Simultaneous sensing, readout, and classification on an intensity-ranking image sensor

Jörgen Ahlberg, Anders Åstrom and Robert Forchheimer

Ahlberg, J., Åstrom, A., Forchheimer, R., (2018), Simultaneous sensing, readout, and classification on an intensity-ranking image sensor, *International journal of circuit theory and applications*, 46(9), 1606-1619. https://doi.org/10.1002/cta.2549

SPECIAL ISSUE PAPER

Simultaneous Sensing, Read-Out, and Classification on an Intensity-Ranking Image Sensor

Jörgen Ahlberg*,[1] Anders Åström,[2] and Robert Forchheimer[3]

[1] *Computer Vision Laboratory, Department of Electrical Engineering, Linköping University, Linköping, Sweden*
[2] *Swedish National Forensic Centre (NFC), Linköping, Sweden*
[3] *Division of Information Coding, Department of Electrical Engineering, Linköping University, Linköping, Sweden*

Correspondence: Dr. Jörgen Ahlberg, Computer Vision Laboratory, Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden. Email: jorgen.ahlberg@liu.se

Received 30 September 2017; Revised 28 June 2018; Accepted 29 June 2018

Summary

We combine the Near-Sensor Image Processing (NSIP) concept with Address-Event Representation (AER) leading to an Intensity-Ranking Image Sensor (IRIS) and show the benefits of using this type of sensor for image classification. The functionality of IRIS is to output pixel coordinates (X- and Y-values) continuously as each pixel has collected a certain number of photons. Thus, the pixel outputs will be automatically intensity-ranked. By keeping track of the timing of these events, it is possible to record the full dynamic range of the image. However, in many cases this is not necessary – the intensity ranking in itself gives the needed information for the task at hand. This paper describes techniques for classification and proposes a particular variant (Groves) which fits the IRIS architecture well as it can work on the intensity rankings only. Simulation results using the CIFAR-10 dataset compare the results of the proposed method with the more conventional Ferns technique. It is concluded that the simultaneous sensing and classification obtainable with the IRIS sensor yields both fast (shorter than full exposure time) and processing-efficient classification.

Keywords: image sensors, image classification, machine learning, near-sensor processing

1 Introduction

Specialized sensors for machine vision have been developed in parallel with the image sensors used for electronic photography. Such specialized sensors include 1D sensors (linear arrays), high-dynamic range sensors, high frame-rate sensors, global-shutter sensors, polar (circular) pixel arrangements, multi-spectral sensors, time-of-flight sensors et cetera. The aim has been to produce appropriate data for the specific applications.

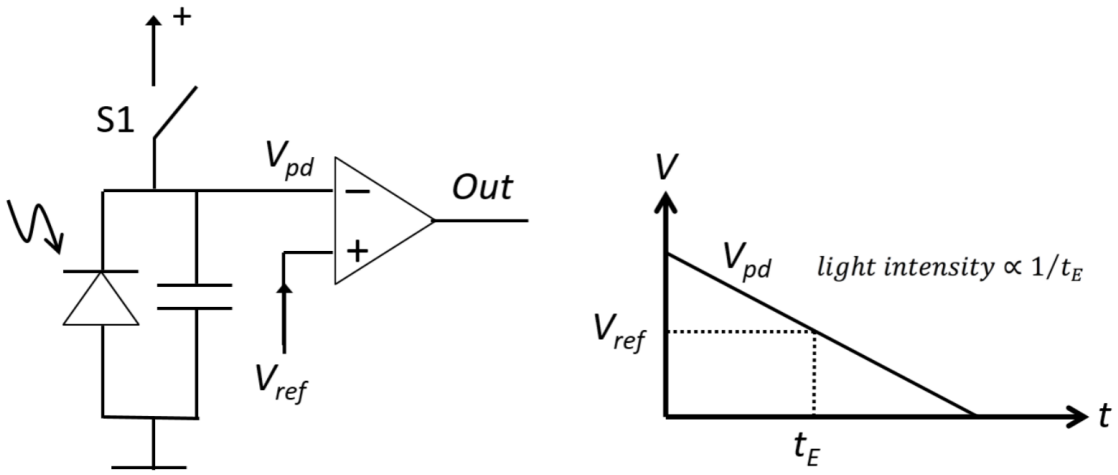

Figure 1: NSIP pixel sensor (left) and intensity-time relationship (right).

A particular class of sensors uses a combination of light sensing device and processing element at each pixel. Such sensors are used where extremely fast processing is required. An example is the Near-Sensor Image Processing (NSIP) concept (1, 2). This was first applied to 1D arrays (3) and was later extended to 2D (4). A characterizing feature in NSIP is the pixel design shown principally in Figure 1 (left). The sensing element is a reverse-biased photo diode/capacitor that is charged through switch S1. When illuminated, the diode discharges approximately linearly in time at a rate that depends on the light intensity (Figure 1, right). After some time t_E the diode voltage reaches a preset threshold V_{ref} which is sensed by a comparator circuit. The comparator outputs a logical 1 to indicate this event. The time t_E then represents the light intensity of that pixel. In addition to the comparator, the NSIP pixel also includes a binary processor (not shown) capable of processing and storing binary data. Although the processor can be used to measure the time t_E and thus output a value corresponding to the light intensity, this is not very often used in the applications of these sensors. Instead, the image processing tasks are redefined to work directly in the "exposure-time domain". As an example, finding object edges is done by logically combining the output from two neighboring comparators and noting when one of them outputs a logical 1 while the other one still outputs a logical 0. Measuring how long this situation lasts indicates the contrast (gradient) of the edge. Numerous other low-level image processing tasks can be rephrased in a similar manner and shown to run faster using less power than the traditional approaches (2).

