Asynchronous Convolutional Networks for Object Detection in Neuromorphic Cameras

Marco Cannici Marco Ciccone Andrea Romanoni Matteo Matteucci
Politecnico di Milano, Italy
{marco.cannici,marco.ciccone,andrea.romanoni,matteo.matteucci}@polimi.it

Abstract

Event-based cameras, also known as neuromorphic cameras, are bioinspired sensors able to perceive changes in the scene at high frequency with low power consumption. Becoming available only very recently, a limited amount of work addresses object detection on these devices. In this paper we propose two neural networks architectures for object detection: YOLE, which integrates the events into surfaces and uses a frame-based model to process them, and fcYOLE, an asynchronous event-based fully convolutional network which uses a novel and general formalization of the convolutional and max pooling layers to exploit the sparsity of camera events. We evaluate the algorithm with different extensions of publicly available datasets, and on a novel synthetic dataset.

1. Introduction

Fundamental techniques underlying Computer Vision are based on the ability to extract meaningful features. To this extent, Convolutional Neural Networks (CNNs) rapidly became the first choice in many computer vision applications such as image classification [18, 45, 13, 48], object detection [42, 41, 25], semantic scene labeling [49, 37, 26], and they have been recently extended also to non-euclidean domains such as manifolds and graphs [16, 31]. In most of the cases the input of these networks are images.

In the meanwhile, neuromorphic cameras [43, 36, 3] are becoming more and more widespread. These devices are bio-inspired vision sensors that attempt to emulate the functioning of biological retinas. As opposed to conventional cameras, which generate frames at a constant frame rate, these sensors output data only when a brightness change is detected in the field of view. Whenever this happens, an event $\mathbf{e} = \langle x, y, ts, p \rangle$ is generated indicating the position (x, y), the instant ts at which the change has been detected and its polarity $p \in \{1, -1\}$, *i.e.*, if the brightness change is positive or negative. The result is a sensor able to produce a stream of asynchronous events that sparsely encodes changes with microseconds resolution and with minimum requirements in terms of power consumption and bandwidth. The growth in popularity of these type of sensors, and their advantages in terms of temporal resolution and reduced data redundancy, have led to fully exploit the advantages of event-based vision for a variety of applications, *e.g.*, object tracking [39, 29, 11], visual odometry [32, 40], and optical flow estimation [2, 24, 47].

Spiking Neural Networks (SNNs) [27], a processing model aiming to improve the biological realism of artificial neural networks, are one of the most popular neural model able to directly handle events. Despite their advantages in terms of speed and power consumption, however, training deep SNNs models on complex tasks is usually very difficult. To overcome the lack of scalable training procedures, recent works have focused on converting pre-trained deep networks to SNNs, achieving promising results even on complex tasks [14, 5, 8].

An alternative solution to deal with event-based cameras is to make use of frame integration procedures and conventional frame-based networks [35] which can instead rely on optimized training procedures. Recently, other alternatives to SNNs making use of hierarchical time surfaces [20] and memory cells [46] have also been introduced. Another solution, proposed in [33], suggests instead the use of LSTM cells to accumulate events and perform classification. An extension of this work making use of attention mechanisms has also been proposed in [4].

Although event-cameras are becoming increasingly popular, only very few datasets for object detection in event streams are available, and a limited number of object detection algorithms has been proposed [23, 6, 38].

In this paper we introduce a novel hybrid approach to extract features for object detection problems using neuromorphic cameras. The proposed framework allows the design of object detection networks able to sparsely compute features while still preserving the advantages of conventional neural networks. More importantly, networks implemented using the proposed procedure are asynchronous, meaning that computation is only performed when a sequence of events arrive and only where previous results need to be recomputed.

In Section 3 the convolution and max-pooling operations are reformulated by adding an internal state, *i.e.*, a memory of the previous prediction, that allows us to sparsely recompute feature maps. An asynchronous fully-convolutional network for event-based object detection which exploits this formulation is finally described in Section 3.4.

2. Background

Leaky Surface. The basic component of the proposed architectures is a procedure able to accumulate events. Sparse events generated by the neuromorphic camera are integrated into a *leaky surface*, a structure that takes inspiration from the functioning of Spiking Neural Networks (SNNs) to maintain memory of past events. A similar mechanism has already been proposed in [7]. Every time an event with coordinates (x_e, y_e) and timestamp ts^t is received, the corresponding pixel of the surface is incremented of a fixed amount Δ_{incr}. At the same time, the whole surface is decremented by a quantity which depends on the time elapsed between the last received event and the previous one. The described procedure can be formalized by the following equations:

$$q^t_{x_s,y_s} = max(p^{t-1}_{x_s,y_s} - \lambda \cdot \Delta_{ts}, 0) \tag{1}$$

$$p^t_{x_s,y_s} = \begin{cases} q^t_{x_s,y_s} + \Delta_{incr} & if (x_s, y_s)^t = (x_e, y_e)^t \\ q^t_{x_s,y_s} & otherwise \end{cases}, \tag{2}$$

where $p^t_{x_s,y_s}$ is the value of the surface pixel in position (x_s, y_s) of the leaky surface and $\Delta_{ts} = ts^t - ts^{t-1}$. To improve readability in following equations, we name the quantity $(ts_t - ts_{t-1}) \cdot \lambda$ as Δ_{leak}. Notice that the effects of λ and Δ_{incr} are related: Δ_{incr} determines how much information is contained in each single event whereas λ defines the decay rate of activations. Given a certain choice of these parameters, similar results can be obtained by using, for instance, a higher increment Δ_{incr} and a higher temporal λ. For this reason, we fix $\Delta_{incr} = 1$ and we vary only λ based on the dataset to be processed. Pixel values are prevented from becoming negative by means of the max operation.

Other frame integration procedures, such as the one in [35], divide the time in predefined constant intervals. Frames are obtained by setting each pixel to a binary value (depending on the polarity) if at least an event has been received in each pixel within the integration interval. With this mechanism however, time resolution is lost and the same importance is given to each event, even if it represents noise. The adopted method, instead, performs continuous and incremental integration and is able to better handle noise.

Similar procedures capable of maintaining time resolution have also been proposed, such as those that make use of exponential decays [7, 19] to update surfaces, and those relying on histograms of events [28]. Recently, the concept of *time surface* has also been introduced in [20] where surfaces are obtained by associating each event with temporal features computed applying exponential kernels to the event neighborhood. Extensions of this procedure making use of memory cells [46] and event histograms [1] have also been proposed. Although these event representations better describe complex scene dynamics, we make use of a simpler formulation to derive a linear dependence between consecutive surfaces. This allows us to design the event-based framework discussed in Section 3 in which time decay is applied to every layer of the network.

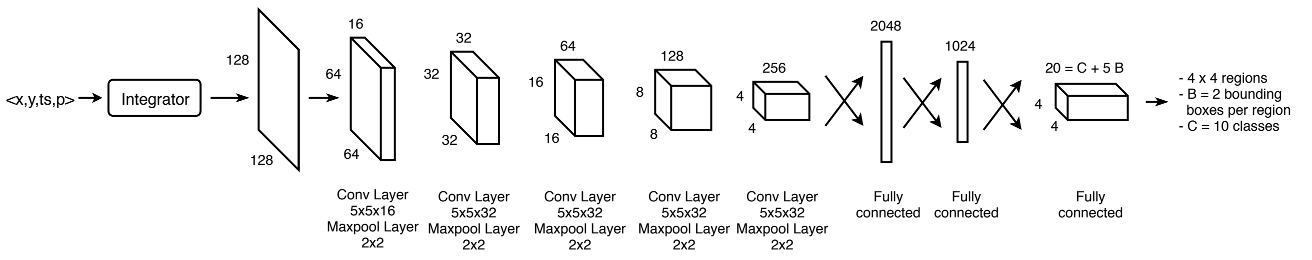
Event-based Object Detection. We identified YOLO [41] as a good candidate model to tackle the object detection problem in event-based scenarios: it is fully-differentiable and produces predictions with small input-output delays. By means of a standard CNN and with a single forward pass, YOLO is able to simultaneously predict not only the class, but also the position and dimension of every object in the scene. We used the YOLO loss and the previous leaky surface procedure to train a baseline model which we called YOLE: "You Only Look at Events". The architecture is depicted in Figure 1. We use this model as a reference to highlight the strengths and weaknesses of the framework described in Section 3, which is the main contribution of this work. YOLE processes 128×128 surfaces, it predicts $B = 2$ bounding boxes for each region and classifies objects into C different categories.

Note that in this context, we use the term YOLO to refer only to the training procedure proposed by [41] and not to the specific network architecture. We used indeed a simpler structure for our models as explained in Section 4. Nevertheless, YOLE, *i.e.*, YOLO + leaky surface, does not exploit the sparse nature of events; to address this issue, in the next section, we propose a fully event-based asynchronous framework for convolutional networks.

3. Event-based Fully Convolutional Networks

Conventional CNNs for video analysis treat every frame independently and recompute all the feature maps entirely, even if consecutive frames differ from each other only in small portions. Beside being a significant waste of power and computations, this approach does not match the nature of event-based cameras.

