scalability to many pixels because we can generate HDL seamlessly and automatically from the same Simulink model/library only by adjusting the scale parameter (= resolution of the touch sensor) as we will see next.

B. TOWARD A LARGER PROBLEM

So far, we implemented up to 5×5 pixels as an input image. Although an automated design might be helpful for having more pixels without wiring complications (Eq. 19), the proposed algorithm has the potential in scaling in principle. In fact, we successfully implemented at least up to 10×10 pixels for now (Figure 9) with straightforward modularizations.

Notice that most of the edges are fixed to be 0 independent of the input touch image because only the edges for neighboring neurons on the lattice can be nonzero. For example, even the densest case of Eq. 19 for 5 × 5 pixels gives a fairly sparse matrix shown in Eq. 19. Therefore, the number of nonzero connections is not $O(N^2)$, but $O(4N)$, fortunately. Here $4N$ appears because each pixel has (up to) four neighbors. As the iterative vector calculation of Eq. 6 is implemented recursively, we do not need to represent the full (sparse) matrix and rather we only need to consider the existence of $4N$ connections in the module depending on the input pixels.

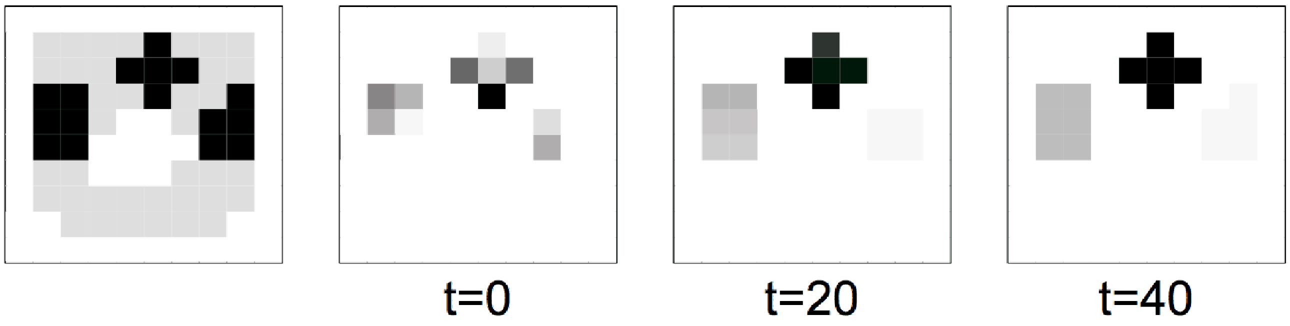

FIGURE 9. A larger example (10 × 10) mimicking Figure 1 in a circuit simulation. (left) The input image. The black color denotes the overlap region (h=2) and the gray color denotes the single-touched region (h=1). (others) Time evolution of v by Eq. 6 for the multiple-touched region ($h > 1$). The number of destinations (different states) gives the number of connected components: $\beta_0(h > 1) = 3$. Note that the connected neurons homogenize their states by mutual interactions.

For the problem with $N = 25$ as in Eq. 16, $4N = 100$ is more tractable than $N^2 = 625$. For the problem with $N = 100$ as in Figure 9, $4N = 400$ is much more tractable than $N^2 = 10000$. Imagine that N^2 can easily get untractably large for larger N. This theoretical estimate demonstrates the scalability of our architecture to high resolutions of pixels.

In the higher level of the design process, designers could use the full matrix for describing Δ. Then, as most of elements of the matrix are actually fixed to zero, it is expected that the compiler automatically optimize the circuits at the logic synthesis from HDL. In any case, we expect that the FPGA resources scale as $O(4N)$ with the number of pixel N after the optimization.

IV. SUMMARY AND DISCUSSIONS

In this paper, we proposed a sensor network that accurately counts the number of touches irrespective of their shapes and positions based on the Euler calculus. The Euler calculus is a generalization of the rule of integral where the integral gives Euler characteristics or touch counts. We designed a fully parallelized touch counter for two dimensional sensor networks. We concretely implemented our proposed circuits for examples of touches in order to demonstrate transparently how the proposed circuit architecture embodies the Euler integral in the form of fully parallelized and recursive vector operations, with scalability to high resolutions of pixels.

