High-speed Light-weight CNN Inference via Strided Convolutions on a Pixel Processor Array

Yanan Liu[12]
yanan.liu@bristol.ac.uk
Laurie Bose[2]
lb7943@bristol.ac.uk
Jianing Chen[3]
jianing.chen@manchester.ac.uk
Stephen J. Carey[3]
stephen.carey@manchester.ac.uk
Piotr Dudek[3]
p.dudek@manchester.ac.uk
Walterio Mayol-Cuevas[24]
Walterio.Mayol-Cuevas@bristol.ac.uk

[1] Bristol Robotics Laboratory
University of Bristol
Bristol, UK
[2] Visual Information Laboratory
University of Bristol
Bristol, UK
[3] School of Electrical & Electronic Engineering
University of Manchester
Manchester, UK
[4] Amazon, Seattle, USA

Abstract

Performance, storage, and power consumption are three major factors that restrict the use of machine learning algorithms on embedded systems. However, new hardware architectures designed with visual computation in mind may hold the key to solving these bottlenecks. This work makes use of a novel visual device: the pixel processor array (PPA), to embed a convolutional neural network (CNN) onto the focal plane. We present a new high-speed implementation of strided convolutions using binary weights for the CNN on PPA devices, allowing all multiplications to be replaced by more efficient addition/subtraction operations. Image convolutions, ReLU activation functions, max-pooling and a fully-connected layer are all performed directly on the PPA's imaging plane, exploiting its massive parallel computing capabilities. We demonstrate CNN inference across 4 different applications, running between 2,000 and 17,500 fps with power consumption lower than $1.5W$. These tasks include identifying 8 classes of plankton, hand gesture classification and digit recognition.

1 Introduction

Convolutional neural networks (CNN) already play a significant role in modern computer vision tasks such as image classification and object recognition. With the ever increasing prevalence of mobile and embedded devices, such as smartphones and mobile robots, there is a strong motivation to enable CNNs on portable lightweight devices [6, 14, 27].

However, state-of-the-art CNN-based methods are typically heavily GPU reliant, and difficult to deploy on the embedded systems without optimisation or modification [39]. Three main issues are the lack of parallel computation power, memory, and battery life, all of

© 2020. The copyright of this document resides with its authors.
It may be distributed unchanged freely in print or electronic forms.

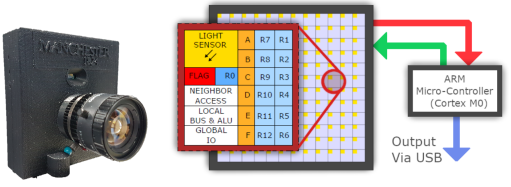

Figure 1: Left: the SCAMP-5d vision system used in this work. Right: SCAMP-5d's hardware architecture. The SCAMP-5d incorporates a 256×256 PPA array of pixel-processors, each containing light sensor, local memory registers and other functional components. A standard ARM processor provides overall program control.

which are required by computationally demanding CNN algorithms. Two potential solutions are (1) hardware acceleration [1, 2, 10] and (2) data compression in terms of storage and complexity using techniques such as network pruning and low-bit quantization of network weights [25, 41].

Rather than using a conventional approach in which a camera streams video frames to processing hardware, this paper focuses on implementing CNNs upon a novel, general-purpose, Pixel Processor Array (PPA) (Figure 1). Our approach takes advantage of the PPAs massively parallel architecture to efficiently execute a binary CNN. Image convolutions, activation functions, max-pooling and fully-connected layer are implemented upon the PPA. By adopting an "in-pixel" weight approach such as [5], our implementation is significantly faster than many existing works [4, 18, 37] and does not rely on external processing. Training is performed offline upon a standard PC while inference experiments are performed entirely upon the PPA. This work seeks to illustrate the potential high speed CNN applications that can be achieved upon such PPA devices.

Contributions: The main contributions of this work are: 1: A new image convolution implementation for PPAs, incorporating variable convolution stride to allow for faster inference times compared to previous works [3, 37], increasing the inference speed across various tasks depending upon the task's level of complexity. 2: Demonstration of our fast SCAMP-5 CNN implementation across a wider and more complex set of tasks than previous works, which had predominately focused upon only demonstrating MNIST classification. We demonstrate real-time hand gesture recognition, plankton classification from the National Data Science Bowl plankton dataset [22] along with digit recognition. PPA inference speed for our approach is extremely fast across all tasks, ranging from 2000 to 17500 fps.

2 Related Work

To achieve high performance CNN inference on embedded devices, a great amount of work has been carried out on network compression, hardware accelerators and unconventional visual sensors.