To exploit the event-based nature of neuromorphic vision, we propose a modification of the forward pass of fully convolutional architectures. In the following the convolution and pooling operations are reformulated to produce the final prediction by recomputing only the features corresponding to regions affected by the events. Feature maps



<x,y,ts,p>
Integrator
128
128
64
64
16
Conv Layer 5x5x16 Maxpool Layer 2x2
32
32
32
Conv Layer 5x5x32 Maxpool Layer 2x2
16
16
64
Conv Layer 5x5x32 Maxpool Layer 2x2
8
8
128
Conv Layer 5x5x32 Maxpool Layer 2x2
4
4
256
Conv Layer 5x5x32 Maxpool Layer 2x2
2048
Fully connected
1024
Fully connected
20 = C + 5 B
4
4
Fully connected
- 4 x 4 regions
- B = 2 bounding boxes per region
- C = 10 classes

Figure 1. The YOLE detection network based on YOLO used to detect MNIST-DVS digits. The input surfaces are divided into a grid of 4×4 regions which predict 2 bounding boxes each.

maintain their state over time and are updated only as a consequence of incoming events. At the same time, the leaking mechanism that allows past information to be forgotten, acts independently on each layer of the CNN. This enables features computed in the past to fade away as their visual information starts to disappear in the input surface. The result is an asynchronous CNN able to perform computation only when requested and at different rates. The network can indeed be used to produce an output only when new events arrive, dynamically adapting to the timings of the input, or to produce results at regular rates by using the leaking mechanism to update layers in absence of new events.

The proposed framework has been developed to extend the YOLE detection network presented in Section 2. Nevertheless, this method can be applied to any convolutional architecture to perform asynchronous computation. A CNN trained to process frames reconstructed from streams of events can indeed be easily converted into an event-based CNN without any modification on its layers composition, and by using the same weights learned while observing frames, maintaining its output unchanged.

3.1. Leaky Surface Layer

The procedure used to compute the leaky surface described in Section 2 is embedded into an actual layer of the proposed framework. Furthermore, to allow subsequent layers to locate changes inside the surface, the following information are also forwarded to the next layer: (i) the list of incoming events. (ii) Δ_{leak}, which is sent to all the subsequent layers to update feature maps not affected by the events. (iii) the list of surface pixels which have been reset to 0 by the max operator in Equation (1).

3.2. Event-based Convolutional Layer (e-conv)

The *event-based convolutional* (e-conv) layer we propose uses events to determine where the input feature map has changed with respect to the previous time step and, therefore, which parts of its *internal state*, *i.e.*, the feature map computed at the previous time step, must be recomputed and which parts can be reused. We use a procedure similar to the one described in the previous section to let features decay over time. However, due to the transformations applied by previous layers and the composition of their activation functions, Δ_{leak} may act differently in different parts of the feature map. For instance, the decrease of a pixel intensity value in the input surface may cause the value computed by a certain feature in a deeper layer to decrease, but it could also cause another feature of the same layer to increase. The update procedure, therefore, must also be able to accurately determine how a single bit of information is transformed by the network through all the previous layers, in any spatial location. We face this issue by storing an additional feature map, $F_{(n)}$, and by using a particular class of activation functions in the hidden layers of the network.

Let's consider the first layer of a CNN which processes surfaces obtained using the procedure described in the previous section and which computes the convolution of a set of filters W with bias b and activation function $g(\cdot)$. The computation performed on each receptive field is:

$$y^t_{ij_{(1)}} = g\left(\sum_h \sum_k x^t_{h+i,k+j} W_{hk_{(1)}} + b_{(1)}\right) = g(\tilde{y}^t_{ij_{(1)}}), \tag{3}$$

where h, k select a pixel $x^t_{h+i,k+j}$ in the receptive field of the output feature (i, j) and its corresponding value in the kernel W, whereas the subscript (1) indicates the hidden-layer of the network (in this case the first after the leaky surface layer).

When a new event arrives, the leaky surface layer decreases all the pixels by Δ_{leak}, *i.e.*, a pixel not directly affected by the event becomes: $x^{t+1}_{hk} = x^t_{hk} - \Delta^{t+1}_{leak}$, with $\Delta^{t+1}_{leak} > 0$. At time $t+1$ Equation (3) becomes:

$$\begin{aligned} y^{t+1}_{ij_{(1)}} &= g\left(\sum_h \sum_k x^{t+1}_{h+i,k+j} W_{hk_{(1)}} + b_{(1)}\right) \\ &= g\left(\sum_h \sum_k (x^t_{h+i,k+j} - \Delta^{t+1}_{leak}) W_{hk_{(1)}} + b_{(1)}\right) \\ &= g\left(\tilde{y}^t_{ij_{(1)}} - \Delta^{t+1}_{leak} \sum_h \sum_k W_{hk_{(1)}}\right). \end{aligned} \tag{4}$$

If $g(\cdot)$ is (i) a piecewise linear activation function $g(x) = \{\alpha_i \cdot x \quad \text{if } x \in D_i\}$, as ReLU or Leaky ReLU, and we assume that (ii) the updated value does not change which linear segment of the activation function the output falls onto and, in this first approximation, (iii) the leaky surface layer does not restrict pixels using $\max(\cdot, 0)$, Equation 4 can be rewritten as it follows:

$$y^{t+1}_{ij_{(1)}} = y^{t}_{ij_{(1)}} - \Delta^{t+1}_{leak} \alpha_{ij_{(1)}} \sum_h \sum_k W_{hk_{(1)}}, \tag{5}$$

where $\alpha_{ij_{(1)}}$ is the coefficient applied by the piecewise function $g(\cdot)$ which depends on the feature value at position (i, j). When the previous assumption is not satisfied, the feature is recomputed as its receptive field was affected by an event (*i.e.*, applying the filter W locally to x^{t+1}).

Consider now a second convolutional layer attached to the first one:

$$\begin{aligned}
y^{t+1}_{ij_{(2)}} &= g\left(\sum_{h,k} y^{t+1}_{i+h,j+k_{(1)}} W_{hk_{(2)}} + b_{(2)}\right) \\
&= g\left(\sum_{h,k}\left(y^{t}_{i+h,j+k_{(1)}} - \Delta^{t+1}_{leak} \alpha_{i+h,j+k_{(1)}} \sum_{h',k'} W_{h'k'_{(1)}}\right) W_{hk_{(2)}} + b_{(2)}\right) \\
&= y^{t}_{ij_{(2)}} - \Delta^{t+1}_{leak} \alpha_{ij_{(2)}} \sum_{h,k}\left(\alpha_{i+h,j+k_{(1)}} \sum_{h',k'} W_{h'k'_{(1)}}\right) W_{hk_{(2)}} \\
&= y^{t}_{ij_{(2)}} - \Delta^{t+1}_{leak} \alpha_{ij_{(2)}} \sum_{h,k} F^{t+1}_{h+i,k+j_{(1)}} W_{hk_{(2)}} = y^{t}_{ij_{(2)}} - \Delta^{t+1}_{leak} F^{t+1}_{ij_{(2)}} .
\end{aligned} \tag{6}$$

The previous equation can be extended by induction as it follows:

$$\begin{gathered}
y^{t+1}_{ij_{(n)}} = y^{t}_{ij_{(n)}} - \Delta^{t+1}_{leak} F^{t+1}_{ij_{(n)}}, \\
\text{with } F^{t+1}_{ij_{(n)}} = \alpha_{ij_{(n)}} \sum_h \sum_k F^{t+1}_{i+h,j+k_{(n-1)}} W_{hk_{(n)}} \quad \text{if } n > 1 ,
\end{gathered} \tag{7}$$

where $F_{ij_{(n)}}$ expresses how visual inputs are transformed by the network in every receptive field (i, j), *i.e.*, the composition of the previous layers activation functions.

Given this formulation, the $\max$ operator applied by the leaky surface layer can be interpreted as a ReLU, and Equation (5) becomes:

$$y^{t+1}_{ij_{(1)}} = y^{t}_{ij_{(1)}} - \Delta^{t+1}_{leak} \alpha_{ij_{(1)}} \sum_h \sum_k F^{t+1}_{i+h,j+k_{(0)}} W_{hk_{(1)}}, \tag{8}$$

where the value $F_{i+h,j+k_{(0)}}$ is 0 if the pixel $x_{i+h,j+k} \leq 0$ and 1 otherwise.

Notice that $F_{ij_{(n)}}$ needs to be updated only when the corresponding feature changes enough to make the activation function use a different coefficient α, *e.g.*, from 0 to 1 in case of ReLU. In this case $F_{(n)}$ is updated locally in correspondence of the change by using the update matrix of the previous layer and by applying Equation 7 only for the features whose activation function has changed. Events are used to communicate the change to subsequent layers so that their update matrix can also be updated accordingly.

The internal state of the e-conv layer, therefore, comprises the feature maps $y^{t-1}_{(n)}$ and the update values $F^{t-1}_{(n)}$ computed at the previous time step. The initial values of the internal state are computed making full inference on a blank surface; this is the only time the network needs to be executed entirely. As a new sequence of events arrives the following operations are performed (see Figure 3(a)):

i. Update $F^{t-1}_{(n)}$ locally on the coordinates specified by the list of incoming events (Eq. (7)). Note that we do not distinguish between actual events and those generated by the use of a different slope in the linear activation function.

ii. Update the feature map $y_{(n)}$ with Eq. (7) in the locations which are not affected by any event and generate an output event where the activation function coefficient has changed.

iii. Recompute $y_{(n)}$ by applying W locally in correspondence of the incoming events and output which receptive field has been affected.

