NeuroHSMD: Neuromorphic Hybrid Spiking Motion Detector

PEDRO MACHADO, JOÃO FILIPE FERREIRA ANDREAS OIKONOMOU, Department of Computer Science, School of Science and Technology, Nottingham Trent University, UK

T.M. MCGINNITY, Intelligent Systems Research Centre, School of Computing, Engineering and Intelligent Systems, Ulster University, UK

Vertebrate retinas are highly-efficient in processing trivial visual tasks such as detecting moving objects, which still represent complex challenges for modern computers. In vertebrates, the detection of object motion is performed by specialised retinal cells named Object Motion Sensitive Ganglion Cells (OMS-GC). OMS-GC process continuous visual signals and generate spike patterns that are post-processed by the Visual Cortex. Our previous Hybrid Sensitive Motion Detector (HSMD) algorithm was the first hybrid algorithm to enhance Background subtraction (BS) algorithms with a customised 3-layer Spiking Neural Network (SNN) that generates OMS-GC spiking-like responses. In this work, we present a Neuromorphic Hybrid Sensitive Motion Detector (NeuroHSMD) algorithm that accelerates our HSMD algorithm using Field-Programmable Gate Arrays (FPGAs). The NeuroHSMD was compared against the HSMD algorithm, using the same 2012 Change Detection (CDnet2012) and 2014 Change Detection (CDnet2014) benchmark datasets. When tested against the CDnet2012 and CDnet2014 datasets, NeuroHSMD performs object motion detection at 720×480 at 28.06 Frames Per Second (fps) and 720×480 at 28.71 fps, respectively, with no degradation of quality. Moreover, the NeuroHSMD proposed in this paper was completely implemented in Open Computer Language (OpenCL) and therefore is easily replicated in other devices such as Graphical Processing Units (GPUs) and clusters of Central Processing Units (CPUs).

CCS Concepts: • **Computing methodologies → Massively parallel algorithms**; **Vision for robotics**; • **Hardware → Neural systems**; • **Computer systems organization → Heterogeneous (hybrid) systems**.

Additional Key Words and Phrases: SNN, HMSD, NeuroHSMD, retinal cells, OMS-GC, FPGA, background subtraction, object motion detection

ACM Reference Format:
Pedro Machado, João Filipe Ferreira Andreas Oikonomou and T.M. McGinnity. 2022. NeuroHSMD: Neuromorphic Hybrid Spiking Motion Detector. 1, 1 (March 2022), 25 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The human brain is characterised by its tolerance to faults/noise, concurrent processing capabilities, flexibility and high level of parallelisation when processing data. Furthermore, the adult human brain has a power consumption of about 400 Kcal per day, equivalent to 25 Watts [1]. Again, the human brain can reach 10-50 petaflops outperforming any Commercial-Off-The-Shelf (COTS) Central Processing Unit (CPU) [2]. Despite CPUs outperforming the human

Authors' addresses: Pedro Machado, João Filipe Ferreira Andreas Oikonomou, Department of Computer Science, School of Science and Technology, Nottingham Trent University, Clifton Lane, Nottingham, UK, NG11 8NS, \{pedro.machado,andreas.oikonomou\}@ntu.ac.uk; T.M. McGinnity, Intelligent Systems Research Centre, School of Computing, Engineering and Intelligent Systems, Ulster University, Northlands Road, Magee Campus, Londonderry, UK, BT48 7JL, tm.mcginnity@ulster.ac.uk.

brain when processing and transmitting sequential signals by several orders of magnitude, the human brain exceeds CPUs processing millions of signals in parallel using its massively parallel circuits [1, 2]. SNNs are well known for their biological plausibility but also by their complexity inherited from biological systems, which are characterised for being massively parallel [3]. Modern computation platforms rely heavily on CPUs to provide compatibility and security with other devices/applications. Although the design CPU paradigm has shifted into multi-core and multi-processor to overcome the limitations associated with the clock speed [4], the multi-core and multi-processor strategy will shortly meet technological limitations related to the increase of power consumption of these solutions [4].

Machado et al. proposed the Hybrid Sensitive Motion Detector (HSMD) algorithm [5] that has proven to be very sensitive to object motion events as a direct consequence of using an Spiking Neural Network (SNN) to emulate the basic functionality observed in Object Motion Sensitive Ganglion Cells (OMS-GC) (see Figure 3). The SNN utilised is composed of 3 layers of neurons interconnected on a 1:1 synaptic connectivity. CPUs which are suitable for sequential operations are not optimised for running massively parallel SNNs architectures. Unlike CPUs that are not optimised for parallel tasks, and Graphical Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs) are parallel processing devices for accelerating paralellisable algorithms. GPUs are specialised electronic circuits with a flexible architecture designed for parallel processing of graphics, video rendering, and accelerating some types of Artificial Intelligence (AI) algorithms [6–9], while FPGAs are Integrated Circuits (ICs) composed of built-in interconnected hardware blocks that can be freely reprogrammable after manufacturing [10]. Despite the fact that both GPUs and FPGAs are suitable for parallel applications, GPUs have a well-defined architecture, whereas FPGAs are flexible devices that allow users to describe new hardware architectures, such as brain-like and neuromorphic architectures [11]. Nevertheless, GPUs have also proven to be suitable to implement SNN architectures [12]. In this work, a FPGA device was used to accelerate a customised 3-layer SNN.

The Neuromorphic Hybrid Sensitive Motion Detector (NeuroHSMD) is the reconfigurable hardware implementation of the HSMD algorithm [5]. A high-end Intel Stratix 10 FPGA(see section 3 for further details) was selected for accelerating the HSMD's SNN. Open Computer Language (OpenCL) was used for describing the SNN because it provides a higher level of abstraction when compared with Hardware Description Languages (HDLs), increased productivity and compatibility with other OpenCL platforms (such as other FPGA devices, GPUs and CPUs).

The paper is structured as follows: the background research is presented in Section 2, the hardware platform details are discussed in Section 3; the NeuroHSMD architecture is presented in Section 4; the results are presented in Section 5 and the discussion of the NeuroHSMD results and future work are in Section 6.

2 BACKGROUND RESEARCH

The detection of moving objects from video frame sequences is a trivial visual task performed by vertebrate retinal Ganglion Cells (GC) [13, 14] and yet a challenge in the Computer Vision (CV) research field. Object Motion Detection (OMD) is one of the most researched fields in computer vision and has been studied for more than 30 years [15]. OMD in videos captured from static and/or moving cameras is essential for a wide range of computer vision applications such as video surveillance, object collision avoidance, Advanced-Driver Assist Systems (ADAS), etc [15–17]. Although the initial OMD models were designed for static cameras, the advances in sensor technology and the accessibility to portable devices fitted with cameras is triggering more challenging scenes where both cameras and objects can move at the same time [15]. OMD includes the following tasks: 1) Background subtraction, 2) noise reduction, 3) threshold selection and 4) moving objects detection (see Figure 1).

Several challenges have been identified in various works [15, 17–21] and can be summarised as follows : 1) **Bootstrapping**: the sequence of images includes objects in both the background and foreground; 2) **Camouflage**: the objects in the foreground are either obstructed by background objects or are composed of similar colours; 3) **Dynamic background**: the objects in the background include parasitic movements such as surface water movement, branches and leafs shaking in trees, flags on windy days, etc; 4) **Camera aperture**: undesired blurred background and foreground as a consequence of the incorrect opening in a lens through which light passes to enter the camera; 5) **Variation of illumination**: instant variations of illumination will increase the number of false-positive detections (i.e. pixels that should belong to the background are classified as foreground); 6) **Low frame rate**: the temporal distance between image frames prevents instant updates of the background and illumination changes, which reduces the accuracy and increases the number of false positives; 7) **Motion blur**: caused by rapid camera movements or jittering, which blurs the image; 8) **Parallax**: the apparent displacement of an object as a consequence of the camera movement. The Parallax will have implications on the background modelling and its compensation; 9) **Moving camera**: moving cameras introduce complexity because the static objects seem to be moving, and objects moving at a similar speed in the same direction of the camera will seem to be static; 10) **Background objects movement**: although static objects can be added to and removed from the background, such objects should still be considered static; 11) **Night videos**: night videos have dim light, lower contrast and reduced colour information; 12) **Noisy images**: low-quality sensors, dust exposure, dirty lens, bright lights and low resolution are examples of factors that cause noisy images; 12) **Shadows**: shadows created by objects when exposed to light sources (e.g. sun rays and artificial illumination) should not be part of the foreground models; 13) **Stationary foreground objects**: a foreground that has stopped moving for a short period should not become part of the background model; 14) **Challenging weather**: weather conditions (such as fog, rainstorms, strong winds, intense sun rays) have a major impact on the image quality and reduce the quality of the image drastically.

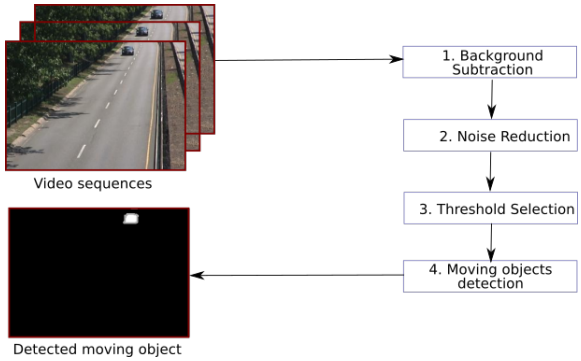

Fig. 1. Object Motion Detection steps.

2.1 Background Subtraction

Several surveys about BS have been published in the literature focused on static or semi-static (i.e. cameras fixed in a given position exhibiting pan-tilt-zoom movements) scenes. McIvor [22] published, in 2000, one of the first OMD surveys where nine BS methods which were only described in detail but not compared. Piccardi [23] presented, in 2004, a comparative study between seven algorithms considering speed, memory resources utilisation and accuracy (see Table 1). Piccardi's study [23] aims to facilitate the BS selection based on speed, memory requirements and accuracy requirements.

Cheung et al. [33] proposed a method for validating foreground regions (blobs) using a slow-adapting Kalman filter and compared the proposed method against six other methods using the recall and precision metrics. Elhabian et al. [34] covered several background removal algorithms and identified that all the BS algorithms follow four significant steps, namely, pre-processing, background modelling, foreground extraction, and validation. Although the review was

Table 1 BS methods and their performance analysis

Method	Speed	Memory	Accuracy
Running Gaussian average [24, 25]	high	low	acceptable
Temporal median filter [26, 27]	high	low	acceptable
Mixture of Gaussians [28]	low	high	very good
Kernel density estimation [29]	low	high	very good
Sequential kernel density approximation [30]	low	acceptable	good
Cooccurence of image variations [31]	acceptable	acceptable	good
Eigenbackgrounds [32]	acceptable	acceptable	good

very comprehensive, the focus was on recursive and non-recursive approaches, which are suitable for background maintenance but less suitable for background modelling. Cristiani et al. [35] reviewed BS methods that can be applied to data captured from different sensor channels (including audio). Elgammal [36] reviewed more than 100 papers about object motion detection for static and moving cameras, highlighting the challenges and suggesting which method to use in each case. Bouwmans et al., Garcia et al. and Chapel et al. published comprehensive surveys [15, 17–21] focusing on traditional, recent, and prospective object motion detection methods.

