IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 9, 2018, accepted April 2, 2018, date of publication April 4, 2018, date of current version April 25, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2823260

Exploiting Lightweight Statistical Learning for Event-Based Vision Processing

CONG SHI[1], (Member, IEEE), JIAJUN LI[2], YING WANG[2], (Member, IEEE), AND GANG LUO[1]
[1]Schepens Eye Research Institute, Massachusetts Eye and Ear, Department of Ophthalmology, Harvard Medical School, Boston, MA 02114 USA
[2]State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100864, China

Corresponding author: Ying Wang (wangying2009@ict.ac.cn)

This work was supported by NIH under Grant R01 AG041794.

ABSTRACT This paper presents a lightweight statistical learning framework potentially suitable for low-cost event-based vision systems, where visual information is captured by a dynamic vision sensor (DVS) and represented as an asynchronous stream of pixel addresses (events) indicating a relative intensity change on those locations. A simple random ferns classifier based on randomly selected patch-based binary features is employed to categorize pixel event flows. Our experimental results demonstrate that compared to existing event-based processing algorithms, such as spiking convolutional neural networks (SCNNs) and the state-of-the-art bag-of-events (BoE)-based statistical algorithms, our framework excels in high processing speed (2× faster than the BoE statistical methods and >100× faster than previous SCNNs in training speed) with extremely simple online learning process, and achieves state-of-the-art classification accuracy on four popular address-event representation data sets: MNIST-DVS, Poker-DVS, Posture-DVS, and CIFAR10-DVS. Hardware estimation shows that our algorithm will be preferable for low-cost embedded system implementations.

INDEX TERMS Address-event representation (AER), dynamic vision sensor (DVS), random ferns, statistical learning, neuromorphic processing.

I. INTRODUCTION

Computer vision systems are pervasively applied in modern society including security surveillance [1], human-machine interface [2], assisted driving [3], and industrial automation [4]. Traditional computer vision systems are equipped with frame-based image sensors. These systems scan and process all the pixels in every frame snapshot by the sensor at a fixed frame rate (e.g. 30 frames per second), regardless of the light intensity change between successive frames. These systems may waste considerable computational power and time to process redundant visual information. On the other hand, these frame-based systems are unable to respond on-the-fly to fast-changing scenes as they have to wait for the next frame tic. In general, the traditional vision systems are not suitable for those applications requiring low system cost, low power consumption and immediate response to environmental dynamics.

To address this problem, the frame-less dynamic vision sensor (DVS) is proposed in recent literature [5]–[7]. DVS is inspired by the visual path structures and functionalities found in the biological visual systems. In typical biological visual systems, neurons asynchronously code and communicate the visual information as spatiotemporally sparse light change in the form of spikes. To mimic such highly-efficient biological mechanism, each pixel in a DVS sensor continuously and independently monitors the change of light intensity cast onto it. When the relative intensity change of one pixel reaches a threshold, the pixel immediately sends out a spike affiliated with its coordinate (address) in the pixel array, as well as the change polarity (increase/decrease in intensity). One pixel sends out information only when perceivable intensity changes are detected due to visual motion or light condition variations. Otherwise it remains inactive to avoid generating redundant data for static scenes. Moreover, each pixel asynchronously responds to its local intensity change on-the-fly (usually several microseconds in latency) without waiting for the other pixels' outputs to synchronize an image frame. Therefore, the output of DVS is a time-continuous stream of spikes encoded with pixel addresses and intensity change polarity, called address-event representation (AER).

Compared to frame-based image sensors, DVS features higher power efficiency and low response latency, and is believed to be very suitable for embedded systems. Please refer to Fig. 3 and Fig. 4 in [7] for an overview of the DVS principle, and [5], [6] for more details.

While more than ten years have elapsed since the first DVS sensor was published in 2008 [5], the design of efficient computer vision algorithms for AER signals remains a challenging task. Most published works employ spiking-based (neuromorphic) algorithms to process the AER data stream [8]–[17]. The neuromorphic algorithm receives AER spikes from a DVS as its input, and outputs another spike sequence on-the-fly to represent the processing result. Some works use kernel filters to extract one layer of convolution maps that represent the key visual features of the original DVS data [8]–[11]. Once a visual pattern that matches with the convolution kernel is detected in the input AER stream, the convolution map immediately outputs a spike to flag such a match. The output spike is also affiliated with its address on the convolution map. However, such shallow structures can only be used to recognize and track objects with simple shapes (e.g., oriented lines, squares, triangles, circles and ellipse) under constrained environment. Some other works inspired by the cortex visual processing model [18] extend to a deeper neuromorphic structure with more spiking convolution layers and learning based classifiers [13]–[17]. More convolution layers allow for extracting more complex visual features, and those classifiers with learning capability can be trained to recognize objects with complicated shape or texture patterns in natural environment. The classifiers in these deep networks are usually implemented as a spiking-based processor for a fully neuromorphic vision system. For instance, the deep convolution networks in [14] employ the tempotron classifier [19], which can be regarded as a spiking version of the single-layer perceptron network [20]. Another deep learning AER algorithm without convolutions is the spiking deep belief network [12]. It also achieves good classification accuracy. However, these deep learning algorithms have three key drawbacks to prevent them from being used in low-cost embedded vision systems: 1) they require large memory space to store model parameters, and many computational resources to realize real-time processing; 2) The training procedures of their classifiers are too complicated to support online learning (i.e., incrementally learning new instances on the fly), which is compulsory for many embedded applications; 3) they are difficult to adapt to traditional processors such as CPUs, GPUs or DSPs, but can only run on expensive neuromorphic processors [15], [21]–[25].

Recently, some researchers have turned to simpler statistical learning methods [26], [27] for AER processing. These methods are much more computationally efficient than their deep learning counterparts. The state-of-the-art work in [26] has opened up a horizon to explore the potential of lightweight statistical learning for processing AER data more efficiently. It proposes statistical features called bag-of-events (BoE) to represent AER events. The BoE features are learned to be discriminative to different AER visual patterns. The learning procedure simply involves counting the number of events in the training set occurred at each pixel location for each object class, and then transforming these counts to BoE features, based on information theory. BoE learning is very fast and simple to be implemented online. The BoE based statistical algorithm outperforms previous deep learning methods in processing speed by a large margin with comparable classification accuracy. The BoE algorithm involves little spiking-based processing and is compatible with low-cost traditional processors. However, the classification of BoE features is completed by Support Vector Machine (SVM) [26] or Random Forests [27]. These classifiers are still complicated for computationally efficient online learning.

