PixelRNN: In-pixel Recurrent Neural Networks for End-to-end–optimized Perception with Neural Sensors

Haley M. So[1], Laurie Bose[2], Piotr Dudek[2], Gordon Wetzstein[1]
[1]Stanford University, [2]The University of Manchester
https://www.computationalimaging.org/publications/pixelrnn/

Abstract

Conventional image sensors digitize high-resolution images at fast frame rates, producing a large amount of data that needs to be transmitted off the sensor for further processing. This is challenging for perception systems operating on edge devices, because communication is power inefficient and induces latency. Fueled by innovations in stacked image sensor fabrication, emerging sensor–processors offer programmability and processing capabilities directly on the sensor. We exploit these capabilities by developing an efficient recurrent neural network architecture, PixelRNN, that encodes spatio-temporal features on the sensor using purely binary operations. PixelRNN reduces the amount of data to be transmitted off the sensor by factors up to 256 compared to the raw sensor data while offering competitive accuracy for hand gesture recognition and lip reading tasks. We experimentally validate PixelRNN using a prototype implementation on the SCAMP-5 sensor–processor platform.

1. Introduction

Increasingly, cameras on edge devices are being used for enabling computer vision perception tasks rather than for capturing images that will be looked at by humans. Applications include various tasks in virtual and augmented reality headsets, wearable computing systems, drones, robotics, and the internet of things, among many others. For such edge devices, low-power operation is crucial, making it challenging to deploy large neural network architectures which traditionally leverage modern graphics processing units for inference.

A plethora of approaches have been developed in the "TinyML" community to address these challenges. Broadly speaking, these efforts focus on developing smaller [34] or more efficient network architectures, often by pruning or quantizing larger models [19]. Platforms like TensorFlow Lite Micro [23] enable application developers to deploy their models directly to power-efficient microcontrollers which process data closer to the sensor. Specialized artificial intelligence (AI) accelerators [1–3] further reduce the power consumption. While these approaches can optimize the processing component of a perception system, they do not reduce the large amount of digitized sensor data that needs to be transmitted to the processor in the first place, via power-hungry interfaces like standard MIPI (Mobile Industry Processor Interface), and stored in the memory. This is highly significant as data transmission and memory access are among the biggest power sinks in these systems [31]. Power-constrained systems, like mixed reality headsets, which require numerous sensors for perception tasks make it even more vital to minimize the communication overhead.

Recent advancements in 3D wafer stacking and high-density interconnects between these wafers create the opportunity for designing more efficient sensing and perception systems. Stacked CMOS image sensors contain layers with light-sensitive photodiodes as well as transistor circuitry that can turn a sensor into a parallel processor [28]. Data movement on these sensor–processors uses significantly less energy, and offers several orders of magnitude more bandwidth than conventional off-sensor communication protocols like MIPI [31, 70, 72]. This raises the question of how to design perception systems where sensing, data communication, and processing components are optimized end to end.

Efficient perception systems could be designed such that important task-specific image and video features are encoded directly on the imaging sensor using in-pixel processing, resulting in the sensor's output being significantly reduced to only these sparse features. This form of in-pixel feature encoding mechanism could significantly reduce the required bandwidth, thus reducing power consumption of data communication, memory management, and downstream processing. Event sensors [30] and emerging sensor–processors [76] are promising hardware platforms for such perception systems because they can directly extract either temporal information or spatial features, re-

spectively, on the sensor. These features can be transmitted off the sensor using low-power communication interfaces supporting low bandwidths.

Our work is motivated by the limitations of existing feature extraction methods demonstrated on these emerging sensor platforms. Rather than extracting simple temporal gradients [30] or spatial-only features via convolutional neural networks (CNNs) [12, 13], we propose in-pixel recurrent neural networks (RNNs) that efficiently extract spatio-temporal features on sensor–processors for bandwidth-efficient perception systems. RNNs are state-of-the-art network architectures for processing sequences, such as video in computer vision tasks [42]. Inspired by the emerging paradigm of neural sensors [51], our in-pixel RNN framework, dubbed PixelRNN, comprises a light-weight in-pixel spatio-temporal feature encoder. This in-pixel network is jointly optimized with a task-specific downstream network. We demonstrate that our architecture outperforms event sensors and CNN-based sensor–processors on perception tasks, including hand gesture recognition and lip reading, while drastically reducing the required bandwidth compared to any traditional sensor based approach. Moreover, we demonstrate that PixelRNN offers better performance and lower memory requirements than larger RNN architectures in the low-precision settings of in-pixel processing.

Our work's contributions include

1. the design and implementation of in-pixel recurrent neural networks for sensor–processors, enabling bandwidth-efficient perception on edge devices;
2. the demonstration that our on-sensor spatio-temporal feature encoding maintains high performance while significantly reducing sensor-to-processor communication bandwidth with several tasks, including hand gesture recognition and lip reading;
3. the experimental demonstration of the benefits of in-pixel RNNs using a prototype implementation on the SCAMP-5 sensor–processor.

2. Related Work

Machine Learning on the Edge. Edge computing devices are often subject to severe power and memory constraints, leading to various avenues of research and development. On the hardware side, approaches include in-memory compute [62], custom application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs), or other energy efficient AI accelerators. However, this does not address the issue of data transmission from imaging sensors, which is one of the main sources of power consumption [31]. To circumvent the memory constraints, network compression techniques are introduced. They fall into roughly five categories [19]: 1. parameter reduction by pruning redundancy [9, 35, 66, 81]; 2. low-rank parameter factorization [25, 37, 69]; 3. carefully designing structured convolutional filters [26, 68, 74]; 4. creating smaller models [4, 15, 33]; 5. parameter quantization [22, 36, 41, 47, 58, 71, 82]. In video perception tasks, transformers are popular. However, even the smallest transformers, such as Lite Transformers (∼92MB) [75], exceed what is currently available on sensor–processors (∼0.5MB). Instead of pushing all the compute onto the sensor plane, we utilize just a small amount of compute for a large decrease of bandwidth. In this work, we also apply ideas and techniques mentioned in this section.

Beyond Frame-based Sensing. Event-based cameras have been gaining popularity [30] as the readout is asynchronous and often sparse, triggered by pixel value changes above a certain threshold. However, these sensors are not programmable and do data compression with a simple fixed function. Another emerging class of sensors, focal plane sensor–processors, also known as pixel processor arrays support traditional sensing capabilities but also have a processing element embedded into each pixel. While conventional vision systems have separate hardware for sensing and computing, sensor–processors perform both tasks "in pixel," enabling efficient, low-latency and low-power computation [17, 29, 49, 54, 60, 78]. Further advances in 3D fabrication techniques, including wafer-level hybrid bonding and stacked CMOS image sensors, set the stage for rapid development of increasingly more capable sensor–processors.

