Neuromorphic algorithms and hardware for event-based processing

Gregor Lenz

THÈSE DE DOCTORAT
DE SORBONNE UNIVERSITÉ

Spécialité : Ingénierie Neuromorphique

École doctorale nº391: Sciences mécaniques, acoustique, électronique et robotique

Sujet de la thèse :

Neuromorphic Algorithms and Hardware for Event-based Processing

réalisée

à l'Institut de la Vision - Équipe vision et calcul naturel

sous la direction de Sio-Hoi Ieng

présentée par

Gregor Lenz

pour obtenir le grade de :

DOCTEUR DE SORBONNE UNIVERSITÉ

soutenue le 6 juillet 2021

devant le jury composé de :

Pr.	Alejandro Linares-Barranco	Rapporteur
Pr.	Sylvain Saïghi	Rapporteur
Pr.	Bruno Gas	Examinateur
Dr.	Sio-Hoi Ieng	Directeur de thèse

Neuromorphic Algorithms and Hardware for Event-based Processing

Abstract: The demand for computing power steadily increases to enable new and more intelligent functionalities in our current technology. The combined computing power of mobile systems such as phones, drones, autonomous vehicles and embedded systems increases rapidly, but each system has a limited power budget. Efficient computation is thus of utmost importance. For the past decades we have relied on the growing amount of transistors per unit area to keep up with computing demand while keeping power consumption in check, but this trend is declining as transistor sizes are reaching physical limits. While architecture improvements stagnate, we find ourselves in the early stages of creating intelligent systems, which raises the question how current system can scale and which makes the exploration of alternative computing principles worth wile. This thesis examines the role of new bio-inspired computation paradigms for low-power computation, to drive a future generation of intelligent systems. *Neuromorphic computing* is an emerging interdisciplinary field that looks at biological systems such as the retina or the brain for inspiration on how to compute efficiently. From that it is possible to create sensors, algorithms and hardware that process information much closer to how the biological model works than current conventional computer architecture. We examine how neuromorphic cameras, algorithms and hardware can gradually replace conventional components to make the system overall use less power. We approach the issue through the lens of efficiency, and propose an event-based face detection algorithm, a framework that brings event-based computer vision to mobile devices with optimised hardware and methods based on precise timing for spiking neural networks on neuromorphic hardware. In this attempt we bring technology into being that starts to resemble the organic counterpart, to show the capabilities of brain-inspired computing.

Keywords: neuromorphic computing, event-based processing, non-von neumann computing, low-power computer vision, neuromorphic hardware, spiking neural networks

Algorithmes et Architectures Matérielles Neuromorphiques pour le Calcul Évènementiel

Résumé : La demande en puissance de calcul augmente régulièrement pour permettre de nouvelles fonctionnalités plus intelligentes au vu de la technologie actuelle. La puissance de calcul disponible des systèmes mobiles tels que les téléphones, les drones, les véhicules autonomes et les systèmes embarqués augmente rapidement, mais chaque système a un budget limité. Le calcul efficace est donc de la plus haute importance. Au cours des dernières décennies, nous nous sommes appuyés sur la densité croissante de transistors integrés dans un processeur, pour répondre à la demande en puissance de calcul tout en maîtrisant la consommation d'énergie, mais cette tendance diminue à mesure que les tailles des transistors atteignent leurs limites physiques. Alors que les améliorations de l'architecture stagnent, nous nous trouvons dans les premières étapes de la création de systèmes intelligents, ce qui rend l'exploration de principes de calcul alternatifs indispensable. Cette thèse examine le rôle des nouveaux paradigmes de calcul bio-inspirés permettant le calcul à faible coût indispensable à la conception de la future génération de systèmes intelligents. Le calcul neuromorphique est un domaine interdisciplinaire émergent qui s'inspire des systèmes biologiques tels que la rétine ou le cerveau pour calculer efficacement. À partir de là, il est possible de créer des capteurs, des algorithmes et du matériel qui traitent les informations de façon bio-inspirée. Nous examinons comment les caméras neuromorphiques, les algorithmes et ainsi que le matériel peuvent remplacer progressivement les composants conventionnels pour arriver à un système moins gourmand en énergie. Nous abordons le problème à travers le prisme de l'efficacité et proposons un algorithme bio-inspiré de détection de visage, puis un cadriciel permettant de développer des algorithmes neuromorphiques sur des smartphones. Enfin nous proposons de porter des méthodes basées sur la précision temporelle dans les réseaux de neurones impulsionnels sur du matériel neuromorphique. Avec cette tentative, nous apportons une technologie qui commence à ressembler à la contrepartie organique, pour montrer les capacités de l'informatique inspirée du cerveau.

