A Differentiable Recurrent Surface for Asynchronous Event-Based Data

Marco Cannici, Marco Ciccone, Andrea Romanoni *, and Matteo Matteucci

Politecnico di Milano, Italy
{marco.cannici,marco.ciccone,andrea.romanoni,matteo.matteucci}@polimi.it

Abstract. Dynamic Vision Sensors (DVSs) asynchronously stream events in correspondence of pixels subject to brightness changes. Differently from classic vision devices, they produce a sparse representation of the scene. Therefore, to apply standard computer vision algorithms, events need to be integrated into a frame or event-surface. This is usually attained through hand-crafted grids that reconstruct the frame using ad-hoc heuristics. In this paper, we propose Matrix-LSTM, a grid of Long Short-Term Memory (LSTM) cells that efficiently process events and learn end-to-end task-dependent event-surfaces. Compared to existing reconstruction approaches, our learned event-surface shows good flexibility and expressiveness on optical flow estimation on the MVSEC benchmark and it improves the state-of-the-art of event-based object classification on the N-Cars dataset.

Keywords: Event-Based Vision, Representation Learning, LSTM, Classification, Optical Flow

1 Introduction

Event-based cameras, such as dynamic vision sensors (DVSs) [17,29,2,25], are bio-inspired devices that attempt to emulate the efficient data-driven communication mechanisms of the brain. Unlike conventional frame-based active pixel sensors (APS), which capture the scene at a predefined and constant frame-rate, these devices are composed of independent pixels that output sequences of asynchronous events, efficiently encoding pixel-level brightness changes caused by moving objects. This results in a sensor having a very high dynamic range (> 120 dB) and high temporal resolution (in the order of microseconds), matched with low power consumption and minimal delay. All these characteristics are key features in challenging scenarios involving fast movements (e.g., drones or moving cars), and abrupt brightness changes (e.g., when exiting a dark tunnel in a car). However, novel methods and hardware architectures need to be specifically designed to exploit these advantages and leverage their potential in complex tasks. Event-cameras only provide a timed sequence of changes that is not directly compatible with computer vision systems which typically work on frames.

* Work done prior to Amazon involvement of the author and does not reflect views of the Amazon company.

Driven by the great success of frame-based deep learning architectures, that learn representations directly from standard APS signals, research in event-based processing is now focusing on how to effectively aggregate event information in grid-based representations which can be directly used, for instance, by convolutional deep learning models. Nevertheless, finding the best mechanism to extract information from event streams is not trivial.Multiple solutions have indeed emerged during the past few years, mostly employing hand-crafted mechanisms to accumulate events. Examples of such representations are mechanisms relying on exponential [6,15,31] and linear [6,4] decays, "event-surfaces" storing the timestamp of the last received event in each pixel and extensions of such mechanism making use of memory cells [31] and voxel-grids [26,36].

Only very recently deep learning techniques have been applied to learn such surfaces in a data-driven manner [10]. In this paper, we focus on this recent trend in event-based processing, and propose a mechanism to efficiently apply a Long Short-Term Memory (LSTM) network [12] as a convolutional filter over the 2D stream of events in order to accumulate pixel information through time and build 2D event representations. The reconstruction mechanism is end-to-end differentiable, meaning that it can be jointly trained with state-of-the-art frame-based architectures to learn event-surfaces specifically tailored for the task at hand. Most importantly, the mechanism specifically focuses on preserving sparsity during computation, enabling the reconstruction process to only focus on pixels receiving events and without requiring events to be densified in a dense tensor during the intermediate feature extraction steps, process that is otherwise necessary when applying standard computer vision approaches, such as ConvLSTM [30], in most of the cases.

Substituting hand-crafted event-surfaces with our trainable layer in state-of-the-art architectures improves their performance substantially without requiring particular effort in hyper-parameter tuning, enabling researchers to exploit event information effectively. The contributions of the paper are summarized as follows:

- We propose Matrix-LSTM, a task-independent mechanism to extract grid-like event representations from asynchronous streams of events. The framework is end-to-end differentiable, it can be used as input of any existing frame-based state-of-the-art architecture and jointly trained to extract the best representation from the events.
- Replacing input representations with a Matrix-LSTM layer in existing architectures, we show that it improves the state-of-the-art on event-based object classification on N-CARS [31] by 3.3% and performs better than hand-crafted features on N-Caltech101 [23]. Finally, it improves optical flow estimation on the MVSEC benchmark [37] up to 30.76% over hand-crafted features [37] and up to 23.07% over end-to-end differentiable ones [10].
- We developed custom CUDA kernels, both in PyTorch [32] and TensorFlow [1], to efficiently aggregate events by position and perform a convolution-like operation on the stream of events using an LSTM as a convolutional filter [1].

[1] Code available at `https://marcocannici.github.io/matrixlstm`

2 Related Work

Event cameras provide outstanding advantages over ordinary devices in terms of time resolution and dynamic range. However, their potentialities are still unlocked, mainly due to the difficulty of building good representations from a sparser, asynchronous and much more rough source of information compared to frame-based data. In this section, we give a brief overview of related works, focusing on representations for event-based data and highlighting the differences and similarities with our work. We refer the reader to [8] for a thorough overview.

Hand-crafted representations. Several hand-crafted event representations have been proposed over the years, ranging from biologically inspired, such as those used in Spiking Neural Networks [19], to more structured ones. Recently, the concept of *time-surface* was introduced [15,20], in which 2D surfaces are obtained by keeping track of the timestamp of the last event occurred in each location and by associating each event with features computed applying exponential kernels on the surface. An extension of these methods, called HATS [31], employs memory cells that retain temporal information from past events. Instead of building the surface using just the last event, too sensitive to noise, HATS uses a fixed-length memory. Histograms are then extracted from the surface and a SVM classifier is finally used for prediction. The use of a memory to compute the event-surface closely relates HATS with the solution presented in this paper. Crucially, the accumulation procedure employed in HATS is hand-crafted, while our work is end-to-end trainable thanks to a grid of LSTM cells [12], which enable to learn a better accumulation strategy directly from data.

In [36], the authors propose the EV-FlowNet network for optical flow estimation together with a new time-surface variant. Events of different polarities are kept separate to build a four-channel grid containing the number of events occurred in each location besides temporal information. A similar representation has also been used in [34]. To improve the temporal resolution of such representations, [38] suggests to discretize time into consecutive bins and accumulate events into a voxel-grid through a linearly weighted accumulation similar to bilinear interpolation. A similar time discretization has also been used in Events-to-Video [26], where the event representation is used within a recurrent-convolutional architecture to produce realistic video reconstructions of event sequences. Despite being slower, the quality of reconstructed frames closely resembles actual gray-scale frames, allowing the method to take full advantage from transferring feature representations trained on natural images.

End-to-end representations. Most closely related to the current work, [10] learns a dense representation end-to-end directly from raw events. A multi-layer perceptron (MLP) is used to implement a trilinear filter that produces a voxel-grid of temporal features. The event time information of each event is encoded using the MLP network and the value obtained from events occurring in the same spatial location are summed up together to build the final feature. A look-up table is then used, after training, to speed-up the procedure. Events

are processed independently as elements of a set, disregarding their sequentiality and preventing the network to modulate the information based on previous events. Our method, instead, by leveraging the memory mechanism of LSTM cells, can integrate information conditioned on the current state and can decide how much each event is relevant to perform the task, and how much information to retain from past events. A recent trend in event-based processing is studying mechanisms that do not require to construct intermediate explicit dense representations to perform the task at hand [3,28,33]. Among these, [22] uses a variant of the LSTM network, called PhasedLSTM, to learn the precise timings of events. While it integrates the events sequentially as in our work, PhasedLSTM employs a single cell on the entire stream of events and can be used only on very simple tasks [5]. The model, indeed, does not maintain the input spatial structure and condenses the 2D stream of events into a single feature vector, preventing the network to be used as input to standard CNNs. Finally, although it has never been adopted with event-based cameras, we also mention here the ConvLSTM [30] network, a convolutional variant of the LSTM that has previously been applied on several end-to-end prediction tasks. Despite its similarity with our method, since both implement the notion of convolution to LSTM cells, ConvLSTM is not straightforward to apply to sparse event-based streams and requires the input to be densified into frames before processing. This involves building very sparse frames of simultaneous events, mostly filled with padding, or dense frames containing uncorrelated events. Our formulation, instead, preserves sparsity during computation and does not require events to be densified, even when large receptive fields are considered.