Classification (recognition) of specific objects in images has traditionally relied on the extraction of salient features followed by fairly simple (such as rule-based or linear discriminant) classifiers. Recent advances in image classification include learning techniques, thus eliminating the manual design and selection of discriminant functions and, in some cases, feature extractors. In this category we find Support Vector Machines, Neural Networks, Decision Trees, Random

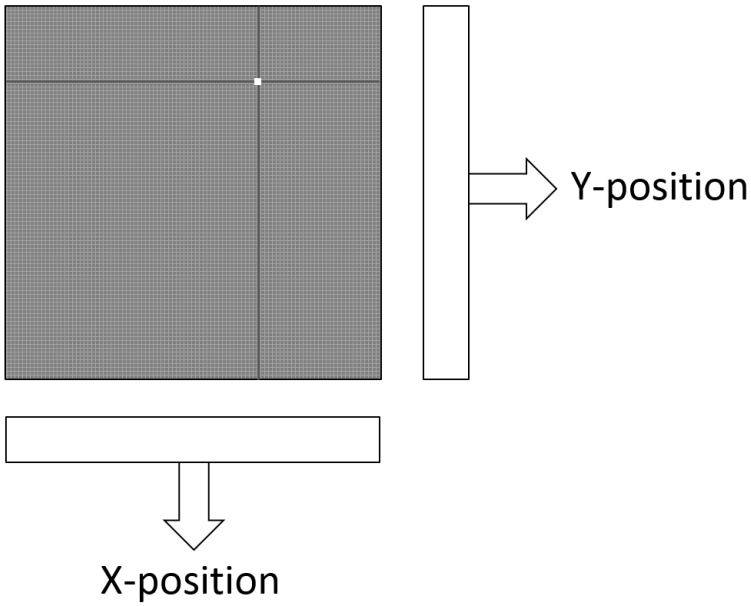

Figure 2: IRIS readout. When a pixel intensity reaches a threshold, its address is output.

Forests, and Random Ferns. This paper was conceived when we found that a special-purpose sensor would fit some of these classification methods well.

The paper is organized as followed. The Intensity-Ranking Image Sensor (IRIS) concept is described in Section 2. Section 3 gives a brief overview of classification using Decision Trees and Ferns. In Section 4, we show the usefulness of the IRIS for image classification using these methods. The results of simulations are given in Section 5 and conclusions are given in Section 7.

2 The IRIS concept

Here, we extend the idea of using time to represent light intensities, by combining it with Address-Event Representation (AER). AER has been described in a number of papers, see e.g. (5, 6), and recently, computer vision methods for AER sensors ("event-based vision") have attracted interest in the research community (e.g. (7, 8)) as well as for commercial implementations. The idea behind AER is to output the address of "events" as they occur in the image. Events could e.g. be any local activity, such as a corner point or local motion. In our case, and as described by Brajovic and Kanade (6), the pixels send out their row and column position globally as the comparator threshold is met, leading to the IRIS concept. Thus, the sensor may be viewed as having an inverse function of a traditional sensor where the read-out is controlled by row and column registers or shift registers which scans sequentially over all the pixels, to read-out their accumulated light levels, see Figure 2.

Provided that the light levels over the array differ sufficiently from each other that an external circuitry is able to save the position and time instant of each reporting pixel, the full image will eventually be collected. However,

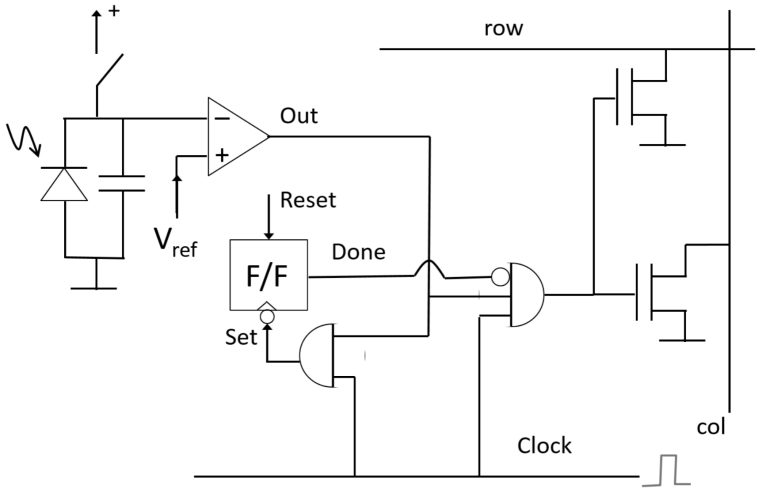

Figure 3: Example pixel circuit.

just as is the case with NSIP, we do not necessarily need the exact time instances. Instead, the relative times, giving information about the intensity relations between the pixels are good enough for our purpose. An example pixel circuit for this sensor is shown in Figure 3.

The pixel circuit consists of the light integrating photo diode as shown in Figure 1, a comparator and a latch holding the "Done"-bit. When the threshold level is reached, the comparator sends out a logical 1 that pulls the global row and column wires to ground. Row and column multiplexers (not shown) at the edge of the sensor chip deliver the pixel coordinates to the external circuitry. This is synchronized with the clock line that also sets the latch, indicating that the pixel has been read-out. This particular pixel will then stay passive until the latch has been reset at the beginning of a new image exposure.

3 Image Classification, Decision Trees, and Ferns

Automatic image classification is the process of assigning a label to an image depending on its content. Image classification is closely related to object detection, as this could be implemented as classification of image patches as object or non-objects (i.e., binary classification).

Image classification methods are often divided into two steps: Feature extraction and classification. First, simple or complex image features are extracted and represented by a feature vector or *image descriptor*. There is a plethora of such descriptors described in the literature, hand-crafted (such as HoG (9), LBP (25 variants in (10)), SIFT (11), SURF (12), BRIEF (13), ORB (14) and BRISK (15), just to mention a few) as well as learnt descriptors (16). A descriptor could be a vector of real (floating point) numbers or a binary vector. Second, a classifier operates on the extracted feature vector, outputting a discrete or binary class label. Examples of such classifiers are Support Vector Machines, Decision Trees, as well as ensemble classifiers (forests and cascades) that combine several simple classifiers

into one complex classifier, and using procedures like boosting to optimize the training (17). Recently, deep convolutional neural networks (CNNs)(18) have shown to be very capable, revolutionizing the computer vision, machine learning, and AI communities. A drawback of deep learning is that it is extremely computationally demanding, and is typically run on Graphics Processing Units (GPUs) with high power consumption. Here, we will focus on computationally inexpensive methods that can be run in short time on near-sensor hardware. Our interest is on binary descriptors that can be extracted and classified with low computational complexity and be implemented directly on the image sensor.

Decision Trees are well-known structures in computer science and machine learning, and provide a simple way to train a classifier from data. At each node in the tree, a test is made on some feature of the input data, until a leaf node is reached and a value pre-stored in that leaf node is used as output. This value is usually binary (positive/negative), but could be any integer or a class-conditional probability. The used features and tests can be complex (such as Support Vector Machines operating on HoG features (19)) or simple (comparison of two pixel values (20)). Simple features can be compensated by more complex trees, and vice versa. Since a single decision tree is easily overtrained, it is a common practice to use a large number of trees and average the results, that is, a *Decision Forest*. In a *Random Forest*, the features to be examined in the nodes are chosen randomly from a large set.

