Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

7 μJ/inference end-to-end gesture recognition from dynamic vision sensor data using ternarized hybrid convolutional neural networks

Georg Rutishauser [a,*], Moritz Scherer [a], Tim Fischer [a], Luca Benini [a,b]

[a] ETH Zürich, Gloriastrasse 35, Zürich, 8092, Switzerland
[b] Università di Bologna, Via Zamboni 33, Bologna, 40126, Italy

ARTICLE INFO

Article history:
Received 8 December 2022
Received in revised form 23 May 2023
Accepted 14 July 2023
Available online 29 July 2023

Keywords:
Dynamic vision sensors
Gesture recognition
Ternary neural networks
Edge computing
Low-power systems

ABSTRACT

Dynamic vision sensor (DVS) cameras enable energy-activity proportional visual sensing by only propagating events produced by changes in the observed scene. Furthermore, by generating these events asynchronously, they offer μs-scale latency while eliminating the redundant data transmission inherent to classical, frame-based cameras. However, the potential of DVS to improve the energy efficiency of IoT sensor nodes can only be fully realized with efficient and flexible systems that tightly integrate sensing, processing, and actuation capabilities. In this paper, we propose a complete end-to-end pipeline for DVS event data classification implemented on the Kraken parallel ultra-low power (PULP) system-on-chip and apply it to gesture recognition. A dedicated on-chip peripheral interface for DVS cameras aggregates the received events into ternary *event frames*. We process these video frames with a fully ternarized two-stage temporal convolutional network (TCN). The neural network can be executed either on Kraken's PULP cluster of general-purpose RISC-V cores or on CUTIE, the on-chip ternary neural network accelerator. We perform extensive ablations on network structure, training, and data generation parameters. We achieve a validation accuracy of 97.7 % on the DVS128 11-class gesture dataset, a new record for embedded implementations. With in-silicon power and energy measurements, we demonstrate a classification energy of 7 μJ at a latency of 0.9 ms when running the TCN on CUTIE, a reduction of inference energy by 67× when compared to the state of the art in embedded gesture recognition. The processing system consumes as little as 4.7 mW in continuous inference, enabling always-on gesture recognition and closing the gap between the efficiency potential of DVS cameras and application scenarios.

1. Introduction

The exponential proliferation of small-scale sensing systems such as Internet of Things (IoT) sensor nodes and wearable devices has coincided with a revolution in data processing, where machine learning (ML) algorithms such as CNNs and transformers have dramatically improved the state of the art in fields such as computer vision, natural language processing or biomedical signal processing [1–4]. As many applications involving the collection of data demand for their interpretation and reaction to the result, these developments are naturally symbiotic. However, the approach of transmitting raw data to cloud servers to be processed with powerful and compute-intensive ML model does not scale to the enormous amounts of data collected and conflicts with the constraints of battery-powered, sensor-driven applications requiring real-time performance: wireless communication is power-intensive and incurs communication latency, and the transmission of raw sensor data raises privacy and security concerns.

This conflict is addressed by the emerging research field of edge AI, which aims to enable efficient and accurate ML-based processing of sensor data on edge sensor nodes. These systems, typically built around Microcontrollers (MCUs), operate under stringent power, memory, and compute resource constraints. A key algorithmic technique in edge AI is *quantization* of neural networks, where parameters and intermediate activations are represented in low-bitwidth data formats, reducing a model's memory and storage footprints. With hardware support for low-precision arithmetic through specialized extensions to general-purpose instruction set architectures (ISAs) [5] or dedicated accelerators, quantization also increases computational throughput and efficiency, leading to lower energy consumption per inference.

However, to optimize a system's power consumption and performance, the entire pipeline from the sensor to the reaction to

* Corresponding author.
E-mail addresses: georgr@iis.ee.ethz.ch (G. Rutishauser), scheremo@iis.ee.ethz.ch (M. Scherer), fischeti@iis.ee.ethz.ch (T. Fischer), lbenini@iis.ee.ethz.ch (L. Benini).

https://doi.org/10.1016/j.future.2023.07.017

the results of processing must be considered. An efficient end-to-end solution consists of a low-power sensor that minimizes redundant data generation, a tightly integrated sensor-processor interface to avoid communication overhead (e.g., from dedicated logic circuits for sensor readout) and an efficiently implemented processing algorithm executed with low latency.

In the context of visual sensing, DVS cameras minimize sensing and communication energy by only capturing and outputting events describing the location and polarity of pixel-wise brightness changes exceeding a certain threshold. Accordingly, DVS communicate at a data rate proportional to the activity in the captured scene, with a corresponding reduction in communication energy when operating in static or low-activity environments.

Spiking neural networks (SNN), a class of brain-inspired neural networks, operate directly on a stream of input events and extend the principle of energy-activity proportionality to the processing domain. As such, they can be viewed as the natural processing paradigm for event-based vision systems and the combination of the two principles has attracted significant research. However, SNNs are still an emerging technology: they lag behind classical ML models such as CNNs both in task accuracy and in hardware support, and have not yet proven their suitability for low-power applications in the edge computing domain [6,7]. Aggregating the DVS event stream into video frames provides an alternative to SNN-based processing and allows the use of more established, highly optimized machine learning models such as aggressively quantized neural networks (QNNs) [8–10].

In this paper, we follow the latter approach, extending our work from [10]. We present a frame-based end-to-end processing pipeline for DVS event data implemented on the RISC-V-based Kraken SoC. Thanks to a dedicated on-chip DVS camera interface implementing configurable event frame aggregation, data transfer, and processing overheads are minimized. By classifying the accumulated frames with a 2-stage TNN, either on the integrated CUTIE accelerator [11] or on a PULP cluster of 8 RISC-V cores, we show that frame-based processing of DVS event data has the potential to enable ultra-low-power gesture recognition in real-time on edge computing nodes. While the CUTIE accelerator offers maximal efficiency at very low latencies, it is a large design with a silicon footprint of 3 mm^2 in a 22 nm process and imposes restriction on the supported network architectures, such as the restriction to TNNs, the number of channels per layer or the maximum kernel size. The fully software-programmable PULP cluster has no such restrictions and can efficiently perform a wide range of compute tasks. An efficient implementation of our network on the cluster is thus an attractive option for area-constrained systems requiring maximal flexibility and a point of reference for applications requiring network architectures not compatible with CUTIE. While [10] reported power results based on post-synthesis simulation of the CUTIE accelerator and DVS peripheral for a single network parametrization, this work presents detailed algorithmic explorations along with power measurements of end-to-end implementations of the processing pipeline on the Kraken SoC.

Our contributions with this work are the following:

- We present a detailed exploration of the impact of architectural parameters and training algorithms on the performance of ternarized hybrid TCNs for gesture recognition from DVS event frames.
- Motivated by ReLU activations' superior performance, we present a procedure to convert networks using 3-level unsigned (ReLU) activations to networks with signed ternary activations commonly supported by TNN accelerators.
- We evaluate the effect of the different parameters in generating event frames from raw DVS camera output on the network's statistical accuracy and latency, achieving state-of-the-art classification accuracy of up to 97.9% on the DVS128 gesture dataset.
- We map the proposed network architecture on both the CUTIE TNN accelerator [11] and a PULP cluster of 8 RISC-V cores with native hardware support for low-precision arithmetic. We report in-silicon measurements of inference energy for both targets, demonstrating continuous end-to-end inference at a power consumption of 4.7 mW and an inference energy of 7 μJ at a latency of 0.9 ms on CUTIE. On the PULP cluster, continuous inference consumes 6.4 mW with a classification energy of 0.41 mJ at a latency of 17.8 ms. The energy per CUTIE inference represents an improvement on the state of the art for embedded end-to-end gesture recognition by a factor of 67×. The energy per inference for cluster-mapped networks is still competitive with the state of the art, demonstrating the viability of software-based processing of DVS events within a sub-10 mW power budget.

2. Related work

As this paper presents a pipeline for power-efficient end-to-end gesture recognition from DVS data, we first focus our review of the related literature on works that perform gesture recognition on embedded, low-power devices with various types of sensors before giving a more detailed overview of works that employ event cameras as the primary sensor. The interested reader may refer to [12] for a wider perspective on deep learning-based techniques for device-free wireless sensing tasks such as person localization, gesture recognition, or fall detection.

2.1. Embedded gesture recognition

Gesture recognition based on electromyography (EMG) has attracted significant research interest. A common approach is to attach an EMG electrode array to the forearm and classify the signals using a deep neural network (DNN). [13] combines a custom analog frontend for 8-channel EMG signal acquisition and analog-to-digital conversion with an ARM Cortex-M4 microcontroller running a support vector machine (SVM) to classify the input. The complete system achieves an average classification accuracy of 89.7% on four user-specific datasets of 7 gestures in a power envelope of 29.7 mW. [14] proposes a larger system based on a 32-channel wireless sensing armband and CNN-based classification on the NVidia Jetson Nano platform, achieving 98.5% classification accuracy on eight gesture classes collected from a single user with 5 ms of inference latency at a total power consumption of 3.1 W. At tens-of-mW power envelopes, other EMG-based embedded gesture recognition systems [15–17] achieve comparable results to [13]. The drawback of EMG-based gesture recognition is that it is not suitable for ad-hoc human–machine interaction for two reasons. First, the models are commonly user-specific, requiring retraining for different users. Second, placing EMG electrodes requires careful setup and preparation. While systems supporting on-device learning based on hyperdimensional computing [17] have addressed the former issue, the latter is inherent to EMG interfaces.

A different sensing approach is to use miniaturized RADAR sensors. Google pioneered this technique with SOLI [18], using custom RADAR application-specific integrated circuits (ASICs) in combination with off-the-shelf (embedded and desktop-scale) processing platforms to achieve up to 92.1% classification accuracy on a four-class hand gesture dataset collected from five users with a random forest-based classifier. [19] uses SOLI hardware

Table 1
Comparison of the results with state of the art embedded gesture recognition implementations.

	[21]	[29]		[28]	[30]	**This work**	
Sensor	SRR	DVS128 [31]/sEMG (Myo)		DVS128	DVS128	DVS132S [32][a]	
P_{sensor}	95 mW	>23 mW[b]		23 mW	23 mW	250 μW	
Dataset	Custom	Custom		DVS128	DVS128	DVS128	
Model	CNN/transformer	SNN		SNN	SNN	Ternarized CNN/TCN	
End-to-end?	✓	✗		✓	✗	✓	
Processor	GAP8	ODIN + MorphIC	Loihi	TrueNorth	Loihi	CUTIE	PULP Cluster
P_{idle}	–	–	29 mW [33]	134.4 mW	29 mW [33]	4.7 mW	3.6 mW
P_{cont}	>7.1 mW[c]	–	33.2 mW[d]	178.8 mW	–	**4.7** mW	6.4 mW
t_{inf}	9 ms	19.5 ms	7.8 ms	–	11 ms	**0.9** ms	17.8 ms
E_{inf}	0.47 mJ	37 μJ[e]	1.1 mJ[e]	18.8 mJ	–	**7** μJ	0.41 mJ
Accuracy	77.15%	89.4%	96.0%	94.6%	90.5%	**97.7%**	**97.7%**

[a]Network accuracy was evaluated on the DVS128 gesture dataset collected with a different sensor.
[b]Power consumption of the Myo armband is not specified.
[c]We assume 15 inf/s and zero idle power consumption as a lower bound for continuous power consumption.
[d]Calculated from idle power, inference energy and inference latency.
[e]Only dynamic energy figures reported.

to collect an 11-class hand gesture dataset, achieving 94.2% classification accuracy with a CNN-based classifier run on desktop hardware. The RADAR approach was subsequently refined and miniaturized in [20], using embedded short-range RADAR sensors produced by Acconeer and performing classification with a hybrid TCN on the RISC-V PULP-based GAP8 processor. The authors report 81.5% classification accuracy at a continuous inference power of 21 mW and a classification energy of 4.52 mJ, bringing RADAR-based gesture recognition into the edge computing space. [21] improves on this work by using a transformer-based classifier, reducing latency and inference energy to 9 ms and 0.47 mJ, respectively. While this shows that RADAR data processing can be performed under strict power constraints, RADAR sensors are generally power-hungry: Google's SOLI RADAR consumes 300 mW per chip and the sensors used in [20] use 90 mW each. This means that sensing power dominates full-system power consumption and makes RADAR-based gesture recognition unsuitable for ultra-low-power applications requiring power envelopes of <10 mW.

Other sensing techniques used for gesture recognition include sensor gloves [22,23], ultrasonic echolocation [24,25] and camera-based visual sensing [26,27]. However, these approaches have not found any application to ultra-low-power embedded systems so far.