3 Method

Event-based cameras are vision sensors composed of pixels able to work independently. Each pixel has its own exposure time and it is free to fire independently by producing an event as soon as it detects a significant change in brightness. Unlike conventional devices, no rolling shutter is used, instead, an asynchronous stream of events is generated describing what has changed in the scene. Each event e_i is a tuple $e_i = (x_i, y_i, t_i, p_i)$ specifying the time t_i, the location $(x, y)_i$ (within a $H \times W$ space) and the polarity $p_i \in \{-1, 1\}$ of the change (brightness increase or decrease). Therefore, given a time interval τ (i.e., the sample length), the set of events produced by the camera can be described as a sequence $\mathcal{E} = \{(x_i, y_i, t_i, p_i) \mid t_i \in \tau\}$, ordered by the event timestamp. In principle, multiple events could be generated at the same timestamp. However, the grid representation of the events at a fixed timestamp t is likely to be very sparse, hence, an integrating procedure is necessary to reconstruct a dense representation $\mathcal{S}_\mathcal{E}$ before being processed by conventional frame-based algorithms.

Note that, in this work, we do not aim to reconstruct a frame that resembles the actual scene, such as a grey-scale or RGB image [26,27], but instead to extract task-aware features regardless of their appearance. In the following, *"surface"*, *"reconstruction"* and *"representation"* are used with this meaning.

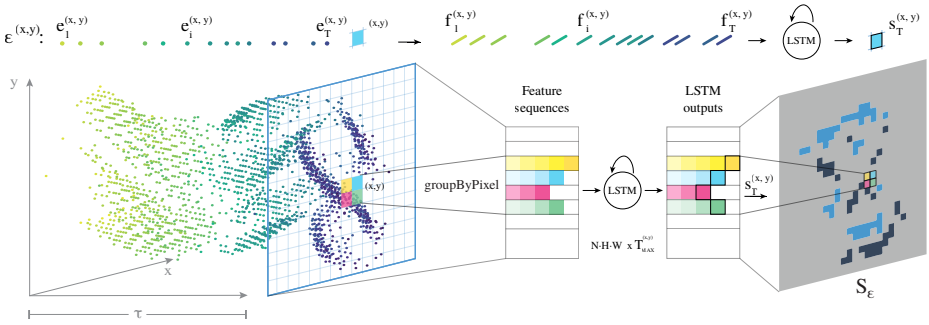


Fig. 1. Overview of Matrix-LSTM (figure adapted from [22]). Events in each pixel are first associated to a set of features $f_i^{(x,y)}$, and then processed by the LSTM. The last output, $s_T^{(x,y)}$, is finally used to construct $\mathcal{S}_\mathcal{E}$. *GroupByPixel* is shown here on a single sample ($N = 1$) highlighting a 2×2 pixel region. Colors refer to pixel locations while intensity indicates time. For clarity, the features dimension is not shown in the figure

3.1 Matrix-LSTM

Analogously to [10], our goal is to learn end-to-end a fully parametric mapping $\mathcal{M} : \mathcal{E} \to \mathcal{S}_\mathcal{E} \in \mathbb{R}^{H \times W \times C}$, between the event sequence and the corresponding dense representation, providing the best features for the task to optimize.

In this work, we propose to implement $\mathcal{M}$ as an $H \times W$ *matrix of LSTM cells* [12] (see Figure 1). Let's define the ordered sequence of events $\mathcal{E}^{(x,y)}$ produced by the pixel (x, y) during interval τ as $\mathcal{E}^{(x,y)} = \{(x_i, y_i, t_i, p_i) \mid t_i \in \tau, x_i = x, y_i = y\} \subset \mathcal{E}$, and its length as $T^{(x,y)} = |\mathcal{E}^{(x,y)}|$, which may potentially be different for each location (x, y). A set of features $f_i^{(x,y)} \in \mathbb{R}^F$ is first computed for each event occurring at location (x, y), typically the polarity and one or multiple temporal features (see Section 4). At each location (x, y), an $LSTM^{(x,y)}$ cell then processes these features asynchronously, keeping track of the current integration state and condensing all events into a single output vector $s^{(x,y)} \in \mathbb{R}^C$. In particular, at each time t, the $LSTM^{(x,y)}$ cell produces an intermediate representation $s_t^{(x,y)}$. Once all the events are processed, the last output of the LSTM cell compresses the dynamics of the entire sequence $\mathcal{E}^{(x,y)}$ into a fixed-length vector $s_T^{(x,y)}$ that can be used as pixel feature (here we dropped the superscript $^{(x,y)}$ from T for readability). The final surface $\mathcal{S}_\mathcal{E}$ is finally built by collecting all LSTMs final outputs $s_T^{(x,y)}$ into a dense tensor of shape $H \times W \times C$. A fixed all-zeros output is used where the set of events $\mathcal{E}^{(x,y)}$ is empty.

Temporal bins. Taking inspiration from previous methods [10,26,38] that discretize time into temporal bins, we also propose a variant of Matrix-LSTM that operates on successive time windows. Given a fixed number of bins B, the original event sequence is split into B consecutive windows $\mathcal{E}_{\tau_1}, \mathcal{E}_{\tau_2}, ..., \mathcal{E}_{\tau_B}$. Each sequence is processed independently, i.e., the output of each LSTM at the end of each interval is used to construct a surface $\mathcal{S}_{\mathcal{E}_b}$ and the LSTMs state is re-initialized

before the next sub-sequence starts. This gives rise of B different reconstructions $\mathcal{S}_{\mathcal{E}_b}$ that are concatenated to form the final surface $\mathcal{S}_{\mathcal{E}} \in \mathbb{R}^{H \times W \times B \cdot C}$. In this formulation, the LSTM input features $f_i^{(x,y)}$ usually contain both global temporal features (i.e., w.r.t. the original uncut sequence) and relative features (i.e., the event position in the sub-sequence). Although LSTMs should be able to retain memory over very long periods, we found that discretizing time into intervals helps , especially in tasks requiring precise time information such as optical flow estimation (see Section 4.2). A self-attention module [13] is then optionally applied on the reconstructed surface to correlate intervals (see Section 4.1).

Parameters sharing. Inspired by the convolution operation defined on images, we designed Matrix-LSTM to enjoy translation invariance. This is implemented by sharing the parameters across all the LSTM cells, as in a convolutional kernel. Sharing parameters not only drastically reduces the number of parameters in the network, but it also allows us to transfer a learned transformation to higher or lower resolutions as in fully-convolutional networks [18].

We highlight that such an interpretation of the Matrix-LSTM functioning also fits the framework proposed in [10], in which popular event densification mechanisms are rephrased as kernel convolutions on the *event field*, i.e., a discretized four-dimensional manifold spanning x and y, and the time and polarity dimensions. We finally report that this formulation is equivalent to a 1×1 ConvLSTM [30] applied on a dense tensor where events are stacked in pixel locations by arrival order. However, as reported in Section 4.1, this formulation has better space and time performance on sparse event sequences. Moreover, in the next section, an extension to larger receptive fields with better accuracy performance on asynchronous event data compared to ConvLSTM, is also proposed.

