A 135-mW 1.70TOPS Sparse Video Sequence Inference SoC for Action Classification

Thomas Chen, *Student Member, IEEE*, Ching-En Lee, *Student Member, IEEE*, Chester Liu, *Student Member, IEEE*, and Zhengya Zhang, *Senior Member, IEEE*

***Abstract*— An inference system-on-chip (SoC) is designed to extract spatio-temporal features from videos for action classification. The SoC contains an inference core that implements a recurrent neural network in three processing layers. High sparsity is enforced in each layer of processing, reducing the complexity by two orders of magnitude and allowing multiply accumulates to be replaced by select accumulates. Spatio-temporal kernel and activation compression are applied to reduce memory by 43% and 64%, respectively. The design is demonstrated in a 2.53-mm^2 40-nm CMOS chip with an OpenRISC core, providing control and classification. With the inference core extracting spatio-temporal features and a soft-max classifier programmed on the OpenRISC core, the SoC classifies KTH Human Action Data Set at a 76.7% accuracy. At 0.9 V and 250 MHz, the SoC achieves 1.70TOPS, dissipating 135 mW.**

***Index Terms*— Action classification, inference processor, recurrent neural network (RNN), sparse coding, video processing.**

I. Introduction

OBJECT detection in videos is employed in a wide range of applications from the smart user interface to surveillance and autonomous navigation. Due to the demanding resolution and frame rate of videos, real-time object detection has been a challenge. Designing real-time object detection on embedded platforms is especially difficult due to the limited energy source available on embedded platforms.

State-of-the-art object detection accelerators [1]–[5] have been designed based on scale invariant feature transform (SIFT) [6], speeded-up robust features (SURF) [7], and deformable part models (DPM) [8] algorithms. These popular algorithms extract 2-D features from images and compare them with features stored in a database [6], [7] or perform classifications [8] on the features to recognize objects. The accelerators target real-time videos, but the base operations (OPs) are on 2-D images.

Video sequence classification, or action classification, operates on sequences of video frames to extract activity or action information from videos. Video sequence classification relies on extracting spatio-temporal features and performing classification on the spatio-temporal features, thus it is expected to demand more computation than the 2-D processing of videos.

Classic video sequence classification relies on engineered features, such as cuboid [9], space-time Harris [10], and Hessian [11]. Each feature selection is tailored to a specific task but may not deliver the best performance for every task. It is desirable to use automatically learned features that are most suitable for the data. An auto-encoder is one such approach that automatically learns sparse, shift-invariant spatio-temporal features [12]. Sparse coding is a similar approach that adapts an overcomplete dictionary of space-time functions (features) to represent time-varying natural images with high sparsity [13]. The space-time features resemble the motion-selective receptive fields (RFs) of simple cells in the mammalian visual cortex, suggesting that the approach may be at work in the visual cortex [13].

In this work, we adopt a sparse coding approach called locally competitive algorithm (LCA) [14]. LCA is formulated as a compressed sensing method. When applied to videos, LCA learns the spatio-temporal RFs (STRFs) and encodes inputs using a sparse set of STRFs. As such, LCA is highly effective in reducing the input size, allowing the most salient STRFs to be extracted for classification.

LCA can be mapped to a spiking recurrent neural network (RNN) [15], [16]. Implemented using iterative forward projection and backward reconstruction, a video sequence inference processor based on spiking RNN can extract spatio-temporal RFs (STRFs), i.e., spatio-temporal features, from videos. The extracted STRFs can, in turn, enable action classification [17] and motion tracking [18] tasks.

Due to the large video data size, spatio-temporal, and iterative processing, the computational requirement of the video sequence inference RNN is high. Even for relatively small-scale processing of a $6 \times 6 \times 64$ video slice using 192 STRFs costs 200M multiply accumulates (MACs). To enable a practical implementation, we adopt a residual formulation of the RNN [19] and apply an algorithm transformation by rectifying the residuals after each inference iteration to ternary spikes without costing classification accuracy. After the transformation, the intermediate data through the compute stages, i.e., activations and residuals, become spikes with a sparsity level well above 90%. The transformed algorithm leads to a sparse, all-spiking video inference processor design that reduces the computational complexity from 200M MACs to 4M select accumulates (SAs) per iteration, making it possible

Manuscript received November 1, 2018; revised March 2, 2019; accepted March 11, 2019. Date of publication April 24, 2019; date of current version June 26, 2019. This work was supported in part by Defense Advanced Research Projects Agency (DARPA), in part by the Systems On Nanoscale Information Fabrics (SONIC) Center, in part by Intel, and in part by National Science Foundation (NSF) under Grant GRFP DGE 1256260. This paper was approved by Associate Editor Edith Beigne. *(Corresponding author: Thomas Chen.)*

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122 USA (e-mail: tcchen@umich.edu; lchingen@umich.edu; cwhliu@umich.edu; zhengya@umich.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSSC.2019.2907406

to support video processing in real time at reasonable power consumption. To reduce the large on-chip storage, we apply non-uniform delta encoding on the highly redundant STRFs and compressed column storage (CCS) on the highly sparse activations to reduce their memory size by 43% and 64%, respectively.

The design is demonstrated in a 2.53-mm^2 40-nm inference system-on-chip (SoC) that integrates a video sequence inference core and an OpenRISC core. The chip is measured to achieve 1.70TOPS at 0.9 V and 250 MHz, dissipating 135 mW at room temperature. With the video sequence inference core extracting the activation response of STRFs, a soft-max classifier programmed on the OpenRISC core achieves a 76.7% classification accuracy on the six-class KTH Human Action Data Set [20].

The rest of this paper is organized as follows. Section II provides an overview of the baseline inference algorithm, and Section III shows how the algorithm is transformed into a sparse, all-spiking formulation to reduce its implementation cost. Section IV presents the design details of each compute layer and memory and quantifies the performance and energy gain. Section V shows the chip implementation and measured results, and Section VI concludes this paper.

II. Video Inference Algorithm Formulation

In this paper, we adopt the LCA algorithm [14] to perform compressed sensing of videos. LCA can be mapped to a recurrent network of spiking leaky integrate-and-fire neurons, where a neuron's potential increases due to input excitation, and decreases due to inhibition by neighboring neurons. The LCA algorithm is described by the following equation:

$$\Delta \mathbf{u} = \eta[\Phi^T \mathbf{x} - (\Phi^T \Phi - \mathbf{I})\mathbf{a} - \mathbf{u}] \qquad (1)$$
$$\mathbf{a} = T_\lambda(\mathbf{u})$$

where $\mathbf{u}$ is the neuron potential; $\Delta\mathbf{u}$ is the potential update; η is the update step size; Φ is the RFs of neurons, also known as the dictionary; $\mathbf{x}$ is the input; $\mathbf{a}$ is the neuron activation; and $\mathbf{I}$ is the identity matrix. $T_\lambda()$ is a binary threshold function and it outputs 1. If its input exceeds λ or 0 otherwise. Dictionary Φ and threshold λ are trained by stochastic gradient descent, which aims to maximize encoding accuracy and the sparsity of neuron activations.