Ferns (21) can be seen as a special kind of Decision Trees, with the following properties

1. All leaf nodes are at the same depth.
2. The same test is used for each node on the same level in the tree.
3. Each test is a comparison between two pixel intensities.

Thus, in contrast to an unrestricted Decision Tree, it is known in advance how many tests will be made and based on which features (pixel values) these tests will be made. Thus, for a Fern with depth T, T tests are performed (pixel pairs compared) and the output is stored in a binary vector of length T. This vector can then be used as an index to a table where the $K = 2^T$ leaf node values are stored. Somewhat surprisingly, Ferns turn out to be as discriminative as Decision Trees (for details and underlying theory, see Ozuysal et al. (21)).

The structure of a Fern with T tests consists of a *Fern Input Table* with T rows telling which features to use (i.e., pixels to compare) at each level of the Fern and a *Fern Output Table* with $K = 2^T$ rows telling the output from the Fern given the results of the tests. In order to illustrate, an example of a 3-test Fern using pixel comparisons on 8×8 images is given in Figure 4.

When classifying an image $I(p)$ using the Fern in Figure 4, the pixel at position (6,0) is compared with the pixel at position (2,1), etc. The results of the comparisons are stored in a *Test Result Table*. Thus, to classify an image, the pixel values at the specified positions are extracted from the image, and the comparisons made.

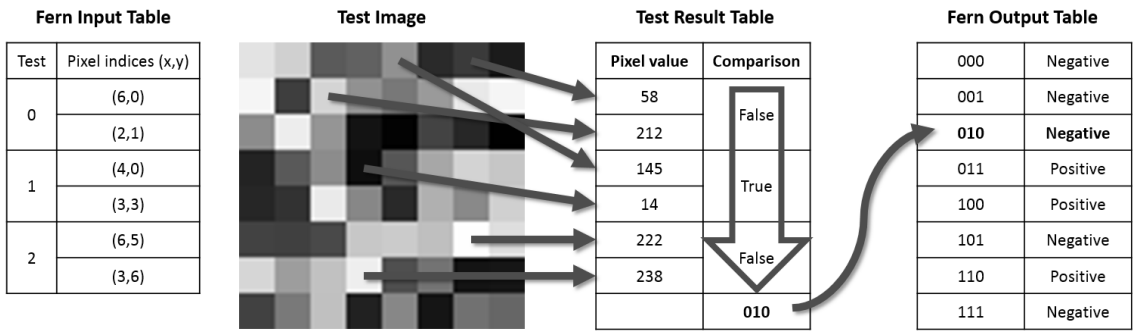

Figure 4: A Fern is defined by two tables; the *Fern Input Table* defining T tests and the *Fern Output Table* defining $K = 2^T$ outputs. In a real implementation, the first column of both tables is unnecessary. When a test image is classified, the values of the pixels with the indices given by the Fern Input Table are read from the image and compared, resulting in a binary vector, in this example {false, true, false}, i.e., 010. This binary vector is used as index to the Fern Output Table, resulting in a negative Fern output. In a real implementation, the pixel values do not need to be stored in the Test Result Table, they are shown here for the sake of clarity

Training a Random Fern is done in three steps:

1. Randomly assign pixel indices to the Fern Input Table (that is, pick pixel indices from a uniform distribution over the image size).

2. Apply the Fern to a set of labeled training images (that is, each image should have a label if it is a positive or negative example). For each leaf node, count the number of positive and negative training examples that evaluate to that node.

3. For each leaf node, if a larger ratio of positive than negative examples evaluated to that leaf node, set that row in the Fern Output Table to Positive (and vice versa).

The generalization to multi-class classification is trivial; let the Fern output table contain integers instead of Positive/Negative.

Alternatively, as advocated by Ozuysal et al. (21), the output value stored in the output table should be the estimate

$$p_{k,c_i} = \frac{N_{k,c_i} + R}{N_{c_i} + C \times R}, \tag{1}$$

of the class-conditional probability for the Fern output k given class c_i. C is the number of classes, N_{k,c_i} the number of training samples from class c_i that evaluated to leaf node k, N_{c_i} is the total number of training samples for class c_i, and R is a regularization parameter (usually set to 1, see (21) for details). The output table then needs a column for each class, as in Table 1. This is called the Naïve Bayes Approach.

When combining several Ferns into a Decision Forest, the two approaches above give two different decision mechanisms. If the output of the Fern is a single bit or an integer, the output of the Decision Forest is determined by

Table 1: A multi-class Fern Output Table, in this particular case the number of classes is 3.

Fern Output Table		
Output		
p_{0,c_1}	p_{0,c_2}	p_{0,c_3}
p_{1,c_1}	p_{1,c_2}	p_{1,c_3}
p_{2,c_1}	p_{2,c_2}	p_{2,c_3}
...	...	...

voting. In the binary case, we can just count the number of Ferns that gave a positive vote, and check if the count is more than half the number of Ferns. If the Naïve Bayes Approach is used and the output is given by Equation 1, then the output from the Decision Forest is the class p_{c_i} that maximizes the probability

$$p_{c_i} = \prod_{m=1}^{M} p_{k_m,c_i}, \tag{2}$$

where k_m is the k that Fern m evaluated to. In practice, this is implemented as a sum of logarithms and maximization of

$$\log p_{c_i} = \sum_{m=1}^{M} \log p_{k_m,c_i}. \tag{3}$$

4 Proposed Classification Method

In the following we will step-by-step modify the Random Fern algorithm and arrive at an algorithm that exploits the IRIS sensor in order to perform classification during image read-out.

4.1 Ferns Applied to Sorted Data

Assume an image sensor, as above, that outputs the pixel values in a sorted sequence, from high to low, during the image acquisition progress. That is, the pixel with the highest value (brightness) is read-out first, then, somewhat later, the pixel with the second-highest value, et cetera. The output from the sensor is a sequence of index-value pairs, sorted by decreasing value. In this case, a Fern could be slightly re-organized as to also contain an indicator whether each test has yet been made, such as in Table 2.

Note that the pixel values themselves are no longer interesting; a test is made simply by observing the order of the pixel indices, and, in fact the sequence of index-value pairs can be reduced to a sequence of indices.