In this work, we aim to seek a more computationally efficient real-time lightweight statistical learning algorithm to classify the AER data. We propose a very simple statistical learning framework based on Random Ferns [28] with a few random binary features. Our framework has two important improvements over the previous work in [26]: 1) each binary feature is derived by simply comparing the counts of events falling into two patches on the pixel array within a time duration. All patches are randomly selected without any training process. 2) The classifier training procedure is extremely simple for efficient online learning (only involving additions and memory accesses). Comparing to existing state-of-the-art spiking deep networks and statistical methods, our statistical framework can run at a much higher speed while achieving comparable classification accuracy on multiple datasets including the challenging CIFAR10-DVS [27].

The main contributions of this work include: 1) to our best knowledge, this is the first AER processing framework supporting online learning classifier with extremely simple operations, while achieving state-of-the-art accuracy. 2) We elaborate the robustness of our framework by tweaking the hyperparameters and analyzing their impacts on classification accuracy and processing speed. 3) We estimate the processing performance and resource consumption for customized hardware implementation of our framework. This paper proceeds as follows. Section II briefly introduces our lightweight statistical learning framework for AER data classification based on Random Ferns. Elaborate experimental results and comparisons with other works are provided and analyzed in Section III. Section IV further explains the experimental results and discusses the hardware friendliness of our framework. Finally, Section V concludes this paper with future directions.

II. LIGHTWEIGHT STATISTICAL LEARNING FRAMEWORK

The overall algorithm flow of our statistical learning framework for AER data classification is depicted in Fig. 1. The framework consists of three modules: a motion detector, a bank of binary feature extractors, and Random Ferns classifier. These modules are described in the following subsections.

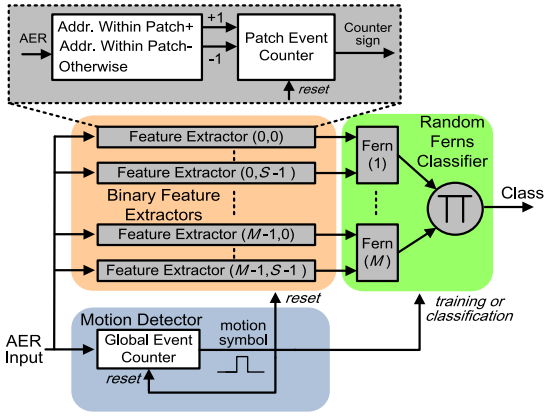


FIGURE 1. The algorithm flow of our simple statistical learning framework for AER data classification.

A. BINARY FEATURE EXTRACTORS

Each binary feature extractor continuously counts the AER motion events (ignoring the event polarity) falling respectively into two square patch regions selected at random locations with random (but identical) size $D \times D$. Then it outputs the comparison results of the two counts as a binary feature. This is equivalent to using only one patch event counter initialized as zero, and increasing/decreasing it by one when the address of an incoming AER event is within the first/second patch region (Patch+/−), as illustrated in Fig. 1. The counter signs are the expected binary features to be fed to the random ferns. The computations required by the feature extractors are rather simple. No training procedure or hand-crafted feature design is involved.

B. MOTION DETECTOR

To avoid performing training or classification all the time (i.e., for every incoming event), a motion detector [14], [26] is employed in our work. For the convenience of comparison with other works, our motion detector uses merely a global event counter initialized as zero, and increases it by one whenever an AER event arrives (ignoring event address and polarity). When the counter exceeds a threshold α, a motion symbol spike is generated. The spike locks the binary features into the random ferns, resets the global and patch event counters to 0, and activates the random ferns to perform online training (for class-labeled AER data) or classification (for unlabeled AER data). In this way, the motion detector actually groups every α AER events as a segment, and triggers training or classification at the end of each segment. Fig. 2 shows some "images" reconstructed from the AER segments. Such motion detector can be regarded as a special variant of the leaky integration & fire (LIF) neuron in the neuromorphic systems [9], [14], [26], with an infinitely large time constant (i.e., no leakage at all).

C. ONLINE RANDOM FERNS

The Random Ferns classifier [28] is a randomized variant from the semi-naive Bayesian classifier family [29]. Let M be the number of ferns and S be the number of binary features

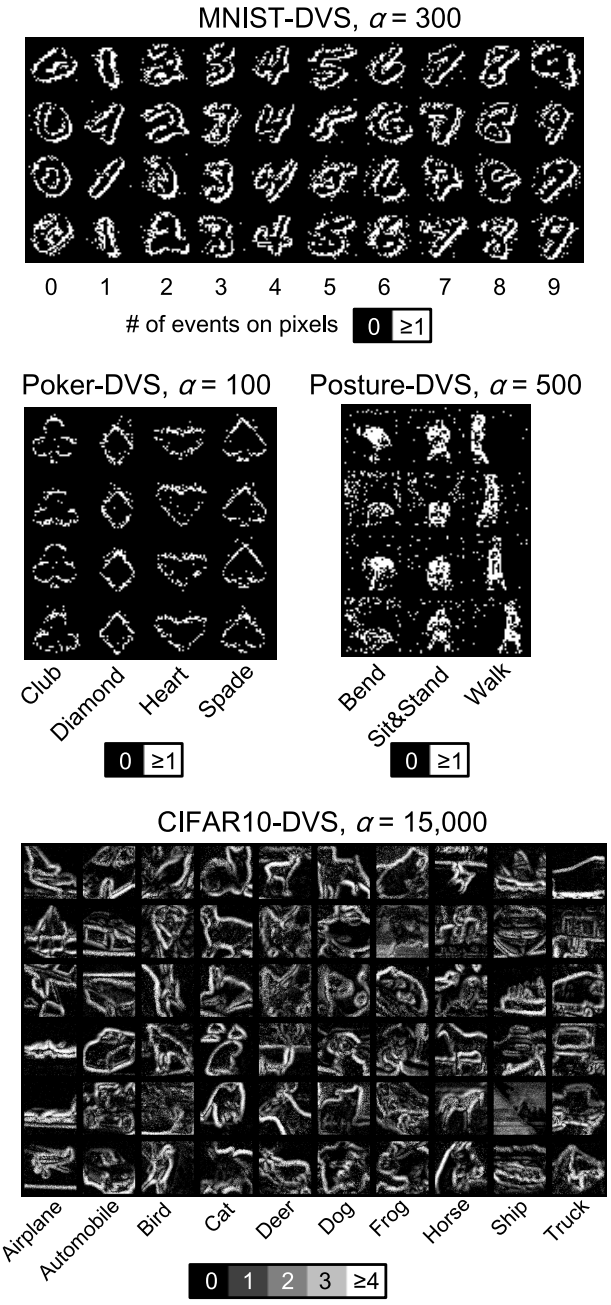


FIGURE 2. The reconstructed images from four AER dataset: MNIST-DVS [30], Poker-DVS [30], [31], Posture-DVS [14] and CIFAR10-DVS [27]. They are much more difficult to be recognized than their grayscale or color versions due to too much motion-induced noise and the loss of light intensity information. The pixel value represents the number of events occurred at that address in a segment.

per fern. We define $f_{m,s}$ as the s-th binary feature in the m-th fern, and the feature group $F_m = (f_{m,0}, f_{m,1}, \ldots, f_{m,S-1})$ can be treated as a radix-2 numeric value represented by the S binary features in sequence. Take $S = 6$ for example, if $(f_{m,0}, f_{m,1}, \ldots, f_{m,S-1}) = (0, 1, 1, 0, 0, 1)$, we can regard F_m as $(011001)_2 = 21$. In this way, F_m can take 2^S different values in each fern.

