# Feedforward Categorization on AER Motion Events Using Cortex-like Features in a Spiking Neural Network

Bo Zhao, *Member, IEEE*, Ruoxi Ding, *Student Member, IEEE*, Shoushun Chen, *Senior Member, IEEE*, Bernabe Linares-Barranco, *Fellow, IEEE* and Huajin Tang, *Member, IEEE*

***Abstract*—This paper introduces an event-driven feedforward categorization system, which takes data from a temporal contrast Address Event Representation (AER) sensor. The proposed system extracts bio-inspired cortex-like features and discriminates different patterns using an AER based tempotron classifier (a network of Leaky Integrate-and-Fire spiking neurons). One of the system's most appealing characteristics is its event-driven processing, with both input and features taking the form of address events (spikes). The system was evaluated on an AER posture dataset and compared to two recently developed bio-inspired models. Experimental results have shown that it consumes much less simulation time while still maintaining comparable performance. In addition, experiments on the MNIST image dataset have demonstrated that the proposed system can work not only on raw AER data but also on images (with a preprocessing step to convert images into AER events) and that it can maintain competitive accuracy even when noise is added. The system was further evaluated on the MNIST-DVS dataset (in which data is recorded using an AER dynamic vision sensor), with testing accuracy of 88.14%.**

***Index Terms*—Feedforward categorization, AER, Spiking Neural Network, event-driven, MNIST.**

## I. Introduction

NEUROMORPHIC engineering is a growing branch of engineering that takes inspiration from biological neural systems in order to optimize engineered systems. By incorporating novel knowledge from neuroscience, researchers in neuromorphic engineering aim to build electronic systems that have the same efficiency as biological computation [1], [2]. Recent years have witnessed increasing efforts in event-driven neuromorphic systems [3]–[7]. One desire behind these efforts is to emulate the biological usage of the asynchronous sparse event-driven signaling as a core aspect of the computational architecture. Address Event Representation (AER) sensors naturally provide a way to incorporate demand-based computation. AER sensors have an output-by-demand nature. They remove a lot of data redundancy in the scene, and only output the relevant information (i.e. "features") as an asynchronous stream of digital events, which makes the following processing systems able to be designed as fully event-driven. In particular, AER vision sensors enable pixel-parallel image processing at the focal plane. Each pixel in the sensor can individually monitor the relative change of light intensity, and it will request to output an event if the change is greater than a user-defined threshold. There are cases wherein multiple pixels request to output events at the same time and, therefore, we need asynchronous row and column arbitration tree circuits to process the pixel requests and arrange the output sequence in a fairly random manner [8]. Only one pixel request is granted at a time. Once the arbitration process is completed, the pixel address is sent out, and the pixel will restart its operation. The output of AER vision sensors is a stream of address events. These sensors are often categorized as temporal contrast AER silicon retinas [9], [10].

Despite many institutions using the AER protocol, interfacing hardware remains difficult and it requires a deep understanding of all the components used. One drawback of AER silicon retinas is the high cost of the silicon area per pixel. Limited feature extraction can be carried out at the pixel level [11], [12] and hence the output events are barely enough for the direct input of classification algorithms. Additional preprocessing, such as segmentation, resizing, repositioning, and even more complicated high level feature extraction, is still needed. However, most existing algorithms are based on conventional frame-driven image sensors. In order to adopt these algorithms, a common practice (jAERViewer [13], for example) is to divide events into fixed time slices (20 ms, for example) and accumulate them into pseudo-pictures. Each incoming event is associated with an address, which is used to "light" the corresponding pixel in the picture. Fig. 1 shows a space-time scatter plot of one piece of address events captured by an AER vision sensor [14]. A person is performing stand-up and sit-down actions in this recording. The lower part of this figure shows several selected frames reconstructed from the address events. By inspecting these reconstructed frames, we can easily find that the human silhouette in some frames is incomplete or totally missing. The main difficulty arrives from the asynchronous nature of "motion" with respect to the time "slice". A motion may fall into two time slices and neither of the two pseudo pictures tells the right story. In fact, this is a common problem when using frame-driven image sensors for motion processing.

In order to fully utilize the power of AER sensors, the concept of event-driven processing should be applied to every

B. Zhao and H. Tang are with the Institute for Infocomm Research, Agency for Science, Technology and Research (A*STAR), Singapore 138632 (e-mail: {zhaob,htang}@i2r.a-star.edu.sg).

R. Ding and S. Chen are with School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798 (e-mail: {ding0075,eechenss}@ntu.edu.sg).

B. Linares-Barranco is with the Institute of Microelectronics of Seville (IMSE-CNM-CSIC), Spain (e-mail: bernabe@imse-cnm.csic.es).

Correspondence: Shoushun Chen. e-mail: eechenss@ntu.edu.sg

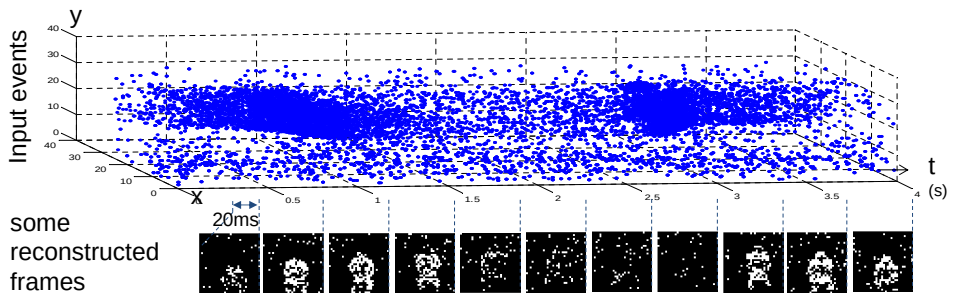



Fig. 1. Example of one piece of address events captured by an AER vision sensor. The lower part shows some reconstructed frames by dividing events into time slices and applying accumulation in each slice. We can find that the human silhouette in some frames is incomplete or totally missing. This is a drawback of frame-driven processing on motion events.

signal processing stage. For example, event-driven object tracking is studied in [15], [16]. In [15], an embedded vision system is designed and combined with an AER vision sensor to achieve real-time object tracking using an efficient event-driven clustering algorithm. Delbruck and Lichtsteiner [16] adopt a similar algorithm for tracking, and they further used the tracking results to control a servo motor goalkeeper. Running on a laptop computer, the system can track and block balls with low latency (approximately 2.8 ms). In addition, the event-driven convolution for feature extraction has been exploited in [17]. AER 2D convolution chips for neuromorphic spike-based cortical processing have been designed to accelerate the convolutions of programmable kernels over the AER visual input. These convolution chips have been combined with other AER processing blocks to build larger neuromorphic systems [18], [19]. The "Convolution AER Vision Architecture for Real-Time" (CAVIAR) project [18] is such a massive neuromorphic vision system that it performs sensory, processing, learning, and actuating in a row under the AER hardware framework. The system senses the motion of objects with a temporal contrast silicon retina. It performs feature extraction through convolutional processing and winner-take-all. Learning chips based on spiking neurons are also included in CAVIAR for spatial-temporal pattern classification. The system can recognize and track a rotating dot of a certain size. Although this application is simple, CAVIAR demonstrates the power and potential of the promising AER technology. Moreover, the event-driven convolution is also applied to Convolutional Networks (ConvNets) in [19] to generate a frame-free event-driven ConvNet for feature extraction and categorization on AER visual events. The event-driven ConvNets have a similar architecture with the conventional frame-driven ConvNets [20], where convolution and subsampling modules interlace. Due to the frame-free processing, event-driven ConvNets have a significant improvement in terms of input-output latency. However, the learning of the event-driven ConvNets is based on mapping from frame-driven ConvNets but not naturally spike-based learning.

This work seeks to adopt event-driven processing at every signal processing stage. We introduce an event-driven feed-forward categorization system, which takes events from a temporal contrast AER vision sensor. The sensor is equipped with direct difference hardware in the pixel and outputs an event if a threshold is reached [14]. The output data is a stream of address events. Each event has an address and a time stamp; the address indicates which pixel the event is from, and the time stamp represents the event's time of occurrence. Each address event is sent in parallel to a battery of orientation filters based on the Gabor functions [21], and the convolution operation is performed on the fly to generate a batch of feature maps (see Fig.2 and section II for more details). The feature extraction unit is inspired by a recent hierarchical model of object categorization in the primate visual cortex [22], [23]. Each neuron competes with other neurons located within its receptive field, and it can only survive and reach the higher layer if it wins a MAX-like operation [24], [25]. In addition, we proposed an asynchronous Motion Symbol Detector to activate another stage of spike generation. The dynamics of the aforementioned survival neurons, which represent the strength of "features", go through a small set of neurons that work in Time-to-First Spike (TFS) mode. The generated spikes, again in the form of AER, are fed to a spiking neural network namely tempotron [26] for classification. Developing integrated spiking network models that include both encoding and learning stages for rapid and efficient pattern recognition has attracted increasing interests [27], [28] recently. The full tempotron network is very large. However, due to the MAX operation and the AER nature of the feature spikes, we can achieve the same results as the full network using a very small network that has only 100 inputs. This will tremendously reduce the hardware cost. Our major contribution resides in two areas: 1) an asynchronous Motion Symbol Detector to capture "motion symbols" and then trigger the classification; 2) a virtually fully connected tempotron network that could greatly reduce the hardware cost.

The ultimate aim of this work is to develop a real-time human posture categorization system using an AER temporal contrast vision sensor, which does not produce intensity images but rather a train of spikes. In the application of assisted living, due to privacy concerns, the elderly may be reluctant to be monitored by conventional image sensors. The AER vision sensor solves this problem since its event output protects the privacy of the person being monitored.

The rest of this paper is organized as follows. Section II describes the system architecture and Sections III - V illustrate its building modules. The experimental results are reported in Section VI. Some discussions are provided in Section VII and conclusions are drawn in Section VIII.

## II. System Architecture

Fig. 2 shows the architecture of the proposed system. One appealing characteristic of our system is its fully event-driven processing. Similar to most categorization systems, it can be divided into two parts, namely feature extraction and classification. The classifier that we use is a spiking neural network constructed with tempotron neurons, which can efficiently learn and discriminate spatiotemporal spike patterns. The flow of information processing is as follows.

