Event Neural Networks

Matthew Dutson, Yin Li, and Mohit Gupta

University of Wisconsin–Madison, Madison WI 53715, USA
{dutson,yin.li,mgupta37}@wisc.edu

Abstract. Video data is often repetitive; for example, the contents of adjacent frames are usually strongly correlated. Such redundancy occurs at multiple levels of complexity, from low-level pixel values to textures and high-level semantics. We propose Event Neural Networks (EvNets), which leverage this redundancy to achieve considerable computation savings during video inference. A defining characteristic of EvNets is that each neuron has state variables that provide it with long-term memory, which allows low-cost, high-accuracy inference even in the presence of significant camera motion. We show that it is possible to transform a wide range of neural networks into EvNets without re-training. We demonstrate our method on state-of-the-art architectures for both high- and low-level visual processing, including pose recognition, object detection, optical flow, and image enhancement. We observe roughly an order-of-magnitude reduction in computational costs compared to conventional networks, with minimal reductions in model accuracy.

Keywords: efficient neural networks; adaptive inference; video analysis

1 Introduction

Real-world visual data is repetitive; that is, it has the property of *persistence*. For example, observe the two frames in Fig. 1 (a). Despite being separated by one second, they appear quite similar. Human vision relies on the persistent nature of visual data to allocate limited perceptual resources. Instead of ingesting the entire scene at high resolution, the human eye points the fovea (a small region of dense receptor cells) at areas containing motion or detail [51]. This allocation of attention reduces visual processing and eye-to-brain communication.

Processing individual frames using artificial neural networks has proven to be a competitive solution for video inference [55,60]. This paradigm leverages advances in *image* recognition (e.g., pose estimation or object detection) and processes each frame independently without considering temporal continuity, implicitly assuming that adjacent frames are statistically independent. This assumption leads to inefficient use of resources due to the repeated processing of image regions containing little or no new information.

There has been recent interest in leveraging temporal redundancy for efficient video inference. One simple solution is to skip processing image regions

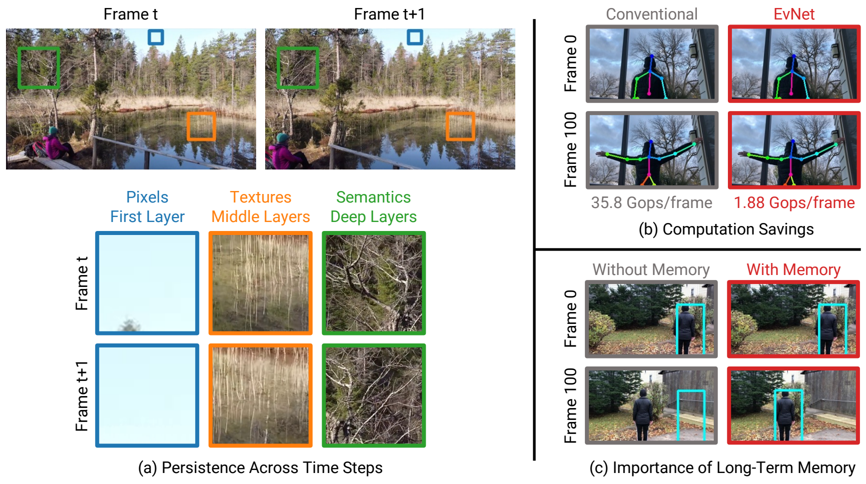

Fig. 1. Event Neural Networks. (a) Two frames from a video sequence, separated by 1 second. Video source: [45]. Over this time, some areas of the image maintain consistent pixel values (sky region). However, these areas only represent a small fraction of the frame. In other regions, the pixel values change but the textures (vertical lines) or semantics (tree branches) remain the same. Each type of persistence corresponds to a different depth in the neural hierarchy. EvNets leverage temporal persistence in video streams across multiple levels of complexity. **(b)** EvNets yield significant computation savings while maintaining high accuracy. **(c)** Event neurons have state variables that encode long-term memory, allowing EvNets to perform robust inference even over long video sequences with significant camera motion. A network without long-term memory (left) fails to correctly track the object due to gradual error accumulation.

containing few changes in pixel values. However, such methods cannot recognize persistence in textures, patterns, or high-level semantics when it does not coincide with persistent pixel values. See Fig. 1 (a).

Because neural networks extract a hierarchy of features from their inputs, they contain a built-in lens for detecting repetition across many levels of visual complexity. Shallow layers detect low-level patterns, and deep layers detect high-level semantics. Temporal repetition at a given level of complexity translates to persistent values at the corresponding depth in the neural hierarchy [14]. Based on this observation, we propose Event Neural Networks (EvNets), a family of neural networks in which neurons transmit (thereby triggering downstream computation) only when there is a significant change in their activation. By applying this strategy over all neurons and layers, we detect and exploit temporal persistence across many levels of complexity.

One of the defining features of EvNets is that each neuron has state variables that provide it with *long-term memory*. Instead of re-computing from scratch for every new input, an EvNet neuron accumulates information over time. Long-term memory allows EvNets to perform robust inference over long video sequences containing significant camera motion. See Fig. 1 (c).