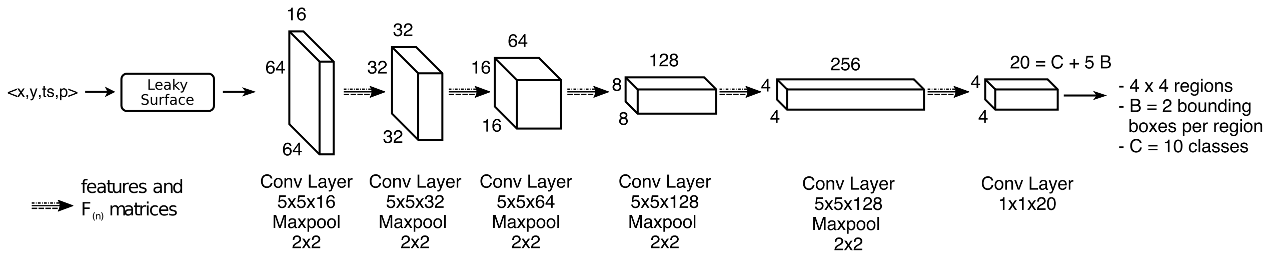
iv. Forward the feature map and the events generated in the current step to the next layer.

3.3. Event-based Max Pooling Layer (e-max-pool)

The location of the maximum value in each receptive field of a max-pooling layer is likely to remain the same over time. An event-based pooling layer, hence, can exploit this property to avoid recomputing every time the position of maximum values.

The internal state of an event-based max-pooling (e-max-pool) layer can be described by a *positional matrix* $I^t_{(n)}$, which has the shape of the output feature map produced by the layer, and which stores, for every receptive field, the position of its maximum value. Every time a sequence of events arrives, the internal state $I^t_{(n)}$ is sparsely updated by recomputing the position of the maximum values in every receptive field affected by an incoming event. The internal state is then used both to build the output feature map and to produce the *update matrix* $F^t_{(n)}$ by fetching the previous layer on the locations provided by the indices $I^t_{ij_{(n)}}$. For each e-max-pool layer, the indices of the receptive fields where the maximum value changes are communicated to the subsequent layers so that the internal states can be updated accordingly. This mechanism is depicted in Figure 3(b).

Notice that the leaking mechanism acts differently in distinct regions of the input space. Features inside the same receptive field can indeed decrease over time with different speeds as their update values $F^t_{ij_{(n)}}$ could be different. Therefore, even if no event has been detected inside a region, the position of its maximum value might change.



<x,y,ts,p>
Leaky Surface
16
64
64
32
32
32
16
16
64
128
8
8
256
4
4
20 = C + 5 B
4
4
- 4 x 4 regions
- B = 2 bounding boxes per region
- C = 10 classes
features and F (n) matrices
Conv Layer 5x5x16 Maxpool 2x2
Conv Layer 5x5x32 Maxpool 2x2
Conv Layer 5x5x64 Maxpool 2x2
Conv Layer 5x5x128 Maxpool 2x2
Conv Layer 5x5x128 Maxpool 2x2
Conv Layer 1x1x20

Figure 2. fcYOLE: a fully-convolutional detection network based on YOLE. The last layer is used to map the feature vectors into a set of 20 values which define the parameters of the predicted bounding boxes.

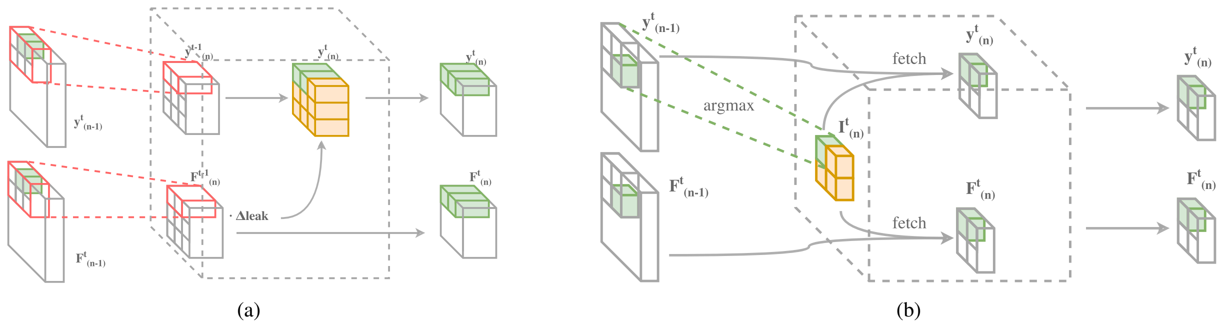


Figure 3. The structure of the e-conv **(a)** and e-max-pooling layers **(b)**. The internal states and the update matrices are recomputed locally only where events are received (green cells) whereas the remaining regions (depicted in yellow) are obtained reusing the previous state.

However, if an input feature M has the minimum update rate $F_{M_{(n-1)}}$ among features in its receptive field R and it also corresponds to the maximum value in R, the corresponding output feature will decrease slower than all the others in R and its value will remain the maximum. In this case, its index $I^t_{(n)_R}$ does not need to be recomputed until a new event arrives in R. We check if the maximum has to be recomputed for each receptive field affected by incoming events and also in all positions where the previous condition does not hold.

3.4. Event FCN for Object Detection (fcYOLE)

To fully exploit the event-based layers presented so far, the YOLE model described in Section 2 needs to be converted into a fully convolutional object detection network replacing all its layers with their event-based versions (see Figure 3). Moreover, fully-connected layers are replaced with 1×1 e-conv layers which map features extracted by the previous layers into a precise set of values defining the bounding boxes parameters predicted by the network. Training was first performed on a network composed of standard layers; the learned weights were then used with e-conv and e-max-pool layers during inference.

This architecture divides the 128×128 field of view into a grid of 4×4 regions that predicts 2 bounding boxes each and classify the detected objects into C different classes. The last 1×1 e-conv layer is used to decrease the dimensionality of the feature vectors and to map them into the right set of parameters, regardless of their position in the field of view.

Moreover, this architecture can be used to process surfaces of different sizes without the need to re-train or redesign it. The subnetworks processing 32×32 regions, in fact, being defined by the same set of parameters, can be stacked together to process even larger surfaces.

4. Experiments

4.1. Datasets

Only few event-based object recognition datasets are publicly available in the literature. The most popular ones are: N-MNIST [34], MNIST-DVS [44], CIFAR10-DVS [22], N-Caltech101 [34] and POKER-DVS [44]. These datasets are obtained from the original MNIST [21], CIFAR-10 [17] and Caltech101 [10] datasets by recording the original images with an event camera while moving the camera itself or the images of the datasets. We performed experiments on N-Caltech101 and on modified versions of N-MNIST and MNIST-DVS for object detection, *i.e.*, *Shifted N-MNIST* and *Shifted MNIST-DVS*, and on an extended version of POKER-DVS, namely *OD-Poker-DVS*. Moreover we also perform experiments on a synthetic dataset, named *Blackboard MNIST*, showing digits written on a blackboard. A detailed description of these datasets is provided in the supplementary materials.

Shifted N-MNIST The N-MNIST [34] dataset is a conversion of the popular MNIST [21] image dataset for computer vision. We enhanced this collection by building a slightly more complex set of recordings. Each sample is indeed composed of two N-MNIST samples placed in two random non-overlapping locations of a bigger 124×124 field of view. Each digit was also preprocessed by extracting its bounding box which was then moved, along with the events, in its new position of the bigger field of view. The final dataset is composed of $60,000$ training and $10,000$ testing samples.

Shifted MNIST-DVS We used a similar procedure to obtain Shifted MNIST-DVS recordings. We first extracted bounding boxes with the same procedure used in Shifted N-MNIST and then placed them in a 128×128 field of view. We mixed MNIST-DVS *scale4*, *scale8* and *scale16* samples within the same recording obtaining a dataset composed of $30,000$ samples.

OD-Poker-DVS The Poker-DVS dataset is a small collection of neuromorphic samples showing poker card pips obtained by extracting 31×31 symbols from three deck recordings. We used the tracking algorithm provided with the dataset to track pips and enhance the original uncut deck recordings with their bounding boxes. We finally divided these recordings into a set of shorter examples obtaining a collection composed of 218 training and 74 testing samples.

Blackboard MNIST We used the DAVIS simulator released by [32] to build our own set of synthetic recordings. The resulting dataset consists of a number of samples showing digits written on a blackboard in random positions and with different scales. We preprocessed a subset of images from the original MNIST dataset by removing their background and by making them look as if they were written with a chalk. Sets of digits were then placed on the image of a blackboard and the simulator was finally run to obtain event-based recordings and the bounding boxes of every digit visible within the camera field of view. The resulting dataset is the union of three simulations featuring increasing level of variability in terms of camera trajectories and digit dimensions. The overall dataset is composed of 2750 training and 250 testing samples.

N-Caltech101 The N-Caltech101 [34] collection is the only publicly available event-based dataset providing bounding boxes annotations. We split the dataset into 80% training and 20% testing samples using a stratified split. Since no ground truth bounding boxes are available for the *background* class, we decided not to use this additional category in our experiments.