In-pixel Perception. In the past few years, there has been a surge of advances in neural networks for vision tasks as well as an increasing desire to perform tasks on constrained mobile and wearable computing systems. Sensor–processors are a natural fit for such systems as they can perform sophisticated visual computational tasks at a significantly lower power than traditional hardware. Some early chips [48, 59] were based on implementing convolution kernels in a recurrent dynamical "Cellular Neural Network" model [21, 61]. In 2019, Bose et al. created "A Camera that CNNs" – one of the first works to implement a deep convolutional neural network on the sensor [12]. Since then, there have been a number of other works in CNNs on programmable sensors [13, 24, 32, 43–46, 67, 73]. These works extract features in the spatial domain, but miss a huge opportunity since they do not exploit temporal information. Purely 2D CNN-based approaches do not utilize or capitalize on the temporal redundancy or information of the sequence of frames. On the other hand, computational imaging works such as [51, 56, 57] utilize spatially varying pixel exposures that inherently encode information across the temporal dimension for high-dynamic-range imaging, video compressive sensing, and image deblurring. However, these works encode solely in the temporal domain. It's worth noting that classical video compression like MPEG [64] are proficient

at reconstruction, but they are not well-suited for in-pixel compute nor optimized for machine perception. Our work introduces light-weight extraction of spatio-temporal features, better utilizing the structure of the visual data while maintaining low bandwidth and high accuracy for machine perception tasks.

3. In-Pixel Recurrent Neural Networks

Emerging sensor–processors with in-pixel processing enable the development of end-to-end-optimized in-pixel and downstream networks off-sensor. In this section, we describe a new in-pixel recurrent spatio-temporal feature encoder that significantly improves upon existing temporal- or spatial-only feature encoders for video processing. The proposed pipeline is illustrated in Figure 1.

3.1. In-Pixel CNN-based Feature Encoder

Convolutional neural networks are among the most common network architectures in computer vision. They are written as

$$\begin{aligned} \text{CNN}\ (\mathbf{x}) &= \left(\phi_{n-1} \circ \phi_{n-2} \circ \ldots \circ \phi_0\right)(\mathbf{x}), \\ \phi_i &= \mathbf{x}_i \mapsto \psi\left(\mathbf{w}_i * \mathbf{x}_i + \mathbf{b}_i\right), \end{aligned} \tag{1}$$

where $\mathbf{w}_i * \mathbf{x}_i : \mathbb{N}^{N_i \times M_i \times C_i} \mapsto \mathbb{N}^{N_{i+1} \times M_{i+1} \times C_{i+1}}$ describes the multi-channel convolution of CNN layer i and $\mathbf{b}_i$ is a vector containing bias values. Here, the input image $\mathbf{x}_i$ has C_i channels and a resolution of $N_i \times M_i$ pixel and the output of layer i is further processed by the nonlinear activation function ψ.

Many edge devices, such as the SCAMP-5 system used in this work, lack native multiplication operations at the pixel level. For this reason, works storing network weights $\mathbf{w}_i$ in pixel typically restrict themselves to using binary, $\{-1, 1\}$, or ternary $\{-1, 0, 1\}$ values. This reduces all multiplications to sums or differences, which are highly efficient native operations.

3.2. In-pixel Spatio-temporal Feature Encoding

Recurrent neural networks (RNNs) are state-of-the-art network architectures for video processing and are more compact than using transformers or temporal convolutions [5]. Whereas a 2D CNN only considers each image in isolation, an RNN extracts spatio-temporal features to process video sequences more effectively. Network architectures for sensor–processors must satisfy two key criteria. First, they should be small and use low-precision weights. Second, they should comprise largely of local operations as the processors embedded within each pixel can only communicate with their direct neighbors (e.g., [17]).

To satisfy these unique constraints, we devise an RNN architecture that combines ideas from convolutional, gated recurrent units (GRUs) [6] and minimal gated units [80]. The resulting simple, yet effective PixelRNN architecture, is written as

$$\mathbf{f}_t = \psi_f\left(\mathbf{w}_f * \text{CNN}\left(\mathbf{x}_t\right) + \mathbf{u}_f * \mathbf{h}_{t-1}\right), \tag{2}$$

$$\mathbf{h}_t = \mathbf{f}_t \odot \mathbf{h}_{t-1}, \tag{3}$$

$$\mathbf{o}_t = \psi_o\left(\mathbf{w}_o * \text{CNN}\left(\mathbf{x}_t\right) + \mathbf{u}_o * \mathbf{h}_{t-1}\right), \tag{4}$$

where $\mathbf{w}_f$, $\mathbf{u}_f$, $\mathbf{w}_o$, $\mathbf{u}_o$ are small convolution kernels and ψ_f is either the sign (when working with binary constraints) or the sigmoid function (when working with full precision). We include an optional nonlinear activation function ψ_o and an output layer $\mathbf{o}_t$ representing the values that are actually transmitted off sensor to the downstream network running on a separate processor. For low-bandwidth operation, the output layer $\mathbf{o}_t$ is only computed, and values transmitted off the sensor, every K frames. The output layer can optionally be omitted, in which case the hidden state $\mathbf{h}_t$ is streamed off the sensor every K frames.

PixelRNN uses what is commonly known as a "forget gate", $\mathbf{f}_t$, and a hidden state $\mathbf{h}_t$, which are updated at each time step t from the input $\mathbf{x}_t$. RNNs use forget gates to implement a "memory" mechanism that discards redundant spatial features over time. PixelRNN's forget gate is also motivated by this intuition, but our update mechanism in Eq. 3 is tailored to work with binary constraints using values $\{-1, 1\}$. In this case, Eq. 3 flips the sign of $\mathbf{f}_t$ in an element-wise manner rather than decaying over time. This mechanism works very well in practice when $\mathbf{h}_t$ is re-initialized to all ones every K time steps.

PixelRNN's architecture includes alternative spatial- or temporal-only feature extractors as special cases. For example, it is easy to see that it models a conventional CNN by omitting the recurrent units. We specifically write out the output gate in our definition to make it intuitive how PixelRNN also approximates a difference camera as a special case, which effectively implement a temporal-only feature encoder. In this case, $\mathbf{h}_t = \mathbf{x}_t, \mathrm{w}_o = 1, \mathrm{u}_o = -1$ and

$$\psi_o\left(x\right) = \begin{cases} -1 & \text{for x} < -\delta \\ 0 & \text{for } -\delta \leq \text{x} \leq \delta \\ 1 & \text{for } \delta < \text{x} \end{cases}, \text{ for some threshold } \delta.$$

Difference cameras are used as an example of temporal-only encoders that closely approximate event cameras [30], with the exception of their asynchronous readout.