In performing inference on video inputs, an input is divided to 3-D segments for processing. In (1), $\mathbf{x}$ is a time series of T number of $X \times Y \times D$ consecutive and overlapping video segments, as shown in Fig. 1. The dictionary Φ is a collection of N RFs, and each RF is a $X \times Y \times D$ spatio-temporal feature, known as STRF. $\mathbf{u}$, $\Delta\mathbf{u}$, and $\mathbf{a}$ are collections of N neurons' potentials, potential updates, and activations, respectively, over T time steps. Mathematically, $\mathbf{x}$ is a $V \times T$ matrix, where $V = XYD$; Φ is a $V \times N$ matrix; $\mathbf{u}$, $\Delta\mathbf{u}$ and $\mathbf{a}$ are $N \times T$ matrices.

The inference described by (1) consists of four functional steps.

1) *Charge*: Input $\mathbf{x}$ is projected to the feature space as described by $\Phi^T\mathbf{x}$. The projection can be understood as encoding the input $\mathbf{x}$ in STRFs, i.e., extracting STRFs from the input. The projection increases, or charges, the neuron potential.
2) *Compete*: To maintain sparse activation, active neurons suppress other neurons. The inhibition weight between a pair of neurons is computed by correlating their STRFs, i.e., $\Phi^T\Phi$. Self-inhibition is removed by subtracting $\mathbf{I}$. The closer the two neurons' STRFs, the stronger the inhibition between the two neurons. Neuron activations trigger inhibitions as described by $(\Phi^T\Phi - \mathbf{I})\,\mathbf{a}$.
3) *Leak*: Neuron potential decreases over time, and the leakage is proportional to the potential.
4) *Activate*: Neuron potential is thresholded to generate binary spikes.

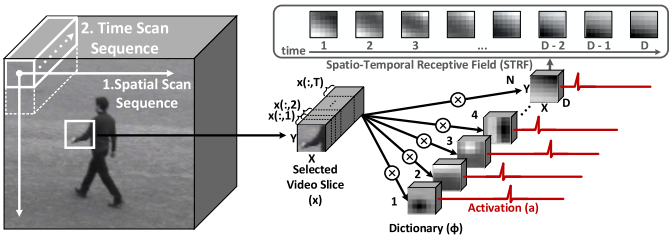


Fig. 1. Illustration of video inference processing.

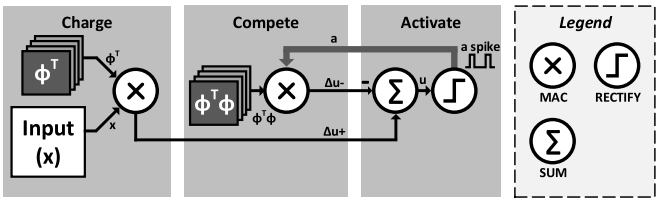


Fig. 2. Baseline implementation of video inference.

TABLE I
Baseline Implementation Complexity of One Iteration of Inference

Function	Storage (# weights)	Compute (# MACs)
Dictionary storage	VN	-
Inhibitary weight storage	N^2	-
Charge step	-	NVT
Compete step	-	N^2TI
Total	$VN + N^2$	$NVT + N^2TI$

The four steps mentioned earlier constitute one iteration of inference. Given an input $\mathbf{x}$, the inference is done by iterating the four steps until convergence. It is common to use a fixed number of iterations I. The baseline implementation is outlined in Fig. 2, where the leak step is omitted for simplicity.

The implementation complexity of one iteration of inference is analyzed and the results are listed in Table I. The dictionary storage requires VN entries. The inhibitory weights are computed by $\Phi^T\Phi - \mathbf{I}$, requiring N^2V MACs. The N^2 inhibitory weights can be computed once and stored in memory.

In every iteration of inference, the charge step requires NVT MACs. Because the two inputs Φ^T and $\mathbf{x}$ to the charge

TABLE II
IMPLEMENTATION COMPLEXITY OF ONE ITERATION OF INFERENCE USING RESIDUAL APPROACH

Function	Storage (# weights)	Compute (# MACs)
Dictionary storage	VN	-
Residual step	-	$NVTI$
Charge step	-	$NVTI$
Total	VN	$2NVTI$

step do not change between iterations, the charge is computed only once per inference regardless of the number of iterations. The compete step is driven by neuron activations, requiring N^2T MACs per iteration for I iterations.

Typically, the number of neurons (N) ranges from hundreds and more for practical applications, and video inference can be particularly challenging due to its large dimensionality and real-time processing requirement. A silicon implementation requires a large area and power.

III. SPARSE AND ALL-SPIKING INFERENCE FORMULATION

Video data are large, but it also contains high redundancy, especially from frame to frame. The redundancy offers opportunities for significant complexity reduction in storage and compute. The sparse coding algorithm also lends itself to an efficient implementation by exploiting its inherent sparsity.

We formulate the algorithm such that all steps operate on spiking inputs. As a result, expensive MACs are replaced by efficient SAs; and OPs are skipped if no spikes are present.

A. Rectification and Sparsification

The LCA equation can be reformulated by factoring the term Φ^T in (1)

$$\begin{aligned} \Delta\mathbf{u} &= \eta[\Phi^T(\mathbf{x}-\Phi\mathbf{a})+\mathbf{a}-\mathbf{u}] \\ \mathbf{a} &= T_\lambda(\mathbf{u}). \end{aligned} \tag{2}$$

The reformulated inference, first proposed in [19], can be interpreted as having four steps: residual, charge, leak, and activate. The leak and activate steps are identical to the original formulation. The residual and charge steps are described below.

1) *Residual*: The input $\mathbf{x}$ is reconstructed, $\hat{\mathbf{x}} = \Phi\mathbf{a}$. The reconstruction is subtracted from the input to obtain the residual $\mathbf{r} = \mathbf{x} - \hat{\mathbf{x}}$.
2) *Charge*: The residual is projected to the feature space, $\mathbf{c} = \Phi^T\mathbf{r}$.

The residual form removes the storage of inhibitory weights and replaces it by computing the weights on the fly. As a result, the storage required is smaller, but the compute complexity poses a challenge, as shown in Table II. To reduce complexity, we propose to quantize the residuals. If the residuals can be quantized to binary spikes (1, -1), the computational complexity of the charge layer can be significantly simplified. However, as shown in Fig. 3(a), the binary quantization has a large impact on the classification accuracy when the activation density is low. With 0 being the binary threshold, small noise values near 0 are amplified, preventing convergence and degrading accuracy.