4.2. Experiments Setup

Event-based datasets, especially those based on MNIST, are generally simpler than the image-based ones used to train the original YOLO architecture. Therefore, we designed the MNIST object detection networks taking inspiration from the simpler LeNet [21] model composed of 6 conv-pool layers for feature extraction. Both YOLE and fcYOLE share the same structure up to the last regression/classification layers.

For what concerns the N-Caltech101 dataset, we used a slightly different architecture inspired by the structure of the VGG16 model [45]. The network is composed by only one layer for each group of convolutional layers, as we noticed that a simpler architecture achieved better results. Moreover, the dimensions of the last fully-connected layers have been adjusted such that the surface is divided into a grid of 5×7 regions predicting $B = 2$ bounding boxes each. As in the original YOLO architecture we used Leaky ReLU for the activation functions of hidden layers and a linear activation for the last one.

In all the experiments the first 4 convolutional layers have been initialized with kernels obtained from a recognition network pretrained to classify target objects, while the remaining layers using the procedure proposed in [12]. All networks were trained optimizing the multi-objective loss proposed by [41] using Adam [15], learning rate 10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The batch size was chosen depending on the dataset: 10 for Shifted N-MNIST, 40 for Shifted MNIST-DVS and N-Caltech101, 25 for Blackboard MNIST and 35 for Poker-DVS with the aim of filling the memory of the GPU optimally. Early-stopping was applied to prevent overfitting using validation sets with the same size of the test set.

4.3. Results and Discussion

Detection performance of YOLE. The YOLE network achieves good detection results both in terms of mean average precision (mAP) [9] and accuracy, which in this case is computed by matching every ground truth bounding box with the predicted one having the highest intersection over union (IOU), in most of the datasets. The results we obtained are summarized in Table 3.

We used the Shifted N-MNIST dataset also to analyze how detection performance changes when the network is used to process scenes composed of a variable number of objects, as reported in Table 4. We denote as *v1* the results obtained using scenes composed of a single digit and with *v2* those obtained with scenes containing two digits in random locations of the field of view. We further tested the robustness of the proposed model by adding some challenging noise. We added non-target objects (*v2fr*) in the form of five 8×8 fragments, taken from random N-MNIST digits using a procedure similar to the one used to build the *Cluttered Translated MNIST* dataset [30], and 200 additional random events per frame (*v2fr+ns*).

In case of multiple objects the algorithm is still able to detect all of them, while, as expected, performance drops both in terms of accuracy and mean average precision when dealing with noisy data. Neverthelesss, we achieved very good detection performance on the Shifted MNIST-DVS,

Table 1. YOLE Top-20 average precisions on N-Caltech101. Full table provided in the supplemental material.

	Motorbikes	airplanes	Faces easy	metronome	laptop	dollar bill	umbrella	watch	minaret	grand piano	menorah	inline skate	saxophone	stapler	windsor chair	rooster	yin yang	Leopards	trilobite	garfield
AP	97.8	95.8	94.7	88.3	88.1	86.5	85.9	84.2	81.3	81.3	80.7	75.1	68.4	68.1	65.2	64.5	63.3	62.9	62.5	62.3
N_{train}	480	480	261	20	49	32	45	145	46	61	53	19	24	27	34	31	36	120	52	22

Table 2. fcYOLE Top-20 average precisions on N-Caltech101. Full table provided in the supplemental material.

	Motorbikes	airplanes	Faces easy	watch	dollar bill	car side	grand piano	menorah	metronome	umbrella	yin yang	saxophone	minaret	soccer ball	Leopards	dragonfly	stop sign	windsor chair	accordion	buddha
AP	97.5	96.8	92.2	75.7	74.4	70.3	69.5	67.7	63.4	61.0	60.4	59.7	59.5	57.3	57.2	55.6	55.1	52.3	48.3	46.5
N_{train}	480	480	261	145	32	75	61	53	20	45	36	24	46	40	120	42	40	34	33	51

Table 3. Performance comparison between YOLE and fcYOLE.

S-MNIST-DVS				Blackboard MNIST			
fcYOLE		YOLE		fcYOLE		YOLE	
acc	mAP	acc	mAP	acc	mAP	acc	mAP
94.0	87.4	96.1	92.0	88.5	84.7	90.4	87.4

OD-Poker-DVS				N-Caltech101			
fcYOLE		YOLE		fcYOLE		YOLE	
acc	mAP	acc	mAP	acc	mAP	acc	mAP
79.10	78.69	87.3	82.2	57.1	26.9	64.9	39.8

Table 4. YOLE performance on S-N-MNIST variants.

	S-N-MNIST				
	v1	v2	v2*	v2fr	v2fr+ns
accuracy	94.9	91.7	94.7	88.6	85.5
mAP	91.3	87.9	90.5	81.5	77.4

Blackboard MNIST and Poker-DVS datasets which represent a more realistic scenario in terms of noise. All of these experiments were performed using the set of hyperparameters suggested by the original work from [41]. However, a different choice of these parameters, namely $\lambda_{coord} = 25.0$ and $\lambda_{noobj} = 0.25$, worked better for us increasing both the accuracy and mean average precision scores (*v2**).

The dataset in which the proposed model did not achieve noticeable results is N-Caltech101. This is mainly explained by the increased difficulty of the task and by the fact that the number of samples in each class is not evenly balanced. The network, indeed, usually achieves good results when the number of training samples is high such as with *Airplanes*, *Motorbikes* and *Faces_easy*, and in cases in which samples are very similar, *e.g.*, *inline_skate* (see Table 1 and supplementary material). As the number of training samples decreases and the sample variability within the class increases, however, the performance of the model becomes worse, behavior which explains the poor aggregate scores we report in Table 3.

Detection performance of fcYOLE. With this fully-convolutional variant of the network we registered a slight decrease in performance w.r.t. the results we obtained using YOLE, as reported in Table 3 and Table 2. This gap in performance is mainly due to the fact that each region in fcYOLE generates its predictions by only looking at the visual information contained in its portion of the field of view. Indeed, if an object is only partially contained inside a region the network has to guess the object dimensions and class by only looking at a restricted region of the surface. It should be stressed, however, that the difference in performance between the two architectures does not come from the use of the proposed event layers, whose output are the same as the conventional ones, but rather from the reduced expressive power caused by the absence of fully-connected layers in fcYOLE. Indeed, not removing them would have allowed us to obtain the same performance of YOLE, but with the drawback of being able to exploit event-based layers only up to the first FC-layer, which has not been formalized yet in an event-based form. Removing the last fully-connected layers allowed us to design a detection network made of only event-based layers and which uses also a significantly lower number of parameters. In the supplementary materials we provide a video showing a comparison between YOLE and fcYOLE predictions.

To identify the advantages and weaknesses of the proposed event-based framework in terms of inference time we compared our detection networks on two datasets, Shifted N-MNIST and Blackboard MNIST. We group events into batches of 10ms and average timings on 1000 runs. In the first dataset the event-based approach achieved a 2x speedup (22.6ms per batch), whereas in the second one it performed slightly slower (43.2ms per batch) w.r.t. a network making use of conventional layers (34.6ms per batch). The second benchmark is indeed challenging for our framework since changes are not localized in restricted regions. Our current implementation is not optimized to handle noisy scenes efficiently. Indeed, additional experiments showed that asynchronous CNNs are able to provide a faster prediction only up to a 80% of event sparsity (where with sparsity we mean the percentage of changed pixels in the reconstructed image). Further investigations are out of the scope of this paper and will be addressed in future works.

