AnalogNet: Convolutional Neural Network Inference on Analog Focal Plane Sensor Processors

Matthew Z Wong*
matthew.wong@live.com.sg
Department of Computing, Imperial College

Benoit Guillard*
benoit.guillard@epfl.ch
School of Computer and Communication Sciences, EPFL

Riku Murai
riku.murai15@imperial.ac.uk
Department of Computing, Imperial College

Sajad Saeedi
s.saeedi@ryerson.ca
Ryerson University

Paul H J Kelly
p.kelly@imperial.ac.uk
Department of Computing, Imperial College

ABSTRACT

We present a high-speed, energy-efficient Convolutional Neural Network (CNN) architecture utilising the capabilities of a unique class of devices known as analog Focal Plane Sensor Processors (FPSP), in which the sensor and the processor are embedded together on the same silicon chip.

Unlike traditional vision systems, where the sensor array sends collected data to a separate processor for processing, FPSPs allow data to be processed on the imaging device itself. This unique architecture enables ultra-fast image processing and high energy efficiency, at the expense of limited processing resources and approximate computations.

In this work, we show how to convert standard CNNs to FPSP code, and demonstrate a method of training networks to increase their robustness to analog computation errors. Our proposed architecture[1], coined AnalogNet, reaches a testing accuracy of **96.9%** on the MNIST handwritten digits recognition task, at a speed of **2260 FPS**, for a cost of **0.7 mJ per frame**.

KEYWORDS

analog computations, convolutional neural networks, embedded computer vision, energy efficiency, high frame rate, inference.

1 INTRODUCTION

Despite their spectacular successes, today's deep neural networks suffer from what has been termed the 'inference efficiency' problem [14]. While these networks perform extremely well when running on specialised hardware such as GPUs, they are in many cases not fast or energy efficient enough to be effectively deployed for real-time applications on less powerful hardware [25]. This has limited the range of potential applications, given that many artificial intelligence and (AI) robotics applications operate in real-time and yet are also power-constrained.

In this paper, we develop and implement a fast and energy efficient Convolutional Neural Network architecture on a class of devices known as Analog Focal-Plane Sensor-Processors (FPSPs). FPSPs integrate the light sensor and the early stage processing of a traditional vision system, by augmenting each photo-diode with rudimentary analog computation capabilities. Computations are said to happen *on the focal plane*. This architecture reduces the need for image data to be transferred to a separate processing device, thus offering the potential for high frame rates (1,000-100,000 FPS, frames per second) and low power consumption. On the other hand, FPSPs are very constrained in terms of memory availability, and only allow for noisy computations because of their analog nature. This paper's experimental work was conducted on the SCAMP-5 FPSP [4].

We introduce novel techniques to convert standard neural network layers to FPSP code, and demonstrate a method of training networks to increase their robustness to the effects of approximate computation. To summarise, the contributions of this paper are:

- a custom training process, providing neural networks parameters for an FPSP, and robust to noisy computations. The training process is general and is independent of a particular FPSP device.
- an optimised and adaptable implementation of CNNs on an FPSP, relying on the strengths of the hardware to demonstrate high throughput and energy efficiency, achieving handwritten digit recognition on the MNIST dataset [20] at 96.9% accuracy at 2260 FPS, using only 0.7mJ per recognised digit.

The rest of the paper is organised as follows: Section 2 covers background and related work. Section 3 describes the training method used for the CNN, taking into account hardware properties. Section 4 explains further improvements for inference. Sections 5 outlines experimental results and comparisons, and finally Section 6 presents the conclusions and future directions.

2 BACKGROUND AND RELATED WORK

2.1 Analog Computation

In digital computing, multiple distinct binary signals (bits) are used to represent a state or a number. In contrast, in current-mode analog computing, electrical charge is used to store a value, which can be read as an electrical current. The hardware needed to do arithmetic computation on analog signals is considerably simpler than on digital systems. For instance, to add two analog values, currents are joined from two sources representing the original values, while when computing in digital form using two 8-bit numbers, an adder needs many transistors (between 6 and 28 transistors per bit depending on circuit design [29]).

*Both authors contributed equally to this research while at Imperial College London.
[1]An implementation can be found at https://github.com/brouwa/CNNs-on-FPSPs