We design various structural components for EvNets – both at the individual neuron level (memory state variables) and the network level (layers and transmission policies). We recognize that transmission policies, in particular, are critical for achieving a good accuracy/computation tradeoff, and we describe the policy design space in detail. We show that, with these components, it is possible to transform a broad class of conventional networks into EvNets *without re-training*. We demonstrate our methods on state-of-the-art models for several high- and low-level tasks: pose recognition, object detection, optical flow, and image enhancement. We observe approximately an order-of-magnitude reduction in arithmetic operations with minimal effects on model accuracy.

Scope and Limitations. In this paper, we focus on the theoretical and conceptual properties of EvNets. Although we show results on several video inference tasks, our goal is not to compete with the latest methods for these tasks in terms of accuracy. Instead, we show that, across a range of models and tasks, EvNets can significantly reduce computational costs without decreasing accuracy.

In most of our analyses we do not assume a specific hardware platform or computation model. We mainly report arithmetic and memory operations (a platform-invariant measure of computational cost) instead of wall-clock time (which depends on many situational variables). An important next step is to consider questions relating to the design of hardware-software stacks for EvNets, in order to minimize latency and power consumption.

2 Related Work

E cient Neural Networks. There are numerous methods for reducing the computational cost of neural networks. Many architectures have been designed to require fewer parameters and arithmetic operations [18,22,29,30,41,59]. Another line of work uses low-precision arithmetic to achieve computation savings [8,21,40,49]. Our approach is complementary to both architecture- and precision-based efficiency methods. These methods reduce the cost of inference on a single time step, whereas EvNets eliminate repetitive computation between multiple time steps. Pruning algorithms [15,16,25,26] remove redundant neurons or synapses during training to improve efficiency. Instead of pruning universally redundant neurons, an EvNet adaptively ignores temporally redundant neurons.

Adaptive Networks. Adaptive models modify their computation based on the input to suit the difficulty of each inference. Prior approaches consider an ensemble of sub-networks [20,50], equip a network with multiple exits [19,48], select the input resolution at inference time [7,32,57], or dynamically choose the feature resolution [53]. These methods are designed for image recognition tasks and do not explore temporal redundancy. Further, many require custom tailoring or re-training for each task and architecture. In contrast, EvNets can be readily integrated into many existing architectures and do not require re-training.

Temporal Redundancy. Several recent approaches consider temporal redundancy for efficient video inference. Many take a keyframe-oriented approach, computing expensive features on keyframes, then transforming those features for the other frames [6,23,27,44,60,61]. Other methods include using visual trackers [56], skipping redundant frames [13,54], reusing previous frame features [33], distilling results from previous time steps [36], two-stream computation [11], and leveraging video compression [52]. In general, these methods require extensive modifications to the network architecture.

Skip-convolution networks (Skip-Conv) [14] are closely related to EvNets. Skip-Conv reuses activation values that have not changed significantly between frames. However, the algorithm only tracks changes between consecutive frames and thus requires frequent re-initialization to maintain accuracy. Re-initialization leads to reduced efficiency, especially in the presence of camera motion. In contrast, the long-term memory in an EvNet maintains accuracy and efficiency over hundreds of frames, even when there is strong camera motion. See Fig. 1 (c).

Sigma-Delta networks [37] exploit temporal redundancy by quantizing the changes in neuron activations. Sigma-Delta networks have been limited so far to simple tasks like digit classification. Unlike Sigma-Delta networks, EvNets do not require quantization (although they do allow it). Compared to Sigma-Delta networks, EvNets achieve superior accuracy/computation tradeoffs (Fig. 5) and generalize better to challenging, real-world tasks (Fig. 7).

DeltaCNN [39] is concurrent work with similar goals to this paper. Like EvNets, DeltaCNN models have mechanisms for integrating long-term changes. They focus on translating theoretical speedups into GPU wall-time savings by enforcing structured sparsity (all channels at a given location transmit together). Despite its practical benefits, this design is inefficient when there is camera motion. In contrast, we emphasize broad conceptual frameworks (e.g., arbitrary sparsity structure) with an eye toward future hardware architectures (Sec. 6).

Event Sensor Inference. Event sensors [28] generate sparse frames by computing a quantized temporal gradient at each pixel. Many networks designed for inference on event-sensor data have efficient, sparse dynamics [4,34]. However, they make strong assumptions about the mathematical properties of the network (e.g., that it is piecewise linear [34]). EvNets place far fewer constraints on the model and are compatible with a broad range of existing architectures.

Recurrent Neural Networks (RNNs). EvNets use long-term memory to track changes, and are thus loosely connected to RNNs. Long-term memory has been widely adopted in RNNs [17]. Several recent works also propose adaptive inference for RNNs by learning to skip state updates [3] or updating state variables only when a significant change occurs [35,38]. Unlike EvNets, these approaches are tailored for RNNs and generally require re-training.

3 Event Neurons

Consider a neuron in a conventional neural network. Let $x = [x_1, x_2, \ldots, x_n]$ be the vector of input values and y be the output. Suppose the neuron composes a linear function g (e.g., a convolution) with a nonlinear activation f. That is,

$$g(x) = \sum_{i=1}^{n} w_i x_i; \quad y = f(g(x)), \tag{1}$$

Fig. 2. Sparse, Delta-Based Transmission. (a) Conventional neurons completely recompute their activations on each time step. (b) Value-based event neurons only transmit activations that have changed signi cantly. However, a value-based transmission can still trigger many computations. (c) Delta-based event neurons only transmit di erential updates to their activations.

where $w = [w_1; w_2; :::; w_n]$ contains the weights of the function g. In a conventional network, every neuron recomputes f and g for every input frame (Fig. 2 (a)), resulting in large computational costs over a video sequence.