3.3. Learning Quantized In-pixel Parameters

PixelRNN uses binary weights to reduce all multiplications to sums. To learn these parameters efficiently, we parameterize each of these values w using a continuous value $\widetilde{w}$ and a quantization function q such that

$$w = q\left(\widetilde{w}\right), \quad q : \mathbb{R} \rightarrow \mathcal{Q}. \tag{5}$$

Here, q maps a continuous value to the closest discrete value in the feasible set $\mathcal{Q}$, i.e., $\{-1, 1\}$. For the binary weights

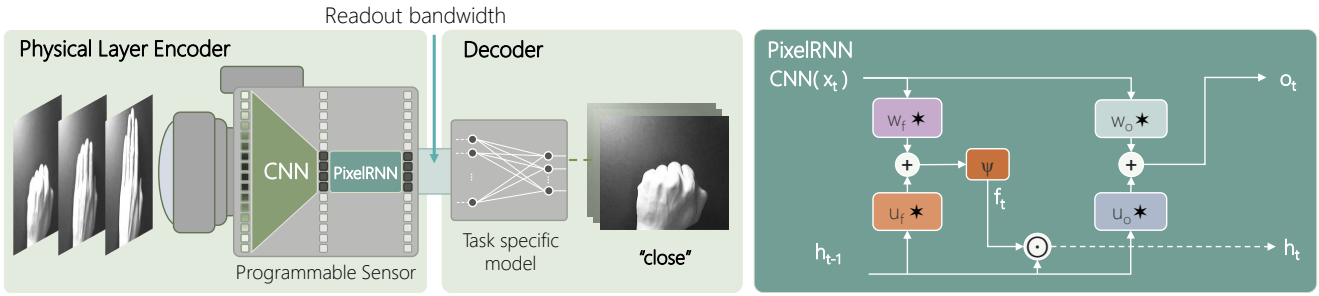

Figure 1. The perception pipeline of PixelRNN can be broken down into an in-pixel encoder and a task-specific decoder. On the left is the camera equipped with a sensor–processor, which offers processing and memory at the pixel level. The captured light is directly processed by a CNN that extracts spatial features, which are further processed by a convolutional recurrent neural network with built-in memory and temporal feature extraction. Here we show our PixelRNN variant on the right, $\star$ being the convolution operator, $\odot$ element-wise multiplication, and ψ a nonlinear function. Instead of sending out full 256×256 values at every time step, our encoder outputs 64×64 once every 16 time steps. While we show this pipeline for hand gesture recognition, the decoder can be designed for any perception task.

we use, $w = q(\tilde{w}) = \text{sign}(\tilde{w} - \zeta)$ with a learned zero-point offset ζ.

One can employ surrogate gradient methods [7, 77], continuous relaxation of categorical variables using Gumbel-Softmax [38, 50], or other approaches to approximately differentiate through q. We found approximating the gradient of the sign function with the derivative of $\tanh(mx)$ produced very good results:

$$\frac{\partial q}{\partial \tilde{w}} \approx m \cdot (1 - \tanh^2(m\tilde{w})) \tag{6}$$

where $m > 0$ controls the steepness of the $\tanh$ function, which is used as a differentiable proxy for $q \approx \tanh(m\tilde{w})$ in the backward pass. The larger m is, the more it resembles the sign function and the more the gradient resembles the delta function. Among different quantization schemes including the straight-through estimator (STE) [8], stochastic binarization [22], and binarizing to the $\{0, 1\}$ regime, we found that binarizing to the $\{-1, 1\}$ regime and utilizing the tanh gradient produced the best results in our application.

3.4. Implementation Details

We implement our per-frame CNN feature encoder, testing both 1 or 2 layers and process images at a resolution of 64×64 pixels by downsampling the raw sensor images from 256×256 before feeding them into the CNN. In all experiments, we set $K = 16$, so our PixelRNN transmits data off the sensor once every 16 frames. Thus, we achieve a reduction in bandwidth by a factor of $256\times$ compared to the raw sensor data. In all of our experiments, we set the function ψ_o to the identity function. Additional implementation details are found in the supplement, and source code will be available on the project website.

4. Experiments

4.1. Evaluating Feature Encoders

As discussed in the previous section, RNNs require a CNN-based feature encoder as part of their architecture. In-pixel CNNs have been described in prior work [13, 43, 44], albeit not in the context of video processing with RNNs. Table 1 summarizes the simulation performance of re-implementations of various in-pixel CNN architectures for image classification using the MNIST, CIFAR-10 dataset, and hand gesture recognition from individual images. Bose et al. [12, 13] describe a 2-layer CNN with 4×4 kernels with ternary weights. The two works have the same architecture but differ drastically in the sensor–processor implementation. Bose uses a stochastic approach for quantization. Liu et al. [43, 44] describe 1- and 3-layer CNNs with strided convolutions with binary weights, group convolutions, and batch norm using similar sensor–processor implementations concepts from Bose. Liu uses the STE training technique. Our feature encoder is a binary 2-layer variant of Bose et al.'s CNN using 5×5 kernels. Each layer has 16 kernels of size 5×5 and are followed with a non-linearity and max-pooling of 4×4. The 16 16×16 channels are then concatenated into a single 64×64 image size to serve as the input to the next convolutional layer or to the PixelRNN. All of these CNNs are roughly on-par with some performing better at some tasks than others. Ours strikes a good balance between accuracy and model size. We do not claim this CNN to be a contribution of our work, but include this brief comparison for completeness.

4.2. Baseline Architectures

We use several baselines for our analysis. The RAW camera mode simply outputs the full-resolution 256×256 pixel input frame at every time step and represents the naive imaging approach. The simulated difference camera represents a simple temporal-only feature encoder. We also

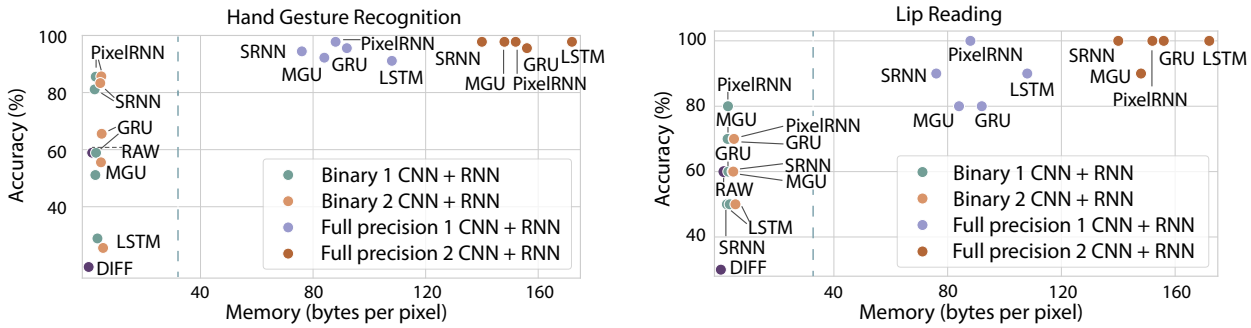

Figure 2. **Network architecture comparison.** We compare baselines, including a RAW and difference camera (DIFF) as well as several RNN architectures, each with 1- and 2-layer CNN encoders and binary or full 32-bit floating-point precision. PixelRNN offers the best performance for the lowest memory footprint, especially when used with binary weights. The dashed vertical line indicates the available memory on our hardware platform, SCAMP-5, showing that low-precision network architectures are the only feasible option in practice.