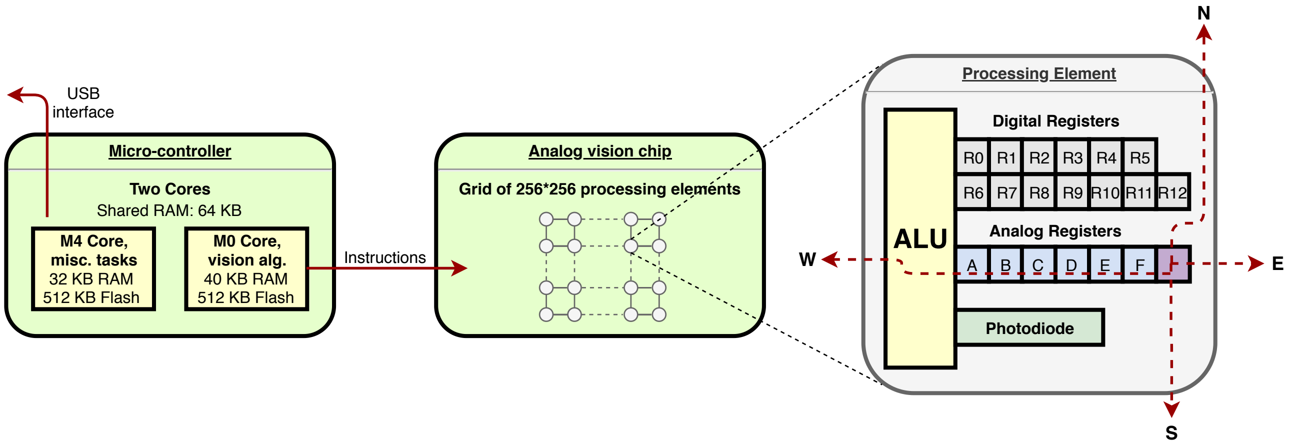

Figure 1: SCAMP5 vision system architecture: A digital micro-controller broadcasts instructions to an array of analog processing elements (PE). Each PE has an Arithmetic Logic Unit (ALU), seven local registers, and data links to its 4 neighbours.

2.2 Analog Focal Plane Sensor Processors

Focal-Plane Sensor-Processor (FPSP) chips are a special class of imaging device in which the sensor array and processor array are embedded together on the same integrated circuit [30]. Unlike traditional vision systems, in which sensor arrays send collected data to a separate processor for processing, FPSPs allow data to be processed in place on the imaging device itself. This architecture enables ultra-fast image processing even on small, low-power devices, because costly transfers of large amounts of data are no longer necessary. Some examples of FPSPs include the ACE16k [21] and the MIPA4k [26], as well as the SCAMP-5 system [4] used to carry out the experiments described in this paper.

Figure 2: A real example of noisy computation on the focal plane of SCAMP5. Left: crop of size 10×10 of a supposedly uniform input array, with value 100. Right: result after applying 4 times division by 2 and multiplication by 2 on this input. We notice a global shift (right image is 30% darker), and an increased amount of noise (left image standard deviation is approx. 2, whereas it is 6.3 for the right image, with one pixel being completely off, due to computation noise).

The SCAMP-5 device is made of a traditional ARM Cortex M4 + M0 dual core digital micro-controller, connected to a light sensitive analog processor chip (the vision chip). The M4 core is responsible for sending instructions to the analog chip, while the M0 core is in charge of miscellaneous tasks such as interfacing. The vision chip consists of a 256×256 grid of pixel-processors, or Processing Elements (PEs). Arithmetic and logic instructions sent by the digital micro-controller are executed in parallel by all PEs on the array (see Figure 1), on the data they locally hold. The computations are massively parallel, at the pixel level: an FPSP is a Single Instruction, Multiple Data (SIMD) computer. Additionally, the PE array is also provided with an interconnection network, allowing each PE to access values stored in its 4 neighbouring PEs' registers (see Figure 1).

Each PE has a light sensitive diode, so the array functions as an image sensor, and is enabled with an Arithmetic Logic Unit (ALU). Each ALU acts on its PE's memory registers - only 13 digital registers each storing one bit, 7 analog registers each storing a voltage. The limited number of analog registers per pixel creates computational challenges in implementations.

The analog nature of the computations enables high processing rates, but it introduces noise to the result. An example is shown in Figure 2. Thus it is essential to account for the impact of the noise.

The idea of reducing digital data transfers from the imaging device to the processing device is also exploited by event based cameras. Classical event based cameras such as the DAVIS240 [3] or the Gen3 ATIS [27] only record and transfer per-pixel intensity changes.

2.3 Vision Algorithms for FPSPs

Vision algorithms serving various functions have been successfully implemented on analog FPSPs. Examples include FAST16 corner detection [8], four degrees-of-freedom visual odometry [12], and depth estimation using a focus-tunable liquid lens [22].

Bose *et al.* implemented a CNN for digit classification [2]. Their work differs from the result of this paper in that their computations are run on digital registers, using a special ternary weight framework for convolutions - whereas we use analog registers. This achieves high speed and energy efficiency, at the price of a decrease in accuracy compared to digital implementations running on traditional hardware.

Debrunner *et al.* describe AUKE [13], a code generator doing the automatic conversion of 2D convolution layers into FPSP code. Given a convolution filter, a target analog register and a set of auxiliary registers that can be used for the computations, AUKE produces the shortest program to applies the convolution on the target register. AUKE aims for the shortest possible program, for obvious execution speed reasons, but also to limit the introduction of noise to the result. The spatial extent of the convolutional filter can be of any size, and is not restricted to immediate neighbours only (3×3 convolutions). The limited instruction set of an FPSP however puts a restriction on the filter weights of the feasible convolutions. The work proposed here utilizes the AUKE framework to convert convolution kernels to FPSP executable codes.

2.4 Hardware Acceleration of Convolutional Neural Networks

CNN computation involves computationally demanding matrix multiplications and occurs in two modes: training and inference. While slow training may not limit their use in real-world applications, slow inference immediately becomes a bottleneck. Several attempts have been made to design customised chips capable of performing efficient convolutions [9], [28], [7], and [24].