There are two ways to combine the ferns during training and classification. The first one is to combine the ferns with

their class-conditional probabilities in a Bayesian (multiplicative) fashion. Let C be the number of object classes and $c \in \{0, 1, \ldots, C\text{-}1\}$ be the class label. To train the ferns with a training set of AER segments, we simply calculate: 1) $N(m, F_m, c)$, the number of training segments belonging to class c with its binary features evaluated to F_m in the m-th fern, with m running from 0 to M - 1, F_m from 0 to 2^S - 1 and c from 0 through C - 1; 2) N_c, the total number of training segments belonging to class c, with c running from 0 to C - 1. Such trivial learning process can be easily realized online: when a training AER segment labeled as class c arrives, we evaluate its binary features F_m, and simply increase corresponding $N(m, F_m, c)$ in each fern and N_c by 1. After training, the ferns can be applied to categorizing an unlabeled AER segment with features F_m, into class $c*$ winning the maximum global posterior probability among all the classes:

$$\begin{aligned} c* &= \arg\max_c P(c|F_0, F_1, \ldots, F_{M-1}) \\ &= \arg\max_c \frac{P(F_0, F_1, \ldots, F_{M-1}|c)P(c)}{P(F_0, F_1, \ldots, F_{M-1})} \\ &\qquad \text{(Bayesian formula)} \\ &= \arg\max_c \frac{P(F_0, F_1, \ldots, F_{M-1}|c)}{P(F_0, F_1, \ldots, F_{M-1})} \\ &\qquad \text{(uniform prior assumption)} \\ &= \arg\max_c P(F_0, F_1, \ldots, F_{M-1}|c) \\ &\qquad \text{(class-irrelevant denominator removed)} \\ &= \arg\max_c \prod_{m=0}^{M-1} P(F_m|c) \\ &\qquad \text{(assumption of independent ferns)} \end{aligned} \tag{1}$$

where the class-conditional probability $P(F_m|c)$ is estimated as

$$P(F_m|c) = \frac{N(m, F_m, c) + N_r}{N_c + 2^S N_r} \tag{2}$$

with N_r usually set to 1 as a smoothing prior [28].

The second way is to combine the ferns with their local class posterior probabilities in an average (additive) fashion. In this way, we only need to count $N(m, F_m, c)$ values during online learning. Then the trained ferns categorizes an unlabeled AER segment with features F_m into class c^* winning the maximum average of the ferns' local posterior probabilities $P(c|F_m)$ among all the classes:

$$\begin{aligned} c* &= \arg\max_c \frac{1}{M} \sum_{m=0}^{M-1} P(c|F_m) \\ &= \arg\max_c \sum_{m=0}^{M-1} P(c|F_m) \\ &\qquad (M \text{ is independent of } c) \end{aligned} \tag{3}$$

where $P(c|F_m)$ of each fern is estimated as

$$P(c|F_m) = \frac{N(m, F_m, c) + N_r}{\sum_{c=0}^{C-1} N(m, F_m, c) + cN_r} \tag{4}$$

We will evaluate and compare the Bayesian and average combination ways regarding classification accuracy, time consumption and hardware friendliness later.

III. EXPERIMENTAL RESULTS

A. AER DATASETS

We used four AER datasets to evaluate the classification performance of our statistical learning framework: MINST-DVS, Poker-DVS, Posture-DVS and the more challenging CIFAR10-DVS, with pixel resolutions of 28 × 28, 32 × 32, 32 × 32 and 128 × 128, respectively, as shown in Fig. 2. Every AER stream in these datasets was generated by moving its corresponding original (gray or color) image in front of real DVS sensors. The MINST-DVS set contains the 10 digits from 0 to 9. The Poker-DVS set has 4 types of poker cards: Club, Diamond, Heart and Spade. The Posture-DVS set consists of 3 classes of human postures: Bend, Sit & Stand, and Walk. The CIFAR10-DVS set contains 10 object classes including Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship and Truck. They are difficult even for humans to distinguish due to the loss of color information and much motion-induced noise. The AER streams in each dataset were segmented by the motion detector with different values of α in following experiments. For each dataset, we randomly selected 90% streams for training, and the remaining 10% for testing. A 10-fold cross validation scheme was used in each session throughout the experiments. Our algorithm was implemented in Matlab for software simulation.

B. HYPERPARAMETER EVALUATION

There are 4 hyperparameters in our algorithm framework to be determined: 1) S, the number of binary features per fern, 2) M, the number of ferns, 3) α, the number of events in one segment, and 4) $[D_{\min}, D_{\max}]$, the size range of the random feature patches. In each session we elaborated the classification accuracy by tweaking only one hyperparameter, while leaving the others fixed at reasonable baseline values, as given in Table 1. The hyperparameters were evaluated in the Bayesian combination way with $N_r = 1$. The evaluation results for classification accuracy, training time on all training AER streams, and classification time on all testing AER streams (both including the time for binary feature extraction) are plotted in Fig. 3. Most of the standard deviation bars are too small to be seen in the figures.

In Fig. 3(a), When the number of binary features per fern increases, the classification accuracy for each AER testing set first increases and then plateaus when the number of features reaches certain points (denoted in this paper as S^*). Adopting more features only ends up with increasing time cost as well as wasting memory and computational resources. By observing the saturating points in Fig. 3(a) in each AER dataset, we give a rough estimation for the optimal feature number S^*:

$$S^* \approx 2\log_{10}(\#\text{ of training segments}) + \log_2(\#\text{ of classes}) \tag{5}$$

TABLE 1. Baseline values of hyperparameters for each AER dataset.

Hyperparameter	M_b (# of ferns)	S_b (# of bin-feature/fern)	α_b (# of segment events)	$[D_{min,b}, D_{max,b}]$ (pixels) (patch size range)
MNIST-DVS	50	12	300	[3, 5]
Poker-DVS	50	8	100	[3, 5]
Posture-DVS	50	10	500	[3, 5]
CIFAR10-DVS	50	14	15,000	[12, 20]

The subscript *b* affiliated with the hyperparameter names stands for *baseline*.