Receptive field size. As in a conventional convolution operation, Matrix-LSTM can be convolved on the input space using different strides and kernel dimensions. In particular, given a receptive field of size $K_H \times K_W$, each LSTM cell processes a local neighborhood of asynchronous events $\mathcal{E}^{(x,y)} = \{(x_i, y_i, t_i, p_i) \mid t_i \in \tau, |x - x_i| < K_W - 1, |y - y_i| < K_H - 1\}$. Events features are computed as in the original formulation, however, an additional coordinate feature (p_x, p_y) is also added specifying the relative position of each event within the receptive field. Coordinate features are range-normalized in such a way that an event occurring in the top-left pixel of the receptive field has feature $(0, 0)$, whereas one occurring in the bottom-right position has features $(1, 1)$. Events belonging to multiple receptive fields (e.g., when the LSTM is convolved with a stride 1×1 and receptive field greater then 1×1) are processed multiple times, independently.

Implementation. The convolution-like operation described in the previous section can be implemented efficiently by means of two carefully designed event grouping operations. Rather than replicating the LSTM unit multiple times on each spatial location, a single recurrent unit is applied over different $\mathcal{E}^{(x,y)}$ sequences in parallel. This requires a reshape operation, i.e., *groupByPixel*, that splits events based on their pixel location maintaining the events relative ordering

within each sub-sequence. A similar procedure, i.e., *groupByTime*, is employed to efficiently split events into consecutive temporal windows without making use of expensive masking operations. An example of the *groupByPixel* operation is provided in Figure 1 while implementation details of both operations, implemented as custom CUDA kernels, are provided in the supplementary materials. We finally highlight that these operations are not specific to Matrix-LSTM, since grouping events by pixel index is a common operation in event-based processing, and could indeed benefit other implementations making use of GPUs.

4 Evaluation

We test the proposed mechanism on two different tasks: object classification (see Section 4.1) and optical flow estimation (see Section 4.2), where the network is required to extract effective temporal features. We evaluated the goodness of Matrix-LSTM features indirectly: a state-of-the-art architecture is taken as a reference and the proposed method is evaluated in terms of the gain in performance obtained by replacing the network representation with a Matrix-LSTM.

4.1 Object classification

We evaluated the model on the classification task using two publicly available event-based collections, namely the N-Cars [31] and the N-Caltech101 [23] datasets, which represent to date the most complex benchmarks for event-based classification. N-Cars is a collection of urban scenes recordings (lasting 100ms each) captured with a DVS sensor and showing two object categories: cars and urban background. The dataset comes already split into $7,940$ car and $7,482$ background training samples, and $4,396$ car and $4,211$ background testing samples. The N-Caltech101 collection is an event-based conversion of the popular Caltech-101 [16] dataset obtained by moving an event-based camera in front of a still monitor showing one of the original RGB images. Like the original version, the dataset contains objects from 101 classes distributed amongst $8,246$ samples.

Network Architectures. We used two network configurations to test Matrix-LSTM on both datasets, namely the classifier used in Events-to-Video [26], and the one used to evaluate the EST [10] reconstruction. Both are based on ResNet [11] backbones and pre-trained on ImageNet [7]. Events-to-Video [26] uses a ResNet18 configuration maintaining the first 3 channels convolution (since reconstructed images are RGB) while adding an extra fully-connected layer to account for the different number of classes in both N-Calthec101 and N-Cars (we refer to this configuration as *ResNet–Ev2Vid*). EST [10] instead uses a ResNet34 backbone and replaces both the first and last layers respectively, with a convolution matching the input features, and a fully-connected layer with the proper number of neurons (we refer to this configuration as *ResNet–EST*).

To perform a fair comparison we replicated the two settings, using the same number of channels in the event representation (although we also tried different

Table 1. Results on N-Cars: **(a)** ResNet18–Ev2Vid, variable time encoding, and normalization; **(b)** ResNet18–EST, variable time encoding and number of bins

ResNet Norm	ts absolute	ts relative	delay relative
✓	$95.22 \pm 0.41\%$	$94.77 \pm 1.01\%$	$95.40 \pm 0.59\%$
	$\mathbf{95.75 \pm 0.27}\%$	$95.32 \pm 0.85\%$	$\mathbf{95.80 \pm 0.53}\%$

(a)

		1 bin	2 bins	9 bins
delay	glob+loc	-	$92.68 \pm 1.23\%$	$92.32 \pm 1.02\%$
	local	$92.64 \pm 1.21\%$	$92.35 \pm 0.83\%$	$92.67 \pm 0.90\%$
ts	glob+loc	-	$\mathbf{93.46 \pm 0.84}\%$	$\mathbf{93.21 \pm 0.49}\%$
	local	$92.65 \pm 0.78\%$	$92.75 \pm 1.38\%$	$93.12 \pm 0.68\%$

(b)

channel values) and data augmentation procedures (random horizontal flips and crops of 224×224 pixels). We perform early stopping on a validation set in all experiments, using 20% of the training on N-Cars and using the splits provided by the EST official code repository [9] for N-Caltech101. ADAM [14] was used as optimizer for all experiments with a learning rate of 10^{-4}. Finally, we use a batch size of 64 and a constant learning rate on N-Cars in both configurations. On N-Caltech101, instead, we use a batch size of 16 while decaying the learning rate by a factor of 0.8 after each epoch when testing on *ResNet–Ev2Vid*, and a batch size of 100 with no decay with the *ResNet–EST* setup. Finally, to perform a robust evaluation, we compute the mean and standard deviation values using five different seeds in all the experiments reported in this section.

Results The empirical evaluation is organized as it follows for both *ResNet–Ev2Vid* and *ResNet–EST*. We always perform hyper-parameters search using ResNet18 on N-Cars, being faster to train and thus allowing to explore a larger parameter space. We then select the best configuration to train the remaining architectures, i.e., ResNet34 on N-Cars and both variants on N-Caltech101.

Matrix-LSTM + ResNet-Ev2Vid. We start out with the *ResNet–Ev2Vid* baseline (setting up the Matrix-LSTM to output 3 channels) by identifying the optimal time feature to provide as input to the LSTM, as reported in Table 1a. We distinguish between *ts* and *delay* features and between *absolute* and *relative* scope. The first distinction refers to the type of time encoding, i.e., the timestamp of each event in the case of *ts* feature, or the delay between an event and the previous one in case of *delay*. Time features are always range-normalized between 0 and 1, with the scope distinction differentiating if the normalization takes place before splitting events into pixels (*absolute* feature) or after (*relative* feature). In the case of *ts*, *absolute* means that the first and last events in the sequence have time feature 0 and 1, respectively, regardless of their position, whereas *relative* means that the previous condition holds for each position (x, y). Note that we only consider relative delays since it is only meaningful to compute them between events of the same pixel. Finally, we always add the polarity, obtaining a 2-value feature $f_i^{(x,y)}$. *Delay relative* and *ts absolute* are those providing the best results, with *ts relative* having higher variance. We select *delay relative* as the best configuration. In Table 1a we also show the effect of applying the same

Table 2. Results on N-Cars with ResNet18–EST: **(a)** *polarity + global ts + local ts* encoding, optional SELayer and variable number of bins; **(b)** *polarity + global ts + local ts* encoding, SELayer and variable number of channels

SE	2 bins	4 bins	9 bins	16 bins
	93.46 ± 0.84%	92.68 ± 0.62%	**93.21 ± 0,49**%	92.01 ± 0.45%
✓	**93.71 ± 0.93**%	92.90 ± 0.62%	**93.30 ± 0,47**%	92.44 ± 0.43%

(a)

	Channels		
bins	4	8	16
1	93.88 ± 0.87%	93.60 ± 0.30%	**94.37 ± 0.40**%
2	93.05 ± 0.92%	93.97 ± 0.52%	**94.09 ± 0.29**%
bins	4	7	8
9	92.42 ± 0.65%	93.56 ± 0.46%	93.49 ± 0.84%

(b)

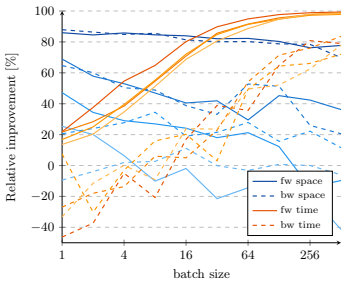
frame normalization used while pre-training the ResNet backbone on ImageNet also to the Matrix-LSTM output. While performing normalization makes sense when training images are very similar to those used in pre-training, as in Events-to-Video [26], we found out that in our case, where no constraint is imposed on the appearance of reconstructions, this does not improve the performance.