For instance, Origami is a power efficient and high GOp/s/W (803 GOp/s/W) digital system, devised to compute on-chip neural networks [5]. Chakradhar et al. designed a hardware architecture for CNNs that dynamically optimizes for performance by configuring the hardware on-the-fly [6]. Similarly, Intel's Movidius Myriad 2 Neural Compute Stick (NCS), is composed of multiple cores configurable for different networks [17]. Eyeriss [10] is an energy efficient CNN accelerator chip, relying on data gating and compression. It runs the convolutional part of AlexNet [19] at 35 FPS with 278 mW power consumption.

In these designs, sensors and processor(s) are not co-located unlike in FPSPs. Instead, the hardware is configurable to run different networks.

3 PROPOSED METHOD: ANALOGNET

We present a method for training a CNN architecture for image classification, that is suitable for deployment on analog FPSPs, given the availability of a training dataset X. We suppose $X = \{(I_i, l_i)\}_{i=0}^{N}$, where each $I_i \in \mathbb{R}^{h*w*3}$ is an image, and $l_i \in \{1, ..., L\}$ its corresponding label. CNN architectures for image classification start with convolutional layers, followed by fully connected layers.

Our method is meant for use with analog FPSPs supported by a digital micro-controller. We propose rapidly carrying out convolutions in parallel on the FPSP, followed by the transmission of intermediate results to the digital micro-controller where fully-connected layers can be computed. This architecture, making use of convolutional layers followed by fully connected layers, is a standard method in image classifiers. Instead of slowly transferring each frame to a sequential processor, one can transfer sparse feature maps which are the result of a pixel-parallel computation on this frame. Sending only short vectors of scalar (corresponding to a traditionally flattened and pooled version of 2D feature maps) greatly reduces the data transfers to the micro-controller. This is what we intend to take advantage of, as visualised on Figure 3. We dub the resulting architecture AnalogNet.

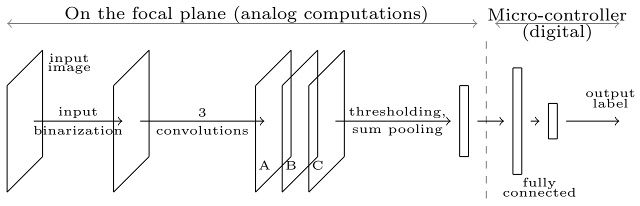
On the focal plane (analog computations)
Micro-controller (digital)
input image
input binarization
3 convolutions
A
B
C
thresholding, sum pooling
fully connected
output label

Figure 3: AnalogNet Architecture: convolutions are executed on the focal plane, and fully connected layers on the digital micro-controller. To reduce data transfers, only a short flat vector of scalar values is sent from the focal plane to the micro-controller, and not entire 2D feature maps.

The training method consists in the following steps:

- **Value Approximation and Custom Regularization**: A standard CNN is first trained on X. To do this, numerical approximation, with a custom regularization are applied, to ensure the resulting convolutions are implementable on the FPSP.
- **Noise-Inclusive Training**: Trained weights in the convolutional layer(s) from the previous step are then converted into FPSP code using the AUKE framework. At this stage, the network consists of FPSP code computing the convolutional layers, followed by the fully-connected layers. A noise function is applied to each FPSP instruction in the network, and the fully connected layers of the network are re-trained on this noisy signal.

Each of these steps are explained in detail below. The result is a CNN architecture that can be used to perform inference on an analog FPSP with minimal performance loss even in the presence of hardware noise.

3.1 Value Approximation and Custom Regularization

Neural networks are normally implemented via multiplication of floating point numbers. However, analog FPSPs such as SCAMP-5 do not support multiplication. Instead, integer multiplication must be carried out through repeated additions.

In such cases, we resort to value approximation for multiplication. As is the case for most FPSPs, the SCAMP-5 device only supports division by two and addition. By combining repeated additions and divisions, we can approximate the result of any floating-point multiplication operation to arbitrary precision [13]. For instance, multiplication of value a by 0.87 can be approximated by the following sequence:

$$0.87 \times a \approx (\frac{1}{2} + \frac{1}{4} + \frac{1}{8}) \times a = 0.785 \times a$$

While theoretically allowing arbitrarily precise multiplication, in practice greater precision requires a greater number of division operations, with each operation introducing additional noise. We therefore limit approximation to a depth n, indicating the maximum number of divisions allowed. In practice, we used $n = 2$.

We can train a neural network that accommodates these limitations, by using a regularization function that incentivises the selection of weights close to intervals of $\frac{1}{2^n}$, as such weights can be approximated with minimal error:

$$L_{reg}(\theta) = cos\left(\pi(2^{n+1} \cdot \theta + 1)\right) + 1 \quad (1)$$

For instance, by setting $n = 2$, we have $L_{reg}(\theta) = 0$ where $\theta \in \{0, \pm 0.25, \pm 0.5, ..\}$. We found that the most effective way of utilising the regularizer is to employ it in a *re-training* loop, carried out on a network first trained without a regularizer. Weights learnt were observed to be close to the target weights, whereupon a final rounding-off was carried out.

3.2 Noise-Inclusive Training

At this stage convolutional layers are frozen and converted into FPSP code using AUKE [13]. We re-train the fully-connected layers to account for the noisy execution of the convolutional layers on the focal plane. We now introduce our approach by which hardware noise can be integrated into the training loop.

