DVSGesture Recognition with Neuromorphic Observation Reduction Techniques

Charles P. Rizzo
Luke McCombs
Braxton Haynie
Catherine D. Schuman
James S. Plank

August, 2023

ACM International Conference on Neuromorphic Computing Systems

Copyright ACM. The official ACM version of this paper will be referenced here when it is published by the ACM.

The online home for this paper may be found at: http://neuromorphic.eecs.utk.edu

Bibtex:

@INPROCEEDINGS{rmh:23:drw,
    author = "C. Rizzo and L. McCombs and B. Haynie and C. D. Schuman and J. S. Plank",
    title = "DVSGesture Recognition with Neuromorphic Observation Space Reduction Techniques",
    booktitle = "International Conference on Neuromorphic Computing Systems (ICONS)",
    publisher = "ACM",
    year = "2023"
}
DVSGesture Recognition with Neuromorphic Observation Space Reduction Techniques

Charles P. Rizzo
crizzo@vols.utk.edu
Department of EECS
University of Tennessee
Knoxville, Tennessee, USA

Luke McCombs
lmccombs@vols.utk.edu
Department of EECS
University of Tennessee
Knoxville, Tennessee, USA

Catherine D. Schuman
cschuman@utk.edu
Department of EECS
University of Tennessee
Knoxville, Tennessee, USA

James S. Plank
jplank@utk.edu
Department of EECS
University of Tennessee
Knoxville, Tennessee, USA

ABSTRACT

Event-based cameras and classification datasets pair nicely with neuromorphic computing. Furthermore, it is attractive from a SWaP perspective to have a fully neuromorphic pipeline from event-based camera output to classification instead of having to preprocess the camera data prior to classification. In this work, we examine how two neuromorphic observation space reduction techniques impact classification performance on the DVSGesture dataset. The two techniques can be implemented as spiking neural networks so that no preprocessing of the camera data is required, and instead, only the routing of the events to the proper input neurons is necessary.

1 INTRODUCTION

Event-based cameras have been dubbed neuromorphic sensors because of their sparse, event-driven output. They are composed of collections of asynchronous pixels that capture changes in light intensity and emit an event (of either positive or negative polarity) when the light luminosity change at a pixel location exceeds a certain threshold. This output modality mimics the spike-based, asynchronous computation methodology that is neuromorphic computing. As an emerging non-von Neumann computing architecture, neuromorphic computing features spiking neural networks that transmit information temporally through spike propagation. This more closely mimics the biology of the human brain which still remains to be the most efficient "computer" that we have – than traditional neural networks do.

A large part of the appeal of neuromorphic computing and its hardware is its low Size, Weight, and Power (SWaP) requirements. Neuromorphic hardware like IBM’s TrueNorth [1], Intel’s Loihi [5], SpiNNaker [7], and several others have become standardized in the research community. This hardware does not naturally implement the traditional, large deep neural networks that have stemmed from the work of Yann Lecun [11], though it is possible to map these types of networks onto many neuromorphic systems. Instead, the hardware supports smaller spiking neural networks; however, until recently, these spiking neural networks have been difficult to train because they are temporal and not necessarily organized neatly in layers as traditional neural networks are. This makes traditional learning with backpropagation difficult or impossible, depending on the structure.

As spiking neural network development techniques emerged, it became evident that event-based camera data would be a good fit for this type of neural network. Using a DVS128 camera by Invataion [12, 13], Amir et al. [2] created the DVSGesture dataset: a dataset featuring 11 hand and arm gestures performed by 29 different users under 3-5 lighting conditions. The 11 gestures include hand waving (both hands), arm rotations for both arms in both the clockwise and counterclockwise directions, forearm rolling, air drumming, air guitar, and an additional "Other" gesture invented by each user. Each gesture lasts for about 6 seconds and is separated by some length of time during which no gesture occurs. The format of the data is simply a stream of either positive or negative polarity events that occur at pixel \((x,y)\) at a certain time \(t\). Unlike other DVS datasets that are conversions from contemporary, frame-based datasets and lack the high temporal resolution of event-based cameras, the DVSGesture dataset (and the few others like it) have...
the high temporal resolution of dynamic vision sensors built in, as well as the natural noise that occurs when collecting data on the physical camera.