Matrix-LSTM + ResNet-EST. We continue the experiments on N-Cars by considering *ResNet–EST* as baseline, where we explore the effect of using bins, i.e., intervals, on the quality of Matrix-LSTM surfaces. Since multiple intervals are involved, we distinguish between *global* and *local* temporal features. The first type is computed on the original sequence $\mathcal{E}$, before splitting events into intervals, whereas the latter locally, within the interval scope $\mathcal{E}_{\tau_b}$. For local features we consider the best options we identified on ResNet-Ev2Vid, namely *delay relative* and *ts absolute*, while we only consider *ts* as global feature since a global delay loses meaning after interval splitting. Results are reported in Table 1b where values for single bin are missing since there is no distinction between *global* and *local* scope. Adding a global feature consistently improves performance. This can indeed help the LSTM network in performing integration conditioned on a global timescale and thus enabling the extraction of temporal consistent features. We use *global ts + local ts* features in next experiments, since this provides better performance and reduced variance, and always add the polarity feature.

The next set of experiments was designed to select the optimal number of bins, searching for the best $B = 2, 4, 9, 16$ as done in EST, while using a fixed *polarity + global ts + local ts* configuration. In these experiments, we also make use of the SELayer [13], a self-attention operation specifically designed to correlate channels. Being the number of channels limited, we always use a reduction factor of 1. Please refer to the paper [13] for more details. As reported in Table 2a, adding the layer consistently improves performance. We explain this by noticing that surfaces computed on successive intervals are naturally correlated and, thus, explicitly modeling this behavior helps in extracting richer features. Finally, we perform the last set of experiments to select the Matrix-LSTM hidden size (which also controls the number of output channels). Results are reported in Table 2b. Note that we only consider $4, 7, 8$ channels with 9 bins to limit the total number of channels after concatenation.

		3x3	5x5
Matrix-LSTM	delay rel	**95.05 ± 0.96%**	**93.38 ± 0.64%**
	ts abs	**94.92 ± 0.74%**	**94.34 ± 0.94%**
ConvLSTM	delay rel	92.33 ± 0.41%	92.65 ± 0.78%
	ts abs	93.97 ± 1.30%	93.61 ± 1.59%

(a)



(b)

Fig. 2. (a) Comparison between Matrix-LSTM and ConvLSTM on N-Cars. **(b)** Space and time relative improvements of Matrix-LSTM over ConvLSTM as a function of input density (from 10% to 100% with 30% steps). Colors refer to different density, from low density (dark colors) to high density (light colors)

Matrix-LSTM vs. ConvLSTM. In Table 2a we compare Matrix-LSTM with ConvLSTM [30] for different choices of kernel size on the N-Cars [31] dataset using the Ev2Vid–ResNet18 backbone. When using ConvLSTM, events are densified in a volume $\tilde{\mathcal{E}}_{dense}$ of shape $N \times T_{max}^{(x,y)} \times H \times W \times F$. Matrix-LSTM performs better on all configurations, despite achieving worst performance than the 1×1 Matrix-LSTM best configuration in Table 1a. Event surfaces produced by the Matrix-LSTM layer are indeed more blurry with larger receptive fields and this may prevent the subsequent ResNet backbone from extracting effective features. Using a 1×1 kernel enables to focus on temporal information while the subsequent convolutional layers deal with spatial correlation.

ConvLSTM, instead, does not properly handle asynchronous data when large receptive fields are considered, and this may explains the performance difference with Matrix-LSTM. Indeed, since pixels at different locations most often fire at different times and with different frequencies, the $\tilde{\mathcal{E}}_{dense}[n, i, :, :, :]$ slice processed by the ConvLSTM in each iteration does not contain all simultaneous events. Using a large ConvLSTM receptive field means to compare a neighborhood of events occurred at different timestamps and therefore not necessarily correlated. Contrary to ConvLSTM, Matrix-LSTM allows for a greater flexibility when large receptive fields are considered since the original events arrival order is preserved and we do not require events to be densified during intermediate steps. We do not compare the two LSTMs on the 1×1 configuration since, when using $\tilde{\mathcal{E}}_{dense}$ as input to ConvLSTM, the two configurations compute the same transformation, despite ConvLSTM having to process more padded values. The two settings are indeed computationally equivalent only in the worst case in which all pixels in the batch happen to receive at least one event (i.e., $P = N \cdot H \cdot W$).

The 1×1 configurations are compared in terms of space and time efficiency in Figure 2b. We use the two layers to extract a 224×224 frame from artificially generated events with increasing density, i.e., the ratio of pixels receiving at least

one event. The reconstruction is performed using PyTorch [32] on a 12GB Titan Xp, by varying the batch size, the LSTM hidden size and the number of events in each active pixel (starting from 1 and increasing by a factor of 2 for the hidden size, while increasing by a factor of 10 for the number of events, until allowed by GPU memory constraints). We compute the relative improvement of Matrix-LSTM in terms of sample reconstruction time and peak processing space (i.e., excluding model and input space) during both forward and backward passes, and finally aggregate the results by batch size computing the mean improvement over all the trials. Matrix-LSTM performs better than ConvLSTM on prediction time, with the time efficiency improving as the batch size increases, while worst than ConvLSTM on memory efficiency in very dense surfaces ($> 70\%$ density). However, this situation is quite uncommon in event-cameras since they only generate events when brightness changes are detected. Uniform parts of the scene that remain unchanged, despite the camera movement, do not appear in the event stream. For instance, the background sky and road in MVSEC [37] make *outdoor_day* sequences only have an average 10% of active pixels.

Discussion. Results of the top performing configurations for both *ResNet-Ev2Vid* and *ResNet-EST* variants on both N-Cars and N-Caltech101 are reported in Table 3. We use *relative delay* with *ResNet-Ev2Vid* and *global ts + local ts* with *ResNet-EST*. Through an extensive evaluation, we show that using Matrix-LSTM representation as input to the baseline networks and training them jointly improves performance by a good margin. Indeed, using the ResNet34-Ev2Vid setup, our solution sets a new state-of-the-art on N-Cars, even surpassing the Events-to-Video model that was trained to extract realistic reconstructions. The same does not happen on N-Caltech101, whose performance usually greatly depends on pre-training also on the original image-based version, and where Events-to-Video has therefore advantage. Despite this, our model only performs 0.9% worse than the baseline. On the ResNet-EST configuration, the model performs consistently better on N-Cars, while slightly worse on N-Caltech101 on most configurations. However, we remark that search for the best configuration was indeed performed on N-Cars, while a hyper-parameter search directly performed on N-Caltech101 would have probably lead to better results.

4.2 Optical flow prediction

For the evaluation of optical flow prediction we used the MVSEC [37] suite. Fusing event-data with lidar, IMU, motion capture and GPS sources, MVSEC is the first event-based dataset to provide a solid benchmark in real urban conditions. The dataset provides ground truth information for depth and vehicle pose and was later extended in [36] with optical flow information extracted from depth-maps. The dataset has been recorded on a range of different vehicles and features both indoor and outdoor scenarios and different lighting conditions.