A pre-defined noise model may not always be available (as was the case for this study), as formulating a noise model is a complex task. Instead we can infer an implicit noise model directly from empirical data. The converted FPSP code obtained above is loaded onto the FPSP device, which is itself placed in front of a computer screen. Convolutions are executed on the focal plane while the dataset images are shown in sequence to the FPSP device. For each image I_i, we collect the flattened output of the convolutional layers $f_i \in \mathbb{R}^s$, and associate it to its ground truth label l_i. This process is used to create a new dataset $X' = \{(f_i, l_i)\}_{i=0}^{N}$, corresponding to intermediate results of the CNN inclusive of true hardware noise. X' is used to train the fully-connected layers, which implicitly learn a model of the hardware noise and compensate for it.

4 INFERENCE OPTIMISATION

To take advantage of the pixel-parallel computations on the focal plane, we presented above how convolutions can be trained for implementation in an analog manner. Their execution on the focal plane is noisy, and their effect is only approximating what they were designed for. Conversely, the fully connected layers do not benefit from pixel parallel computations. Moreover, we expect them to compensate for the inaccurate nature of the convolutions that precede them in the network. For this reason, they are executed on the digital micro-controller which is adjacent to the focal plane.

In this section, we explain further how the result of convolutions are aggregated and sent to the micro-controller, and how the fully connected layers are executed.

4.1 Output Event Binning Process

The feature maps created by the convolutional part of the network are binarized: on the focal plane, locations above a certain threshold are assigned a 1, and others a 0. The corresponding threshold value is learned as part of the training process, and can be thought of as a bias for each convolution in the final layer. The binary feature maps are then collected by the adjacent digital micro-controller

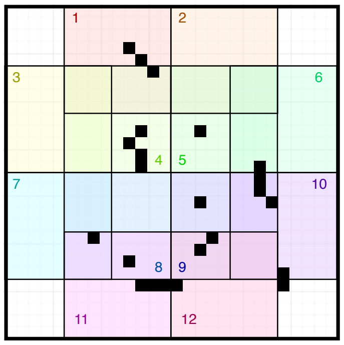

Figure 4: AnalogNet's event binning process. Events (i.e. activated binary features) fall into twelve overlapping bins. Black pixels correspond to a possible events configuration, here yielding the output vector (3, 0, 0, 5, 2, 1, 1, 2, 6, 4, 5, 3)**.**

in the form of a collection of *events*: each event is a tuple of 2D coordinates, corresponding to the location of an activated feature. Transferring the coordinates of activated features instead of whole arrays is an effective way of reducing data transfers from the analog vision chip to the digital micro-controller.

The micro-controller is responsible for spatially binning the received events, in order to transform each feature map into a vector of fixed length. Each event falls within a spatial bin, and increments its count. This step can be seen a sum-pooling a binary feature map. As shown on Figure 4, the binning procedure we propose uses 12 overlapping bins. Each convolutional feature map thus yields 12 values for the fully connected. This design uses the prior knowledge that the most informative locations of an image usually are in its central area. For this reason, we suggest to discard events located in the four corners, and to use different bins for the central events - to preserve more information about their location.

4.2 Fully Connected Layers Computations

The vectors resulting from the above binning process (one per feature map) are aggregated into a single vector, and given as input to a fully connected network. Classically, taking the argmax of the final layer's output produces the predicted label for the input image.

As introduced earlier, the FPSP's micro-controller is very rudimentary, and clocked at low frequency. Digital computations can thus easily become the speed bottleneck of an FPSP program. Consequently, to not hinder the great advantage brought by the execution of convolutions on the focal plane, the digital computations of the fully connected layers had to be optimised. The execution of a single fully connected layer can be seen as matrix-vector multiplication, with the addition of a point-wise bias - the matrix carries the weight of the layer, and the vector consists in its input values. A standard implementation relies on double loops for matrix-vector multiplication.

In our case, once the training is done, the network structure and weights are fixed. As a result, we know in advance (at compilation time), the number of iterations needed in each loop, and the coefficient of each multiplication. For this reason, we unroll the loop before compilation, and hard-code the weights in the unrolled loop.

Table 1: Comparison of computation time of a 2 layers fully connected network (36 input units, 50 hidden, 10 output), with three different methods used for matrix-vector multiplication, on the SCAMP-5 vision system.

Method	Time for fully-connected computation (μs)
Standard double loop	483
Unrolled loop	465
Ours: unrolled loop and hardcoded weights	**136**

This removes the overhead of accessing memory, and even frees some RAM. The length of the program is considerably increased, but the resulting speed-up is substantial, as presented in Table 1. This technique is equivalent to putting some manual optimisation upstream of the compiler.

5 EXPERIMENTS

We conducted experiments demonstrating that a network architecture trained and implemented using the method described above can successfully achieve competitive performance on FPSP hardware. We then evaluated it on the MNIST hand-written digit classification dataset [20], comparing the accuracy, inference time and power consumption of this architecture as implemented on FPSP (SCAMP-5), CPU (Intel i7-4930K), GPU (Nvidia GTX1080) and Vision Processing Unit (VPU, Intel Myriad2 Neural Compute Stick).