Consecutive frame difference, background modelling and optical flow are the main categories for BS. Consecutive frame difference methods are the simplest to implement and require less computational resources, but are also the most sensitive to the challenges listed above [15, 17]. In contrast, optical flow methods are the most robust but require more computational resources and, consequently, are not suitable for real-time applications [15, 17]. Therefore, background modelling methods are commonly used methods for extracting the foreground from the background in real-time applications [15, 17]. BS generic steps are detailed in Figure 2.

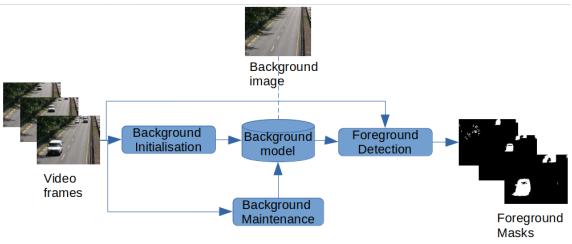

Fig. 2. Background subtraction steps.

Stauffer & Grimson [28], and KaewTraKulPong & Bowden [37] suggested modelling each pixel as a Mixture of Gaussians (MOG) where the Gaussian distributions of the adaptive mixture model are analysed for determining which ones are likely to belong to the background process. All the pixel values that do not fit in the background distributions are considered foreground [28]. Zivkovic [38] proposes an efficient adaptive algorithm using the Gaussian Mixture Probability Density (MOG2) for enhancing the MOG algorithm. MOG2 selects automatically the number of components per pixel, which results in complete adaptation to the observed scene. Zivkovic & Heijden [39] identified recursive equations for updating the parameters of the MOG and proposed Mixture of Gaussians K Nearest Neighbours (KNN) for the automatic selection of the pixel components. The Gaussian mixture based algorithms (MOG, MOG2 and KNN) show good robustness when exposed to noise and losses due to image compression but lack sensitivity to intermittent object motion, moving background objects and abrupt illumination changes.

In 2016, Sagi Zeevi [40] proposed the CNT algorithm, which performed better on the 2014 Change Detection (CDnet2014) dataset [41] and targets embedded platforms (e.g. Raspberry PI[1]). The CNT uses minimum pixel stability for a specified period for modelling the background; this can vary from 170 ms, default for swift movements, up to hundreds of seconds, where 60s is the default value [42]. Godbehere *et al.* [43] suggested a single-camera statistical segmentation and tracking algorithm named the GMG by combining per-pixel Bayesian segmentation, a bank of Kalman filters, and Gale-Shapley matching for the approximation of the solution to the multi-target problem. The proposed GMG algorithm is limited when processing video streams susceptible to camouflage, losses due to image compression, and noise.

Guo *et al.* [44] reported an adaptive Background subtraction model enhanced by a local Local Single Value Decomposition Binary Pattern (LSBP) for addressing illumination changes. The proposed LSBP algorithm enhances the robustness of the motion detection to illumination changes, shadows, and noise. However, the LSBP is less effective when processing video streams susceptible to camouflage, losses due to image compression, or external noise. More recently, in 2017, Open Computer Vision (OpenCV) released an improved version of the LSBP algorithm, also known as Google Summer of Code (GSOC) [45, 46], developed during the Google Summer of Code event [47], which enhances the LSBP algorithm by using colour descriptors and stabilisation heuristics for motion compensation [46, 48]. The GSOC algorithm demonstrates better performance on the 2012 Change Detection (CDnet2012) [49], and CDnet2014 [41] datasets [48, 50] when compared to other algorithms available on the OpenCV library.

More recently, Braham et al. [51] proposed a Semantic Background Segmentation (SBS) that uses object-level semantics to meet a range of problematic background subtraction conditions. The proposed SBS reduces false positive detections by integrating the output information of a semantic segmentation method, expressed as a probability for each pixel, with the output of existing BS methods. Inspired by Braham's work [51], Zeng et al. [52] proposed a Real-Time Semantic Segmentation (RTSS) for performing BS. The RTSS consists of two components: a BS segmenter B and a semantic segmenter S that work in parallel for foreground segmentation. The RTSS achieves state-of-the-art performance among most unsupervised background subtraction methods while functioning in real-time as compared to other BS methods [52]. Liang et al. [53] proposed a deep background subtraction method using a directed learning strategy that learns a specific Convolutional Neural Network (CNN) model for each video without manually labelling. Zeng et al. [54] proposed a Multiscale Fully Convolutional Network (MFCN) architecture for background subtraction that takes advantage of diverse layer features. The deep features learned from MFCN improves foreground detection, and the complexity of the background subtraction process can be easily handled during the subtraction operation itself.

BS can be done using methods based on signal processing, Machine Learning (ML), Deep Neural Network (DNN) or mathematical models. Although signal processing, ML and DNN tend to exhibit better accuracy than mathematical models, these algorithm types are also computationally intensive, introducing undesirable latencies. Nevertheless, mathematical models exhibit lower accuracy but require fewer computational resources and, therefore, are suitable for real-time applications. Moreover, the methods proposed by Braham et al. [51] and Zeng et al. [55] demonstrated that existing BS algorithms can be improved when combined with semantic segmentation models. Therefore, in this work, we propose an approach to accelerate the HSMD's SNN which was identified to be the bottleneck of the HSMD algorithm [5].

[1] Available online, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/, last accessed 28/03/2022

2.2 Spiking Neural Networks

FPGAs have been used, for many decades, accelerating applications, including edge/cloud computing. FPGAs are flexible devices because of their flexible architecture enables developers to describe customised architectures. Such flexibility comes with a downside because FPGAs are also known by their associated complexity. There are two main FPGA manufacturers, namely, Intel[2] and Xilinx[3].

Mishra et al. [56] identified in their survey that many SNN usually have about $10^4 \sim 10^8$ neurons and $10^{10} \sim 10^{14}$ synapses and that high-performance neural hardware is essential for practical application. Li et al. [57] proposed the implementation of visual cortex neurons on FPGAs. The implemented visual cortex neurons exhibited the same dynamics as those recorded from real neurons using multi-electrodes arrays. Li et al.[58] implemented 256 fully connected neurons, and its performance was assessed by storing four patterns and applying similar patterns containing errors. The implemented system was capable of operating using a 100 MHz clock, which enables the acceleration of the system 40 times above the real-time operation [58]. Cassidy et al. [59] proposed the use of FPGAs to accommodate spiking neurons and unsupervised Spike Timing Dependent Plasticity (STDP) learning structures. In this work, Cassidy et al. [59] demonstrated that digital neuron abstraction is preferable to more realistic analogue neurons; they also emulated the massive parallelism connectivity and high neuron density as observed in nature; the neuron states were also multiplexed to take advantage of clock frequencies and dense Static Random Access Memorys (SRAMs). Chen et al. [60] described a Central Pattern Generator (CPG) composed of two reciprocally inhibitory neurons. To reduce the FPGA resources usages, Chen et al. [60] has optimised the CPG to avoid using multipliers (FPGAs have a low quantity of multiplier blocks), and the non-linear parts of the Komendantov-Kononenko neuron model [61] were removed. Cheung et al. [62] proposed the NeuroFlow, a scalable SNN simulator suitable to be implemented on FPGA clusters. It was possible to simulate about 600,000 neurons and to get a real-time performance for up to 400,000 neurons simulated using NeuroFlow on 6 FPGAs [62]. Podobas and Matsuoka [63] proposed the use of OpenCL, an High Level Synthesis (HLS) tool, to increase productivity by facilitating the SNN design (provide a higher level of hardware abstraction) on FPGAs. Two different neuron models, their axons and synapses, were designed using OpenCL and the authors claim a speed performance of up to 2.25 GSpikes/second. Sakellariou et al. [64] suggested a spiking accelerator base on FPGAs to enable users to develop SNNs targeting ML applications and promise an acceleration of up to 800 times for inference and up to 500 times for training compared to Software SNN simulations.

Machado et al. [5] proposed the HSMD model (see Figure 3) inspired by the object motion functionality exhibited by vertebrate retinas, in which OMS-GC determine the difference between a local patch's motion trajectory and the background [13]. The HSMD uses a 3-layer SNN to enhance the GSOC BS algorithm [5, 46, 48].

The works reviewed in this section demonstrated that FPGAs offer flexibility, high efficiency, low-power, and high degree of parallelism, making FPGAs is the most suitable devices for implementing brain-like circuits. FPGAs enable the design of complex biological plausible neuron models and massively parallel SNN composed of thousands of Leaky-Integrate-and-Fire (LIF) neurons capable of generating complex biological like patterns. Although FPGA devices are normally programmed using complex HDL tools, HLS tools such as OpenCL can be used to increase the productivity of the FPGAs design process by providing hardware abstraction which reduces the implementation complexity. In this paper, we utilise OpenCL to design the complex SNN architecture of the HSMD. The NeuroHSMD reported in this paper, improves the speed of the HSMD [5] without degradation of the background subtraction accuracy using an

[2]Available online, https://www.intel.co.uk/content/www/uk/en/products/programmable/fpga.html, last accessed: 04/03/2021
[3]Available online, https://www.xilinx.com/, last accessed: 04/03/2021

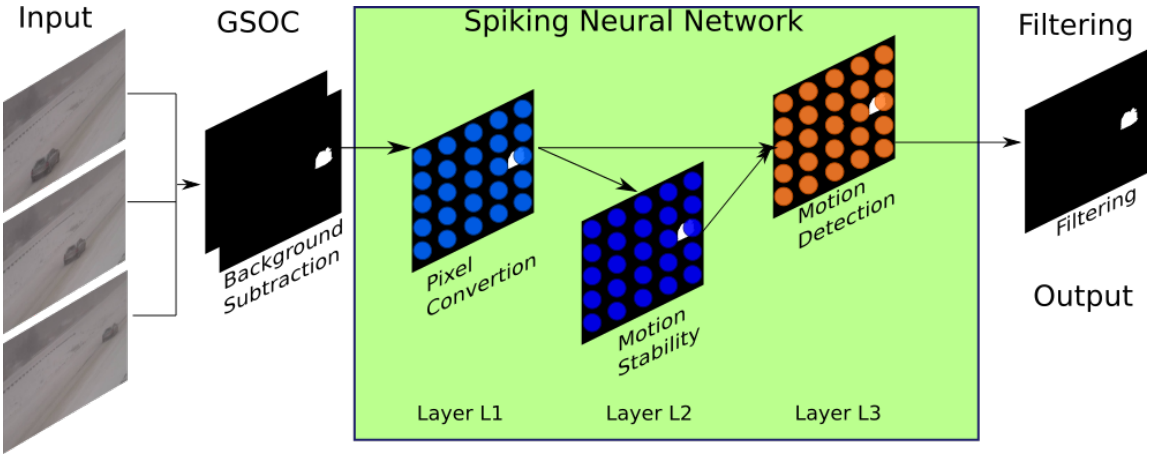

Fig. 3. HSMD architecture implemented by Machado et al. [5]. The GSOC algorithm performs the background subtraction, the customised SNN performs the OMD and the filter removes noise. The customised SNN is composed of 3 layers for 1) converting pixel values conversion into currents, 2) motion stability and 3) perform motion detection.

high-end FPGA device.