Network Architecture. We used the EV-FlowNet [36] architecture as reference model. To perform a fair comparison between Matrix-LSTM and the original hand-crafted features, we built our model on top of its publicly available

Table 3. Matrix-LSTM best configurations compared to state-of-the-art

Method	Classifier	Channels (bins)	N-Cars	N-Caltech101
H-First [24]	spike-based	-	56.1	0.54
HOTS [15]	histogram similarity	-	62.4	21.0
Gabor-SNN [31]	SVM	-	78.9	19.6
HATS [31]	SVM	-	90.2	64.2
	ResNet34–EST [10]	-	90.9	69.1
	ResNet18–Ev2Vid [26]	-	90.4	70.0
Ev2Vid [26]	ResNet18–Ev2Vid	3	91.0	**86.6**
Matrix-LSTM (Ours)	ResNet18–Ev2Vid	3 (1)	$\mathbf{95.80 \pm 0.53}$	84.12 ± 0.84
	ResNet34–Ev2Vid	3 (1)	$\mathbf{95.65 \pm 0.46}$	85.72 ± 0.37
EST [10]	ResNet34–EST	2 (9)	92.5	81.7
	ResNet34–EST	2 (16)	92.3	83.7
Matrix-LSTM (Ours)	ResNet18–EST	16 (1)	$\mathbf{94.37 \pm 0.40}$	81.24 ± 1.31
	ResNet34–EST	16 (1)	$\mathbf{94.31 \pm 0.43}$	78.98 ± 0.54
	ResNet18–EST	16 (2)	$\mathbf{94.09 \pm 0.29}$	83.42 ± 0.80
	ResNet34–EST	16 (2)	$\mathbf{94.31 \pm 0.44}$	80.45 ± 0.55
	ResNet18–EST	2 (16)	$\mathbf{92.58 \pm 0.68}$	$\mathbf{84.31 \pm 0.59}$
	ResNet34–EST	2 (16)	92.15 ± 0.73	83.50 ± 1.24

codebase [35]. The code contains few minor upgrades over the paper version, which we made sure to remove as reported in the supplementary materials.

The original network uses a 4-channels event-surface, collecting in pairs of separate channels based on the event polarity, the timestamp of the most recent event, and the number of events occurred in every spatial location. We replaced this representation with a Matrix-LSTM making use of 4 output channels, as well. We trained the model on the *outdoor_day1* and *outdoor_day2* sequences for $300,000$ iterations, as in the original paper. We used the ADAM optimizer with batch size 8, and an initial learning rate of 10^{-5}, exponentially decayed every 4 epochs by a factor of 0.8. We noticed that EV-FlowNet is quite unstable at higher learning rates, while Matrix-LSTM could benefit from larger rates, so we multiply its learning rate, i.e., the Matrix-LSTM gradients, by a factor of 10 during training. Test was performed on a separate set of recordings, namely *indoor_flying1*, *indoor_flying2* and *indoor_flying3*, which are visually different from the training data. The network performance is measured in terms of average endpoint error (AEE), defined as the distance between the endpoints of the predicted and ground truth flow vectors. In addition, as proposed in the KITTI benchmark [21] and as done in [36], we report the percentage of outliers, namely points with endpoint error greater than 3 pixels and 5% of the magnitude ground truth vector. Finally, following the procedure used in [36], we only report the error computed in spatial locations where at least one event was generated.

Results. In the previous classification experiments, we observed that the type of temporal features and the number of bins play an important role in extracting effective representations. We expect time resolution to be a key factor of performance in optical flow, hence, we focus here on measuring how different interval choices impact on the flow prediction. We decided to always use the *polarity + global ts + local ts* configuration, which worked well on N-Cars while considering different bin setups. Results are reported in Table 4.

Table 4. Optical flow estimation on MVSEC

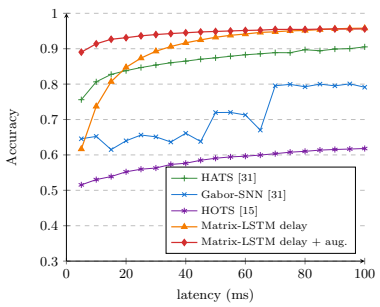
Method		*indoor_flying1*		*indoor_flying2*		*indoor_flying3*	
		AEE	%Outlier	AEE	%Outlier	AEE	%Outlier
Two-Channel Image [20]		1.21	4.49	2.03	22.8	1.84	17.7
EV-FlowNet [36]		1.03	2.20	1.72	15.1	1.53	11.9
Voxel Grid [38]		0.96	1.47	1.65	14.6	1.45	11.4
EST [10]	exp. kernel	0.96	1.27	1.58	10.5	1.40	9.44
	learnt kernel	0.97	0.91	1.38	8.20	1.43	6.47
Matrix-LSTM (Ours)	1 bin	1.017	2.071	1.642	13.88	1.432	10.44
	2 bins	**0.829**	**0.471**	**1.194**	**5.341**	**1.083**	**4.390**
	4 bins	0.969	1.781	1.505	11.63	1.507	12.97
	8 bins	0.881	0.672	1.292	6.594	1.181	5.389
	2 bins + SELayer	**0.821**	**0.534**	**1.191**	**5.590**	**1.077**	**4.805**

As performed on classification, we study the effect of adding a SELayer on the best performing configuration. Correlating the intervals slightly improves the AEE metric in all test sequences but increases the number of outliers. As expected, varying the number of bins has a great impact on performance. The AEE metric, indeed, greatly reduces by simply considering two intervals instead of one. Interestingly, we achieved the best performance by considering only 2 intervals, as adding more bins hurts performance. We believe this behavior resides on the nature of optical flow prediction, where the network is implicitly asked to compare two distinct temporal instants. This configuration consistently improves the baseline up to 30.76% on *indoor_flying2*, highlighting the capability of the Matrix-LSTM to adapt also to low-level tasks.

4.3 Time performance analysis

We compared the time performance of Matrix-LSTM with other event representations following EST [10] and HATS [31] evaluation procedure. In Table 3b we report the time required to compute features on a sample averaged over the whole N-Cars training dataset for both ResNet–Ev2Vid and ResNet–EST configurations. Our surface achieves similar time performance than both HATS and EST, performing only ∼2ms slower than EST on the same setting (9 bins and 2 channels). Similarly, in Table 3c, we compute the mean surface reconstruct time for MVSEC *indoor_flying* test sequences. While EST can exploit parallel batch computation of events within the same sample, since each event feature is processed independently, Matrix-LSTM relies on sequential computation to reconstruct the surface. The custom CUDA kernels we designed, however, enable bins and pixel sequences to be processed in parallel, drastically reducing the processing time. Please, refer to the additional materials for more details. All evaluations are performed with PyTorch on a GeForce GTX 1080Ti GPU.

In Figure 3a we analyze the accuracy-vs-latency trade-off on the N-Cars dataset, as proposed in [31], using the ResNet18-Ev2Vid configuration. While the performance of the model, trained on 100ms sequences, significantly drops when very few milliseconds of events are considered, the proposed method still shows good generalization, achieving better performance than the baselines when more than 20ms of events are used. However, fixing the performance loss on



(a)

Method	Bins	Channels	Asynch.	Time [ms]	Speed [kEV/s]
Gabor-SNN [31]	-	-	Yes	285.95	14.15
HOTS [15]	-	-	Yes	157.57	25.68
HATS [31]	-	-	Yes	7.28	555.74
EST [10]	9	2	No	**6.26**	**632.9**
Matrix-LSTM	1	3	No	10.89	385.7
(Ours)	9	2	No	8.25	468.36

(b)

	Mean reconstruction time (on GPU) [ms]		
Bins	Ev-FlowNet [36] surf.	EST [10]	Matrix-LSTM
1	2.53 ± 2.74	3.62 ± 2.35	3.20 ± 0.97
2	2.01 ± 1.22	3.94 ± 1.47	5.18 ± 1.68
9	2.04 ± 1.20	9.09 ± 1.96	4.92 ± 1.47

(c)

Fig. 3. (a) Accuracy as a function of latency (adapted from [31]). **(b)** Average sample computation time on N-Cars and number of events processed per second. **(c)** Average time to reconstruct the event surface in MVSEC test sequences

small latencies is just a matter of training augmentation: by randomly cropping sequences to variable lengths (from 5ms to 100ms), our method consistently improves the baselines, dynamically adapting to sequences of different lengths.