1) **Feature Map Construction and Neuron Competition** Each address event from the AER vision sensor will be projected onto a group of simple $S1$ filters. Each filter models a neuron cell that has a certain size of receptive field and responds best to a basic feature of a certain orientation. The response of each $S1$ neuron is changing dynamically due to the event-driven convolution as well as a forgetting mechanism. Leakage is introduced to eliminate the impact of very old motion events on current response. Each $S1$ neuron competes with other neurons that are located within its receptive field. It can only survive and reach the higher layer $C1$ if it wins the MAX operation. The survived $C1$ neurons represent some salient bar features [24], [25].
2) **Motion Symbol Detection and Feature Spike Generation** Note that $S1$ and $C1$ maps are updated for each incoming AER motion event. In order to avoid carrying out classification on the feature maps all the time, a "Motion Symbol Detector" module is introduced in our system. This module consists of a leaky integration neuron and a peak detection unit. Each input event initiates a postsynaptic potential to this neuron. The total potential is continuously monitored by the peak detection unit. When a peak is detected, a pulse will be triggered to turn $ON$ the switches in Fig. 2. At that moment, $C1$ feature maps are fed to a set of "TFS neurons", where $C1$ responses are converted into spikes.
3) **Categorization by a Spiking Neuron Network** The classifier that we use is a network of tempotron neurons. In principle, we need all the $C1$ responses for classification. The number of inputs of the tempotron network is the same as the number of $C1$ responses. Thanks to the MAX operation and the AER nature of the feature spikes, we can achieve a virtually fully connected system by physically activating only a very small subset of the network. Only a small portion of neurons survive in $C1$ feature maps after the MAX operation and, therefore, we only need to build a few "TFS neurons" for the response-to-spike conversion. Each feature spike is associated with an address, which can be used to access a lookup table (LUT) and fetch the corresponding weight.

Note that, in our system, spikes are used in all the processing stages. This is driven by a few design criteria: 1) to avoid falling back to frame-driven processing; 2) to avoid processing the dynamic responses all the time; 3) to reduce the resource requirements for hardware implementation.

## III. Feature Map Construction and Neuron Competition

### A. Related Works

Primates' vision is extremely accurate and efficient in object categorization. This is ascribed to the ventral pathway processing in the visual cortex. It has been a hot topic for decades to model the feature representations in the visual cortex and design systems that mimic the cortical information processing. Until today, our understanding of the visual cortex has been boosted by massive research works in neurobiology and neurophysiology. The current theory of the cortical mechanism responsible for "rapid categorization" has been pointing to a hierarchical and mainly feedforward organization [29], [30]. This organization can provide hierarchical features of increasing complexity and invariance to size and position, making object categorization a multi-layered and tractable problem [31], [32].

Among many neurophysiologically plausible models of information processing in the visual cortex, HMAX, proposed by Riesenhuber and Poggio [22], is one of the most popular feedforward theories. HMAX extends the Hubel and Wiesel classical models of complex cells built from simple cells [33]. It summarizes the basic facts about the ventral visual stream ($V1$-$V2$-$V4$-$IT$). HMAX consists of a hierarchy of "S" layers and "C" layers ("S" and "C" follow the notation of Fukushima [34]). The "S" layer cells increase feature complexity by using linear weighted summation of the inputs; while "C" layer cells increase invariance through the nonlinear MAX operation.

The HMAX model was further extended by Serre et al. [23]. The whole feedforward architecture remained ($S1$-$C1$-$S2$-$C2$). The $S1$ and $C1$ layer correspond to the simple and complex cells in primary visual cortex $V1$, while $S2$ and $C2$ are roughly related to $V2$ and $V4$, respectively. The first two layers of the Serre model are mostly consistent with the original HMAX (differences exist in the adoption of Gabor filters [21] rather than difference of Gaussians [35]). The last two layers ($S2$ and $C2$) are where Serre et al. have made significant modifications. Learning is introduced at stage $S2$. A number of patches are randomly extracted from the $C1$ maps of the training images. Then for each image, the Gaussian radial basis function [36] is applied to the distance between $C1$ maps and patches, followed by a MAX operation to generate the shift- and scale- invariant $C2$ features. Promising results comparable to state-of-the-art computer vision systems have been achieved in object recognition tasks on natural images [23].

### B. Proposed Cortex-like Feature Extraction

Inspired by the aforementioned feedforward models of the cortical information processing (HMAX and Serre model), we propose a convolution-based network to extract features from motion events. For the purpose of simplicity, we only adopt a hierarchy of two layers ($S1$ and $C1$). Note that, in our model, we use a different MAX operation in $C1$ layer. The event-driven convolution with a forgetting mechanism is introduced in the $S1$ layer for continuously event-driven processing. The

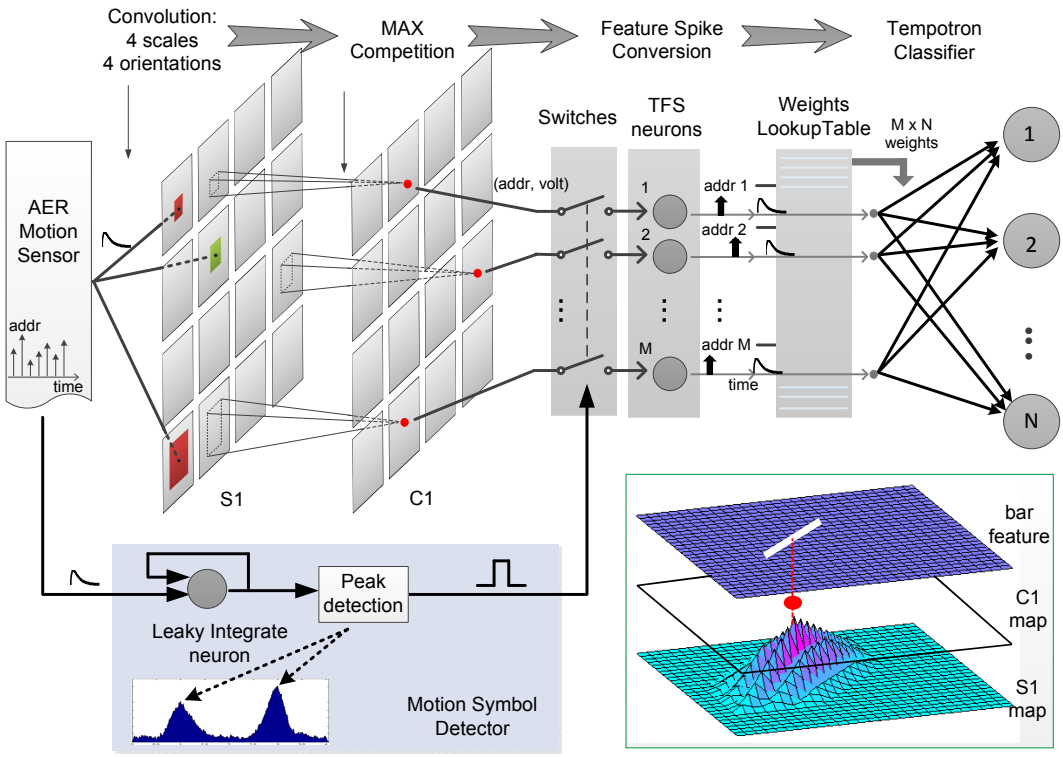



Fig. 2. The architecture of the proposed categorization system. The system consists of several building blocks, namely Convolution and Competition, Feature Spike Conversion, Motion Symbol Detector and Tempotron [26] classifier. Each input event is projected onto a group of dynamic $S1$ feature maps through event-driven convolution with a forgetting mechanism. $S1$ neurons compete with local neighbors through the MAX operation to strive for survival in $C1$ layer. Survived $C1$ neurons represent some salient bar features. The "Motion Symbol Detector" can detect a burst of events in a short time period and then take a snapshot of the dynamic $C1$ feature maps. Survived $C1$ neurons at that snapshot go through a small set of "TFS neurons" to be converted into spikes, which are further fed to a network of tempotron neurons for classification. We use the address of each feature spike to fetch its corresponding weight from the "Weights Lookup Table". The final categorization decision is made according to the output of tempotron neurons. The lower right part of the figure illustrates the concept of our feature extraction. One S1 map is shown at the bottom. The corresponding $C1$ map in the middle has only one survived neuron due to the MAX competition. The neuron's position is the same as that of the $S1$ peak. The survived $C1$ neuron represents a bar feature of a certain size and orientation at that position.

overall data flow can be summarized as $motion\ events \rightarrow S1\ maps \rightarrow C1\ maps$.

Simple cells ($S1$) are used to build feature selectivity. This is performed by convolving the input event with a network of Gabor filters [21]. Each filter models a neuron cell that has a certain size of receptive field and responds best to a basic feature of a certain orientation. Considering both the coverage of various sizes and orientations and the complexity of implementing the algorithm into hardware, we trade-off the network to 4 scales (ranging from 3 to 9, with a step length of 2) and 4 orientations ($0^o$, $45^o$, $90^o$ and $135^o$). The function of Gabor filter can be described as:

$$G(x,y) = \exp\left(-\frac{X^2+\gamma^2Y^2}{2\sigma^2}\right) \times \cos\left(\frac{2\pi}{\lambda}X\right) \qquad (1)$$

where $X = x\cos\theta + y\sin\theta$ and $Y = -x\sin\theta + y\cos\theta$. The filter parameters (orientation $\theta$, aspect ratio $\gamma$, effective width $\sigma$ and wavelength $\lambda$) have been well tuned in pioneering work [23], [37], and here we adopt a similar set of these parameters. The filter parameters are listed in Table I.

The event-driven convolution is illustrated in Fig. 3. When an input address event comes in, the convolution kernel is overlaid onto the response map at the position specified by the input event's address. Each element of the convolution kernel is then added to the corresponding original response.

TABLE I
PARAMETERS OF GABOR FILTERS

| filter sizes | 3 | 5 | 7 | 9 |
|---|---|---|---|---|
| effective width $\sigma$ | 1.2 | 2.0 | 2.8 | 3.6 |
| wavelength $\lambda$ | 1.5 | 2.5 | 3.5 | 4.6 |
| aspect ratio $\gamma$ | 0.3 | | | |
| orientations $\theta$ | $0; \frac{\pi}{4}; \frac{\pi}{2}; \frac{3\pi}{4}$ | | | |

The response map is thereby updated. In addition, in order to eliminate the impact of very old events on the current response map, a forgetting mechanism is adopted. Each pixel in the response map will decrease (or increase) towards the resting potential (usually set as 0) as time goes by. For implementation simplicity, we use a constant linear leakage.

It is in this way that we get 16 $S1$ convolution maps. For a certain feature (say a bar), each neuron in the 16 maps gives a response. $C1$ cells are obtained by performing the MAX-like operation over simple $S1$ units. The MAX operation is performed across the local neighborhood to find the center of the feature. As illustrated in Fig. 4, the neurons located in different-scale $S1$ maps have different receptive fields, such as 3×3, 5×5, 7×7 and 9×9. Each neuron competes with all the other neurons located within its receptive field. It can only survive and reach the $C1$ layer if it is the MAX in this area.