2.2. DVS gesture recognition

In recent years, gesture recognition from DVS event data has seen considerable research interest. The most common approach is to process the event stream using SNNs. In [28], the authors introduced the 11-class DVS128 full-body gesture dataset, which has become the de-facto standard for DVS gesture recognition. They implement an end-to-end gesture recognition system on the NS1e development board based on IBM's TrueNorth neuromorphic processor. The event stream is preprocessed with a cascade of temporal filters and classified with a 16-layer convolutional SNN at 1 ms timesteps. This pipeline achieves a classification accuracy of 91.8%, The application of a sliding-window filter to the output further increases accuracy to 94.6%. TrueNorth's power consumption is measured at 178.8 mW, making this system unsuitable for edge computing applications.

Similarly, [30] proposes an SNN converted from a 5-layer CNN to classify the event stream from the DVS128 dataset. In notable contrast to [28], events are accumulated over much longer timeframes (300 ms for the best-performing network) which are divided into multiple input channels. This processing approach is equivalent to feeding event frames to a SNN. Classification accuracy is reported as 90.5% at a processing latency of 15 ms. While the authors do not provide information on power consumption, [33] reports a single Loihi chip's idle power consumption as 29 mW, providing a lower bound. It should also be noted that the acquisition and preprocessing of the event stream are not performed on Loihi.

[29] performs sensor fusion on DVS and EMG data, using dual-branched SNN models with a joint classifier layer to perform hand gesture recognition on a custom 5-class dataset. The model mapped to Loihi, consisting of a spiking CNN handling DVS data and a spiking multi-layer perceptron (MLP) to process EMG data, achieves a classification accuracy of 96% at an inference latency of 7.8 ms. The authors also map a model consisting of two spiking MLPs to a pair of research SNN accelerators, ODIN [34] and MorphIC [35]. Like the Loihi model, this smaller model operates on a 200 ms window of input data and achieves 89% classification accuracy at a latency of 19.5 ms. The authors report only the dynamic energy consumption of both networks, which is measured at 1.1 mJ for Loihi and calculated (using detailed power models) at 37 μJ for ODIN+MorphIC. As with [30], the preprocessing step is not accounted for in these figures and is performed offline.

Other works are purely algorithmic and do not include power or energy figures on embedded platforms. [36] proposes a spiking convolutional recurrent neural network, achieving 90.3% validation accuracy on the DVS128 dataset. [37–39] propose different methods of training SNNs, achieving validation accuracies of up to 97.6%. [9] proposes a method to aggregate events into decimal pixel values to encode time information, processing these event frames with a large 3D CNN. This approach achieves a near-perfect validation accuracy of 99.6% on the DVS128 dataset, the highest result reported in the literature.

In conclusion, our literature study reveals that accessible end-to-end gesture recognition on extreme-edge devices is an unsolved problem: existing edge solutions use sensing technologies that are either impractical to use (e.g., EMG) or too power-hungry to fit the constraints of the IoT. DVS cameras have the potential to close this gap but have not yet been used to their full potential in ultra-low-power sensor nodes. We propose to close this gap with an event frame-based processing pipeline based on a hybrid TNN mapped to the Kraken SoC, which integrates the sensor interface with efficient processing units (namely, the CUTIE accelerator and 8-core PULP cluster) to perform both inference and general-purpose processing tasks. To the best of our knowledge, our system outperforms all embedded solutions in both classification accuracy (97.7% on DVS128) and efficiency (7 μJ/inf., 4.7 mW continuous inference power) while providing the versatility of a fully-featured MCU, enabling always-on gesture detection on battery-powered sensor nodes at the extreme edge. Table 1 compares our system to the state of the art in embedded gesture recognition systems.

3. Background

In this section, we give a brief overview of the operating principle of DVS cameras, their advantages and drawbacks compared to conventional cameras, and the approaches adopted to process DVS data. We then provide background on the training and inference of QNN and a short description of the architecture of the CUTIE TNN accelerator.

3.1. DVS cameras and processing DVS data

DVS, first proposed in 1991 [40], are an emerging class of visual sensors that detect and transmit information on brightness changes in the captured scene. In contrast to conventional cameras, which produce a stream of *frames* at fixed time intervals, DVS cameras emit a stream of *events* describing the location and polarity of an individual pixel's change in brightness. An event can be described as a tuple (t, x, y, p), where t denotes the time at which the event is produced, (x, y) are the coordinates of the pixel which produced the event and the polarity $p \in \{-1, 1\}$ indicates whether brightness at the respective pixel increased or decreased by a specific threshold since the last event emitted.

An advantage of DVS cameras over conventional image sensors is that they enable *energy-proportional sensing*: As only localized brightness changes are sensed and transmitted, the pixel array's activity and transmission data rate (and, with them, the respective power consumptions) are proportional to the activity level in the captured scene. In contrast, conventional cameras capture and transmit frames at a constant data rate, leading to constant and activity-independent power consumption. A second advantage lies in their low latency (typically $<200\,\mu$s) due to their asynchronous nature: Each pixel in a DVS array operates independently, emitting information about a change in the captured scene as soon as it occurs. Sensing latency is thus limited only by the readout circuitry and the physical interface used to transmit the event stream. DVS pixels also typically have a high maximum event rate per pixel of >300 e/s, making DVS cameras suitable for high-speed applications. Due to the logarithmic brightness response of DVS pixels, DVS cameras also support high dynamic ranges of >120 dB [31,41,42], enabling their application in a wide range of lighting conditions.

Compared to shutter-based cameras, DVS cameras have some limitations. The first is a lack of color representation; while there are some research works exploring this [43], commercially available DVS cameras are exclusively monochrome. Availability and price are further obstacles to widespread adoption of DVS technology; while camera modules for embedded applications are mass-market products, DVS camera availability is very limited and prices range in the equivalent of thousands of dollars.

The processing of DVS data presents a significant challenge, and [8] divides approaches into two broad categories. The first class aggregates event data into frame-like groups. An example of this is time surfaces, where the value of each pixel describes how recently the last event at that pixel occurred. The most basic approach in this class is that of *event frames*, which we also adopt. Event frames are constructed by dividing the event stream into fixed time intervals, aggregating the events occurring during each frame interval into pixel values for each spatial location in the frame [9,10,44]. These approaches enable processing with conventional image processing algorithms such as DNNs, but omit some of the information in the event stream. E.g., in periodically sampled time surfaces, there is no information about how many events on a given pixel occurred during the frame interval.

The second category of processing algorithms for DVS data operates on individual events. This category's most prevalent algorithmic paradigm is that of SNNs. A class of artificial neural networks (ANNs) inspired by the working principle of the human brain, SNNs, consist of one or multiple layers of neurons that behave according to a biologically-motivated model. In the most commonly utilized model, the leaky integrate-and-fire (LIF) neuron, each neuron in the network is connected to neurons in the previous layer and takes discrete spikes as its input. The input spikes are weighted with parameters specific to their source and added to the neuron's internal membrane potential, which decays exponentially independent of input spiking activity. When a neuron's membrane potential reaches a parametrizable threshold, it fires and propagates a spike to the neurons of the next layer to which it is connected. Upon firing, the membrane potential resets to a rest potential. SNNs are fundamentally asynchronous and time-continuous, but their dynamics can be discretized in time, a fundamental step for their implementation in digital circuits [6,37]. As DVS events can be directly mapped to input spikes, SNNs are a natural fit for processing DVS data and this pairing has accordingly attracted widespread research attention. Examples of their successful application include optical flow estimation [45,46] and image classification [37,47,48] on event datasets converted from traditional datasets consisting of static images [49–51]. Gesture recognition, the application we target, has also been implemented with SNNs, achieving accuracies of up to 94.6% on the DVS128 dataset [28].

However, on all the above tasks, the accuracy of SNN-based methods does not match that of traditional DNNs [6,9,37,52]. Furthermore, the integration of flexible SNN accelerators suitable for executing complex networks into systems capable of end-to-end processing is not well-established. So far, it has been restricted to large-scale designs such as Intel's Loihi family of systems [53,54] or IBM's TrueNorth platform [55].

3.2. Training and inference of quantized neural networks

Low-power DNN inference on edge devices has been the focus of intense research activity. *quantized neural networks (QNNs)* have crystallized as a key technique to maximize inference efficiency on highly resource-constrained platforms [5,56,57]. In a QNN, model weights and intermediate activations are represented as low-bitwidth integers. This offers three key benefits: Quantizing model parameters reduces storage requirements, while quantizing intermediate activations to low precisions reduces the memory footprint for model execution. Finally, both weight and activation quantization enable the use of low-precision multiply-accumulate (MAC) units to perform the model calculations, increasing the throughput per area and reducing the energy per operation. This hardware support can either be implemented as ISA extensions [5] to general-purpose processing cores or through specialized accelerators [58,59].

The vast majority of approaches to neural network quantization are based on uniform quantization. To discretize a full-precision value to n_{lvl} levels, it is clipped to an interval $[a, b]$ and a uniformly spaced staircase function with step width $\varepsilon = \frac{b-a}{n_{lvl}-1}$ is applied. Mapping a tensor $\mathbf{X}$ to its quantized counterpart $\tilde{\mathbf{X}}$ in this manner can be expressed as in Eq. (1).

$$\begin{gathered}\tilde{\mathbf{X}} = Q(\mathbf{X}, a, b, n_{lvl}) = \lfloor \frac{clip(\mathbf{X}, a, b)}{\varepsilon} \rceil \times \varepsilon, \text{ where} \\ clip(\mathbf{X}, a, b) = \min\left(\max\left(\mathbf{X}, a\right), b\right)\end{gathered} \tag{1}$$

The quantizer in Eq. (1) will map values to a discrete grid, but the quantization steps of this grid are still generally real numbers. A model quantized in this way is commonly referred to as being *fake-quantized (FQ)*. To execute such a model with (low-precision) integer arithmetic, it must be integerized by mapping the FQ tensors to their integer images, which is equivalent to dividing by their respective ε. We provide more details on the process of integerization in Section 4.3 and a detailed mathematical derivation can be found in [60].

Methods to convert a full-precision network to a FQ one can be divided into two categories: Post-Training Quantization (PTQ) and QAT.

PTQ methods [61–63] generally start from a pre-trained full-precision model and do not optimize the model parameters any further. They determine suitable clipping bounds for each tensor to be quantized by calibration with real [62,64] or synthetic data [63,65].

In contrast, QAT methods train a network's parameters (or fine-tune them from a converged full-precision checkpoint) while incorporating quantization. Most such methods [66–69] are variations of the Straight-Through-Estimator (STE). Introduced in [70], the STE is implemented by applying a (non-differentiable) uniform quantizer as in Eq. (1) in the forward pass of the training procedure and defining its derivative as

$$\frac{dQ(x, a, b, n_{lvl})}{dx} = \mathbb{1}_{a \leq x \leq b},$$

i.e., treating it as a clipped identity function during the backward pass. Different QAT algorithms provide various ways of finding the clipping bounds $\{a, b\}$. The Learned Step Size Quantization (LSQ) [68] and TQT [67] methods apply the STE to the quantizer (1) to derive a gradient with respect to the quantization step size ε, which is used to optimize ε directly (in [68]) or the logarithm of the clipping interval $\log_2(b-a)$ (in [67]).

In contrast to the previously introduced methods, INQ [71] quantizes weights without employing the STE. This is achieved by iteratively quantizing and freezing a fraction of all weights and retraining the remaining full-precision weights until all weights are quantized and frozen. As this procedure does not apply to activations, they must still be quantized using an STE-based technique.

3.2.1. Extreme quantization - Binarized and ternarized networks

The techniques introduced in Section 3.2 can be applied to achieve quantization to any desired precision, with lower-precision quantization generally leading to more significant degradation of task accuracy. The most extreme forms of QNNs are BNNs [72,73] and TNNs [74,75]. In BNNs, all weights and activations are quantized to values in $\{-1, 1\}$. TNNs can be considered an extension of BNNs, with tensor elements taking values in the set $\{-1, 0, 1\}$. Consequently, arithmetic operations on binary and ternary values can be implemented with simple hardware: Computing the dot product of two N-element binary vectors can be achieved by N XNOR gates and an N-bit popcount unit [72]. The dot product of two N-element ternary vectors can be computed using N ternary multipliers and 2 N-element popcount units. This property makes BNNs and TNNs highly attractive targets for acceleration. While binary and ternary quantization schemes allow for the design of simple arithmetic units which improves inference energy efficiency considerably [11], statistical inference accuracy is typically degraded, even when employing QAT. Significant research has been conducted to combat this negative impact on accuracy and significant advances have been reported [76,77]; however, low-bitwidth quantization typically still carries a non-negligible penalty on statistical accuracy. As such, binary and ternary neural networks are best suited for applications where the penalty on accuracy is acceptable, such as in low-complexity tasks or always-on sensing or wake-up trigger applications. State-of-the-art digital BNN and TNN accelerators achieve high efficiencies of 145-392 TOp/J [11,78,79].