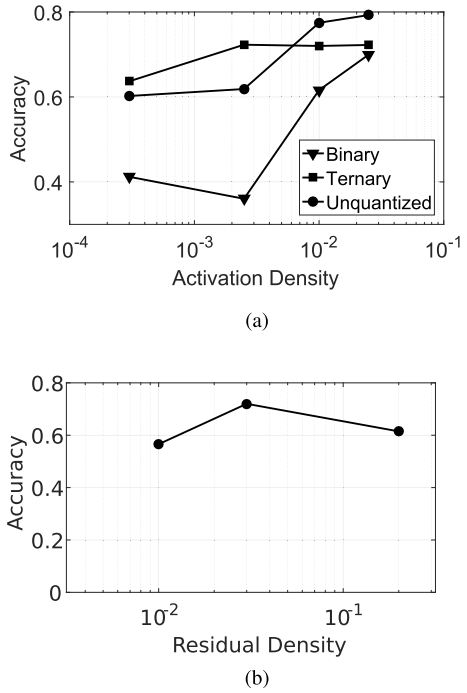


Fig. 3. Effect of (a) activation density and (b) residual density on classification accuracy.

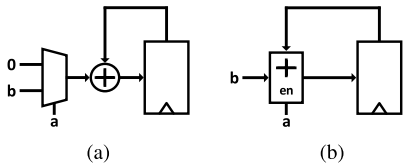


Fig. 4. SA implementations. (a) Select-add. (b) Skip-add.

To fix this problem, we propose a min/max rectification to the residuals to quantize the residuals to ternary spikes. The residual rectification is done by applying thresholds of λ_r and $-\lambda_r$ to quantize the residuals to 1 (above λ_r), 0 (between $-\lambda_r$ and λ_r), and -1 (below $-\lambda_r$). With appropriate threshold choices, the ternary quantization outperforms binary quantization by a large margin and can even match the unquantized accuracy, as shown in Fig. 3(a). The updated equation is given in (3), where T_{λ_r} is the min/max rectification function.

$$\begin{aligned} \Delta\mathbf{u} &= \eta[\Phi^T T_{\lambda_r}(\mathbf{x}-\Phi\mathbf{a})+\mathbf{a}-\mathbf{u}] \\ \mathbf{a} &= T_\lambda(\mathbf{u}). \end{aligned} \tag{3}$$

A key advantage of quantizing the residuals to binary or ternary spikes is that the multiplication by these quantized values and accumulating the partial sums no longer requires a MAC. Instead, a simpler SA can be used. Suppose a is binary (0 or 1), multiplying a by b followed by accumulation can be done using an SA that is implemented as in Fig. 4(a), where a is used as the select input in the multiplexer to choose whether 0 (if a is 0) or b (if a is 1) is accumulated by the adder. The accumulated sum is saved in a register. Alternatively, SA can

TABLE III

IMPLEMENTATION COMPLEXITY OF ONE ITERATION OF INFERENCE USING SPARSE AND ALL-SPIKING APPROACH

Function	Storage (# weights)	Compute (# SAs)
Dictionary storage	VN	-
Residual step	-	$NVTS_aI$
Charge step	-	$NVTS_rI$
Total	VN	$NVT(S_a+S_r)I$

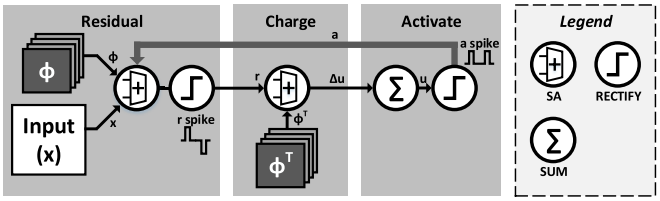


Fig. 5. Sparse, all-spiking implementation of video inference.

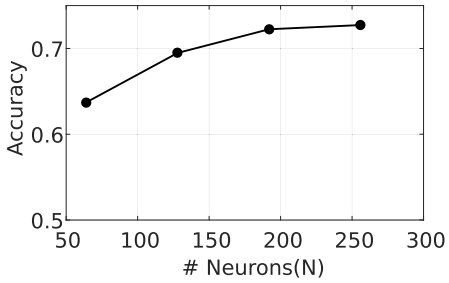


Fig. 6. Effect of number of neurons on classification accuracy.

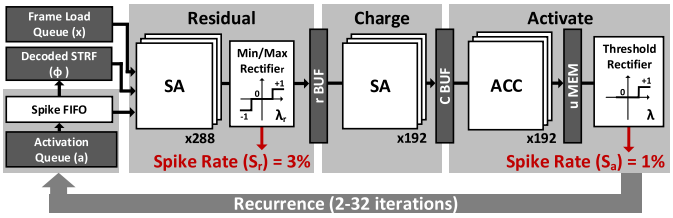


Fig. 7. Architectural sketch of three-layer implementation of video inference processor.

be implemented using a skip-add as shown in Fig. 4(b), where a is used as the enable input to the adder to decide whether to accumulate b (if a is 1) or not (if a is 0). Although the example was shown for the binary spike case, the implementation can be easily modified to support ternary spikes.

Similar to the residual rectification, neuron activation is obtained by rectifying neuron potentials to produce sparse, binary spikes. Binary spikes allow the reconstruction in the residual step to be implemented using SAs, presenting another opportunity for significant complexity and power reduction.

Taking advantage of both residual rectification and neuron activation, the sparse, all-spiking approach can be implemented as shown in Fig. 5. It features a lower complexity compared to the conventional residual approach as summarized in Table III, where S_a and S_r refer to the density, or fraction of non-zero entries, in neuron activations and the residuals, respectively. The sparser the inputs (i.e., the lower density), the less the amount of effectual workload. However, sparsifying the inputs (activations or residuals) can degrade the classification accuracy. The effects are illustrated in Fig. 3. The activation density $S_a = 1\%$ and residual density $S_r = 3\%$ are nearly optimal for the KTH Human Action Data Set [20]. Below or above the optimal density results in under- or over-representation of the input, and degradation in classification accuracy.

B. Design Specification and Parameter Settings

We present a prototype video inference processor to demonstrate the sparse, all-spiking LCA approach. The prototype design, including the model and parameters, is based on the KTH data set [20]. The inference processor takes video inputs in $6 \times 6 \times 64$ slices, and divides into 57 $6 \times 6 \times 8$ ($T = 57$, $V = 6 \times 6 \times 8 = 288$) consecutive and overlapping segments for processing.

The optimal $X - Y$ patch size is determined by the size of features for a data set. For the KTH data set, 6×6 patch size provides the best accuracy. More spatial overlap (smaller spatial stride) produces better results. However, in the prototype design, we chose no overlap to reduce the processing complexity. It degrades accuracy by only 2%.

The optimal STRF depth is determined by the action sequence duration for a data set. For the KTH data set, a larger depth yields better accuracy. We used a depth of 8, below which the accuracy drops by about 2% per depth reduction of 1. Temporal overlap (small temporal stride) is essential for guaranteeing a good accuracy, e.g., increasing the temporal stride from 1 to 4 reduces the accuracy by more than 8%. In the prototype design, we chose a stride of 1.