After the MAX operation, each survival neuron in $C1$ maps represents a feature, i.e., a line segment with a certain size and

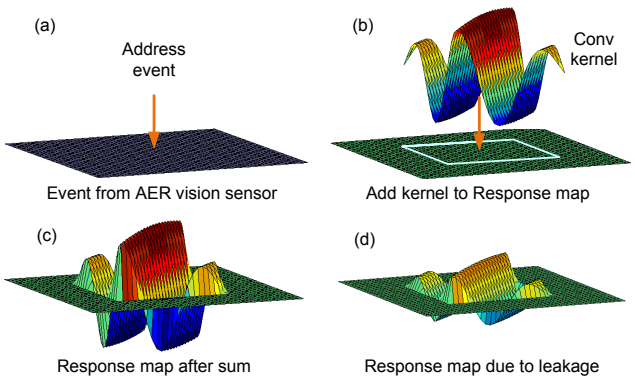



Fig. 3. Event-driven convolution with a forgetting mechanism. (a) The input event comes in. (b) The convolution kernel is overlaid onto the response map at the position specified by the event address. (c) shows the updated response map after adding the convolution kernel to the map. (d) shows the decayed response map after a while.

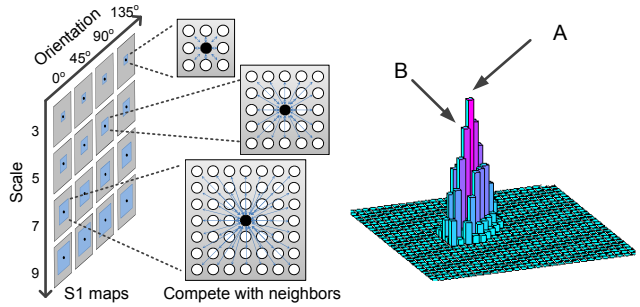



Fig. 4. MAX over local neighborhood. Neurons located in different-scale $S1$ maps have different receptive fields, such as 3×3, 5×5, 7×7 and 9×9. Each neuron competes with all the other neurons located within its receptive field. It can survive in $C1$ layer only when it is the MAX in this area. The right 3D figure shows an example of one $S1$ map, in which neuron A will survive in $C1$ layer but neuron B will not.

orientation (see the lower right part of Fig .2).

Note that in the proposed system, S1 and C1 are updated together for each input event. This process is illustrated in Fig. 5 using a $3 \times 3$ receptive field as an example. Each input address event from the sensor triggers the event-driven convolution and MAX operation. The input event's address specifies the operational window (the $3 \times 3$ red dots in Fig. 5). The convolution involves updating the leakage for these $3 \times 3$ S1 neurons and then adding the kernel to the S1 map. After convolution, each S1 neuron in the operational window (i.e. each red dot) competes with its $3 \times 3$ local neighbors (more exactly, $3 \times 3 - 1$ neighbors), and will only be fed (written) to the C1 map if it is the maximum among its neighbors. Note that the blue neurons in the figure need to be refreshed to make their values up to date (i.e. to update their leakage) before the MAX operation. This is because the lateral competition/inhibition has to be applied to the responses of the same timing. It does not make sense if a neuron compares its current response to another neuron's previous response.

## IV. Motion Symbol Detection and Feature Spike Generation

As mentioned above, in frame-driven sensors, a motion may be wrongly segmented into different frames due to the asynchronous nature of "motion" with respect to the time

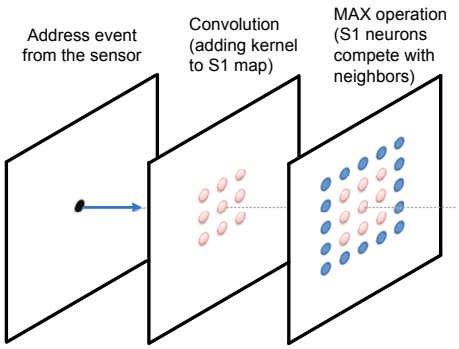



Fig. 5. Event-driven convolution and MAX operation using a $3 \times 3$ receptive field as an example. The input event's address specifies the operational window (the $3 \times 3$ red dots), where the convolution and the MAX operation are performed. The blue neurons in the figure need to be refreshed before the MAX operation.

"slice". On the other hand, in the event-driven system, the $C1$ feature maps are updated for each input event. Then when is a good time for classification? Note that the time interval between two consecutive events from the AER motion sensor can be very small (100 nanoseconds or less depending on the handshaking speed of the sensor). To avoid carrying out classification all the time, we propose a time domain clustering algorithm and introduce a "Motion Symbol Detector" module to the system.

The word "symbol" is borrowed from the terms used in speech recognition. The AER motion sensor only outputs a few noise events when capturing a static scene, whereas it generates a burst of output events when presented with moving objects. Here, we use the word "symbol" to denote one slice from such a burst of output events. The "Motion Symbol Detector" module consists of a leaky integration neuron and a peak detection unit. As illustrated in Fig. 6(a), each input event contributes a postsynaptic potential (PSP) to the neuron. For an input event received at time $t_i$, the normalized PSP kernel $K$ is defined as:

$$K(t-t_i) = V_0 \times (\exp(\frac{-(t-t_i)}{\tau_m}) - \exp(\frac{-(t-t_i)}{\tau_s})) \quad (2)$$

where $\tau_m$ and $\tau_s$ denote the two decay time constants of membrane integration and synaptic currents, respectively. For simplicity, $\tau_s$ is set to be $\tau_m/4$. $V_0$ normalizes PSP so that the maximum value of the kernel is 1.

The neuron's total potential is then obtained by superposition.

$$V(t) = \sum_{t_i} K(t-t_i) + V_{rest} \quad (3)$$

where $V_{rest}$ is the resting potential of the neuron, which is typically set as 0. A peak detection unit is thereafter applied on the neuron's total potential to locate temporal peaks. The principle of peak detection is as follows. For a certain timing $t_0$, the potential at that timing is considered as a peak if the following criterion is met:

$$V(t_0) \geq V(t);\ \forall\, t \in [t_0 - t_{SR}/2, t_0 + t_{SR}/2] \quad (4)$$

where $t_{SR}$ denotes the time span of the search range. This means the potential at time $t_0$ compares itself with all the potentials within its search range $[t_0 - t_{SR}/2, t_0 + t_{SR}/2]$. If

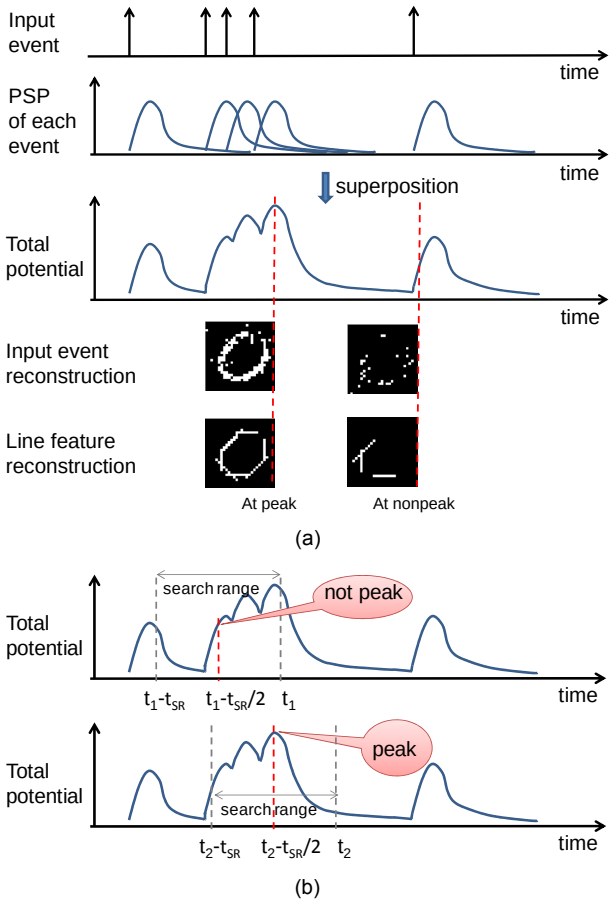



Fig. 6. Motion Symbol Detector. (a) Each input event generates a postsynaptic potential. The integration neuron's total potential can then be obtained by superposition. (b) Peak detection on the total potential. The potential at a certain time compares with other potentials in its temporal search range. If it is the maximum in the search range, it is considered as a peak; otherwise, it is not.

its potential is the maximum, it is then considered as a peak. If we denote $t_0 + t_{SR}/2$ as $t_c$ (current timing), then the potential at timing $t_0 = t_c - t_{SR}/2$ is considered as a peak if

$$V(t_c - t_{SR}/2) \geq V(t);\ \forall\, t \in [t_c - t_{SR}, t_c] \tag{5}$$

Fig. 6(b) illustrates two examples of peak detection. The upper one shows that $V(t_1)$ is not a peak since it is not the maximum among its search range $[t_1 - t_{SR}, t_1]$; while the lower one $V(t_2)$ is considered as a peak since (5) is met.

When a peak is identified, a pulse will be triggered to turn $ON$ the switches in Fig. 2. At that particular moment, $C1$ feature maps are fed to the following processing stages. Note that, in order to avoid the detection of very small peaks caused by background noise events, a threshold should also be applied. In addition, we can also add a refractory time to limit the frequency of the output pulse, i.e., the Motion Symbol Detector will remain halted for a while after a pulse has been generated.

$C1$ feature maps at a certain moment selected by the Motion Symbol Detector will be fed forward to a set of TFS neurons. Each TFS neuron is in charge of the conversion of one response. All TFS neurons work in parallel and should be triggered simultaneously. As stated in its name, each TFS neuron generates only one spike. The higher the response, the shorter the time to first spike. Let $m \times n$ denote the spatial resolution of the input AER motion events. After the convolution and MAX operation, each $C1$ feature map has the same size as the input resolution, and the number of all the responses in the $C1$ layer is thus $4 \times 4 \times m \times n$ (we use $4 \times 4$ filters). A fully parallel response-to-spike conversion would require $4 \times 4 \times m \times n$ TFS neurons, and thereby lead to huge hardware resource usage. Fortunately, due to the MAX surviving operation, only a small amount (refer to Section VI-A for the detail analysis) of neurons survive in the $C1$ layer (i.e., most $C1$ responses equal to zero). Instead of using all $C1$ responses, we only forward the survival neurons' responses (nonzero ones) together with their unique addresses (positions within 16 $C1$ maps). After conversion, the addresses of the original responses should be preserved and fed forward together with the corresponding spikes. In this way, the features are encoded back to AER spikes (also called spatiotemporal spikes). Each spike has a time stamp and an address. The time stamp is inversely proportional to the strength of the $C1$ response, and the address indicates the $C1$ neuron's position. Thereafter, we can use a bio-inspired spiking neural network named tempotron to make the categorization decision.