2 RELATED WORK

Classifying gestures is not a novel idea. Over the years, there has been work with deep neural networks like ConvNet to classify gestures as conventional frames [24] and even optical flow [22]. There have also been solutions that apply Recurrent Neural Networks (RNNs) [3] and long-short term memory (LSTM) [9] layers to the same problem. As is usually the case with deep neural networks (DNNs) in general, classification performance is not an issue. However, the unwieldy size of these models has made them unfeasible for implementation in edge-computing hardware. Spiking neural networks (SNNs) have been another tool for gesture recognition and have appeared in notable works like [6, 16, 23]. These works often use techniques like Spike Time Dependent Plasticity (STDP) [15], spiking backpropagation [21], and even reservoir computing models [10, 14] to train SNNs.

On the DVSGesture dataset specifically, Amir et al. [2] introduced an architecture featuring a 16-layer convolutional neural network capable of gesture classification accuracy upwards of 96%. Their architecture is composed of temporal cascade filters, convolutional layers, a Winner-take-all (WTA) decoding layer, and a sliding window filter that cleans up the instantaneous gesture classifications. What is most impressive about the architecture is that they managed to put the entirety of it on a TrueNorth [1] chip and introduce a hardware demonstration where gestures were captured by a DVS128 camera and then classified by the network on the chip. Their setup had only a 105 ms latency between gesture and prediction that facilitated 1000 classifications per second, and as a whole, their system consumed less than 200mW of power. While this setup is impressive, preprocessing was still necessary to convert the events from the camera into frame-based representations, and furthermore, the network was not actually trained on TrueNorth.

More recent work on the DVSGesture dataset was performed by George et al. [8] where convolution was used to determine spatial features and a spiking reservoir was used to determine temporal features. They created event-frames in order to apply the convolutional layers, and they also performed temporal compression of the event stream since the DVSGesture dataset captures events at a 10⁶ temporal resolution. In their work, they predicted on 8 out of the 11 provided classes (omitting the counterclockwise arm rotations and the unique per user “Other” gesture) which made testing accuracy raise from 59% on the full 11 class dataset to 88%. Their results show testing accuracies on the full 11 class experiment in the range of 59-65%, and their total network parameter counts for this level of performance range from 800,000 to 3.2 million.

Another more recent work on classifying the DVSGesture dataset was performed by Xing et al. [25]. It features a spiking convolutional recurrent neural network trained with Slayer and boasts a full 11 class testing accuracy of 90% and a 10 class testing accuracy (omitting the “Other” gesture) of 96.5%. However, like with George et al. and Amir et al., the solution involved data preprocessing in the form of aggregating events into event-frames in order to apply convolution. Furthermore, none of the prior work has addressed the time that occurs in between gestures in the event stream dataset. It appears that classification is performed only on the event activity that occurs within the predefined gesture time windows, and any event activity that falls outside of those time windows is disregarded.

This work aims to classify the DVSGesture dataset in a slightly different way. By applying neuromorphic computing techniques, we can avoid any event-frame aggregation preprocessing between the DVS camera and the spiking neural network. This provides for a solution, similar to that which is presented by Amir et al., where a neuromorphic network can be loaded onto neuromorphic hardware and placed directly behind the DVS camera, performing classification in real-time. We also address the impact of the transition periods between gestures and choose not to ignore them. We put pressure on the classifier to recognize periods of inactivity or transition and recall them in addition to the other gestures that compose the dataset. To facilitate this goal, we apply neuromorphic preprocessing techniques to the DVSGesture dataset and evaluate their impact on a random forest classifier’s performance. We use the random forest classifier because decision trees, and by extension random forests, have been implemented neuromorphically [19]. To date, no work examining the impact that neuromorphic observation space reduction techniques have on DVS-based classification has been performed.

3 NEUROMORPHIC SPATIAL REDUCTION TECHNIQUES

3.1 Spiking Threshold Pooling

The spiking threshold pooling technique is a way to downsample DVS camera data by applying the traditional pooling operation over the camera window but in a spiking manner. At a high level, the camera’s output space (128 × 128 in the case of the DVSGesture dataset) is divided into chips or sub-regions that are valued at either 0 or 1 depending on the amount of events that occur within that region. We want to stress that we are using this technique as a means of organizing the events as they come from the camera. We are not building out event-frames in memory and striding over them performing convolution, as other works have done [2, 8, 25]. Our goal is to preserve the asynchronous, event-based characteristic of DVS camera data.