The number of neurons, i.e., the number of STRFs, is dependent on the input size and it affects the classification accuracy as shown in Fig. 6. In testing the prototype design, we employ 192 neurons ($N = 192$). Each neuron's STRF is sized $6 \times 6 \times 8$. The STRF weights are quantized to 8 bits. Simulations show that six to eight iterations are sufficient, beyond which the accuracy saturates. We used eight iterations ($I = 8$) for measurement in this work. Based on the STRFs extracted from video, action classification can be performed.

To realize this prototype chip, 54-kB memory is needed to store the dictionary. The density of neuron activations and residuals is optimally set to $S_a = 1\%$ and $S_r = 3\%$, respectively. The sparse, all-spiking approach reduces the number of OPs per inference from 200M MACs to 4M SAs, which translates to a significant reduction in complexity and power consumption.

IV. DESIGN OF VIDEO INFERENCE PROCESSOR

The video inference processor is made of three compute layers: residual layer, charge layer, and activate layer as illustrated in Fig. 7. Each layer corresponds to one step outlined in Section III (the leak step is absorbed as part of the charge layer). The residual and charge layers are the workhorse of the inference processor. The inputs to the residual layer are sparse binary neuron spikes. The inputs to the charge layer

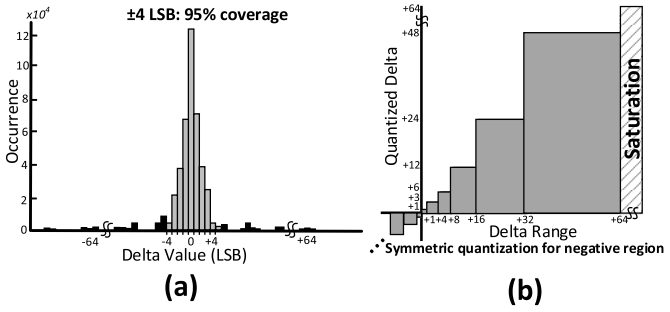


Fig. 8. (a) Distribution of deltas between frames of STRFs. (b) Non-uniform quantization of deltas.

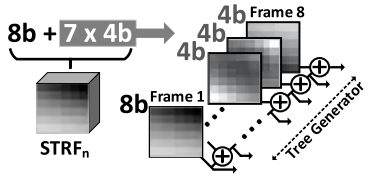


Fig. 9. Tree generator for decompressing delta-encoded STRF.

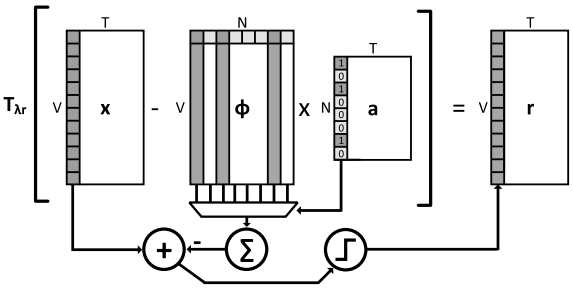


Fig. 10. Visualization of residual compute.

are sparse residuals in the form of ternary spikes. Inputs are streamed through the three layers and back to the residual layer for the next iteration.

A. Dictionary Compression and Non-Uniform Quantization

The dictionary Φ and its transpose Φ^T are accessed by the residual layer and the charge layer, respectively. Since the two layers operate concurrently in a streaming pipeline and the dictionary elements' access orders are different, both Φ and Φ^T are stored on-chip, requiring 108 kB of memory for the prototype design. Due to the high access bandwidth needed for highly parallel processing, the dictionary memory is divided into banks, sacrificing the storage efficiency. The dictionary memory alone is estimated to take 2 mm^2 chip area in a 40-nm CMOS technology.

In the prototype design, each dictionary element is a $6\times6\times8$ 8-bit STRF that is essentially a sequence of eight 6×6 frames. Redundancy exists between consecutive frames, making it possible to compress each STRF to save memory, chip size, and power. In Fig. 8(a), we plot the distribution of the pixel-by-pixel differences between consecutive frames of STRFs that are learned by training on the KTH Data Set. The results show that 95% of the pixel-by-pixel differences cover a narrow range of only four LSBs.

The similarity between consecutive frames motivates the delta encoding of STRFs by storing the first 6×6 8-bit frame as the anchor frame, and subsequent frames as 4-bit deltas to the previous frame. The delta encoding reduces the dictionary storage by 43%.

Although 4 bits are sufficient to cover 95% of the deltas, a better result requires a larger coverage. To keep deltas to 4 bits while increasing the range of coverage, we propose the non-uniform quantization of deltas as shown in Fig. 8(b). The non-uniform quantization is specifically tailored to the delta distribution: smaller quantization step sizes are used at the lower end, and increasingly larger quantization step sizes are used toward the higher end to keep the number of quantization steps to 15.

The delta-encoded dictionary elements need to be decompressed before being used in compute. We employ a tree generator, as shown in Fig. 9 to take the anchor frame as the base, and sequentially add the deltas to recover the remaining frames. With delta encoding and taking into account the overhead of tree generator, the dictionary memory storage in our prototype design, including compression and decompression, occupies 27% less area compared to the baseline.

B. Residual Layer

The residual layer computes the reconstruction $\hat{\mathbf{x}}$ $(V\times T)$ by multiplying Φ $(V\times N)$ by $\mathbf{a}$ $(N\times T)$. Recall that since $\mathbf{a}$ consists of binary activations, the matrix multiplication is done by SAs. The input $\mathbf{a}$ is provided to the residual layer one column at a time, as the time-series output of N neurons from the activate layer. Since activations are sparse, we use a spike detector to skip 0 activations and provide the addresses of the activated neurons.

The residual layer computation is illustrated in Fig. 10. For each column of $\mathbf{a}$, the spike detector looks at a block of entries at a time and finds the address of the first entry that is 1. Suppose in processing column i of $\mathbf{a}$, the spike detector outputs j as the first entry in column i that is 1, then column j of Φ is read from memory, decompressed by the tree generator, and accumulated by the SA array as the temporary output of column i of $\hat{\mathbf{x}}$. We employ an array of V SAs to compute one vector accumulation at a time. The process continues with the spike detector providing the next non-zero entry. Upon completion, the reconstruction is subtracted from the input $\mathbf{x}$; and the results are rectified to obtain the residuals. Since the reconstruction is computed column by column, the residuals are obtained column by column and provided to the charge layer in this order.

An implementation of the residual layer is shown in Fig. 11. The number of actual accumulations done by the SA array is $NVTS_a$, with S_a being the density of 1s in $\mathbf{a}$. Since V SAs operate in parallel, the residual layer takes on average $NTS_a = 192\times57\times1\% = 109$ cycles.

C. Charge Layer

The charge layer computes the charge $\mathbf{c}$ by multiplying Φ^T $(N\times V)$ by $\mathbf{r}$ $(V\times T)$. Since $\mathbf{r}$ is a collection of ternary spikes $\{0, -1, 1\}$, the matrix multiplication is also done by SAs.