Network Compression: There are many types of quantization methods to compress the trained weights to binary or ternary values which significantly reduce the size of the model and speed up computation, such as the BinaryConnect [12], XNOR-Net [33], BinaryNet [11] and Ternary Weight Networks[24]. Another method, network pruning [20, 38] reduces the storage requirement of deep neural networks by getting rid of unimportant connections among neurons.

Hardware Accelerators: The on-going work on implementing hardware accelerators for efficient execution of CNN on edge devices has resulted in numerous architectures and prototypes proposed in recent years by academic groups, for example [2, 10, 19, 35, 40], as

well as commercially available NN accelerator IP blocks [16] or dedicated hardware devices [21, 36]. The need for co-optimisation of the architecture, from image sensor, through image signal processing, to NN acceleration is recognised as an important aspect of vision system design for embedded systems [22].

Unconventional Visual Devices: Recent works using unconventional visual devices for CNNs have mainly focused on Dynamic Visual Sensors (DVS) and PPAs. DVS sensors produce data in the form of sparse contrast-change events, that facilitate low-latency visual processing using external computational hardware [26, 28, 29]. PPA devices enable sensor-level computation. Bose *et al*. proposed a CNN for digit classification [4] implemented using binary computations in the PPA, and a CNN using in-pixel weights and analog computation [5]. The AnalogNet2 [18] extends the earlier work in [37], implementing a CNN which reaches 96.9% accuracy on the MNIST dataset at a speed of 2260 fps, but which requires all fully connected layers to be performed externally to the PPA array. CNN implementations on PPAs can be also found in [13] where automated code generation for efficient convolution kernels is presented.

3 SCAMP-5 Vision System

In this work, we implement our algorithms on the SCAMP-5 Pixel Processor Array (PPA) device [7]. Different from a conventional image sensor where images are read out and then processed externally to the sensor, the SCAMP-5 features on-board parallel processing, outputting computation results directly to a high-level controller. This on-board processing enables a range of potential applications, such as visual odometry [3], mobile robot tracking [17], proximity estimation [9], real-time depth estimation [30] and CNN inference [4].

Figure 1 illustrates the main hardware components within the SCAMP-5 system. The vision chip integrates 256×256 Processing Elements (PE). Each PE includes a light sensor, 7 analogue registers (A - F), 13 digital registers (R0 - R12), and arithmetic and logic operation units. All PEs execute identical instructions synchronously on their registers, enabling parallel image processing on both gray scale analogue and digital binary images. Data stored in one PE in the array can be accessed directly by its 4 neighbours (east, west, north, south). Moreover, some operations like event readout, flooding, Gaussian blur, and area summation are implemented in hardware to accelerate their operations. Instructions for the vision chip are dispatched by an ARM-based microcontroller with a Cortex M0 processor core. The system also integrates an additional ARM Cortex M4 core, providing IO services and running additional user programs. Serial IO buses, such as USB2.0, SPI, and UART, allow the output from the vision system to be sent directly to a variety of other devices [8]. The peak power consumption of the entire SCAMP-5d camera system is 2.3 W (The PPA chip consumes below 1.3 W and provides up to 655 GOPS performance [7]).

4 Approach

To achieve high-speed CNN inference, both the computation and weight-storage should be contained within the PEs of the processing array itself to fully exploit the PPA's parallelism and minimise data transfers. To this end, it is necessary to find a way to train the CNN with binary weights that can fit entirely within the PPA's array. This section describes the network training and implementation of high-speed CNNs for the SCAMP-5d PPA.

4.1 Convolutional Neural Network with Binary Weights

In our work, the BinaryConnect scheme [12] is adopted and used to train binary weight networks. This produces simplified binary neural networks, whose weights can be stored

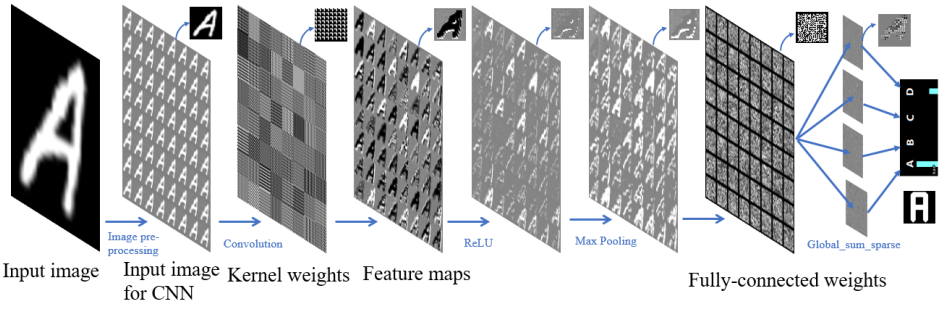

Figure 2: Parallel inference process by combining different registers and operations.

entirely within the memory registers of the PPA array, but which still achieves acceptable accuracy. Additionally these binary networks are trained without neuron bias, further simplifying the CNN implementation [31].