Model Name	Accuracy MNIST	Accuracy CIFAR-10	Accuracy Hand Gest.	Model Params	Size (MB)
Bose [12, 13]	95.0%	39.8%	43.4%	257	0.05
Liu 2020 [44]	80.0%	32.5%	57.4%	258	0.03
Liu 2022 [43]	**95.1**%	32.6%	60.2%	2,374	0.30
Our CNN	90.9%	**43.1**%	**68.1**%	802	0.10

Table 1. **Comparing CNN Feature Encoders.** We simulated different in-pixel CNNs using image classification on MNIST, CIFAR-10, and the Cambridge hand gesture recognition based on different implementations. All CNN architectures perform roughly on par with our binary 2-layer CNN encoder striking a good balance between accuracy and model size. The model size is computed by multiplying the number of model parameters by the quantization of the values.

Model Name	# Model Parameters with 1 layer / 2 layer CNN	# Values Read Out (per 16 time steps)
RAW	0 / 0	$256^2 \cdot 16 = 1{,}048{,}576$
Difference Camera	0 / 0	$64^2 \cdot 16 = 65{,}536$
SRNN	451 / 852	$64^2 \cdot 1 = 4{,}096$
LSTM	601 / 1,002	4,096
GRU	551 / 952	4,096
MGU	501 / 902	4,096
PixelRNN	501 / 902	4,096

Table 2. **Overview of Baselines.** We list the number of model parameters and the number of values read out per 16 time steps for each baseline. The RAW and difference camera stream data off the sensor at every frame. All RNNs only stream data off the sensor once every 16 frames, providing the benefit of decreased readout.

include several RNN architectures, including convolutional long short-term memory (LSTM) [63], gated recurrent unit (GRU) [6], minimal gated unit (MGU) [80], a simple RNN (SRNN), and our PixelRNN. Moreover, we evaluated each of the RNN architectures using 1-layer and 2-layer CNN feature encoders. The output of all RNNs is read off the sensor only once every 16 time steps. All baselines represent options for in-pixel processing and their respective output is streamed off the sensor. In all cases, a single fully-connected network layer processes this output on a downstream processor to compute the final classification scores. This fully-connected layer is trained end to end for each of the baselines. While this fully-connected layer is simple, it is effective and could be customized for a specific task. Note that a larger decoder could be used, but that does not accomplish the goal of decreasing the bandwidth readout to the processor. Additional details on these baselines, including formulas and training details, are listed in the supplement. Table 2 shows an overview of the number of model parameters and the readout bandwidth.

Datasets. For the hand gesture recognition task, we use the Cambridge Hand Gesture Recognition dataset [39]. This dataset consists of 900 video clips of 9 gesture classes; each class contains 100 videos. For the lip reading task, we use the Tulips1 dataset [55]. It is a small Audiovisual database of 12 subjects saying the first 4 digits in English.

4.3. Accuracy vs. Memory

In Figure 2, we evaluate the accuracy of several baseline architectures on two tasks: hand gesture recognition (left) and lip reading (right). The term 'memory,' depicted along the x-axis, represents all intermediate features, per pixel, that need to be stored during a single forward pass through the encoder. We do not count the network parameters in this plot, because they do not dominate the memory requirements and can be shared among the pixels. In this study, we compare the baselines described above, each with 1- and 2-layer CNN encoders and binary or full 32-bit floating point precision. For the full-precision networks, all RNN variants perform well. PixelRNN achieves an accuracy comparable to the best models, but it provides one of the lowest memory footprints. Comparing the networks with binary weights, PixelRNN also offers the best accuracy with a memory footprint comparable to the next best method. Surprisingly, larger architectures, such as LSTMs, do not perform well when used with binary weights. This can be explained by the increasing difficulty

of reliably training increasingly large networks with binary parameter constraints. With more gates, more information can be lost through the binarization. Leaner networks, such as SRNN and PixelRNN, can be trained more robustly and reliably in these settings. In either the full-precision or quantized setting, we are able to achieve good accuracy while significantly reducing the readout bandwidth. We include additional modes of compression in the supplement.

Constraints of the Experimental Platform. Our hardware platform, SCAMP-5, provides an equivalent of 8 bytes of memory per pixel, resulting in 32 bytes available to store intermediate features in a 4×4 macropixel. This limit is illustrated as dashed vertical lines in Figure 2, indicating that only low-precision RNN networks are feasible on this platform. The available memory is computed as follows. SCAMP-5 has 6 analog registers per pixel, each we assume is equivalent to 1 byte: 2 registers store the model weights, 2 are reserved for computation, leaving 2 bytes per pixel for the intermediate features. SCAMP-5 consists of an array of 256×256 pixel processors, however our approach operates on a smaller effective image size of 64×64. This allows us to consider a single "macropixel" to comprise a block of 4×4 pixels, increasing the effective storage for immediate features to 32 bytes per macropixel.

4.4. Accuracy vs. Bandwidth

We select a readout bandwidth of 4,096 values (every 16 frames) based on the available bandwidth of our hardware platform, the SCAMP-5 sensor. In Figure 3 we evaluate the effect of further reducing this bandwidth on the accuracy for the PixelRNN architecture. Bandwidth is controlled using a max-pooling layer operating at differing sizes from 1×1 through 8×8 and then at multiples of 8 up to 64×64 before inputting the intensity images to PixelRNN. The resulting output bandwidths range between 1 to 4,096. We ran each experiment ten times and plot the mean and standard deviations for hand gesture recognition and lip reading. We observe that the bandwidth could be further reduced to about 1,000 values every 16 frames without significantly degrading the accuracy on these datasets. However, decreasing the bandwidth beyond this also reduces the accuracy.

5. Experimental Prototype

5.1. Pixel-Level Programmable Sensors

SCAMP-5 [17] is one of the emerging programmable sensors representative of the class of focal-plane sensor–processors (FPSP) and has been used to prototype for a variety of tasks [11, 14, 18, 52, 53, 65]. Unlike conventional image sensors, each of the 256×256 pixels is equipped with an arithmetic logic unit, 6 local analog registers, (named *A, B, C, D, E, F*), 13 digital bit registers, control and I/O circuitry, and access to certain registers of the four neighboring pixels. For clarity, while SCAMP-5 is fabricated with planar technology, we refer to register "plane" *A* as the aggregation of all corresponding registers *A* across the entire image sensor, and similarly for planes B through F. SCAMP-5 operates in single-instruction multiple-data (SIMD) mode with capabilities to address individual pixels, patterns of pixels, or the whole array. It features mixed-mode operation execution that allows for low-power compute prior to A/D conversion. Most importantly, SCAMP-5 is programmable.