To illustrate, let us use the same pixel values as in the above example (in Figure 4). The sensor delivers the list of pixel indices $\{ \ldots, (3,6), \ldots, (6,5), \ldots, (2,1), \ldots, (4,0), \ldots, (6,0), \ldots, (3,3), \ldots \}$

Table 2: A Fern Input Table and a Test Result Table for sorted data.

Fern Input Table	
p_1	p_2
(6,0)	(2,1)
(4,0)	(3,3)
(6,5)	(3,6)

Test Result Table	
Test: $I(p_1) > I(p_2)$?	Done
?	false
?	false
?	false

Table 3: A Test Result Table after arrival of the first pixel index (3,6).

Test Result Table	
Test: $I(p_1) > I(p_2)$?	Done
?	true
?	false
false	true

Each time a pixel index arrives, we check if that pixel index is present in our Fern Input Table, and, if it is, we update the Test Result Table. Thus when index (3,6) arrives, we update the Test Result Table as in Table 3. Slightly later, when index (6,5) arrives, we find that that test is already marked as done, and we make no update. When index (4,0) arrives, all three tests in the Fern are made, and we need not to care about the rest of the list of indices. We can read the test column ({false, true, false} $\rightarrow 010_{\text{binary}} \rightarrow 5_{\text{decimal}}$) and use it as the index to the Fern Output Table and get the result of the classification. Note that this is done before we have received all pixel indices from the sensor, that is, we can with high probability deliver a classification result before the entire image is read-out![1]

4.2 Forests and Groves

In a realistic scenario, we will have a forest of a large number of Ferns; tens or even hundreds. Instead of searching a large number of Fern Input Tables for the arriving pixel index, all used pixel indices can be stored in one table. Under the restriction that each pixel index is used only once in the entire forest, this table can simply be a list of tests where the one-dimensional pixel index $p = x + W \cdot y$, where W is the width of the image sensor, itself is used

[1] The only time when this is not the case, is when a test in a Fern uses two of the darkest pixels in the image, in which case the classification result might appear a few clock cycles after the last pixel.

Table 4: Top: The contents (rows) of a Grove Table defining the features of one Fern (with the number 13). Bottom: The Test Result Table for Fern #13 when pixel index 51 has arrived form the sensor.

Grove Table			
Pixel index	Fern	Test	Comp
4	13	2	true
6	13	2	true
10	13	0	false
27	13	1	false
46	13	1	true
51	13	0	false

Test Result Table #13	
$Test : I(p_1) > I(p_2)$?	Done
	false
	false
false	true

as test index. We call a forest under that restriction a *Grove*. We hypothesize that for images with high enough resolution, this can be enforced in the training phase without loss of classification performance.

Assigning the example Fern above as #13, the table entries for that Fern would be as in Table 4.

Note that we do not need to store which pixel the arriving pixel should be compared to. We only store if the comparison should be true or false if the pixel index arrives before the other pixel index in the pair, that is, what value should be inserted in the Fern Output Table. Thus, when pixel index 51 arrives from the sensor, the Comp value (false) is copied to left column at row 2 in the Test Result Table and the test is marked as done.

4.2.1 Binary Classification and Voting

Assuming binary classification and final decision by voting, it is extremely simple to implement the decision making. As the pixel indices arrive from the sensor, we keep track of the number of positive minus the number of negative votes from the Ferns. When this number is larger than the number of remaining Ferns (that is, ferns that have not yet delivered a vote), we stop the process and deliver a positive result (and analogously for negative result).

4.3 Summary of the Proposed Classification Method

By adopting the restriction of having the same test in all the nodes at the same level in a decision tree (that is, we turn the Tree into a Fern), we show how we can make an algorithm that classifies the image during read-out. By adding the restriction that each pixel index should be used in maximum one Fern in each Grove, we can make an

efficient implementation where all tests and decisions are made by simple table look-ups and integer arithmetics. The algorithm in pseudo-code can be seen as Algorithm 1 on page 20.

4.4 Related binary descriptors

The algorithm above can easily be modified to implement other binary descriptors. A Random Fern consists of uniformly distributed binary pixel difference tests. A BRIEF descriptor (13) is similar, but the random tests are drawn from a Gaussian distribution centered around the center of the image. Thus, just by changing the random assignments in the training, the Random Fern is changed into a BRIEF. The ORB (14) and BRISK descriptors (15) also consist of a set of binary tests, but include pre-processing in order to compensate for rotation, which makes them less suitable for the proposed architecture.

5 Experiments

5.1 Purpose and set-up

The purpose of our experiment is three-fold. First, we want to examine how Groves compare to the baseline method, that is, Forests of Random Ferns. Second, we want to compare the voting scheme to the Naïve Bayes scheme. As a performance measure, we chose the correct classification rate, and the simplest way of improving the classification performance is to increase the number of Ferns or Tests, which will increase the memory usage as the tables grow. Thus, we choose to evaluate the performance as a function of memory size.

Third, we want to investigate how fast the classification will be finished in relation to the image read-out.

The classifier is simulated by a C# program running on a Windows PC.

5.2 Data and training

We use the well-known CIFAR-10 dataset (22) which is a labeled subset of the Tiny Images Dataset (23). The CIFAR-10 dataset consists of 60 000 color (24-bit RGB) images with size 32×32 pixels in 10 classes, see Figure 5. There are 50 000 training images and 10 000 test images evenly distributed over the 10 classes. As our method works on pixel intensities only, we convert the images to grayscale by adding the red, green, and blue channels, resulting in a 10-bit integer number.

We train Ferns by randomly assigning tests (pairs of pixel indices) to the nodes, thus creating Grove tables or Fern Input Tables. Each Fern is trained independently of the others by applying it to each of the training images, counting the number of examples from each class that ends up in each leaf node, and computing the probability according to

Figure 5: The image classes in CIFAR-10 and 10 images from each class. Note the large in-class variations, making CIFAR a very challenging data set.

Equation 1. This value is either used when making the decision according to Equation 3 (Naïve Bayes) *or* used to give a binary vote from each Fern (Voting).

Since there is a random element in the training procedure, we perform all tests ten times and record the average classification accuracy and its standard deviation.

5.3 Results

We train Groves and Forests, varying the number of tests in the Ferns from 3 to 8 and the number of Ferns in the Forest or Grove from 10 to 80. We plot the average classification accuracies vs the memory requirements and compare the results for Forests and Groves, see Figure 6. We try both binary classification and multi-class classification; in the binary case, we select two random classes ("horse" and "ship") in the CIFAR-10 dataset, in the multi-class scenario, we train a 10-class classifier. Obviously, the latter is a much more difficult problem, which is also reflected in the classification results. On average, the Forest gives an improved classification accuracy of 0.0015 compared to Groves. This improvement is six times less than the standard deviation over our 10 trainings for each configuration (0.0095). We thus conclude that Groves perform as good as Forests.