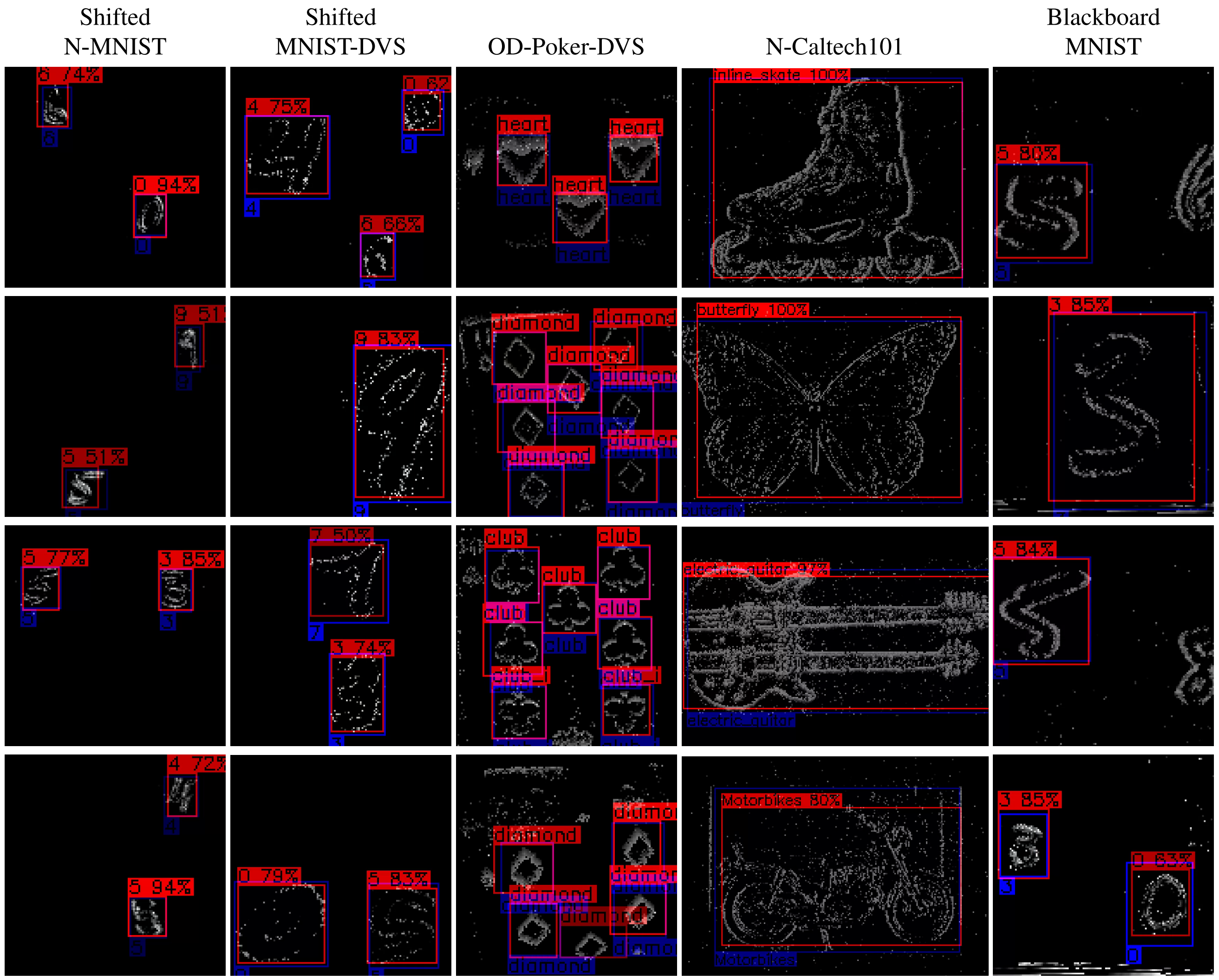


Figure 4. Examples of YOLE predictions.

5. Conclusions

We proposed two different methods, based on the YOLO architecture, to accomplish object detection in event-based cameras. The first one, namely YOLE, integrates events into a unique leaky surface. Conversely, fcYOLE relies on a more general extension of the convolutional and max pooling layers to directly deal with events and exploit their sparsity by preventing the whole network to be reprocessed. The obtained asynchronous detector dynamically adapts to the timing of the events stream by producing results only as a consequence of incoming events and by maintaining its internal state, without performing any additional computation, when no events arrive. This novel event-based framework can be used in every fully-convolutional architecture to make it usable with event-cameras, even conventional CNN for classification, although in this paper it has been applied to object detection networks.

We analyzed the timing performance of this formalization obtaining promising results. We are planning to extend our framework to automatically detect at runtime when the use of event-based layers speeds up computation (*i.e.*, changes affect few regions of the surface) or a complete recomputation of the feature maps is more beneficial in order to exploit the benefits of both approaches. Nevertheless, we believe that a ad-hoc hardware implementation, would allow to better exploit the advantages of the proposed method enabling a fair timing comparison with SNNs, which are usually implemented in hardware.

Acknowledgements We would like to thank Prophesee for helpful discussions on YOLE. The research leading to these results has received funding from project TEINVEIN: TEcnologie INnovative per i VEicoli Intelligenti, CUP (Codice Unico Progetto - Unique Project Code): E96D17000110009 - Call “Accordi per la Ricerca e lInnovazione”, cofunded by POR FESR 2014-2020 (Programma Operativo Regionale, Fondo Europeo di Sviluppo Regionale Regional Operational Programme, European Regional Development Fund).

References

[1] L. Y. Alex Zihao Zhu. Ev-flownet: Self-supervised optical flow estimation for event-based cameras. *Robotics: Science and Systems*, Jan 2018. 2

[2] P. Bardow, A. J. Davison, and S. Leutenegger. Simultaneous optical flow and intensity estimation from an event camera. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 884–892, 2016. 1

[3] R. Berner, C. Brandli, M. Yang, S.-C. Liu, and T. Delbruck. A 240×180 10mw 12us latency sparse-output vision sensor for mobile applications. pages C186–C187, 01 2013. 1

[4] M. Cannici, M. Ciccone, A. Romanoni, and M. Matteucci. Attention mechanisms for object recognition with event-based cameras. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1127–1136, Jan 2019. 1

[5] Y. Cao, Y. Chen, and D. Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, 2015. 1

[6] N. F. Y. Chen. Pseudo-labels for Supervised Learning on Dynamic Vision Sensor Data, Applied to Object Detection under Ego-motion. *arXiv*, Sep 2017. 1

[7] G. K. Cohen. *Event-Based Feature Detection, Recognition and Classification*. PhD thesis, Université Pierre et Marie Curie - Paris VI, Sep 2016. 2

[8] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. Liu, and M. Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2015. 1

[9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *Int. J. Comput. Vision*, 88(2):303–338, Jun 2010. 6

[10] L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(4):594–611, Apr 2006. 5

[11] D. Gehrig, H. Rebecq, G. Gallego, and D. Scaramuzza. Asynchronous, photometric feature tracking using events and frames. In *Eur. Conf. Comput. Vis.(ECCV)*, 2018. 1

[12] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *PMLR*, pages 249–256, Mar 2010. 6

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[14] S. Kim, S. Park, B. Na, and S. Yoon. Spiking-yolo: Spiking neural network for real-time object detection. *arXiv preprint arXiv:1903.06530*, 2019. 1

[15] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv*, Dec 2014. 6

[16] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017. 1

[17] A. Krizhevsky. Learning multiple layers of features from tiny images. 04 2009. 5

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 1

[19] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman. HOTS: A Hierarchy of Event-Based Time-Surfaces for Pattern Recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(7), Jul 2016. 2

[20] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman. Hots: a hierarchy of event-based time-surfaces for pattern recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(7):1346–1359, 2017. 1, 2

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, Nov 1998. 5, 6

[22] H. Li, H. Liu, X. Ji, G. Li, and L. Shi. CIFAR10-DVS: An Event-Stream Dataset for Object Classification. *Front. Neurosci.*, 11:309, May 2017. 5

[23] J. Li, F. Shi, W. Liu, D. Zou, Q. Wang, H. Lee, P.-K. Park, and H. E. Ryu. Adaptive temporal pooling for object detection using dynamic vision sensor. *British Machine Vision Conference (BMVC)*, 2017. 1

[24] M. Liu and T. Delbruck. Adaptive time-slice block-matching optical flow algorithm for dynamic vision sensors. Technical report, 2018. 1

[25] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 1

[26] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 1

[27] W. Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997. 1

[28] A. I. Maqueda, A. Loquercio, G. Gallego, N. Garca, and D. Scaramuzza. Event-based vision meets deep learning on steering prediction for self-driving cars. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2

[29] A. Mitrokhin, C. Fermuller, C. Parameshwara, and Y. Aloimonos. Event-based moving object detection and tracking. *arXiv preprint arXiv:1803.04523*, 2018. 1

[30] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014. 6

[31] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017. 1

[32] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza. The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM. *arXiv*, Oct 2016. 1, 6

[33] D. Neil, M. Pfeiffer, and S.-C. Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*, pages 3882–3890, 2016. 1

[34] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor. Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades. *Front. Neurosci.*, 9, Nov 2015. 5, 6

[35] J. A. Pérez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen, and B. Linares-Barranco. Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward ConvNets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(11):2706–2719, Nov 2013. 1, 2

[36] C. Posch, D. Matolin, and R. Wohlgenannt. A QVGA 143 dB Dynamic Range Frame-Free PWM Image Sensor With Lossless Pixel-Level Video Compression and Time-Domain CDS. *IEEE J. Solid-State Circuits*, 46(1):259–275, Jan 2011. 1

[37] A. Raj, D. Maturana, and S. Scherer. Multi-scale convolutional architecture for semantic segmentation. page 14, 01 2015. 1

[38] B. Ramesh, H. Yang, G. Orchard, N. A. L. Thi, and C. Xiang. DART: Distribution Aware Retinal Transform for Event-based Cameras. *arXiv*, Oct 2017. 1

[39] B. Ramesh, S. Zhang, Z. W. Lee, Z. Gao, G. Orchard, and C. Xiang. Long-term object tracking with a moving event camera. 2018. 1

[40] H. Rebecq, T. Horstschaefer, G. Gallego, and D. Scaramuzza. Evo: A geometric approach to event-based 6-dof parallel tracking and mapping in real time. *IEEE Robotics and Automation Letters*, 2(2):593–600, 2017. 1

[41] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 1, 2, 6, 7

[42] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 1

[43] T. Serrano-Gotarredona and B. Linares-Barranco. A 128 $\times$ 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*, 48(3):827–838, Mar 2013. 1

[44] T. Serrano-Gotarredona and B. Linares-Barranco. Poker-DVS and MNIST-DVS. Their History, How They Were Made, and Other Details. *Front. Neurosci.*, 9, Dec 2015. 5

[45] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1, 6

[46] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman. Hats: Histograms of averaged time surfaces for robust event-based object classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1731–1740, 2018. 1, 2

[47] T. Stoffregen and L. Kleeman. Simultaneous optical flow and segmentation (sofas) using dynamic vision sensor. *arXiv preprint arXiv:1805.12326*, 2018. 1

[48] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017. 1

[49] F. Yu and V. Koltun. Multi-Scale Context Aggregation by Dilated Convolutions. In *International Conference on Learning Representations (ICLR)*, 2016. 1

Asynchronous Convolutional Networks for Object Detection in Neuromorphic Cameras Supplementary material

Marco Cannici Marco Ciccone Andrea Romanoni Matteo Matteucci
Politecnico di Milano, Italy
{marco.cannici,marco.ciccone,andrea.romanoni,matteo.matteucci}@polimi.it

In this document we describe our novel event-based datasets adopted in the paper "Asynchronous Convolutional Network for Object Detection in Neuromorphic Cameras".