Under the training rate of 90% as mentioned above, there are 19992, 467, 22175, 111850 training segments and 10, 4, 3, 10 classes for the 4 AER datasets, respectively. Using Eq. (5), we can estimate their optimal feature numbers as 11.9, 7.3, 10.3 and 13.4, respectively. These values justify our choice for the baseline feature numbers (12, 8, 10 and 14, respectively) in Table 1. Our classifier plateaus around 100% accuracy for Poker-DVS and Posture-DVS, which only contains 4 and 3 different categories with sufficient number of training segments. The accuracy on MNIST-DVS is a bit lower than 80%, as it has as many as 10 different digits to be classified, but only with approximately the same number of training segments as Posture-DVS. The classification of CIFAR10-DVS is more challenging as it contains 10 classes and is quite noisy. Moreover, since it was produced by moving slowly the original static CIFAR10 images before the DVS sensor, the segmented ''images'' in each AER stream are very similar to each other, thus reducing the diversity of the training set. Not surprisingly, the accuracy on CIFAR10-DVS reaches only 30%. But it is still comparable to that of previous state-of-the-art statistic BOE method [27].

Fig. 3(b) shows the dependency of classification accuracy on the number of ferns. The accuracy is nearly constant for values of this hyperparameter above 50. Similar to the evaluation results on the number of binary features, adding more ferns barely helps with accuracy improvement but only increases the processing time linearly and wastes memory and computational resources. The classification accuracy is also robust to the number of events in each segment as illustrated in Fig. 3(c), unless the framework has only collected too few events before it makes one training or classification (e.g., $\alpha/\alpha_b = 0.2$ in Fig. 3(c)). Although increasing the number of segment events can reduce the processing time, it comes at the cost of sparser online training or classification operations along the AER stream. Such sparsity may not be acceptable in some industrial applications requiring an immediate response. Therefore, the selection of number of segment events should depend on different applications. In this work, our baseline numbers of segment events listed in Table 1 were selected for the purpose of a fair comparison among our work and previous algorithms, as will be shown in the next subsection. Fig. 3(d) demonstrates the robustness of our algorithm against the size of binary feature patches. We can see that choosing a small size range around 1/8 of the DVS sensor dimensions (i.e., the width and height of the sensor pixel array) would always be a good practice.

C. EVALUATION OF FERN COMBINATION WAYS AND SMOOTHING PRIOR

Fig. 4 further evaluates the classification accuracies on the AER datasets using the two different ways of fern combination. Each combination goes with and without the smoothing prior term N_r. When the number of binary feature is small, the accuracy with the smoothing prior (i.e., $N_r = 1$) and that without the smoothing prior (i.e., $N_r = 0$) under the same combination are close to each other. However, as the number of binary feature increases, the accuracy without N_r drastically degrades to an unacceptable level. On the other hand, when the number of binary features is small, the accuracy using the Bayesian combination is obviously higher than that using the average combination, if both with $N_r = 1$. However, once the number of binary features reaches optimal number S^* as discussed earlier, the two combination methods with $N_r = 1$ produce almost equal accuracies. We will further discuss this in the next section. The training and classification time using the average combination is always less ($0.4\sim0.8\times$) than that using the Bayesian combination (see Table 2). However, such difference is mainly due to the Matlab software simulation environment. In fact, the Bayesian combination can run more efficiently on customized hardware than the average one, as will be explained in the next section. Introducing the smoothing prior N_r or not has very subtle impact on the processing time. In sum, we would prefer to using the Bayesian combination with $N_r = 1$.

D. COMPARISON TO STATE-OF-THE-ARTS

We compared our method with previous state-of-the-art AER classification algorithms and systems, as demonstrated in Table 2. The baseline hyperparameter values listed in Table 1 are used in our framework throughout the comparison unless otherwise stated. The classification accuracy, training time and classification time (including the feature extraction stage) are compared on the four DVS-AER datasets used above. Our algorithm framework was implemented and simulated in the Matlab (ver. 2015a) environment without any compiled C/C++ library on a desktop computer (Intel i7-4790 CPU @ 3.60GHz). All of the works in the comparison shared the same numbers of segment events as listed in Table 1. Those values for MNIST-DVS, Poker-DVS and Posture-DVS segments strictly followed those used in [26], while the value for CIFAR10-DVS segments was determined to approximate a duration of 100 ms [27]. All the methods in the comparison employed the 10-fold cross validation scheme