3.2.2. The CUTIE TNN accelerator

The Completely Unrolled Ternary Inference Engine (CUTIE) is an accelerator for TNNs, introduced in [11]. In contrast to most neural network accelerator architectures, CUTIE uses a compute architecture that is fully parallel in the computation of each output pixel by processing elements termed output channel compute unit (OCU). Each OCU computes the sum of dot products corresponding to a single filter in a fully unrolled manner and in a single cycle. To parallelize the computation of each pixel, CUTIE uses one OCU per output channel. Each OCU has a dedicated latch-based weight buffer that holds all filter weights. In order to reuse input kernel windows optimally, CUTIE features a 3-line buffer, which dispatches full 3×3 kernel windows.

CUTIE's maximally parallel design not only increases arithmetic optimization by forming large adder trees over the sum of dot products in each OCU, but also minimizes data movement since every pixel in the input feature map and every weight in every filter is only loaded once per layer. These characteristics make CUTIE highly efficient for neural network inference at the edge while enabling throughput at rates exceeding 1000 inf./s. Kraken features an updated version of CUTIE, described and evaluated in detail in [80]. The main improvements from the original version are support for causal, dilated 1D convolutions and the addition of an intermediate buffer for a window of up to 24 CNN output vectors, enabling the execution of ternarized TCNs. The CUTIE accelerator in Kraken features 96 OCUs, computing one output pixel with up to 96 input/output channels per cycle and resulting in a peak throughput of 56 TOp/s when clocked at 54 MHz for maximum efficiency in a 22 nm implementation.

4. Event frame processing pipeline

4.1. Overview

The class of cameras we target produce events consisting of three values each: Two spatial coordinates (x_e, y_e) describing the location of the event on the sensor grid, and a polarity $p_e \in \{-1, 1\}$ to indicate a decrease or increase in brightness, respectively. The first stage of our proposed pipeline performs data preparation, aggregating events detected by the DVS camera into 2-dimensional frames. The resulting frames are natively ternary: Pixels where at least one event occurred during a frame interval take the value of the most recent event's polarity, while pixels with no activity take the value 0. The prepared data is then processed by a two-stage hybrid CNN inspired by [20]. In the first processing stage, C_{in} sequential frames are fed into a fully ternarized 2D CNN. By processing multiple frames per inference, the 2D CNN can analyze short-term temporal dependencies, which are encoded into ternary feature vectors by the last layer.

In the third and final stage, a fully ternarized TCN analyzes the longer-term temporal dependencies by performing inference on a sliding window covering N_{TCN} feature vectors. Its output is a vector of class scores $V_p \in \mathbb{Z}^{N_c}$, where N_c is the number of classes. For the DVS128 dataset, $N_c = 11$. Finally, the class label of the sequence is given as $Cls = \arg\max_i (V_p)$.

Fig. 1 shows a diagram of the proposed processing pipeline. The frame extraction process is shown in Fig. 2 and described in more detail in Section 4.5.

4.2. Network design

The ternarized hybrid CNN/TCN adopts a simple feed-forward architecture, i.e., there are no residual branches in the network. All layers but the first have N_{ch} output channels. The two networks' topologies are listed in Table 2. The final, fully ternarized network consists only of convolutional layers with no bias,

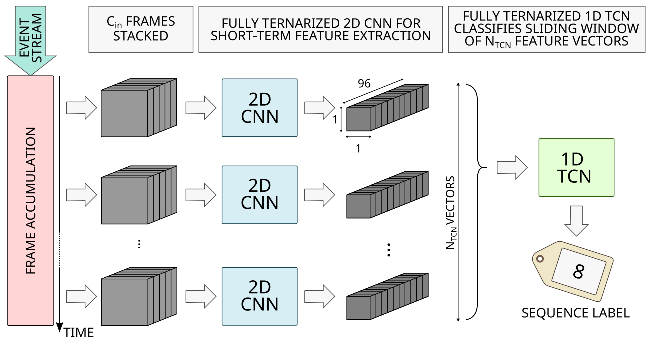

EVENT STREAM
C_in FRAMES STACKED
FULLY TERNARIZED 2D CNN FOR SHORT-TERM FEATURE EXTRACTION
FULLY TERNARIZED 1D TCN CLASSIFIES SLIDING WINDOW OF N_TCN FEATURE VECTORS
FRAME ACCUMULATION
2D CNN
96
1
1
N_TCN VECTORS
1D TCN
8
SEQUENCE LABEL
TIME

Fig. 1. Overview of the proposed processing pipeline.

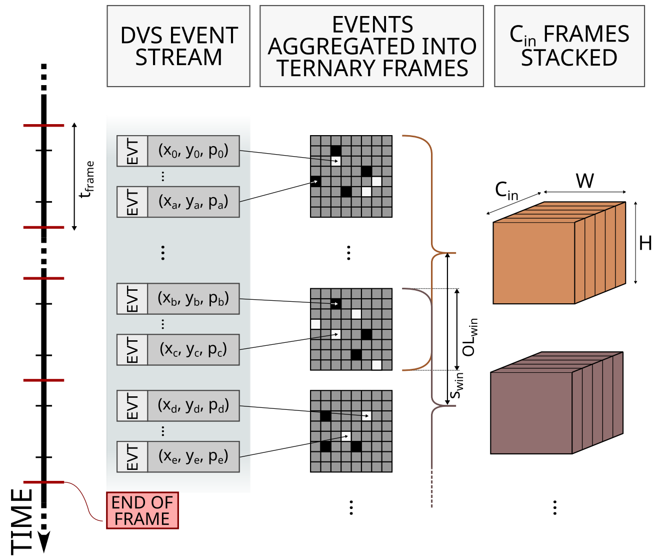


Fig. 2. Frame aggregation for processing by the 2D CNN. All events occurring during a time window of length t_{frame} are assigned to the same frame. The 2D CNN takes C_{in} frames as input, and two successive stacks of C_{in} frames overlap by OL_{win} frames.

ternary activations (3) and pooling layers. Consider a layer stack consisting of a convolutional layer with N_i/N_o input/output channels respectively and an optional pooling layer and denote the ternary values as $\mathcal{T} \triangleq \{-1, 0, 1\}$. The layer stack takes as input a tensor $\mathbf{X} \in \mathcal{T}^{N_i \times H_X \times W_X}$, where H_X/W_X are the spatial dimensions. $\mathbf{X}$ is convolved with the convolutional weights $\mathbf{W} \in \mathcal{T}^{N_o \times N_i \times k \times k}$ and pooled, to yield pre-activations $\mathbf{Z} \in \mathbb{Z}^{N_o \times H_Y \times W_Y}$, shown in Eq. (2). $\mathbf{Z}$ is then mapped to ternary activations $\mathbf{Y} \in \mathcal{T}^{N_o \times H_Y \times W_X}$ by channel-wise thresholding as shown in Eq. (3) with $\mathbf{t}^{lo}$ and $\mathbf{t}^{hi}$, $\mathbf{t}^{lo}, \mathbf{t}^{hi} \in \mathbb{Z}^{N_o}$.

$$\mathbf{Z} = pool\,(\mathbf{X} * \mathbf{W}) \tag{2}$$

$$y_{i,x,y} = \begin{cases} -1, & z_{i,x,y} < t_i^{lo} \\ 0, & t_i^{lo} \leq z_{i,x,y} < t_i^{hi} \\ 1, & z_{i,x,y} \geq t_i^{hi} \end{cases} \tag{3}$$

4.3. Quantization-aware training

To train the ternarized networks we evaluate, we perform QAT using two algorithms: INQ [71] and TQT [67]. To determine the impact of the activation function used to train the network, we compare Hard Hyperbolic Tangent (HtanH)-activated networks to ReLU-based ones. Note that both types of FQ networks are ultimately converted to fully ternarized networks as described in Section 4.2. QAT is performed in 4 steps (3 in the case of HtanH activations):

1. Training full-precision network to convergence,
2. quantization of activations,
3. (incremental) weight quantization, and, if ReLU activations were used,
4. fine-tuning to compensate for non-zero padding.

For all training algorithm and activation function combinations, we train networks with Batch Normalization (BN) layers inserted after the convolutional layers to allow for channel-wise scaling. After full-precision training has converged, HtanH and ReLU activations of layer l are replaced with symmetrical and asymmetrical 3-level step functions (4) and (5), respectively.

$$A^l_{symm}(x) = \begin{cases} \varepsilon^l_A, & x \geq \varepsilon^l_A/2 \\ 0, & -\varepsilon^l_A/2 \leq x < \varepsilon^l_A/2 \\ -\varepsilon^l_A, & x < -\varepsilon^l_A/2 \end{cases} \tag{4}$$

$$A^l_{asymm}(x) = \begin{cases} 2\varepsilon^l_A, & x \geq 3/2\varepsilon^l_A \\ \varepsilon^l_A, & \varepsilon^l_A/2 \leq x < 3/2\varepsilon^l_A \\ 0, & x < \varepsilon^l_A/2 \end{cases} \tag{5}$$
$$= A^l_{symm}(x - \varepsilon^l_A) + \varepsilon^l_A$$

The network is retrained for a few epochs before weight quantization is applied. When using INQ, weight and activation quantization step sizes are kept constant at 1, i.e., $\varepsilon_W = \varepsilon_A = 1$. Weights are frozen to the nearest values in $\mathcal{T}$ incrementally in order of decreasing magnitude, and the unmodified STE is used to propagate gradients through quantized activations. After each freezing step, the network's remaining learnable parameters (free weights and BN parameters) are retrained to convergence. When using TQT, both weight and activation clipping bounds are learned, with weight clipping bounds of each layer learned individually for each output channel. When quantizing weights with TQT, fake-quantization is applied to all weights simultaneously. The detailed training parameters are listed in Section 6.1.

4.4. Network ternarization

To be deployed on CUTIE, FQ networks must be mapped to fully ternarized models containing only ternary parameters and

Table 2
Topology of the proposed hybrid CNN architecture, consisting of a 2D CNN for short-term feature extraction and a 1D TCN to capture longer-term temporal dependencies. **C{1,2}D**: 1/2-dimensional convolution, **MP**: Max-Pooling. **(C)S/V** indicate (**c**ausal) **s**ame/**v**alid padding, respectively.

	Layer	Out Ch.	Out Res.	Pad	Dilation
2D CNN	Input	C_{in}	64×64	N/A	N/A
	C2D (3×3)	32	64×64	S	1
	MP (2×2)	32	32×32	V	1
	C2D (3×3)	N_{ch}	32×32	S	1
	MP (2×2)	N_{ch}	16×16	V	1
	C2D (3×3)	N_{ch}	16×16	S	1
	MP (2×2)	N_{ch}	8×8	V	1
	C2D (3×3)	N_{ch}	8×8	S	1
	MP (2×2)	N_{ch}	4×4	V	1
	C2D (3×3)	N_{ch}	2×2	V	1
	MP (2×2)	N_{ch}	1×1	V	1
1D TCN	Input	N_{ch}	N_{TCN}	N/A	N/A
	C1D (2)	N_{ch}	N_{TCN}	CS	1
	C1D (2)	N_{ch}	N_{TCN}	CS	2
	C1D (2)	N_{ch}	N_{TCN}	CS	4
	C1D (N_{TCN})	11	1	V	1

thresholding activation functions of the form (3). For FQ networks using asymmetrical (ReLU) activations (5), this requires the replacement of those activations by symmetrical ones (4). (5) is equivalent to (4) shifted by ε_A^l in both the argument and the output. The shift of the argument translates to shifted thresholds, while the shift in the output is equivalent to a bias of ε_A^l being added to the next layer's input, which can be propagated through the convolution operation to an equivalent output bias. However, this transformation would require padding the edge regions of the input feature map to layer $l+1$ with $-\varepsilon_A^l$ (FQ)/-1 (integerized), which CUTIE does not support. We compensate for this by fine-tuning the network after fake-quantization with padding values of $+\varepsilon_A^l$, which corresponds to the zero-padding applied by CUTIE on the transformed network.