Inspired by prior methods that exploit persistence in activations [3,14,37], we describe a class of *event neurons* with *sparse*, *delta-based* transmission.

Sparse, Delta-Based Transmission. An event neuron transmits its output to subsequent layers only when there is a su cient change between its current activation and the previous transmission. This gating behavior makes the layer output *sparse*. However, a *value* transmission may still trigger many downstream computations (neurons receiving updated input values must recompute their activations from scratch). See Fig. 2 (b). Therefore, instead of transmitting an activation *value*, an event neuron transmits a *delta* (di erential).

Suppose a neuron receives a vector of incoming di erentials $_{\text{in}}$ (one element per incoming synapse). $_{\text{in}}$ is sparse, i.e., it only contains nonzero values for upstream neurons that have transmitted. The updated g is given by

$$g(x + \ _{\text{in}}) = g(x) + g(\ _{\text{in}}): \qquad (2)$$

Instead of computing $g(x + \ _{\text{in}})$ from scratch, an event neuron stores the value of $g(x)$ in a state variable a. When it receives a new input $_{\text{in}}$, the neuron retrieves the old value of $g(x)$ from a, computes $g(\ _{\text{in}})$, and saves the value $g(x) + g(\ _{\text{in}})$ in a. This process only requires computing products $w_i x_i$ for nonzero elements of $_{\text{in}}$, i.e., computation scales linearly with the number of transmissions.

The activation function f is nonlinear, so we cannot update it incrementally like g. Whenever a changes, we recompute $f(a)$, then store the updated value in another state variable. f is usually a lightweight function (e.g., ReLU), so the cost of recomputing f is far smaller than computing the products $w_i x_i$.

3.1 Building Event Neurons

An event neuron consists of three state variables, as shown in Fig. 3. The *accumulator* (a) stores the current value of $g(x)$. The *best estimate* (b) stores the current value of $f(a)$. The *di erence* (d) stores di erence between b and the most

Fig. 3. Building Event Neurons. The state variables and update rules in an event neuron. The incremental updates to a convert from a delta-based representation to value-based. The subtraction f (a) b returns the output to delta-based.

recent output. When a neuron receives a di erential update $^{(t)}_{\text{in}}$ at time t from one or more of its inputs, it updates these state variables as follows:

$$a^{(t+1)} = a^{(t)} + g(\ ^{(t)}_{\text{in}}); \quad d^{(t+1)} = d^{(t)} + f(a^{(t+1)})\ \ b^{(t)}; \quad b^{(t+1)} = f(a^{(t+1)}): \tag{3}$$

A neuron transmits an output $_{\text{out}}$ when some condition ond is satis ed. This condition is de ned by the transmission policy. The transmission policy also gives the relationship betweend and $_{\text{out}}$. The policies in this paper simply set $_{\text{out}} = d$. However, other relationships are possible, and the properties described in Sec. 3.2 hold for other relationships. After a neuron transmits, it setsd to d $_{\text{out}}$. See Sec. 4.3 for more details on transmission policies.

3.2 Properties of Event Neurons

Long- and Short-Term Memory. The state variable d accumulates all not-yet-transmitted corrections to the neuron output. It represents the neuron's long-term memory, whereas b represents its short-term memory. Including a long-term memory keeps the neuron from discarding information when it does not transmit. This error-retention property grants certain guarantees on the neuron's behavior, as we demonstrate next.

Error Retention. Consider an event neuron receiving a series of inputs over T time steps, $^{(1)}_{\text{in}};\ ^{(2)}_{\text{in}}; : : : ;\ ^{(T)}_{\text{in}}$. Assume that the state variables a, b, and d have initial values $a^{(0)}$, $f(a^{(0)})$, and zero, respectively. Let the transmitted output values at each time step be $^{(1)}_{\text{out}};\ ^{(2)}_{\text{out}}; : : : ;\ ^{(T)}_{\text{out}}$ (some of these may be zero). By repeatedly applying the neuron update rules, we arrive at the state

$$\begin{gathered} a^{(T)} = a^{(0)} + g\left(\sum_{t=1}^{T}\ ^{(t)}_{\text{in}}\right); \quad b^{(T)} = f(a^{(T)}); \\ d^{(T)} = b^{(T)}\ \ b^{(0)}\ \ \sum_{t=1}^{T}\ ^{(t)}_{\text{out}}: \end{gathered} \tag{4}$$

See the supplementary material for a detailed derivation. Observe thatd is equal to the di erence between the actual and transmitted changes in the activation. This is true regardless of the order or temporal distribution of the $_{\text{in}}$ and

$_{out}$. Because the neuron stores d, it always has enough information to bring the transmitted activation into exact agreement with the true activation b. We can use this fact to bound the error within an EvNet. For example, we can constrain each neuron's error to the range $[\ h; +h]$ by transmitting when $|d| > h$.

The Importance of Long-Term Memory. For comparison, consider a model in which neurons compute the di erence between b on adjacent time steps, then either transmit or discard this di erence without storing the remainder. This is the model used in Skip-Conv [14]. Under this model, the nal state of a neuron depends strongly on the order and temporal distribution of inputs.