Second, we use the Groves above to compare the Voting and the Naïve Bayes approaches. For the binary classification, the Naïve Bayes scheme gives an expected small, but seemingly quite consistent, performance improvement, see Figure 7 (left). Since we made ten tests on each point, we can relate it to the standard deviation; the difference between the Voting and Naïve Bayes approach varies between 0.5 and 1.8 standard deviations and in more than

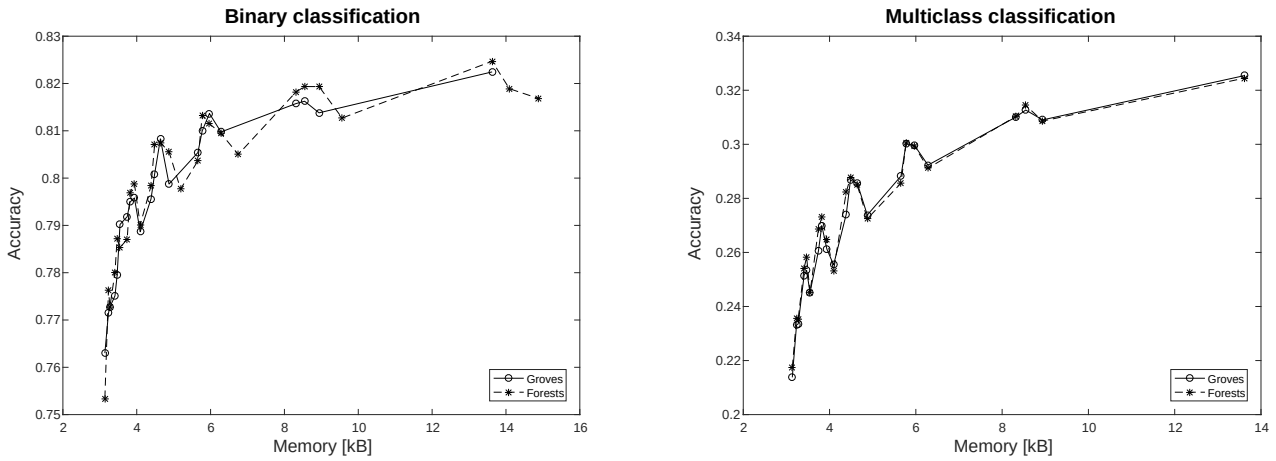

Figure 6: Classification accuracy of Groves vs Forests as functions of memory requirements.

half the cases it is less than one standard deviation. On average, the improvement is 0.0097 and the standard deviation 0.0100. Bear in mind also that the improvement comes with a higher computational cost since floating point arithmetics are needed.

For the multi-class case, the Naïve Bayes comes out significantly worse, see Figure 7 (right). This is due to the Fern Output Table's increased size as the number of classes grow.

Third, we train 10 binary classifiers, each with 40 Ferns with depth 6 (memory size 6 kB), that is, 240 binary tests on the 1024 pixels. We run the classifiers on 2000 test images, and for each of the 20 000 runs, we record how many pixels are needed to make a decision. On average, 96% of the pixels are needed; the full distribution is shown in Figure 8. For a simpler classification task, this number reduces; for example, with 80 binary tests, the average is 92%.

Compared to state-of-the-art results using convolutional neural networks, the results above are non-competitive. Relatively simple networks report classification accuracies of around 90%, however, even those simple networks have more than one million parameters and take several hours to train (25, 26). A Grove can be trained in minutes. In the next section, we will discuss requirements on the hardware.

6 Practical considerations

The proposed algorithm has been described and evaluated in terms of classification performance. However, to be interesting for practical use, additional aspects must be considered. Thus, in this section, we will address the performance in terms of speed (throughput), computational complexity and power consumption in comparison with neural networks, memory requirements, and sensitivity to sensor noise.

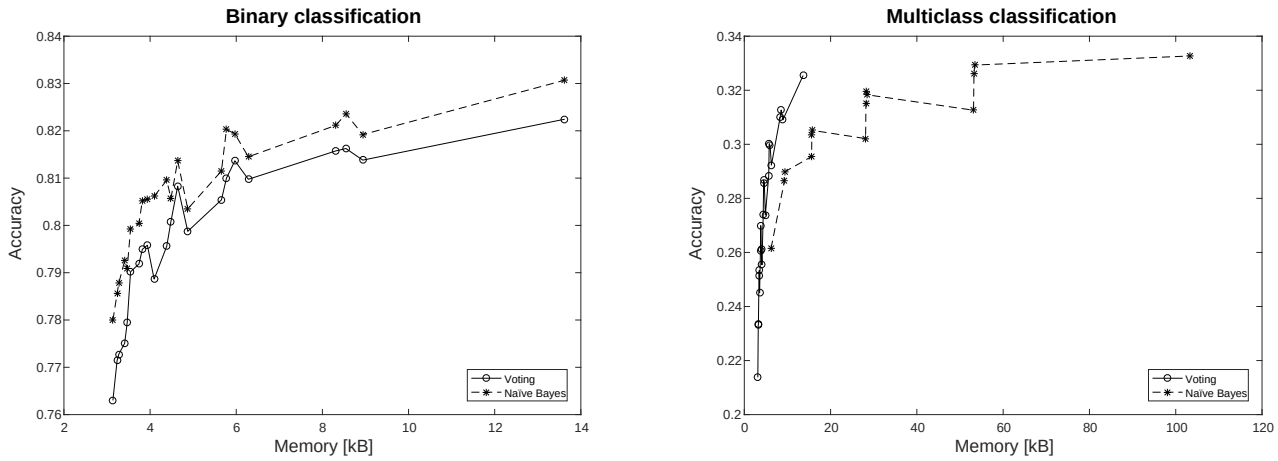

Figure 7: Classification accuracy of Groves using Voting or Naïve Bayes as functions of memory requirements.