To generate the fully integerized network, all floating-point parameters of an activation-convolution-BN(-pooling)-activation layer stack are folded into channel-wise integer thresholds of the ternary activation layer, terminating the stack. Consider the lth stack with (FQ) convolutional weights $\mathbf{W}^l = \left[\mathbf{W}_0^l, \dots, \mathbf{W}_{N_o-1}^l\right]$ and bias $\mathbf{B}^l$, input and output activation quanta ε_A^{l-1} and ε_A^l (as the input data is already ternarized, $\varepsilon_A^0 = 1$), channel-wise weight quanta $\boldsymbol{\varepsilon}_W^l$ and N_O^l output channels. We fold the convolutional bias and the BN parameters the into channel-wise affine transformation parameters $\hat{\boldsymbol{\gamma}}^l$ and $\hat{\boldsymbol{\beta}}^l$, shown in Eq. (6). From these, we can compute the integer threshold vectors $\hat{\mathbf{t}}^{lo,l}$ and $\hat{\mathbf{t}}^{hi,l}$ for layer l as in Eq. (7). Because negative entries of $\hat{\boldsymbol{\gamma}}$ flip the inequalities in (3), we sign-invert the corresponding output channels' weights and flip their thresholds before integerizing them, respectively (8), (9). In the final network, $\mathbf{W}^l$ is replaced with its integerized counterpart $\tilde{\mathbf{W}}^l$ and BN-activation sequences are replaced with a single channel-wise integer thresholding activation of the form (3).

$$\mathbb{1}_R = \begin{cases} 1 & \text{Net uses ReLU activations} \\ 0 & \text{Otherwise} \end{cases}$$

$$\hat{B}_c^l = \begin{cases} B_c^l & l = 1 \\ B_c^l + \mathbb{1}_R \varepsilon_A^{l-1} \sum_i W_{c,i}^l & l > 1 \end{cases} \quad \forall\ 0 \le c < N_o$$

$$\hat{\boldsymbol{\beta}}^l = \boldsymbol{\gamma}^l \frac{(\hat{\mathbf{B}}^l - \boldsymbol{\mu}^l)}{\boldsymbol{\sigma}^l} + \boldsymbol{\beta}^l, \qquad \hat{\boldsymbol{\gamma}}^l = \frac{\boldsymbol{\gamma}^l}{\boldsymbol{\sigma}^l} \tag{6}$$

$$\hat{\mathbf{t}}^{lo} = \frac{(\mathbb{1}_R - 0.5)\varepsilon_A^l - \hat{\boldsymbol{\beta}}}{\hat{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}_W^l \varepsilon_A^{l-1}}, \qquad \hat{\mathbf{t}}^{hi} = \frac{(\mathbb{1}_R + 0.5)\varepsilon_A^l - \hat{\boldsymbol{\beta}}}{\hat{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}_W^l \varepsilon_A^{l-1}} \tag{7}$$

$$\tilde{\mathbf{W}}_c^l = \left(\mathbf{W}_c^l - 2\mathbf{W}_c \mathbb{1}_{\gamma_c < 0}\right) / \varepsilon_{W,c}^l \qquad \forall\ 0 \le c < N_o \tag{8}$$

$$t_c^{lo,hi} = \left\lceil \hat{t}_c^{lo,hi} - 2\hat{t}_c^{lo,hi} \mathbb{1}_{\gamma_c < 0} \right\rceil \qquad \forall\ 0 \le c < N_o \tag{9}$$

Integerization for deployment on the PULP cluster is more straightforward and follows the procedure laid out in [60]. The ternarized weights are mapped to 2-bit integers, and activation operations are mapped to requantization layers (also called *Integer Channel Normalization* in [81]). The lth requantization layer folds ε scaling and BN into a single channel-wise affine transformation with parameters $\tilde{\boldsymbol{\gamma}}^l$ and $\tilde{\boldsymbol{\beta}}^l$ followed by an arithmetic right shift by a layer-wise parameter D_l, effectively performing fixed-point arithmetic with integer operations. The result is finally clipped to the appropriate integer range, implementing the nonlinear activation function. Eq. (10) shows the ternary requantization operation for layer l (assuming a ReLU activation) and Eqs. (11), (12) show how $\tilde{\boldsymbol{\gamma}}^l$ and $\tilde{\boldsymbol{\beta}}^l$ are computed. Note that by adding 0.5 to the folded bias term, the flooring operation in (10) is converted into a rounding operation.

$$RQ^l(\boldsymbol{X}) = clip\left(\lfloor\left(\left(\tilde{\boldsymbol{\gamma}}\boldsymbol{X} + \tilde{\boldsymbol{\beta}}\right) >> D_l\right)\rfloor, 0, 2\right) \tag{10}$$

$$\tilde{\boldsymbol{\gamma}}^l = \left\lfloor 2^D \frac{\boldsymbol{\varepsilon}_W^l \boldsymbol{\varepsilon}_A^{l-1} \boldsymbol{\gamma}^l}{\boldsymbol{\varepsilon}_A^l \boldsymbol{\sigma}^l} \right\rceil \tag{11}$$

$$\tilde{\boldsymbol{\beta}}^l = \left\lfloor \tilde{\boldsymbol{\gamma}}^l \left(\boldsymbol{B}^l - \boldsymbol{\mu}^l\right) + 2^D \left(\frac{\boldsymbol{\beta}^l}{\boldsymbol{\varepsilon}_A^l} + 0.5\right) \right\rceil \tag{12}$$

While thresholding-based activations are completely equivalent to their fake-quantized equivalents, the requantizing activation (10) introduces some numerical mismatches due to the rounding of $\tilde{\boldsymbol{\gamma}}$ and $\tilde{\boldsymbol{\beta}}$. Higher values of D decrease the discrepancy, so D is chosen as large as possible while avoiding integer overflows resulting from the multiplication with $\mathbf{X}$ in (10). In our implementation, we choose $D = 19$ to achieve equivalent integerized classification accuracy to that of the FQ network.

4.5. Data preparation

Unlike previous works [28,30], which employ fully event-based processing schemes and rely on SNNs to classify event data directly, our proposed approach first aggregates the event stream into ternary 2D images. This incurs some overhead for data preparation (frames need to be assembled and buffered) and decouples the processing energy from the density of the event stream. In practice, however, the power required for data preparation is vanishingly small in comparison to the idle power of the running system (see Section 6.2). The lack of fine-grained energy-activity proportionality is more than compensated by the unparalleled efficiency of TNN-based processing on CUTIE, which is indeed owed to the regularity of DNNs. Finally, activity-energy proportionality is still supported in a coarse-grain manner by only triggering inference when the input event density detected by the DVS peripheral surpasses a programmable threshold.

4.5.1. Parameters in data generation

The process of generating frame data from the event stream, illustrated in Fig. 2 has several degrees of freedom:

- Frame rate *FPS*, or, equivalently, frame time $t_{frame} = \frac{1}{FPS}$
- CNN input window size C_{in}; this parameter dictates the number of input channels to the CNN's first layer
- temporal CNN input window stride s_{win}, which gives rise to the input window overlap $OL_{win} = C_{in} - s_{win}$, and
- downsampling factor D - from our experimental observations, we choose $D = 2$, i.e., the height and width of the original frame are both halved, and the resulting frame has 1/4 the resolution of the raw camera data.

While all of these parameters must be fixed for network training, the DVS peripheral performing the frame aggregation has been designed to leave them configurable at runtime at negligible hardware overhead. The receptive time interval t_p of the full hybrid network is given by the number of input vectors to the TCN N_{TCN}, the number of frames encoded in a single vector C_{in} and the stride of the CNN input windows s_{win} as $t_p = C_{in} + (N_{TCN} - 1)s_{win}$. Once running, the latency l of the system to react is determined by the window stride and the frame rate, as well as the processing time t_{inf}: $l = s_{win}/FPS + t_{inf} \approx s_{win}/FPS$. When running networks on CUTIE, we can approximate $t_{inf} \approx 0$ due to CUTIE's very high throughput compared to the frame time (see Section 6.2).

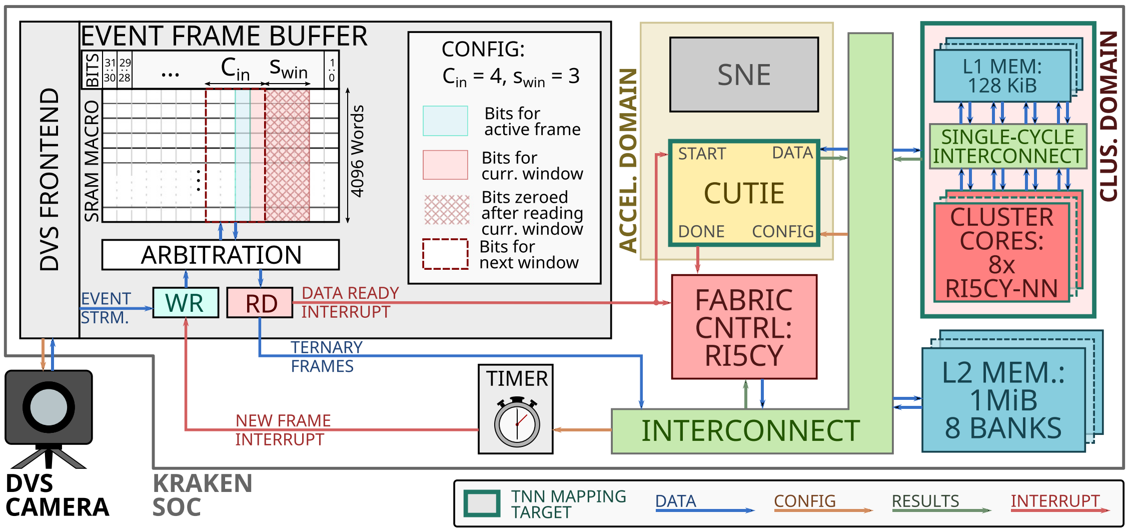


Fig. 3. Overview of Kraken SoC with detailed illustration of frame buffer operation. Ternary frames aggregated from DVS events are stored in a frame buffer. After s_{win} frames have been received, C_{in} frames are written to CUTIE's internal memories or L2 memory. From there, they can be processed by CUTIE or the PULP cluster (green outlines indicate processing units to which the TNN can be mapped). When running the TNN on CUTIE, the only interaction required from the FC core is readout and interpretation of the class scores.

5. SoC architecture: Kraken

We deploy the trained hybrid TCNs on the *Kraken* PULP SoC [82]. Kraken has three main processing domains, each individually power-gateable: The *SoC domain*, the *cluster domain* and the *accelerator domain* (see Fig. 3).

The SoC domain contains a single RI5CY core, implementing the XpulpV2 extension [83] to the RISC-V ISA. This core is designated as the *FC* and is responsible for orchestrating system operation and managing peripherals. The SoC domain contains 1MiB of SRAM-based L2 memory, which serves as the main working memory of Kraken. A wide range of peripherals for off-chip communication, including a DVS camera interface (described in detail in Section 5.1), are also located in the SoC domain and are managed by the μDMA engine [84], which allows operation of peripherals with minimal involvement of the FC core. Off-chip peripherals are powered by the same supply rail as the FC and the L2 memory but are clocked by a separate Frequency-Locked Loop (FLL) clock generator.

The cluster domain contains a PULP cluster of 8 RI5CY cores to perform compute-intensive processing tasks. Additionally to the XpulpV2 extension, the cluster cores implement the XpulpNN extension [5]. XpulpNN offers optimized hardware support for low-precision arithmetic via custom MAC-and-load instructions. These instructions combine low-precision MAC operations on packed 2-bit, 4-bit, or 8-bit operands with memory access and pointer update operations. This approach greatly mitigates the von Neumann bottleneck of load-store ISAs for matrix multiplication kernels, enabling the efficient execution of low-precision neural network layers. To minimize memory access overhead, the cluster has 128 KiB of L1 tightly-coupled data memory (TCDM), which is divided into 16 interleaved banks and provides single-cycle access to temporary data.

Finally, the accelerator domain contains two application-specific accelerators: CUTIE, targeting TNN inference and SNE [85], an all-digital SNN accelerator. Each accelerator can be clock-gated and power-gated individually. For Kraken, CUTIE has been modified from its first-generation architecture and configuration by changing the maximum number of input and output channels to 96, the inclusion of an additional memory bank to store ternary CNN output vectors and the capability to process those vectors with TCN layers by allowing for 1-D kernels with configurable dilation and causal padding.

5.1. DVS interface

With the DVS interface (DVSI), Kraken possesses a versatile peripheral unit to interface with DVS cameras such as the DVS132S [32]. Its task is to read event data from the camera unit and write it to a configurable memory address. Event readout can be triggered by writing to a configuration register or by an on-chip timer/counter peripheral. As the DVS peripheral is implemented as a μDMA peripheral, it can operate without any intervention from the FC. The peripheral can write event data to the configured location in two different formats. For processing by algorithms operating on discrete events (e.g., SNNs), it can encode each event in a 32-bit word and write event data to consecutive memory addresses. Alternatively, the peripheral can buffer up to 15 event frames in the format required by our proposed TCN. For this purpose, it contains an SRAM macro of 4096×32 bits, which is used as a frame buffer. Each word corresponds to one pixel of a 64×64 feature map, with each channel's value occupying 2 bits in a word for NHWC ordering. An incoming event at location (y, x) is written to bits $[2c_{curr} + 1, 2c_{curr}]$ at address $64y + x$, where c_{curr} is a wrapping counter ranging from 0 to 15, indexing the bits at which the currently active frame is stored. As a new frame interval starts, c_{curr} is incremented. The frame buffer is configurable in C_{in} and s_{win}. After s_{win} frames have been written into the buffer, the readout logic streams out the most recent C_{in} frames by reading each word, circularly shifting it to the right by $2c_{curr}$ bits and masking bits $[31 : 2C_{in}]$ of the resulting word to zero before writing it to the configured memory location. After reading a word, the s_{win} timesteps no longer used in the next input window are zeroed to avoid contamination of future frames by old events. The selective writing of events to bit indices and zeroing of stale data in single write operations is made possible by bit-selection signals exposed by the memory macros used in Kraken. The operation of the frame buffer is illustrated in Fig. 3.