3 HARDWARE PLATFORM

The HSMD algorithm [5] has proven to be very sensitive to object motion events triggered by objects, as a direct consequence of using an SNN to emulate the basic functionality observed in OMS-GC. The SNN utilised is composed of 4 layers of neurons interconnected on a 1:1 synaptic connectivity.

3.1 Field Programmable Gate Array

The target FPGA board is fitted with a state-of-the-art Stratix 10 SoC FPGA device[4]. The Intel Stratix family is composed of logic array blockss (LABs) made of 10 basic building blocks called adaptive logic modules (ALMs). Each ALM is composed of fractionable Look-Up-Tables, also known as adaptive look-up-table (ALU), two-bit full adder and four registers. LABs can be freely reconfigured to implement logic and arithmetic functions. Furthermore, up to a quarter of the LABs can be used as memory logic array blockss (MLABs). Each LAB contains dedicated logic elements used to driving control signals to ALMs. Each MLAB supports up to 640 bits of simple dual-port Random Access Memory (RAM). It is possible to configure each ALM in an MLAB as 32×2 memory blocks equivalent to $32 \times 2 \times 10$ simple dual-port RAM blocks. Dual-port RAMs are low-latency memory devices that only takes a clock cycle to perform a read/write operation (for example, Synchronous Dynamic Random Access Memory (SDRAM) in CPUs takes thousands of clock cycles to complete read/write operations). Furthermore, the Stratix 10 offer variable-precision digital signal processing (DSP) blocks that can support fixed-point arithmetic and single-precision floating-point arithmetic.

[4] Available online, https://www.intel.co.uk/content/www/uk/en/products/programmable/soc/stratix-10.html, last accessed: 07/04/2021

3.2 Open Computer Language (OpenCL)

OpenCL applications are split into two parts, namely, **host** application(s) and **device** kernel(s). The **host** applications are always compiled on the host Operating System and run on a CPU. **Host** applications are also used to launch the target kernels on the target **devices**. Kernels are special functions written in OpenCL C/C++ to perform parallelisable computations on accelerators such as GPUs and FPGAs [65]. For instance, consider two m×n matrices A and B where it is expected to do the operation C=A+B where C is the third matrix of m×n. In this case, the kernel could just perform, in parallel, the addition of matrices A and B elements and store the result in C. Unlike in CPUs, where it would take m×n operations to complete this matrix addition, GPUs and FPGAs could parallelise this operation depending on the resources available per device resulting in the acceleration of the application. Buffer objects within a context are used in OpenCL to exchange data between the host and device [66]. The Intel FPGA Software Development Kit (SDK) for OpenCL offline compiler optimises the kernel throughput by adjusting buffer sizes during the kernel compilation process [67]. OpenCL provides both mapped and asynchronous buffers, enabling the application to continue to run while additional data is exchanged. Software Developers have to carefully analyse the code to be optimised and only select the sections that may benefit from the hardware acceleration because the maximum speed is always dictated by the PCIe bus speed. Another big challenge for Software Developers is the low debugging capabilities available while the code is being executed on the device.

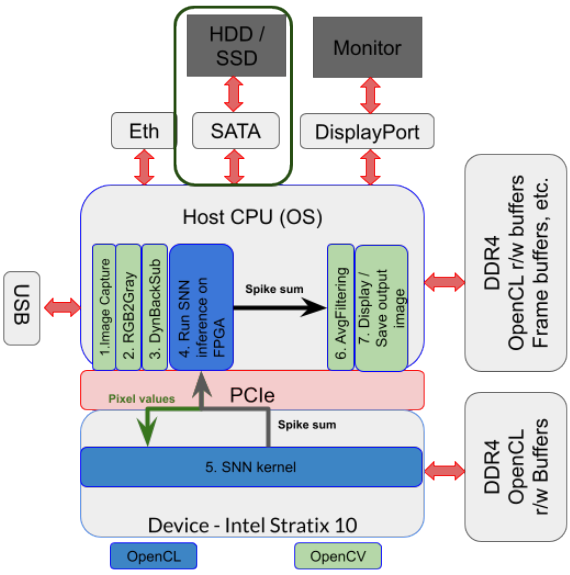

Fig. 4. NeuroHSMD architecture. The diagram represents the computation stages that run both on the CPU and FPGA. Shows that the FPGA is connected to the host CPU via the Peripheral Component Interconnect Expres (PCIe) bus. It also shows the dedicated memory of the CPU and FPGA device. Includes also how external devices (e.g. cameras, monitor and Hard Disk Drive (HDD)/Solid State Drive (SSD)) connect to the host CPU via different interfaces (e.g. ethernet (eth), Serial Advanced Technology Attachment (SATA), display port and Universal Serial Bus (USB)). The computations stages implemented in OpenCL are in blue and OpenCV in green.

Although it is possible to use OpenCL to program FPGA and GPU devices, GPUs are specialised devices designed for video rendering and graphics processing. At the same time, FPGAs are customisable devices that can be freely reconfigurable. Therefore, FPGAs offer more flexibility than GPUs, which is desirable for accelerating SNNs because they can be modelled using the Network-on-Chips (NoCs) concept. Each individual spiking neuron can be considered a node that interconnects to one or more nodes (neurons) of the same SNN. The flexibility offered by both FPGAs and OpenCL makes the selection of FPGAs over GPUs the obvious choice.

The Intel FPGA SDK for OpenCL (IOCL) provides a compiler and powerful tools to build and run OpenCL applications targeting Intel FPGA devices. The IOCL generates two main components: the host application and the FPGA programming bitstream(s). The IOCL offline compiler (AOC) first compiles the custom kernel(s) to an image file (*.aocx) that will be used to program the FPGA. In contrast, the host-side C/C++ compiler compiles the host application and then links it to the IOCL runtime libraries.

Fixed-point data representations, which only retain the necessary data resolution for calculations and can result in hardware savings, are a popular choice among hardware developers. However, the IOCL standard lacks support for fixed-point representation and floating operations must be carried out using IEEE754 single-precision floating-point [68]. In this work we adhered to this approach (the OpenCL standard) which then enabled the hardware compilation tools to perform optimisations using appropriate masking operations in the source code.

The IOCL compiles one or more OpenCL kernels and creates a hardware configuration file. A successful compilation results in a ***.aocr**, ***.aoco**, ***.aocx** and reports/report.html files. The report.html contains the estimated resource usage and a preliminary assessment of area usage. The intermediary ***.aoco** and ***.aocr** are only used in the generation of the ***.aocx** which is then used to program the FPGA.

The Terasic DE10-pro development board [69] equipped with a state-of-the-art high-end Intel Stratix 10 FPGA device was used for implementing the NeuroHSMD discussed in this section. Terasic states that DE10 pro was designed to fulfil the demands of AI, Data Center, and High-Performance Computing. Furthermore, the DE10-pro development board takes advantage of the latest Intel Stratix 10 to obtain high-speed and low-power (with up to 70% lower power when compared with the previous generation - i.e. Stratix V). It is equipped with 32GB DDR4 memory module running at over 150 Gbps, up to 15.754 GB/s data transfer via PCIe Gen 3 x16 edge between FPGA and host workstation, and 4 onboard QSFP28 (100GbE) connectors. The DE10 pro was installed on the host workstation equipped with an Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz and 16 GB of DDR3 using the PCIe slot.

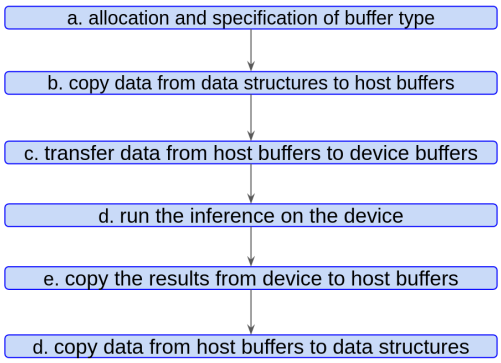

Fig. 5. OpenCL stages.

4 NEUROHSMD ARCHITECTURE

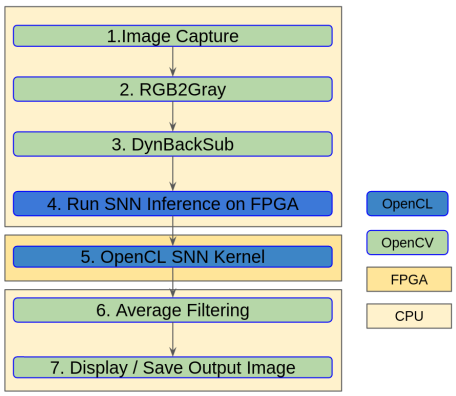

Fig. 6. NeuroHSMD computation stages.

The NeuroHSMD architecture is described in this section. In OpenCL, the data exchanged between the host application and the FPGA kernels flow is as follows: a) allocation and specification of buffer type (i.e. read or write) on the host and device; b) copy data from the application data structures to host buffers; c) transfer data from host buffers to device buffers; d) run the inference on the device; e) copy the results from device to host buffers; f) copy data from the host buffers to application data structures (see Figure 5).

The NeuroHSMD algorithm performs the following stages of computation: 1) image capture, 2) conversion from colour to grey, and 3) dynamic background subtraction using the OpenCL's GSOC algorithm and copy

resulting pixel values are buffered and transferred to the FPGA device; 4) run the inference on the FPGA and wait for the spike results; 5) run the SNN kernel, 6) Filtering using an average filter, and 7) Display and save the output image. The NeuroHSMD computation stages are summarised in Figure 6 and the NeuroHSMD architecture is depicted in Figure 4.

Furthermore, the NeuronHSMD OpenCL is composed of the NeuronHSMD Host Application (NHA) and the NeuronHSMD device kernels (NDK). Details about the NHA and NDK are given in sections 4.4 and 4.5.

4.1 Spiking Neuron Model

This study employed the LIF spiking neuron model due to its simplicity, computational efficiency, and applicability for near-real-time picture processing. When compared to real biological neurons, the LIF spiking neuron model has comparable but less complex dynamics [70]. There are additional sophisticated spiking neuron models, such as Hodgkin-Huxley, however these need substantial computational resources and have a larger effect on computational performance (e.g. Izhikevich [71]). The dynamics of the LIF neuron follow equation 1 [70].

$$I(t) = \frac{Vm(t) - Vm_{rest}}{R} + C\frac{\partial Vm}{\partial t} \tag{1}$$

$$\tau_m \frac{\partial Vm}{\partial t} = -[Vm(t) - Vm_{rest}] + RI(t) \tag{2}$$

Where C is the membrane capacitance, R is the membrane resistance, $I(t)$ is the current in a given time t, $Vm(t)$ is the membrane potential in a given time t, τ_m is constant given by the resister R times the capacitor C and the Vm_{rest} is the reset potential.