5.1 Architecture

Here we explain the reasoning and experiments resulting in the final choice of architecture for AnalogNet.

5.1.1 Single convolutional layer. We first showed the relevance of our training method in the case of a single layer CNN. In this single layer, the number of convolutions remains to be chosen. For this reason, we studied the accuracy of single layer CNNs using increasingly many convolution kernels (or *feature maps*). All other parameters are fixed: each convolution kernel is of size 3×3, each feature map is pooled according to the binning method presented above, and the fully connected layer has two layers - 50 hidden units, 10 output units.

Figure 5 reports the resulting testing accuracy, both for a digital implementation in Tensorflow [1], and for a final implementation on the SCAMP5 device. For each kernel number between 1 and 7, the full training process is run. For a given kernel number, the testing accuracy reported for the digital implementation is greater than the final implementation on the SCAMP-5 device: this is due to noisy nature of computations on the focal plane. As one could expect, a network with more kernels reaches a higher accuracy, but the returns of adding new kernels is diminishing. The drawbacks of using more kernels are an increased latency and power consumption of the convolutional part, which scale linearly with the number of feature maps it has. Figure 6 presents estimated inference duration and frames per second.

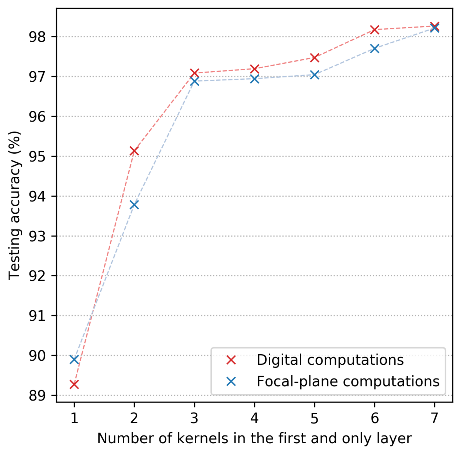

Figure 5: Testing accuracy, with increasingly many kernels in a single layer CNN.

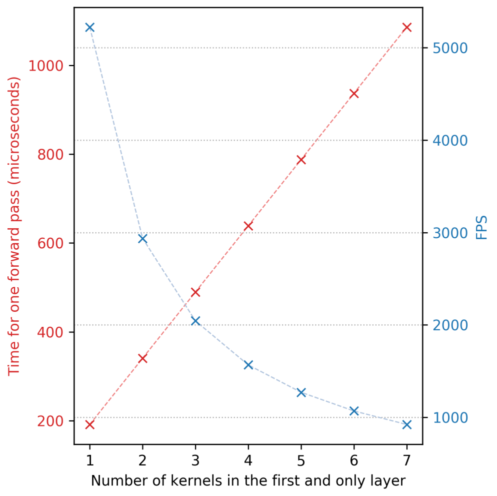

Figure 6: Estimation of the time required for one forward pass of a single layer network with increasingly many kernels, and the resulting FPS.

This systematic search of the architecture space of single layer CNNs clearly exhibits the trade-off at stake. We consider the three kernels, single layer CNN to be at the sweet spot of this design space. With more kernels, the networks fall below the landmark of 2000 FPS. Being at the inflexion point on Figure 5, adding more than 3 kernels gives diminishing returns in terms of testing accuracy.

5.1.2 Experimenting with two layers of convolutions. We then briefly explored the feasibility of implementing multiple convolutional layers on the SCAMP-5 device. The main design issue that arises is caused by the very limited register availability. In a traditional 2-layer CNN, each feature map of the second convolutional layer is the sum of the results of convolutions applied to each feature map in the first layer. This creates the need to store partial results, since all feature maps of the first layer are required until the very last feature map of the second layer is computed. This is a challenging problem, due to limited number of analog registers (AREGs) available for each pixel.

To store more partial intermediate feature maps, we tried storing some analog values in boolean digital registers (DREGs) for later use, when not presently needed for any computation. Using a quantisation procedure, one AREG's content can be encoded in binary and moved to multiple DREGs. The AREG can then be freed and used for current computations, while its content is saved and available for later use. An AREG stores a voltage representing integer values between -127 and 127, while a DREG stores either a 0 or a 1. These A/D conversions are extremely fast, and data remains on the focal plane - we do not give up to the benefits of FPSP computations. The precision level of the quantisation procedure can be adjusted, to choose between very frugal DREG use and high precision quantisation. Theoretically, the whole dynamic range of an AREG requires 8 DREGs to be precisely stored. In practice, operations are noisy, and using 8 DREGs provides an unnecessary amount of precision. In our experiments, we found 3 to 4 DREGs to be sufficient to capture the content of one AREG accurately enough.

Quantization is an idea that has already been explored to reduce the memory footprint of CNNs, as in [15], [31] or [11]. In our case, we used a two layer CNN, each layer having 3 feature maps. The quantisation procedure is used to binarise the first layer's feature maps when required. The best result we could achieve showed a final accuracy more than 1% below the single layer one that uses 3 kernels. We therefore decided not to use it as our final implementation of AnalogNet.

We also experimented another technique based on spatial pooling. Pooling feature maps decreases their spatial dimensionality, allowing for multiple ones to be stored on a single AREG. This idea of interleaving multiple arrays of data on a single AREG was explored by [23], but showed disappointing performance in our case.