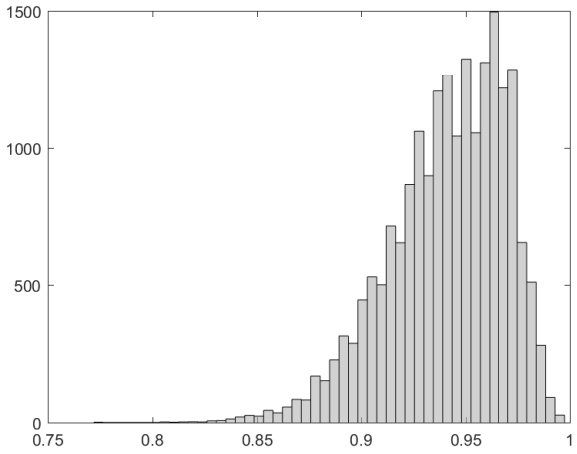

Figure 8: Distribution of the number of pixels needed to read.

6.1 Classification throughput

It is generally not possible to compute a specific value for the classification throughput (e.g. in terms of number of classified images per second) as it depends on several factors such as the available light (exposure time) and how many pixels are needed to reach a classification result.

With sufficient light level, the main bottlenecks will be the read-out rate of the sensor and the time to process the data. AER sensors described in the literature have read-out rates in the range of 13 to 20 Meps (24, 27). Assuming a sensor with 512×512 pixels, and that essentially all pixels need to be read-out, the data is available after less than 13 ms. A classical solution with a frame-based read-out, would obtain a similar value. However, such a solution would require associated external circuitry (full-frame memory, addressing circuits) which would add to the complexity and introduce additional delays.

Below, we make an effort to estimate the number of operations required, which will also give us a basic figure about the delay introduced by the processing.

6.2 Memory Requirements

The factors determining the memory requirements for the various tables are the number of Ferns (M), the number of tests in each Fern (T), and the number of pixels of the sensor (P).

The memory requirement of the algorithm is moderate; the only significant memory allocated during execution is for the two $M \times T$ arrays for the Test Result Table (growing linearly with the number of Ferns M and the number of tests per Fern T). The static storage of the Grove is somewhat more memory consuming; the Fern Output Table has $M \cdot 2^T$ entries and thus grows exponentially with T. Each entry of this table contains one binary digit, which typically is implemented as one bit or one byte.[2] The Grove Table has P entries, and thus grows linearly with the number of pixels of the sensor. In a typical implementation, each entry would require three bytes.

The total memory requirement (TMR) for a Grove using voting is thus

$$\mathrm{TMR}_{\mathrm{vote}} = M \cdot (2T + 2^T) + 3P \ \ [\mathrm{bytes}] \tag{4}$$

$$\approx M \cdot 2^T + 3P \ \ [\mathrm{bytes}]. \tag{5}$$

In practice, M is quite large (tens or even hundreds) and T is quite small (in the range $6 - 10$). For example, with a Grove of 100 Ferns with 10 tests and a 512×512 sensor, the memory requirement is 2 kB for the Test Result Table, 100 kB for the Fern Output Table, and 768 kB for the Grove Table. Note that less than one percent of the entries of the Grove Table is non-empty, and at a cost of computation time, the memory requirement could be reduced drastically.

If using the Naïve Bayes approach, the memory requirements increase, especially for multi-class classification when the Fern Output Table needs one column per class, and each entry is a floating point number (see Equation 1). With single precision numbers (4 bytes), the total memory requirement is

$$\mathrm{TMR}_{\mathrm{NB}} = M \cdot (2T + 4C \cdot 2^T) + 3P \ \ [\mathrm{bytes}] \tag{6}$$

$$\approx MC \cdot 2^{T+2} + 3P \ \ [\mathrm{bytes}]. \tag{7}$$

where C is the number of classes.

[2] One byte also accommodates for multi-class output.

6.3 Computational complexity and power consumption

In this section we estimate the complexity in terms of number of operations. We will further make a comparison with convolutional neural networks which represent the state-of-art in image classification. As above, we assume a Grove with M Ferns and T tests in each Fern, used to process an image with P pixels. A fraction $f \leq \frac{2MT}{P}$ of these pixels are used altogether by the Grove.

Following Section 4, each event from the image sensor requires one table look-up in the Grove Table and a check to see if the fern index is non-zero. In those cases, which appear $f \cdot P$ times during a frame exposure, further access is done to the selected Fern Table and the corresponding Test Result Table to see if the Done bit is set or not. On an average the Done bit is set in 50% of the cases. When not set, the Comp bit is copied into the Result column of the Test Result Table and the Done bit is set. When all the Done bits are set (which happens on an average $\frac{1}{T}$ times, the result bits are used to look up the contribution from that particular tree. The iterations are stopped when either a sufficient number of pixels have been processed or, if a sufficient number of trees vote for an unambiguous result.

Based on the assumption that a table look-up, a comparison, a write-to-memory operation, and an increment/decrement have similar complexity (e.g. one instruction cycle in a sequential processor), it is seen that the total number of cycles (TNC) will be in the order of

$$\mathrm{TNC}_{\mathrm{Grove}} = 2P + f \cdot P \cdot (2 + 2 \cdot 0.5 + \frac{4}{T}) \tag{8}$$

$$\leq 2P + 6MT + 8M. \tag{9}$$

This sum will be dominated by $2P$ for reasonable values of $\{P, M, T\}$, and thus $\mathrm{TNC}_{\mathrm{Grove}} \approx 2P$. Since the processing of the pixels is interleaved with the sensor readout, it is thus sufficient that the instruction rate is approximately twice as fast as the data rate from the sensor. If this is the case, the processing will not become a bottleneck.

Comparing with a convolutional neural network, we note that the first layer of such a network consists of a number of linear filters leading to

$$\mathrm{TNC}_{\mathrm{NN}} = a \cdot \frac{1}{s} \cdot b \cdot P \tag{10}$$

multiply-and-accumulate (MACC) operations, where a is the number of coefficients in the filters, s is the downsampling factor ("stride") and b is the number of different filters. Typically $a \approx s \approx 10$ and $b \approx 10$ so that the expression can be simplified to $10P$.

The next layer is usually a max-pooling layer where the $\frac{bP}{s}$ filtered outputs are compared to find the maximum value within an environment, such as a 5×5 window. This will lead to $\frac{25bP}{s} \approx 25P$ additional operations. Subsequent

layers introduce similar computations but we can discard their contributions since the data size is much smaller, that is, the amount of computations will be dominated by the first layers and thus $\mathrm{TNC}_{\mathrm{NN}} \approx 35P$.