The action potential $t(f)$ that is emitted by a given neuron is known as the firing time. The firing time $t^{(f)}$ is given by equation 3 [70].

$$\lim_{\delta \to 0; \delta > 0} t^{(f)} : Vm(t^{(f)}) = Th \tag{3}$$

The firing time $t^{(f)}$ is generated when the potential is reset to a new value $Vm_r < Th$ where Th is the threshold (see equation 4 [70]).

$$\lim_{\delta \to 0; \delta > 0} Vm(t^{(f)} + \delta) = Vm_r \tag{4}$$

For $t > t^{(f)}$ the dynamics given by equations 1 and 2 until the next threshold Th crossing occurs. (δ is the Dirac function). The combination of leaky integration in equations 1 and 2 and reset 3 is given by equation 5 [70].

$$S_i(t) = \sum_f \delta(t - t_i^{(f)}) \tag{5}$$

The Euler method was employed to numerically solve the Ordinary Differential Equations (ODEs) that model the behaviour of the LIF neurons. This method was used to approximate the solutions of the ODEs at the simulation's time-steps throughout the simulation. At each time-step, the Euler method used the derivative of the current solution to estimate the next solution, effectively advancing the state of the neuron across time.

4.2 Layers and interconnectivity

4.2.1 *Input Layer: BS and reduction.*

Each $n \times m$ image frame (i.e. camera, video sequence or image sequences) is transformed into greyscale. The GSOC [45] provides an adaptive BS utilising colour descriptors and different stabilisation heuristics [46, 48] while computing the frames pixel-by-pixel and exploiting the scalability offered by the OpenCV [48].

4.2.2 *Layer 2: Pixel intensities to currents encoding.*

Pixel intensity values are transformed to proportionate currents and delivered to the spiking neurons in Layer 2 through a 1:1 connection. The Layer 1 neurons were trained to generate spike events proportional to the pixel intensities, and are ruled by the equation 6.

$$i_c(x, y) = I(x, y).c \tag{6}$$

where $i_c(x, y)$ is the corresponding current for the image light intensity value $I(x, y)$ at coordinates x and y, and c is a conversion constant obtained experimentally (in our case, c=17.5).

4.2.3 *Layer 3: Motion stability.*

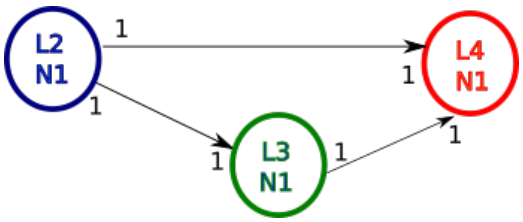

Fig. 7. HSMD connectivity. In this example, it can be seen that the neuron 1 (N1) of each layer connects to the N1 of the subsequent layer.

The neurons in Layer 3 are utilised to stabilise motion by creating local buffers and delaying the transmission of spike occurrences. Neurons in layer 2 connect to Layer 3 neurons and Layer 3 neurons to the Layer 4 neurons for creating local delays. The Layer 3 neurons are used to buffer spike events for one simulation time-step (δt, in this work, $\delta t = 10ms$) and forwarded to the Layer 4 neurons for the subsequent simulation time-step. The Layer 2 neurons extend 1:1 synaptic connectivities to the Layer 3 neurons and Layer 3 neurons also extend 1:1 synaptic connectivities to the Layer 4 neurons as depicted in Figure 7. All the synapses were assigned the value 1370, which was obtained experimentally.

4.2.4 *Layer 4: Motion detection.*

The neurons in Layer 4 receive excitatory synaptic connections from the neurons in both Layers 2 and 3 and utilise the spike patterns generated to sense motion. The generation of spike events by Layer 4 neurons is the result of dynamic

changes triggered by successive image frames. Signals received directly from Layer 2 neurons allow for the detection of differences between the current image frame n and the prior image frame n-1. In contrast, Layer 3 neurons are used for comparing image frame n-1 to image frame n-2. Layer 4 spike events are mapped to the matching area of the current image area. The synaptic weight of 1370 for all the synapses was obtained experimentally. The Layer 2 to Layer 4 weights were adjusted to feed-forward all the Layer 2 spike events. While the Layer 3 to Layer 4 synaptic weights were adjusted to generate spike events from the Layer 4 neurons for every pair of two consecutive spike events, the primary objective was to give higher priority to recent spike events (frame [n] - frame [n-1]) and lower priority to older spike events (frame [n-1] - frame [n-2]).

4.2.5 *Layer 5: Filtering.*

The spike events matrix of Layer 4 neurons is mapped into a motion matrix M_d of the same dimensions as the current image frame (i.e., $ntimesm$). The events in the M_d matrix are filtered via an averaging filter ruled by the following equations 7 and 8:

$$H(u,v) = \frac{1}{u.v}\left(\begin{bmatrix} w_{0,0} & \dots & w_{0,u} \\ \dots & \dots & \dots \\ w_{v,0} & \dots & w_{v,u} \end{bmatrix}\right) \tag{7}$$

$$Y_d(x,y) = M_d(x,y) * H(u,v) \tag{8}$$

where $Y_d(x,y)$ is the filtered motion detection matrix, $H(u,v)$ is the averaging filter, u and v are the convolution window length and height respectively, $*$ is the convolution operator, w is the filter window.

4.3 Neuronal parameters

The SNN was parameterised using the recommended settings in the references [72, 73]. Therefore, the simulation was configured with a time step of δt=10 ms, V_m=-55.0 mV, E_L = -55.0 mV, C_m = 10.0 pF, R=1.0 MΩ, V_{reset}=-70.0 mV, V_{min}=-70.0 ms, V_{th}=-70.0 mV, τ=10.0 ms, t_{ref}=2 ms, w_{syn} = 1370 (neurons L3 and L4) and w_{p2i}=8.0 (L2 neurons only).

4.4 Host application

The NHA is used to interface two NDKs, one with implements the HSMD and a second version that includes a speed optimisation (see next section for further details). Algorithm 1 summarises each of the computation stages that occur in the NHA. The communication between the NHA and the NDK is limited by the PCIe bus speed (16 GB/s [5]). In the HSMD, the limitations are only dictated by the CPU speed and the DDR4 memory speed (34.1 GB/s[6]) which is 2 times faster than the PCIe bus speed.

[5] Available online, https://www.trentonsystems.com/blog/pcie-gen4-vs-gen3-slots-speeds, last accessed: 21/06/2021
[6] Available online, https://www.crucial.com/support/articles-faq-memory/understanding-cpu-limitations-with-memory, last accessed: 21/06/2021

4.5 Device kernels

Algorithm 1 NeuroHSMD host application.

Input:
img: image frame;
Output:
post_proc_img: post processed image;
stats: computation statistics;

Main Algorithm:

```
initialise_opencl()
for iterator ← list_folders.begin() to list_folders.end() do
    (x, y) ← get_image_size();
    num_layers ← 3
    tot_neurons ← x.y.num_layers
    reset_opencl_buffers(tot_neurons )
    gsoc ← initialise_gsoc_back_subtraction
    for iterator2 ← files_list.begin() to files_list.end() do
        img ← read_image(iterator2);
        < pixel_values >← gsoc.compute(img)
        NeuroHSMDv1(< pixel_values >) →< spike_sum > {Alg. 2}
        OR
        NeuroHSMDv2(< pixel_values >) →< spike_sum > {Alg. 3}
        post_proc_img ← get_spikes_sum_l3(< spike_sum >)
        save(post_proc_img)
        stats ← compute_stats(time)
    end for
    save(stats)
end for
```

The NHA performs the background subtraction using the GSOC algorithm, performs inference of the SNN kernel (described on the FPGA), and uses the inference results to compute the BS. Furthermore, the NHA can process both live images captured from camera devices or extracted from videos stored on USB or SATA devices.

The NDKs section covers the two kernels (i.e. NeuroHSMDv1 and NeuroHSMDv2) that have been implemented. The Intel Quartus via the Intel FPGA SDK for OpenCL optimised the device kernels to use of the variable-precision DSP blocks, offering IEEE754 single-precision floating point resolution required for performing the target operations. Both kernels use the IEEE754 single-precision floating point representation, and therefore all computations were performed using single-precision arithmetic. The NeuroHSMDv1 is the equivalent implementation to the HSMDv1 where the spike sum per neuron is computed for all the neurons, while NeuroHSMDv2 is the equivalent to HSMDv2 where the spike sum per neuron is only computed for neurons that have pixel intensity values greater than 0.0. Both, NeuroHSMDv1 and NeuroHSMDv2 NDKs implement the HSMD's SNN (see Figure 8). Furthermore, the synaptic weights were stored in vectors and were exchanged via the device and host CPU memory buffers.

The *NeuroHSMDv1* was parallelised by a factor of 16. The parallell factor of 16 was the optimal parallelism coefficient and was obtained experimentally. The NDK version 1 is the equivalent implementation of the HSMD algorithm [5] which is referred to as HSMDv1 in this section. The HSMDv2 is an optimised version of the HSMD where spike sums per neuron are only computed for neurons that have pixel intensity values greater than 0.

The Algorithm 3, inferred by the NHP, summarises the computation steps required to compute the spike sum.

Fig. 8 depicts the seven stages of computation and exhibits the place where each stage occurs (i.e. CPU or FPGA).

Algorithm 2 NeuroHSMDv1

Parallel Circuits: 16

Constants:

R: membrane resistance;

τ: membrane time constant;

dt: time step;

$p2c$: pixel values to current;

$steps$: number of steps;

$s2c$: spike to current;

$number\_neurons$: obtained from largest image to be processed

Input:

$< pixel\_val >$: pixel values;

num_neuron_layer: number of neurons per layer (one per pixel);

Output:

$< spk\_sum >$: spike sum;

Main Algorithm:

for $neuron\_idx \leftarrow 0$ **to** number_neurons **do**
  $I(t) \leftarrow pixel\_val[neuron\_idx].p2c$;
  for $dt1 \leftarrow 0$ **to** steps **do**
    Layer 1:
    $compute\_V_m\_l1[neuron\_idx](I(t))$; {**Eq. 1**}
    $update\_spike\_sum\_l1[neuron\_idx](V_m\_l1)$ {**Eq. 5**};
    Layer 2:
    $I(t)\_l2 \leftarrow spike\_sum\_l1[[neuron\_idx].s2c$;
    $compute\_V_m\_l2[[neuron\_idx](I(t)\_l2)$; {**Eq. 1**}
    $update\_spike\_sum\_l2[neuron\_idx](V_m\_l2)$; {**Eq. 5**};
    Layer 3:
    $I(t)\_l3 \leftarrow spike\_sum\_l2[neuron\_idx].s2c$;
    $compute\_V_m\_l3[neuron\_idx](I(t)\_l2 + I(t)\_l3)$; {**Eq. 1**}
    $update\_spk\_sum[neuron\_idx](V_m\_l3)$; {**Eq. 5**};
  end for
end for

Algorithm 3 NeuroHSMDv2

Parallel Circuits: 16

Constants:

R: membrane resistance;

τ: membrane time constant;

dt: time step;

$p2c$: pixel values to current;

$steps$: number of steps;

$s2c$: spike to current;

$number\_neurons$: obtained from largest image to be processed

Input:

$< pixel\_val >$: pixel values;

num_neuron_layer: number of neurons per layer (one per pixel);

Output:

$< spk\_sum >$: spike sum;

Main Algorithm:

for $neuron\_idx \leftarrow 0$ **to** number_neurons **do**
  if $pixel\_val[neuron\_idx] > 0.0$ **then**
    $I(t) \leftarrow pixel\_val[neuron\_idx].p2c$;
    for $dt1 \leftarrow 0$ **to** steps **do**
      Layer 1:
      $compute\_V_m\_l1[neuron\_idx](I(t))$; {**Eq. 1**}
      $update\_spike\_sum\_l1[neuron\_idx](V_m\_l1)$ {**Eq. 5**};
      Layer 2:
      $I(t)\_l2 \leftarrow spike\_sum\_l1[[neuron\_idx].s2c$;
      $compute\_V_m\_l2[[neuron\_idx](I(t)\_l2)$; {**Eq. 1**}
      $update\_spike\_sum\_l2[neuron\_idx](V_m\_l2)$; {**Eq. 5**};
      Layer 3:
      $I(t)\_l3 \leftarrow spike\_sum\_l2[neuron\_idx].s2c$;
      $compute\_V_m\_l3[neuron\_idx](I(t)\_l2 + I(t)\_l3)$; {**Eq. 1**}
      $update\_spk\_sum[neuron\_idx](V_m\_l3)$; {**Eq. 5**};
    end for
  end if
end for

Similar to NeuroHSMDv1, the *NeuroHSMDv2* was parallelised by a factor of 16. The parallel factor of 16 was the optimal parallelism coefficient and was obtained experimentally. The NDK version 1 contains an optimisation where the spike sum for a given neuron of layer 1 is only computed if the pixel intensity value is greater than 0.0. This optimisation was also applied to the original HSMD Algorithm [5]. The optimised version of the HSMD is called HSMDv2 in this section.