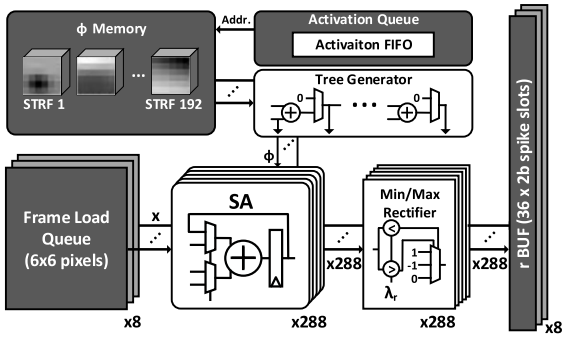


Fig. 11. Residual layer design.

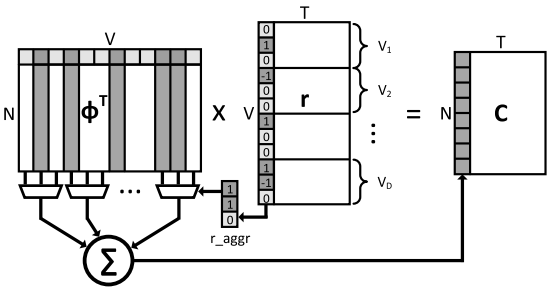


Fig. 12. Visualization of charge compute.

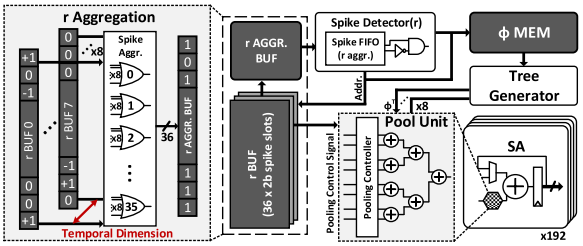


Fig. 13. Charge layer design.

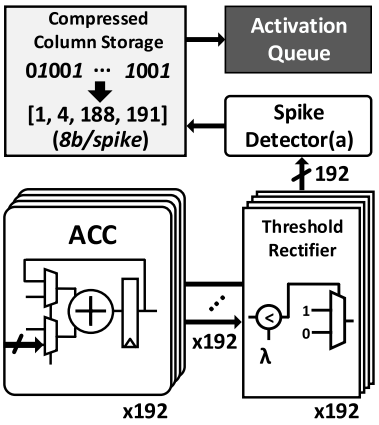


Fig. 14. Activate layer design.

A similar architecture as the residual layer can be designed to implement the charge layer. The input $\mathbf{r}$ is provided one column at a time, as shown in Fig. 12. In processing a column of $\mathbf{r}$, a non-zero entry triggers the accumulation of a column of Φ^T to compute $\mathbf{c}$. An array of N SAs is employed. The number of actual accumulations done by the SA array is $NVTS_r$. Since N SAs operate in parallel, the charge layer takes VTS_r to complete. Given the prototype specification, the charge layer takes 492 cycles.

To balance the layers, we apply temporal aggregation to shorten the latency of the charge layer. Each column of $\mathbf{r}$ represents a $X \times Y \times D$ frame. We compress $\mathbf{r}$ to $\mathbf{r}_a$ by pooling pixels at the same (x, y) location across D frames in a time series. If at least one of the D pixels is non-zero, pooling will output 1 for the pixel. After pooling, each entry of $\mathbf{r}_a$ represents an "aggregated" pixel i (in the xy plane) across D frames. Note that temporal aggregation does not make use of any approximation. It essentially collects a vector of inputs and applies parallel processing. The technique has no impact on the encoding fidelity or classification accuracy.

Temporal aggregation enables shorter latency. As shown in Fig. 13, $\mathbf{r}_a$ is passed to a spike detector to output the first entry that is non-zero. As illustrated in Fig. 12, suppose the spike detector outputs address i (in the xy plane), the address is used to read the D columns of Φ^T that correspond to pixel i, and the D $\mathbf{r}$ values that are associated with pixel i. The D columns of Φ^T are vector summed by the pool units located inside the SA array, as shown in Fig. 13, with the D $\mathbf{r}$ values as the control bits that determine whether the respective columns are zeroed, added, or subtracted.

The aggregate processing increases the parallelism by a factor of D. The temporal aggregation the D frames to one aggregate frame increases the density of 1s in the aggregate frame. If the D frames are completely independent, the density S_r' increases by D. However, the D frames belong to a time series and are highly correlated. In the prototype design, the density increases from 3% to 5%. With temporal aggregation and aggregate processing, the charge layer latency is reduced to $XYTS_r' = 6 \times 6 \times 57 \times 5\% = 103$ cycles on average for the prototype design.

D. Activate Layer

The activate layer accumulates potential updates $\Delta\mathbf{u}$ $(N \times T)$ to compute new neuron potentials. $\Delta\mathbf{u}$ is received column by column from the charge layer. The activate layer uses an array of N accumulators to update one column of potentials at a time. The potentials are thresholded to obtain binary activations $\mathbf{a}$.

The activations $\mathbf{a}$ $(N \times T)$ are binary and sparse. As described in Section IV-B, $\mathbf{a}$ is fed to a spike detector to locate the non-zero entries for processing in the residual layer. The spike detector can be used to encode $\mathbf{a}$ in a CCS format, referring to storing only the addresses of non-zero entries in every column, as illustrated in Fig. 14.

Due to high sparsity, we can limit the number of non-zero entries in a column to a small fixed number. Simulations show that at least four non-zero activations need to be stored to ensure a high accuracy. If only two non-zero activations are stored, the accuracy is reduced by 10%. In the prototype design, we allow up to eight non-zero activations to be stored. Additional non-zero entries are dropped with negligible

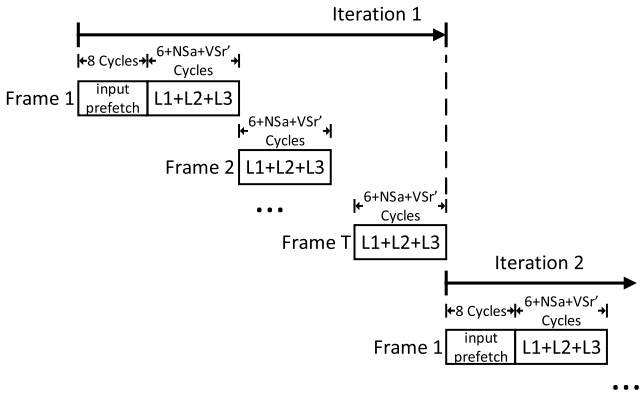


Fig. 15. Timing illustration.

impact on the accuracy due to the extremely low likelihood of occurrence. CCS effectively reduces the storage by 64%.

Putting the three layers together, the timing diagram for processing one $6 \times 6 \times 64$ input is illustrated in Fig. 15. The input is divided into $T = 57$ temporally overlapped frames to be dispatched to the three-layer processing in series. The processing is repeated for $I = 8$ iterations. Input data stream through the layers in sequence.