## V. Categorization by a Spiking Neural Network

In this section, we will illustrate how we perform classification on extracted feature spikes using a network of spiking neurons. Various models have been proposed in the literature to describe the dynamics of a single spiking neuron, such as the Leaky Integrate-and-Fire (LIF) model [38], Hodgkin-Huxley model [39], and Izhikevich model [40]. Among these models, LIF has the simplest structure and thus has been widely used. By combining multiple spiking neurons and storing weight information in synapses, we can construct a spiking neural network to learn and discriminate spatiotemporal spike patterns. Experimental studies in neuroscience have revealed a phenomenon namely spike-timing-dependent plasticity (STDP). The synaptic strength will be regulated by the relative timing of presynaptic and postsynaptic spike. Researchers have observed a long-term potentiation of synaptic strength (when a presynaptic neuron fires shortly before a postsynaptic neuron) and a long term depression (when a presynaptic neuron fires shortly after a postsynaptic neuron) [41]. STDP-based rules have been studied in [41]–[43] for the unsupervised learning of spike patterns. In addition to unsupervised STDP rules, supervised learning schemes, such as tempotron [26] and ReSuMe [44], have also been widely exploited. Compared to ReSuMe, which specifies a desired firing time, the tempotron learning rule only needs to label the status of firing or not, and thus it is more suitable for our real-world stimuli categorization tasks.

Tempotron is a model of supervised temporal learning that allows a spiking neuron to efficiently discriminate spatiotemporal spike patterns. It utilizes spike timing information and integrates postsynaptic potentials from afferent spikes with different addresses. These properties make tempotron by nature a perfect match for our extracted AER feature spikes.

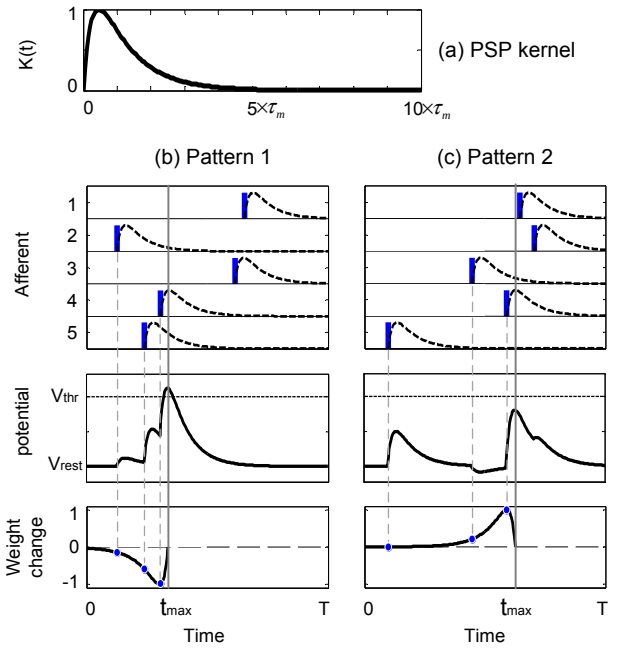



Fig. 7. The dynamics and the learning rule of the tempotron neuron. (a) shows the PSP kernel. (b) and (c) illustrate the operations of tempotron using two spatiotemporal patterns. The vertical thick bars stand for spikes, and the dash curve beside each bar denotes the PSP kernel generated by the corresponding spike. For $pattern1$ in (b), the total potential crosses the threshold, which means the neuron would fire for this input. If this is an error (the neuron should not fire for this input), then we find all the spikes before $t_{max}$ and decrease the weights of corresponding afferent synapses. $Pattern2$ in (c) does not make the neuron fire, if this is an error, the weights of those afferent synapses which have spikes before $t_{max}$ will be increased. Note that the curve of weight change is just the mirror of the PSP kernel.

### A. Tempotron Learning Rule

Tempotron uses the LIF neuron model. Each input spike initiates a PSP, which has a fast-rising and slow-decaying shape, as can be seen in Fig. 7 (a). The neuron's membrane potential is the weighted summation of the PSPs from all the input spikes:

$$V(t) = \sum_i \omega_i \sum_{t_i} K(t - t_i) + V_{rest} \tag{6}$$

where $\omega_i$ and $t_i$ are the synaptic efficacy (weight) and the firing time of the $i$th afferent synapse, respectively. $V_{rest}$ is the resting potential of the neuron. $K$ denotes the normalized PSP kernel as defined in (2).

If the neuron's potential is higher than a specified threshold, the neuron will fire an output spike and then reset its potential to the resting level. Fig. 7 (b) and (c) illustrate the dynamics and the learning rule of the tempotron neuron using two sample spike patterns. In Fig. 7 (b) the neuron fires since the membrane potential caused by $pattern1$ exceeds the threshold. After firing, the neuron shunts all the following input spikes and the potential gradually decreases to the resting level. In other words, the spikes arriving after the firing time have no impact on the postsynaptic potential any more. In Fig. 7 (c), the neuron does not fire since the membrane potential fails to cross the threshold.

The tempotron learning rule aims to train the weights so that the output neuron can fire or not according to its class label. If the neuron is supposed to fire (or not fire, on the other hand) but it actually fails to do so (or does fire, vice versa), then the weights should be modified in the following way. First, we find the peak potential during the effective period and label the corresponding time stamp as $t_{max}$. Second, we update the weights using the equation:

$$\Delta\omega_i = \begin{cases} \lambda \sum\limits_{t_i<t_{\max}} K(t_{\max} - t_i), & \text{if fail to fire} \\ -\lambda \sum\limits_{t_i<t_{\max}} K(t_{\max} - t_i), & \text{if fire wrongly} \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

where $\lambda$ denotes the learning rate.

For example, in Fig. 7 (c), the neuron fails to fire. If this is an error, we need to increase the weights of those afferents that have spikes arriving before $t_{max}$.

### B. Virtually Connected Tempotron Network

In principle, we need all the $C1$ responses for classification. In an $N$-class categorization task, we need $N$ tempotron neurons, with one for each category. Therefore, the tempotron network has $N$ outputs and $4 \times 4 \times m \times n$ inputs, where $m \times n$ denotes the resolution of each $C1$ map and $4 \times 4$ represents the number of $C1$ maps. The total number of weights (synapses) is $4 \times 4 \times m \times n \times N$. The size of the tempotron network is quite large. However, thanks to the beautiful nature of the spatiotemporal AER spikes and the MAX operation where only very few neurons survive after competition, we can achieve a virtually fully connected system by physically activating only a very small subset of the network. We use a LUT to store all the weights (refer to Fig. 2). Each feature spike is associated with an address, which can be used to access the LUT and fetch the corresponding weight.

During the training process, for an $N$-class categorization task, we label the $N$ tempotron neurons using the one-hot coding scheme. If a pattern belongs to the first class, then the first tempotron neuron's output is labeled 1 (which means it should fire), and all the other neurons' outputs are labeled 0 (not fire). During testing, the decision making for each input pattern is easy: just to check which neuron fires. In order to further improve the performance, we can use multiple neurons for each category [28]. Since the initial weights are set randomly, these neurons will have different weights after training. We then use the majority voting scheme to make the final decision: to check which category has the largest number of firing neurons.

Note that throughout our experiments the timings of all feature spikes fed to the tempotron network were normalized into the range of $[0, 1]$. In other words, the total time window of tempotron was set as $T = 1$. There was no time unit in our tempotron network. The membrane time constant $\tau_m$ in the tempotron was set as 0.1, the learning rate was set as $\lambda = 0.1$ and the number of tempotron neurons for each category was set as 10. These parameters are summarized in Table II.

## VI. Experimental Results

### A. On AER Posture Dataset

We have evaluated the performance of the proposed algorithm on real AER motion events captured from our dynamic

TABLE II
PARAMETERS FOR THE TEMPOTRON NETWORK

| | |
|---|---|
| Total time window $T$ of tempotron | 1 |
| Membrane time constant $\tau_m$ of tempotron | 0.1 |
| Learning rate $\lambda$ of tempotron | 0.1 |
| Number of tempotron neurons per category | 10 |

vision sensor. Our AER vision sensor uses logarithmic response pixel circuits, in which the output voltage is a logarithmic function of the amount of light striking a pixel. In addition, the circuits need a threshold to generate temporal difference motion events. The threshold is set to be 100 mv. We captured three human actions, namely bending to pick something up ($BEND$), sitting down and standing up ($SITSTAND$), and walking back and forth ($WALK$). Fig. 8 shows a few reconstructed sample images. Each row corresponds to one action; images are reconstructed from the AER motion events, using the aforementioned fixed time slice approach with a frame interval of 20 ms.

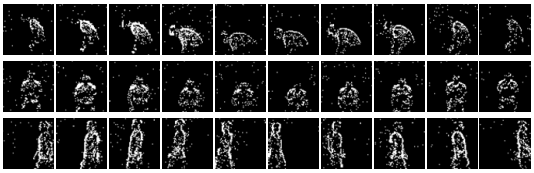


Fig. 8. Some reconstructed frames from our posture dataset. There are three kinds of human actions, each row shows an action.

Note that in the proposed system, we only focus on the detection and recognition of abrupt action transitions. We do not focus on the movements that happen at a constant speed since they can be inferred from the last action transition. The system performs recognition only when abrupt changes of body movement occur, such as suddenly bending down, sitting down, and suddenly changing the walking direction. Compared to constant movements, abrupt changes tend to generate more events in the sensor output, causing a burst effect. In our system, we use a Motion Symbol Detector to detect such a burst of events (i.e. a motion symbol) generated by abrupt changes, and then trigger the classification at those moments.

*1) Event-driven Centroid Computation:* The human's position may vary in the field of view, especially for the $WALK$ action. In this case, position invariance is necessary for the algorithm. This can be achieved by aligning the human posture silhouette to the center of the scene using the centroid information. The alignment process is simple. We can simply offset the address of each incoming motion event before it is fed to the $S1$ maps. An alternative way is to align the address of $C1$ feature spikes. The latter method involves less computation since the number of survived $C1$ neurons is very small.

Fig. 9 illustrates the event-driven centroid calculation. Similar to feature extraction, a map of leaky integration neurons are built. Each incoming event initiates a PSP kernel in the neuron specified by the event's address. The PSP kernel is the same as equation (2). Let $k_i$ denote the PSP kernel of the neuron with address $x_i$, and let $n$ denote the number of neurons. Using PSP potential as the weight of each address, we can easily calculate

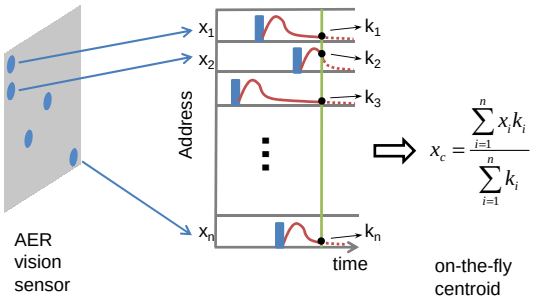



Fig. 9. Event-driven centroid calculation. Each incoming event initiates a PSP kernel for the corresponding neuron. The vertical bars represent events, and the fast-rising and slow-decaying curves depict PSP kernels. By using PSP potential as the weight of each address, the centroid can be easily calculated using the equation shown.