As the DVS peripheral can be configured to write to any address in Kraken's memory range, it can transfer input data directly to CUTIE's internal activation memories. After an input window has been streamed out, the peripheral raises an interrupt line which is connected both to the FC and CUTIE, allowing autonomous operation of the entire processing pipeline by directly triggering CUTIE's inference without redundant data transfers

Table 3
Hyperparameters for QAT with TQT.

Parameter	Value
Epochs	50
Batch size	128
Opt.	Adam
LR_0	0.01
LR decr.[a]	Cosine annealing [86]
E_{Q_Wt}/E_{Q_Act} [b]	0/8
Act. clip init.[c]	Const. 6.0

[a]We perform one cycle of cosine annealing over 50 epochs.
[b]Activations and weights are quantized starting from the specified epoch.
[c]Const. x: clipping bounds initialized to x.

or other intervention by the FC core. After CUTIE has finished running the TNN on the input data, it raises an interrupt line and the FC can process and interpret the network's output.

5.2. Mapping TNNs on Kraken

The two processing domains we target for the deployment of our networks are the PULP cluster and CUTIE. In both cases, networks are executed sequentially in a layer-by-layer fashion. To efficiently use the cluster's computational resources, the data on which the cores operate must be stored in the high-bandwidth L1 memory. However, Kraken's L1 memory is too small to hold the inputs, outputs, and weights, necessitating *tiled* execution of layers: the complete input, weight, and output buffers for a layer are held in L2 memory and the layer is divided into smaller execution units by tiling the input, output, and weight tensors along the input channel, output channel or spatial dimensions. For the execution of each tile, the corresponding inputs and weights are transferred by direct memory access (DMA) from L2 to L1 memory and the cluster computes the partial output, which is then transferred back to L2. Double buffering is used to hide the latency of the DMA transfers.

In contrast, CUTIE specializes in efficient processing of TNNs by implementing dedicated activation and weight memories, which offer sufficient memory to avoid tiling of the network. In order to maximize energy efficiency of the system, inferences are executed on the accelerator in a self-contained fashion. To prepare for network execution, weights are written to CUTIE's internal weight memory and parameters for each layer (e.g., kernel size, input size, stride, etc.) are configured, both via CUTIE's external data interface. A ternary input tensor can then be written to CUTIE's activation memory. Inference is triggered by writing to a configuration register or if CUTIE's start interrupt line is raised. CUTIE executes networks autonomously on its internal memories and raises an interrupt when inference has concluded, allowing the host system to read and interpret the results.

6. Results

In this section, we present the results of our algorithmic evaluations as well as power measurements conducted on the Kraken system. First, we show the impact of data preparation, architectural, and training parameters on the network's classification accuracy. Then, we show power consumption and latency measurements for networks mapped both to CUTIE and to Kraken's PULP cluster.

6.1. Network design, data preparation and QAT algorithms

We evaluate the impact of network design, training, and data preparation parameters on statistical accuracy from multiple aspects. The hyperparameters for QAT with TQT are shown in Table 3, the training schedule for QAT with INQ is shown in Fig. 4.

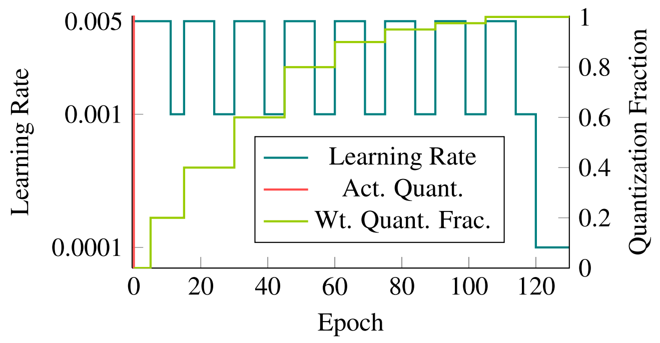


Fig. 4. Quantization and learning rate schedules used to ternarize full-precision networks with the INQ algorithm. The left y-axis measures the learning rate, the right y-axis measures the fraction of weights quantized. Hyperparameters not shown in this plot are identical to those used for QAT with TQT.

6.1.1. Receptive time interval t_p

The receptive time interval t_p is tied to the network's reaction time to changes in the input: For larger t_p, it will take longer for the new inputs to propagate through the network and saturate the TCN's input window. To minimize the system's reaction time, it is thus desirable to keep t_p as short as possible. However, a very short t_p limits the temporal context of the network's input, which may negatively impact classification accuracy. In this context, we consider a configuration *Pareto-optimal* if there is no network that achieves higher accuracy with a shorter t_p, and the set of Pareto-optimal networks forms a *Pareto front*. As described in Section 4.5, t_p is influenced both by network design (C_{in}, N_{TCN}) and data preparation parameters (s_{win}, FPS). Setting $N_{ch} = 96$, we trained 47 networks with parameter combinations resulting in t_p ranging from 42 ms to 1433 ms. We trained full-precision networks on the dataset split used by most previous works (users 1–23 in the training set and users 24–29 as the validation set). Of the full-precision networks forming the Pareto front, we also trained ternarized versions using the TQT algorithm and ReLU activations using 4-fold cross-validation (CV). The results are shown in Fig. 5 for both the standard dataset split and with 4-fold CV, selecting different users as the validation set. From Fig. 5, we can observe multiple points of interest. First, at $t_p = 300$ ms, a ternarized network already reaches 96.86% validation accuracy on the standard dataset split, only 0.74 percentage points below the maximum observed accuracy of 97.7%. The accuracy drop from ternarization is below 1.5 percentage points in all evaluated networks, with an average drop of 0.6 percentage points. Quantization even results in increased accuracy in some cases, confirming that the problem is well-suited to be solved by TNNs. Lastly, the average statistical accuracy when using 4-fold CV is lower by 1.55 percentage points than with the standard dataset split, indicating that the standard split is easier, with the training set representing the validation set accurately.

6.1.2. Network design, dataset generation and QAT algorithms

As a fully exhaustive network design, training, and dataset generation parameter search would be infeasible, we present ablation results over those parameters and design choices we found to have the largest impact on statistical accuracy. Fig. 6 shows the impact of varying FPS while maintaining $t_p = 900$ ms by changing C_{in}. We observed that increasing framerates significantly improves classification accuracy only up to 60 FPS, after which it stagnates between 97.3% and 97.9%. Next, we sought to determine the impact of the network's TCN stage on statistical accuracy by training two of the Pareto-optimal networks shown in Fig. 5 (t_p = 300 ms and t_p = 900 ms) without the TCN. Instead, the N_{TCN} outputs of the CNN are fed directly into a linear

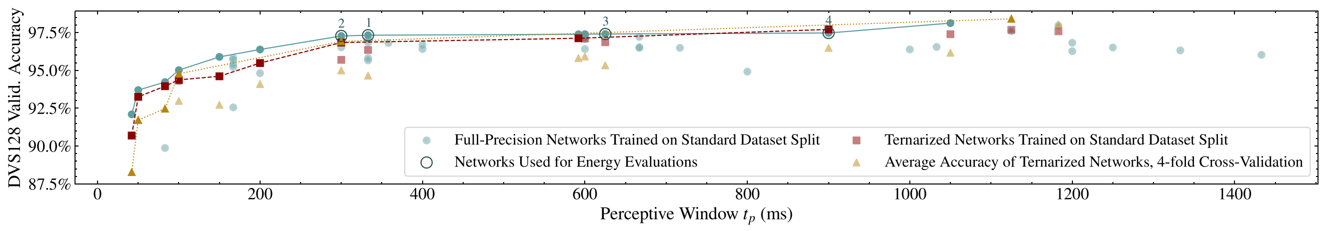


Fig. 5. Validation accuracy vs. t_p for various network architecture and dataset generation parametrizations. Data points connected by lines form the Pareto front of the respective experiments.

Table 4
Impact of QAT algorithm, quantization levels and the TCN stage on classification accuracy. Classification accuracy is specified for ternarized networks trained on the standard dataset split with full-precision accuracy in parentheses. n_{lvls} denotes the number of quantization levels, with 2 yielding a BNN and 3 a TNN. For the full parametrization of the networks, see Table 6.

Net (t_p)	n_{lvls}	Act.	QAT	TCN?	Acc. Quant. (FP)
Net 1 (300 ms)	3	ReLU	TQT	✗	95.3% (96.9%)
	3	ReLU	TQT	✓	**96.8**% (97.3%)
	3	ReLU	INQ	✓	93.9% (97.3%)
	3	HtanH	TQT	✓	95.5% (95.5%)
	3	HtanH	INQ	✓	93.8% (95.5%)
	2	ReLU	TQT	✓	95.8% (97.3%)
Net 4 (900 ms)	3	ReLU	TQT	✗	96.3% (96.9%)
	3	ReLU	TQT	✓	**97.7**% (97.5%)
	3	ReLU	INQ	✓	97.2% (97.5%)
	3	HtanH	TQT	✓	96.4% (96.1%)
	3	HtanH	INQ	✓	95.5% (96.1%)
	2	ReLU	TQT	✓	97.0% (97.5%)

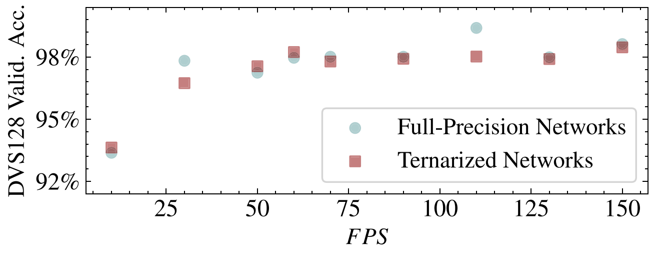


Fig. 6. Validation accuracy vs. *FPS*. Apart from *FPS* and C_{in}, the network's parametrization corresponds to network 4 from Table 6 with C_{in} chosen so that $t_p = 900\,\text{ms}$, i.e., $C_{in} = FPS/10$. The highest accuracy is achieved at 150*FPS* with 97.9%.

classifier layer. Table 4 shows that the inclusion of the TCN stage improves classification accuracy by 1.5 and 1.4 percentage points respectively. Finally, we evaluated the impact of the choice of QAT algorithm and activation function on accuracy. As Table 4 shows, the combination of ReLU activations and TQT yields the highest classification accuracy. Furthermore, ReLU activations and TQT outperform HtanH and INQ individually. To determine the impact of the number of quantization levels on accuracy, we also trained BNNs with equivalent architectures using ReLU activations and the TQT algorithm. Since our DVS event frame representation is ternary, the first layer of each BNN processes ternary inputs but uses binary weights. As Table 4 shows, these networks still perform well but achieve lower statistical accuracies by 1.0 and 0.7 percentage points than their ternary counterparts.

6.2. End-to-end gesture recognition on the Kraken SoC

We evaluated the real-world energy consumption of our processing pipeline on the Kraken SoC with the four network parametrizations shown in Table 6. On both CUTIE and the PULP cluster, we run inference under realistic operating conditions by matching the inference rates to the evaluated networks' training and accounting for the power consumption of idle domains. Our

Table 5
Operating conditions for power measurements and power consumption of the different power domains on the Kraken SoC.

	FC	Cluster	CUTIE
f_{clk} (MHz)	40	115	15
V_{DD} (V)	0.55	0.55	0.5
P_{idle} (mW)	2.0	1.6	2.7
P_{inf} (mW)	2.0	21.1	5.3
t_{inf} (ms)	N/A	17.8	0.9

results show that the CUTIE-based implementation of our TCN network requires only 7 μJ, 58.5 × less than a comparable implementation on the state-of-the-art RISC-V cluster using efficient, specialized ISA extensions.

6.2.1. Experimental setup

To map the network to the PULP cluster, we use DORY [87], an open-source DNN mapping utility targeting PULP systems, to generate test applications. The network layers are mapped to 2-bit kernels from the PULP-NN library [88] which take advantage of the XpulpNN ISA extension. As the DVSI's operation does not result in a measurable increase in power consumption, our test applications store input samples in L2 memory. An on-chip timer generates interrupts to the FC at the rate appropriate for each tested network, upon which the FC triggers inference either on the cluster or on CUTIE. As CUTIE operates on data from its internal activation memory, the FC transfers input activations before triggering the computation, which is slower than direct transfer by the DVSI and results in pessimistic latency and energy estimations. Table 5 details the operating conditions for our experiments, the power consumption of each domain during each phase of inference, and the inference latency t_{inf}. While the tested networks differ in the values of both N_{TCN} and C_{in}, this has no measurable impact on inference latency. Due to CUTIE's fully unrolled architecture, the number of input channels does not influence latency. For cluster networks, the number of input channels must be padded to a multiple of 16 to comply with the

Table 6
Key power and energy figures for 4 Pareto-optimal networks from Fig. 5. The networks were mapped to both the PULP cluster and CUTIE.