The advantage of our implementation resides in the parallelism and recursiveness of the main connected component counter module. Fully parallelized circuit implementation in the form of distributed sensory neurons with only local connections can potentially save computational time drastically, compared with algorithms on serial system PCs. The proposed circuit has scalability with many pixels because all the computations are implemented as recursive vector operations. For example, while the computational time for the multiplication of a matrix to a vector can be $O(N^3)$ on serial PC, it is $O(1)$ if neurons update their states at once. We also save space ($O(N)$) by computing the core iterative vector calculation in Eq. 6 in a recursive manner as discussed at the end of Sec. 3. We designed a functional circuit by constructing the block diagram of Matlab Simulink in a bottom-up manner and generating HDL in a top-down manner. The reduction of many input pixels into a few integers is where recurrent neural networks are good at and topological invariants appear. Although we treated only zero-th order Laplacian and Betti number in this paper, our iterative scheme can be directly generalized to those of higher orders. Thus our parallelization can lead the way to FPGA or Digital Signal Processor implementations of topological algorithms with scalability to high resolutions of pixels.

As we aimed to understandably explain the background theory and the algorithm by way of iterative vector operations, we focused on instructive problems with up to 10×10 pixels in our circuit implementation and just estimated theoretically that our circuit has scalability. Further improvement may be needed for more practical applications. We would like to leave realistic and large problems as well as the details of parameter tuning for the future work.

The realization of shape-invariant touch counter in a fully parallelized neural network could give an insight into the information processing in the brain. Our naturally-parallelized algorithm of Euler calculus, which we call ''neural implementation'' from biological plausibility, can be interpreted as a brain model, as it can be readily compared with the biological neurons. The concrete construction of a functional neural network will help the understanding of the brains ability to recognize unity from the global consistency of the tactile or visual stimuli.

APPENDIX A MATHEMATICAL BACKGROUND OF TIME EVOLUTION OF NEURONS IN EQ. 6

In counting the number of connected component via Eq. 6, we utilized the fact that the problem of counting the connected component in a binary image can be reduced to the problem of calculating a zero eigenvector v of the Laplacian matrix Δ [27], [31]:

$$\Delta v = 0. \tag{25}$$

To be precise, the dimension of the solution vector space for zero eigenvalue is equal to the number of the connected components. This is because the solution of Eq. 25 is the function that is constant on each connected component as Δ is essentially a differential operator. For our example, the eigenvalues of Δ_{ex} are $\lambda = (0, 0, 2, 2)$ and the zero-eigenvector is given by

$$v_{ex} = \begin{pmatrix} a \\ a \\ b \\ b \end{pmatrix}, \tag{26}$$

where the constants a and b are arbitrary because the vector space for zero-eigenvalue is two dimensional. Because there are two levels a and b, you can conclude that the graph represented by this example Laplacian matrix Δ_{ex} has two blocks of connected components (n1-n2 and n3-n4 in Figure 3).

As our final goal is to implement the above algorithm in a parallel and scalable manner, we compute the eigenvectors for zero-eigenvalue by the iterative vector operation in Eq. 6 mimicking the time evolution of a neural network with only local connections.

When you take small enough coefficient α in Eq. 6, non-zero eigenvectors shrink and disappear with iterations. For the example Laplacian matrix Δ_{ex} in the main text, the eigenvalues of $(I-\alpha\Delta_{ex})$ are $\lambda = (1, 1, 1-2\alpha, 1-2\alpha)$ and the eigenvalues of $(I-\alpha\Delta_{ex})^n$ are $\lambda = (1, 1, (1-2\alpha)^n, (1-2\alpha)^n)$. Thus the components for non-zero eigenvalues disappear through multiplications of $(1 - 2\alpha)(< 1)$. In this way, we get the final state, v_{final}, of the form in Eq. 26.

Note that the dimension or the number of different values $(= 2)$ in the final state, v_{final}, is equal to the number of connected components consisting of only active neurons, which we wanted for the input image in Eq 4 or Figure 3.

APPENDIX B
OPTIMAL VALUE OF α FOR ITERATION IN EQ. 6

Here we prove that the maximal convergence rate is achieved when $\alpha = 1/8$ (if oscillations are not allowed).