4.6 Datasets and Benchmarking

The NeuroHSMDv1, NeuroHSMDv2, HSMDv1 and HSMDv2 were tested against the CDnet2012 [49] and CDnet2014 [41].

The average performance obtained for each category using each BS method and the HSMD and NeuroHSMD algorithms are characterised via the eight metrics, as shown below. The four base qualitative metrics are: True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) [41, 49].

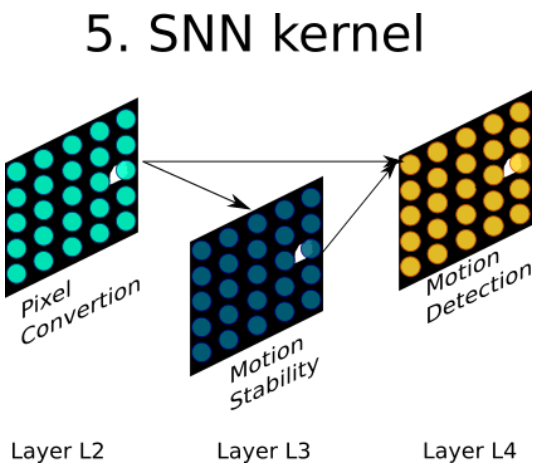

Fig. 8. NDKs implementation. The HSMD's SNN includes: Layer L2 - conversion of pixel values to currents, Layer L3 - motion stability and Layer L4 - motion detection

Recall (Re): $Re = \frac{TP}{TP+FN}$

Re: ranked by **descending order**;

Specificity (Sp): $Sp = \frac{TN}{TN+FP}$;

Sp ranked by **descending order**;

False Positive Rate (FPR): $FPR = \frac{FP}{FP+TN}$;

FPR ranked by **ascending order**;

False Negative Rate (FNR): $FNR = \frac{FN}{FN+TP}$;

FNR ranked by **ascending order**;

Wrong Classifications Rate (WCR): $WCR = \frac{FN+FP}{TP+FN+FP+TN}$;

WCR ranked by **ascending order**;

Correct Classifications Rate (CCR): $CCR = \frac{TP+TN}{TP+FN+FP+TN}$;

CCR ranked by **descending order**;

Precision (Pr): $Pr = \frac{TP}{TP+FP}$;

Pr ranked by **descending order**;

F-measure (F1): $F1 = 2 \times \frac{Pr.Re}{Pr+Re}$

$F1$ ranked by **descending order**;

Average Ranking (R): $R = \frac{\overline{Re}+\overline{Sp}+\overline{FPR}+\overline{FNR}+\overline{WCR}+\overline{CCR}+\overline{F1}}{nMet}$;

R ranked by **ascending order**;

$\overline{AverageRankingacrossallCategories(RC)}$: $\overline{RC} = \frac{Re+Sp+FPR+FNR+WCR+CCR+F1}{nMet}$;

$\overline{RC}$ ranked by **ascending order**;

where $nMet$ is the number of metrics (8 in this case).

The benchmark of the four algorithms (i.e. HSMDv1, HSMDv2, NeuroHSMDv1 and NeuroHSMDv2) and is required to ensure that the four algorithms produce comparative results to that of the original HSMDv1 results when tested against CDnet2012 and CDnet2014 datasets. Furthermore, the OpenCL calls the Intel Quartus, which performs several

Table 2 FPGA resources usage.

Summary					
Info					
Project Names:	snn_pc_v1 snn_pci_v2				
Target Family, Device, Board	Stratix 10, 1SG280LU3F50E1VGS1, de10_pro:s10_sh2e1_4Gx2				
AOC Version	19.1.0 Build 240				
Quartus Version	19.1.0 Build 240 Pro				
Quartus Fit Clock Summary					
Frequency (MHz)					
NeuroHSMDv1 kernel	306.25 (fmax)				
NeuroHSMDv2 kernel	300 (Kernel fmax)				
Quartus Fit Resource Utilisation Summary					
	ALMs	**FFs**	**RAMs**	**DSP blocks**	**MLABs**
NeuroHSMDv1 kernel	488257.1	1060084	2484	1024	3871
NeuroHSMDv2 kernel	486808.4	1034661	2486	1024	3933
Kernel Resource Usage					
	ALUs	**FFs**	**RAMs**	**DSP blocks**	**MLABs**
NeuroHSMDv1 kernel	488257.1	1060084	2484	1024	3871
NeuroHSMDv2 kernel	486808.4	1034661	2486	1024	3933
Global Interconnect					
NeuroHSMDv1 kernel	10629	16485	61	0	0
NeuroHSMDv2 kernel	10629	16485	61	0	0
Board Interface					
NeuroHSMDv1 kernel	13132	20030	112	0	0
NeuroHSMDv2 kernel	13132	20030	112	0	0
System description ROM					
NeuroHSMDv1 kernel	2	71	2	0	0
NeuroHSMDv2 kernel	2	71	2	0	0
Total					
NeuroHSMDv1 kernel	567523 (30%)	907449 (24%)	2963 (25%)	976 (17%)	3786
NeuroHSMDv2 kernel	554199 (30%)	881379 (24%)	3043 (26%)	976 (17%)	3844

hardware optimisations that may include converting from floating-point to fixed-point representation [68], which might affect the accuracy of the NeuroHSMD algorithms during the synthesis step (one of the steps of the OpenCL design flow).

5 RESULTS

The HSMDv1, HSMDv2, NeuroHSMDv1 and NeuroHSMDv2 were all tested on the same computer equipped with a quad-core Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 16GB of DDR3 @ 1600 MHz and 1TB of HDD.

The results section is divided into three parts. Namely, Section 5.1 shows the resources' usage to enable the comparison between the two kernels' complexity, the speed performance results are presented in Section 5.2 and Section 5.3 shows the benchmark results when tested against the CDnet2012 and CDnet2014 datasets.

5.1 Resources Usage

The resources' usage are provided in the report generated by the *aoc* after the successful completion of the kernel compilation, which can take several hours (typically between 6h and 24h depending on the kernel complexity for the DE10pro). The resources' usage for the compilation of the NeuroHSMDv1 and NeuroHSMDv2 kernels is given in Table 2.

From the analysis of Table 2 it can be seen that the estimated resource utilisation is more pessimistic than the final resource utilisation, which is a direct consequence of the Intel Quartus's optimisations during the synthesis and routing phases and to ensure that the circuit fits in the FPGA device. Nevertheless, it takes about 5 minutes to get the *estimated resources usage* and between 6h to 24h to get the *resources' utilisation summary*. Therefore, it is a good practice defining the coefficient **N** in the statement **# PRAGMA UNROLL N** based on the *estimated resources' usage*.

The NeuroHSMDv2 compared with the NeuroHSMDv1, consumes 1448.7 ALMs less, 25423 FFs less, 2 RAMs less and the same number of DSPs. Nevertheless, the NuroHSMDv2 kernel max frequency is 300MHz, while the NuroHSMDv1 kernel max frequency is 306.25 MHz. The NuroHSMDv2 enables saving of less than 1% of resources and introduces an 2% increase in latency.

The coefficient N should always be a multiple of 2^n to ensure the optimal use of FPGA resources. For example, the resources' usage of $N = 48$ is equivalent to $N = 64$. Moreover, both NDKs had failed to compile when $N = 32$ because there was not enough ALUs. The design required more ALUs than those available on the device, violating the compilation rules because the design would not fit on the device.

Finally, the same neuronal parameters were used for all the neurons to reduce resource usage and the amount of data to be exchanged between the host CPU and the FPGA device via the PCIe bus. It is important to highlight that it is only possible to increase the size of the SNN by reducing the complexity of the neuron model, and vice versa.

5.2 Speed Performance

Table 3 displays the speed results obtained for the four algorithms tested against the CDnet2012.

From Table 3 can be seen that both the NeuroHSMDv1 and NeuroHSMDv2 have performed better than the software versions (i.e. HSMDv1 and HSMDv2). It is also apparent that the NeuroHSMDv2 performs better in images with higher resolution (i.e. 720×480 and 720×576) while the NeuroHSMDv1 in lower resolutions (i.e below 720×480). Unlike in software, where it is consistent that the HSMDv2 is always faster than the HSMDv1 (non-opimised version), FPGA optimisations require the utilisation of more resources which might increase latency. Therefore, the NeuroHSMDv2 is only more efficient for resolutions above 288×432.

Overall, the NeuroHSMDv1 had an average frame rate of 71.50 fps, NeuroHSMDv2 63.20 fps, HSMDv1 40.26 fps, HSMDv2 36.11 fps. Finally, the average frame rate for processing images with the native resolution of 720×480 per algorithm is i) NeuroHSMDv2 28.06 fps, NeuroHSMDv1 25.45 fps, HSMDv2 11.19 fps and HSMDv1 9.97 fps.