This training scheme generates 1-bit weights representing values $\{-1,1\}$ for both convolutional layers and fully connected layers. This allows rapid inference of various CNN layers to be performed using only native PPA arithmetic operations (additions/subtractions). The weights for convolutional and fully connected layers are directly stored in 1-bit digital registers on the array. This in-pixel weight approach first proposed in [5] allows for parallel and efficient implementation of CNN layers compared to methods which sequentially read weights from the controller [4, 18, 37].

Figure 2 shows the inference process of a CNN on SCAMP-5, with each step executed upon the image plane. First, input images are uploaded or directly captured into the PEs of the array. To execute many convolution filters in parallel, this input image is pre-processed at runtime on the array, being down-scaled and then replicated to fill all 256×256 processing elements. In Figure 2 the input image is shrunk to 32×32 and replicated 64 times across the array. Each replicated image is associated with a different kernel filter, with 64 kernel filters arranged in-line with the 64 replicated image blocks. From this the convolutional layer generates 64 feature maps in parallel, followed by parallel activation function (ReLU) and max-pooling. Weights for the fully-connected layer are stored upon digital registers similar to that of the convolutional layer and are multiplied in parallel with their associated activation data. Finally, approximated sums of all pixels associated with each label are calculated by using 'sparse global summation' on the SCAMP-5 array, with the largest resulting sum representing the CNN's understanding of the image.

4.2 Implementation of Convolutional Layer

This paper implements the image convolution in a way that takes full advantage of the speed offered by the PPA parallel processing resources. Each kernel filter is replicated to the size of each input image block (Figure 2). Then the source image is "multiplied" by the corresponding kernel filters coefficients (+1 or -1) in parallel, with the convolution result obtained by the summation of pixels in the filter block. Moreover, strided convolutions (i.e. stride 1, 2, or 4) can be applied here for different applications to speedup inference process. This method allows the convolutional layer to be performed entirely on the PPA array using only native addition, subtraction, and image shifting operations.

Referring to Figure 3, 4×4 binary kernel filters for the convolutional layer are stored in 4×4 PE blocks using digital registers. Efficient multiplication of stored data by these binary weights can then be performed. The detailed layout of the 4×4 kernel filters is illustrated in

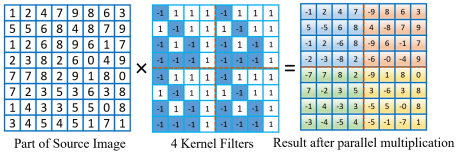

Figure 3: The parallel implementation of multiplication. Each pixel of source image either remains unchanged or becomes negative according to the binary weights stored directly in registers.

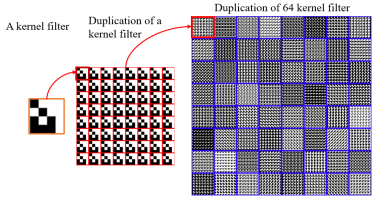

Figure 4: The layout of 64 binary kernel filters in a digital register. Each filter can extract corresponding features from the initial input images to the downstream layers.

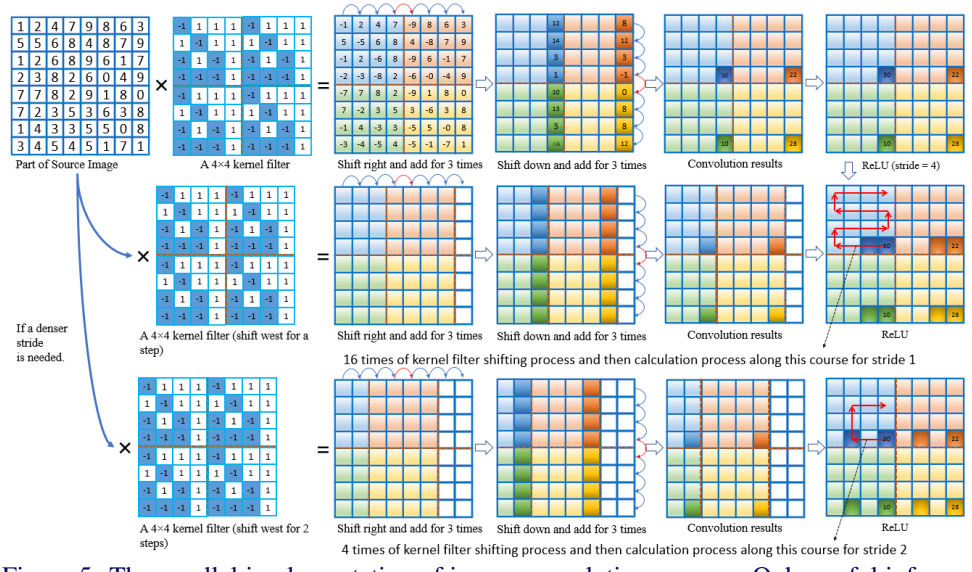

Figure 5: The parallel implementation of image convolution process. Only useful information is stored at the right bottom corner in every 4×4 block. The final result in this example can be regarded as a CNN with a stride = 4. Stride can also be set to 1 or 2 according to the requirements of different applications considering efficiency and accuracy.