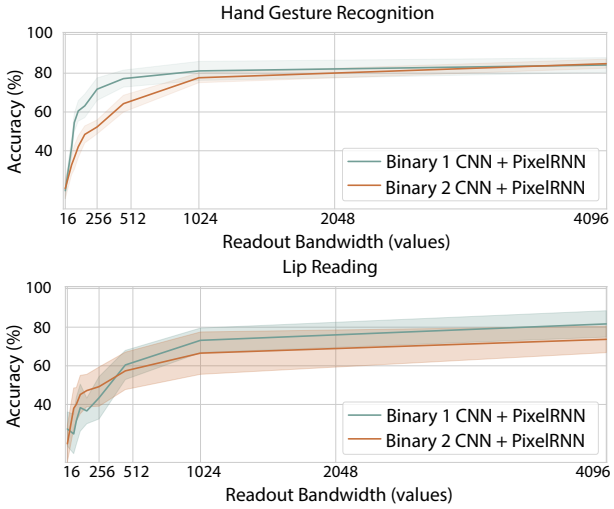

Figure 3. **Bandwidth Analysis.** We can control the bandwidth of data read off the sensor using increasingly larger max-pooling layers at the cost of decreased accuracy.

5.2. Implementation of PixelRNN on SCAMP-5.

We implement the binary 1-layer CNN + PixelRNN encoder on sensor as we find that encoder to have low complexity and high performance. The pipeline for our prototype feature extractor is shown in Figure 4. To efficiently calculate the CNN and PixelRNN on SCAMP-5, we break the sensor up into 16 parallel processor elements (PE) of size 64×64. This allows us to calculate 16 depth-wise convolutions [20] with 5×5 kernels in parallel. The features undergo a ReLU activation, maxpooling and binarization to 16 16×16 and then are concatenated to create a single 64×64 input to the RNN. We utilize image transformation methods for SCAMP-5 introduced by [10]. Our RNN uses the output of the CNN and the current hidden state to update the new hidden state every time step and to compute an output every 16 time steps. The RNN gates are calculated via convolution and element-wise multiplication. To suit the SCAMP-5 architecture, we limited operations to addition, XOR, and negation, and trained a binary version of PixelRNN, binarizing weights and features to -1 and 1.

Convolution Operation. A single pixel cannot hold all of the weights of a single kernel, so the weights are spread

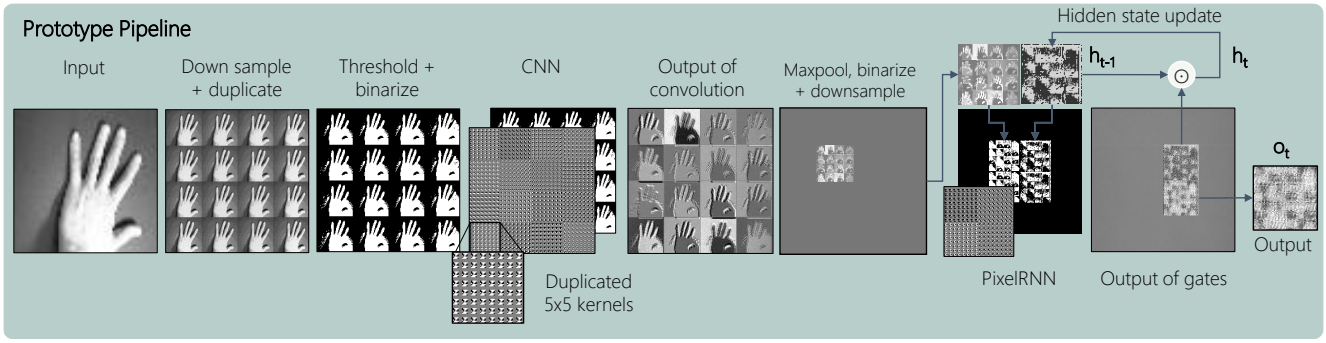

Figure 4. This pipeline shows the sequence of operations from left to right. The input image is downsampled, duplicated, and binarized. Stored convolutional weights perform 16 convolutions, to produce 16 feature maps in the 4×4 grid of processor elements. A ReLU activation is applied, followed by max-pooling, downsampling, and binarization. This can either be fed to another CNN layer or to the input of the RNN. The RNN takes in the output of the CNN and the previous hidden state $\mathbf{h}_{t-1}$. The hidden state $\mathbf{h}_t$ is updated every timestep. The output $\mathbf{o}_t$ is read out every 16 frames, yielding 64× decrease in bandwidth.

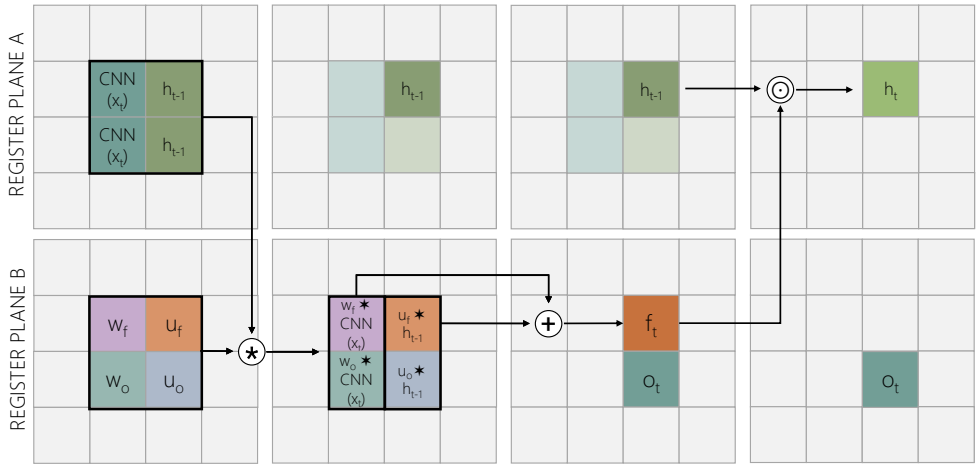

Figure 5. To implement PixelRNN on SCAMP-5, the image plane is split into a 4×4 grid of processor elements shown above. Two analog register planes are used, Register planes A and B. Above, we show the sequence of operations from left to right. The input from the CNN and the previous hidden state are duplicated in A. These 4 PEs are convolved $*$ with the corresponding gate weights stored in plane B. The resulting convolutions in the second column are then added to compute the output $\mathbf{o}_t$ and the forget gate $\mathbf{f}_t$. Note that an in-place binarization is applied to $\mathbf{f}_t$. The hidden state $\mathbf{h}_t$ is updated via an element-wise multiplication $\odot$ of $\mathbf{h}_{t-1}$ and $\mathbf{f}_t$.

across an analog register plane. as shown in Figure 4. To perform a convolution, SCAMP-5 iterates through all 25 weights in the 5×5 kernel, each time multiplying it with the whole image and adding to a running sum. The image is then shifted, the next weight fills the register plane, and the process continues until the feature is computed. This process builds on [13]. We include a detailed diagram in the supplement.