As an example, suppose we define an event counting network where the chip size is 10×10, the stride value is 10, and the threshold value is 5. For the 128×128 observation space of the DVS128 camera, when we apply the spiking threshold pooling operation, we reduce the observation space to

\[
\text{ceil} \left( \frac{128}{10} \right) \times \text{ceil} \left( \frac{128}{10} \right) = 13 \times 13 = 169
\] (1)

binary-valued observations. The chip value is 1 (or ‘of interest’) if five or more events occur in that sub-region of the camera’s observation space over some length of time. It is 0 otherwise. Effectively, the counting network is doing nothing more than counting or thresholding event activity at a certain region over some time interval. The benefits of this technique are that the counting network can be tiled and organized into a layer, producing a downsampling input
signal that can be used as either inputs to successive downsampling passes or to a classification agent. This technique can be implemented neuromorphically which facilitates real-time classification by placing layers of downsampling networks behind a camera’s output to reduce the input space for the final classifier network. Readers are referred to [18] for more details on the network variants and an example use case in control theory.

3.2 Row/Column Collapse

The row/column collapse technique flattens a two dimensional observation space by effectively performing an ‘OR’ operation across the rows and columns. This would be applied after a series of zero or more of spiking threshold pooling (or downsampling) layers and immediately precede the classifier. Continuing from the example shown in Equation 1, suppose after the sole layer of downsampling that we perform a row/column collapse. The $13 \times 13$ observation space is then transformed to a $13 + 13$ sized observation space. Figure 1 shows what this operation looks like intuitively.

In the case of the DVSGesture dataset, different users perform the “right hand wave” gesture at different heights. The classifier might classify waving at the top of the observation space as something different from waving at the bottom of the observation space. With this technique, the rows and columns are flattened so that the signal for waving at the top of the observation space is the same as waving at the bottom of the observation space. Another benefit, like with the spiking threshold pooling mechanism, is that this technique can be performed by a spiking neural network which further facilitates a real-time, neuromorphic classification pipeline.

The spiking neural network for this technique, assuming an $r \times c$ observation size immediately preceding it ($I_{r,c}$), contains $r + c$ neurons and $2 + r + c$ synapses. The $r + c$ neurons are both input and output neurons, and they’re defined as two sets:

1. $N_{0,c-1}$
2. $N_{r+1,c-1}$

Similarly, there are two sets of synapses defined as:

1. For $0 \leq i < r$ and $0 \leq j < c$, $I_{i,j}$ to $N_{0,k-1}$
2. For $0 \leq i < r$ and $0 \leq j < c$, $I_{i,j}$ to $N_{k+1,c-1}$

The neuronal thresholds are all 1 and the the synaptic weights and delays are also all 1. Neurons $[0,c - 1]$ each have $r$ incoming synapses, and neurons $[c, r + c - 1]$ each have $c$ incoming synapses. As the example in Figure 1 shows, if a column or row in the observation space has 1 or more chips of interest (or events if this is applied directly behind the camera’s output), that column or row is represented by a neuron fire (a red 1 in Figure 1c) in the collapse row/column network.

4 DATASET PREPROCESSING

The DVSGesture dataset is presented as a collection of .aedat files generated by a DVS128 camera and corresponding label files that detail which of the eleven possible gestures is occurring between two timestamps, which are of microsecond temporal precision. Each .aedat file is the recorded event stream of a user in a specific lighting condition stored in the Address Event Representation (AER) [4] format, and initially for our work, we convert each file to a CSV representation where each event is represented by the following quadruple: $x$, $y$, timestamp, polarity. For our work, we shift the timestamps of the events so that the first event in a file occurs at timestamp 0 and all successive events occur at timestamps relative to the first event in the file. An example of the first five events and their conversions from the user 1 fluorescent lighting file is shown below.