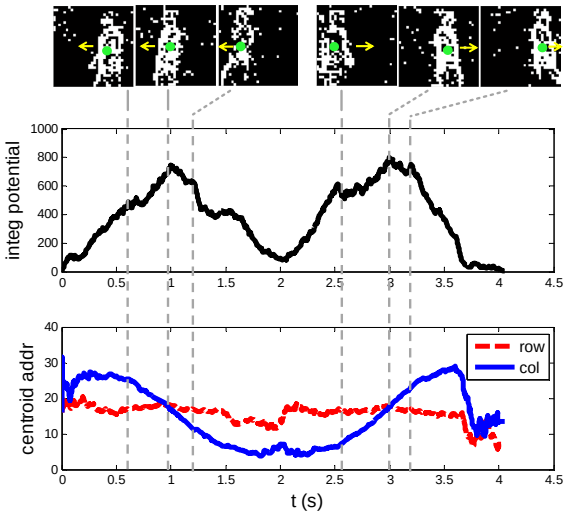



Fig. 10. Simulation results of event-driven centroid calculation on a stream of address events. A human is first walking to the left and then back to the right side. From bottom to top, the three rows respectively show the centroid curves, the potential curve of the integration neuron in "Motion Symbol Detector", and some images reconstructed at selected timing points. The green dot depicts the calculated centroid and the arrow means the moving direction. We can see that the centroid curves match the human action well.

the centroid address $x_c$ using the following equation:

$$x_c = \sum_{i=1}^{n} x_i k_i \Big/ \sum_{i=1}^{n} k_i \tag{8}$$

This process can be visualized in Fig. 10. A person is walking to the left and then back to the right side. The middle and lower figures illustrate the potential curve of the Motion Symbol Detector neuron and the event-driven centroid calculation results, respectively. The upper row shows the images reconstructed at several selected timings. The green dot highlights the centroid and the arrow indicates the moving direction. We can see that the calculated centroid follows the human's action quite well. Note that in this experiment, we assume there is only one person in the scene. If more people exist, we could resort to event-driven clustering algorithms (e.g. methods presented in [15], [16]) to obtain the position of each cluster (person).

*2) Parameters Selection:* Although very few neurons survive after MAX operation in $C1$, the number varies for different input scenarios. For future hardware implementation

consideration, we need to fix the number. We define a rule as follows: we first get the statistics of the survived neurons in $C1$ layer (e.g. mean $\mu$, standard deviation $\sigma$), and then the number of feature spikes (as well the number of "TFS neurons") is determined by:

$$M \geq \mu + 3 \times \sigma \tag{9}$$

The statistics of the three posture groups are shown in Table III.

TABLE III
NUMBER OF SURVIVED $C1$ NEURONS

| | Three Groups of Postures | | |
|---|---|---|---|
| | BEND | SITSTAND | WALK |
| mean ($\mu$) | 40 | 45 | 65 |
| std ($\sigma$) | 10 | 7 | 9 |

We can see that the number of survived neurons in $C1$ layer is small. According to (9), $M \geq 65+3\times 9 = 92$ and, therefore, using 100 TFS neurons is enough.

There are several parameters in our algorithm that need to be tuned according to specific applications. When using our AER vision sensor to observe walking humans, a minimum time of approximately 10-20 ms is needed to reconstruct a human-like silhouette. Therefore, we set the membrane time constant $\tau_m$ in the Motion Symbol Detector as $20ms$. In addition, transition actions like bending and sitting-down last no more than one second during our data collection, thus the search range parameter in the Motion Symbol Detector is set to be $t_{SR} = 1s$. The leakage rate in the event-driven convolution is set to be $1/\tau_m = 50s^{-1}$. These parameters are summarized in Table IV. We also provide the MATLAB codes of the proposed algorithm, which can be accessed from our lab website [45].

TABLE IV
PARAMETERS FOR POSTURE DATASET

| | |
|---|---|
| Time constant $\tau_m$ in Motion Symbol Detector | 20 ms |
| Search range $t_{SR}$ in Motion Symbol Detector | 1 s |
| Leakage rate in event-driven convolution | 50 $s^{-1}$ |

*3) Performance:* The posture dataset consists of 191 $BEND$, 175 $SITSTAND$ and 118 $WALK$ actions. We randomly pick out 80% of these actions for training and the others for testing. By repeating this evaluation process ten times, we get the average performance. For the training set, we obtain a correct rate of 100%; while for the testing set, the correct rate is 99.48% on average, with a standard deviation of 0.35%. We then ran the algorithm on a continuous event stream which is combined from all the testing actions. The result is shown in Fig. 11. The blue line represents the ground truth of classification, and the red circles denote the decisions made by our algorithm. We can see that the decisions match very well with the ground truth.

The proposed system was compared with two popular biologically inspired algorithms: the original HMAX scheme [22] and the model proposed by Serre et al. [23]. The MATLAB codes of these two models can be downloaded from the Web. For the original HMAX scheme, there are 256 C2 features. For the Serre model, 1000 patches are randomly extracted from the C1 layer of the training images and then used for template matching in layers S2 and C2. This leads to 1000 C2 features. Some of the patches extracted are blank due to sparse input data, but the ratio is very small (about 1.6%) and will not have a substantial impact on the results. Both the HMAX and the Serre model use the linear Support Vector Machine (SVM) for classification. To perform multi-class categorization on the three-class posture dataset, we implemented the One-Versus-All (OVA) SVM scheme using the LIBSVM library [46]. Since both the HMAX and the Serre model are designed to recognize 2D frames/images instead of events, our AER posture data cannot be used directly. We use the Motion Symbol Detector in our system to select motion symbols. Each motion symbol is a piece of events that took place before a peak timing (that is found by the Motion Symbol Detector). We reconstruct each motion symbol into an image. The reconstructed images are then fed to the HMAX and the Serre model for performance comparison. We randomly pick out 80% of these images for training and the others for testing. The testing results of these two models (averaged from 10 runs) are shown in Table V, where they are compared to the performance of the proposed algorithm. In this table we also report the simulation time taken by per motion symbol (or per image) running on a workstation with two Xeon E5 2.4GHz CPUs and 32GB RAM. The original HMAX scheme has the worst performance and the shortest simulation time due to its relatively simple computation. The proposed system has a performance that is comparable to the Serre model, but its simulation time is approximately 50% less than that of the Serre model.

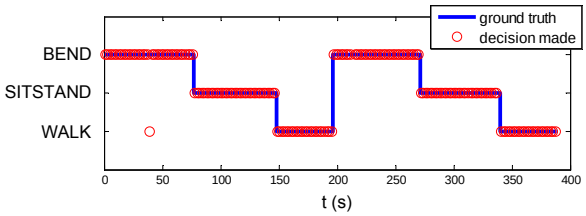



Fig. 11. The performance of the proposed algorithm on our posture dataset. All testing actions are connected one by one into a continuous event stream and then fed to the system for evaluation. We can see that the decisions made by our algorithm (red circles) match very well with ground truth (blue line).

TABLE V
PERFORMANCE COMPARISON ON THE AER POSTURE DATASET

| Models | Accuracy on Testing Set (mean $\pm$ standard deviation) | Simulation time per symbol/image |
|---|---|---|
| HMAX [22]+SVM | 78.65% $\pm$ 3.37% | 0.0285 s |
| Serre [23]+SVM | 99.84% $\pm$ 0.25% | 1.0071 s |
| this work | 99.48% $\pm$ 0.35% | 0.4991 s |

## B. On MNIST Image Dataset

We have further evaluated our algorithm on a standard handwritten digit dataset MNIST [47], which has ten digits (0-9) and 70,000 images in total. Fig. 12 shows some sample images of this dataset.

Our algorithm works on AER events instead of images and, therefore, we have to convert these pictures into events. We use a basic thresholding method to convert grey level MNIST images into binary images. The black pixels stand for the background and the white ones for the foreground. Address

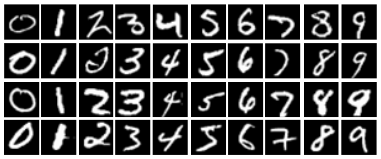


Fig. 12. Some sample images from MNIST hand-written digits dataset.

events are generated from all foreground pixels (digits), assuming that pixels fire at the same time and the events are driven out following a random priority. Each foreground pixel generates one event (note that for our algorithm, one event per pixel is enough, but multiple events per pixel as in rate coding also work fine.); each image generates about 200 events. The average length of the converted event stream is approximately 20 $\mu s$, with a mean interspike interval of 100 $ns$.

The membrane time constant $\tau_m$ and search range $t_{SR}$ in the Motion Symbol Detector are both set to be $20\mu s$. The leakage rate in event-driven convolution is set as $1/\tau_m = 5 \times 10^4 s^{-1}$. These parameters are summarized in Table VI.

TABLE VI
PARAMETERS FOR MNIST DATASET

| | |
|---|---|
| Time constant $\tau_m$ in Motion Symbol Detector | 20 $\mu s$ |
| Search range $t_{SR}$ in Motion Symbol Detector | 20 $\mu s$ |
| Leakage rate in event-driven convolution | $5 \times 10^4$ $s^{-1}$ |

The MNIST dataset has 60,000 images in the training set and 10,000 images in the testing set. Our algorithm achieved success rates of approximately $99.36\%$ for the training set and $91.29\%$ for the testing set. To emulate the noise of the AER sensor output, we also added salt and pepper noise to the MNIST images before converting them into AER events. The results are summarized in Table VII. One can see that the proposed algorithm maintains competitive accuracy even when noise is added.

TABLE VII
PERFORMANCE ON MNIST DATASET WITH DIFFERENT NOISE DENSITY

| Noise density | 0 | 0.05 | 0.1 | 0.2 | 0.3 | 0.5 |
|---|---|---|---|---|---|---|
| Training set (%) | 99.36 | 98.31 | 97.60 | 95.71 | 92.14 | 78.01 |
| Testing set (%) | 91.29 | 89.60 | 87.36 | 81.19 | 73.76 | 47.10 |

Note that the proposed event-driven categorization system is designed mainly for processing the motion events from the AER temporal contrast vision sensor. The purpose of testing the proposed system on the MNIST dataset is not to compete with state-of-the-art algorithms but to demonstrate that the proposed system can work not only on raw AER data but also on images (with a preprocessing step to convert images into AER events, but finding an optimized conversion method is out of the scope of this paper). Since the algorithm is not designed for the recognition of images, it has a relatively lower performance than other highly optimized frame-driven algorithms.

### C. On MNIST-DVS Dataset

Our algorithm was also evaluated on the actual event-based MNIST dataset (i.e. the MNIST-DVS dataset) [48].