For example, suppose a neuron transmits if the frame-to-frame di erence exceeds a threshold . Consider a scenario where the neuron's activation gradually increases from 0 to 2 in steps $0.1\ ;\ 0.2\ ;\ldots;\ 2$. Gradual changes like this are common in practice (e.g., when panning over a surface with an intensity gradient). Because $0.1\ <$, the neuron never transmits and ends in a state with error 2 . The neuron carries this error into all of its future computations. Furthermore, because the neuron discards non-transmitted activations, it has no way to know that this 2 error exists.

4 Event Networks

4.1 Building Event Networks

So far, we have considered the design and characteristics of individual event neurons. In this section, we broaden our view and consider layers and networks. A \layer" is an atomic tensor operation (e.g., a convolution). By this de nition, g and f as de ned in Sec. 3.1 correspond to two di erent layers.

We de ne three new layer types. An accumulator layer consists of a state vector a that contains the a variables for a collection of neurons. A gate layer contains state vectors b and d and the transmission policy. A bu er layer stores its inputs in a state vector x for future use by the next layer; this is required before non-pointwise, nonlinear layers like max pooling. The state vectors a, b and d are updated using vectorized versions of the rules in Eq. 3. An accumulator layer converts its input from delta-based to value-based, whereas a gate converts from value-based to delta-based.

To create an EvNet, we insert gates and accumulators into a pretrained network such that linear layers receive delta inputs and nonlinear layers receive value inputs (Fig. 4). Note that residual connections do not require any special treatment { in an EvNet, residual connections simply carry deltas instead of values. These deltas are added or concatenated to downstream deltas when the residual branch re-joins the main branch (like in a conventional network).

We place a gate at the beginning of the network and an accumulator at the end. At the input gate, we use pixel values instead of $f\ (a)$ and update b and d at every timestep. At the output accumulator, we update a sparsely but read all its elements at every frame. Throughout the model, the functions computed by the preexisting layers (the f and g) remain the same.

Fig. 4. Building Event Networks. We insert accumulators and gates to make the input to linear layers (e.g., convolutions, fully-connected layers) delta-based and the input to nonlinear layers (e.g., ReLU activations) value-based.

4.2 Network Initialization

The equations in Sec. 3.1 de ne how to update the neuron state variables, but they do not specify those variables' initial values. Consider a simple initialization strategy where $a = 0$ and $d = 0$ for all neurons. Since the activation function f is nonlinear, the value of the state variable $b = f(a)$ may be nonzero. This nonzero b usually translates to a nonzero value of a in the next layer. However, we initialized $a = 0$ for all neurons. We have an inconsistency.

To address this problem, we de ne the notion of internal consistency. Consider a neuron with state variables a, d, and b. Let b_{in} and d_{in} be vectors containing the states of the neurons in the previous layer. We say that a network is in an internally consistent state if, for all neurons,

$$a = g(b_{in} \quad d_{in}); \quad b = f(a): \tag{5}$$

The simplest way to satisfy these criteria is to ush some canonical input through the network. Starting with neurons in the rst layer and progressively moving through all subsequent layers, we set $a = g(b_{in})$, $b = f(a)$, and $d = 0$. In our experiments, we use the rst input frame as the canonical input.

4.3 Transmission Policies

A transmission policy de nes a pair of functions $M(d)$ and $P(d)$ for each layer. M outputs a binary mask m indicating which neurons should transmit. P outputs the values of $_{out}$. In this subsection, we describe the transmission policy design space. The choice of transmission policy is a critical design consideration, strongly in uencing the accuracy and e ciency of the nal model.

Locality and Granularity. Policies may have di erent levels of locality, de ned as the number of elements from d required to compute each element of m and $_{out}$. A global policy considers all elements of d when computing each value m_i and $_{out;i}$. A local policy considers some strict subset of d, and an isolated policy considers only the element d_i.

Fig. 5. Policy Design and Quantization. (a) A few sample response functions (assuming an isolated, singular policy). (b) A comparison between our policy and a rounding policy (used in Sigma-Delta networks [37]). Results are for a 3-layer fully-connected network on the Temporal MNIST dataset [37].

In addition to its locality, each policy has a granularity. The granularity de- nes how m-values are shared between neurons. A chunked policy ties neurons together into local groups, producing one value of m for each group. Neurons in the same group re in unison. This might be practically desirable for easy parallelization on the hardware. In contrast, a singular policy assigns every neuron a separate value of m, so each neuron res independently.

A Linear-Cost Policy. In this work, we use an isolated, singular policy based on a simple threshold. Speci cally,

$$m_i = H(jd_i j \quad h_i); \qquad {}_{out;i} = d_i; \tag{6}$$

where H is the Heaviside step function and h_i is the threshold for neuron i. A key advantage of this policy is its low overhead. On receiving an incoming transmission, a neuron evaluates $jdj > h$ (one subtraction) in addition to the usual updates to a, d, and b. Neurons not receiving any updates (e.g., those in a static image region) do not incur any overhead for policy computations. In other words, the policy's cost is linear in the number of updated neurons. Combined with the linear cost of computing the neuron updates, this results in EvNets whose overall cost scales linearly with the amount of change in the input, not with the quantity of input data received

This linear cost has important implications for networks processing data from high-speed sensors (e.g., event sensors [28] or single-photon sensors [10]). Here, the di erences between adjacent inputs are often minuscule, and the cost of a policy with xed per-frame overhead (e.g., a Gumbel gate [14]) could come to dominate the runtime. EvNets with a linear-overhead policy are a natural solution for processing this type of high-speed data.