The speed results obtained for the four algorithms when tested against the CDnet2014 are depicted in Table 4

Table 4 shows that the NeuroHSMDv1 and NeuroHSMDv2 have performed better than the software versions (i.e. HSMDv1 and HSMDv2) when tested against the CDnet2014 dataset. Once again, the NeuroHSMDv2 performs
The results for each of the eleven categories shared by both CDnet2012 and CDnet2014 are shown in Figure 9. better

Table 3 CDnet2012 speed result

Category	number of images	height [pixels]	width [pixels]	NeuroHSMDv2 [fps]	NeuroHSMDv1 [fps]	HSMDv2 [fps]	HSMDv1 [fps]
baseline/PETS2006	1199	576	720	23.66	20.45	8.40	9.73
cameraJitter/badminton	1149	480	720	28.33	25.48	10.01	11.50
dynamicBackground/fall	3999	480	720	27.28	25.01	9.87	11.08
shadow/copyMachine	3399	480	720	28.56	25.86	10.04	10.98
dynamicBackground/fountain01	1183	288	432	55.17	58.30	27.40	29.99
dynamicBackground/fountain02	1498	288	432	56.12	58.13	27.64	30.37
intermittentObjectMotion/abandonedBox	4499	288	432	55.93	58.45	26.88	29.89
intermittentObjectMotion/tramstop	3199	288	432	55.35	58.66	27.05	29.59
thermal/park	599	288	352	55.72	59.96	28.99	33.02
shadow/peopleInShade	1198	244	380	66.85	74.06	36.05	38.74
baseline/highway	1699	240	320	72.48	85.70	46.97	53.00
baseline/office	2049	240	360	69.49	78.86	40.24	46.71
baseline/pedestrians	1098	240	360	69.59	78.73	40.18	47.00
cameraJitter/boulevard	2499	240	352	68.96	78.81	41.21	46.98
cameraJitter/sidewalk	1199	240	352	69.04	78.54	39.49	47.32
cameraJitter/traffic	1569	240	320	72.17	85.29	43.95	51.40
dynamicBackground/boats	7998	240	320	71.86	84.29	45.33	51.70
dynamicBackground/canoe	1188	240	320	71.98	83.86	44.15	52.95
dynamicBackground/overpass	2999	240	320	71.98	84.56	43.69	49.31
intermittentObjectMotion/parking	2499	240	320	72.98	85.21	44.68	48.55
intermittentObjectMotion/sofa	2749	240	320	73.56	86.46	44.37	47.96
intermittentObjectMotion/streetLight	3199	240	320	72.01	84.95	44.31	47.76
intermittentObjectMotion/winterDriveway	2499	240	320	72.98	85.88	44.72	49.21
shadow/backdoor	1999	240	320	72.06	85.77	44.21	48.87
shadow/bungalows	1699	240	360	68.61	78.36	39.80	42.25
shadow/busStation	1249	240	360	68.79	78.78	39.85	42.91
shadow/cubicle	7399	240	352	70.87	80.47	41.07	44.52
thermal/corridor	5399	240	320	73.80	87.01	44.58	48.10
thermal/diningRoom	3699	240	320	73.38	86.60	44.38	47.82
thermal/lakeSide	6499	240	320	73.91	86.80	45.62	49.64
thermal/library	4899	240	320	74.83	87.05	44.40	49.36

Best results are highlighted using grey.

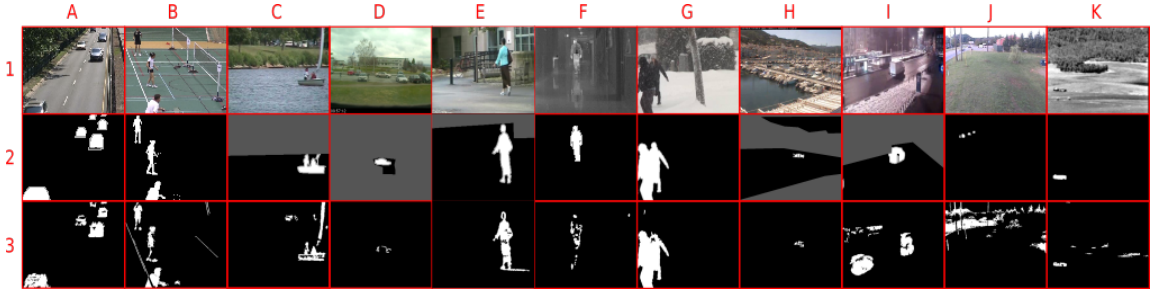

Fig. 9. Results obtained for each of the eleven of the five categories (columns A to F) are common to both CDnet2012 and CDnet2014 datasets, while the remaining six categories (columns G to K) are only available on CDnet2014 dataset. Column A: baseline; B: camera jitter; C: dynamic background; D: dynamic object motion; E: shadow, F: thermal, G: bad weather, H: low frame rate; I: night videos, J: PTZ and K: turbulence. Row 1: RGB image; 2: ground-truth; and 3: NeuroHSMD binarised. The raw images, shown in the first row, demonstrate the scenarios that can be found in both datasets. The corresponding ground truth images, presented in the second row, show the 5 labels, namely, i) static [greyscale value 0], ii) shadow [greyscale value 50], iii) non-Region of Interest (ROI) [greyscale value 85], iv) unknown [greyscale value 170] and v) moving [greyscale value 255]. The corresponding binarised images generated by the NeuroHSMD are shown in the third row.

Table 4 CDnet2014 speed results

Category	number of images	height [pixels]	width [pixels]	NeuroHSMDv2 [fps]	NeuroHSMDv1 [fps]	HSMDv2 [fps]	HSMDv1 [fps]
badWeather/blizzard	6999	480	720	29.70	24.55	10.12	11.21
badWeather/snowFall	6499	480	720	29.61	24.31	10.16	11.11
badWeather/wetSnow	3499	540	720	26.54	21.76	8.93	10.19
baseline/PETS2006	1199	576	720	25.04	20.36	8.52	9.53
cameraJitter/badminton	1149	480	720	28.58	25.07	9.80	11.00
dynamicBackground/fall	3999	480	720	27.48	24.75	13.90	10.96
shadow/copyMachine	3399	480	720	28.76	25.66	14.10	11.51
turbulence/turbulence0	4999	480	720	28.59	25.07	14.26	11.62
turbulence/turbulence1	3999	480	720	28.24	25.21	14.28	11.35
turbulence/turbulence3	2199	486	720	28.72	25.15	14.07	11.49
PTZ/continuousPan	1699	480	704	28.49	23.91	9.88	10.85
lowFramerate/tunnelExit_0_35fps	3999	440	700	31.45	28.23	15.94	12.47
nightVideos/fluidHighway	1363	450	700	31.17	27.64	15.27	12.10
turbulence/turbulence2	4499	315	645	41.85	39.59	23.90	18.93
lowFramerate/port_0_17fps	2999	480	640	31.35	28.17	15.75	12.39
lowFramerate/tramCrossroad_1fps	899	350	640	39.51	36.73	21.37	16.71
nightVideos/busyBoulvard	2759	364	640	38.86	36.00	20.90	16.43
nightVideos/bridgeEntry	2499	430	630	34.52	31.85	17.90	14.20
nightVideos/winterStreet	1784	420	624	35.92	32.57	18.26	14.52
nightVideos/streetCornerAtNight	5199	245	595	52.86	51.56	32.83	25.35
PTZ/twoPositionPTZCam	2299	340	570	43.30	41.11	17.91	18.92
PTZ/intermittentPan	3499	368	560	40.65	38.52	16.39	17.71
badWeather/skating	3899	360	540	43.00	40.79	17.20	19.08
nightVideos/tramStation	2999	295	480	53.15	52.62	33.46	26.15
dynamicBackground/fountain01	1183	288	432	55.38	57.06	37.95	30.16
dynamicBackground/fountain02	1498	288	432	56.56	56.97	38.22	30.49
intermittentObjectMotion/abandonedBox	4499	288	432	55.64	58.03	38.19	30.26
intermittentObjectMotion/tramstop	3199	288	432	56.58	57.56	38.03	28.99
shadow/peopleInShade	1198	244	380	67.31	71.54	49.84	41.11
baseline/office	2049	240	360	71.72	75.86	37.85	45.39
baseline/pedestrians	1098	240	360	70.75	75.87	40.13	45.66
shadow/bungalows	1699	240	360	69.25	75.88	55.53	45.12
shadow/busStation	1249	240	360	69.51	74.76	55.41	46.24
cameraJitter/boulevard	2499	240	352	70.44	75.44	39.33	42.60
cameraJitter/sidewalk	1199	240	352	69.44	75.95	38.54	42.36
shadow/cubicle	7399	240	352	71.78	77.42	57.46	46.65
thermal/park	599	288	352	57.07	58.39	43.26	36.78
PTZ/zoomInZoomOut	1129	240	320	72.70	80.89	41.94	44.97
baseline/highway	1699	240	320	74.12	82.40	42.85	46.89
cameraJitter/traffic	1569	240	320	72.87	81.54	40.97	45.95
dynamicBackground/boats	7998	240	320	73.14	82.14	60.35	47.99
dynamicBackground/canoe	1188	240	320	72.64	81.62	61.19	46.91
dynamicBackground/overpass	2999	240	320	72.85	82.26	61.72	50.00
intermittentObjectMotion/parking	2499	240	320	73.56	81.88	61.94	51.23
intermittentObjectMotion/sofa	2749	240	320	73.57	82.95	62.15	47.82
intermittentObjectMotion/streetLight	3199	240	320	72.77	82.48	62.27	48.13
intermittentObjectMotion/winterDriveway	2499	240	320	74.19	82.81	62.96	48.48
lowFramerate/turnpike_0_5fps	1499	240	320	73.10	82.38	60.99	46.52
shadow/backdoor	1999	240	320	73.73	83.84	62.82	51.77
thermal/corridor	5399	240	320	74.86	83.65	62.48	51.11
thermal/diningRoom	3699	240	320	74.42	83.86	62.34	50.98
thermal/lakeSide	6499	240	320	74.93	83.60	63.23	51.68
thermal/library	4899	240	320	75.79	83.72	62.17	47.23

Best results are highlighted using grey.

Table 5 CDnet2012 Overall ranks

Method	$\overline{RC}\downarrow$	Re ↑	Sp ↑	FPR ↓	FNR ↓	WCR ↓	CCR ↑	F1 ↑	Pr ↑
HSMDv1	2.8	0.52	0.994	0.006	0.23	0.024	0.976	0.77	0.62
HSMDv2	2.8	0.52	0.994	0.006	0.23	0.024	0.976	0.77	0.62
NeuroHSMDv1	2.8	0.52	0.994	0.006	0.23	0.024	0.976	0.77	0.62
NeuroHSMDv2	2.8	0.52	0.994	0.006	0.23	0.024	0.976	0.77	0.62
GSOC [5]	3.5	0.54	0.993	0.007	0.25	0.024	0.976	0.75	0.63
MOG2 [5]	3.8	0.37	0.995	0.004	0.24	0.026	0.974	0.76	0.50
GMG [5]	3.9	0.20	0.998	0.002	0.21	0.033	0.967	0.79	0.32
KNN [5]	4.3	0.39	0.995	0.005	0.26	0.025	0.975	0.74	0.51
MOG [5]	4.5	0.32	0.996	0.004	0.26	0.030	0.970	0.74	0.44
CNT [5]	6.1	0.73	0.927	0.073	0.71	0.081	0.919	0.29	0.41
LSBP[5]	7.3	0.57	0.90	0.096	0.80	0.109	0.891	0.20	0.29

↑: the highest score is the best. ↓: the lowest result is the best.
All the 4 methods were ranked first because no changes were made to the customised SNN. *Re* stands for Recall, *Sp* Specificity, FPR False Positive Rate, FNR False Negative Rate, WCR Wrong Classifications Rate, CCR Correct Classifications Rate, Pr Precision, F1 F-score and $\overline{RC}$ Average Ranking across all Categories.

in images with higher resolution (i.e. equal or higher than 480×295) while the NeuroHSMDv1 in lower resolutions (i.e below 480×295). Again, the NeuroHSMDv2 is only more efficient for resolutions above 288×432 because of the complexity and latency introduced by the optimisation circuit.