The MNIST-DVS dataset consists of a set of Dynamic Vision Sensor (DVS) recordings of different handwritten digits. A total of 10,000 original $28 \times 28$ pixel handwritten digit images from the MNIST were enlarged to three different scales (scale-4, scale-8 and scale-16) using smoothing interpolation algorithms. Each scaled digit was then displayed on a liquid crystal display monitor with slow movements and a $128 \times 128$ pixel AER DVS [10] was used to record the moving digit.

The proposed algorithm was evaluated on scale-4 of the MNIST-DVS dataset. There are totally 10,000 recordings for the scale-4 digits. Each recording has a time length about 2 s. A digit in scale-4 roughly fits into a $28 \times 28$ patch. To provide a proper $28 \times 28$ input scene, we used an event-driven cluster-tracking algorithm [16] to track the moving digits from the original $128 \times 128$ DVS recordings. The generated $28 \times 28$ event streams were then sent to our algorithm for evaluation. $90\%$ of them were randomly selected for training and the others were used for testing. This evaluation process was repeated ten times to obtain the average performance. The accuracy was $99.13\% \pm 0.02\%$ for the training set and $88.14\% \pm 0.70\%$ for the testing set. We also examined the impact of the time length of recordings on the accuracy. We evaluated 100 ms, 200 ms, 500 ms and full length (about 2 s). The results are listed in Table VIII. As expected, the accuracy increases when longer recordings are used. The parameters used for the MNIST-DVS dataset are shown in Table IX.

TABLE VIII
PERFORMANCE ON THE MNIST-DVS DATASET

| Used time length for each recording | Training Accuracy (mean $\pm$ std) (%) | Testing Accuracy (mean $\pm$ std) (%) |
|---|---|---|
| 100 ms | 98.86 $\pm$ 0.09 | 76.86 $\pm$ 1.27 |
| 200 ms | 99.08 $\pm$ 0.04 | 82.61 $\pm$ 1.17 |
| 500 ms | 99.19 $\pm$ 0.02 | 85.89 $\pm$ 0.84 |
| full (about 2 s) | 99.13 $\pm$ 0.02 | 88.14 $\pm$ 0.70 |

TABLE IX
PARAMETERS FOR MNIST-DVS DATASET

| | |
|---|---|
| Time constant $\tau_m$ in Motion Symbol Detector | 20 ms |
| Search range $t_{SR}$ in Motion Symbol Detector | 30 ms |
| Leakage rate in event-driven convolution | 10 $s^{-1}$ |

## VII. DISCUSSION

### A. About Feature Spike Conversion

This work aims to develop a spike-based categorization framework, which consists of an AER vision sensor and a vision processing system. The AER events from the sensor fit the spiking neurons, but they cannot be directly fed to the classifier. Our sensor currently only performs pixel-level motion detection. Due to the hardware limitation, no high level feature extraction (such as corner and edge detection) is performed in the sensor. Therefore, a feature extraction unit is still required in the vision processing system.

In our case, convolution and MAX operation are used to model S1 and C1 cells in the primary visual cortex. The C1 responses are converted into spikes through a set of TFS neurons.

Converting C1 values into spikes can provide benefits for the computation. To perform the comparison, we applied the

OVA multiclass SVM directly on C1 values, and compared it with our method (i.e. C1 values → spikes → tempotron). The results are shown in Table X. It can be seen that the tempotron consumes much less simulation time than SVM while still maintaining competitive accuracy.

TABLE X
PERFORMANCE COMPARISON OF SVM AND TEMPOTRON ON C1 VALUES

| Dataset | Classifier | Test Accuracy (%) | Simulation Time (s) | |
|---|---|---|---|---|
| | | | Training | Testing |
| Posture | SVM | 99.22 | 3.02 | 3.79 |
| | tempotron | 99.48 | 0.64 | 0.03 |
| MNIST | SVM | 95.01 | 32827 | 8649 |
| | tempotron | 91.29 | 332 | 11 |

Spike-domain computation is very efficient since the computation only takes place when there is an input spike, whereas in the conventional time-step approach the computation has to be performed at every time-step. In the case of the tempotron, the computation is very simple. A tempotron neuron's membrane potential is updated at the timing of each input spike. The total computation involved in the tempotron network is linear with respect to the number of input spikes and the number of tempotron neurons (see the Appendix for details). Also, note that the time window $T$ is only a way of feature normalization during the conversion from C1 responses to spike timings. This number has no relationship with the computation latency of the classification.

### B. About Peak Detection in the Motion Symbol Detector

The peak detection in the Motion Symbol Detector introduces a $t_{SR}/2$ delay. If this delay is not affordable in a specific application, this problem can be addressed by using the thresholding method instead of the current peak searching. When the total potential reaches the threshold, an ON pulse is triggered. The peaks are where we are actually interested. By using the thresholding method, we can approximately select the timings that are close to the peaks. Thresholding can avoid the delay, but its corresponding recognition performance is a bit lower than that of peak detection, yet still comparable if the threshold is properly tuned. The reason we use peak detection instead of thresholding in the current system is that the peak detection would convey our idea in a better way.

### C. About Spatio-temporal Information in Feature Maps

In conventional frame-driven synchronous systems, recognition is performed on every frame. Each frame contains only spatial but little temporal information (light signal integration during the exposure time makes it a bit temporal to a limited extent). For robust human action recognition, longer temporal information is required. Jhuang et al. [49] propose a biologically inspired action recognition system which extends the $C2$ spatial shape features of Serre et al. [23] to be spatio-temporal. The original 2D Gabor filters are added one more dimension (temporal) to their receptive fields and, therefore, the generated $S1$ (and also $C1$) maps are three-dimensional. A set of spatio-temporal patches are randomly extracted from the $C1$ layer of frames of training videos. Thereafter, $S2$ feature maps are calculated through template matching between $C1$ maps and each patch. A global MAX operation across all positions is taken for each $S2$ map to generate $C2$ units. The spatio-temporal $C2$ features achieve very impressive results on various action datasets. However, similar with the Serre model, the algorithm proposed by Jhuang et al. is designed for conventional cameras.

Assume the frame rate of the camera is 30 frames per second, and then each frame captures the information that happens within 33 ms. Jhuang's feature extraction algorithm considers multiple (9) frames at a time to introduce motion analysis. It processes the information that occurs within $9 \times 33 = 297$ ms.

Our system is based on an AER dynamic vision sensor. Each pixel in the sensor can monitor the relative change of light intensity in real time (by direct light differencing circuits). There is no exposure time involved and thus there is no signal integration in the detection stage. Each address event from the sensor is sent to a batch of Gabor filters, and convolution is performed on the fly. The convolution response maps ($S1$) are updated for each input event. In conventional frame-driven convolution, the $S1$ maps are reset and recalculated for every frame. However, in the event-driven case, the $S1$ maps cannot be reset. The responses should be integrated all the time. We introduce a leakage mechanism to forget the impact of very old address events. Due to the non-resetting convolution and the leakage mechanism, the generated $S1$ maps naturally contain temporal information. The range of this temporal dimension can be adjusted by varying the leakage rate. Therefore, in short, our $S1$ maps do contain temporal information, which is equivalent to the concept of multiple frames, but in an asynchronous way.

## VIII. CONCLUSION

This paper presents an event-driven feedforward categorization system which processes data from an AER temporal contrast vision sensor. Sparse Features are extracted by using hierarchical maps of leaky integration neurons, which is inspired by a model of object categorization in the primate visual cortex. The features are then encoded into a limited number of spikes through a set of TFS neurons. A virtually connected tempotron network efficiently discriminates the spatiotemporal feature spike patterns. Two types of event-driven co-processing are also explored, namely "Motion Symbol Detector" and the centroid calculation. The overall system has been evaluated by extensive simulations and comparisons. The experimental results have shown that the proposed event-driven system reduces the computation (approximately $50\%$ less in terms of simulation time) and that it maintains competitive accuracy even when noise is added.

## APPENDIX A
## COMPUTATION ANALYSIS OF THE EVENT-DRIVEN CONVOLUTION AND MAX OPERATION

The computational process of the event-driven convolution and MAX operation (for a $3 \times 3$ receptive field as shown in Fig. 5) can be summarized as follows.

For each input address event:

1) Read $5 \times 5$ S1 values (the red and blue dots in Fig. 5) from RAM into registers. Note that each S1 value records not only the response but also the time of last update. This involves $5 \times 5$ memory read access.
2) Calculate the time difference between the current time (the timestamp of the input event) and each S1 neuron's last update time. This includes $5 \times 5$ subtractions.
3) Calculate the leakage for these $5 \times 5$ neurons by multiplying each time difference with the constant leakage rate. This involves $5 \times 5$ multiplications.
4) Subtract the leakage from the original responses. This involves $5 \times 5$ subtractions.
5) Add the convolution kernel to the operational window (the $3 \times 3$ red neurons in Fig. 5). This involves $3 \times 3$ additions.
6) MAX operation for each S1 neuron in the operational window. Each neuron competes with its $3 \times 3 - 1$ neighbors. This step totally involves $(3 \times 3 - 1) \times (3 \times 3)$ comparisons.
7) Write updated S1 values (responses as well as the update time) into RAM. Write C1 values (i.e. S1 local MAX if any, responses as well as the update time) into RAM. This involves up to $5 \times 5 + 3 \times 3$ memory write access.

The total computation involved for each input event is therefore:

- Number of memory access $\leq 5 \times 5 + (5 \times 5 + 3 \times 3)$.
- Number of multiplication = $5 \times 5$.
- Number of addition/subtraction/comparison = $5 \times 5 + 5 \times 5 + 3 \times 3 + (3 \times 3 - 1) \times (3 \times 3)$.

To generalize it, let the receptive field be denoted as $s \times s$ ($s$ = 3, 5, 7, 9), the computation involved for each input event is:

- Number of memory access $\leq$
$$\sum\nolimits_s [(2s-1)^2 + (2s-1)^2 + s^2]$$
- Number of multiplication = $\sum_s [(2s-1)^2]$.
- Number of addition/subtraction/comparison =
$$\sum\nolimits_s [(2s-1)^2 + (2s-1)^2 + s^2 + (s^2-1) \times s^2]$$

## Appendix B
## Computation Analysis of the Tempotron Network

The computation of tempotron is simple. A tempotron neuron is in fact a Leaky Integrated-and-Fire neuron. Its membrane potential is the integration results of each input spike. In other words, each input spike updates the tempotron neuron's potential. The neuron's potential is updated when and only when an input spike comes in. The total computation of a tempotron neuron is linear with respect to the number of input spikes. In what follows we analyze the computation involved in a tempotron neuron.

The PSP kernel is the difference of two exponential decays, as can be seen in Equation (2) and Fig. A1. It is the difference between a slower exponential decay (with a time constant of $\tau_m = 0.1$) and a faster exponential decay (with a time constant of $\tau_s = \tau_m/4 = 0.025$). Note that $V_0$ is a normalization coefficient that makes the peak value of the final PSP kernel to be 1.