$56,78,80046394,0$ $\rightarrow$ $56,78,0,0$
$63,78,80046412,0$ $\rightarrow$ $63,78,18,0$
$46,90,80046414,0$ $\rightarrow$ $46,90,20,0$
$69,87,80046427,0$ $\rightarrow$ $69,87,33,0$
$69,89,80046433,0$ $\rightarrow$ $69,89,39,0$
$\ldots$
For every user and lighting combination file, we create time series data samples. Borrowing from notation used in [25], we define \( \text{time\_segmentation\_length} \) as the length of time in microseconds that comprises a segment. For example, if a dataset file is 114170024 microseconds long, and we segment the file with \( \text{time\_segmentation\_length} = 5000 \), the result is

\[
\text{floor} \left( \frac{114170024}{5000} \right) = 22834
\]

segments. The first segment in the file contains the events that occur within the first 5000 microseconds (or 5 milliseconds). The second segment in the file contains the events that occur between microseconds [5000,9999], and so on. Furthermore, we define an additional parameter called \( \text{segments\_per\_sample} \) that dictates how many segments compose one timeseries sample. If we assume \( \text{segments\_per\_sample} = 150 \), then a timeseries sample is composed of 150 5000-microsecond long segments. In other words, one timeseries sample is 0.75 real-time seconds. To determine the number of timeseries samples in our example file, we do the following:

\[
\text{floor} \left( \frac{114170024}{5000} \right) = 152
\]

Our example file has 152 time series samples where the first 5000-microsecond segment in the file has time 0, the next segment has time 1, and so on up until segment 149, which has time 149. Then, at segment 150, we start a new timeseries sample, so segment 150 has time 0, segment 151 has time 1, and so on. This temporal compression is similar to what George et al. [8] do in order to gain richer spatial information through a more dense event collection at each timestep. The label for each time series sample is determined by whatever gesture window its last segment’s last event’s original timestamp occurs in. Because of the way the DVSGesture dataset is composed, there are lengths of time between each gesture that are unlabelled that we call transitional. The way we form our dataset makes it possible for a time series sample to start in this transitional time window (or even the previous gesture’s time window) and end in a different gesture’s time window. In the spirit of advocating for a more real-time, event stream approach of processing and classifying the DVSGesture dataset, we argue that this characteristic is realistic for classifying in real-time with a network right behind a camera.

Additionally, we alter the labelling scheme slightly. In the DVSGesture dataset, label 11 corresponds to the "Other" gesture that is different from user to user. It has been shown in many prior works that this label complicates classification and yields lower accuracies [8, 25]. Like these prior works, we choose to remove this label and its samples; instead, we assign label 11 to any time series sample that end in the transitional period of time between gestures. We believe this is a fair trade-off: instead of trying to classify a gesture that differs from user to user, we try to classify the periods of time between gestures as “transitional” and not one of the other ten standardized gestures. This change further supports the goal of classifying DVS camera data in real-time.

For our work, we use the the train/test user split recommended by Amir et al. [2]. This results in users 1-23 and all of their lighting conditions composing the training dataset and users 24-29 and all of their lighting conditions composing the testing dataset. While there are five lighting conditions possible, not every user is captured under every lighting condition. Table 1 shows which user and lighting condition combinations are not included in the dataset.

## 5 HYPERPARAMETER GRID SEARCH AND EVALUATION

Our goal is to determine the effect of different neuromorphic spatial reduction techniques on a random forest classifier’s performance on our version of the DVSGesture dataset. We perform a small grid search to examine how different downsampling parameters and layer combinations influence the classifier’s performance on the testing dataset.

While both the \( \text{time\_segmentation\_length} \) and \( \text{segments\_per\_sample} \) values are parameterizable, changing them fundamentally changes event sparsity in the dataset. This work focuses more closely on the effect of downsampling parameters on a particular dataset and its event sparsity and not the impact of different levels of event sparsity. Table 2 shows the combinations evaluated.

To limit the size of our search, we only work with square chip sizes and stride values that are either equal to the chip size or half of it. We also only test up to two layers of downsampling and constrain the parameters for the second layers of downsampling. As an example, the first downsampling layer may count either events or unique pixels that fire in a region, but any secondary downsampling layer will only count the amount of chips from the first layer that “fire,” and not the number of unique chips in its region that fire. Additionally, the threshold value for the second downsampling layer is always 1. However, the second downsampling layer features an additional chip size of 5×5 that is not used in any first downsampling layer.