Our hypothesis is that the poor performance of two-layer convolutional layers on the SCAMP-5 device is mostly due to noise accumulating at each operation. For this reason, the final AnalogNet architecture consists of a single convolutional layer, with 3 convolutions.

5.1.3 Noise accumulation. To demonstrate the negative effect of noise accumulation when running multiple layers of convolutions on the SCAMP-5 device, the performance of *depth-separable* CNNs is assessed. In depth-separable convolutions, feature maps are not combined with summation as with traditional convolutional layers. Instead, series of convolutions and non-linearities are applied to the input images, each one forming an independent thread in the computational graph. Depth-separable convolutions are most often used to reduce memory footprint of CNNs [16]. In our case, this greatly decreases the requirement to store partial results on the focal plane. Since feature maps are not combined, feature extraction is not as powerful and efficient as with standard convolutional layers. However, the accumulation of non-linearities and convolutions still helps in improving testing accuracy.

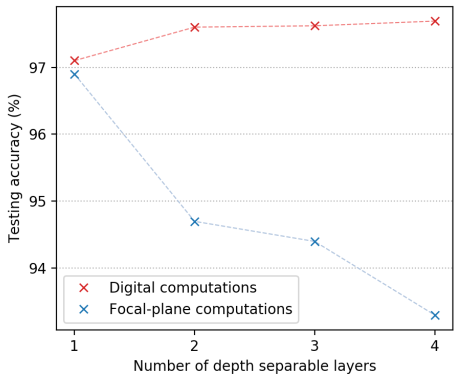

Figure 7: MNIST testing accuracy of CNNs using increasingly many layers of depth separable convolutions. Each layer is made of three 3*3 convolutions, and a ReLU activation function. The case with 1 layer corresponds to AnalogNet.

Figure 7 shows the MNIST testing accuracy of CNNs using increasingly many layers of depth separable convolutions, both in our noiseless digital simulations and on the SCAMP5 device. In simulations, having more depth-separable layers yields better results, with diminishing returns. On the SCAMP5 device, the exact inverse phenomenon happens, with results getting worse as each new layer is added.

This results confirm our hypothesis that despite being more powerful in theory - i.e. with noiseless and precise digital implementations - networks involving longer chains of computation perform poorly on the SCAMP5 device. The computation chain between the input image and the output feature maps is longer, and the information is supposed to be refined, but is in reality diluted in increasing amounts of noise. What really is detrimental here is the accumulation of noise.

5.2 Accuracy

In Table 2, we compare AnalogNet's accuracy on the MNIST test dataset with the accuracy reported by Bose et al. [2]. Similarly to the noise aware training process presented in Section 3, the FPSP is placed in front of a computer screen, which is displaying the whole MNIST test set. The label predicted by the SCAMP-5 is compared to the ground truth one.

Our solution achieves decent accuracy, at 96.9%. However, although this aspect is not quantified, the method from Bose et al. seems more robust to image misalignment and variable point of view. We also compared to a digital (noiseless) implementation of AnalogNet, and RMDL [18], which achieves state of the art MNIST classification accuracy, but relies on a much more complex architecture than AnalogNet.

5.3 Inference Time

For the SCAMP-5, VPU, CPU and GPU, inference time was estimated by measuring the average per-frame computation time

Table 2: MNIST testing accuracy comparison. A *digital* device refers to either CPU, GPU or VPU.

Architecture	Hardware	Testing accuracy
AnalogNet	SCAMP-5	**96.9%**
Bose [2]	SCAMP-5	94.2%
AnalogNet	digital	97.1%
Bose [2]	digital	95.4%
RMDL [18]	*digital*	*99.82%*

Table 3: Estimated per-frame computation time in microseconds, for AnalogNet architecture. Image data retrieval time is excluded for CPU, GPU, and VPU, while it effectively is zero for SCAMP-5.

Hardware	Inference time
SCAMP-5	442
VPU	422
GPU	758
CPU	651

incurred on each system. It was measured by their respective system utilities. Table 3 shows the recorded per-frame computation time. Excluding the substantial time cost of capturing and retrieving the image data from a separate camera for computation on the CPU, GPU and VPU, the SCAMP-5 FPSP is already much faster than the CPU and the GPU, and reaches almost the same speed as the VPU. Note that these figures include the total time for inference, i.e. both the convolutional part and the fully connected one - and the transfer of binary features from the focal plane to the micro-controller in the case of SCAMP-5.

In practice this data retrieval bottleneck is a major limitation on frame rates achievable on CPU, GPU and VPU. The FPSP does not face this bottleneck because image data is processed in place at the pixel level. Depending on the application, the image retrieval process can be fully parallelized, and be faster than the inference time. In these cases, in the continuous regime, the data retrieval bottleneck vanishes. In other situations, a real time, low latency prediction is needed - such as in high speed robotics. In these cases, the latency of a standard camera can become the first obstacle in achieving low latency. For instance, transferring an 8 bit image of size 28×28 pixels from the sensor of the SCAMP-5 camera (which can also be used as a standard webcam) to a nearby computer via USB 2.0 takes 1584 μs.