Net	t_p (ms)	FPS	C_{in}	N_{TCN}	inf./s	Acc.	E_{inf} (mJ)		P_{cont} (mW)		E_{win} (mJ)	
							Cluster	CUTIE	Cluster	CUTIE	Cluster	CUTIE
1	333	60	4	5	15	96.3%	0.41	**0.007**	8.9	4.68	**2.74**	**1.26**
2	**300**	120	4	9	30	96.8%			14.1	4.74	4.08	1.27
3	625	120	15	5	8	96.9%			**6.4**	4.68	3.59	2.35
4	900	60	6	9	10	**97.7%**			7.1	4.69	6.02	3.75

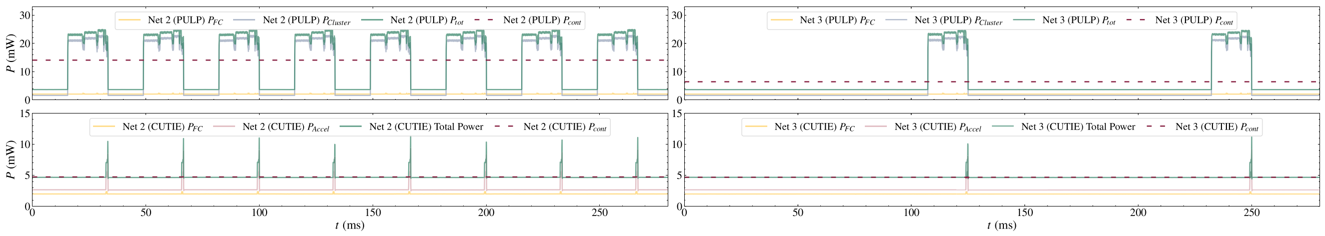


Fig. 7. Power traces for networks 2 and 3. Measurements for cluster-mapped implementations are shown in the top row, CUTIE implementations are shown in the bottom row. The differences in peak power for different inferences on CUTIE are a result of our power analyzer's temporal resolution, which is lower than the duration of the shortest current spikes.

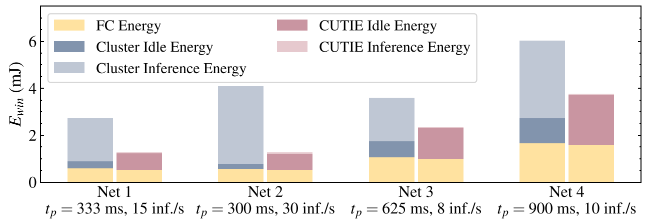


Fig. 8. Breakdown of E_{win} into contributions from FC, idle and inference energies. CUTIE-mapped nets exhibit very low inference energy with larger contributions from idle time, while cluster-mapped networks see large contributions from inference.

constraints of XpulpNN instructions. The differences in computation load caused by varying N_{TCN} are so small as to be unmeasurable. During idle phases, the processing units (PULP cluster or CUTIE) are clock-gated, but not power-gated. We also correct our power numbers for the leakage current drawn by oversized power gates for the accelerators in our design by measuring the leakage current with both accelerators power-gated and subtracting it from our current measurements. This makes our reported results equivalent to those of a design without any power-gating features. As the FC, cluster and accelerator domains have separate supply rails, unused domains can be turned off by switching off their external power supplies and we calculate the total power consumption as the summed power draw of the used domains.

6.2.2. Power measurements

Table 6 reports the key figures of merit of the four networks we evaluated on the Kraken SoC. The inference energy E_{inf} is given as the total energy consumed by the FC and CUTIE or the PULP cluster for a single prediction update, i.e., a single inference of the complete network. A complete inference on the PULP cluster takes 17.8 ms and consumes less than 500 µJ. Running the network on CUTIE reduces these figures further to 0.9 ms and 7 µJ, including the inefficient transfer of input data by the FC. This represents an improvement of 5× over the closest result reported in literature while accounting for the complete power consumption of the processing system including data transfer, rather than only the processing cores. Compared to previous end-to-end systems, our approach achieves a 67× lower inference energy. During inference, the operational efficiency of the system is 363 GOp/J when mapping the network to the cluster and 21 TOp/J for CUTIE-mapped networks. Fig. 7 shows the power traces from repeated inferences on two of the evaluated networks.

To evaluate the efficiency of our processing pipeline in real-world applications, we consider two scenarios. The first is that of always-on inference at the rate for which the network was trained (inf./s in Table 6). In this scenario, the metric of interest is P_{cont}, the processing system's average power consumption while performing continuous inference. Table 6 shows that the cluster and CUTIE implementations behave very differently in this respect: While P_{cont} strongly correlates with the inference rate for cluster-mapped networks, it is dominated by the idle power consumption for CUTIE-mapped networks. With higher inference rates, the efficiency advantage of CUTIE-mapped networks grows. While gesture recognition does not benefit from high inference rates, CUTIE's extremely high throughput makes it optimal for low-latency applications (e.g., perception pipelines for fast-flying nano-drones). Higher inference rates would allow it to amortize its leakage power consumption by increasing the utilization of its extremely efficient datapath.

The second scenario is event-triggered inference: The system is put to sleep until it is awakened by, e.g., increased event activity (which can be detected by Kraken's DVS peripheral). Upon awakening, a complete window of N_{TCN} inferences must be completed to produce a reliable prediction. We calculate the energy required to compute this prediction as $E_{win} = N_{TCN}E_{inf} + (N_{TCN} - 1)(\frac{1}{(inf/s)} - t_{inf}) \times P_{idle}$. Consequently, shorter t_p reduces the idle energy contribution and lower inference rates reduce the computation energy contribution. As with P_{cont}, idle energy dominates CUTIE networks' E_{win} and CUTIE performs best on networks with short t_p, while cluster networks' energy consumption rises sharply with N_{TCN} at similar t_p, as seen on networks 1 and 2. Fig. 8 visualizes the breakdown of E_{win} for the four evaluated nets mapped to CUTIE and the PULP cluster.

To determine the individual contributions of the components of our processing pipeline to the overall energy efficiency, we implemented the event-frame conversion in software. The inference energy breakdown of Network 1 mapped to the cluster and to CUTIE, using the software frame buffer and the hardware buffer in the DVSI peripheral, is shown in Fig. 9. CUTIE's inference energy is two orders of magnitude lower and the total inference energy is reduced again by almost 2× by using the hardware frame buffer.

7. Conclusion

In this work, we have presented an end-to-end pipeline for frame-based gesture recognition from DVS camera data,

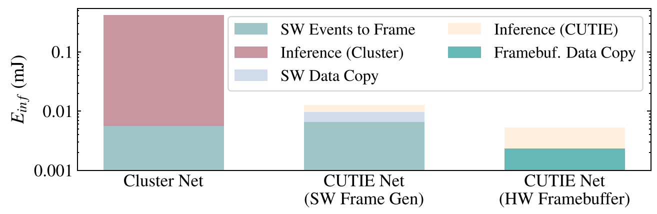


Fig. 9. Breakdown of inference energy for network 1 mapped to the PULP cluster as well as to CUTIE, using both a software-based event-to-frame mapping and the hardware frame buffer in the DVS peripheral. Note the logarithmic y-axis, causing the equal energy consumption of CUTIE inference to appear different visually.

implemented on the Kraken SoC. A dedicated on-chip peripheral aggregates the event stream from the DVS camera into ternary event frames. We classify the event frames with a fully ternarized hybrid TCN mapped either to the CUTIE accelerator or to the 8-core PULP cluster. To the best of our knowledge, our network sets a new state of the art for embedded implementations with 97.7% validation accuracy on the DVS128 gesture dataset, and the most accurate network we trained achieves 97.9%. On the CUTIE accelerator, we achieve a classification energy of 7 μJ, 67× lower than the previous state of the art for end-to-end gesture recognition at an inference latency of 0.9 ms. We further show that our approach can perform competitively even on software-programmable RISC-V cores with ISA extensions for sub-byte arithmetic. The cluster implementation exhibits an inference energy of 0.41 mJ at a latency of 17.8 ms for the same network running on Kraken's PULP cluster. With a continuous classification power consumption of 4.7 mW (CUTIE)/6.4 mW (cluster) for all involved processing components at 96.6% classification accuracy, our implementation highlights the added value of complete integration of sensor interface, preprocessing and efficient compute units. Kraken's programmability and flexibility allow for the implementation of a variety of processing scenarios (e.g., split execution of mixed-precision networks between CUTIE and the PULP cluster) to be explored in future works.

CRediT authorship contribution statement

Georg Rutishauser: Conceptualization, Methodology, Software, Visualization, Writing – original draft, Investigation. **Moritz Scherer:** Conceptualization, Software, Writing – review & editing. **Tim Fischer:** Software, Writing – review & editing. **Luca Benini:** Methodology, Writing – review & editing, Resources, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Public dataset: DVS128; accelerator & inference libraries are open-source

Acknowledgments

The authors would like to thank *armasuisse Science & Technology, Switzerland* for funding this research. Furthermore, we gratefully acknowledge support by the EU H2020 Fractal project funded by ECSEL-JU, Switzerland grant agreement #877056 and the HCSCA project #180625 funded by the Croatian-Swiss Research Programme, Switzerland.

References

[1] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, S. Xie, A ConvNet for the 2020s, in: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, IEEE, 2022, pp. 11966–11976, http://dx.doi.org/10.1109/CVPR52688.2022.01167, arXiv:2201.03545.
[2] H. Jiang, P. He, W. Chen, X. Liu, J. Gao, T. Zhao, SMART: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Stroudsburg, PA, USA, 2020, pp. 2177–2190, http://dx.doi.org/10.18653/v1/2020.acl-main.197.
[3] A. Dubatovka, J.M. Buhmann, Automatic detection of atrial fibrillation from single-lead ECG using deep learning of the cardiac cycle, BME Front. 2022 (2022) 1–12, http://dx.doi.org/10.34133/2022/9813062.
[4] A.Y. Hannun, P. Rajpurkar, M. Haghpanahi, G.H. Tison, C. Bourn, M.P. Turakhia, A.Y. Ng, Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network, Nat. Med. 25 (1) (2019) 65–69, http://dx.doi.org/10.1038/s41591-018-0268-3, arXiv:1707.01836v1.
[5] A. Garofalo, G. Tagliavini, F. Conti, L. Benini, D. Rossi, XpulpNN: Enabling energy efficient and flexible inference of quantized neural networks on RISC-V based IoT end nodes, IEEE Trans. Emerg. Top. Comput. 9 (3) (2021) 1489–1505, http://dx.doi.org/10.1109/TETC.2021.3072337, arXiv:2011.14325.
[6] J.D. Nunes, M. Carvalho, D. Carneiro, J.S. Cardoso, Spiking neural networks: A survey, IEEE Access 10 (2022) 60738–60764, http://dx.doi.org/10.1109/ACCESS.2022.3179968.
[7] A. Basu, L. Deng, C. Frenkel, X. Zhang, Spiking neural network integrated circuits: a review of trends and future directions, in: 2022 IEEE Custom Integrated Circuits Conference, CICC, IEEE, 2022, pp. 1–8, http://dx.doi.org/10.1109/CICC53496.2022.9772783.
[8] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A.J. Davison, J. Conradt, K. Daniilidis, D. Scaramuzza, Event-based vision: A survey, IEEE Trans. Pattern Anal. Mach. Intell. 44 (1) (2022) 154–180, http://dx.doi.org/10.1109/TPAMI.2020.3008413, arXiv:1904.08405.
[9] S.U. Innocenti, F. Becattini, F. Pernici, A. Del Bimbo, Temporal binary representation for event-based action recognition, in: 2020 25th International Conference on Pattern Recognition, ICPR, 2020, pp. 10426–10432, http://dx.doi.org/10.1109/ICPR48806.2021.9412991, arXiv:2010.08946.
[10] G. Rutishauser, M. Scherer, T. Fischer, L. Benini, Ternarized TCN for μJ/Inference gesture recognition from DVS event frames, in: 2022 Design, Automation & Test in Europe Conference & Exhibition, DATE, IEEE, 2022, pp. 736–741, http://dx.doi.org/10.23919/DATE54114.2022.9774592.
[11] M. Scherer, G. Rutishauser, L. Cavigelli, L. Benini, CUTIE: Beyond petaop/s/w ternary DNN inference acceleration with better-than-binary energy efficiency, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 41 (4) (2022) 1020–1033, http://dx.doi.org/10.1109/TCAD.2021.3075420, arXiv:2011.01713.
[12] R. Zhang, X. Jing, S. Wu, C. Jiang, J. Mu, F. Richard Yu, Device-free wireless sensing for human detection: The deep learning perspective, IEEE Internet Things J. 8 (4) (2021) 2517–2539, http://dx.doi.org/10.1109/JIOT.2020.3024234.
[13] S. Benatti, F. Casamassima, B. Milosevic, E. Farella, P. Schonle, S. Fateh, T. Burger, Q. Huang, L. Benini, A versatile embedded platform for EMG acquisition and gesture recognition, IEEE Trans. Biomed. Circuits Syst. 9 (5) (2015) 620–630, http://dx.doi.org/10.1109/TBCAS.2015.2476555.
[14] S. Tam, M. Boukadoum, A. Campeau-Lecours, B. Gosselin, A fully embedded adaptive real-time hand gesture classifier leveraging HD-sEMG and deep learning, IEEE Trans. Biomed. Circuits Syst. 14 (2) (2020) 232–243, http://dx.doi.org/10.1109/TBCAS.2019.2955641.
[15] M. Zanghieri, S. Benatti, A. Burrello, V. Kartsch, F. Conti, L. Benini, Robust real-time embedded EMG recognition framework using temporal convolutional networks on a multicore IoT processor, IEEE Trans. Biomed. Circuits Syst. 14 (2) (2020) 244–256, http://dx.doi.org/10.1109/TBCAS.2019.2959160.
[16] S. Benatti, G. Rovere, J. Bosser, F. Montagna, E. Farella, H. Glaser, P. Schonle, T. Burger, S. Fateh, Q. Huang, L. Benini, A sub-10 mW real-time implementation for EMG hand gesture recognition based on a multi-core biomedical SoC, in: 2017 7th IEEE International Workshop on Advances in Sensors and Interfaces, IWASI, IEEE, 2017, pp. 139–144, http://dx.doi.org/10.1109/IWASI.2017.7974234.
[17] S. Benatti, F. Montagna, V. Kartsch, A. Rahimi, D. Rossi, L. Benini, Online learning and classification of EMG-based gestures on a parallel ultra-low power platform using hyperdimensional computing, IEEE Trans. Biomed. Circuits Syst. 13 (3) (2019) 516–528, http://dx.doi.org/10.1109/TBCAS.2019.2914476.
[18] J. Lien, N. Gillian, M.E. Karagozler, P. Amihood, C. Schwesig, E. Olson, H. Raja, I. Poupyrev, Soli, ACM Trans. Graph. 35 (4) (2016) 1–19, http://dx.doi.org/10.1145/2897824.2925953.