5 Conclusion

We proposed Matrix-LSTM, an effective method for learning dense event representations from event-based data. By modeling the reconstruction with a spatially shared LSTM we obtained a fully differentiable procedure that can be trained end-to-end to extract the event representation that best fits the task at hand. Focusing on efficiently handling asynchronous data, Matrix-LSTM preserves sparsity during computation and surpasses other popular LSTM variants on space and time efficiency when processing sparse inputs. In this regard, we proposed an efficient implementation of the method that exploits parallel batch-wise computation and demonstrated the effectiveness of the Matrix-LSTM layer on multiple tasks, improving the state-of-the-art of object classification on N-Cars by 3.3% and the performance on optical flow prediction on MVSEC by up to 23.07% over previous differentiable techniques [10]. Although we only integrate windows of events, the proposed mechanism can be extended to process a continuous streams thanks to the LSTM memory that is able to update its representation as soon as a new event arrives. As a future line of research, we plan to explore the use of Matrix-LSTM for more complex tasks such as gray-scale frame reconstruction [26], ego-motion and depth estimation [38,34].

Acknowledgments. We thank Alex Zihao Zhu for his help on replicating Ev-FlowNet results and the ISPL group at Politecnico di Milano for GPU support. This research is supported from project TEINVEIN, CUP: E96D17000110009 - Call "Accordi per la Ricerca e l'Innovazione", cofunded by POR FESR 2014-2020 (Regional Operational Programme, European Regional Development Fund).

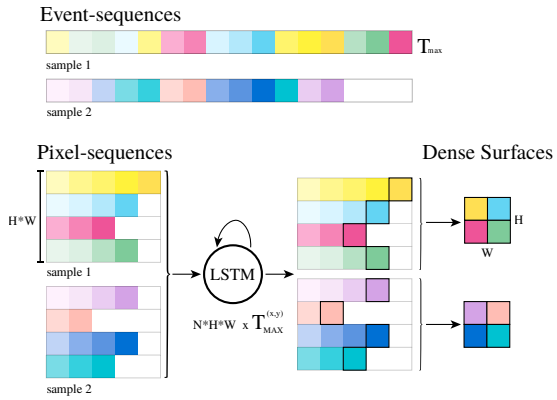


Fig. 4. An example of the *groupByPixel* operation on a batch of $N = 2$ samples and a 2×2 pixel resolution. Different colors refer to different pixel locations while intensity indicates time. For clarity, the features dimension is not shown in the figure

Appendix A Implementation

The Matrix-LSTM feature extraction process can be implemented efficiently by means of two order-aware reshape operations. These two operations, namely *groupByPixel* and *groupByTime*, allow event streams to be split based on the pixel location and temporal bin. After being reshaped, the input is ready to be processed by a single LSTM network, implementing parameter sharing across all pixel locations and temporal windows. In the following we give a detailed overview of the two reshape operators.

A.1 GroupByPixel

This operation translates from event-sequences to pixel-sequences. Let X be a tensor of shape $N \times T_{max} \times F$, representing the features $f_{n,i}^{(x,y)}$ of a batch of N samples, where T_{max} is the length of the longest sequence in the batch. We define the *groupByPixel* mapping on X as an order-aware reshape operation that rearranges the events into a tensor of pixel-sequences of shape $P \times T_{max}^{(x,y)} \times F$ where $T_{max}^{(x,y)}$ is the length of the longest pixel sequence $\mathcal{E}_n^{(x,y)}$ and P is the number of active pixels (i.e., having at least one event) in the batch, which equals $N \cdot H \cdot W$ only in the worst case. Pixel-sequences shorter than $T_{max}^{(x,y)}$ are padded with zero events to be processed in parallel.

The tensor thus obtained is then processed by the LSTM cell that treats samples in the first dimension independently, effectively implementing parameter sharing and applying the transformation in parallel over all the pixels. The LSTM output tensor, which has the same shape of the input one, is then sampled by taking the output corresponding to the last event in each pixel-sequence $\mathcal{E}_n^{(x,y)}$, ignoring values computed on padded values, and the obtained values are then

Table 5. Comparison between Matrix-LSTM and ConvLSTM on both Ev2Vid and EST ResNet18 configurations on the N-Cars dataset

		delay relative		ts absolute	
		3×3	5×5	3×3	5×5
Ev2Vid with 3 chans, 1 bin	Matrix-LSTM (ours)	$\mathbf{95.05 \pm 0.96\%}$	$\mathbf{93.38 \pm 0.64\%}$	$\mathbf{94.92 \pm 0.74\%}$	$\mathbf{94.34 \pm 0.94\%}$
	ConvLSTM [30]	$92.33 \pm 0.41\%$	$92.65 \pm 0.78\%$	$93.97 \pm 1.30\%$	$93.61 \pm 1.59\%$
EST with 16 chans, 1 bin	Matrix-LSTM (ours)	$\mathbf{93.14 \pm 0.77\%}$	$\mathbf{92.18 \pm 0.28\%}$	$\mathbf{92.83 \pm 1.32\%}$	$\mathbf{92.15 \pm 0.67\%}$
	ConvLSTM [30]	$90.39 \pm 0.94\%$	$90.73 \pm 1.05\%$	$92.52 \pm 1.26\%$	$92.05 \pm 0.56\%$

used to populate the dense representation. To improve efficiency, for each pixel-sequence $\mathcal{E}_n^{(x,y)}$, *groupByPixel* keeps also track of the original spatial position (x, y), the index of the sample inside the batch and the length of the pixel-sequence $T_n^{(x,y)}$, namely the index of the last event before padding. Given this set of indexes, the densification step can be performed as a simple slicing operation. See Figure 4 for visual clues. *groupByPixel* is implemented as a custom CUDA kernel that processes each sample in parallel and places each event feature in the output tensor maintaining the original temporal order.

A.2 GroupByTime

The Matrix-LSTM variant that operates on temporal bins performs a similar pre-processing step. Each sample in the batch is divided into a fixed set of intervals. The *groupByTime* cuda kernel pre-processes the input events generating a $N * B \times T_{max}^b \times F$ tensor where the B bins are grouped in the first dimension and taking care of properly padding intervals (T_{max}^b is the length of the longest bin in the batch). The Matrix-LSTM mechanism is then applied as usual and the resulting $N * B \times H \times W \times C$ tensor is finally reshaped into a $N \times H \times W \times B * C$ event-surface.

Appendix B Matrix-LSTM vs ConvLSTM

In Figure 2a of the paper we report a comparison between ConvLSTM and Matrix-LSTM using the Ev2Vid–ResNet18 configuration, i.e., the configuration of choice for all comparisons in the paper. For completeness, in Table 5 we extend the evaluation also to EST, using the 16 channels and 1 bin setting, which is one of our best performing EST configuration on N-Cars. Matrix-LSTM performs better on all experiments, highlighting its capabilities to better handle asynchronous event-based data if compared to ConvLSTM. We also highlight that the performance improvement is greater on *delay relative* temporal features than on *ts absolute* ones. ConvLSTM, indeed, processes temporal slices containing potentially uncorrelated events that happened at different time instants. While delays are always consistent within each pixel sequence, they are not within the ConvLSTM kernel receptive field. Using an absolute temporal encoding alleviates this issue on both Ev2Vid and EST architectures, while still performing worst than Matrix-LSTM. Our extraction layer, indeed, preserves

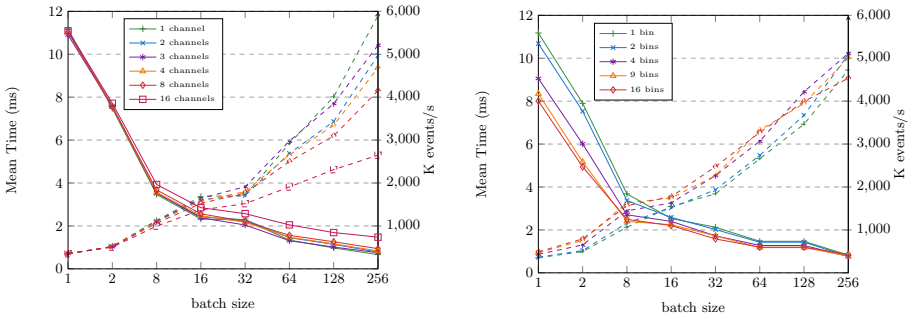


Fig. 5. Number of processed events per second (dashed lines) and timing (solid lines) with varying number of channels **(a)**, and bins **(b)**

the original events arrival order within each receptive field during features extraction, which allows to achieve better performance both on *ts absolute* and *delay relative* input features. Moreover, structured *delay relative* features perform better on Matrix-LSTM than simple absolute features.