Taking this into account shows the clear advantage of realising most computations directly on the light sensor, in an analog manner. The image retrieval latency is simply reduced to zero. Implemented on the SCAMP-5 device, AnalogNet infers the label of an MNIST digit in 442 μs, giving an framerate of 2260 FPS. The latter is a calculated figure, since our computer display setup cannot achieve this refresh rate.

AnalogNet compares very favourably to Bose [2], which reports a maximum frame-rate of 210 FPS, equivalent to a minimum inference time of approximately 4700μs. Note that the GPU inference time is disappointingly high. Our speculation is that GPUs are optimised for treating large batches of images, but not single instances sequentially, in an online manner.

Table 4: Estimated per-frame energy consumption, in milliJoules. Energy consumption incurred in capturing and retrieving an image is excluded for CPU, GPU, and VPU, while the SCAMP-5 figure is 'all-in'. For the VPU and CPU, energy drawn by the host computer is conservatively lower bounded.

Hardware	Host	Inference	Total
SCAMP-5	0	0.7	**0.7**
VPU	≥0.9	0.3	≥1.2
GPU	≥1.5	28.6	≥30.1
CPU	0	18.9	18.9

5.4 Energy Consumption

Per-frame energy consumption was determined by estimating the power drawn by each device multiplied by the per-frame inference time. The power draw was measured using Nvidia SMI for the GPU, Intel RAPL for the CPU, and an USB power meter for the VPU and the SCAMP-5. Table 4 shows the estimated per frame energy consumption. AnalogNet inference on the SCAMP-5 FPSP uses far less energy per-frame than a CPU or a GPU and is comparable to a low-power VPU.

Note that the FPSP figure is an 'all-in' one, while the CPU, GPU and VPU estimates do not consider the energy consumption required for image capture and data transfer, i.e. the energy cost of capturing and retrieving the image data from a separate camera or a digital storage. Furthermore, energy consumption estimates for the VPU and GPU have to account the power drawn by a host computer. This is conservatively lower bounded by 2 Watts, which is the power drawn at idle by a Raspberry Pi 3.

5.5 Discussion: Limitations

While state-of-the-art CNN architectures use many convolutional layers, our work was limited to a single convolutional layer, thereby restricting our CNN experiments to the MNIST dataset. There are two fundamental challenges involved in computing additional layers on an analog FPSP:

- **Data Loss:** In analog systems, every computation introduces additional noise to the original signal. Additional layers naturally require a greater number of computations.
- **Data Sparsity:** Data is stored geographically on the FPSP, resulting in pooled data being spatially dispersed. To continue computation, the data needs to be spatially compressed, introducing additional noise.

In this application, only the central 28×28 subarray of a 256×256 device is used, and hence the computational capabilities of the focal plane are not fully exploited.

6 CONCLUSION

In this paper, we demonstrated the potential for analog FPSPs to bring computer vision capabilities embedded applications that are constrained by power budgets and requiring very low latency, such as robotics. We developed a method for designing customised CNNs for use on analog FPSPs that are able to overcome the challenges inherent in working with such devices. We performed experiments on the MNIST dataset, showing that digit recognition using a CNN can indeed be carried out on an analog FPSP. By using FPSPs to process data in the focal plane and limiting data movement only to important features, we showed that we can increase the frame-rate and reduce the overall power consumption significantly.

We realise that the greatest limitation of our work is the very simple architecture of AnalogNet in regard to the current state of deep learning. Further work should focus both on designing additional network architectures capable of operating within current hardware constraints, and on developing new FPSP hardware designs that alleviate the limitations encountered in this paper. One significant hardware contribution would be more memory at each processing element (PE). Currently, SCAMP-5 PEs have access to extremely limited memory. Additional memory (analog or digital) would facilitate implementation of complex neural networks, with an increase in the amount of memory available directly corresponding to an increase in the ability of the FPSP to handle convolutional layers of greater breadth.

Accounting for noise for CNN inference also remains an important issue. It could better be integrated in the training, if a precise differentiable noise model was available. Another potential solution could be to mitigate the effects of noise at inference time, for instance by running each operation multiple times and averaging their result.

ACKNOWLEDGMENTS

This research is supported by Engineering and Physical Sciences Research Council [grant number EP/K008730/1], A Panoramic View of the Many-core Landscape (PAMELA). We thank Piotr Dudek and his colleagues, Stephen Carey and Jianing Chen, at the University of Manchester for kindly providing access to the SCAMP device and simulator.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A System for Large-scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16).* 265–283.

[2] Laurie Bose, Jianing Chen, Stephen J Carey, Piotr Dudek, and Walterio Mayol-Cuevas. 2019. A Camera That CNNs: Towards Embedded Neural Networks on Pixel Processor Arrays. In *Proceedings of the IEEE International Conference on Computer Vision.* 1335–1344.

[3] Christian Brandli, Raphael Berner, Minhao Yang, Shih-Chii Liu, and Tobi Delbrück. 2014. A 240 x 180 130 dB 3 μs Latency Global Shutter Spatiotemporal Vision Sensor. *J. Solid-State Circuits* 49, 10 (2014), 2333–2341.

[4] S. J. Carey, A. Lopich, D. R. W. Barr, B. Wang, and P. Dudek. 2013. A 100,000 fps vision sensor with embedded 535GOPS/W 256*256 SIMD processor array. In *2013 Symposium on VLSI Circuits.* C182–C183.