As is seen, the complexity of the neural network in terms of numbers of operations ($N_{\mathrm{NN}} > 35P$) is at least an order of magnitude higher than for the Grove ($N_{\mathrm{Grove}} \approx 2P$). Furthermore a large part of the computations consists of MACC operations which are typically in themselves much more complex than table look-ups and comparisons. If the processing is done by a Complementary Metal Oxide Semiconductor (CMOS)-based CPU, this translates directly to a corresponding relation in energy between the two systems.

Dedicated neural network chips are beginning to emerge, such as the Intel Movidius Myriad (28). Due to their high degree of parallelism, they can use lower clock rates which benefit their power requirements. On the other hand, to save space, they need to implement the MACC operations in hardware, which partially offsets this gain. The above estimations are exemplified by the power requirements of a micro-controller system that is able to run the proposed algorithm at 10 frames/s using < 80 mW (CPU + memory), versus the mentioned neural network chip which requires around 1000 mW.

The typical hardware requirements in terms of memory, number of operations, and power consumption on commercially available hardware are summarized in Table 5.

Table 5: Typical hardware requirements.

Method	**Memory [bytes]**	**Complexity (no. of operations)**	**Power [mW]**
Grove (voting)	$M \cdot 2^T + 3P$	$2P + 6MT + 8M \approx 2P$	80
Grove (Naïve Bayes)	$MC \cdot 2^{T+2} + 3P$	$2P + 6MT + 8M \approx 2P$	80
CNN	Dependent on architecture	$> 35P$	1000
M: #Ferns in each the Grove. T: #tests in each Fern. C: #classes. P: #pixels.			

6.4 Sensitivity to sensor noise

Early AER sensors suffered from high fix-pattern noise (FPN). Compared to frame-based sensors, it is more complicated to compensate for the individual variations between the pixels. To do so would require compensation circuits to be integrated into every pixel. Recent AER implementations, however, show a sufficiently low FPN figure which avoids the need for compensation. As examples, Kim and Culurciello (24) reports FPN values at 0.18% and Posch et al.(29) at 0.25%. These numbers are low enough to yield more than 9 bits of precision both in light intensity and in event addresses (6).

The pixels of the used training and testing images are represented as 10-bit integers (as explained in Section 5.2). In order to evaluate the sensitivity to noise, an additional experiment was performed, where the lower bits of the

pixels were subject to noise. This was done by setting the least significant bits to randomly and equiprobably to 0 or 1. A Grove with 40 Ferns with 8 tests each was used for multi-class classification; the results are given in Table 6.

In conclusion, the proposed method is surprisingly resilient to noise, and keeps its performance way beyond any realistic noise level for an intensity-ranking sensor.

Table 6: Classification accuracy on noisy data.

Precision (#bits)	**Multi-class**	**Binary**
10	0.33	0.83
9	0.32	0.83
8	0.32	0.83
7	0.32	0.82
6	0.32	0.82
5	0.32	0.82
4	0.32	0.82
3	0.31	0.81
2	0.28	0.79
1	0.21	0.69

7 Conclusion

We have introduced a new type of image sensor and shown how image classification can be performed on such a sensor during image read-out. With minor modifications to Random Ferns, an established image classification technique, we can achieve classification results even before the entire image is read from the sensor, with an insignificant loss in classification performance. We also compare two decision mechanisms and show how a simple voting scheme allow us to avoid any floating point operations at a minor cost in classification performance. Compared to state-of-the-art image classification methods based on convolutional neural networks, the proposed method compensates its lower accuracy with extremely low hardware requirements, in terms of processing power as well as memory. Finally, the proposed method is shown to be resilient to any level of realistic sensor noise.

Acknowledgement

The research was partly funded by the Swedish Research Council (Vetenskapsrådet) through the project Energy Minimization for Computational Cameras (2014-6227).

ORCID

Jörgen Ahlberg http://orcid.org/0000-0002-6763-5487

Robert Forchheimer http://orcid.org/0000-0002-8382-2725

References

1. Forchheimer R, Åström A. Near-Sensor Image Processing. A New paradigm. *IEEE Transactions on Image Processing*. 1994;3(6):735–746 https://doi.org/10.1109/83.336244.

2. Åström A, Forchheimer R. Near-Sensor Image Processing. *Advances in Imaging and Electron Physics*. 1999;105 https://doi.org/10.1016/S1076-5670(08)70175-7.

3. Forchheimer R, Ödmark A. A Single Chip Linear Array Picture Processor. In: Proc. SPIE 0397, Applications of Digital Image Processing V; 1983; Geneva, Switzerland. http://doi.org/10.1117/12.935332.

4. Eklund J-E, Svensson C, Åström A. Implementation of a Focal Plane Processor—A Realization of the Near-Sensor Image Processing Concept. *IEEE Transactions on VLSI Systems*. 1996;4 https://doi.org/10.1109/92.532033.

5. Leñero-Bardallo JA, Serrano-Gotarredona T, Linares-Barranco B. A 3.6 μs Latency Asynchronous Frame-Free Event-Driven Dynamic-Vision-Sensor. *IEEE Journal of Solid-State Circuits*. 2011;46(6):1443–1455 https://doi.org/10.1109/JSSC.2011.2118490.

6. Brajovic V, Kanade T. A VLSI sorting image sensor: global massively parallel intensity-to-time processing for low-latency adaptive vision. *IEEE Transactions on Robotics and Automation*. 1999;15(1):67–75 https://doi.org/10.1109/70.744603.

7. Gallego G, Rebecq H, Scaramuzza D. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2018; Salt Lake City, UT:3867–3876.

8. Mueggler E, Bartolozzi C, Scaramuzza D. In: British Machine Vision Conference (BMVC); 2017; London.

9. Freeman WT, Roth M. *Orientation Histograms for Hand Gesture Recognition*. TR94-03: Mitsubishi Electric Research Laboratories, Cambridge, MA; 1994.

10. Bouwmans T, Silva C, Marghes C, Zitouni MS, Bhaskar H, Frelicot C. *On the Role and the Importance of Features for Background Modeling and Foreground Detection*. https://arxiv.org/abs/1611.09099v1; 2016

11. Lowe DG. Object recognition from local scale-invariant features. In: International Conference on Computer Vision (ICCV); 1999; Kerkyra, Greece:1150–1157. https://doi.org/10.1109/ICCV.1999.790410.

12. Bay H, Ess A, Tuytelaars T, Van Gool L. SURF: Speeded Up Robust Features. *Computer Vision and Image Understanding (CVIU)*. 2008;110(3):346–359 https://doi.org/10.1016/j.cviu.2007.09.014.