Policy Design and Quantization. When a policy is both isolated and singular, we can characterize the functions $M(d)$ and $P(d)$ by scalar functions $M(d_i)$ and $P(d_i)$. Taking the product $M(d_i) \quad P(d_i)$ gives a response function $R(d_i)$ that describes the overall behavior of the neuron. Fig. 5 (a) illustrates several possible response functions.

Some response functions employ quantization to reduce the cost of computing dot product terms $w_i x_i$ (Eq. 1). Sigma-Delta networks [37] use a rounding policy to quantize neuron outputs; a neuron transmits if this rounded value is nonzero. This rounding policy has signi cantly worse accuracy-computation tradeo s (Fig. 5 (b)) compared to our proposed policy. This might be caused by coupling the ring threshold with the quantization scale. To increase its output precision a Sigma-Delta network must reduce its ring threshold, possibly resulting in unnecessary transmissions.

5 Experiments and Results

EvNets are widely applicable across architectures and video inference tasks. Any network satisfying a few basic requirements (i.e., frame-based and composing linear functions with nonlinear activations) can be converted to an EvNet without re-training . To demonstrate this, we select widespread, representative models for our main experiments: YOLOv3 [41] for video object detection and OpenPose [5] for video pose estimation. Additionally, we conduct ablation experiments and report results on low-level tasks (optical ow and image enhancement).

In the supplement, we include additional results on HRNet [47] for pose estimation. We also include ablations for the e ect of layer depth on computation cost, variations in computation over time, the e ect of granularity on savings, improved temporal smoothness, and a comparison to simple interpolation.

5.1 Video Pose Estimation

Dataset and Experiment Setup. We conduct experiments on the JHMDB dataset [24] using the widely adopted OpenPose model [5]. We use weights pretrained on the MPII dataset [2] from [5] and evaluate the models on a subset of JHMDB with 319 videos and over 11k frames, following [14]. We report results on the combination of the three JHMDB test splits. We use the PCK metric [58] with a detection threshold of $= 0{:}2$, consistent with prior works [14].

Implementation Details. We resize all videos to 320 240, padding as needed to preserve the aspect ratio of the content. The joint de nitions in MPII (the training dataset for OpenPose) di er slightly from those in JHMDB. During evaluation, we match the JHMDB \neck," \belly," and \face" joints to the MPII \upper neck," \pelvis," and \head top" joints, respectively.

Baselines. We consider the following baselines, all using the OpenPose model.

{ Conventional : This is the vanilla OpenPose model without modi cations.

{ Skip-Conv : This is a variant of the Skip-Conv method with norm gates and without periodic state resets.

{ Skip-Conv-8 : This adds state resets to Skip-Conv by re- ushing every 8 frames to reduce the e ect of long-term activation drift.

We recognize that Skip-Conv networks can also incorporate a learnable gating function (the Gumbel gate) that uses information from a local window around each neuron. This can also be used for our EvNets (it is local and chunked

Fig. 6. Pareto Curves. The performance of an EvNet over several di erent thresholds, with baselines for comparison. The \Skip-Conv-8" model re- ushes the network every 8 frames. EvNets give signi cant computation savings without sacri cing accuracy. See the supplementary material for a table with this data.

rather than isolated and singular), but it requires re-training of the network and can incur a higher computational overhead. To keep the analysis fair, we only compare to the Skip-Conv norm gate.

Results. Fig. 6 (a) presents our results. We vary the policy threshold h to characterize the accuracy/computation Pareto frontier. For both Skip-Conv and EvNets, increasing the threshold reduces the computational cost but increases the error rate. EvNets consistently outperform their direct competitors (Skip-Conv and Skip-Conv reset) on the Pareto frontier, achieving signi cantly higher accuracy when using a similar amount of computation. Surprisingly, compared to the conventional OpenPose model, EvNets sometimes have slightly better accuracy, even with a large reduction in computation. We hypothesize that this is caused by a weak inter-frame ensembling e ect.

Table 1 summarizes the accuracy and computation at the best operating point on the Pareto curve. For each model, we choose the highest threshold that reduces PCK by less than 0:5 %. To better understand the accuracy-computation tradeo , we further report the compute and memory overhead of our EvNets (at the best operating point) in Table 2. We report overhead operations both as a number of operations and as a percentage. This percentage gives the ratio between \extra operations expended" and \number of arithmetic operations saved." For example, an arithmetic overhead of 0:12 % indicates that the neuron updates and transmission policy require 0:12 extra arithmetic operations for every 100 operations saved. Overall, EvNets add minimal operation overhead and manageable additional memory.

5.2 Video Object Detection

Dataset, Experiment Setup, and Baselines. We evaluate on the ILSVRC 2015 VID dataset [42] using the popular YOLOv3 model [41] with pre-trained weights from [43]. We report all results on the validation set with 555 videos and over 172k frames, using mean Average Precision (mAP) with an IoU threshold of 0:5 (following previous works [6,43,61]). We evaluate the same model variants as in Sec. 5.1 (conventional, EvNet, Skip-Conv, and Skip-Conv reset).