Mots clés : calcul neuromorphique, calcul évènementiel, vision par ordinateur à faible puissance, matériel neuromorphique, efficacité énergétique, réseaux de neurones impulsionnels

Acknowledgments

I would like to thank the many people who have helped this thesis to see the light of the day. I would like to express my gratitude:

To Ryad for showing me how to walk off the beaten path and the fact that he give me a chance to prove myself.

To Sio for making sure that I do things the right way.

To Serge for helping me out in a difficult situation.

To Alexandre for his way to explain things in detail, stimulating discussions and writing great software that made a lot of my work possible.

To Lena, who is amazing at what she does and incredibly humble at the same time. I can learn a lot from you.

To Gerhard and Vera who have always supported me, no matter what path I chose.

To Ozan, Jose, Carlos, Jorge and Jonathan, from whom I learned so much over the years. I'm lucky to be able to call you my friends and I cannot wait for the next time we meet!

To *les loulous*, with whom I enjoyed exploring the beautiful city of Paris and its hidden corners!

To Marco and Dounia, with whom I had really good times in- and outside the lab. I miss you!

To Clemi for an amazing friendship and bond. Many more years to come!

To my catamaran sailing gym buddy, Iftar host and good friend Omar, who helped me keep my sanity and without whom I probably couldn't have completed this manuscript.

To Ira Bunny, who has supported me with all her heart, whom I admire deeply and whose love I cherish.

List of contributions

Journals

- **Lenz G**, Ieng SH and Benosman R. *High Speed Event-based Face Detection and Tracking Using the Dynamics of Eye Blinks*, Frontiers of Neuroscience 2020 [1].
- **Lenz G**, Oubari O, Orchard G and Ieng SH. *Neural Computation Using Precise Timing on Loihi*, in preparation 2021.
- **Lenz G** and Ieng SH. *A Framework for Event-based Computer Vision on a Mobile Device*, in preparation 2021.
- Oubari O, Exarchakis G, **Lenz G**, Benosman R and Ieng SH. *Efficient Spatio-temporal Feature Clustering for Large Event-based Datasets*, submitted 2021.

Conferences

- Maro JM, **Lenz G**, Reeves C and Benosman R. *Event-based Visual Gesture Recognition with Background Suppression running on a smart-phone*, 14th ICAG 2019 [2].
- Haessig G, Lesta DG, **Lenz G**, Benosman R and Dudek P. *A Mixed-Signal Spatio-Temporal Signal Classifier for On-Sensor Spike Sorting*, ISCAS 2020 [3].

Awards

- 14th IEEE International Conference on Automatic Face & Gesture Recognition Best Demo Award, 2019.

Open source software

- **Frog**: An Android framework for event-based vision.
- **Loris**: Python library to handle files from neuromorphic cameras.
- **Tonic**: Event-based datasets and transformations based on PyTorch.
- **Quartz**: ANN to SNN conversion using temporal coding.

Contents

Acronyms

AI Artificial Intelligence. 2, 4, 5, 17, 18, 20

ANN Artificial Neural Network. 12, 14–16, 58, 70–72, 77, 78, 81, 88, 96, 97

ATIS Asynchronous Time-based Image Sensor. 10, 13, 24, 25, 41–43, 91, 95

CF Correlation Filter. 32, 36, 37

CNN Convolutional Neural Network. 10, 32, 96

CPU Central Processing Unit. 3, 5, 12, 17, 32, 38, 45, 48, 55, 79, 80, 86

DAVIS Dynamic and Active pixel Vision Sensor. 10, 12

DVS Dynamic Vision Sensor. 10, 13, 24, 42, 91

EDP Energy Delay Product. 78–80, 82, 88

FPGA Field-Programmable Gate Array. 12, 42, 45, 78

fps frames per second. 32, 38

FRCNN Faster Region Based Convolutional Neural Network. 32, 36–38

GPU Graphics Processing Unit. 2, 4, 10, 12, 15, 17, 18, 25, 38, 55–57, 77, 80, 83, 85, 86, 88, 90, 96

HATS Histogram of Averaged Time Surfaces. 38

HOTS Hierarchy of Time Surfaces. 12, 38, 51, 52

IoT Internet of Things. 10, 18, 89

MIPI Mobile Industry Processor Interface. 56, 95

NDK Native Development Kit. 41, 46, 52, 54, 55

NEF Neural Engineering Framework. 73, 80, 86

RISC Reduced Instruction Set Computer. 17, 39, 89, 90

RNN Recurrent Neural Network. 15, 20, 88

SNN Spiking Neural Network. 13–16, 18, 21, 57, 58, 70–72, 77–79, 81, 82, 86–88, 96