1. Event-based object detection datasets

Due to the lack of object detection datasets with event cameras, we extended the publicly available N-MNIST, MNIST-DVS, Poker-DVS and we propose a novel dataset based on MNIST, i.e., Blackboard MNIST. They will be soon released, however, in Figure 1 we reported some example from the four datasets.

1.1. Shifted N-MNIST

The original N-MNIST [5] extends the well-known MNIST [3]: it provides an event-based representation of both the full training set ($60,000$ samples) and the full testing set ($10,000$ samples) to evaluate object classification algorithms. The dataset has been recorded by means of event camera in front of an LCD screen and moved to detect static MNIST digits displayed on the monitor. For further details we refer the reader to [5].

Starting from the N-MNIST dataset, we built a more complex set of recordings that we used to train the object detection network to detect multiple objects in the same scene. We created two versions of the dataset, Shifted N-MNIST v1 and Shifted N-MNIST v2, that contains respectively one or two non overlapping 34×34 N-MNIST digits per sample randomly positioned on a bigger surface. We used different surface dimensions in our tests which vary from double the original size, 68×68, up to 124×124. The dimension and structure of the resulting dataset is the same of the original N-MNIST collection.

To extend the dataset for object detection evaluation, the bounding boxes ground truth are required. To estimate them we first integrate events into a single frame as described in Section 2 of the original paper. We remove the noise by considering only non-zero pixels having at least other ρ non-zero pixels around them within a circle of radius R. All the other pixels are considered noise. Then, with a custom version of the DBSCAN [2] density-based clustering algorithm we group pixels into a single cluster. A threshold min_{area} is used to filter out small bounding boxes extracted in correspondence of low events activities. This condition usually happens during the transition from a saccade to the next one as the camera remains still for a small fraction of time and no events are generated. We used $\rho = 3$, $R = 2$ and $min_{area} = 10$. The coordinates of these bounding boxes are then shifted based on the final position the digit has in the bigger field of view.

For each N-MNIST sample, another digit was randomly selected in the same portion of the dataset (training, testing or validation) to form a new example. The final dataset contains $60,000$ training samples and $10,000$ testing samples, as for the original N-MNSIT dataset. In Figure 2 we illustrate one example for v1 and the three variants of v2 we adopted (and described) in the paper.

1.2. Shifted MNIST-DVS

The MNIST-DVS dataset [6] is another collection of event-based recordings that extends MNIST [3]. It consists of $30,000$ samples recorded by displaying digits on an screen in front of a event camera, but differently from N-MNIST, they move digits on the screen instead of the sensors, and they use the digits at three different scales, i.e., *scale4*, *scale8* and *scale16*. The resulting dataset is composed of $30,000$ event-based recordings showing each one of the selected $10,000$ MNIST digits on thee different dimensions. Examples of these recordings are shown in Figure 3.

We used MNIST-DVS recordings to build a detection dataset by means of a procedure similar to the one we used to create the Shifted N-MNIST dataset. However in this case we mix together digits of multiple scales. All the MNIST-DVS samples, despite of the actual dimensions of the digits being recorded, are contained within a fixed 128×128 field of view. Digits are placed centered inside

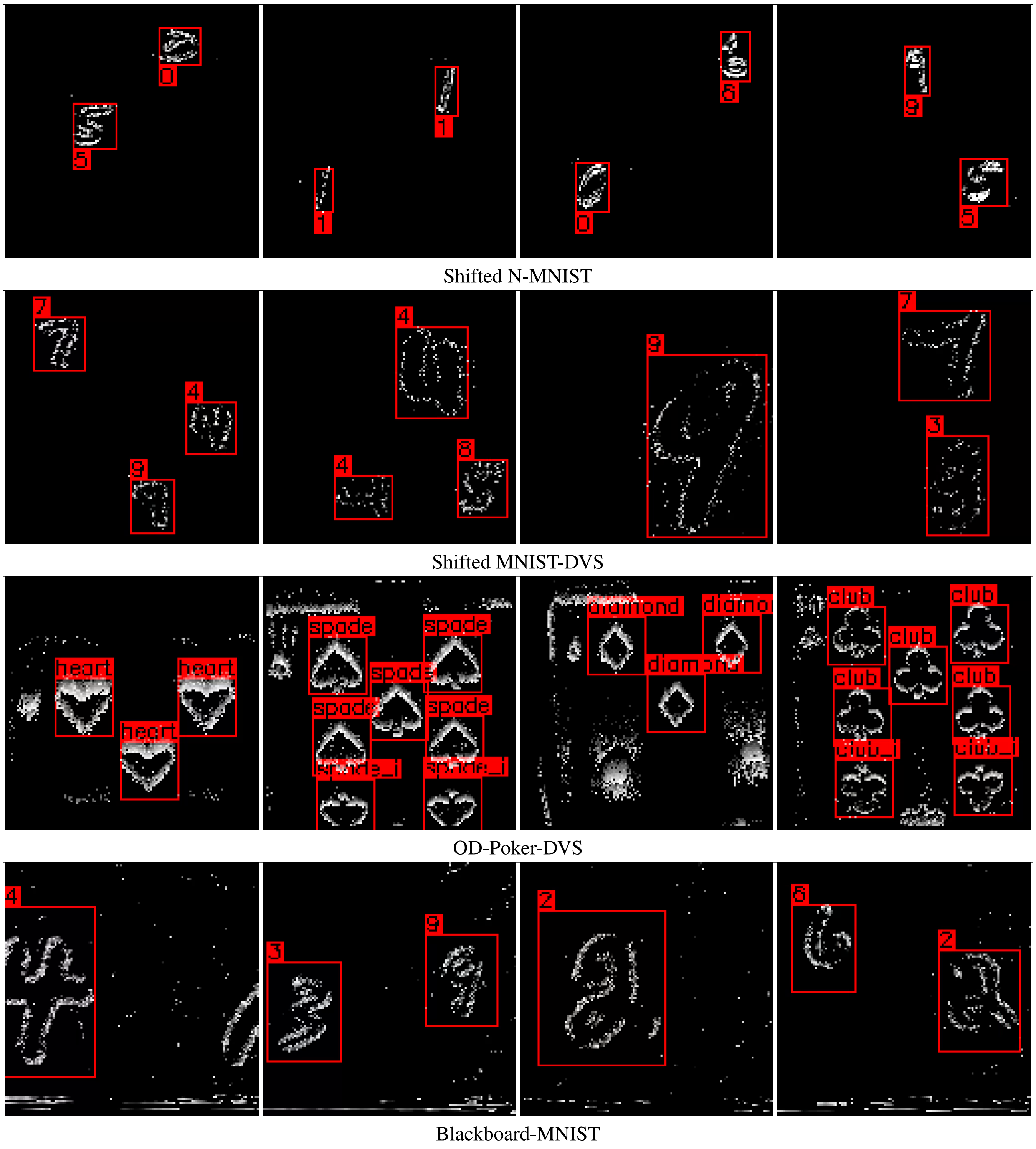


Figure 1: Examples of samples from the proposed datasets.

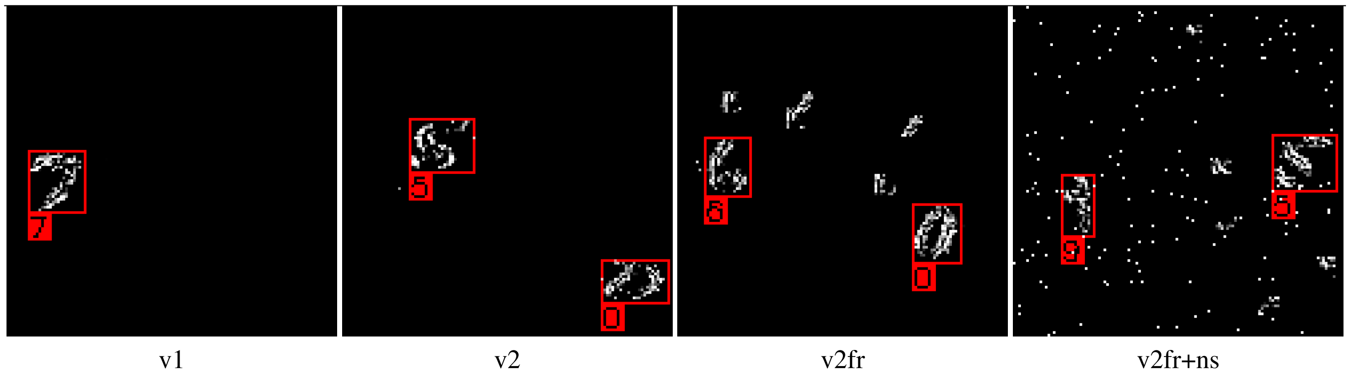


Figure 2: Different versions of Shifted N-MNIST.