[19] S. Wang, J. Song, J. Lien, I. Poupyrev, O. Hilliges, Interacting with soli, in: Proceedings of the 29th Annual Symposium on User Interface Software and Technology, ACM, New York, NY, USA, 2016, pp. 851–860, http://dx.doi.org/10.1145/2984511.2984565.

[20] M. Scherer, M. Magno, J. Erb, P. Mayer, M. Eggimann, L. Benini, TinyRadarNN: Combining spatial and temporal convolutional neural networks for embedded gesture recognition with short range radars, IEEE Internet Things J. 8 (13) (2021) 10336–10346, http://dx.doi.org/10.1109/JIOT.2021.3067382, arXiv:2006.16281.

[21] A. Burrello, M. Scherer, M. Zanghieri, F. Conti, L. Benini, A microcontroller is all you need: Enabling transformer execution on low-power IoT endnodes, in: 2021 IEEE International Conference on Omni-Layer Intelligent Systems, COINS, IEEE, 2021, pp. 1–6, http://dx.doi.org/10.1109/COINS51742.2021.9524173.

[22] D.W.O. Antillon, C.R. Walker, S. Rosset, I.A. Anderson, Glove-based hand gesture recognition for diver communication, IEEE Trans. Neural Netw. Learn. Syst. (2022) 1–13, http://dx.doi.org/10.1109/TNNLS.2022.3161682.

[23] X. Huang, Q. Wang, S. Zang, J. Wan, G. Yang, Y. Huang, X. Ren, Tracing the motion of finger joints for gesture recognition via sewing RGO-coated fibers onto a textile glove, IEEE Sens. J. 19 (20) (2019) 9504–9511, http://dx.doi.org/10.1109/JSEN.2019.2924797.

[24] F. Zhou, X. Li, Z. Wang, Efficient high cross-user recognition rate ultrasonic hand gesture recognition system, IEEE Sens. J. 20 (22) (2020) 13501–13510, http://dx.doi.org/10.1109/JSEN.2020.3004252.

[25] Y. Qifan, T. Hao, Z. Xuebing, L. Yin, Z. Sanfeng, Dolphin: Ultrasonic-based gesture recognition on smartphone platform, in: 2014 IEEE 17th International Conference on Computational Science and Engineering, IEEE, 2014, pp. 1461–1468, http://dx.doi.org/10.1109/CSE.2014.273.

[26] H. Lakhotiya, H.S. Pandita, R. Shankarmani, Real time sign language recognition using image classification, in: 2021 2nd International Conference for Emerging Technology, INCET, IEEE, 2021, pp. 1–4, http://dx.doi.org/10.1109/INCET51464.2021.9456432.

[27] S. Hussain, R. Saxena, X. Han, J.A. Khan, H. Shin, Hand gesture recognition using deep learning, in: 2017 International SoC Design Conference, ISOCC, IEEE, 2017, pp. 48–49, http://dx.doi.org/10.1109/ISOCC.2017.8368821.

[28] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, D. Modha, A low power, fully event-based gesture recognition system, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Vol. 27, IEEE, 2017, pp. 7388–7397, http://dx.doi.org/10.1109/CVPR.2017.781.

[29] E. Ceolini, C. Frenkel, S.B. Shrestha, G. Taverni, L. Khacef, M. Payvand, E. Donati, Hand-gesture recognition based on EMG and event-based camera sensor fusion: A benchmark in neuromorphic computing, Front. Neurosci. 14 (August) (2020) 1–15, http://dx.doi.org/10.3389/fnins.2020.00637.

[30] R. Massa, A. Marchisio, M. Martina, M. Shafique, An efficient spiking neural network for recognizing gestures with a DVS camera on the loihi neuromorphic processor, in: 2020 International Joint Conference on Neural Networks (IJCNN), No. July, IEEE, 2020, pp. 1–9, http://dx.doi.org/10.1109/IJCNN48605.2020.9207109, arXiv:2006.09985.

[31] P. Lichtsteiner, C. Posch, T. Delbruck, A 128 $\times$ 128 120 db 15 μs latency asynchronous temporal contrast vision sensor, IEEE J. Solid-State Circuits 43 (2) (2008) 566–576, http://dx.doi.org/10.1109/JSSC.2007.914337.

[32] C. Li, L. Longinotti, F. Corradi, T. Delbruck, A 132 by 104 10 μm-pixel 250 μW 1kefps dynamic vision sensor with pixel-parallel noise and spatial redundancy suppression, in: 2019 Symposium on VLSI Circuits, Vol. 2019-June, IEEE, 2019, pp. C216–C217, http://dx.doi.org/10.23919/VLSIC.2019.8778050.

[33] P. Blouw, X. Choo, E. Hunsberger, C. Eliasmith, Benchmarking keyword spotting efficiency on neuromorphic hardware, in: Proceedings of the 7th Annual Neuro-Inspired Computational Elements Workshop, ACM, New York, NY, USA, 2019, pp. 1–8, http://dx.doi.org/10.1145/3320288.3320304, arXiv:1812.01739.

[34] C. Frenkel, M. Lefebvre, J.-D. Legat, D.A. Bol, 0.086-Mm^2 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28 nm CMOS, IEEE Trans. Biomed. Circuits Syst. 13 (1) (2018) 1, http://dx.doi.org/10.1109/TBCAS.2018.2880425.

[35] C. Frenkel, J.-D. Legat, D. Bol, MorphIC: A 65-nm 738k-synapse/mm^2 quad-core binary-weight digital neuromorphic processor with stochastic spike-driven online learning, IEEE Trans. Biomed. Circuits Syst. 13 (5) (2019) 999–1010, http://dx.doi.org/10.1109/TBCAS.2019.2928793.

[36] Y. Xing, G. Di Caterina, J. Soraghan, A new spiking convolutional recurrent neural network (SCRNN) with applications to event-based hand gesture recognition, Front. Neurosci. 14 (November) (2020) http://dx.doi.org/10.3389/fnins.2020.590164.

[37] S.B. Shrestha, G. Orchard, SLAYER: Spike layer error reassignment in time, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems, in: NIPS'18, Vol. 2018-Decem, Curran Associates Inc., Red Hook, NY, USA, 2018, pp. 1419–1428, http://dx.doi.org/10.5555/3326943.3327073, arXiv:1810.08646.

[38] L. Cordone, B. Miramond, S. Ferrante, Learning from event cameras with sparse spiking convolutional neural networks, in: 2021 International Joint Conference on Neural Networks, IJCNN, IEEE, 2021, pp. 1–8, http://dx.doi.org/10.1109/IJCNN52387.2021.9533514, arXiv:2104.12579.

[39] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, Y. Tian, Incorporating learnable membrane time constant to enhance learning of spiking neural networks, in: 2021 IEEE/CVF International Conference on Computer Vision, ICCV, IEEE, 2021, pp. 2641–2651, http://dx.doi.org/10.1109/ICCV48922.2021.00266, arXiv:2007.05785.

[40] M.A. Mahovald, C. Mead, The silicon retina, Sci. Am. 264 (1991) 76–83.

[41] C. Brandli, R. Berner, Minhao Yang, Shih-Chii. Liu, T. Delbruck, A 240 $\times$ 180 130 db 3 μs latency global shutter spatiotemporal vision sensor, IEEE J. Solid-State Circuits 49 (10) (2014) 2333–2341, http://dx.doi.org/10.1109/JSSC.2014.2342715.

[42] T. Finateu, A. Niwa, D. Matolin, K. Tsuchimoto, A. Mascheroni, E. Reynaud, P. Mostafalu, F. Brady, L. Chotard, F. LeGoff, H. Takahashi, H. Wakabayashi, Y. Oike, C. Posch, 5.10 A 1280 $\times$ 720 back-illuminated stacked temporal contrast event-based vision sensor with 4.86 μm pixels, 1.066geps readout, programmable event-rate controller and compressive data-formatting pipeline, in: 2020 IEEE International Solid- State Circuits Conference, ISSCC, IEEE, 2020, pp. 112–114, http://dx.doi.org/10.1109/ISSCC19947.2020.9063149.

[43] D.P. Moeys, C. Li, J.N. Martel, S. Bamford, L. Longinotti, V. Motsnyi, D. San Segundo Bello, T. Delbruck, Color temporal contrast sensitivity in dynamic vision sensors, in: 2017 IEEE International Symposium on Circuits and Systems, ISCAS, IEEE, 2017, pp. 1–4, http://dx.doi.org/10.1109/ISCAS.2017.8050412.

[44] A. Zhu, L. Yuan, K. Chaney, K. Daniilidis, EV-FlowNet: Self-supervised optical flow estimation for event-based cameras, in: Robotics: Science and Systems XIV, Robotics: Science and Systems Foundation, 2018, http://dx.doi.org/10.15607/RSS.2018.XIV.062, arXiv:arXiv:1802.06898v4.

[45] F. Paredes-Valles, K.Y.W. Scheper, G.C.H.E. de Croon, Unsupervised learning of a hierarchical spiking neural network for optical flow estimation: From events to global motion perception, IEEE Trans. Pattern Anal. Mach. Intell. 42 (8) (2020) 2051–2064, http://dx.doi.org/10.1109/TPAMI.2019.2903179.

[46] G. Haessig, A. Cassidy, R. Alvarez, R. Benosman, G. Orchard, Spiking optical flow for event-based sensors using IBM's TrueNorth neurosynaptic system, IEEE Trans. Biomed. Circuits Syst. 12 (4) (2018) 860–870, http://dx.doi.org/10.1109/TBCAS.2018.2834558.

[47] Y. Li, Y. Guo, S. Zhang, S. Deng, Y. Hai, S. Gu, Differentiable spike: Rethinking gradient-descent for training spiking neural networks, in: A. Beygelzimer, Y. Dauphin, P. Liang, J. Wortman Vaughan (Eds.), Advances in Neural Information Processing Systems, 2021, pp. 1–14.

[48] J.H. Lee, T. Delbruck, M. Pfeiffer, Training deep spiking neural networks using backpropagation, Front. Neurosci. 10 (November) (2016) http://dx.doi.org/10.3389/fnins.2016.00508.

[49] G. Orchard, A. Jayawant, G.K. Cohen, N. Thakor, Converting static image datasets to spiking neuromorphic datasets using saccades, Front. Neurosci. 9 (November) (2015) 1–11, http://dx.doi.org/10.3389/fnins.2015.00437.

[50] H. Li, H. Liu, X. Ji, G. Li, L. Shi, CIFAR10-DVS: An event-stream dataset for object classification, Front. Neurosci. 11 (May) (2017) 1–10, http://dx.doi.org/10.3389/fnins.2017.00309.

[51] T. Serrano-Gotarredona, B. Linares-Barranco, A 128 $\times$ 128 1.5 sensitivity 0.9 vision sensor using transimpedance preamplifiers, IEEE J. Solid-State Circuits 48 (3) (2013) 827–838, http://dx.doi.org/10.1109/JSSC.2012.2230553.