Table 1. Accuracy and Computation. Results on the best threshold for each model. We choose the highest threshold that reduces PCK or mAP by less than 0:5 %. See the supplement for a complete list of the points shown in Fig. 6.

	Pose Estimation			Object Detection		
Model	Thresh.	PCK (%)	Operations	Thresh.	mAP (%)	Operations
Conventional	{	76:40	7:055 10^{10}	{	55:38	1:537 10^{10}
Skip-Conv	0:01	76:03	1:027 10^{10}	0:01	54:13	7:340 10^{9}
Skip-Conv-8	0:01	76:21	1:092 10^{10}	0:01	54:06	8:111 10^{9}
EvNet	0:04	76:37	6:780 10^{9}	0:08	56:19	3:061 10^{9}

Table 2. Overhead. \Weights" gives the amount of memory required for model weights. \Variables" gives the amount of memory required for the state variables a, b, and d. \Arithmetic" indicates the number of extra arithmetic operations expended for neuron updates (Eq. 3) and policy-related computations. \Load and Store" indicates the number of extra memory access operations. See the text for an explanation of the percentage notation.

		Memory Costs		Operation Overhead	
Model	Thresh.	Weights	Variables	Arithmetic	Load and Store
OpenPose	0:04	206:8 MB	346:2 MB	7:570 10^{7} (0:12 %)	1:342 10^{8} (0:21 %)
YOLO	0:08	248:8 MB	232:2 MB	6:417 10^{7} (0:52 %)	1:040 10^{8} (0:85 %)

Implementation Details. We resize all videos to 224 384, padding as needed to preserve the aspect ratio. Unlike OpenPose, YOLOv3 includes batch normalization (BN) layers. BN gives us a convenient way to estimate the distribution of activations at each neuron. We use this information to adjust the threshold values. Speci cally, we scale the threshold at each neuron by =1 (where is the learned BN re-scaling parameter). This scaling makes the policy more sensitive for neurons with a narrower activation distribution, where we would expect equal-sized changes to be more meaningful.

Results. Fig. 6 presents our results with varying thresholds. Again, we observe that our EvNets outperform Skip-Conv variants, and sometimes have slightly higher accuracy than the conventional model with greatly reduced compute cost. Table 1 presents the accuracy and computation at the best operating points.

5.3 Low-Level Vision Tasks

We have so far considered only high-level inference tasks. However, EvNets are also an e ective strategy for low-level vision. We consider PWC-Net [46] for optical ow computation and HDRNet [12] for video frame enhancement. For brevity, we only show sample results in Fig. 7 and refer the reader to the supplementary material for more details. As with the high-level models, we observe minimal degradation in accuracy and signi cant computation savings.

Fig. 7. Versatility of EvNets. We demonstrate that EvNets are an effective strategy for many high- and low-level vision tasks. Across tasks, we see significant computation savings while maintaining high-quality output. This frame shows a person mid-jump. The EvNet tracks the subject correctly under rapid motion.

5.4 Ablation and Analysis

Rounding Policy Comparison. Fig. 5 (b) compares our transmission policy and the rounding policy used in a Sigma-Delta network [37]. We obtain these results by evaluating the fully-connected model from the Sigma-Delta paper (with the authors' original weights) on the Temporal MNIST dataset [37]. We evaluate EvNets with thresholds of the form 10^p, where $p \in \{-1.5, -1.4, \ldots, -0.3, -0.2\}$. We obtain results for the Sigma-Delta network using the original authors' code, which involves training the quantization threshold (the Pareto frontier is a consequence of varying a training penalty scale).

Ablation of Long-Term Memory. Fig. 8 shows the effect of ablating the long-term memory d (resetting it to zero after each input). We evaluate the OpenPose model on the JHMDB dataset. Other than resetting d, the two models shown are identical. Both models use a threshold of 0.05. We see that long-term memory is critical for maintaining stable accuracy.

Camera Motion. Global camera or scene motion (e.g., camera shake or scene translation) reduces the amount of visual persistence in a video. We would therefore expect camera motion to reduce the savings in an EvNet. To confirm this, we evaluate the OpenPose and YOLO models on a custom-labeled video dataset. We label the camera motion in each video as "none" (perfectly stationary camera), "minor" (slight camera shake), or "major." See the supplement for details. We test OpenPose with a threshold of 0.05 and YOLO with a threshold of 0.06. Because this dataset does not have frame-level labels for pose or object detection, we do not explicitly evaluate task accuracy. However, the thresholds we use here give good accuracy on JHMDB and VID. For OpenPose, the computation savings for "none," "major," and "minor" camera motion are $17.3\times$, $11.3\times$, and $8.40\times$, respectively. For YOLO, the savings are $6.64\times$, $3.95\times$, and $2.65\times$. As expected, we see a reduction in savings when there is strong camera motion, although we still achieve large reductions relative to the conventional model.

Fig. 8. Ablation of Long-Term Memory. Removing the memory d, as considered in Skip-Conv [14], causes a rapid decay in accuracy. Results using the OpenPose model on the JHMDB dataset. See Sec. 5 for details.

Wall-Time Savings. We now show preliminary results demonstrating wall-time savings in EvNets. We consider the HRNet [47] model (see supplementary) on the JHDMB dataset. We evaluate on an Intel Core i7 8700K CPU.