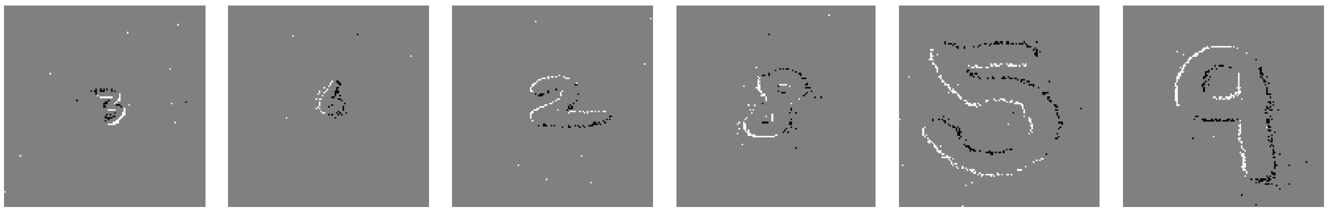


Figure 3: Examples of the three different scales of MNIST-DVS digits. Two samples at scale *scale4*, two at *scale8* and two at *scale16*.

the scene and occupy a limited portion of the frame, especially those belonging to the smallest and middle scales. In order to place multiple examples on the same scene we first cropped the three scales of samples into smaller recordings occupying 35×35, 65×65 and 105×105 spatial regions respectively. The bounding boxes annotations and the final examples were obtained by means of the same procedure we used to construct the Shifted N-MNIST dataset. These recordings were built by mixing digits of different dimensions in the same sample. Based on the original samples dimensions, we decided to use the following four configurations (which specify the number of samples of each category used to build a single Shifted MNIST-DVS example): (i) three *scale4* digits, (ii) two *scale8* digits, (iii) two *scale4* digits mixed with one *scale8* digit (iv) one *scale16* digit placed in random locations of the field of view. The overall dataset is composed of $30,000$ samples containing these four possible configurations.

1.3. OD-Poker-DVS

The original Poker-DVS [6] have been proposed to test object recognition algorithms; it is a small collection of neuromorphic recordings obtained by quickly browsing custom made poker card decks in front of a DVS camera for 2-4 seconds. The dataset is composed of 131 samples containing centered pips of the four possible categories (spades, hearts, diamonds or clubs) extracted from three decks recordings. Single pips were extracted by means of an event-based tracking algorithms which was used to follow symbols inside the scene and to extract 31×31 pixels examples.

With OD-Poker-DVS we extend its scope to test also object detection. To do so we used the event-based tracking algorithm provided with the original dataset to follow the movement of the 31×31 samples in the uncut recordings and extract their bounding boxes. The final dataset was obtained using a procedure similar to the one used in [7]. Indeed, we divided the sections of the three original decks recordings containing visible digits into a set of shorter examples, each of which about 1.5ms long. Examples were split in order to ensure approximately the same number of objects (i.e., ground truth bounding boxes) in each example. The final detection dataset is composed of 292 small examples which we divided into 218 training and 74 testing samples.

Even if composed of a limited amount of samples, this dataset represents an interesting real-world application that highlights the potential of event-based vision sensors. The nature of the data acquisition is indeed particularly well suited to neuromorphic cameras due to their very high temporal resolution. Symbols are clearly visible inside the recordings even if they move at very high speed. Each pip, indeed, takes from 10 to 30 ms to cross the screen but it can be easily recognized within the first 1-2 ms.

1.4. Blackboard MNIST

The two dataset based on MNIST presented in Section 1.1 and 1.2 have the drawback of recording digits at predefined sizes. Therefore, in Blackboard MNIST we propose a more challenging scenario that consists of a number of samples showing digits (from the original MNIST dataset) written on a blackboard in random positions and with different scales.

We used the DAVIS simulator released by [4] to build our own set of synthetic recordings. Given a three-dimensional virtual scene and the trajectory of a moving camera within it, the simulator is able to generate a stream of events describing the visual information captured by the virtual camera. The system uses Blender [1], an open-source 3D modeling tool, to generate thousands of rendered frames along a predefined camera trajectory which are then used to reconstruct the corresponding neuromorphic recording. The intensity value of each single pixel inside the sequence of rendered frames, captured at a constant frame-rate, is tracked. As Figure 4a shows, an event is generated whenever the log-intensity of a pixel crosses an intensity threshold, as in a real event-based camera. A piecewise linear time interpolation mechanism is used to determine brightness changes in the time between frames in order to simulate the microseconds timestamp resolution of a real sensor. We extended the simulator to output bounding boxes annotations associated to every visible digit.

We used Blender APIs to place MNIST digits in random locations of a blackboard and to record their position with respect to the camera point of view. Original MNIST images depict black handwritten digits on a white background. To mimic the chalk on the blackboard, we removed the background, we turned digits in white and we roughen their contours to make them look like if their were written with a chalk. An example is shown in Figure 4b.

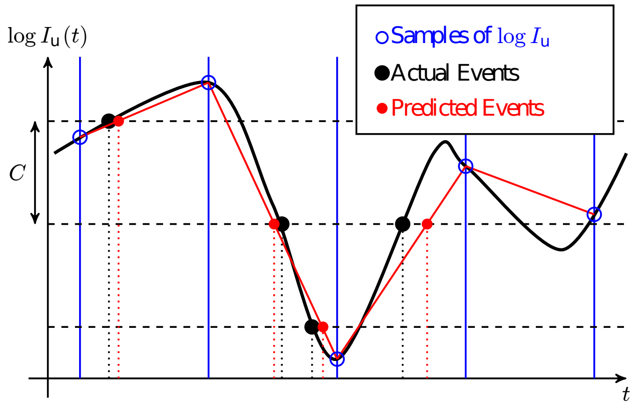
The scene contains the image of a blackboard on a vertical plane and a virtual camera with 128×128 resolution that moves horizontally on a predefined trajectory parallel to the blackboard plane (see Figure 5). The camera points a hidden object that moves on the blackboard surface, synchronized with the camera, following a given trajectory. To introduce variability in the camera movement, and to allow all the digits outline to be seen (and possibly detected), we used different trajectories that vary from a straight path to a smooth or triangular undulating path that makes the camera tilt along the transverse axis while moving (Figure 5b).

Before starting the simulation, we randomly select a number of preprocessed MNIST digits and place them in a random portion of the blackboard. The camera moves so that all the digits will be framed during the camera movement. The simulation is then finally started on this modified scene to generate neuromorphic recordings. Every time a frame is rendered during the simulation, the bounding boxes of all the visible digits inside the frame are also extracted. This operation is performed by computing the camera space coordinates (or normalized device coordinates) of the top-left and bottom-right vertex of all the images inside the field of view. Since images are slightly larger than the actual digits they contain, we cropped every bounding box to better enclose each digit and also to compensate the small offset in the pixels position introduced by the camera motion and by the linear interpolation mechanism. In addition, bounding boxes corresponding to objects which are only partially visible are also filtered out. In order to build the final detection dataset, this generation process is executed multiple times, each time with different digits.

We built three sub-collections of recordings with increasing level of complexity which we merged together to obtain our final dataset: *Blackboard MNIST EASY*, *Blackboard MNIST MEDIUM*, *Blackboard MNIST HARD*. In Blackboard MNIST EASY, we used digits of only one dimension (roughly corresponding to the middle scale of MNIST-DVS samples) and a single type of camera trajectory which moves the camera from right to left with the focus object moving in a straight line. In addition, only three objects were placed on the blackboard using only a fixed portion of its surface. We collected a total of $1,200$ samples ($1,000$ training, 100 testing, 100 validation).

Blackboard MNIST MEDIUM features more variability in the number and dimensions of the digits and in the types of camera movements. Moreover, the portion of the blackboard on which digits were added varies and may cover any region of the blackboard, even those near its edges. The camera motions were also extended to the set of all possible trajectories that combine either left-to-right or right-to-left movements with variable paths of the focus object. We used three types of trajectories for this object: a straight line, a triangular path or a smooth curved trajectory, all parallel to the camera trajectory and placed around the position of the digits on the blackboard. One of these path was selected randomly for every generated sample. Triangular and curved trajectories were introduced as we noticed that sudden movements of the camera produce burst of events that we wanted our detection system to be able to handle. The number and dimensions of the digits were chosen following three possible configurations, similarly to the Shift MNIST-DVS dataset: either six small digits (with sizes comparable to *scale4* MNIST-DVS digits), three intermediate-size digits (comparable to the MNIST-DVS *scale8*) or two big digits (comparable to the biggest scale of the MNIST-DVS dataset, *scale16*). A set of $1,200$ recordings was generated using the same splits of the first variant and with equal amount of samples in each one of the three configurations.