13. Calonder M, Lepetit V, Strecha C, Fua P. BRIEF: Binary Robust Independent Elementary Features. In: European Conference on Computer Vision (ECCV), Lecture Notes in Computer Science, vol 6314; 2010. https://doi.org/10.1007/978-3-642-15561-1_56.

14. Rublee E, Rabaud V, Konolige K, Bradski GR. ORB: An efficient alternative to SIFT or SURF. In: International Conference on Computer Vision (ICCV); 2011; Barcelona, Spain:2564–2571. https://doi.org/10.1109/ICCV.2011.6126544.

15. Leutenegger S, Chli M, Siegwart RY. BRISK: Binary robust invariant scalable keypoints. In: International Conference on Computer Vision (ICCV); 2011; Barcelona, Spain:2548–2555. https://doi.org/10.1109/ICCV.2011.6126542.

16. Markuš N, Pandžić IS, Ahlberg J. Learning Local Descriptors by Optimizing the Keypoint-Correspondence Criterion. In: International Conference on Pattern Recognition (ICPR); 2016; Cancun, Mexico:2380–2385. https://doi.org/10.1109/ICPR.2016.7899992.

17. Bishop CM. *Pattern Recognition and Machine Learning*: Springer-Verlag New York; 2006.

18. Goodfellow I, Bengio Y, Courville A. *Deep Learning*: MIT Press; 2016. http://www.deeplearningbook.org. Accessed September 15, 2017.

19. Dalal N, Triggs B. Histograms of oriented gradients for human detection. In: International Conference on Computer Vision and Pattern Recognition (CVPR); 2005; San Diego, CA:886-893. https://doi.org/10.1109/CVPR.2005.177.

20. Markuš N, Frljak M, Pandžić IS, Ahlberg J, Forchheimer R. *Object Detection with Pixel Intensity Comparisons Organized in Decision Trees*. https://arxiv.org/abs/1305.4537; 2013

21. Ozuysal M, Calonder N, Lepetit V, Fua P. Fast Keypoint Recognition Using Random Ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2010;32(3):448–461 https://doi.org/10.1109/TPAMI.2009.23.

22. Krizhevsky A. *Learning Multiple Layers of Features from Tiny Images*: MSc Thesis, University of Toronto; 2009. https://www.cs.toronto.edu/~kriz/cifar.html. Accessed September 15, 2017.

23. Fergus R, Torralba A, Freeman WT. *Tiny Images Dataset*: New York University. http://horatio.cs.nyu.edu/mit/tiny/data. Accessed September 15, 2017

24. Kim D, Culurciello E. Tri-Mode Smart Vision Sensor With 11-Transistors/Pixel for Wireless Sensor Networks. *IEEE Sensors Journal*. 2013;13(6):2102–2108 https://doi.org/10.1109/JSEN.2013.2249061.

25. Springenberg J T, Dosovitskiy A, Brox T. Striving for Simplicity: The All Convolutional Net; 2014. http://arxiv.org/abs/1412.6806.

26. Lee C-Y, Xie S, Gallagher P, Zhang Z, Tu Z. Deeply supervised nets. In: International Conference on Artificial Intelligence and Statistics (AISTATS), San Diego, CA. Proceedings of Machine Learning Research, Vol. 38; 2014:568–570. http://proceedings.mlr.press/v38/.

27. Serrano-Gotarredona T, Linares-Barranco B. 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 Journal of Solid-State Circuits*. 2013;48(3):827–838 https://doi.org/10.1109/JSSC.2012.2230553.

28. *Intel Movidius Myriad*. https://www.movidius.com. Accessed: 2018-05-17

29. Posch C, Matolin D, Wohlgenannt R. A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. *IEEE Journal of Solid-State Circuits*. 2011;46(1):259–275 https://doi.org/10.1109/JSSC.2010.2085952.

Algorithm 1 Binary image classification using a Voting Grove of Ferns on a sorting sensor

Parameters:
integer M ▷ The number of Ferns.
integer T ▷ The number of tests in each Fern.
integer P ▷ The number of pixels in the image.
integer$[M, K]$ $FernOutput$ ▷ The Fern Output Tables.
tuple$<Empty, Fern, Test, Comp>[P]$ $Grove$ ▷ The Grove Table.

Input:
integer[P] $\{p_1, \ \ldots, \ p_P\}$ ▷ A list of pixel indices sorted after pixel value.

Output:
A Positive/Negative classification of the input image.

function CLASSIFY($p_1, \ldots, p_P$)
Variables:
bool$[M, T]$ $FernTestResult$ ▷ The "Test" columns of the Test Result Table.
bool$[M, T]$ $FernTestDone \leftarrow false$ ▷ The "Done" columns of the Test Result Table.
integer $TotalVote \leftarrow 0$ ▷ The number of positive minus the number of negative votes.
integer $FernsToGo \leftarrow M$ ▷ The number of Ferns not yet evaluated.
integer m, t, k ▷ Fern index, test index, and test result.

for $j = 1, \ldots, P$ **do**
if $Grove[p_j].Empty$ **then**
continue ▷ This pixel index is not used; continue with the next pixel index.
end if
$m \leftarrow Grove[p_j].Fern$
$t \leftarrow Grove[p_j].Test$
if $FernTestDone[m, t]$ **then**
continue ▷ The test is marked as done; continue with the next pixel index.
end if
$FernTestResult[m, t] \leftarrow Grove[p_j].Comp$ ▷ Copy the comparison sign as the test result.
$FernTestDone[m, t] \leftarrow$ true ▷ Mark the test as done.

if $FernTestDone[m, i] \ \forall \ i \in \{1, \ldots, T\}$ **then** ▷ If all tests in the Fern are marked as done.
$k \leftarrow [FernTestResult[m, 1], \ldots, FernTestResult[m, K]]$ ▷ Binary vector as integer.
$TotalVote \leftarrow TotalVote + FernOutput[m, k]$ ▷ Accumulate the votes.
$FernsToGo \leftarrow FernsToGo - 1$
end if

if $TotalVote > FernsToGo$ **then**
return Positive ▷ A majority of the Ferns voted for a positive result.
end if
if $TotalVote \leq -FernsToGo$ **then**
return Negative ▷ A majority of the Ferns voted for a negative result.
end if
end for
end function

How cite this article: J. Ahlberg, A. Åström, and R. Forchheimer (2018), Simultaneous Image Sensing, Read-Out, and Classification, *Int. J. Circ. Theor. Appl.*, *2018*. https://doi.org/10.1002/cta.2549