Figure 6: Left: 64 feature maps generated in parallel by the convolutional layer on PPA. Right side: left to right: input images, images after convolution, images after activation function ReLU, images after max pooling.

Figure 4, showing how each of the 64 kernels is replicated multiple times to fill the 32×32 block of PEs holding the image it will operate on. Following the result of image multiplication, image convolutions (of stride 4) on the PPA are calculated by iteratively performing

image shifting and addition a total of 6 times. As shown in Figure 5, the convolution results are stored in the bottom right corner of each 4×4 block. Convolutions of stride 1 and 2 can be calculated by simply repeating this process for stride 4 multiple times (×16 for stride 1, ×4 for stride 2. The second and third rows in Figure 5) illustrate this, using a different shifted copy of the kernel filter for each iteration. It should be noted, for each iteration, only one pixel out of 4×4 block stores the correct value for image convolution. Hence, some degree of power efficiency is sacrificed compared to calculating 16 valid convolutional results for once. Despite this, even at stride 1 our implementation is still significantly faster at performing convolutional layers than many previous works [4, 18, 37] as multiple convolutional filters are executed in parallel across the array rather than sequentially.

4.3 Activation function and Max-pooling layer

We make use of the rectified linear unit (ReLU) as it is both a common choice of activation function and can be efficiently performed in parallel across the SCAMP-5d array, using a short sequence of native operations. Max-pooling can similarly be implemented in an efficient parallel manner on the PPA array, using simple shift and addition operations. Specifically 2×2 is achieved by comparing each PE to is north neighbour in parallel, overwriting each PEs data with the larger of the two values. This process is then repeated for each east neighbour, resulting in every PE containing the greatest value in its local 2×2 block.

Algorithm 1 Parallel 2×2 max-pooling.

INPUT: Register *B*
OUTPUT: Register *F*
D = Move B to the north for one pixel
E = D - B
WHERE (E > 0)
B = D
D = Move B to the east for one pixel
E = D - B
WHERE (E > 0)
B = D
return *B*

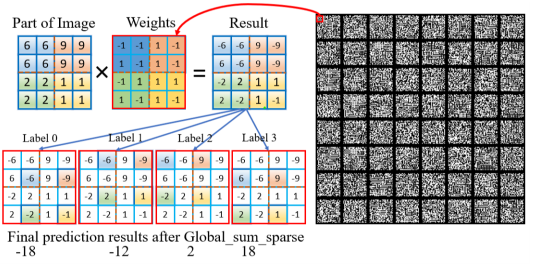

Figure 7: The parallel implementation of fully-connected layer.

4.4 Parallel Fully-connected Layer

The first step in performing a fully-connected layer is multiplication between max-pooled image data and the fully-connected weights as shown in Figure 7. The image on the right visualises the binary weights of the fully-connected layer, encoded in 1-bit digital registers. The key to this part lies in the layout of the fully-connected weights and max-pooled image. In this schematic diagram , the fully-connected weights for 4 labels are stored in the 2×2 blocks. After multiplication, pixels that contain information for each label are spread in a checkered pattern. The native *global_sum_sparse* function can return the approximated summation of values from a given selection of analogue registers. This can then be used to get the approximated sum of pixels associated with each label. The biggest value out of these global summations gives the final prediction of the neural network.

5 SCAMP-5 Inference, Experiments, and Evaluation

This section demonstrates four experiments[1]: plankton classification, real-time hand-gesture recognition, rock-paper-scissors and digit recognition. Each is demonstrated using a different CNN network running upon SCAMP-5, using either 64 4×4 or 16 4×4 kernel filters in the convolutional layer.