E. Summary of Design Optimizations

From Sections IV-A–IV-D, we present the design techniques based on the prototype specification. The techniques are generally applicable and not limited to the given specification.

To quantify the benefits of the design techniques, we synthesized a baseline design in 40-nm CMOS, along with design points after every step of the optimization. The results are shown in Fig. 16. The baseline design employs a V-parallel MAC array in the residual layer, an N-parallel MAC array in the charge layer, and an N-parallel accumulator array in the activate layer. The design uses dense processing without spike detectors, and the residuals are not rectified. The baseline design reflects a standard parallel implementation without any sparsity or spiking optimizations. The latency of one iteration of processing is 211k cycles. The design is estimated to occupy 2.83 mm^2 and consume 168 mW.

The residual and charge layers account for the majority of the workload. Introducing sparsity optimizations has a major impact on the performance and the energy efficiency. In the first step of the optimization, we take advantage of sparse binary neuron activations to change the MAC array in the residual layer to an SA array and use a spike detector to skip computations when activation is 0. The area and power increase by 1% and 4%, respectively, to support the net increase of the spike detection overhead minus the savings of the SA array, and the processing latency decreases by 36%. The latency is now entirely dominated by the charge layer.

In the second step, we apply ternary rectification to the residuals to change the MAC array to an SA array, and apply temporal aggregation to the charge layer. The area and power are reduced by 7% and 11%, respectively, and the latency is reduced by 32×.

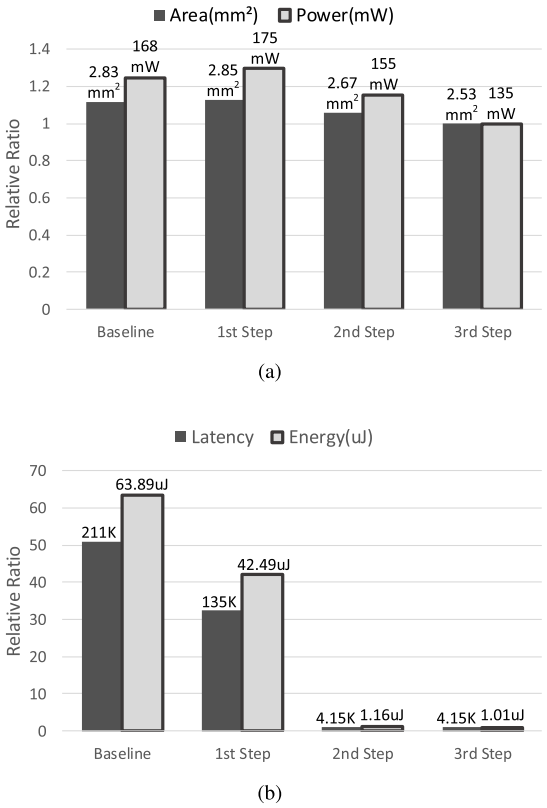


Fig. 16. (a) Area and power and (b) latency and energy after three design optimization steps: 1) sparse activation; 2) residual rectification; and 3) compressed activations.

In the third step, we compress the activations stored in the activate layer. The compression results in 5% area reduction and 13% power reduction.

In total, the three optimization steps increase the throughput by 51×, reduce the energy by 63×, and the area is reduced by 11%. Assume the KTH data set with $6 \times 6 \times 64$ inputs and the following parameter settings: $N = 192$, $X \times Y \times D = 6 \times 6 \times 8$, temporal stride of 1, spatial stride of 6, $I = 8$, $S_a = 1\%$, and $S_r = 3\%$. At a clock frequency of 240 MHz, the real-time processing of 1080p HD video at 60 frames per second (fps) requires the processing of a $6 \times 6 \times 64$ input to be completed in 4.16k cycles. The optimizations proposed in this paper are crucial for meeting this latency requirement.

Finally, note that activation sparsity and ternary rectification of residuals caused most of the accuracy loss as shown in Fig. 3. However, these two techniques also contributed most of the performance and energy efficiency gain, as shown in Fig. 16.

V. Prototype Implementation, Measured Results, and Comparison

We design a prototype SoC for video inference applications. The system block diagram is shown in Fig. 17. The core of the SoC chip is the video inference processor that is made of three compute layers and memory to store dictionary, neuron potentials, and input video frames for testing. The SoC also consists of an OpenRISC processor for programming,

TABLE IV
ACTION CLASSIFICATION RESULTS OF KTH HUMAN ACTION DATABASE

	Boxing	Clapping	Waving	Jogging	Running	Walking	Average
On-chip softmax classifier	70.0%	68.4%	85.0%	73.7%	94.4%	70.0%	76.7%
Off-chip SVM classifier	85.0%	78.9%	85.0%	73.7%	94.4%	80.0%	82.8%

TABLE V
COMPARISON WITH PRIOR WORK

	JSSC'15 Lee [3]	JSSC'17 Suleiman [5]	JSSC'18 Liu [16]	This Work
Application	Object matching	Object detection	Feature & depth extraction	Action classification
Algorithm	Vocabulary forest	Deformable parts model	LCA	LCA
Process	65nm	65nm	40nm	40nm
Core Area (mm^2)	2.3	12.8	2.56	2.53
Voltage (V)	1.2	0.77 - 1.11	0.6 - 0.9	0.65 - 0.9
Frequency (MHz)	250	62.5 - 125	120 - 380	50 - 250
Power (mW)	27.6	58.6 - 217	45 - 257	29.2 - 135
Performance (TOPS)	0.191	0.068 - 0.137	0.227 - 0.718 [a]	0.340 - 1.700 [b]
Power Efficiency (TOPS/W)	6.920	1.169 - 0.624	5.038 - 2.793	14.946 [c] - 12.583

[a] 1 Operation (OP) is defined as an 8b MAC. [b] 1 Operation (OP) is defined as an 8b add. (including skipped OPs)
[c] Power efficiency is 14.946TOPS/W at 0.65V, 100 MHz, (including skipped OPs)

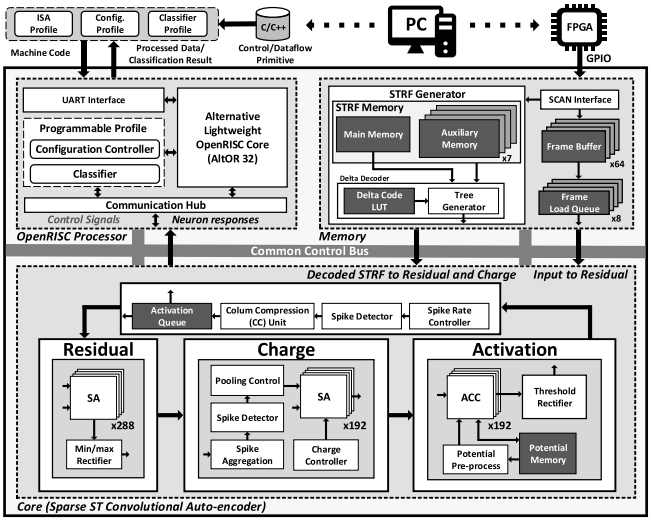


Fig. 17. System-level design of video inference processor.