RNN Operation. Figure 5 shows the layout of the sequence of operations in the RNN. In Figure 5, the 256×256 pixels are split into the 16 larger parallel processor elements (PE) of size 64×64. In register plane *A*, we take the output from the CNN and the previous hidden state and duplicate it to two other PEs in plane A. Register plane *B* holds the corresponding weights $\mathbf{w}_f$, $\mathbf{u}_f$, $\mathbf{w}_o$, $\mathbf{u}_o$ for the convolution operators needed. 4 convolutions are simultaneously run on one register plane. The outputs in plane *B* are shifted and added. A binarization is applied to get $\mathbf{f}_t$. This then updates the hidden state via element-wise multiplication every time step. Every 16 time steps, SCAMP-5 outputs the 64×64 image corresponding to the output gate $\mathbf{o}_t$. Our encoder distills the salient information while greatly reducing readout.

Accounting for Analog Uncertainty. As with all analog compute, a certain amount of noise is to be expected. There is uncertainty in the precision and uniformity of the values, and analog registers decay over time. The decay is exacerbated if one moves information from pixel to pixel such as in shifting an image. In the RNN, these effects can accumulate for 16 frames, leading to a significant amount of noise. To account for these effects, we trained binary models in simulation with varying amounts of added Gaussian noise in the CNN and RNN prior to quantization of the features.

	Test Accuracy
Hand Gesture Recognition	
Accuracy in Simulation	85.6%
SCAMP-5 Accuracy with model trained without noise	68.9%
SCAMP-5 Accuracy with noise-trained model	84.4%
Lip Reading	
Accuracy in Simulation	80.0%
SCAMP-5 Accuracy with model trained without noise	60.0%
SCAMP-5 Accuracy with noise-trained model	80.0%

Table 3. **Experimental Results.** We implemented two models on SCAMP-5 for each task, one trained with noise to account for analog noise, and one trained without. The test set performance significantly improved with noise added during training.

5.3. Assessment

To test our prototype, we processed all video datasets on SCAMP-5 (see supplement for additional details). With our current implementation, it takes roughly 3.5 ms to run a single frame through the on-sensor encoder. The 64×64 output region then goes through the off-sensor linear layer decoder. We evaluate the performance using models trained with and without noise against the performance of the binary 1-layer CNN + PixelRNN in simulation. The results shown in Table 3 highlight the benefits of training with noise. Without the noise-trained model, we reached 68.9% on the hand gesture recognition test set. Performance improved to 84.4% when we used the weights from training with noise. Similarly, the test performance on lip reading was boosted to 80.0% when using a model trained on noise, reaching the same performance as in simulation. Adding noise during training helps mitigate analog noise effects of SCAMP-5 and create robust models.

6. Discussion

In the traditional computer vision pipeline, RAW images are sent from camera to processors for different perception tasks. While completely viable in systems not limited by compute, memory, or power, many edge devices do not have this luxury. For AR/VR devices, robotics, and wearables, low-power operation is crucial, and even more so if the system requires multiple cameras. The community has worked on creating more efficient models and specialized accelerators. Existing work, however, overlooks the communication bandwidth between camera and processor, which consumes a significant portion of the power budget. In this work, we demonstrate how running a simple in-pixel spatio-temporal feature extractor can decrease the bandwidth, and hence power associated with readout. Even with highly quantized encoders and a very simple decoder, we still maintain good performance on hand gesture recognition and lip reading. We studied different RNN architectures and presented PixelRNN that performs well in highly quantized settings, studied just how small we could make the bandwidth before affecting performance, and implemented a physical prototype with one of the emerging sensors, SCAMP-5, that is paving the way for future sensors. PixelRNN is potentially valuable for other tasks that exhibit spatio-temporal redundancy as well, such as action recognition, event classification, anomaly detection, and environment monitoring.

Limitations and Generalizability. One of the biggest challenges of working with SCAMP-5 is the limited amount of in-pixel digital memory. Because SCAMP-5 is not a stacked CMOS image sensor, the sensor suffers from a direct tradeoff between pixel fill factor / image quality and amount of memory. Future sensor–processors built on stacked wafer technology do not suffer from this tradeoff and offer a significantly increased amount of memory. This advantage can increase the precision of numbers on the sensor from binary to 8 or more bits; increase the encoder size to unlock more expressive network architectures; and increase the number of intermediate features within the in-pixel encoder. Having more compute will also allow us to apply tools like architecture search to optimize where compute happens in a system [27].

To validate that PixelRNN generalizes to future sensor–processors, other decoder architectures, and datasets beyond those previously shown, we perform additional benchmarks on a challenging dynamic hand gesture recognition dataset, EgoGesture, that contains 83 classes [16, 79]. For this purpose, we implement the PixelRNN encoder with floating point precision and add it as a data-compressing pre-processor to a state-of-the-art decoder network [40] for EgoGesture. Accuracy for various compression ratios are reported in our supplement. With a readout bandwidth compression of $8\times$ compared to the RAW mode, for example, our approach maintains 88.9% accuracy while naive temporal downsampling to the same bandwidth resulted in 74.5%. In addition, as PixelRNN is size agnostic, it will naturally scale to upcoming higher resolution sensors.

7. Conclusion

Motivated by the potential of in-pixel computing, we designed the first lightweight in-pixel RNN-based spatio-temporal encoder to significantly decrease readout bandwidth. In our work, we demonstrated bandwidth reduction by factors of $256\times$ and above compared to the raw sensor data. We also showed the efficacy in real hardware. The in-pixel module is a versatile solution applicable across various video perception tasks and can complement other advances in power reduction for edge devices. We believe our work paves the way for other inference tasks of future artificial intelligence–driven sensor–processors.

Acknowledgements: This project was in part supported by the National Science Foundation and Samsung.

References

[1] Ambarella CV5S 8K AI Vision Processor. https://www.ambarella.com/wp-content/uploads/Ambarella_CV5S_Product_Brief.pdf. Accessed: 2024-03-26. 1

[2] Hailo-8 AI processor. https://hailo.ai/files/hailo-8-product-brief-en/. Accessed: 2024-03-26.