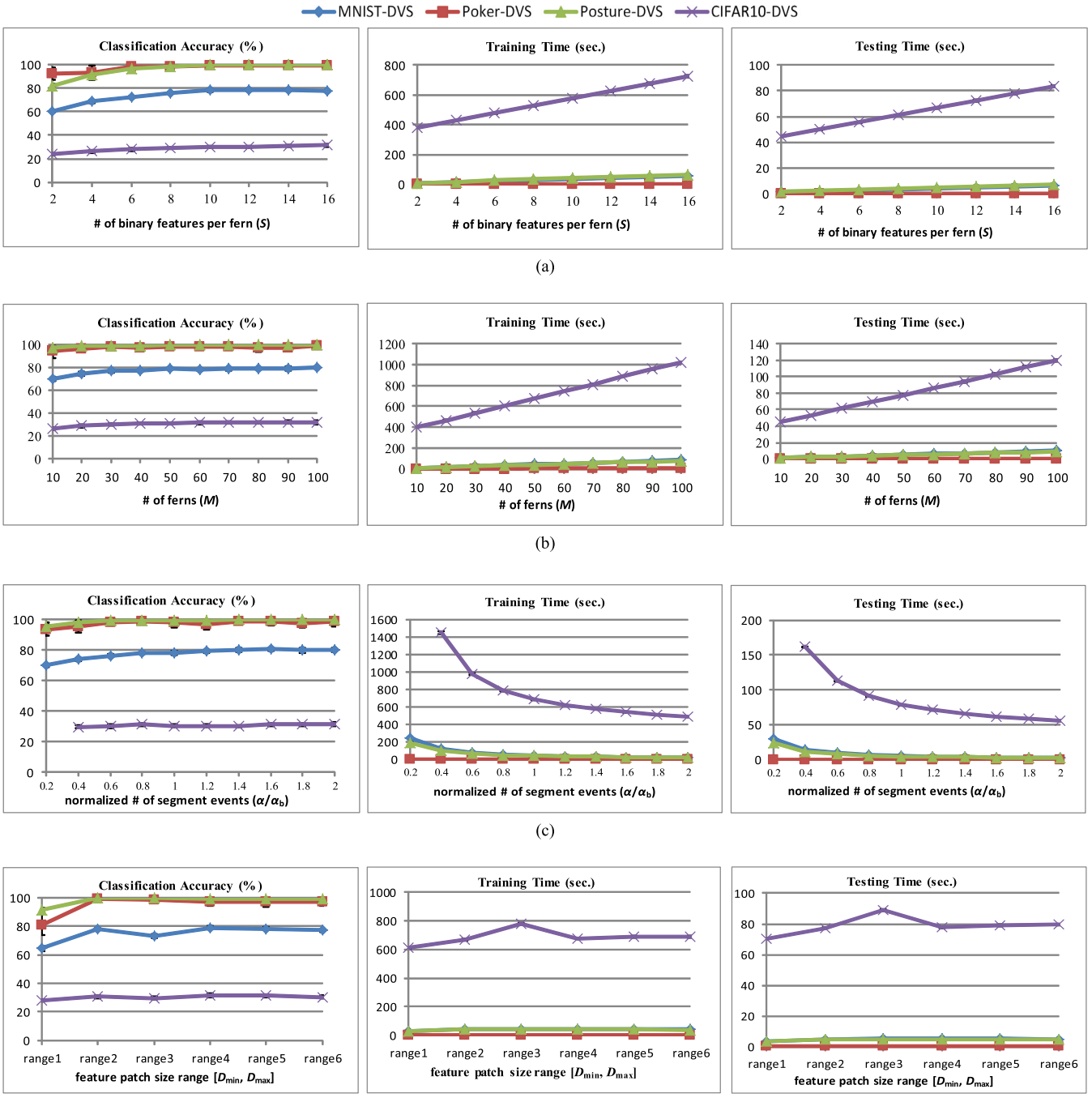


FIGURE 3. The classification accuracy, training time on all training AER streams, and classification time on all testing AER streams (both including the binary feature extraction time) for hyperparameter evaluations under the Bayesian combination with $N_r = 1$. In each session we scan only one hyperparameter while leaving others fixed at their baseline values in Table 1. The mean and stand deviation are obtained using a 10-fold cross validation. Range1 $\sim$ Range6 in (d) represent different ranges $[D_{\min}, D_{\max}]$ that the binary feature patch size D can be randomly chosen from: for MNIST-DVS, Poker-DVS and Posture-DVS, they are [1, 1], [4, 4], [7, 7], [3, 5], [2, 6], [1, 7] pixels, while for CIFAR10-DVS, they are [4, 4], [16, 16], [28, 28], [12, 20], [8, 24], [4, 28], respectively. (a) Evaluation on S, the number of binary features per fern. (b) Evaluation on M, the number of ferns. (c) Evaluation on α/α_b, the number of segment events normalized by its baseline value. (d) Evaluation on $[D_{\min}, D_{\max}]$, the feature patch size range.

to compute the average classification accuracy and time consumption. In each validation fold, 90% data of one AER set were randomly picked out for training and the other 10% for classification testing. The result data for Chen's, Zhao's and BoE + SVM methods on MNIST-DVS, Poker-DVS and Posture-DVS datasets in Table 2 were obtained (copied) from [26], while the data for Zhao's, BOE + SVM and BoE + Forests methods on the CIFAR10-DVS dataset were obtained from [27].

The highest classification accuracies and the least processing time consumptions are underscored in Table 2. Note that our method outperforms all the others in classification

TABLE 2. Work comparison for classification accuracy and processing time.

Methods	MNIST-DVS			Poker-DVS			Posture-DVS			CIFAR10-DVS		
	acc. (%)	t_{train} (s)	t_{class} (s)	*acc.* (%)	t_{train} (s)	t_{class} (s)	*acc.* (%)	t_{train} (s)	t_{class} (s)	*acc.* (%)	t_{train} (s)	t_{class} (s)
Chen's [13]	63.50	1208	7825	92.53	31	23.34	91.88	1548	11817	-	-	-
Zhao's [14]	75.52	8805	982	91.76	73.12	7.91	95.61	11794	2948	22.10	-	-
BoE+SVM [26]	74.82	31.5	27.4	93.00	**0.03**	**0.02**	98.66	45.46	44.64	24.21	-	-
BoE+Forests [27]	-	-	-	-	-	-	-	-	-	29.67	-	-
Ours (Average, N_r=1)	75.56	**16.1**	**2.0**	**97.43**	0.3	0.05	**99.88**	**20.6**	**2.5**	**31.2**	**491.5**	**56**
Ours (Bayesian, N_r=1)	**78.08**	41.3	5.0	97.2	0.6	0.1	99.59	39.3	5.0	30.59	679.7	78.2