We consider the condition for the vector in the iteration to converge to a zero-eigenvector for sure. The problem can be reduced to compute the (maximum) eigenvalues of Δ, $(\lambda_1 = 0 \leq \lambda_2 \leq \ldots \leq \lambda_{max})$, as the iteration in Eq. 6 multiplies

$$1 - \alpha\lambda_i \tag{27}$$

to i-th eigenspace. Then, for all the eigenspaces except for zero-eigenspace to shrink to zero,

$$|1 - \alpha\lambda_i| < 1 \tag{28}$$

is required. This is marginally achieved when

$$|1 - \alpha\lambda_{max}| = 0 \quad \text{or} \quad \alpha = 1/\lambda_{max} \tag{29}$$

if oscillations are not allowed and achieved when

$$|1 - \alpha\lambda_{max}| = -1 \quad \text{or} \quad \alpha = 2/\lambda_{max}. \tag{30}$$

if oscillations are allowed. We will show below that $\lambda_{max} \leq 8$ and thus we must set $\alpha < 1/8$ to ensure non-oscillatory convergence. The optimal convergence rate is achieved when $\alpha \approx 1/8$ because the larger α the faster the convergence. As our precision in circuit implementation is 8-bit after the decimal point, we actually used $\alpha = 0.117$ throughout the paper. Notice that the convergence rate itself can be estimated from the second smallest eigenvalue λ_2, which depends on problems [32], [33].

Finally we will prove that $\lambda_{max} \leq 8$. Let us sort and scale v, that satisfies $\lambda v = \Delta v$, so that $1 = v_1 > |v_i|$ for $i \geq 2$. Then the first component of the eigen equation for any λ leads to

$$|\lambda| = |\lambda|v_1 = |\sum_j \Delta_{1j}v_j| \leq \sum_j |\Delta_{1j}| \leq 2 * (\text{max degree}) = 8. \tag{31}$$

REFERENCES

[1] J. L. Krichmar and H. Wagatsuma, *Neuromorphic and Brain-Based Robots*. Cambridge, U.K.: Cambridge Univ. Press, 2012.

[2] T. Furukawa, "SOM of SOMs: An extension of SOM from 'map' to 'homotopy'," in *Neural Information Processing* (Lecture Notes in Computer Science), vol. 4232. Berlin, Germany: Springer-Verlag, 2006, pp. 950–957.

[3] A. Nieder, D. J. Freedman, and E. K. Miller, "Representation of the quantity of visual items in the primate prefrontal cortex," *Science*, vol. 297, no. 5587, pp. 1708–1711, 2002.

[4] B. Butterworth, *What Counts: How Every Brain is Hardwired for Math*. New York, NY, USA: Free Press, 1999.

[5] K. Devlin, *The Math Gene: How Mathematical Thinking Evolved and Why Numbers are Like Gossip*. New York, NY, USA: Basic Books, 2001.

[6] R. Romo, L. Lemus, and V. de Lafuente, "Sense, memory, and decision-making in the somatosensory cortical network," *Current Opinion Neurobiol.*, vol. 22, no. 6, pp. 914–919, 2012.

[7] M. Minsky and S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969.

[8] H. Okuno and T. Yagi, "Image sensor system with bio-inspired efficient coding and adaptation," *IEEE Trans. Biomed. Circuits Syst.*, vol. 6, no. 4, pp. 375–384, Aug. 2012.

[9] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, nos. 1–3, pp. 239–255, Dec. 2010.

[10] O. L. Savkay, N. Yildiz, E. Cesur, M. E. Yalcin, and V. Tavsanoglu, "Realization of preprocessing blocks of CNN based CASA system on FPGA," in *Proc. IEEE Eur. Conf. Circuit Theory Design (ECCTD)*, Sep. 2013, pp. 1–4.

[11] M. Ishikawa, "Learning of modular structured networks," *Artif. Intell.*, vol. 75, no. 1, pp. 51–62, 1995.

[12] M. Ishikawa, "Structural learning with forgetting," *Neural Netw.*, vol. 9, no. 3, pp. 509–521, 1996.

[13] W. Fulton, *Algebraic Topology: A First Course*. New York, NY, USA: Springer-Verlag, 1995.

[14] R. Bott and L. W. Tu, *Differential Forms in Algebraic Topology*. New York, NY, USA: Springer-Verlag, 1982.

[15] A. Hatcher, *Algebraic Topology*. Cambridge, U.K.: Cambridge Univ. Press, 2002.

[16] H. Edelsbrunner and J. L. Harer, *Computational Topology*. Providence, RI, USA: AMS, 2009.

[17] T. Kaczynski, K. Mischaikow, and M. Mrozek, *Computational Homology*. New York, NY, USA: Springer-Verlag, 2010.