5.1 Plankton classification

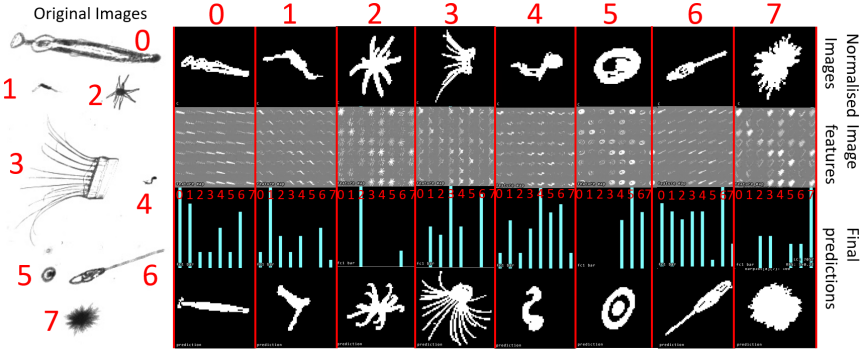

Figure 8: CNN inference performing plankton classification on SCAMP-5d. Plankton images are normalised in size and centred before being input into the PPA array as shown in the top row for each class. The second row shows the max-pooled data fed into the following fully-connected layer. Rows three and four show the final predictions for each class and an example image from the correct class.

Plankton organisms are at the bottom of the food chain in the marine ecosystem, real-time monitoring of which can be used to determine ocean health levels [32]. Due to the capacity of the proposed neural network, we select 8 of the most numerous plankton species (0:chaetognaths, 1:coppods, 2:echinoderm, 3:hydromedusae, 4:pelagictunicate, 5:protists, 6:siphonophores and 7:trichode-smium) from an imbalanced scale plankton database considering the number of samples for each species[2], to show the performance of the proposed CNN.

class	0.chaetognaths	1.coppods	2.echinoderm	3.hydromedusae	4.pelagictunicate	5.protists	6.siphonophores	7.trichodesmium
0.chaetognaths	**188**	0	1	2	1	0	8	0
1.coppods	3	**176**	1	0	14	2	4	0
2.echinoderm	0	3	**182**	0	1	1	4	0
3.hydromedusae	1	3	5	**181**	0	3	7	0
4.pelagictunicate	0	26	2	1	**138**	10	23	0
5.protists	0	0	1	1	6	**183**	8	1
6.siphonophores	52	12	9	8	24	9	**85**	1
7.trichodesmium	0	0	17	1	0	20	2	**160**

Table 1: Confusion matrix for plankton classification with 200 samples for each label.

As shown in the Figure 8, we utilise 64 4×4 kernel filters, acting upon 32×32 input images with 2×2 max-pooling. After training with binary weight neural network on a computer, the validation accuracy is 83.6% and 80.5% on the PPA. The reason for the accuracy gap lies in the inevitable computation error on analogue registers[15] and approximated analogue summation used in the fully-connected layer. Moreover, Table 1 visualises the perfor-

[1]Experimental video: https://youtu.be/3Qh4ujmsh7E
[2]Dataset available at https://www.kaggle.com/c/datasciencebowl

Component	Plankton	Hand Gesture	Roshambo	0 or 1
Image capturing and thresholding (μs)	-	6	6	-
Character duplication (μs)	28	28	28	28
Image convolution(μs)	165	165	52	12
Activation function (μs)	5	5	5	5
Max pooling (μs)	4	36	12	-
First fully-connected layer (μs)	47	213	18	12
Second fully-connected layer (μs)	-	24	-	-
Total running time (μs)	249	478	121	57
Inference speed (fps)	**4,016**	**2,092**	**8,264**	**17,543**
Accuracy (Computer/SCAMP-5d)	83.6%/80.5%	98.7%/-	97.73%/-	99.7%/99.1%
Number of binary weights	100,608	921,664	43,264	29,056

Table 2: Computation time, performance and weights for different neural networks. Notice that all the live demos are demonstrated with a fixed distance between the SCAMP-5d and the hand.

mance of the proposed CNN in SCAMP-5 on 1600 samples. The accuracy for siphonophores and pelagictunicate is lower due to their visual similarity with chaetognaths and coppods respectively, which, as a whole, is in line with the bar chart shape in Figure 8.

5.2 Real-time hand gesture recognition

Figure 9: Samples of eight common hand gestures for classification with PPA device.

Hand gesture recognition is increasingly used in human-computer interaction, human-robotics interaction and computer games[34]. This section demonstrates real-time hand gesture recognition as another potential application of the proposed CNN framework. The experiment demonstrates real-time recognition of 8 types of hand gesture (Figure 9) with image capturing, pre-processing and CNN inference performed on the PPA in a parallel manner.

5.2.1 Data collection and Training

We created a hand gestures dataset by capturing commonly used 8 types of hand gestures[3]. Each hand gesture class in the dataset is collected by capturing a dynamic left hand moving randomly within the view-field of the SCAMP-5. More than 1000 images are captured for each class in this way. The CNN used for classification consists of a single 4×4 kernel convolution layer using 16 filters with an input image size of 64×64, followed by a 4×4 max-pooling layer and two fully-connected layers. The choice of two fully connected layers was taken to boost accuracy, with the first performed upon the PPA array and second on the ARM controller. There are 32 intermediate neurons in the first fully-connected layer and 8 in the second. The training with the binary CNN shows the validation result has an accuracy of 98.7% .