[5] L. Cavigelli and L. Benini. 2017. Origami: A 803-GOp/s/W Convolutional Network Accelerator. *IEEE Transactions on Circuits and Systems for Video Technology* 27, 11 (2017), 2461–2475.

[6] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. 2010. A Dynamically Configurable Coprocessor for Convolutional Neural Networks. In *Proceedings of the Annual International Symposium on Computer Architecture* (Saint-Malo, France) *(ISCA '10).* ACM, New York, NY, USA, 247–257.

[7] H. G. Chen, S. Jayasuriya, J. Yang, J. Stephen, S. Sivaramakrishnan, A. Veeraraghavan, and A. Molnar. 2016. ASP Vision: Optically Computing the First Layer of Convolutional Neural Networks Using Angle Sensitive Pixels. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 903–912.

[8] Jianing Chen, Stephen J Carey, and Piotr Dudek. 2017. Feature Extraction using a Portable Vision System. In *Workshop on Vision-based Agile Autonomous Navigation of UAVs.* Vancouver, BC, Canada.

[9] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam. 2014. DaDianNao: A Machine-Learning Supercomputer. In *Annual IEEE/ACM International Symposium on Microarchitecture.* 609–622.

[10] Yu-Hsin Chen, Tushar Krishna, Joel Emer, and Vivienne Sze. 2016. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. In *IEEE International Solid-State Circuits Conference, ISSCC.* 262–263.

[11] Matthieu Courbariaux and Yoshua Bengio. 2016. BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *CoRR* (2016). arXiv:1602.02830

[12] Thomas Debrunner, Sajad Saeedi, Laurie Bose, Andrew J Davison, and Paul H J Kelly. 2019. Camera Tracking on Focal-Plane Sensor-Processor Arrays. In *Workshop on Programmability and Architectures for Heterogeneous Multicores (MULTIPROG).* Vancouver, BC, Canada.

[13] Thomas Debrunner, Sajad Saeedi, and Paul H. J. Kelly. 2019. AUKE: Automatic Kernel Code Generation for an Analogue SIMD Focal-Plane Sensor-Processor Array. *TACO* 15, 4 (2019), 59:1–59:26.

[14] Google. 2018. Fixed Point Quantization | TensorFlow. https://www.tensorflow.org/performance/quantization

[15] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1510.00149

[16] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. (4 2017). https://doi.org/arXiv:1704.04861

[17] Intel Movidius. 2018. Intel Movidius Myriad VPU. https://www.movidius.com/myriad2.

[18] Kamran Kowsari, Mojtaba Heidarysafa, Donald E Brown, Kiana Jafari Meimandi, and Laura E Barnes. 2018. Rmdl: Random multimodel deep learning for classification. In *Proceedings of the 2nd International Conference on Information System and Data Mining.* 19–28.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F Pereira, C J C Burges, L Bottou, and K Q Weinberger (Eds.). Curran Associates, Inc., 1097–1105.

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[21] G Linan, S Espejo, R Dominguez-Castro, and A Rodriguez-Vazquez. 2002. Architectural and basic circuit considerations for a flexible 128×128 mixed-signal SIMD vision chip. *Analog Integrated Circuits and Signal Processing* 33, 2 (2002), 179–190.

[22] Julien Martel, Lorenz K. Muller, Stephen J Carey, Jonathan Muller, Yulia Sandamirskaya, and Piotr Dudek. 2018. Real-Time Depth from Focus on a Programmable Focal Plane Processor. *IEEE Transactions on Circuits and Systems I: Regular Papers* 65, 3 (2018), 925–934.

[23] Julien N. P. Martel, Miguel Chau, Matthew Cook, and Piotr Dudek. 2015. Pixel interlacing to trade off the resolution of a cellular processor array against more registers. In *European Conference on Circuit Theory and Design, ECCTD 2015, Trondheim, Norway, August 24-26, 2015.* 1–4.

[24] Eriko Nurvitadhi, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason Ong Gee Hock, Yeong Tat Liew, Krishnan Srivatsan, Duncan Moss, Suchit Subhaschandra, and Guy Boudoukh. 2017. Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.* 5–14.

[25] Nvidia. 2017. Accelerating AI Inference Performance in the Data Center and Beyond | The Official NVIDIA Blog. https://blogs.nvidia.com/blog/2017/09/25/ai-inference/

[26] Jonne Poikonen, Mika Laiho, and Ari Paasio. 2009. MIPA4k: A 64×64 cell mixed-mode image processor array. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on.* IEEE, 1927–1930.

[27] Prophesee. 2019. Prophesee Event Based Cameras. https://www.prophesee.ai/event-based-evk/. https://www.prophesee.ai/event-based-evk/ [Online; accessed 12-August-2019].

[28] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA).* 14–26.

[29] Subodh Wairya, Rajendra Kumar Nagaria, and Sudarshan Tiwari. 2012. Performance Analysis of High Speed Hybrid CMOS Full Adder Circuits for Low Voltage VLSI Design. *VLSI Design* 2012 (2012), 7:7–7:7.

[30] Akos. Zarandy. 2011. *Focal-plane sensor-processor chips.* Springer.

[31] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. *CoRR* (2017). http://arxiv.org/abs/1702.03044