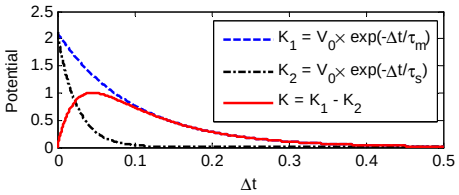



Fig. A1. PSP kernel of the tempotron. Difference of two exponential decays.

The total potential of the tempotron neuron is the weighted summation of the PSP kernels from all input spikes. Here, the PSP kernel is the difference of two exponential decays. For better illustration, let us first look at the case of a single exponential decay PSP kernel.

$$K_1(\Delta t) = V_0 \times \exp(-\Delta t/\tau_m)$$

Note that the weighted summation of exponentially decaying PSPs is equivalent to implementing an exponential decay directly on the neuron membrane potential. Therefore, the update process of a tempotron neuron's potential for each input spike is as follows.

1) Calculate the time difference $\Delta t$ between the current time (i.e. the timestamp of the input spike) and the last update time. This involves one subtraction. Note that we also need to overwrite the last update time using the current time, which is only an assignment and involves no computation.
2) Refresh the neuron's membrane potential by implementing the exponential decay on it using the equation
$$V_1 \leftarrow V_1 \times \exp(-\Delta t/\tau_m)$$
where $V_1$ denotes the membrane potential. Note that $\exp(-\Delta t/\tau_m)$ can be pre-calculated and stored in a lookup table (LUT). Assume that the time resolution of the LUT is $dt$, the address that is used to access the LUT is then $\Delta t \times (1/dt)$, where $(1/dt)$ is a pre-calculated coefficient. So, this step involves one LUT access and two multiplications (one for the LUT address calculation and one for the potential decay).
3) Add $\omega_i \times V_0$ to the neuron's membrane potential.
$$V_1 \leftarrow V_1 + \omega_i \times V_0$$
where $\omega_i$ is the weight associated with the channel (address) of the input spike. This step involves one multiplication and one addition.

For the second exponential decay PSP kernel $K_2(\Delta t) = V_0 \times \exp(-\Delta t/\tau_s)$, step 2) and 3) need to be replicated. If we put these two steps together, it will be

$$V_2 \leftarrow V_2 \times \exp(-\Delta t/\tau_s) + \omega_i \times V_0$$

where $V_2$ is the membrane potential that is corresponding to PSP kernel $K_2$.

The final membrane potential of the tempotron neuron is then:

4) $V = V_1 - V_2$. This step involves one subtraction.

In the end, we need to:

5) Check whether the membrane potential is larger than the threshold or not. This involves one comparison.

To sum it up, using the difference of two exponential decays as the PSP kernel, the computation for each input spike is:

- Number of LUT access = $1 \times 2 = 2$
- Number of multiplication = $(2 + 1) \times 2 = 6$
- Number of addition/subtraction/comparison = $1+1 \times 2+ 1 + 1 = 5$

where the "$\times 2$" is due to two exponential decays.

Note that the peak of the final PSP kernel $K$ does not happen at the time of the input spike; instead it has a delay from the input spike timing. This can be seen in Fig. A1. The PSP peak does not happen at $\Delta t = 0$. We found that it roughly occurs at $\Delta t \approx 0.462 \times \tau_m$ (for $\tau_s = \tau_m/4$). To obtain the correct output (firing or not), we need to check the firing status at the PSP peak time in addition to the input spike time. This can be easily done by generating a dummy spike for each input spike. The dummy spike has a timestamp of $0.462 \times \tau_m + t$, where $t$ denotes the input spike timing. For the dummy spike, the computation does not involve step 3). The computation for each dummy spike is:

- Number of LUT access = $1 \times 2 = 2$
- Number of multiplication = $2 \times 2 = 4$
- Number of addition/subtraction/comparison = $1+1+1 = 3$

In our system, the number of spikes fed to tempotron is less than 100. Let us use the worst case (i.e. 100 input spikes) for the calculation. Note that each input spike also generates a dummy spike. The computation involved in a tempotron neuron is therefore:

- Number of LUT access = $(2 + 2) \times 100 = 400$
- Number of multiplication = $(6 + 4) \times 100 = 1,000$
- Number of addition/subtraction/comparison = $(5 + 3) \times 100 = 800$

For a three-class categorization task, we need three tempotron neurons for the one-hot coding scheme. In addition, we use multiple neurons (10 in our experiment) for each category. Therefore, there are 30 tempotron neurons in total. The total computation involved in the classification is:

- Number of LUT access = $400 \times 30 = 12,000$
- Number of multiplication = $1000 \times 30 = 30,000$
- Number of addition/subtraction/comparison = $800 \times 30 = 24,000$

Note that the total computation is linear with respect to the number of input spikes and the number of tempotron neurons. Beside this, the final computation time depends on how fast the hardware (CPU or ASIC/FPGA) performs each addition/subtraction/comparison, multiplication, and LUT access.

The current PSP kernel used in the tempotron neuron (i.e. the difference of two exponential decays) models the transmission delay from a presynaptic spike to the postsynaptic potential. Considering the transmission delay makes the neuron model more biologically plausible. However, this delay can usually be ignored in most practical engineering problems. That is, to use only a single exponential decay PSP kernel. In that case, the computation involved in a tempotron neuron for each input spike will be even less, and no dummy spike is needed.

## APPENDIX C
## COMPARISON OF HMAX-LIKE MODELS AND THE PROPOSED ALGORITHM

Compared to HMAX-like models (such as the Serre Model), the proposed method has lower complexity. For feature extraction, HMAX-like models have four layers of processing, i.e. S1-C1-S2-C2. The computation of layers S2 and C2 requires a large number of C1 patches to be stored in memory and used for template matching. Our algorithm only has the S1 and C1 layers. It does not need to extract and store patches. In addition, it has fewer filters in the S1 layer (4 scales x 4 orientations, compared to 16 scales x 4 orientations in the Serre model). The comparison is summarized in Table A1. As for the classifier, HMAX-like models use SVM, whereas our algorithm utilizes a tempotron spiking neural network, which is much faster (as shown in Section VII-A).

TABLE A1
COMPARISON OF HMAX-LIKE MODELS AND THE PROPOSED ALGORITHM

| | HMAX-like models | this work |
|---|---|---|
| Feature Extraction | S1-C1-S2-C2 | S1-C1 |
| S1 filters | 16 scales | 4 scales |
| C1 patches | 1000 | Nil |
| Classifier | SVM | Tempotron |

## APPENDIX D
## SOME DESIGN CONSIDERATIONS

The proposed system is designed to process the continuous event stream from an AER vision sensor. S1 and C1 are updated for each input event and the following part is triggered when the Motion Symbol Detector finds a peak. This architecture is justified as follows.

### *A. S1 must be running all the time together with the Motion Symbol Detector*

Each neuron in the S1 layer is a Leaky Integration cell (no fire), so is the neuron used in Motion Symbol Detector. Each incoming event is integrated immediately by all these neurons (the S1 neurons and the neuron in Motion Symbol Detector) and then discarded. There is no need to store input events in our system.

The potentials (responses) of these Leaky Integration neurons are the integration results of past events. A burst of input events (a Motion Symbol) will cause a peak potential in the Motion Symbol Detector. If the S1 integration begins after a peak has been detected, it will be too late, since the events have already passed.

### *B. C1 can be placed before or after the switches in Fig.2*

Placing C1 after the switches (i.e. triggering the MAX operation only when the Motion Symbol Detector finds a peak) will reduce the computation for each event. In fact, this is what we did in the MATLAB simulation of categorizing offline AER

data. However, this method will result in more computation after peak detection. Note that in this method, the MAX operation needs to be performed for whole-map S1 neurons. Also note that in hardware implementations, S1 responses typically need to be stored in RAMs instead of registers due to the large amount of data they contain. Because of the huge volume of RAM access involved, the MAX operation for whole-map S1 neurons will therefore take a relatively long period of time to compute, causing a relatively long delay from peak detection to output decision.

On the other hand, placing C1 before the switches will reduce the computation after peak detection. In this method, the C1 layer is updated together with the S1 layer using the aforementioned event-driven convolution and MAX operation, which only affects a small window for each input event. Note that the window MAX operation is performed directly using the S1 responses that were already loaded into registers from the RAM during the convolution. This avoids repeated S1 RAM access. After peak detection, there is no longer any need to perform a whole-map MAX operation; this method therefore obviates the large amount of RAM access that would be required in whole-map MAX operation, resulting in a relatively short delay from peak detection to output decision.

The placing of C1 before or after the switches can also be compared in terms of the total computation process from an input event to an output decision (assuming the input event turns the switches ON). The processes involved when placing C1 before and after the switches are, respectively: Event $\rightarrow$ convolution $\rightarrow$ window MAX $\rightarrow$ spike conversion $\rightarrow$ tempotron, and Event $\rightarrow$ convolution $\rightarrow$ whole-map MAX $\rightarrow$ spike conversion $\rightarrow$ tempotron. One can see that placing C1 before switches will reduce the delay from the input event to the output decision. Therefore, we consider it preferable to place C1 before the switches and update C1, together with S1, using the event-driven convolution and MAX.

## ACKNOWLEDGMENTS

This work was supported by ACRF Project (M4010343.040).

## REFERENCES

[1] G. Indiveri, S.-C. Liu, T. Delbruck, and R. Douglas, "Neuromorphic systems," *Encyclopedia of Neuroscience*, pp. 521–528, 2009.

[2] S.-C. Liu and T. Delbruck, "Neuromorphic sensory systems," *Current Opinion in Neurobiology*, vol. 20, pp. 288–295, 2010.

[3] E. Chicca *et al.*, "A multi-chip pulse-based neuromorphic infrastructure and its application to a model of orientation selectivity," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, pp. 981–993, May 2007.

[4] R. Vogelstein *et al.*, "A multi-chip neuromorphic system for spike-based visual information processing," *Neural Computation*, 2007.

[5] G. Indiveri *et al.*, "Neuromorphic silicon neuron circuits," *Frontiers in Neuroscience*, vol. 5, no. 73, 2011.

[6] T. Hamilton *et al.*, "An active 2-d silicon cochlea," *IEEE Trans. Biomed. Circuits Syst.*, vol. 2, no. 1, pp. 30–43, Mar. 2008.

[7] V. Chan, S.-C. Liu, and A. van Schaik, "AER EAR: A matched silicon cochlea pair with address event representation interface," *IEEE Trans. Circuits Syst. I: Reg. Papers*, vol. 54, no. 1, pp. 48–59, 2007.

[8] K. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 47, no. 5, pp. 416–434, 2000.

[9] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 $\times$ 128 120 dB 15 $\mu$s latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, Feb. 2008.