5.2.2 SCAMP-5d Inference and Evaluation

Inference evaluation is performed by a hand randomly changing poses in front of a SCAMP-5d. Figure 10 illustrates the prediction results of the proposed neural network. The frame

[3]Dataset available at https://github.com/yananliusdu/scamp/tree/master

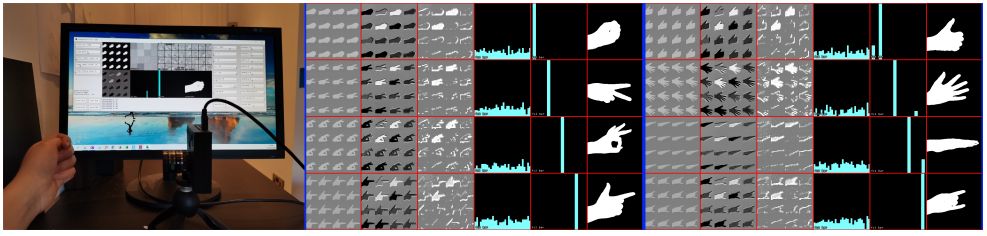

Figure 10: Examples of high-speed hand gesture classification by CNN inference on SCAMP-5d. From left to right for each column: (1) Experiment set up showing SCAMP-5d capturing hand gestures while the monitor in the background displays results from the CNN inference being performed on-board. (2) Captured images pre-processed and fed into the CNN, (3) Convolutional layer results, (4) Feature maps after activation and max-pooling, (5) Outputs of the first fully-connected layer and the height of each bar represents value for each neuron, (6) Prediction of the CNN, (7) Visualisation of predicted class.

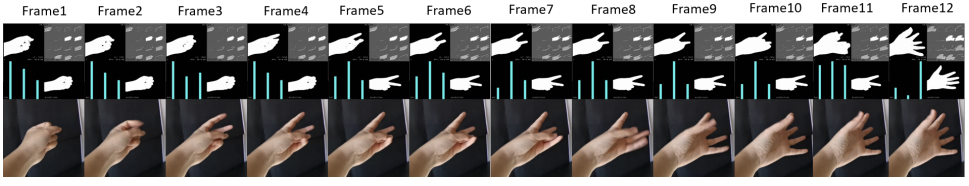

Figure 11: Rock-paper-scissors recognition inference process. The image at the bottom is the real hand gesture. Image on the top left is the input for the CNN and the prediction results can be seen at the bottom left for each 4×4 block at the top.

rate of the CNN inference for hand gesture recognition reaches 2092 fps (478 μs) (Table 2).

5.3 High-speed CNN inference on the PPA

To show the high-speed performance of the parallel embedded CNN on SCAMP-5, we implemented a rock-paper-scissors recognition and digit 0/1 recognition with stride = 2 and 4 respectively.

Rock-Paper-Scissors recognition: For this application with 3 labels, a stride = 2 (Figure 5) with a single convolutional layer and a fully-connected layer is utilised to achieve a trade-off between the efficiency and robustness. We train a binary neural network with 16 kernel filters on SCAMP-collected hand gesture dataset and get an accuracy of 97.73% (Table 2). Figure 11 shows the inference process for 12 frames sampled from a 0.3 second period which includes all the time of intermediate result transmission and displaying on the SCAMP-5 host interface for visualisation purpose. Our network can operate with latency of 121 microseconds (from image acquisition to classification result available in the micro-controller), and the frame rate of over 8,200 fps.

0/1 recognition: We trained another network to classify the digits 0 and 1 from the MNIST [23] dataset, to explore how fast CNN inference speed could be pushed for simple tasks. This network uses a single convolutional layer (of stride = 4) followed directly by a fully-connected layer. This approach requires only 12 μs for convolutional layer and fully connected layer respectively, achieving a total inference time of only 57 μs (Table 2) equivalent to 17,543 fps, and an accuracy of 99.1%.

6 Discussion

Our new implementation of convolutions allows more flexibility (different strides and different max-pooling setup) to modify a CNN for different tasks and achieves higher speeds 2,000-17,000 fps. Compared to works [4, 18, 37] which only test on MNIST, we expand to Plankton and 2 live hand gesture tasks. [4] uses ternary-weighted CNNs and achieves 94.2% at 210 fps. [18] claimed it reaches 2260 fps and quoted an accuracy of 96.9% on MNIST, but only uses 3 convolutional filters which may be insufficient to generalise to other tasks. Moreover, its frame rate drops to around 1000 fps with 7 convolutional filters indicating the nature of parallelism on the PPA is not fully exploited. [18] implemented both max-pooling and fully-connected layers in Micro-controller and the maximum inference reaches 3000 fps with a sacrificed accuracy of 90.2%.