[18] Z. Arai, H. Kokubu, and P. Pilarczyk, "Recent development in rigorous computational methods in dynamical systems," *Jpn. J. Ind. Appl. Math.*, vol. 26, nos. 2–3, pp. 393–417, 2009.

[19] M. Gameiro, Y. Hiraoka, S. Izumi, M. Kramar, K. Mischaikow, and V. Nanda, "Topological measurement of protein compressibility via persistence diagrams," *MI Preprint Ser.*, vol. 6, pp. 1–10, 2012.

[20] C. Curto, V. Itskov, A. Veliz-Cuba, and N. Youngs, "The neural ring: An algebraic tool for analyzing the intrinsic structure of neural codes," *Bull. Math. Biol.*, vol. 75, no. 9, pp. 1571–1611, 2013.

[21] Z. Chen, S. N. Gomperts, J. Yamamoto, and M. A. Wilson, "Neural representation of spatial topology in the rodent hippocampus," *Neural Comput.*, vol. 26, no. 1, pp. 1–39, 2014.

[22] Y. Baryshnikov and R. Ghrist, "Target enumeration via euler characteristic integrals," *SIAM J. Appl. Math.*, vol. 70, no. 3, pp. 825–44, 2009.
[23] Y. Baryshnikov and R. Ghrist, "Euler integration over definable functions," *Proc. Nat. Acad. Sci. USA*, vol. 107, no. 21, pp. 9525–9530, 2010.
[24] J. Curry, R. Ghrist, and M. Robinson, "Euler calculus with applications to signals and sensing," in *Proc. Symp. Appl. Math.*, vol. 70. 2012, pp. 75–146.
[25] Z. Arai, K. Hayashi, and Y. Hiraoka, "Mayer–Vietoris sequences and coverage problems in sensor networks," *Jpn. J. Ind. Appl. Math.*, vol. 28, no. 2, pp. 237–250, 2011.
[26] A. Tahbaz-Salehi and A. Jadbabaie, "Distributed coverage verification in sensor networks without location information," in *Proc. 47th IEEE Conf. Decision Control*, Dec. 2008, pp. 4170–4176.
[27] F. R. K. Chung, *Spectral Graph Theory*. Providence, RI, USA: AMS, 1997.
[28] J. W. Alexander, "A proof of the invariance of certain constants of analysis situs," *Trans. Amer. Math. Soc.*, vol. 16, no. 2, pp. 148–154, 1915.
[29] L. Pontryagin, "The general topological theorem of duality for closed sets," *Ann. Math.*, vol. 35, no. 4, pp. 904–914, 1934.
[30] P. S. Aleksandrov, *Combinatorial Topology*. Rochester, NY, USA: Graylock, 1956.
[31] M. Nakahara, *Geometry, Topology and Physics*. Boca Raton, FL, USA: CRC Press, 2003.
[32] B. Mohar, "The Laplacian spectrum of graphs," in *Graph Theory, Combinatorics, and Applications*, vol. 2. New York, NY, USA: Wiley, 1991, pp. 871–898.
[33] M. Tanaka. (2013). "Multi-way expansion constants and partitions of a graph." [Online]. Available: http://arxiv.org/abs/1112.3434

KEIJI MIURA received the Ph.D. degree in science from Kyoto University, Kyoto, Japan, in 2006. From 2006 to 2008, he was a JSPS Research Fellow with the University of Tokyo, Tokyo, Japan.

He was a JST PRESTO Researcher with Harvard University, Cambridge, MA, USA, from 2008 to 2011. Since 2011, he has been an Assistant Professor with Tohoku University, Sendai, Japan. His research area is mathematical neuroscience.

KAZUKI NAKADA received the B.Sc. degree in mathematics, and the M.E. and Ph.D. degrees in electrical engineering from Hokkaido University, Sapporo, Japan, in 2002 and 2005, respectively.

He was an Assistant Professor with the Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology, Kitakyushu, Japan, from 2005 to 2011. In 2011, he joined the Advanced Electronics Research Division at the INAMORI Frontier Research Center, Kyushu University, Fukuoka, Japan, as a Research Associate. Since 2013, he has been with the Graduate School of Informatics and Engineering, University of Electro-Communications, Tokyo, Japan. His main research interests are the mathematical analysis and physical implementation of unconventional computing, including cognitive neuromorphic computing. Toward the research aim, he is currently exploring novel design principles exploiting cooperative phenomena in silicon electronics and spintronics.