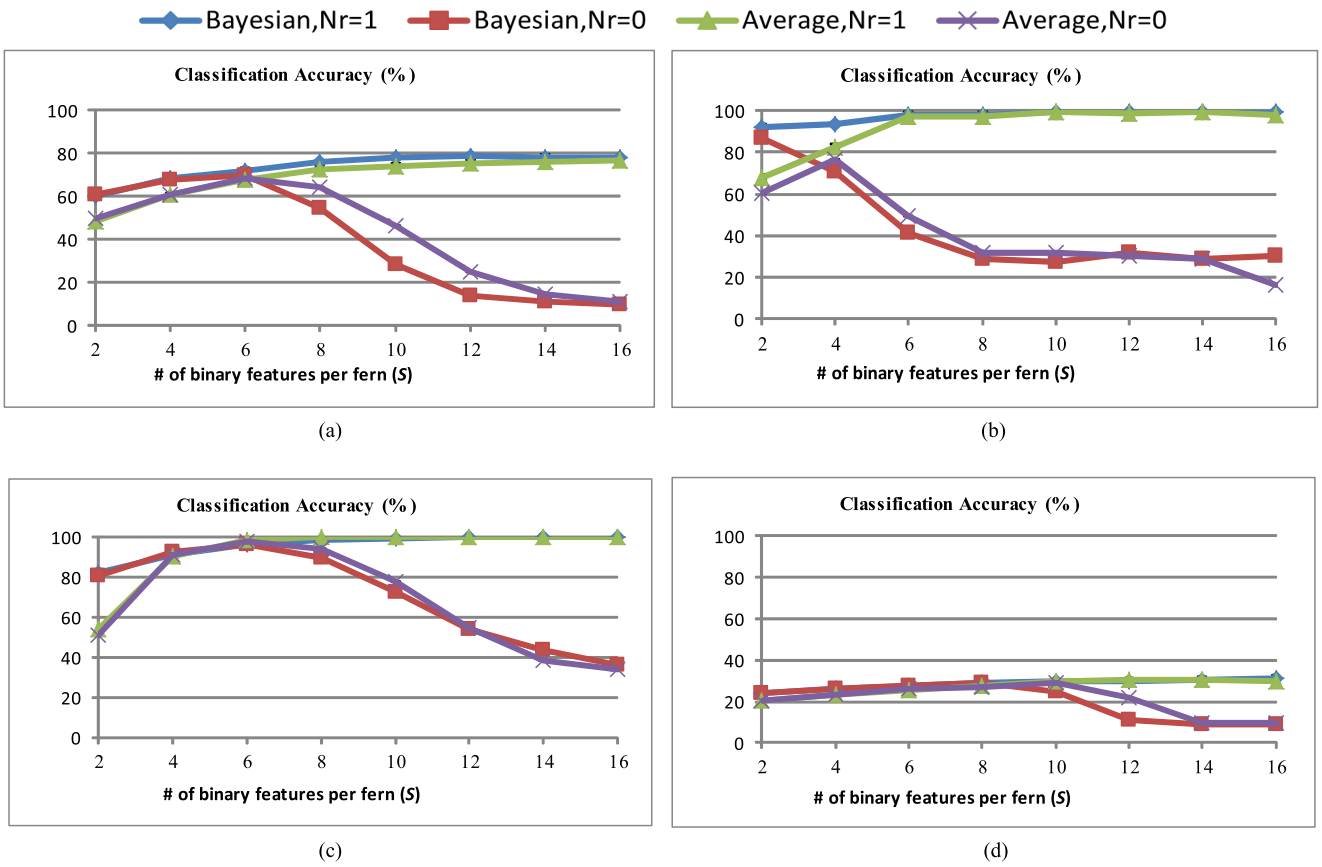


FIGURE 4. The classification accuracies using the two ways of fern combination, each with and without the smoothing prior term N_r. (a) MNIST-DVS. (b) Poker-DVS. (c) Posture-DVS. (d) CIFAR10-DVS.

accuracy. Although the BoE + SVM method accelerated its SVM part by a compiled C/C++ library while our framework ran as a pure Matlab script, our algorithm consumed less processing time in most cases. Our method with the average combination way runs twice as faster than the BoE statistical method during training on the MNIST-DVS and Posture-DVS datasets. When compared to other spiking convolutional neural networks [13], [14], our method runs hundreds to thousands of times faster during both training and classification. The large processing time consumptions in those previous methods prevent them from efficient online learning. In Table 2, our Random Ferns method combined in either Bayesian or average way exhibits comparable classification accuracy. The average way seems to run faster than the Bayesian one in software simulation. But it will not be the case when considering customized hardware implementation, as will be discussed in Section IV-B.

IV. DISCUSSION

A. SMOOTHING PRIOR

From Fig. 4, we can see that dropping the smoothing prior (i.e., setting $N_r = 0$ in Eq. (2) and (4)) greatly diminishes the classification accuracy for both Bayesian and average combination ways. Take the Bayesian way for an explanation:

since we only have a finite set of training segments in any case, some F_m feature branches in a certain fern may have no training segments belonging to a certain class. This will result in a zero class-conditional probability by Eq. (2) for the class in that fern. However, such artifact arises only due to the limited number of training segments. It fails to reflect the real probability distributions. Combining such artifact with other ferns in a multiplicative Bayesian way by Eq. (1) eventually leads to an unreliable zero posterior probability for the class, regardless how strong the class-conditional probabilities in other ferns are. This problem is more serious when the number of binary features S goes high in Fig. 4 (Note that there are totally 2^S branches of F_m in each fern for each class, as mentioned in Section II-C). But keeping S relatively small forbids the capability of employing more features to discriminate among different classes. Introducing the smoothing prior $N_r = 1$ in Eq. (2) solves the dilemma. It is equivalent to assuming that there is already one training sample from each class for each F_m branch in each fern, before observing any real training segment. It helps improve the classification accuracy steadily along with increasing binary features. The smoothing prior is also indispensable for the additive average combination way, though an artifact zero value of local posterior probability in a few ferns by Eq. (4) will not crush the final probability by Eq. (3) to zero. Such concept of prior smoothing might generalize to other similar learning algorithms to improve their classification accuracy, such as to the Random Forests algorithm [32] when combining leave probabilities of all trees.

B. HARDWARE IMPLEMENTATION ESTIMATION

We estimated the performance and resource cost of our algorithm framework with the Bayesian combination way for a customized FPGA hardware implementation. For the feature extractors shown in Fig. 1, supposing the pixel array size in the AER sensor is $p \times p$, the event address can be represented in $P = [\log_2 p]$ bits, where the function $[x]$ returns the minimum integer no less than x. So we need $4 \times 2 \times S \times M$ P-bit registers to store vertex addresses of all positive and negative feature patches. The same amount of P-bit comparators is needed to compare the addresses of one AER event and all patch vertices in parallel. Next, we need to feed the comparison results into $2 \times S \times M$ 4-input AND gates to judge if the AER event has fallen into those patches, and accordingly update the patch event counters. Suppose each such counter on hardware is based on a 10-bit register to handle most cases, and it should support four operations: $+1$, -1, hold and reset, with a few logic resources. We totally need $S \times M$ such 10-bit bidirectional counters. The amount of hardware resources for the feature extractors is insignificant. Even for CIFAR10-DVS with $p = 128$, $S = 14$ and $M = 50$ as in Table 1, we would just consume 5600 7-bit registers, 5600 7-bit comparators, 1400 4-input AND gates, 700 10-bit bidirectional counters, and no multiplier (the most expensive computational resource in most systems) at all. The feature extractors only involve very simple operations, and can process one AER event in only two clock cycles. In the first cycle, the addresses of the event and the patches are compared. In the second cycle, the comparison results are used to update the event counters. The motion detector in Fig. 1 can similarly tackle each AER event in two clock cycles, but consumes much fewer resources: only a one-way counter based on a 16-bit register supporting three operations: $+1$, hold and reset.

For random ferns with the Bayesian combination way, two hardware optimization techniques can be used without degrading any classification accuracy. The first one is to bypass the time- and resource-consuming division operations appeared in Eq. (2). Based on the monotonicity of the logarithm function, Eqs. (1) (2) are equivalent to:

$$\begin{aligned} c^* &= \arg\max_c \prod_{m=0}^{M-1} P(F_m|c) \\ &= \arg\max_c \sum_{m=0}^{M-1} \log P(F_m|c) \\ &= \arg\max_c \sum_{m=0}^{M-1} \log \frac{N(m, F_m, c) + N_r}{N_c + 2^S N_r} \\ &= \arg\max_c \Big(\sum_{m=0}^{M-1} \log(N(m, F_m, c) + N_r) \\ &\qquad - M \log(N_c + 2^S N_r) \Big) \end{aligned} \tag{6}$$

A small memory-based lookup table (LUT) can be employed to realize the logarithm operations. Note this technique does not apply to the average combination way, as the summation of divisions in Eqs. (3) (4) cannot be converted to additions and subtractions in the logarithmic domain. The other optimization is to introduce two new variables $N^*(m, F_m, c) = N(m, F_m, c) + N_r$, and $N_c^* = N_c + 2^S N_r$. Therefore, we can 1) initialize $N^*(m, F_m, c)$ to N_r and N_c^* to $2^S N_r$, instead of initializing $N(m, F_m, c)$ and N_c to 0, before training, and 2) then increase corresponding $N^*(m, F_m, c)$ and N_c^* entries by 1 on each AER training segment. After training, the ferns can classify new AER segments via a simplified formula equivalent to Eq. (6):

$$c* = \arg\max_c \Big(\sum_{m=0}^{M-1} \log N*(m, F_m, c) - M \log N*_c \Big) \tag{7}$$

Suppose each entry of $N^*(m, F_m, c)$ and N_c^* is represented by 8 and 16 bits, respectively. These bit precisions are sufficient in most cases. In a C-class categorization application, we need to allocate $(2^S \times C) \times 8$ bits memory space for $N^*(m, F_m, c)$ to each of the M ferns, and $C \times 16$ bits memory space for N_c^*. Assume the input and output of the logarithm LUT are both represented by 16 bits to achieve high-precision computation. So it requires $2^{16} \times 16$ bits $= 128$ KB memory space. The LUT is shared among all ferns during classification to save memory. This would not degrade the system throughput, as will be further discussed soon. Besides those memory

TABLE 3. Hardware performance comparison of OLINE SVM and online random Ferns.