[10] T. Serrano-Gotarredona and B. Linares-Barranco, "A 128 $\times$ 128 1.5% contrast sensitivity 0.9% fpn 3 $\mu$s latency 4 mw asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 3, pp. 827–838, Mar. 2013.

[11] B. Zhao *et al.*, "A 64x64 CMOS image sensor with on-chip moving object detection and localization," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, pp. 581–588, 2012.

[12] S. Chen, W. Tang, X. Zhang, and E. Culurciello, "A 64x64 pixels UWB wireless temporal-difference digital image sensor," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 20, pp. 2232–2240, 2012.

[13] Java AER Open Source Project. [Online]. Available: http://jaer.wiki.sourceforge.net/

[14] X. Zhang and S. Chen, "A high-speed-pass asynchronous motion detection sensor," in *IEEE Intern. Symp. Circuits Syst.*, May 2013.

[15] M. Litzenberger *et al.*, "Embedded vision system for real-time object tracking using an asynchronous transient vision sensor," in *12th Signal Processing Education Workshop*, Sept. 2006, pp. 173–178.

[16] T. Delbruck and P. Lichtsteiner, "Fast sensory motor control based on event-based hybrid neuromorphic-procedural system," in *IEEE International Symposium on Circuits and Systems*, May 2007, pp. 845–848.

[17] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jimenez, C. Serrano-Gotarredona, J. Perez-Carrasco, B. Linares-Barranco, A. Linares-Barranco, G. Jimenez-Moreno, and A. Civit-Ballcels, "On real-time AER 2D convolutions hardware for neuromorphic spike based cortical processing," *IEEE Trans. Neural Networks*, vol. 19, no. 7, pp. 1196–1219, July 2008.

[18] R. Serrano-Gotarredona *et al.*, "CAVIAR: A 45k neuron, 5M synapse, 12G connects/s AER hardware sensory processing learning actuating system for high-speed visual object recognition and tracking," *IEEE Trans. Neural Networks*, vol. 20, no. 9, pp. 1417–1438, Sep. 2009.

[19] J. Perez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen, and B. Linares-Barranco, "Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate-coding. Application to feed forward ConvNets," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, pp. 2706–2719, Nov. 2013.

[20] Y. LeCun *et al.*, "Convolutional networks and applications in vision," in *IEEE Intern. Symp. Circuits Syst.*, 2010, pp. 253–256.

[21] J. G. Daugman, "Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters," *Journal of the Optical Society of America A*, vol. 2, no. 7, pp. 1160–1169, Jul 1985.

[22] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex," *Nature Neuroscience*, vol. 2, pp. 1019–1025, 1999.

[23] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust object recognition with cortex-like mechanisms," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 3, pp. 411–426, 2007.

[24] S. Chen, P. Akselrod, B. Zhao, J. Perez-Carrasco, B. Linares-Barranco, and E. Culurciello, "Efficient feedforward categorization of objects and human postures with address-event image sensors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 2, pp. 302–314, Feb. 2012.

[25] B. Zhao and S. Chen, "Realtime feature extraction using max-like convolutional network for human posture recognition," in *IEEE Intern. Symp. Circuits Syst.*, May 2011, pp. 2673–2676.

[26] R. Gutig and H. Sompolinsky, "The tempotron: a neuron that learns spike timing-based decisions," *Nature Neuroscience*, vol. 9, no. 3, pp. 420–428, Feb. 2006.

[27] J. Hu, H. Tang, K. C. Tan, H. Li, and L. Shi, "A spike-timing based integrated model for pattern recognition," *Neural Computation*, vol. 25, no. 2, pp. 450–472, 2013.

[28] Q. Yu, H. Tang, K. C. Tan, and H. Li, "Rapid feedforward computation by temporal encoding and learning with spiking neurons," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 24, pp. 1539–1552, Oct. 2013.

[29] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," *Nature*, vol. 381, pp. 520–522, june 1996.

[30] S. J. Thorpe and M. Fabre-thorpe, "Seeking categories in the brain," *Science*, vol. 291, pp. 260–263, 2001.

[31] T. Serre and M. Riesenhuber, "Realistic modeling of simple and complex cell tuning in the HMAX model, and implications for invariant object recognition in cortex," AI Memo 2004-017/CBCL Memo 239, MIT, Tech. Rep., July 2004.

[32] T. Serre, A. Oliva, and T. Poggio, "A feedforward architecture accounts for rapid categorization," *Proceedings of The National Academy of Sciences*, vol. 104, pp. 6424–6429, 2007.

[33] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architectures in cats visual cortex," *Journal of Physiology*, vol. 160, pp. 106–154, 1962.

[34] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[35] D. Marr and E. Hildreth, "Theory of edge detection," *Proceedings of the Royal Society of London. Series B, Biological Sciences*, vol. 207, no. 7, pp. 1160–1169, Jul 1985.

[36] M. D. Buhmann and M. D. Buhmann, *Radial Basis Functions*. New York, USA: Cambridge University Press, 2003.

[37] T. Serre, "Learning a dictionary of shape-components in visual cortex: Comparison with neurons, humans and machines," Ph.D. dissertation, MIT, Apr. 2006.

[38] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*, 1st ed. Cambridge University Press, Aug. 2002.

[39] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, pp. 500–544, Aug. 1952.

[40] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, Nov. 2003.

[41] T. Masquelier *et al.*, "Spike timing dependent plasticity finds the start of repeating patterns in continuous spike trains," *PLoS ONE*, vol. 3, no. 1, 2008.

[42] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS Comput Biol*, vol. 3, no. 2, Feb. 2007.

[43] T. Masquelier *et al.*, "Competitive STDP-based spike pattern learning," *Neural computation*, vol. 21, no. 5, pp. 1259–1276, May 2009.

[44] F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with ReSuMe: Sequence learning, classification, and spike shifting," *Neural Computation*, vol. 22, no. 2, pp. 467–510, Feb. 2010.

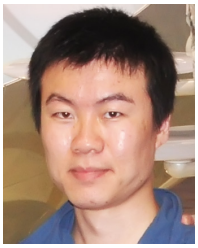
[45] http://www.ntu.edu.sg/home/eechenss/Research/2013-AER-System/code.zip.

[46] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 27, pp. 27:1–27:27, Apr. 2011.

[47] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[48] T. Serrano-Gotarredona and B. Linares-Barranco, "The MNIST-DVS database," URL: http://www2.imse-cnm.csic.es/caviar/MNISTDVS.html.

[49] H. Jhuang *et al.*, "A biologically inspired system for action recognition," in *IEEE Intern. Conf. Computer Vision*, Oct. 2007, pp. 1–8.

**Bo Zhao** (M'11) received the B.Eng. and M.Eng. degree in Electronic Engineering from Beijing Jiaotong University, Beijing, China, in 2007 and 2009, respectively. He received the Ph.D. degree in Electrical and Electronic Engineering from Nanyang Technological University, Singapore, in 2014. He is currently a research scientist in the Institute of Infocomm Research (I2R), Agency for Science, Technology and Research (A*STAR), Singapore.

His research interests are in neuromorphic vision processing, spiking neural networks, biologically inspired object recognition and VLSI circuits and systems design.

**Ruoxi Ding** (S'13) received the B.S. degree in Electronic Engineering from Sichuan University, Chengdu, China, in 2013. He is currently working towards to M.Eng. Degree in the school of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. His research interests are in Neuromorphic vision processing, neural networks, machine learning and reconfigurable circuits and systems design.

**Shoushun Chen** (M'05-SM'14) received his B.S. degree from Peking University, M.E. degree from Chinese Academy of Sciences and Ph.D degree from Hong Kong University of Science and Technology in 2000, 2003 and 2007, respectively. He held a post-doctoral research fellowship in the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology for one year after graduation. From February 2008 to May 2009 he was a post-doctoral research associate within the Department of Electrical Engineering, Yale University. In July 2009, he joined Nanyang Technological University as an assistant professor.

Dr. Chen is a Senior Member of IEEE. He serves as a technical committee member of Sensory Systems, IEEE Circuits and Systems Society (CASS); Associate Editor of IEEE Sensors Journal; AssociateEditor of Journal of Low Power Electronics and Applications; Program Director (Smart Sensors) of VIRTUS, IC Design Centre of Excellence; Regular reviewer for a number of international conferences and journals such as TVLSI, TCAS-I/II, TBioCAS, TPAMI, Sensors, TCSVT, etc.

**Bernabe Linares-Barranco** (Fellow 10) received the B. S. degree in Electronic Physics, the M. S. degree in Microelectronics, and a first Ph.D. degree in June 1990 from the University of Sevilla, Sevilla, Spain, in 1986, 1987, and 1990, respectively, and a second Ph.D degree from Texas A&M University, College-Station, USA, in 1991.

Since 1991, he has been with the Sevilla Microelectronics Institute (IMSE-CNM), from the "Spanish Research Council" (CSIC) of Spain, where he currently is Full Professor of Research. He has been visiting professor/fellow at the Johns Hopkins University, Baltimore (USA), Texas A&M University (USA), and The University of Manchester (UK). His recent interests are in Address-Event-Representation VLSI, real-time AER vision sensing and processing chips, memristor circuits, and extending AER to the nanoscale.

He has received two IEEE Transactions Best Paper Awards and has been Associate Editor of the IEEE Transactions on Circuits and Systems-Part II, IEEE Transactions on Neural Networks, and Frontiers in Neuromorphic Engineering. From March 2011 until November 2013 he was Chair of the IEEE CAS Society Spain Chapter. He is an IEEE Fellow since 2010.

**Huajin Tang** (M'01) received B.Eng. degree from Zhejiang University, M.Eng. degree from Shanghai Jiao Tong University, China, and Ph.D. degree in electrical and computer engineering from National University of Singapore, Singapore, in 1998, 2001, and 2005, respectively.

Dr. Tang was an R&D Engineer with STMicroelectronics, Singapore, from 2004 to 2006. From 2006 to 2008, he was a Postdoctoral Fellow with Queensland Brain Institute, University of Queensland, Australia. He is currently a Research Scientist and Leader of the Cognitive Computing Group, Institute for Infocomm Research, Singapore. He has published one monograph (Springer-Verlag 2007) and over 30 international journal papers. His current research interests include neural computation, neuromorphic systems, cognitive computing, and neuro-robotics. He is an Associate Editor of IEEE Transactions on Neural Networks and Learning Systems and Editorial Board Member of Frontiers in Robotics and AI. He serves as guest editor for Neurocomputing (2014) Special Issue on Brain Inspired Models of Cognitive Memory, and for IEEE Trans. on Neural Networks and Learning Systems (2015) Special Issue on Learning in Neuromorphic Systems and Cyborg Intelligence. He serves as organizing committee member for Asia Pacific Symposium of Intelligent and Evolutionary Systems (IES) 2014 and Program Chair for IEEE International Conference on Cybernetics and Intelligent Systems (CIS) 2015.