[3] Intel® Movidius™ Myriad™ X Vision Processing Unit (VPU) with Neural Compute Engine. https://www.intel.com/content/www/us/en/products/docs/processors/movidius-vpu/myriad-x-product-brief.html. Accessed: 2024-03-26. 1

[4] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2014. 2

[5] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *ArXiv*, abs/1803.01271, 2018. 3

[6] Nicolas Ballas, Li Yao, Chris Pal, and Aaron Courville. Delving deeper into convolutional networks for learning video representations. *arXiv preprint arXiv:1511.06432*, 2015. 3, 5

[7] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 4

[8] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *ArXiv*, abs/1308.3432, 2013. 4

[9] Davis W. Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John V. Guttag. What is the state of neural network pruning? *ArXiv*, abs/2003.03033, 2020. 2

[10] Laurie Bose and Piotr Dudek. Mapping image transformations onto pixel processor arrays, 2024. 6

[11] Laurie Bose, Jianing Chen, Stephen J Carey, Piotr Dudek, and Walterio Mayol-Cuevas. Visual odometry for pixel processor arrays. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4604–4612, 2017. 6

[12] Laurie Bose, Piotr Dudek, Jianing Chen, Stephen Carey, and Walterio Mayol-Cuevas. A camera that cnns: Towards embedded neural networks on pixel processor arrays. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1335–1344, 2019. 2, 4, 5

[13] Laurie Bose, Piotr Dudek, Jianing Chen, Stephen J. Carey, and Walterio W. Mayol-Cuevas. Fully embedding fast convolutional networks on pixel processor arrays. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX*, page 488–503, Berlin, Heidelberg, 2020. Springer-Verlag. 2, 4, 5, 7

[14] Laurie Bose, Jianing Chen, Stephen J. Carey, and Piotr Dudek. Pixel processor arrays for low latency gaze estimation. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pages 970–971, 2022. 6

[15] Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *KDD '06*, 2006. 2

[16] Congqi Cao, Yifan Zhang, Yi Wu, Hanqing Lu, and Jian Cheng. Egocentric gesture recognition using recurrent 3d convolutional neural networks with spatiotemporal transformer modules. In *IEEE International Conference On Computer Vision (ICCV)*, 2017. 8

[17] Stephen J Carey, Alexey Lopich, David RW Barr, Bin Wang, and Piotr Dudek. A 100,000 fps vision sensor with embedded 535gops/w 256× 256 simd processor array. In *2013 Symposium on VLSI Circuits*, pages C182–C183. IEEE, 2013. 2, 3, 6

[18] Jianing Chen, Yanan Liu, Stephen J. Carey, and Piotr Dudek. Proximity estimation using vision features computed on sensor. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2689–2695, 2020. 6

[19] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1):126–136, 2018. 1, 2

[20] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, Los Alamitos, CA, USA, 2017. IEEE Computer Society. 6

[21] L.O. Chua and L. Yang. Cellular neural networks: theory. *IEEE Transactions on Circuits and Systems*, 35(10):1257–1272, 1988. 2

[22] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, page 3123–3131, Cambridge, MA, USA, 2015. MIT Press. 2, 4

[23] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, Pete Warden, and Rocky Rhodes. Tensorflow lite micro: Embedded machine learning for tinyml systems. In *Proceedings of Machine Learning and Systems*, pages 800–811, 2021. 1

[24] Thomas Debrunner, Sajad Saeedi, and Paul H. J. Kelly. Auke: Automatic kernel code generation for an analogue simd focal-plane sensor-processor array. *ACM Trans. Archit. Code Optim.*, 15(4), 2019. 2

[25] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, page 1269–1277, Cambridge, MA, USA, 2014. MIT Press. 2

[26] Sander Dieleman, Jeffrey Fauw, and Koray Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks. 2016. 2

[27] Xin Dong, Barbara De Salvo, Meng Li, Chiao Liu, Zhongnan Qu, H.T. Kung, and Ziyun Li. Splitnets: Designing neural architectures for efficient distributed computing on head-

mounted systems. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12559–12569, 2022. 8

[28] Piotr Dudek, Thomas Richardson, Laurie Bose, Stephen Carey, Jianing Chen, Colin Greatwood, Yanan Liu, and Walterio Mayol-Cuevas. Sensor-level computer vision with pixel processor arrays for agile robots. *Science Robotics*, 7(67): eabl7755, 2022. 1

[29] Jorge Fernández-Berni, Ricardo Carmona-Galán, and Luis Carranza-González. Flip-q: A qcif resolution focal-plane array for low-power image processing. *IEEE Journal of Solid-State Circuits*, 46(3):669–680, 2011. 2

[30] Guillermo Gallego, Tobi Delbrück, G. Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jörg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44:154–180, 2022. 1, 2, 3

[31] Jorge Gomez, Saavan Patel, Syed Shakib Sarwar, Ziyun Li, Raffaele Capoccia, Zhao Wang, Reid Pinkham, Andrew Berkovich, Tsung-Hsun Tsai, Barbara De Salvo, and Chiao Liu. Distributed on-sensor compute system for AR/VR devices: A semi-analytical simulation framework for power estimation. *CoRR*, abs/2203.07474, 2022. 1, 2

[32] Benoit Guillard. *Optimising convolutional neural networks for super fast inference on focal-plane sensor-processor arrays*. PhD thesis, 2019. 2

[33] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015. 2

[34] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *arxiv*, 2017. 1

[35] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. *ArXiv*, abs/1707.01213, 2018. 2

[36] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. page 4114–4122, Red Hook, NY, USA, 2016. Curran Associates Inc. 2

[37] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *BMVC 2014 - Proceedings of the British Machine Vision Conference 2014*, 2014. 2

[38] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 4

[39] Tae-Kyun Kim, Shu-Fai Wong, and Roberto Cipolla. Tensor canonical correlation analysis for action classification. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007. 5

[40] Okan Köpüklü, Ahmet Gunduz, Neslihan Kose, and Gerhard Rigoll. Real-time hand gesture detection and classification using convolutional neural networks. In *2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)*, page 1–8. IEEE Press, 2019. 8

[41] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 344–352, Red Hook, NY, USA, 2017. Curran Associates Inc. 2

[42] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning, 2015. 2

[43] Yanan Liu and Yao Lu. On-sensor binarized fully convolutional neural network for localisation and coarse segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 3629–3638, 2022. 2, 4, 5

[44] Yanan Liu, Laurie Bose, Jianing Chen, Stephen J. Carey, Piotr Dudek, and W. Mayol-Cuevas. High-speed light-weight cnn inference via strided convolutions on a pixel processor array. In *BMVC*, 2020. 4, 5

[45] Yanan Liu, Laurie Bose, Rui Fan, Piotr Dudek, and Walterio Mayol-Cuevas. On-sensor binarized cnn inference with dynamic model swapping in pixel processor arrays. *Frontiers in Neuroscience*, 16, 2022.