	Hardware Consumption (on Xilinx Device)				Training Time (ms)					Classification Time (ms/seg)
	f_{clock} (MHz)	Register	DSP48E (Multiplier)	BRAM (4 KB)	512 Segments	2048 Segments	8192 Segments	16384 Segments	131072 Segments	
Online SVM [33]	150	17627	228	281	220.56	1690.19	80993.68	519800*	1.409×10^{8}*	N/A
Online Random Ferns (Estimated)	150	30	1	544(MNIST-DVS) 45(POKER-DVS)	0.0068	0.0273	0.1092	0.2185	1.7476	0.001

* extrapolated from Table III in [33].

consumptions, we also need a few computational resources to accomplish the ferns on hardware. For training, we need $(M + 1)$ adders to update (increase by 1) corresponding entries of $N^*(m, F_m, c)$ in all M ferns and N_c^* in parallel. Note one of the two inputs of each such adder is constantly 1. Thus the adder structure can be simplified to save many resources. For classification, we need one more adder to accumulate the first term in Eq. (7), and one unsigned multiplier to compute the second term. Taking into consideration the bit precisions of $N^*(m, F_m, c)$, N_c^* and the typical value of M in Table 1, the adder can be equipped with two 24-bit inputs and one 24-bit output, and the multiplier with one 16-bit input, one 8-bit input and one 24-bit output. We also need a 24-bit register to store the intermediate result and another 6-bit counter to store the fern index during the accumulation in Eq. (7). Following the parameters in Table 1 and above estimations, the hardware consumption of random ferns for CIFAR10-DVS includes: $(50 \times 2^{14} \times 10 + 10) \times 8$ bits $\approx$ 8 MB memory for storing $N^*(m, F_m, c)$ and N_c^*, 128 KB memory for storing logarithm LUT, 50 simplified 8-bit adders, one simplified 16-bit adders, one standard 24-bit adder, one unsigned 16×8-bit multiplier, 30 bits registers and very few other necessary control logics. With regard to other datasets, we only need $50 \times 2^{12} \times 10 \times 8 + 10 \times 16$ bits ≈ 2 MB for MNIST-DVS, $50 \times 2^{8} \times 4 \times 8 + 4 \times 16$ bits $\approx$ 50 KB for Poker-DVS and $50 \times 2^{10} \times 3 \times 8 + 3 \times 16$ bits $\approx$ 150 KB for Posture-DVS, respectively, to store their $N^*(m, F_m, c)$ and N_c^*. The consumptions on logarithm LUT and other computational resources remain the same for all datasets. Such resource consumptions are much fewer than most deep neural networks. The above estimations also suggest that the hardware ferns would be memory-centric: a moderate amount of memory consumption plus very few computational resources. This is very plausible for embedded systems as memory units are much cheaper than computational units on most computing devices.

The hardware ferns can be trained with the features of one AER segment in only two clock cycles. In the first cycle, corresponding $N^*(m, F_m, c)$ and N_c^* entries are accessed from the memories in parallel, according to the features F_m and the label c. In the next cycle, those entries are increased by 1 and written back to the memories in parallel. The ferns would take a little more time to classify the features of an AER segment. It takes 3 clock cycles to compute the first term in Eq. (7) for each fern: one cycle to access the corresponding $N^*(m, F_m, c)$ entry, one cycle to obtain the logarithm result from the logarithm LUT, and another cycle to accumulate the logarithm result. The second term will similarly consume 3 clock cycles. Since there is only one shared LUT as mentioned earlier, the classification time for one AER segment is $3 \times M + 3$ clock cycles, which equals to 153 when $M = 50$. As all operations involved are rather simple, we can conservatively estimate a 150 MHz clock on the hardware. Therefore, this time consumption is only 1.02 μs. Further, the classification operation is triggered sparsely by the motion detector along AER streams. So we can employ a pipeline scheme to hide the classification time, at a mere cost of M S-bit registers (700 bits for CIFAR10-DVS) to hold the features of the previous one AER segment. Thus we can take the classification operation for previous AER segment by the ferns, and at the same time collect the features for current segment by the feature extractors.

To further demonstrate the hardware-friendliness of our framework, Table 3 compares the performance of the estimated hardware random ferns and an online learning hardware SVM [33]. The online SVM could be regarded as an online variant of the off-line SVM used in [26]. According to the routine of Xilinx FPGA devices, one DSP48E module is used as one multiplier, and one BRAM module is a memory block with 4KB capacity in Table 3. Although the ferns consume more memory resources for some datasets, they require nearly no multipliers that are much more expensive than the memory blocks on the chip. Therefore, the random ferns are more suitable for low-cost systems than SVM. In Table 3 we do not compare other computational resources such as the adders. Because those resources consumed by ferns are very few and not as expensive as the multipliers. On the other hand, the ferns outperform the SVM by a large margin in training speed, which is a critical merit for online learning classifiers. The training speed of ferns is thousands to millions of times higher than that of the SVM, especially when the number of training samples (AER segments) is large. The reason is that the ferns only involve very simple operations requiring just 2 clock cycles to pass a training sample, as already stated earlier.

V. CONCLUSION

This work proposes a lightweight statistical learning framework based on random ferns for AER data classification. Compared with existing methods for AER classification, our

framework exhibits higher processing speed, less computational resource requirement, simplest online learning process and state-of-the-art classification accuracy. The software simulation results and the estimated hardware performance demonstrate that our statistical framework is a promising candidate for high-speed low-cost embedded AER systems. Our future work will focus on: 1) exploring more lightweight statistical learning methods with further improved performance for AER data processing, and 2) implementing those methods on customized system-on-chips with integrated AER sensors.

REFERENCES

[1] U. L. Puvvadi, K. Di Benedetto, A. Patil, K.-D. Kang, and Y. Park, "Cost-effective security support in real-time video surveillance," *IEEE Trans. Ind. Informat.*, vol. 11, no. 6, pp. 1457–1465, Dec. 2015.