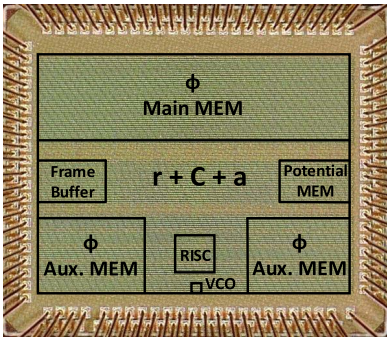


Fig. 18. Microphotograph of the video inference SoC chip in 40-nm CMOS.

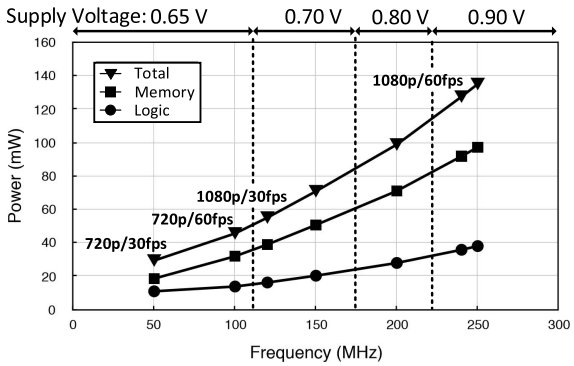


Fig. 19. Measured power and performance of the video inference SoC chip.

control, configuration, and classification. Through the OpenRISC processor, the video inference processor is configurable with several settings: 64, 128, or 192 neurons (N), frame size ($X \times Y$) from 1 to 36, and depth (D) from 1 to 8.

The video inference SoC chip is implemented in 40-nm CMOS, occupying 3.98 mm^2. The core area measures 1.77 mm × 1.43 mm. The chip photograph is shown in Fig. 18. The chip is tested for the KTH data set with $6 \times 6 \times 64$ inputs and the following parameter settings: $N = 192$, $X \times Y \times D = 6 \times 6 \times 8$, temporal stride of 1, spatial stride of 6, $I = 8$, $S_a = 1\%$, and $S_r = 3\%$. At room temperature, the chip is measured to achieve an effective performance of 1.70TOPS (including skipped OPs) at 0.9 V and 240 MHz. The performance meets the 60 fps 1920 × 1080 HD video data rate, while dissipating 135 mW. The measured power and performance at room temperature are shown in Fig. 19.

The six-class KTH Human Action Data Set [20] is used for action classification testing, with 600 samples and a training/testing split ratio of 5:1. Using the core extracting the activation response of STRFs and a soft-max classifier

programmed on the OpenRISC processor, the SoC achieves a 76.7% classification accuracy.

We also designed an support-vector machine (SVM) classifier based on a feed-forward network with two hidden layers of 40 and 50 neurons. The inputs to the classifier are extracted STRF features, i.e., spiking neuron outputs of the feature extraction network; and the outputs are the action class labels. The SVM is trained using a conjugate gradient method. The SVM classifier achieves an 82.8% accuracy as shown in Table IV.

In software and full precision, the state of the art for the KTH data set classification has now reached 92% accuracy [21]. The approach used differential gating of long short-term memory (LSTM), and the LSTM model consists of 450 input units, 300 memory cell state units, and 6 output units. There is not yet a clear path toward an efficient implementation of such a large model. In comparison, we sacrificed about 10% accuracy to obtain an efficient hardware implementation.

In Table V, this work is compared with video processors for key point matching [3] in SIFT-based object recognition, and DPM-based object detection [5], as well as a convolutional sparse coding processor for feature and depth extraction [16]. Direct comparisons are not possible due to the major differences in algorithms and applications. This work is the first video action classification processor that extracts spatio-temporal features from video for sequence classification. The 2.53 mm^2, 40-nm test chip achieves up to 1.70TOPS at a power efficiency above 12.5TOPS/W (including skipped OPs). The performance and power efficiency are competitive with the other designs. Compared to [16] that used a similar algorithm for feature extraction and depth extraction, this work demonstrates higher performance and power efficiency.

VI. Conclusion

We present an inference SoC for video sequence classification based on an RNN implementing LCA, a neuro-inspired compressed sensing algorithm. Due to the large video data size, spatio-temporal, and iterative processing, the computational requirement of the RNN is high. We adopt a residual form of the LCA algorithm and apply a transformation by rectifying the residuals after each inference iteration to ternary spikes.

The algorithm reformulation leads to a sparse all-spiking RNN architecture realized in three layers: residual layer, charge layer, and activate layer. All layers are implemented primarily in SAs. Data are seamlessly streamed across the layers in iterations. To balance the processing layers and avoid stalling, we use a temporal aggregation and aggregate processing technique to shorten the processing latency of the slowest charge layer. To reduce the chip area and power, we apply delta compression and non-uniform quantization to STRFs to reduce the memory by 42% and CCS encoding to sparse activations to reduce the memory by 64%. In all, the algorithm and architecture techniques increase the processing throughput and reduce the energy by 51× and 63×, respectively, while the area is kept nearly constant.

The design is prototyped in a 2.53 mm^2 40-nm CMOS video inference SoC chip. The chip is measured to achieve 1.70TOPS (including skipped OPs) at 0.9 V and 250 MHz, dissipating 135 mW. Tested with the six-class KTH Human Action Data Set, the chip provides a 76.7% classification accuracy.

Not every video application in practice can directly benefit from a design that supports $XY \leq 36$. As a small research prototype, we chose $XY = 36$ to target the KTH data set. Even for this relatively small data set, multiple optimizations are needed to keep the hardware complexity within bounds. Video sequence classification is a demanding task. For larger and practical applications, we expect more substantial compute resources to be needed. The same optimizations demonstrated in this work are equally applicable to larger and more demanding applications.

Acknowledgment

The authors would like to thank Prof. B. Olshausen for his advice.

References

[1] J. Oh *et al.*, "A 320 mW 342 GOPS real-time dynamic object recognition processor for HD 720 p video streams," *IEEE J. Solid-State Circuits*, vol. 48, no. 1, pp. 33–45, Jan. 2013.

[2] G. Kim, J. Oh, S. Lee, and H.-J. Yoo, "An 86 mW 98 GOPS ANN-searching processor for Full-HD 30 fps video object recognition with Zeroless locality-sensitive hashing," *IEEE J. Solid-State Circuits*, vol. 48, no. 7, pp. 1615–1624, Jul. 2013.

[3] K. J. Lee, G. Kim, J. Park, and H.-J. Yoo, "A vocabulary forest object matching processor with 2.07 M-vector/s throughput and 13.3 nJ/vector per-vector energy for Full-HD 60 fps video object recognition," *IEEE J. Solid-State Circuits*, vol. 50, no. 4, pp. 1059–1069, Apr. 2015.