The bottleneck that limits further performance improvement on SCAMP-5 in terms of accuracy and speed is due to the insufficient engineering resources available to academic research. If the PPA is built with state-of-the-art technology (current PPA device is manufactured with 180 *nm* CMOS silicon technology [7]), these limitations will be greatly mitigated. Finer silicon process implementation will provide more digital storage per pixel and an expanded ALU, while silicon stacking technology allows extra advantages of analogue pixel computing to still be exploited (e.g. low power, global sum, blur, etc).

7 Conclusion and Future Work

In this work we demonstrated performing CNN inference upon a PPA sensor-processor device across various tasks. Our implementation exploits the parallel computation of the entire PPA array, compared to various previous work which only utilised a small area. As a result our CNN inference is shown to be significantly faster than these works. Further our proposed convolution approach allows convolutions of stride 1,2 and 4 enabling extremely high inference speeds over 17500Hz on certain tasks to which stride 4 is applicable. The range of tasks demonstrated illustrate the potential such PPA devices may hold for future embedded applications. Though the current limitations of PPA hardware restrict us to smaller networks, it is reasonable to assume that future devices will see a significant increases in PE memory, power efficiency, and processing speed. The work presented here could quickly be adapted to take advantage of such improvement and thus can be used as a stepping stone towards more complex computational vision applications.

8 Data Access Statement and Acknowledgements

This work was supported by UK EPSRC EP/M019454/1, EP/M019284/1, EPSRC Centre for Doctoral Training in Future Autonomous and Robotic Systems: FARSCOPE and China Scholarship Council (No. 201700260083). The nature of the task and PPA means that the SCAMP-5 images in this work are not recorded.

References

[1] A Aimar, H Mostafa, E Calabrese, A Rios-Navarro, R Tapiador-Morales, IA Lungu, MB Milde, F Corradi, A Linares-Barranco, SC Liu, et al. Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps. *IEEE transactions on neural networks and learning systems*, 30(3):644–656, 2019.

[2] Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini. Yodann: An architecture for ultralow power binary-weight cnn acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(1):48–60, 2017.

[3] 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.

[4] Laurie Bose, Jianing Chen, Stephen J Carey, Piotr Dudek, and Walterio Mayol-Cuevas. A camera that cnns: Towards embedded neural networks on pixel processor arrays. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1335–1344, 2019.

[5] Laurie Bose, Jianing Chen, Stephen J Carey, Piotr Dudek, and Walterio Mayol-Cuevas. Fully embedding fast convolutional networks on pixel processor arrays. *arXiv preprint arXiv:2004.12525*, 2020.

[6] Matthew Browne, Saeed Shiry Ghidary, and Norbert Michael Mayer. Convolutional neural networks for image processing with applications in mobile robotics. In *Speech, Audio, Image and Biomedical Signal Processing using Neural Networks*, pages 327–349. Springer, 2008.

[7] Stephen J Carey, Alexey Lopich, David R W Barr, Bin Wang, and Piotr Dudek. A 100,000 fps Vision Sensor with Embedded 535GOPS / W 256x256 SIMD Processor Array C182 C183. pages 182–183, 2013.

[8] Jianing Chen, Stephen J Carey, and Piotr Dudek. Scamp5d vision system and development framework. In *Proceedings of the 12th International Conference on Distributed Smart Cameras*, page 23. ACM, 2018.

[9] Jianing Chen, Yanan Liu, Stephen J Carey, and Piotr Dudek. Proximity estimation using vision features computed on sensor. In *International Conference on Robotics and Automation (ICRA)*, pages 2689 – 2695, 31 May - 31 August 2020.

[10] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

[11] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or- 1. arxiv 2016. *arXiv preprint arXiv:1602.02830*.

[12] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[13] Thomas Debrunner, Sajad Saeedi, and Paul HJ Kelly. Auke: Automatic kernel code generation for an analogue simd focal-plane sensor-processor array. *ACM Transactions on Architecture and Code Optimization (TACO)*, 15(4):1–26, 2019.

[14] Paul Drews, Grady Williams, Brian Goldfain, Evangelos A. Theodorou, and James M. Rehg. Aggressive deep driving: Combining convolutional neural networks and model predictive control. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 133–142, 2017.

[15] Piotr Dudek. Accuracy and efficiency of grey-level image filtering on vlsi cellular processor arrays. In *Proc. CNNA*, pages 123–128, 2004.