SoC System on Chip. 17

SSD Single Shot Detector. 32, 36–38

STDP Spike-Time Dependent Plasticity. 15, 16

STICK Spike Time Computation Kernel. 57, 59–62, 70, 72, 74, 77, 79, 80, 82, 87

TPU Tensor Processing Unit. 18, 25

TTFS Time To First Spike. 71, 72, 78, 81, 88

USB Universal Serial Bus. 10, 41–48, 56, 85, 95

VJ Viola-Jones. 32, 36–38

List of Figures

List of Tables

Chapter 1

Introduction

How much of a machine is human? And how much of a human is a machine? This question will become somewhat more important and at the same time more difficult to answer in the future as the dividing lines will gradually get blurrier [4]. Modern technology is a powerful tool that provides humanity with the means to transform cognitive abilities into skills and machines. Our biological bodies will inevitably merge with this technology to extend and offload capabilities. It will feel very natural, as new generations raised in the information age experience the extraordinary benefits and looming drawbacks of being online on-demand and in an instant. We are not far now from the point where we will have a direct, physical connection to a mobile system such as a brain machine interface [5, 6]. To be implanted such an interface will become as much of a routine treatment as getting dental braces or replacing a hip joint. To make the connection between human body and human-made technology as natural as possible, one can imagine that a system that processes information much like a biological organism does, is much easier to interface with [7, 8].

From a certain standpoint, humans can already be considered cyborgs [9, 10], defined as beings with both organic and artificial body parts. Examples of such parts are pacemakers, prostheses or neurostimulators. We might not have a physical connection to our beloved handheld devices such as phones and tablets, but it would not be an overstatement to say that we are at least psychologically attached to them. One could even go as far as to claim that we are overly dependent on them [11]. It's specifically the *smart* ones, those that can tell us jokes and enable us to connect to anyone and anything on the internet around the world in a matter of seconds. Mobile handheld devices have truly taken the world by storm since the year the smartphone took off in 2008. With the iPhone and Android fresh into the market back then, 17% of people in the Western world owned a smartphone at that time. Since then this number has risen to 93% across adults in the Western world [12]. In developing countries, mobile communication has largely leapfrogged wired telephone lines and conventional banking all together [13, 14]. Our smartphones or tablets are now embedded in our daily lives and act as a central hub to an even larger array of connected devices, such as smart speakers, watches or cameras.

As those devices become somewhat more intelligent and human-like, Artificial Intelligence (AI)-driven features become critical differentiating factors for a saturated market of smart technology. Voice or automated driving assistants add an undeniable surplus value to existing systems, and no company that produces consumer goods can afford to ignore this trend. The training of modern AI models consumes enormous amounts of energy, and these energy requirements are growing at a breathtaking rate. In the deep learning era, the computational resources needed to produce a best-in-class AI model has on average doubled every 3.4 months [15] as illustrated in Figure 1.1. As a result these models are costly to train and develop, both financially, due to the cost of hardware and electricity or cloud compute time, and environmentally, due to the carbon footprint required to fuel modern tensor processing hardware [16]. Generative Pre-trained Transformer-3, the latest language model by OpenAI to produce human-like text, consists of 175 billion parameters, more than a 100 times more than the previous year's biggest model [17]. It took an estimated 355 Graphics Processing Unit (GPU)-years, $4.6m and 1 GWh of energy to train it. We see diminishing returns from scaling up machine learning models with a breathtaking rate.