We implement the model in PyTorch. For the EvNet, we replace the standard convolution with a custom sparse C++ convolution. Our convolution uses an input-stationary design (i.e., an outer loop over input pixels) to skip zero deltas e ciently. In the conventional model, we use a custom C++ convolution with a standard output-stationary design (i.e., an outer loop over output pixels). We use a custom operator in the conventional model to ensure a fair comparison, given the substantial engineering e ort invested in the default MKL-DNN library. We implement both operators with standard best practices (e.g., maximizing data-access locality). We compile with GCC 9.4 with the -Ofast ag.

For evaluation, we use an input size of 256 256 and an EvNet threshold of 0:1. The EvNet achieves a PCK of 90:46 % and runs in an average of 0:3497 s (7:361 10^8 ops) per frame. The conventional model achieves a PCK of 90:37 % and runs in 1:952 s (1:019 10^{10} ops) per frame.

6 Discussion

Hardware Platforms. Mainstream GPU hardware is designed for parallel, block-wise computation with coarse control ow. EvNets with neuron-level transmission are ine cient under this computation model. In the long term, we expect to achieve the best performance on specialized hardware designed for extreme parallelism and distributed control. It is important to emphasize that event neurons do not need to operate by a shared clock. Each neuron operates independently { consuming new input as it arrives and transmitting output once it is computed. This independence permits an asynchronous, networked execution model in contrast to the ordered, frame-based model in conventional machine learning. Spiking neural networks (SNNs) [31] share this asynchronous computation model and have motivated the development of neuromorphic hardware platforms [1,9] that could be re-purposed for e cient implementation of EvNets.

Acknowledgments. This research was supported by NSF CAREER Award 1943149.

References

1. Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., Imam, N., Nakamura, Y., Datta, P., Nam, G.J., Taba, B., Beakes, M., Brezzo, B., Kuang, J.B., Manohar, R., Risk, W.P., Jackson, B., Modha, D.S.: TrueNorth: Design and tool ow of a 65 mW 1 million neuron programmable neurosynaptic chip. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34(10), 1537{1557 (Oct 2015). https://doi.org/10.1109/TCAD.2015.2474396
2. Andriluka, M., Pishchulin, L., Gehler, P., Schiele, B.: 2D human pose estimation: New benchmark and state of the art analysis. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3686{3693 (2014)
3. Campos, V., Jou, B., Giro-i Nieto, X., Torres, J., Chang, S.F.: Skip RNN: Learning to skip state updates in recurrent neural networks. In: International Conference on Learning Representations (2018)
4. Cannici, M., Ciccone, M., Romanoni, A., Matteucci, M.: Asynchronous convolutional networks for object detection in neuromorphic cameras. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). pp. 1656{1665 (Jun 2019). https://doi.org/10.1109/CVPRW.2019.00209
5. Cao, Z., Simon, T., Wei, S.E., Sheikh, Y.: Realtime multi-person 2D pose estimation using part a nity elds. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7291{7299 (2017)
6. Chen, Y., Cao, Y., Hu, H., Wang, L.: Memory enhanced global-local aggregation for video object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10337{10346 (2020)
7. Chin, T.W., Ding, R., Marculescu, D.: AdaScale: Towards real-time video object detection using adaptive scaling. arXiv:1902.02910 [cs] (Feb 2019)
8. Courbariaux, M., Bengio, Y., David, J.P.: BinaryConnect: Training deep neural networks with binary weights during propagations. Advances in Neural Information Processing Systems 28, 3123{3131 (2015)
9. Davies, M., Srinivasa, N., Lin, T.H., Chinya, G., Cao, Y., Choday, S.H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C.K., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., Weng, Y.H., Wild, A., Yang, Y., Wang, H.: Loihi: A neuromorphic many-core processor with on-chip learning. IEEE Micro 38(1), 82{99 (Jan 2018). https://doi.org/10.1109/MM.2018.112130359
10. Dutton, N.A.W., Gyongy, I., Parmesan, L., Gnecchi, S., Calder, N., Rae, B.R., Pellegrini, S., Grant, L.A., Henderson, R.K.: A SPAD-based QVGA image sensor for single-photon counting and quanta imaging. IEEE Transactions on Electron Devices 63(1), 189{196 (Jan 2016). https://doi.org/10.1109/TED.2015.2464682
11. Feichtenhofer, C., Fan, H., Malik, J., He, K.: SlowFast networks for video recognition. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 6202{6211 (2019)
12. Gharbi, M., Chen, J., Barron, J.T., Hasino , S.W., Durand, F.: Deep bilateral learning for real-time image enhancement. ACM Transactions on Graphics 36(4), 118:1{118:12 (Jul 2017). https://doi.org/10.1145/3072959.3073592
13. Ghodrati, A., Bejnordi, B.E., Habibian, A.: Frameexit: Conditional early exiting for e cient video recognition. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 15608{15618 (2021)
14. Habibian, A., Abati, D., Cohen, T.S., Bejnordi, B.E.: Skip-convolutions for e cient video processing. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2695{2704 (2021)

15. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. Advances in Neural Information Processing Systems **28**, 1135–1143 (2015)
16. Hassibi, B., Stork, D.: Second order derivatives for network pruning: Optimal brain surgeon. Advances in Neural Information Processing Systems **5**, 164–171 (1992)
17. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
18. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861 [cs] (Apr 2017)
19. Huang, G., Chen, D., Li, T., Wu, F., Van Der Maaten, L., Weinberger, K.Q.: Multi-scale dense networks for resource efficient image classification. In: International Conference on Learning Representations (2018)
20. Huang, G., Sun, Y., Liu, Z., Sedra, D., Weinberger, K.Q.: Deep networks with stochastic depth. In: European conference on computer vision. pp. 646–661. Springer (2016)
21. Hwang, K., Sung, W.: Fixed-point feedforward deep neural network design using weights +1, 0, and -1. In: Workshop on Signal Processing Systems (SiPS). pp. 1–6 (Oct 2014). https://doi.org/10.1109/SiPS.2014.6986082
22. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. arXiv:1602.07360 [cs] (Nov 2016)
23. Jain, S., Wang, X., Gonzalez, J.E.: Accel: A corrective fusion network for efficient semantic segmentation on video. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8866–8875 (2019)
24. Jhuang, H., Gall, J., Zuffi, S., Schmid, C., Black, M.J.: Towards understanding action recognition. In: International Conference on Computer Vision (ICCV). pp. 3192–3199 (Dec 2013)
25. LeCun, Y., Denker, J., Solla, S.: Optimal brain damage. Advances in Neural Information Processing Systems **2**, 598–605 (1989)
26. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient ConvNets. arXiv:1608.08710 [cs] (Mar 2017)
27. Li, Y., Shi, J., Lin, D.: Low-latency video semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5997–6005 (2018)
28. Lichtsteiner, P., Posch, C., Delbruck, T.: A 128×128 120 dB 15 Ms latency asynchronous temporal contrast vision sensor. IEEE Journal of Solid-State Circuits **43**(2), 566–576 (Feb 2008). https://doi.org/10.1109/JSSC.2007.914337
29. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollar, P.: Focal loss for dense object detection. In: International Conference on Computer Vision (ICCV). pp. 2980–2988 (2017)
30. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: SSD: Single shot MultiBox detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) European Conference on Computer Vision (ECCV). pp. 21–37. Lecture Notes in Computer Science, Springer International Publishing (2016). https://doi.org/10.1007/978-3-319-46448-0_2
31. Maass, W.: Networks of spiking neurons: The third generation of neural network models. Neural Networks **10**(9), 1659–1671 (Dec 1997). https://doi.org/10.1016/S0893-6080(97)00011-7

32. Meng, Y., Lin, C.C., Panda, R., Sattigeri, P., Karlinsky, L., Oliva, A., Saenko, K., Feris, R.: AR-Net: Adaptive frame resolution for efficient action recognition. In: European Conference on Computer Vision. pp. 86–104. Springer (2020)
33. Meng, Y., Panda, R., Lin, C.C., Sattigeri, P., Karlinsky, L., Saenko, K., Oliva, A., Feris, R.: Adafuse: Adaptive temporal fusion network for efficient action recognition. In: International Conference on Learning Representations (2021)
34. Messikommer, N., Gehrig, D., Loquercio, A., Scaramuzza, D.: Event-based asynchronous sparse convolutional networks. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.M. (eds.) European Conference on Computer Vision (ECCV). pp. 415–431. Lecture Notes in Computer Science, Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-58598-3_25
35. Neil, D., Lee, J.H., Delbruck, T., Liu, S.C.: Delta networks for optimized recurrent network computation. In: Proceedings of the 34th International Conference on Machine Learning. pp. 2584–2593. PMLR (Jul 2017)
36. Nie, X., Li, Y., Luo, L., Zhang, N., Feng, J.: Dynamic kernel distillation for efficient pose estimation in videos. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 6942–6950 (2019)
37. O'Connor, P., Welling, M.: Sigma delta quantized networks. arXiv:1611.02024 [cs] (Nov 2016)
38. Pan, B., Lin, W., Fang, X., Huang, C., Zhou, B., Lu, C.: Recurrent residual module for fast inference in videos. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1536–1545 (2018)
39. Parger, M., Tang, C., Twigg, C.D., Keskin, C., Wang, R., Steinberger, M.: DeltaCNN: End-to-end CNN inference of sparse frame differences in videos. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 12497–12506 (2022)
40. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: ImageNet classification using binary convolutional neural networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) European Conference on Computer Vision (ECCV). pp. 525–542. Lecture Notes in Computer Science, Springer International Publishing (2016). https://doi.org/10.1007/978-3-319-46493-0_32
41. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Conference on Computer Vision and Pattern Recognition (CVPR). pp. 779–788 (2016)
42. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet large scale visual recognition challenge. International Journal of Computer Vision (IJCV) **115**(3), 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y
43. Sabater, A., Montesano, L., Murillo, A.C.: Robust and efficient post-processing for video object detection. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 10536–10542 (Oct 2020). https://doi.org/10.1109/IROS45743.2020.9341600
44. Shelhamer, E., Rakelly, K., Hoffman, J., Darrell, T.: Clockwork convnets for video semantic segmentation. In: Hua, G., Jégou, H. (eds.) Computer Vision – ECCV 2016 Workshops. vol. 9915, pp. 852–868. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-49409-8_69
45. Sillerkiil: Eesti: Aegviidu siniallikad (Aegviidu blue springs in Estonia) (Apr 2021)
46. Sun, D., Yang, X., Liu, M.Y., Kautz, J.: PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 8934–8943 (2018)