[16] Greg Efland, Sandip Parikh, Himanshu Sanghavi, and Aamir Farooqui. High performance dsp for vision, imaging and neural networks. In *Hot Chips Symposium*, pages 1–30, 2016.

[17] Colin Greatwood, Laurie Bose, Thomas Richardson, Walterio Mayol-Cuevas, Jianing Chen, Stephen J Carey, and Piotr Dudek. Tracking control of a uav with a parallel visual processor. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4248–4254. IEEE, 2017.

[18] Benoit Guillard. Optimising convolutional neural networks for super fast inference on focal-plane sensor-processor arrays. *Master's thesis, Imperial College London-Department of Computing*, 2019.

[19] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Jincheng Yu, Junbin Wang, Song Yao, Song Han, Yu Wang, and Huazhong Yang. Angel-eye: A complete design flow for mapping cnn onto embedded fpga. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(1):35–47, 2017.

[20] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[21] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12, 2017.

[22] Yuming Kuang. Deep neural network for deep sea plankton classification. Technical report, Technical Report, 2015.

[23] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2:18, 2010.

[24] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[25] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural networks with few multiplications. *arXiv preprint arXiv:1510.03009*, 2015.

[26] Alejandro Linares-Barranco, Antonio Rios-Navarro, Ricardo Tapiador-Morales, and Tobi Delbruck. Dynamic vision sensor integration on fpga-based cnn accelerators for high-speed visual classification. *arXiv preprint arXiv:1905.07419*, 2019.

[27] Qingzhong Liu, Zhaoxian Zhou, Sarbagya Ratna Shakya, Prathyusha Uduthalapally, Mengyu Qiao, and Andrew H Sung. Smartphone sensor-based activity recognition by using machine learning and deep learning algorithms. *International Journal of Machine Learning and Computing*, 8(2):121–126, 2018.

[28] Iulia-Alexandra Lungu, Federico Corradi, and Tobi Delbrück. Live demonstration: Convolutional neural network driven by dynamic vision sensor playing roshambo. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–1. IEEE, 2017.

[29] Iulia Alexandra Lungu, Shih-Chii Liu, and Tobi Delbruck. Fast event-driven incremental learning of hand symbols. In *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 25–28. IEEE, 2019.

[30] Julien NP 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, 2017.

[31] Manu Mathew, Kumar Desappan, Pramod Kumar Swami, and Soyeb Nagori. Sparse, quantized, full frame cnn for low power embedded devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 11–19, 2017.

[32] Eric C Orenstein, Oscar Beijbom, Emily E Peacock, and Heidi M Sosik. Whoi-plankton-a large scale fine grained visual recognition benchmark dataset for plankton classification. *arXiv preprint arXiv:1510.00745*, 2015.

[33] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[34] Siddharth S Rautaray and Anupam Agrawal. Vision based hand gesture recognition for human computer interaction: a survey. *Artificial intelligence review*, 43(1):1–54, 2015.

[35] Jaehyeong Sim, Jun-Seok Park, Minhye Kim, Dongmyung Bae, Yeongjae Choi, and Lee-Sup Kim. A 1.42 tops/w deep convolutional neural network recognition processor for intelligent ioe systems. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 264–265. IEEE, 2016.

[36] Baohua Sun, CA Milpitas, Daniel Liu, Leo Yu, Jay Li, Helen Liu, Wenhan Zhang, and Terry Torng. System demonstration of mram co-designed processing-in-memory cnn accelerator for mobile and iot applications.

[37] Matthew Wong. Analog vision-neural network inference acceleration using analog simd computation in the focal plane. *Master's thesis, Imperial College London-Department of Computing*, 2018.

[38] Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian. Variational convolutional neural network pruning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[39] Ruizhe Zhao, Xinyu Niu, Yajie Wu, Wayne Luk, and Qiang Liu. Optimizing cnn-based object detection algorithms on embedded fpga platforms. In Stephan Wong, Antonio Carlos Beck, Koen Bertels, and Luigi Carro, editors, *Applied Reconfigurable Computing*, pages 255–267, Cham, 2017. Springer International Publishing. ISBN 978-3-319-56258-2.

[40] Chaoyang Zhu, Kejie Huang, Shuyuan Yang, Ziqi Zhu, Hejia Zhang, and Haibin Shen. An efficient hardware accelerator for structured sparse convolutional neural networks on fpgas. *arXiv preprint arXiv:2001.01955*, 2020.

[41] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

[42] Yuhao Zhu, Matthew Mattina, and Paul Whatmough. Mobile machine learning hardware at arm: a systems-on-chip (soc) perspective. *arXiv preprint arXiv:1801.06274*, 2018.