[46] Yanan Liu, Rui Fan, Jianglong Guo, Hepeng Ni, and Muhammad Usman Maqboo Bhutta. In-sensor visual perception and inference. *Intelligent Computing*, 2:0043, 2023. 2

[47] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *ECCV*, 2018. 2

[48] G. Liñan-Cembrano, Servando Espejo, Rafael Castro, and A Rodriguez-Vazquez. Ace4k: An analog i/o 64×64 visual microprocessor chip with 7-bit analog accuracy. *International Journal of Circuit Theory and Applications*, 30:89–116, 2002. 2

[49] Alexey Lopich and Piotr Dudek. An 80×80 general-purpose digital vision chip in 0.18um cmos technology. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 4257–4260, 2010. 2

[50] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016. 4

[51] Julien N.P. Martel, L.K. Muller, S.J. Carey, P. Dudek, and G. Wetzstein. Neural Sensors: Learning Pixel Exposures for HDR Imaging and Video Compressive Sensing with Programmable Sensors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. 2

[52] Julien N. P. Martel, Lorenz K. Müller, Stephen J. Carey, and Piotr Dudek. High-speed depth from focus on a programmable vision chip using a focus tunable lens. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2017. 6

[53] Julien N. P. Martel, Lorenz K. Müller, Stephen J. Carey, Jonathan Müller, Yulia Sandamirskaya, and Piotr Dudek. Real-time depth from focus on a programmable focal plane processor. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(3):925–934, 2018. 6

[54] Wei Miao, Qingyu Lin, Wancheng Zhang, and Nan-Jian Wu. A programmable simd vision chip for real-time vision applications. *IEEE Journal of Solid-State Circuits*, 43(6):1470–1479, 2008. 2

[55] Javier R. Movellan. Visual speech recognition with stochastic networks. In *NIPS*, 1994. 5

[56] Shree K. Nayar and Tomoo Mitsunaga. High dynamic range imaging: spatially varying pixel exposures. *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*, 1:472–479 vol.1, 2000. 2

[57] Cindy M Nguyen, Julien NP Martel, and Gordon Wetzstein. Learning spatially varying pixel exposures for motion deblurring. In *IEEE International Conference on Computational Photography (ICCP)*, 2022. 2

[58] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Computer Vision – ECCV 2016*, pages 525–542, Cham, 2016. Springer International Publishing. 2

[59] A. Rodriguez-Vazquez, G. Linan-Cembrano, L. Carranza, E. Roca-Moreno, R. Carmona-Galan, F. Jimenez-Garrido, R. Dominguez-Castro, and S.E. Meana. Ace16k: the third generation of mixed-signal simd-cnn ace chips toward vsocs. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 51(5):851–863, 2004. 2

[60] Angel Rodríguez-Vázquez, Rafael Domínguez-Castro, Francisco Jiménez-Garrido, Sergio Morillas, Alberto García, Cayetana Utrera, Ma. Dolores Pardo, Juan Listan, and Rafael Romay. *A CMOS Vision System On-Chip with Multi-Core, Cellular Sensory-Processing Front-End*, pages 129–146. Springer US, Boston, MA, 2010. 2

[61] T. Roska and L.O. Chua. The cnn universal machine: an analogic array computer. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 40 (3):163–173, 1993. 2

[62] Abu Sebastian, Manuel Le Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou. Memory devices and applications for in-memory computing. *Nature Nanotechnology*, 15(7):529–544, 2020. 2

[63] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Convolutional lstm network: a machine learning approach for precipitation nowcasting. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, page 802–810, Cambridge, MA, USA, 2015. MIT Press. 5

[64] T. Sikora. Mpeg digital video-coding standards. *IEEE Signal Processing Magazine*, 14(5):82–100, 1997. 2

[65] Haley So, Julien N. P. Martel, Piotr Dudek, and Gordon Wetzstein. Mantissacam: Learning snapshot high-dynamic-range imaging with perceptually-based in-pixel irradiance encoding. In *IEEE International Conference on Computational Photography (ICCP)*, 2022. 6

[66] Suraj Srinivas and R. Venkatesh Babu. Data-free parameter pruning for deep neural networks. In *BMVC*, 2015. 2

[67] Edward Stow, Riku Murai, Sajad Saeedi, and Paul Kelly. Cain: Automatic code generation for simultaneous convolutional kernels on focal-plane sensor-processors, 2021. 2

[68] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, page 4278–4284. AAAI Press, 2017. 2

[69] Cheng Tai, Tong Xiao, Xiaogang Wang, and E Weinan. Convolutional neural networks with low-rank regularization. *arXiv: Learning*, 2016. 2

[70] Ashraf Takla and C.K. Lee. Mipi c-physm d-physm dual mode subsystem performance and use cases. MIPI Alliance Developers Conference, DEVCON, 2027. 1

[71] Wei Tang, Gang Hua, and Liang Wang. How to train a compact binary neural network with high accuracy? In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, page 2625–2631. AAAI Press, 2017. 2

[72] Pascal Vivet, Eric Guthmuller, Yvain Thonnart, Gael Pillonnet, Guillaume Moritz, Ivan Miro-Panades, Cesar Fuguet, Jean Durupt, Christian Bernard, Didier Varreau, Julian Pontes, Sebastien Thuries, David Coriat, Michel Harrand, Denis Dutoit, Didier Lattard, L. Arnaud, J. Charbonnier, Perceval Coudrain, and Fabien Clermidy. 2.3 a 220gops 96-core processor with 6 chiplets 3d-stacked on an active interposer offering 0.6ns/mm latency, 3tb/s/mm 2 inter-chiplet interconnects and 156mw/mm 2 @ 82pages 46–48, 2020. 1

[73] Matthew Z. Wong, Benoit Guillard, Riku Murai, Sajad Saeedi, and Paul H. J. Kelly. Analognet: Convolutional neural network inference on analog focal plane sensor processors. *https://arxiv.org/abs/2006.01765*, 2020. 2

[74] Bichen Wu, Forrest Iandola, Peter Jin, and Kurt Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. pages 446–454, 2017. 2

[75] Zhanghao Wu*, Zhijian Liu*, Ji Lin, Yujun Lin, and Song Han. Lite transformer with long-short range attention. In *International Conference on Learning Representations (ICLR)*, 2020. 2

[76] Akos Zarandy. *Focal-Plane Sensor-Processor Chips*. Springer, 2011. 1

[77] Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):1514–1541, 2018. 4

[78] Wancheng Zhang, Qiuyu Fu, and Nan-Jian Wu. A programmable vision chip based on multiple levels of parallel processors. *IEEE Journal of Solid-State Circuits*, 46(9): 2132–2147, 2011. 2

[79] Yifan Zhang, Congqi Cao, Jian Cheng, and Hanqing Lu. Egogesture: A new dataset and benchmark for egocentric hand gesture recognition. 20(5):1038–1050, 2018. 8

[80] Guo Bing Zhou, Guo Bing Zhou, Chen Lin Zhang, and Zhi Hua Zhou. Minimal gated unit for recurrent neural networks. In *International Journal of Automation and Computing*, pages 13, 226–234. Springer, 2016. 3, 5

[81] Hao Zhou, Jose M. Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *Computer Vision – ECCV 2016*, pages 662–677, Cham, 2016. Springer International Publishing. 2

[82] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv e-prints*, art. arXiv:1606.06160, 2016. 2