Overall, the NeuroHSMDv1 has an average frame rate of 43.51 fps, NeuroHSMDv2 39.94 fps, HSMDv2 29.30 fps, and HSMDv1 25.18 fps. It is essential to highlight that the CDnet2014 has more categories and image sequences, leading to different frame rates for the CDnet2012 and CDnet2014 datasets.

The average frame rate for processing images with the native resolution of 720×480 per algorithm was i) NeuroHSMDv2 28.71 fps, NeuroHSMDv1 24.95 fps, HSMDv2 12.37 fps and HSMDv1 11.25 fps. These results are in line with the results obtained for the 4 algorithms when tested against the CDnet2012 dataset. Although HSMDv1 and HSMDv2 have archived near-real time performance, it is also clear that CPUs are not ideal for accelerating SNNs. Furthermore, to achieve such performance on the CPU all the non-essential applications were closed to ensure that all the memory and computational resources were free to compute the HSMDv1 and HSMDv2 with maximum efficiency. Furthermore, the parallelisation process takes place in the FPGA, where the device kernel is defined in its internal logic. Data is exchanged between the device and host SDRAMs through buffers. The FPGA implementation contains 16 parallel circuits, which were configured using the IOCL. In this work, we did not explore learning rules (such as STDP, Hebbian, Oja, etc.) as these would require additional FPGA resources.

5.3 Benchmark

Table 5 shows the results obtained after testing the 4 methods against the CDnet2012 ground-truth images using the scripts provided by Nil Goyette et al. [49].

From the results shown in Table 5 it is possible to infer that the results obtained with the four methods are the same because all the methods were ranked in first place with the same values per metric. Indexing all the algorithms in the

Table 6 CDnet2014 Overall ranks

Method	$\overline{RC}\downarrow$	Re ↑	Sp ↑	FPR ↓	FNR ↓	WCR ↓	CCR ↑	F1 ↑	Pr ↑
HSMDv1	2.9	0.55	0.993	0.007	0.35	0.018	0.982	0.65	0.60
HSMDv2	2.9	0.55	0.993	0.007	0.35	0.018	0.982	0.65	0.60
NeuroHSMDv1	2.9	0.55	0.993	0.007	0.35	0.018	0.982	0.65	0.60
NeuroHSMDv2	2.9	0.55	0.993	0.007	0.35	0.018	0.982	0.65	0.60
GSOC [5]	3.0	0.40	0.995	0.005	0.38	0.017	0.983	0.62	0.48
KNN [5]	3.5	0.34	0.996	0.004	0.32	0.019	0.981	0.68	0.45
GMG[5]	4.3	0.24	0.997	0.003	0.36	0.022	0.978	0.64	0.35
MOG [5]	4.4	0.58	0.991	0.009	0.39	0.019	0.981	0.61	0.60
MOG2 [5]	4.5	0.39	0.994	0.006	0.42	0.018	0.982	0.58	0.47
LSBP [5]	6.5	0.58	0.945	0.055	0.79	0.064	0.936	0.21	0.31
CNT	7.0 [5]	0.72	0.930	0.070	0.80	0.075	0.925	0.20	0.32

↑: the highest score is the best. ↓: the lowest result is the best.
All the 4 methods were ranked first because no changes were made to the customised SNN. *Re* stands for Recall, *Sp* Specificity, FPR False Positive Rate, FNR False Negative Rate, WCR Wrong Classifications Rate, CCR Correct Classifications Rate, Pr Precision, F1 F-score and $\overline{RC}$ Average Ranking across all Categories.

first place was expected because the speed optimisation in version 2 of the NeuroHSMD and HSMD should not interfere with the model dynamics. Furthermore, the HSMDv1, HSMDv2, NeuroHSMDv1 and NeuroHSMDv2 showed poor performance in both dynamic backgrounds and low frame rate conditions, indicating that the spiking neuron model is not effective in accurately distinguishing the type of motion. This is likely due to the fact that in the vertebrate retina, only ganglion cells are spiking cells and the distinction between the main object and shadows is performed by other non-spiking cells. However, the integration of the GSOC algorithm and the SNN in a new approach has significantly improved the accuracy of the GSOC algorithm by mimicking the basic functionality of OMS-GC.

Table 6 depicts the results obtained after testing 4 methods against the CDnet2014 ground-truth images using the scripts provided by Nil Goyette et al. [49].

From the results shown in Table 6 it is possible to infer that the results obtained with the four methods are the same because the four methods were ranked first with the same values per metric. These results are important because it is possible to infer that there has been no degradation in accuracy as a consequence of the hardware acceleration. Again, the HSMDv1, HSMDv2, NeuroHSMDv1, and NeuroHSMDv2 demonstrated subpar results in dynamic backgrounds and low frame rate situations, demonstrating that the spiking neuron model is not capable of accurately identifying the type of motion. This is likely a result of only ganglion cells being spiking cells in the vertebrate retina, where the distinction between the main object and shadows is handled by non-spiking cells. However, combining the GSOC algorithm and the SNN in a novel approach has significantly enhanced the accuracy of the GSOC algorithm by replicating the basic operations of OMS-GC.

6 CONCLUSIONS AND FUTURE WORK

Two bio-inspired NeuroHSMD have been proposed to accelerate the HSMD algorithm [5]. The NeuroHSMDv1 and NeuroHSMDv2 (speed optimisation) were tested against the CDnet2012 and CDnet2014 datasets. The NeuroHSMDv1 has

lower latency when processing images with resolutions equal to or greater than 480×295. The NeuronHSMDv2 (speed optimisation) has a lower latency when processing images with resolutions smaller than 480×295. Two HSMD versions were used (the original HSMDv1 algorithm [5] and HSMDv2 with speed optimisation) for ensuring a fair comparison between the software and hardware implementations. The HSMDv1, HSMDv2, NeuroHSMDv1 and NeuroHSMDv2 where all tested on the same computer equipped with a quad-core Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 16GB of DDR3 @ 1600 MHz and 1TB of HDD. The average frame rate for processing images with the native resolution of 720×480 per algorithm was: 1) **CDnet2012:** i) NeuroHSMDv2 28.06 fps, NeuroHSMDv1 25.45 fps, HSMDv2 11.19 fps and HSMDv1 9.97 fps; and 2) **CDnet2014:** i) NeuroHSMDv2 28.71 fps, NeuroHSMDv1 24.95 fps, HSMDv2 12.37 fps and HSMDv1 11.25 fps.

The four methods were also tested against the ground-truth images available in the CDnet2012 and CDnet2014 datasets using the eight metrics, were used to assess and compare the quality of the HSMD algorithm. The four methods obtained the same values for all the metrics and were all ranked first. The first place acquired by the four methods is an indication that there was no degradation with the hardware acceleration.

Finally, the NeuroHSMD is the first Neuromorphic SNN accelerator capable of accelerating thousands of spiking neuron models in parallel using OpenCL. Moreover, the proposed method can be used by non-engineering background users for accelerating SNNs in different heterogenous platforms using OpenCL. It is also possible to conclude that the results on the FPGA are the same as the results obtained in the CPU implementation, meaning that the target FPGA offers sufficient IEEE754 single-precision DSP blocks to accelerate the SNN kernel without degradation of the results.

Future work includes optimising the HSMD algorithm to detect and track motion in challenging scenarios (e.g. low frame rate, dynamic background and camera jitter) and optimise those SNNs to run in affordable lower-end FPGAs. The implementation and evaluation of more complex retinal cells (such as direction sensitive and predictive cells) using SNNs and target lower-end and affordable FPGA devices is also planned.

REFERENCES

[1] L. A Pastur-Romay, A. B Porto-Pazos, F. Cedrón, and A. Pazos, "Parallel computing for brain simulation," *Current Topics in Medicinal Chemistry*, vol. 17, no. 14, pp. 1646–1668, 2017.

[2] R. Brooks, D. Hassabis, D. Bray, and A. Shashua, "Is the brain a good model for machine intelligence?," *Nature*, vol. 482, no. 7386, p. 462, 2012.

[3] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking neural networks hardware implementations and challenges: A survey," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15, no. 2, pp. 1–35, 2019.

[4] M. ZHANG, G. Zonghua, and P. Gang, "A survey of neuromorphic computing based on spiking neural networks," *Chinese Journal of Electronics*, vol. 27, no. 4, pp. 667–674, 2018.

[5] P. Machado, A. Oikonomou, J. F. Ferreira, and T. McGinnity, "Hsmd: An object motion detection algorithm using a hybrid spiking neural network architecture," *IEEE Access*, pp. 1–1, 2021. Available onine, https://doi.org/10.1109/ACCESS.2021.3111005, last accessed: 2021-07-31.

[6] J. F. Ferreira, J. Lobo, and J. Dias, "Bayesian real-time perception algorithms on gpu," *Journal of Real-Time Image Processing*, vol. 6, no. 3, pp. 171–186, 2011.

[7] J. F. Ferreira, M. Castelo-Branco, and J. Dias, "A hierarchical bayesian framework for multimodal active perception," *Adaptive Behavior*, vol. 20, no. 3, pp. 172–190, 2012.

[8] J. Ferreira, J. Lobo, P. Bessiere, M. Castelo-Branco, and J. Dias, "A bayesian framework for active artificial perception," *IEEE transactions on cybernetics*, vol. 43, no. 2, pp. 699–711, 2013.

[9] Intel, "What Is a GPU? Graphics Processing Units Defined," 2020. Available online, https://www.intel.co.uk/content/www/uk/en/products/docs/processors/what-is-a-gpu.html, last accessed: 2021-03-10.

[10] Intel, "FPGA vs. GPU for Deep Learning Applications – Intel."

[11] R. Duarte, J. Lobo, J. F. Ferreira, and J. Dias, "Synthesis of bayesian machines on fpgas using stochastic arithmetic," in *2nd International Workshop on Neuromorphic and Brain-Based Computing Systems (NeuComp 2015), Design Automation Test Europe (DATE2015)*, 2015.

[12] J. C. Knight and T. Nowotny, "Gpus outperform current hpc and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model," *Frontiers in neuroscience*, p. 941, 2018.