<table>
<thead>
<tr>
<th>Lighting</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural</td>
<td>7, 8, 10, 12, 18, 20, 23, 24, 25</td>
</tr>
<tr>
<td>Fluorescent</td>
<td>12, 16, 25</td>
</tr>
<tr>
<td>LED</td>
<td>11, 21</td>
</tr>
<tr>
<td>Fluorescent LED</td>
<td>3, 4, 11, 12, 14, 20, 24, 25, 27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chip Sizes</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>5×5**, 10×10, 15×15, 20×20, 25×25</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strides</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{chip_size} ), ( \text{chip_size} )</td>
<td></td>
</tr>
</tbody>
</table>

| Thresholds*  | 10, 20, 30                    |
| Count Type*  | Event, Pixel                  |
| Layers       | 1, 2                          |
| Row/Col Collapse | True, False                |
| Transitions  | True, False                  |
There are some combinations that we define as illegal and leave out of the experiment. An example of an illegal parameter scheme would be where we have a first downsampling layer with chip size of $25 \times 25$ and a stride value of 25. This would reduce our observation to $(128 - 25) \times (128 - 25) = 6 \times 6$. If we use a second layer of downsampling where the chip size is $10 \times 10$, we do not have enough observations to fully occupy one chip. We declare this combination of parameters illegal. With this in mind, the result is 504 legal parameter combinations.

We have two sets of experiments. In the first, we include the transitional data samples, resulting in 11 classifications. In the second, we omit the transitional data samples. Even though our goal involves being able to process these transitional samples, we perform the experiment without the transitional samples to have a closer comparison with how classification is traditionally performed on the DVSGesture dataset in the literature. This yields a total of 1008 combinations. Finally, as a control, we test a combination where the chip size is $1 \times 1$ with a stride of 1 and a threshold of 1. This effectively performs no downsampling of the observation space at all and serves as a comparison data point to determine whether or not these neuromorphic spatial reduction techniques benefit classifier performance or not. This brings our total number of combinations up to 1010. We use scikit-learn [17] to create and train the random forest models on all combinations of the neuromorphic observation space reduction techniques.

### 6 RESULTS

Figure 2 summarizes the results of the two experiments. It plots the F1-score for each random forest on the testing dataset. We chose F1-score as the evaluation metric because the sets of data have differing numbers of labels, and the transitional samples cause an imbalance in the label representation of the dataset.

Each parameter combination is numbered by how well it performs on the experiment without transition samples. The performance trend is similar for both experiments, and as expected, the overall performance on the experiment without transition samples is better than the experiment with transition samples. It is interesting that for the parameter combinations that perform poorly, they perform better on the dataset with transition labels. However, as the F1-Score rises above 0.5, the performance is markedly better without transition samples. We have no explanation for this phenomenon.
Figures 3 and 4 plot the performance of the classifiers with and without transitions, respectively, as a function of the number of features after downsampling and/or collapsing. In each figure, the best performance comes from reducing the input space of $128 \times 128$ to between 10 and 50 input features. This is an interesting result for two reasons. First, reducing the number of features, even though it loses resolution, can improve the performance of the classifier. Second, when the number of input features is reduced to the 10-50 range, the resulting size of the classifier, and therefore the SNN that implements it, is smaller, which is beneficial for SWaP.

Figure 5 shows the performance of classifying the data without transition samples, as a function of the number of neurons required when the random forest is converted to a spiking neural network [19]. This graph underscores the advantages of downsampling – the networks that perform best are in fact the smaller networks. This may be because of overfitting. We do not show the number of synapses because they scale linearly with the number of neurons (roughly 1.2 synapses per neuron). Moreover, we also omit the results of the data with transition samples. Although they require more neurons and synapses, their trends are the same.

Figure 6 shows the parameter settings of the 50 highest-scoring networks on the dataset without transition samples. Above each data point are shaded boxes indicating the parameter settings of the data point. For the boolean parameters (Stride=Half, 2nd Downsample, Collapse and Event), a white box indicates true and a dark box indicates false. In the other two parameters (Threshold and Chip Size), lighter boxes indicate higher values.

There are three trends we can glean from this data. First, the majority of these networks did not use a second round of downsampling. Second, using a stride equal to half of the chip size generates better random forests. Third, row/column collapse was advantageous significantly more than it was not. We can use these observations to narrow the scope of further parameter searches, or to allow us to broaden the scope with respect to other parameters.