[4] D. Jeon *et al.*, "An energy efficient full-frame feature extraction accelerator with shift-latch fifo in 28 nm CMOS," *IEEE J. Solid-State Circuits*, vol. 49, no. 5, pp. 1271–1284, May 2014.

[5] A. Suleiman, Z. Zhang, and V. Sze, "A 58.6 mW 30 frames/s real-time programmable multiobject detection accelerator with deformable parts models on full HD 1920 × 1080 videos," *IEEE J. Solid-State Circuits*, vol. 52, no. 3, pp. 844–855, Mar. 2017.

[6] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. 7th IEEE Int. Conf. Comput. Vis.*, vol. 2, Sep. 1999, pp. 1150–1157.

[7] H. Bay, T. Tuytelaars, and L. V. Gool, "Surf: Speeded up robust features," in *Proc. Eur. Conf. Comput. Vis.*, 2006, pp. 404–417.

[8] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, Sep. 2010.

[9] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie, "Behavior recognition via sparse spatio-temporal features," in *Proc. IEEE Int. Workshop Vis. Surv. Perform. Eval. Tracking Surv.*, Oct. 2005, pp. 65–72.

[10] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, "Learning realistic human actions from movies," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2008, pp. 1–8.

[11] G. Willems, T. Tuytelaars, and L. Van Gool, "An efficient dense and scale-invariant spatio-temporal interest point detector," in *Proc. Eur. Conf. Comput. Vis.*, 2008, pp. 650–663.

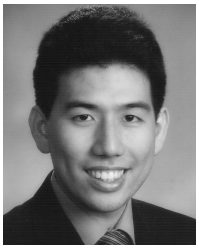
[12] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt, "Spatio-temporal convolutional sparse auto-encoder for sequence classification," in *Proc. Brit. Mach. Vis. Conf.*, Jan. 2012, pp. 1–12.

[13] B. A. Olshausen, "Learning sparse, overcomplete representations of time-varying natural images," in *Proc. Int. Conf. Image Process.*, vol. 1, Sep. 2003, pp. 41–44.

[14] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen, "Sparse coding via thresholding and local competition in neural circuits," *Neural Comput.*, vol. 20, no. 10, pp. 2526–2563, Oct. 2008.

[15] P. Knag, J. K. Kim, T. Chen, and Z. Zhang, "A sparse coding neural network ASIC with on-chip learning for feature extraction and encoding," *IEEE J. Solid-State Circuits*, vol. 50, no. 4, pp. 1070–1079, Apr. 2015.

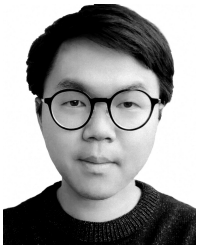
[16] C. Liu, S.-G. Cho, and Z. Zhang, "A 2.56-mm^{2}718 GOPS configurable spiking convolutional sparse coding accelerator in 40-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 53, no. 10, pp. 2818–2827, Oct. 2018.
[17] S. Savarese, A. DelPozo, J. C. Niebles, and L. Fei-Fei, "Spatial-temporal correlatons for unsupervised action classification," in *Proc. IEEE Workshop Motion Video Comput.*, Jan. 2008, pp. 1–8.
[18] K. Zhang, L. Zhang, and M.-H. Yang, "Real-time compressive tracking," in *Proc. Eur. Conf. Comput. Vis.*, 2012, pp. 864–877.
[19] P. F. Schultz, D. M. Paiton, W. Lu, and G. T. Kenyon.(2014). "Replicating kernels with a short stride allows sparse reconstructions with fewer independent kernels." [Online]. Available: https://arxiv.org/abs/1406.4205
[20] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: A local SVM approach," in *Proc. IEEE 17th Int. Conf. Pattern Recognit.*, vol. 3, Aug. 2004, pp. 32–36.
[21] V. Veeriah, N. Zhuang, and G.-J. Qi, "Differential recurrent neural networks for action recognition," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2015, pp. 4041–4049.



Thomas Chen (S'15) received the B.S. and M.S. degrees in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2013 and 2015, respectively, where he is currently pursuing the Ph.D. degree in electrical engineering.

He did an internship with the Circuits Research Lab, Intel Corporation, Hillsboro, OR, USA, in 2015. His research interests are high-speed and low-power VLSI circuits and systems.

Mr. Chen received the Rackham Merit Fellowship from the University of Michigan in 2013 and the NSF Graduate Research Fellowship in 2015.

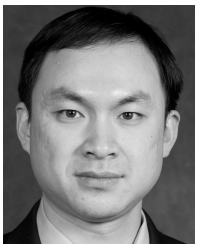


Ching-En Lee (S'15) received the B.S. degree in electrical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 2012, and the M.S. degree in electrical engineering from the University of California at Los Angeles, Los Angeles, CA, USA, in 2015. He is currently pursuing the Ph.D. degree in electrical engineering and computer science with the University of Michigan, Ann Arbor, MI, USA.

From 2015 to 2016, he was with Intel Labs, Hillsboro, OR, USA, where he worked on real-time machine learning hardware acceleration for full-duplex radios. From 2018 to 2019, he was with Iluvatar Corex, San Jose, CA, USA, where he led the development of deep learning inference SoCs and systems for edge computing applications. His current research interests focus on efficient domain-specific computing architectures and full-stack systems design for machine learning, deep learning, computer vision, and robotics.

Chester Liu (S'14) received the B.S. degree in electrical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 2008, and the M.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2010. He is currently pursuing the Ph.D. degree in electrical engineering and computer science with the University of Michigan, Ann Arbor, MI, USA.

From 2010 to 2013, he was with MediaTek, Hsinchu, Taiwan, where he worked on the platform design and verification for smartphone SoCs. Since 2014, he has been with the University of Michigan. His current research interests include neuromorphic computing, efficient hardware accelerator design for machine learning and robotics, and 2.5-D integration technologies.



Zhengya Zhang (S'02–M'09–SM'17) received the B.A.Sc. degree in computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2003, and the M.S. and Ph.D. degrees in electrical engineering from the University of California at Berkeley, Berkeley (UC Berkeley), CA, USA, in 2005 and 2009, respectively.

Since 2009, he has been a Faculty Member with the University of Michigan, Ann Arbor, MI, USA, where he is currently an Associate Professor with the Department of Electrical Engineering and Computer Science. His current research interests include low-power and high-performance VLSI circuits and systems for computing, communications, and signal processing.

Dr. Zhang serves on the Technical Program Committees of Symposium on VLSI Circuits and IEEE Custom Integrated Circuits Conference (CICC). He was a recipient of the David J. Sakrison Memorial Prize from UC Berkeley in 2009, the National Science Foundation CAREER Award in 2011, and the Intel Early Career Faculty Award in 2013. He was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS PART I: REGULAR PAPERS from 2013 to 2015 and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS PART II: EXPRESS BRIEFS from 2014 to 2015. He has been an Associate Editor of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS since 2015.