[13] T. Gollisch and M. Meister, "Eye smarter than scientists believed: neural computations in circuits of the retina," *Neuron*, vol. 65, no. 2, pp. 150–164, 2010.
[14] H. Kolb, "How the retina works: Much of the construction of an image takes place in the retina itself through the use of specialized neural circuits," *American scientist*, vol. 91, no. 1, pp. 28–35, 2003.
[15] M.-N. Chapel and T. Bouwmans, "Moving objects detection with a moving camera: A comprehensive review," *Computer science review*, vol. 38, p. 100310, 2020.
[16] P. K. Tadiparthi, S. Ponnada, K. Jhansi, P. K. Bheemavarapu, and A. Gottimukkala, "A comprehensive review of moving object identification using background subtraction in dynamic scenes," *Solid State Technology*, vol. 64, no. 2, pp. 4114–4124, 2021.
[17] B. Garcia-Garcia, T. Bouwmans, and A. J. R. Silva, "Background subtraction in real applications: Challenges, current models and future directions," *Computer Science Review*, vol. 35, p. 100204, 2020.
[18] T. Bouwmans, F. El Baf, and B. Vachon, "Statistical background modeling for foreground detection: A survey," in *Handbook of pattern recognition and computer vision*, pp. 181–199, World Scientific, 2010.
[19] T. Bouwmans, "Traditional and recent approaches in background modeling for foreground detection: An overview," *Computer science review*, vol. 11, pp. 31–66, 2014.
[20] T. Bouwmans, F. Porikli, B. Höferlin, and A. Vacavant, *Background modeling and foreground detection for video surveillance.* CRC press, 2014.
[21] B. Garcia-Garcia, T. Bouwmans, and A. J. R. Silva, "Background subtraction in real applications: Challenges, current models and future directions," *Computer Science Review*, vol. 35, p. 100204, 2020.
[22] A. M. McIvor, "Background subtraction techniques," *Proc. of Image and Vision Computing*, vol. 4, pp. 3099–3104, 2000.
[23] M. Piccardi, "Background subtraction techniques: a review," in *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, vol. 4, pp. 3099–3104, IEEE, 2004.
[24] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland, "Pfinder: Real-time tracking of the human body," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 19, no. 7, pp. 780–785, 1997.
[25] D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, and S. Russell, "Towards robust automatic traffic scene analysis in real-time," in *Proceedings of 12th International Conference on Pattern Recognition*, vol. 1, pp. 126–131, IEEE, 1994.
[26] B. P. L. Lo and S. Velastin, "Automatic congestion detection system for underground platforms," in *Proceedings of 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing. ISIMP 2001 (IEEE Cat. No. 01EX489)*, pp. 158–161, IEEE, 2001.
[27] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati, "Detecting moving objects, ghosts, and shadows in video streams," *IEEE transactions on pattern analysis and machine intelligence*, vol. 25, no. 10, pp. 1337–1342, 2003.
[28] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Proceedings. 1999 IEEE computer society conference on computer vision and pattern recognition (Cat. No PR00149)*, vol. 2, pp. 246–252, IEEE, 1999.
[29] B. Han, D. Comaniciu, and L. Davis, "Sequential kernel density approximation through mode propagation: Applications to background modeling," in *proc. ACCV*, vol. 4, pp. 818–823, Citeseer, 2004.
[30] A. Elgammal, D. Harwood, and L. Davis, "Non-parametric model for background subtraction," in *European conference on computer vision*, pp. 751–767, Springer, 2000.
[31] M. Seki, T. Wada, H. Fujiwara, and K. Sumi, "Background subtraction based on cooccurrence of image variations," in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 2, pp. II–II, IEEE, 2003.
[32] N. M. Oliver, B. Rosario, and A. P. Pentland, "A bayesian computer vision system for modeling human interactions," *IEEE transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 831–843, 2000.
[33] S.-C. S. Cheung and C. Kamath, "Robust background subtraction with foreground validation for urban traffic video," *EURASIP Journal on Advances in Signal Processing*, vol. 2005, no. 14, pp. 1–11, 2005.
[34] S. Y. Elhabian, K. M. El-Sayed, and S. H. Ahmed, "Moving object detection in spatial domain using background removal techniques-state-of-art," *Recent patents on computer science*, vol. 1, no. 1, pp. 32–54, 2008.
[35] M. Cristani, M. Farenzena, D. Bloisi, and V. Murino, "Background subtraction for automated multisensor surveillance: a comprehensive review," *EURASIP Journal on Advances in signal Processing*, vol. 2010, pp. 1–24, 2010.
[36] A. Elgammal, "Background subtraction: Theory and practice," *Synthesis Lectures on Computer Vision*, vol. 5, no. 1, pp. 1–83, 2014.
[37] P. KaewTraKulPong and R. Bowden, "An improved adaptive background mixture model for real-time tracking with shadow detection," in *Video-based surveillance systems*, pp. 135–144, Springer, 2002.
[38] Z. Zivkovic, "Improved adaptive gaussian mixture model for background subtraction," in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 2, pp. 28–31, IEEE, 2004.
[39] Z. Zivkovic and F. Van Der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," *Pattern recognition letters*, vol. 27, no. 7, pp. 773–780, 2006.
[40] S. Zeevi, "BackgroundSubtractorCNT: A Fast Background Subtraction Algorithm," Dec. 2016. Available online, https://doi.org/10.5281/zenodo.4267853, last accessed: 23/11/2020.
[41] Y. Wang, P.-M. Jodoin, F. Porikli, J. Konrad, Y. Benezeth, and P. Ishwar, "Cdnet 2014: An expanded change detection benchmark dataset," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 387–394, 2014.

[42] S. Zeevi, "Fastest background subtraction is BackgroundSubtractorCNT," 2016. Available onine, https://www.theimpossiblecode.com/blog/fastest-background-subtraction-opencv/, last accessed: 09/11/2020.
[43] A. B. Godbehere and K. Goldberg, "Algorithms for visual tracking of visitors under variable-lighting conditions for a responsive audio art installation," in *Controls and art*, pp. 181–204, Springer, 2014.
[44] L. Guo, D. Xu, and Z. Qiang, "Background subtraction using local svd binary pattern," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 86–94, 2016.
[45] OpenCV, "Background Subtractor GSOC," 2020. Available online, https://docs.opencv.org/4.5.0/d4/dd5/classcv_1_1bgsegm_1_1BackgroundSubtractorGSOC.html, last accessed: 06/11/2020.
[46] V. Samsonov, "Improved background subtraction algorithm," Nov. 2017. Available online, https://zenodo.org/record/4269865, last accessed: 23/11/2020.
[47] Google, "Google Summer of Code Archive," 2017. Available online, https://summerofcode.withgoogle.com/archive/2017/projects/, last accessed: 2021-07-28.
[48] V. Samsonov, "Improvement of the background subtraction algorithm," 2017. Available online, https://summerofcode.withgoogle.com/archive/2017/projects/6453014550282240/, last accessed: 23/11/2020.
[49] N. Goyette, P.-M. Jodoin, F. Porikli, J. Konrad, and P. Ishwar, "Changedetection. net: A new change detection benchmark dataset," in *2012 IEEE computer society conference on computer vision and pattern recognition workshops*, pp. 1–8, IEEE, 2012.
[50] OpenCV, "OpenCV: Operations on arrays," 2021. Available online, https://docs.opencv.org/3.4/d4/dd5/classcv_1_1bgsegm_1_1BackgroundSubtractorGSOC.html, last accessed: 2021-05-27.
[51] M. Braham, S. Pierard, and M. Van Droogenbroeck, "Semantic background subtraction," in *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 4552–4556, Ieee, 2017.
[52] D. Zeng, M. Zhu, and A. Kuijper, "Combining background subtraction algorithms with convolutional neural network," *Journal of Electronic Imaging*, vol. 28, no. 1, p. 013011, 2019.
[53] X. Liang, S. Liao, X. Wang, W. Liu, Y. Chen, and S. Z. Li, "Deep background subtraction with guided learning," in *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, IEEE, 2018.
[54] D. Zeng and M. Zhu, "Background subtraction using multiscale fully convolutional network," *IEEE Access*, vol. 6, pp. 16010–16021, 2018.
[55] D. Zeng, X. Chen, M. Zhu, M. Goesele, and A. Kuijper, "Background subtraction with real-time semantic segmentation," *IEEE Access*, vol. 7, pp. 153869–153884, 2019.
[56] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, no. 1-3, pp. 239–255, 2010.
[57] G. Li, V. Talebi, A. Yoonessi, and C. L. Baker Jr, "A fpga real-time model of single and multiple visual cortex neurons," *Journal of neuroscience methods*, vol. 193, no. 1, pp. 62–66, 2010.
[58] J. Li, Y. Katori, and T. Kohno, "An fpga-based silicon neuronal network with selectable excitability silicon neurons," *Frontiers in neuroscience*, vol. 6, p. 183, 2012.
[59] A. S. Cassidy, J. Georgiou, and A. G. Andreou, "Design of silicon brains in the nano-cmos era: Spiking neurons, learning synapses and neural architecture optimization," *Neural Networks*, vol. 45, pp. 4–26, 2013.
[60] Q. Chen, J. Wang, S. Yang, Y. Qin, B. Deng, and X. Wei, "A real-time fpga implementation of a biologically inspired central pattern generator network," *Neurocomputing*, vol. 244, pp. 63–80, 2017.
[61] A. O. Komendantov and N. I. Kononenko, "Deterministic chaos in mathematical model of pacemaker activity in bursting neurons of snail, helix pomatia," *Journal of theoretical biology*, vol. 183, no. 2, pp. 219–230, 1996.
[62] K. Cheung, S. R. Schultz, and W. Luk, "Neuroflow: a general purpose spiking neural network simulation platform using customizable processors," *Frontiers in neuroscience*, vol. 9, p. 516, 2016.
[63] A. Podobas and S. Matsuoka, "Designing and accelerating spiking neural networks using opencl for fpgas," in *2017 International Conference on Field Programmable Technology (ICFPT)*, pp. 255–258, IEEE, 2017.
[64] V. Sakellariou and V. Paliouras, "An fpga accelerator for spiking neural network simulation and training," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2021.
[65] Intel, "Using Intel FPGA SDK for OpenCL on DE-Series Boards," 2019. Available online, https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Tutorials/OpenCL_On_DE_Series_Boards.pdf, last accessed: 2021-04-02.
[66] CodeProjects, "Part 5: OpenCL Buffers and Memory Affinity - CodeProject," 2011. Available online, https://www.codeproject.com/Articles/201258/Part-5-OpenCL-Buffers-and-Memory-Affinity, last accessed: 2021-04-02.
[67] Intel, "Intel FPGA SDK for OpenCL Pro Edition Best Practices Guide," 2021. Available online, https://www.intel.com/content/www/us/en/docs/programmable/683521/21-4/introduction-to-pro-edition-best-practices.html, last accessed: 2021-04-02.
[68] Intel, "Intel quartus prime pro edition user guide: Design optimization - 3.3.1. floating-point versus fixed-point representations." https://www.intel.com/content/www/us/en/docs/programmable/683176/18-1/floating-point-versus-fixed-point-representations.html.
[69] Terasic, "Terasic - DE10-Pro," 2021. Available onine, https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=13&No=1144&PartNo=2, last accessed: 2021-07-31.
[70] W. Gerstner and W. M. Kistler, *Spiking Neuron Models*. Cambridge: Cambridge University Press, 2002.
[71] E. M. Izhikevich, "Which model to use for cortical spiking neurons?," *IEEE transactions on neural networks*, vol. 15, no. 5, pp. 1063–1070, 2004.

[72] R. Jolivet, T. J. Lewis, and W. Gerstner, "Generalized integrate-and-fire models of neuronal activity approximate spike trains of a detailed model to a high degree of accuracy," *Journal of neurophysiology*, vol. 92, no. 2, pp. 959–976, 2004.

[73] R. Brette and W. Gerstner, "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity," *Journal of neurophysiology*, vol. 94, no. 5, pp. 3637–3642, 2005.