[2] J. Yu and Z.-F. Wang, "A video, text, and speech-driven realistic 3-D virtual head for human–machine interface," *IEEE Trans. Cybern.*, vol. 45, no. 5, pp. 991–1002, May 2015.

[3] X. Fu, X. Guan, E. Peli, H. Liu, and G. Luo, "Automatic calibration method for driver's head orientation in natural driving environment," *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 1, pp. 303–312, Mar. 2013.

[4] L. Bodenhagen *et al.*, "An adaptable robot vision system performing manipulation actions with flexible objects," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 3, pp. 749–765, Jul. 2014.

[5] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120 dB 15 μs latency asynchronous temporal contrast vision sensor," *IEEE J. Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, Feb. 2008.

[6] J. A. Leñero-Bardallo, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 3.6 μs latency asynchronous frame-free event-driven dynamic-vision-sensor," *IEEE J. Solid-State Circuits*, vol. 46, no. 6, pp. 1443–1455, Jun. 2011.

[7] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck, "Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output," *Proc. IEEE*, vol. 102, no. 10, pp. 1470–1484, Oct. 2014.

[8] 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 Netw.*, vol. 20, no. 9, pp. 1417–1438, Sep. 2009.

[9] L. Camuñas-Mesa, C. Zamarreño-Ramos, A. Linares-Barranco, A. J. Acosta-Jiménez, T. Serrano-Gotarredona, and B. Linares-Barranco, "An event-driven multi-Kernel convolution processor module for event-driven vision sensors," *IEEE J. Solid-State Circuits*, vol. 47, no. 2, pp. 504–517, Feb. 2012.

[10] J. A. Pérez-Carrasco, B. Acha, C. Serrano, L. Camuñas-Mesa, T. Serrano-Gotarredona, and B. Linares-Barranco, "Fast vision through frameless event-based sensing and convolutional processing: Application to texture recognition," *IEEE Trans. Neural Netw.*, vol. 21, no. 4, pp. 609–620, Apr. 2010.

[11] X. Lagorce, C. Meyer, S.-H. Ieng, D. Filliat, and R. Benosman, "Asynchronous event-based multikernel algorithm for high-speed visual features tracking," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 8, pp. 1710–1720, Aug. 2015.

[12] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network," *Frontiers Neurosci.*, vol. 7, p. 178, Oct. 2013.

[13] S. Chen, P. Akselrod, B. Zhao, J. A. 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.

[14] B. Zhao, R. Ding, S. Chen, B. Linares-Barranco, and H. Tang, "Feedforward categorization on AER motion events using cortex-like features in a spiking neural network," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 9, pp. 1963–1978, Sep. 2015.

[15] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman, "HFirst: A temporal approach to object recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 10, pp. 2028–2040, Oct. 2015.

[16] J. A. Perez-Carrasco *et al.*, "Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward ConvNets," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 11, pp. 2706–2719, Nov. 2013.

[17] H. Wang, J. Xu, Z. Gao, C. Lu, S. Yao, and J. Ma, "An event-based neurobiological recognition system with orientation detector for objects in multiple orientations," *Frontiers Neurosci.*, vol. 10, p. 498, Nov. 2016.

[18] 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, Mar. 2007.

[19] R. Gütig and H. Sompolinsky, "The tempotron: A neuron that learns spike timing–based decisions," *Nature Neurosci.*, vol. 9, no. 3, p. 420, 2006.

[20] 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. Learn. Syst.*, vol. 24, no. 10, pp. 1539–1552, Oct. 2013.

[21] G. Indiveri and S.-C. Liu, "Memory and information processing in neuromorphic systems," *Proc. IEEE*, vol. 103, no. 8, pp. 1379–1397, Aug. 2015.

[22] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[23] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, Aug. 2014.

[24] B. V. Benjamin *et al.*, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[25] D. Neil and S.-C. Liu, "Minitaur, an event-driven FPGA-based spiking network accelerator," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 12, pp. 2621–2628, Dec. 2014.

[26] X. Peng, B. Zhao, R. Yan, H. Tang, and Z. Yi, "Bag of events: An efficient probability-based feature extraction method for AER image sensors," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 4, pp. 791–803, Apr. 2017.

[27] H. Li, H. Liu, X. Ji, G. Li, and L. Shi, "CIFAR10-DVS: An event-stream dataset for object classification," *Frontiers Neurosci.*, vol. 11, p. 309, May 2017.

[28] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua, "Fast keypoint recognition using random ferns," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 3, pp. 448–461, Mar. 2010.

[29] C. M. Bishop, *Pattern Recognition and Machine Learning*. Medford, MA, USA: Springer, 2006.

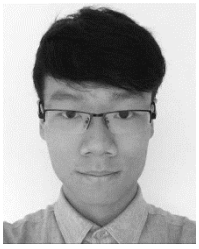
[30] T. Serrano-Gotarredona and B. Linares-Barranco, "Poker-DVS and MNIST-DVS. Their history, how they were made, and other details," *Frontiers Neurosci.*, vol. 9, p. 481, Dec. 2015.

[31] T. Serrano-Gotarredona and B. Linares-Barranco, "A 128×128 1.5% contrast sensitivity 0.9% FPN 3 μs latency 4 mW asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers," *IEEE J. Solid-State Circuits*, vol. 48, no. 3, pp. 827–838, Mar. 2013.

[32] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[33] S. Wang, Y. Peng, G. Zhao, and X. Peng, "Accelerating on-line training of LS-SVM with run-time reconfiguration," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2011, pp. 1–6.

CONG SHI (M'17) received the Ph.D. degree in electronic engineering from Tsinghua University, China, and jointly from the Institute of Semiconductors, Chinese Academy of Sciences, China, in 2014. Since 2015, he has been a Post-Doctoral Fellow with the Schepens Eye Research Institute, Harvard Medical School. His current research interests focuses on bio-inspired algorithms and VLSI systems for image processing, computer vision, and machine learning in medical applications.



JIAJUN LI received the bachelor's degree from the Department of Information Engineering Institute, China University of Geosciences, in 2011. He is currently pursuing the Ph.D. degree with the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include deep learning and software-hardware co-optimization.

YING WANG (M'14) received the B.S. and M.S. degrees in electrical engineering from the Harbin Institute of Technology, in 2007 and 2009, respectively, and the Ph.D. degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, in 2014. He is currently an Associate Professor with ICT, CAS. His research interests include computer architecture and VLSI design, specifically memory system, energy-efficient architecture, and machine learning systems.

GANG LUO received the Ph.D. degree from Chongqing University, China, in 1997. In 2002, he finished his Post-Doctoral Fellow training at the Harvard Medical School, where he is currently an Associate Professor. His primary research interests include basic vision science, image processing, and technology related to driving assessment, driving assistance, low vision, and mobile vision care.