It is evident that the neuromorphic spatial reduction techniques can bear a large impact on the performance of the random forest classifier. The wrong combination of parameters can yield f1-scores on the testing dataset that are lower than 30%; however, the best combinations of parameters can yield scores as high as 85% when “transition” samples are ignored and 72% if they are not. The highest performing classifiers share the following similarities: the feature space is reduced to less than fifty, the row/column collapse technique is used, only one downsampling or spiking threshold pooling layer is used, and the chip stride is typically half the length of the square chip size.

7 DISCUSSION

A point of confusion with this work is that our dataset is a timeseries dataset and is therefore not directly amenable to using a random forest classifier. We still fit a random forest model to the training data by “unrolling” each timeseries sample. If we suppose that we have downsampled to an observation space that is $5 \times 5$ and we know that segments_per_sample is 150, we can treat each feature at each timestep as a unique feature such that the classifier is working on a dense $5 \times 5 \times 150 = 3750$ feature observation.

At first glance, this seems to contradict the goal of being able to classify gestures in real time on neuromorphic hardware without additional preprocessing aside from the neuromorphic spatial reduction techniques. However, for the spiking neural network implementation, a dense observation need not be created. The camera’s event outputs need only be routed to the proper input neurons at the proper time.

Additionally, the best performing networks have about 20,000 total parameters (neurons and synapses) when the transitional data is ignored and about 30,000 total parameters when the transitional data is recalled. While these totals are an order of magnitude smaller than the network sizes reported in [8, 25], these totals do not include the additional hardware requirements of the neuromorphic
observation space reduction techniques that are employed. However, it is assumed that any spiking threshold pooling networks will be time multiplexed over the DVS camera’s observation space. This involves using only one or a handful of counting networks in a spiking threshold pooling layer and striding them over the input space over time. This is similar in methodology to what has been presented by Severa et al. [20] previously. When using time multiplexing, only a few hundred additional neurons and synapses are necessary (depending on the chip size). We argue that the additional hardware requirements of the spiking threshold pooling layer(s) and row/column collapse techniques are negligible and do not total more than a couple thousand neurons and synapses.

As one of the first attempts evaluating neuromorphic observation space reduction techniques on DVS camera data, the authors acknowledge the following shortcomings with this body of work. Three parameter combinations from each experiment did not finish running due to memory errors. Each parameter combination had access to 96 Gb of memory, but if the feature space was too large (as it was for the 1 × 1 downsampling with a stride of 1 control experiment), the model could not be created and stored in memory alongside the data. This is likely due to segments_per_sample being too large. For the control experiment, the random forest classifier would have had to fit data containing 128 × 128 × 150 = 2457600 features per time series data sample. While this is not ideal, it can still be stated that the neuromorphic spatial reduction techniques enable random forest classification of the DVSGesture dataset. Without it and with too large of an “unrolled” feature space for the classifier to use for training, random forest will not work on the dataset at all.

8 CONCLUSIONS AND FUTURE WORK
This work aimed to study the impact of neuromorphic spatial reduction techniques on the random forest classifier’s ability to classify our version of the DVS Gesture dataset. There are a few key takeaways from this work, but the most important one is that the neuromorphic spatial reduction techniques’ parameters greatly impact the classifier’s performance. In fact, without using the techniques, random forest cannot train on the data at all. Again, we specifically care about the random forest classifier because it can be represented as a spiking neural network on hardware and placed directly behind a camera for real-time classification. Furthermore, with this method of training, our resultant best performing networks are an order of magnitude smaller than other state of the art architectures. The tradeoff is that performance is roughly 10-15% worse; however, with a larger parameter space exploration, the authors believe this performance gap can be closed substantially. Another takeaway is that the row/column collapse technique yields the best performance and should likely be used as a final neuromorphic observation space reduction technique before the classifier.

Going forward, there are plenty of different aspects of this work that can be explored in more detail. A larger parameter grid search can be employed, or a different hyperparameter optimization technique like Bayesian optimization can be used instead to better explore the parameter space. Additionally, we can test different values for time_segmentation_length and segments_per_sample to examine how different levels of event sparsity affect the random forest classifier for the same parameter search space. This search might also lead to a combination where the 1 × 1 chip size with stride 1 experiment can be evaluated and directly compared against.

ACKNOWLEDGMENTS
This material is based in part on research sponsored by Air Force Research Laboratory under agreements FA8750-21-1-1018. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily repre- senting the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government.

REFERENCES
[17] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss,