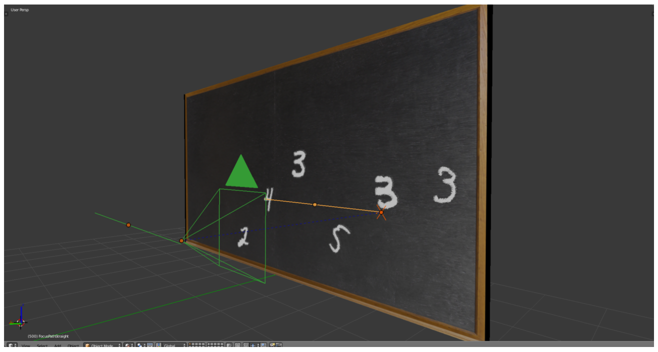
Finally, Blackboard MNIST HARD contains digits



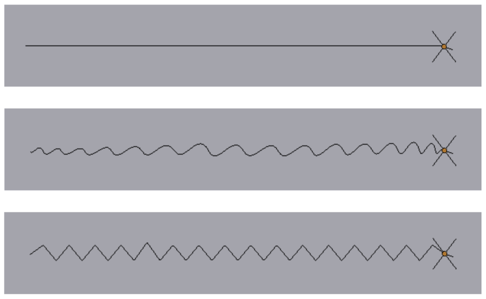
(a)

(b)

Figure 4: *(a)* The image shows in black the intensity, expresses as $\log I_u(t)$, of a single pixel $\mathbf{u} = (x, y)$. This curve is sampled at a constant rate when frames are generated by Blender, shown in figure as vertical blue lines. The sampled values thus obtained (blue circles) are used to approximate the pixel intensity by means of a simple piecewise linear time interpolation (red line). Whenever this curve crosses one of the threshold values (horizontal dashed lines) a new event is generated with the corresponding predicted timestamp. (Figure from [4]) **(b)** A preprocessed MNIST digit on top of the blackboard's background.



(a)



(b)

Figure 5: **(a)** The 3D scene used to generate the Blackboard MNIST dataset. The camera moves in front of the blackboard along a straight trajectory while following a *focus object* that moves on the blackboard's surface, synchronized with the camera. The camera and its trajectory are depicted in green, the focus object is represented as a red cross and, finally, its trajectory is depicted as a yellow line. **(b)** The three types of focus trajectories.

recorded by using the second and third configuration of objects we described previously. However, in this case each image was resized to a variable size spanning from the original configuration size down to the previous scale. A total of 600 new samples (500 training, 50 testing, 50 validation) were generated, 300 of them containing three digits each and the remaining 300 consisting of two digits with variable size.

The three collections can be used individually or jointly; the whole Blackboard MNIST dataset contains $3,000$ samples in total (2500 training, 250 testing, 250 validation). Examples of different objects configurations are shown in Figure 6. Samples were saved by means of the AEDAT v3.1 file format for event-based recordings.

2. Results

Table 1 provides a comparison between the average precision of YOLE and fcYOLE on N-Caltech101 classes. We also provide a qualitative comparison between the two models in the video attachment.

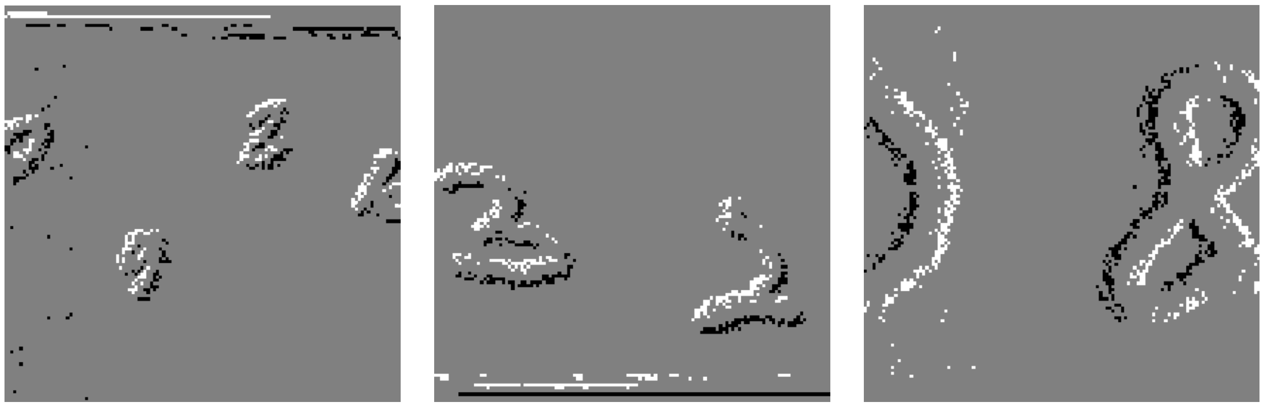


Figure 6: Examples of the three types of objects configurations used to generate the second collection of the Blackboard MNIST dataset.

Table 1: YOLE and fcYOLE average precisions on N-Caltech101

	Motorbikes	airplanes	Faces_easy	watch	Leopards	chair	bonsai	car_side	ketch	flamingo	ant	chandelier	crocodile	grand_piano	brain	hawksbill	butterfly	helicopter	menorah	starfish
AP fcYOLE	97.5	96.8	92.2	75.7	57.2	7.5	30.2	70.3	42.3	2.3	2.4	34.8	0.0	69.5	35.3	19.6	33.5	8.6	67.7	23.2
AP YOLE	97.8	95.8	94.7	84.2	62.9	17.3	59.3	61.7	52.9	10.0	25.8	55.7	1.6	81.3	53.3	29.1	46.3	14.9	80.7	32.7
N_{train}	480	480	261	145	120	109	78	75	70	68	66	65	61	61	60	60	55	54	53	52
	scorpion	kangaroo	trilobite	sunflower	buddha	ewer	revolver	laptop	llama	ibis	minaret	umbrella	crab	electric_guitar	cougar_face	dragonfly	crayfish	dalmatian	ferry	euphonium
AP fcYOLE	2.5	3.4	41.2	29.1	46.5	35.3	20.3	40.0	1.4	1.5	59.5	61.0	5.0	23.2	21.8	55.6	7.3	24.9	29.7	43.5
AP YOLE	6.9	5.0	62.5	43.3	57.2	51.3	57.4	88.1	10.2	6.5	81.3	85.9	19.5	29.7	39.8	59.9	9.5	33.0	34.0	53.6
N_{train}	52	52	52	51	51	51	50	49	48	48	46	45	45	45	43	42	42	41	41	40
	lotus	stop_sign	joshua_tree	soccer_ball	elephant	schooner	dolphin	lamp	stegosaurus	rhino	wheelchair	cellphone	yin_yang	cup	sea_horse	pyramid	windsor_chair	hedgehog	bass	nautilus
AP fcYOLE	18.1	55.1	30.2	57.3	11.4	37.3	6.5	17.7	34.5	5.8	25.8	46.5	60.4	5.6	1.9	41.1	52.3	13.3	4.2	13.0
AP YOLE	27.6	61.7	29.8	51.5	6.0	56.8	11.5	45.3	44.6	11.3	25.3	54.6	63.3	17.5	8.8	48.6	65.2	9.8	4.0	50.4
N_{train}	40	40	40	40	40	39	39	37	37	37	37	37	36	35	35	35	34	34	34	33
	pizza	emu	accordion	dollar_bill	tick	crocodile_head	gramophone	rooster	camera	pagoda	cougar_body	barrel	ceiling_fan	beaver	cannon	mandolin	flamingo_head	brontosaurus	stapler	pigeon
AP fcYOLE	17.4	5.6	48.3	74.4	28.2	9.8	30.6	26.6	15.1	34.0	0.0	30.7	40.8	0.5	0.0	6.6	2.7	0.2	46.0	6.2
AP YOLE	54.5	5.0	52.7	86.5	55.2	10.0	48.4	64.5	32.7	33.3	12.2	41.2	50.1	0.0	0.3	43.4	9.3	25.8	68.1	43.1
N_{train}	33	33	33	32	31	31	31	31	30	29	29	29	29	28	27	27	27	27	27	27
	headphone	anchor	scissors	wrench	okapi	lobster	panda	saxophone	mayfly	water_lilly	garfield	wild_cat	gerenuk	platypus	binocular	octopus	strawberry	snoopy	metronome	inline_skate
AP fcYOLE	10.7	14.6	28.2	14.7	10.0	0.0	4.7	59.7	0.5	3.1	23.4	0.0	4.8	13.1	0.4	14.0	0.5	8.3	63.4	37.2
AP YOLE	21.1	17.3	47.5	29.7	44.6	0.0	12.2	68.4	0.7	14.7	62.3	0.0	7.2	34.7	11.8	13.8	29.4	53.1	88.3	75.1
N_{train}	26	26	25	25	25	25	24	24	24	23	22	22	22	22	21	21	21	21	20	19

References

[1] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2017. 4

[2] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages

226–231. AAAI Press, 1996. 1

[3] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, Nov 1998. 1

[4] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza. The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM. *arXiv*, Oct 2016. 4, 5

[5] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor. Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades. *Front. Neurosci.*, 9, Nov 2015. 1

[6] T. Serrano-Gotarredona and B. Linares-Barranco. Poker-DVS and MNIST-DVS. Their History, How They Were Made, and Other Details. *Front. Neurosci.*, 9, Dec 2015. 1, 3

[7] E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco. An Event-Driven Classifier for Spiking Neural Networks Fed with Synthetic or Dynamic Vision Sensor Data. *Front. Neurosci.*, 11, Jun 2017. 3