[52] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, N. Houlsby, Big transfer (BiT): General visual representation learning, in: A. Vedaldi, H. Bischof, T. Brox, J.-M. Frahm (Eds.), Computer Vision – ECCV 2020, Springer International Publishing, Cham, 2020, pp. 491–507, http://dx.doi.org/10.1007/978-3-030-58558-7_29, arXiv:1912.11370.

[53] M. Davies, N. Srinivasa, T.H. Lin, G. Chinya, Y. Cao, S.H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.H. Weng, A. Wild, Y. Yang, H. Wang, Loihi: A neuromorphic manycore processor with on-chip learning, IEEE Micro 38 (1) (2018) 82–99, http://dx.doi.org/10.1109/MM.2018.112130359.

[54] G. Orchard, E.P. Frady, D.B.D. Rubin, S. Sanborn, S.B. Shrestha, F.T. Sommer, M. Davies, Efficient neuromorphic signal processing with loihi 2, in: 2021 IEEE Workshop on Signal Processing Systems (SiPS), No. 1, IEEE, 2021, pp. 254–259, http://dx.doi.org/10.1109/SiPS52927.2021.00053, arXiv:arXiv:2111.03746v1.

[55] M.V. Debole, B. Taba, A. Amir, F. Akopyan, A. Andreopoulos, W.P. Risk, J. Kusnitz, C.O. Otero, T.K. Nayak, R. Appuswamy, P.J. Carlson, A.S. Cassidy, P. Datta, S.K. Esser, G.J. Garreau, K.L. Holland, S. Lekuch, M. Mastro, J. Mckinstry, C. Di Nolfo, J. Sawada, B. Paulovicks, K. Schleupen, B.G. Shaw, J.L. Klamo, M.D. Flickner, J.V. Arthur, D.S. Modha, TrueNorth: Accelerating from zero to 64 million neurons in 10 years, Computer 52 (5) (2019) 20–29, http://dx.doi.org/10.1109/MC.2019.2903009.

[56] K. Wang, Z. Liu, Y. Lin, J. Lin, S. Han, HAQ: Hardware-aware automated quantization with mixed precision, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vol. 2019-June, IEEE, 2019, pp. 8604–8612, http://dx.doi.org/10.1109/CVPR.2019.00881, arXiv:1811.08886.

[57] Z. Yao, Z. Dong, Z. Zheng, A. Gholami, J. Yu, E. Tan, L. Wang, Q. Huang, Y. Wang, M.W. Mahoney, K. Keutzer, HAWQV3: Dyadic neural network quantization, in: M. Meila, T. Zhang (Eds.), Proceedings of the 38th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 11875–11886, arXiv:2011.10680.

[58] Y. Umuroglu, L. Rasnayake, M. Sjalander, BISMO: A scalable bit-serial matrix multiplication overlay for reconfigurable computing, in: 2018 28th International Conference on Field Programmable Logic and Applications, FPL, IEEE, 2018, pp. 307–3077, http://dx.doi.org/10.1109/FPL.2018.00059, arXiv:arXiv:1806.08862v1.

[59] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, H. Esmaeilzadeh, Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network, in: 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture, ISCA, IEEE, 2018, pp. 764–775, http://dx.doi.org/10.1109/ISCA.2018.00069, arXiv:1712.01507.

[60] F. Conti, Technical report: NEMO DNN quantization for deployment model, 2020, pp. 1–12, CoRR abs/2004.0. arXiv:2004.05930.

[61] R. Banner, Y. Nahshan, D. Soudry, Post-training 4-bit quantization of convolutional networks for rapid-deployment, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc., Vancouver, 2019, pp. 1–9, http://dx.doi.org/10.5555/3454287.3455001.

[62] Y. Choukroun, E. Kravchik, F. Yang, P. Kisilev, Low-bit quantization of neural networks for efficient inference, in: 2019 IEEE/CVF International Conference on Computer Vision Workshop, ICCVW, IEEE, 2019, pp. 3009–3018, http://dx.doi.org/10.1109/ICCVW.2019.00363, arXiv:1902.06822.

[63] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M.W. Mahoney, K. Keutzer, Zeroq: A novel zero shot quantization framework, in: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, IEEE, 2020, pp. 13166–13175, http://dx.doi.org/10.1109/CVPR42600.2020.01318, arXiv:2001.00281.

[64] M. Eldad, F. Alexander, A. Uri, G. Mark, Same, same but different: Recovering neural network quantization error through weight factorization, in: 36th International Conference on Machine Learning, ICML 2019, Vol. 2019-June, 2019, pp. 7886–7896, http://dx.doi.org/10.48550/arXiv.1902.01917, arXiv:1902.01917.

[65] M. Nagel, M.V. Baalen, T. Blankevoort, M. Welling, Data-free quantization through weight equalization and bias correction, in: Proceedings of the IEEE International Conference on Computer Vision 2019-Octob, 2019, pp. 1325–1334, http://dx.doi.org/10.1109/ICCV.2019.00141, arXiv:1906.04721.

[66] J. Choi, S. Venkataramani, V.V. Srinivasan, K. Gopalakrishnan, Z. Wang, P. Chuang, Accurate and efficient 2-bit quantized neural networks, in: A. Talwalkar, V. Smith, M. Zaharia (Eds.), Proceedings of Machine Learning and Systems, Vol. 1, 2019, pp. 348–359.

[67] S.R. Jain, A. Gural, M. Wu, C.H. Dick, Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks, in: Proceedings of Machine Learning and Systems, 2020, pp. 112–128, arXiv:1903.08066.

[68] S.K. Esser, J.L. McKinstry, D. Bablani, R. Appuswamy, D.S. Modha, Learned step size quantization, in: International Conference on Learning Representations, 2020, pp. 1–12, arXiv:1902.08153.

[69] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Quantized neural networks: Training neural networks with low precision weights and activations, J. Mach. Learn. Res. 18 (1) (2017) 6869–6898, http://dx.doi.org/10.5555/3122009.3242044, arXiv:1609.07061.

[70] Y. Bengio, N. Léonard, A. Courville, Estimating or propagating gradients through stochastic neurons for conditional computation, 2013, pp. 1–12, CoRR abs/1308.3. arXiv:1308.3432.

[71] A. Zhou, A. Yao, Y. Guo, L. Xu, Y. Chen, Incremental network quantization: Towards lossless CNNs with low-precision weights, in: International Conference on Learning Representations, 2017, arXiv:1702.03044.

[72] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, XNOR-net: ImageNet classification using binary convolutional neural networks, in: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), in: LNCS, vol. 9908, 2016, pp. 525–542, http://dx.doi.org/10.1007/978-3-319-46493-0_32, arXiv:arXiv:1603.05279v4.

[73] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks: Training deep neural networks with weights and activations constrained to 1 or -1, 2016, arXiv:1602.02830.

[74] L. Deng, P. Jiao, J. Pei, Z. Wu, G. Li, GXNOR-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework, Neural Netw. 100 (2018) 49–58, http://dx.doi.org/10.1016/j.neunet.2018.01.010, arXiv:1705.09283.

[75] H. Alemdar, V. Leroy, A. Prost-Boucle, F. Petrot, Ternary neural networks for resource-efficient AI applications, in: 2017 International Joint Conference on Neural Networks, IJCNN, IEEE, 2017, pp. 2547–2554, http://dx.doi.org/10.1109/IJCNN.2017.7966166.

[76] A. Gholami, S. Kim, Z. Dong, Z. Yao, M.W. Mahoney, K. Keutzer, A survey of quantization methods for efficient neural network inference, 2021, arXiv preprint arXiv:2103.13630.

[77] L. Deng, G. Li, S. Han, L. Shi, Y. Xie, Model compression and hardware acceleration for neural networks: A comprehensive survey, Proc. IEEE 108 (4) (2020) 485–532, http://dx.doi.org/10.1109/JPROC.2020.2976475.

[78] B. Moons, D. Bankman, L. Yang, B. Murmann, M. Verhelst, BinarEye: An always-on energy-accuracy-scalable binary CNN processor with all memory on chip in 28 nm CMOS, in: 2018 IEEE Custom Integrated Circuits Conference, CICC, IEEE, 2018, pp. 1–4, http://dx.doi.org/10.1109/CICC.2018.8357071, arXiv:1804.05554.

[79] R. Andri, G. Karunaratne, L. Cavigelli, L. Benini, ChewBaccaNN: A flexible 223 TOPS/W BNN accelerator, in: 2021 IEEE International Symposium on Circuits and Systems, ISCAS, IEEE, 2021, pp. 1–5, http://dx.doi.org/10.1109/ISCAS51556.2021.9401214.

[80] M. Scherer, A.D. Mauro, T. Fischer, G. Rutishauser, L. Benini, TCN-CUTIE: A 1, 036-TOp/s/W, 2.72-μ j/inference, 12.2-mW all-digital ternary accelerator in 22-nm FDX technology, IEEE Micro 43 (1) (2023) 42–48, http://dx.doi.org/10.1109/MM.2022.3226630, arXiv:arXiv:2212.00688v1.

[81] M. Rusci, A. Capotondi, L. Benini, Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers, in: I. Dhillon, D. Papailiopoulos, V. Sze (Eds.), Proceedings of Machine Learning and Systems, Vol. 2, 2020, pp. 326–335, arXiv:1905.13082.

[82] A. Di Mauro, M. Scherer, D. Rossi, L. Benini, Kraken: A direct event/frame-based multi-sensor fusion SoC for ultra-efficient visual processing in nano-UAVs, in: 2022 IEEE Hot Chips 34 Symposium, HCS, IEEE, 2022, pp. 1–19, http://dx.doi.org/10.1109/HCS55958.2022.9895621.

[83] M. Gautschi, P.D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F.K. Gurkaynak, L. Benini, Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 25 (10) (2017) 2700–2713, http://dx.doi.org/10.1109/TVLSI.2017.2654506.

[84] A. Pullini, D. Rossi, G. Haugou, L. Benini, μDMA: An autonomous I/O subsystem for IoT end-nodes, in: 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), Vol. 1, IEEE, 2017, pp. 1–8, http://dx.doi.org/10.1109/PATMOS.2017.8106971.

[85] A. Di Mauro, A.S. Prasad, Z. Huang, M. Spallanzani, F. Conti, L. Benini, SNE: an energy-proportional digital accelerator for sparse event-based convolutions, in: 2022 Design, Automation & Test in Europe Conference & Exhibition, DATE, IEEE, 2022, pp. 825–830, http://dx.doi.org/10.23919/DATE54114.2022.9774552.

[86] I. Loshchilov, F. Hutter, SGDR: Stochastic gradient descent with warm restarts, in: International Conference on Learning Representations, 2017, pp. 1–16, arXiv:1608.03983.

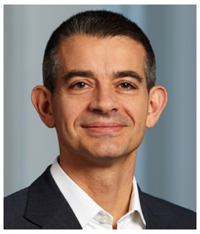
[87] A. Burrello, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi, F. Conti, DORY: Automatic end-to-end deployment of real-world DNNs on low-cost IoT MCUs, IEEE Trans. Comput. 70 (8) (2021) 1253–1268, http://dx.doi.org/10.1109/TC.2021.3066883, arXiv:2008.07127.

[88] A. Garofalo, M. Rusci, F. Conti, D. Rossi, L. Benini, Pulp-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors, Phil. Trans. R. Soc. A 378 (2164) (2020) http://dx.doi.org/10.1098/rsta.2019.0155, arXiv:1908.11263.

Georg Rutishauser received the B.Sc. and M.Sc. degrees in electrical engineering and information technology from ETH Zürich in 2015 and 2018, respectively, where he is currently pursuing the Ph.D. degree with the Integrated Systems Laboratory. His research interests include algorithms and hardware for reduced-precision deep learning, and their application in computer vision and embedded systems. Contact him at georgr@iis.ee.ethz.ch.

Moritz Scherer received the B.Sc. and M.Sc. degree in electrical engineering and information technology from ETH Zürich in 2018 and 2020, respectively, where he is currently pursuing a Ph.D. degree at the Integrated Systems Laboratory. His current research interests include the design of ultra-low power and energy-efficient circuits and accelerators as well as system-level and embedded design for machine learning and edge computing applications. Contact him at scheremo@iis.ee.ethz.ch.

Tim Fischer received the B.Sc. and M.Sc. degree in electrical engineering and information technology from ETH Zürich in 2018 and 2021, respectively, where he is currently pursuing a Ph.D. degree at the Integrated Systems Laboratory. Contact him at fischeti@iis.ee.ethz.ch.



Luca Benini holds the chair of digital Circuits and systems at ETHZ and is Full Professor at the Università di Bologna. He received a Ph.D. from Stanford University. Dr. Benini's research interests are in energy-efficient parallel computing systems, smart sensing micro-systems and machine learning hardware. He has published more than 1000 peer-reviewed papers and five books. He is a Fellow of the IEEE, of the ACM and a member of the Academia Europaea. He received the IEEE Mac Van Valkenburg award in 2016 and the ACM/IEEE A. Richard Newton Award in 2020. Contact him at lbenini@iis.ee.ethz.ch.