Appendix C Time Performance

While the performance reported in Figure 3b of the paper are computed on each sample independently to enable a fair comparison with the other methods, in Figure 5a and Figure 5b we study instead how the mean time required to process a sample over all the N-Cars training dataset and the corresponding events throughput change as a function of the batch size. Both performance dramatically increase when multiple samples are processed simultaneously in batch. This is crucial at training time, when optimization techniques greatly benefit from batch computation.

Furthermore, while increasing the number of output channels, for the same choice of batch size, increases the time required to process each sample (since the resulting Matrix-LSTM operates on a larger hidden state), increasing the number of bins has an opposite behaviour. Multiple intervals are indeed processed independently and in parallel by the Matrix-LSTM that has to process a smaller number of events in each spatial location, sequentially. In both configurations, finally, increasing the batch size reduces the mean processing time.

Appendix D Classification

We perform additional experiments on the N-MNIST dataset [23] and on the newly introduced ASL-DVS [3] dataset. On N-MNIST we directly compare with

Table 6. Classification accuracy (%) on the N-MNIST [23] dataset.

Method	Classifier	Channels (bins)	Accuracy
H-First	spike-based	-	
HOTS [15]	histogram similarity	-	80.8
HATS [31]	SVM	-	99.1
G-CNN [3]	Graph CNN	-	98.5
RG-CNN [3]	Graph CNN	-	**99.0**
Events Count [3]	ResNet50	2 (1)	98.4
Ev2Vid [26]	Ev2Vid custom convnet	1 (1)	98.3
Matrix-LSTM (Ours)	Ev2Vid custom convnet	1 (1)	$\mathbf{98.9 \pm 0.21}$

Table 7. Classification accuracy (%) on the ASL-DVS [3] dataset.

Method	Classifier	Channels (bins)	Accuracy
G-CNN [3]	Graph CNN	-	87.5
RG-CNN [3]	Graph CNN	-	90.1
Events Count [3]	ResNet50	2 (1)	88.6
EST [10]	ResNet50	2 (1)	99.57
Matrix-LSTM (Ours)	ResNet50	2 (1)	$\mathbf{99.73 \pm 0.04}$

the Ev2Vid [26] reconstruction procedure, where the custom convolutional network proposed in [26] is used as backbone, while we compare with the EST [10] surface on ASL-DVS, making use of ResNet50 [11] as backbone. On both cases, Matrix-LSTM performs better than other event-surface mechanisms and also outperforms alternative classification architectures.

Appendix E Optical Flow Prediction

E.1 Ev-FlowNet Baseline Results

We performed optical flow experiments starting from the publicly available Ev-FlowNet codebase [35] and replacing the original hand-crafted features with the proposed Matrix-LSTM layer. We first made sure to revert the baseline architecture to the original configuration, checking that we were able to replicate the paper results. Indeed, the public code contains minor upgrades over the paper version. We contacted the authors that provided us with the needed modifications. These consist of removing the batch normalization layers, setting to 2 the number of output channels of the layer preceding the optical flow prediction layer, and disabling random rotations during training. For completeness, we report the results we obtained by training the baseline from scratch with these fixes in Table 8.

To test how the network adapts to different flow magnitudes, the Ev-FlowNet [36] was tested on two evaluation settings for each test sequence: with input frames and corresponding events that are one frame apart (denoted as *dt=1*), and with frames and events four frames apart (denoted as *dt=4*). While we were able to closely replicate the results of the first configuration (*dt=1*), with a minor

Table 8. Effect of adding a Squeeze-and-Excitation layer on the optical flow prediction task

Method		*indoor_flying1*				*indoor_flying2*				*indoor_flying3*			
		dt=1		*dt=4*		*dt=1*		*dt=4*		*dt=1*		*dt=4*	
		AEE	%Outlier	AEE	%Outlier	AEE	%Outlier	AEE	%Outlier	AEE	%Outlier	AEE	%Outlier
Ev-FlowNet [36]	-	1.03	2.2	2.25	24.7	2.12	15.1	4.05	45.3	1.53	11.9	3.45	39.7
Ev-FlowNet (ours)	-	1.015	2.736	3.432	48.685	1.606	12.089	5.957	63.226	1.548	11.937	5.247	57.662
Matrix-LSTM (Ours)	1 bin	1.017	2.071	3.366	42.022	1.642	13.89	5.870	65.379	1.432	10.44	5.015	57.094
	2 bins	0.829	**0.471**	**2.269**	**23.558**	1.194	**5.341**	**3.946**	**42.450**	1.083	**4.390**	**3.172**	**31.975**
	2 bins + SELayer	**0.821**	0.534	2.378	25.995	**1.191**	5.590	4.333	45.396	**1.077**	4.805	3.549	36.822
	4 bins	0.969	1.781	3.023	36.085	1.505	11.63	4.870	49.077	1.507	12.97	4.652	43.267
	4 bins + SELayer	**0.844**	**0.634**	**2.330**	**24.777**	**1.213**	**6.057**	**4.322**	**44.769**	**1.070**	**4.625**	**3.588**	**36.442**
	8 bins	**0.881**	**0.672**	**2.290**	**24.203**	1.292	**6.594**	**3.978**	**42.230**	1.181	5.389	**3.346**	**33.951**
	8 bins + SELayer	0.905	0.885	2.308	24.597	**1.286**	6.761	4.046	44.366	**1.177**	**5.318**	3.391	35.452

improvement in the *indoor_flying2* sequence, the performance we obtain on the *dt=4* setup is instead worse on all sequences, as reported on the first two rows of Table 8.

Despite this discrepancy, which prevents the Matrix-LSTM performance on *dt=4* settings to be directly compared with the results reported on the Ev-FlowNet paper, we can still evaluate the benefits of our surface on larger flow magnitudes. Indeed, this work evaluates the Matrix-LSTM layer based on the relative performance improvement obtained by substituting the original features with our layer. Using our Ev-FlowNet results as baseline, we show that Matrix-LSTM is able to improve the optical flow quality even on the *dt=4* setting, highlighting the capability of the layer to adapt to different sequence lengths and movement conditions. We report an improvement of up to 30.426% on *dt=1* settings and up to 39.546% on *dt=4* settings using our results as baseline.

E.2 Squeeze-and-Excitation Layer

Optical flow prediction is a complex task that requires neural networks to extract accurate features precisely describing motion inside the scene. An event aggregation mechanism is therefore required to extract rich temporal features from the events. In Section 4.2 of the paper we show that time resolution is a key factor for extracting effective feature with Matrix-LSTM. In particular, increasing the number of bins has great impact on the predicted flow and allows the network to retain temporal information over long sequences. Here we focus, instead, on the effect of correlating temporal features by adding a SELayer to the Matrix-LSTM output. Table 8 reports the performance obtained using this additional layer on the MVSEC [37] task. The results we obtained show that adding an SELayer only improves performance on the 4 bins configuration for the *dt=4* benchmark, while it consistently helps reducing the *AEE* metric on the *dt=1* setting.

By comparing features obtained from subsequent intervals, the SELayer adaptively recalibrates features and helps modelling interdependencies between time instants, which is crucial for predicting optical flow. We believe that a similar approach can also be applied to other event aggregation mechanisms based on voxel-grids of temporal bins to improve their performance, especially those employing data driven optimization mechanisms [10].

Appendix F Qualitative Results

The event aggregation process performed by the Matrix-LSTM layer is incremental. Events in each pixel location are processed sequentially; state and output of the LSTM are updated each time. We propose to visualize the Matrix-LSTM surface as an RGB image by using the ResNet18-Ev2Vid configuration and interpreting the 3 output channels as RGB color. A video of such visualization showing the incremental frame reconstruction on N-Caltech101 samples is provided at this url: `https://marcocannici.github.io/matrixlstm`.

We use a similar visualization technique to show optical flow predictions for *indoor_flying* sequences. Since we use our best performing model that uses 2 temporal bins, we decide to only show the first 3 channels of each temporal interval. Moreover, instead of visualizing how the event representation builds as new events arrive, we only show the frame obtained after having processed each window of events.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: Tensorflow: A system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). pp. 265–283 (2016)
2. Berner, R., Brandli, C., Yang, M., Liu, S.C., Delbruck, T.: A 240× 180 10mw 12us latency sparse-output vision sensor for mobile applications. In: 2013 Symposium on VLSI Circuits. pp. C186–C187. IEEE (2013)
3. Bi, Y., Chadha, A., Abbas, A., Bourtsoulatze, E., Andreopoulos, Y.: Graph-based object classification for neuromorphic vision sensing. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 491–501 (2019)
4. Cannici, M., Ciccone, M., Romanoni, A., Matteucci, M.: Asynchronous convolutional networks for object detection in neuromorphic cameras. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 0–0 (2019)
5. Cannici, M., Ciccone, M., Romanoni, A., Matteucci, M.: Attention mechanisms for object recognition with event-based cameras. In: 2019 IEEE Winter Conference on Applications of Computer Vision (WACV). pp. 1127–1136. IEEE (2019)
6. Cohen, G.K.: Event-Based Feature Detection, Recognition and Classification. Theses, Université Pierre et Marie Curie - Paris VI (Sep 2016)
7. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)
8. Gallego, G., Delbruck, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., Leutenegger, S., Davison, A., Conradt, J., Daniilidis, K., et al.: Event-based vision: A survey. arXiv preprint arXiv:1904.08405 (2019)
9. Gehrig, D., Loquercio, A., Derpanis, K.G., Scaramuzza, D.: End-to-end learning of representations for asynchronous event-based data. `https://github.com/uzh-rpg/rpg_event_representation_learning`
10. Gehrig, D., Loquercio, A., Derpanis, K.G., Scaramuzza, D.: End-to-end learning of representations for asynchronous event-based data. In: IEEE International Conference of Computer Vision (ICCV) (October 2019)
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
12. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
13. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 7132–7141 (2018)
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. Int. Conf. on Learning Representations (ICLR) (2015)
15. Lagorce, X., Orchard, G., Galluppi, F., Shi, B.E., Benosman, R.B.: Hots: a hierarchy of event-based time-surfaces for pattern recognition. IEEE transactions on pattern analysis and machine intelligence **39**(7), 1346–1359 (2016)
16. Li Fei-Fei, Fergus, R., Perona, P.: One-shot learning of object categories. IEEE Transactions on Pattern Analysis and Machine Intelligence **28**(4), 594–611 (April 2006)

17. Lichtsteiner, P., Posch, C., Delbruck, T.: A 128 × 128 120 db 15μs latency asynchronous temporal contrast vision sensor. IEEE journal of solid-state circuits **43**(2), 566–576 (2008)
18. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2015)
19. Maass, W.: Networks of spiking neurons: the third generation of neural network models. Neural networks **10**(9), 1659–1671 (1997)
20. Maqueda, A.I., Loquercio, A., Gallego, G., García, N., Scaramuzza, D.: Event-based vision meets deep learning on steering prediction for self-driving cars. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5419–5427 (2018)
21. Menze, M., Geiger, A.: Object scene flow for autonomous vehicles. In: Conference on Computer Vision and Pattern Recognition (CVPR) (2015)
22. Neil, D., Pfeiffer, M., Liu, S.C.: Phased lstm: Accelerating recurrent network training for long or event-based sequences. In: Advances in neural information processing systems. pp. 3882–3890 (2016)
23. Orchard, G., Jayawant, A., Cohen, G.K., Thakor, N.: Converting static image datasets to spiking neuromorphic datasets using saccades. Frontiers in neuroscience **9**, 437 (2015)
24. Orchard, G., Meyer, C., Etienne-Cummings, R., Posch, C., Thakor, N., Benosman, R.: Hfirst: a temporal approach to object recognition. IEEE transactions on pattern analysis and machine intelligence **37**(10), 2028–2040 (2015)
25. Posch, C., Serrano-Gotarredona, T., Linares-Barranco, B., Delbruck, T.: Retinomorphic event-based vision sensors: bioinspired cameras with spiking output. Proceedings of the IEEE **102**(10), 1470–1484 (2014)
26. Rebecq, H., Ranftl, R., Koltun, V., Scaramuzza, D.: Events-to-video: Bringing modern computer vision to event cameras. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3857–3866 (2019)
27. Scheerlinck, C., Rebecq, H., Stoffregen, T., Barnes, N., Mahony, R., Scaramuzza, D.: Ced: Color event camera dataset. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 0–0 (2019)
28. Sekikawa, Y., Hara, K., Saito, H.: Eventnet: Asynchronous recursive event processing. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3887–3896 (2019)
29. Serrano-Gotarredona, T., Linares-Barranco, B.: A 128 × 128 1.5% contrast sensitivity 0.9% fpn 3μs latency 4mw asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers. IEEE Journal of Solid-State Circuits **48**(3), 827–838 (2013)
30. SHI, X., Chen, Z., Wang, H., Yeung, D.Y., Wong, W.k., WOO, W.c.: Convolutional lstm network: A machine learning approach for precipitation nowcasting. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 802–810. Curran Associates, Inc. (2015)
31. Sironi, A., Brambilla, M., Bourdis, N., Lagorce, X., Benosman, R.: Hats: Histograms of averaged time surfaces for robust event-based object classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1731–1740 (2018)
32. Steiner, B., DeVito, Z., Chintala, S., Gross, S., Paszke, A., Massa, F., Lerer, A., Chanan, G., Lin, Z., Yang, E., et al.: Pytorch: An imperative style, high-

performance deep learning library. Advances in Neural Information Processing Systems **32** (2019)
33. Wang, Q., Zhang, Y., Yuan, J., Lu, Y.: Space-time event clouds for gesture recognition: From rgb cameras to event cameras. In: 2019 IEEE Winter Conference on Applications of Computer Vision (WACV). pp. 1826–1835. IEEE (2019)
34. Ye, C., Mitrokhin, A., Fermüller, C., Yorke, J.A., Aloimonos, Y.: Unsupervised learning of dense optical flow, depth and egomotion from sparse event data. arXiv preprint arXiv:1809.08625 (2018)
35. Zhu, A., Yuan, L., Chaney, K., Daniilidis, K.: Ev-flownet: Self-supervised optical flow estimation for event-based cameras. `https://github.com/daniilidis-group/EV-FlowNet`
36. Zhu, A., Yuan, L., Chaney, K., Daniilidis, K.: Ev-flownet: Self-supervised optical flow estimation for event-based cameras. In: Proceedings of Robotics: Science and Systems. Pittsburgh, Pennsylvania (June 2018)
37. Zhu, A.Z., Thakur, D., Özaslan, T., Pfrommer, B., Kumar, V., Daniilidis, K.: The multivehicle stereo event camera dataset: An event camera dataset for 3d perception. IEEE Robotics and Automation Letters **3**(3), 2032–2039 (2018)
38. Zhu, A.Z., Yuan, L., Chaney, K., Daniilidis, K.: Unsupervised event-based learning of optical flow, depth, and egomotion. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 989–997 (2019)