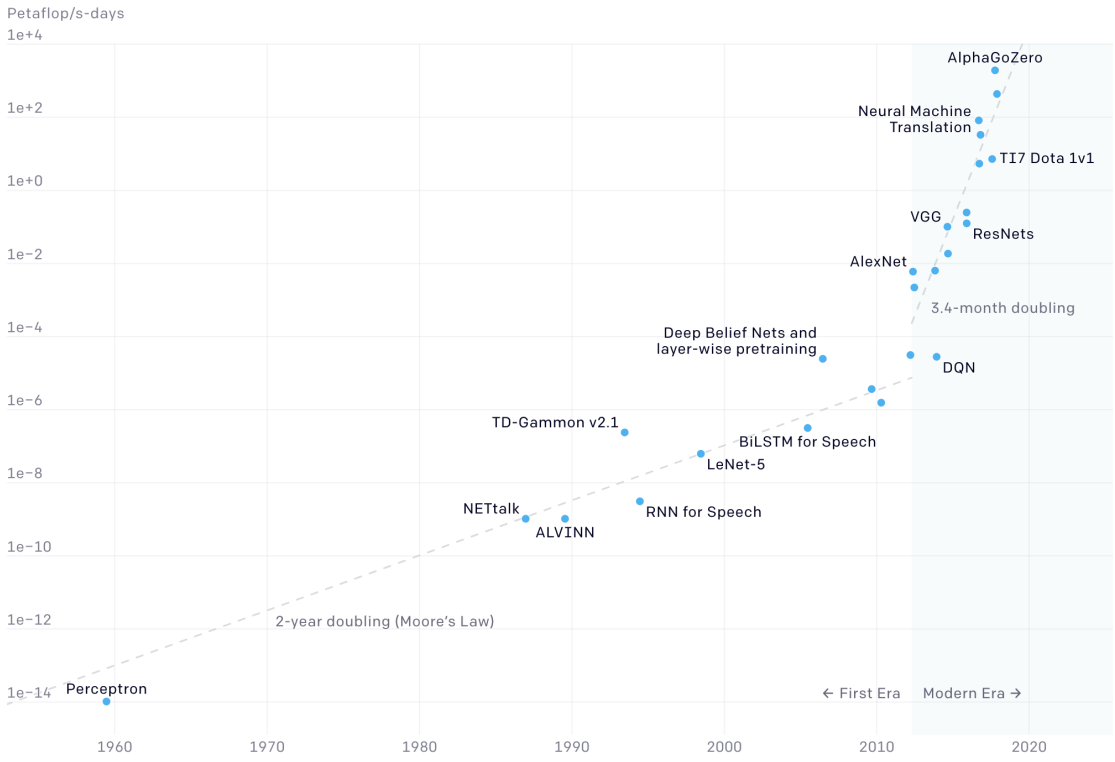

Fig. 1.1 *Number of total operations needed to train some of the most well-known models in computer vision, natural language processing and reinforcement learning. Image taken from OpenAI's blog post [15].*

This stands in contrast to battery density that has been improving at an averaged rate of 7.5 percent a year between 2011 to 2017 [18]. While the daily usage of smart mobile devices especially among younger adults has steadily increased [19], the time a device can be used before it needs recharging has stagnated. Over time, more and more features have merged into our mobile devices, which is very costly in terms of computation. Therefore a lot of *smart* functionality is offloaded to be computed in the cloud, on a remote server. But the continuous exchange of private data over the network raises privacy concerns for end users. It also means that many devices stop being *smart* when problems with server availability or Wi-Fi connections arise. For the smart functionality that does not require a network connection, we have relied on the growth of numbers of transistors per unit area to satisfy the growing demand in computing power. Nowadays we are encountering the physical limits of this scaling process [20], with fabrication processes constructing transistors as little as 5 nm apart [21, 22]. Not only does processing logic reach physical speed limits, but already today Central Processing Unit (CPU)s spend a lot of their time waiting for new data to be fetched from memory, which is an issue that is only going to become more problematic with growing amounts of data to be processed. In response to that, industry has largely changed to horizontal scaling such as increasing the number of cores among other mitigation tactics in order to guarantee performance improvements. But the issue remains one around bandwidth and the sequential nature of reading instructions from memory, processing them and storing the result.

The race is on to explore new computing architectures and paradigms, which seem more promising than ever before. The field of neuromorphic engineering is one alternative route, exploring biological concepts of information processing in order to imitate them on a hardware level. It takes inspiration from neuroscience, machine learning and electrical engineering to build hardware that computes using silicon neurons [23, 24]. The guiding philosophy is not to copy the *wetware* such as our brain in complete detail, but to search for organising principles that can be applied in practical devices [25]. The essential components are artificial neurons, which emit short electrical pulses of action-potentials called spikes to other neurons via synaptic connections. Even so, building and connecting large numbers of artificial neurons on its own is not enough. With a new kind of hardware comes the need for new algorithms, which handle spikes from neurons in an asynchronous fashion. By doing that, the hope is to find a more efficient way to represent information and to compute. This parallel track to classic computing will not replace clocked, synchronous, high-throughput computation anytime soon. Rather it should be seen as catering to a growing demand for efficient, fault-tolerant, low-power computation. This demand is especially pressing on mobile systems, where specialised chips can naturally co-exist with current systems on a single device, devices that have an increasing gamut of functionalities that we happily rely on.

1.1 Motivation and Objectives

The current success story of deep learning and AI is powered by the interplay of data, algorithms and dedicated hardware. This hardware is based on the same computing architecture since the inception of the modern computer that separates processing unit from memory. Given that we stand just at the beginning of an age of AI, we have to ask the question how current trends in machine learning model sizes can continue to scale. Moore's law has been surpassed by Huang's law, named after NVIDIA's chief executive officer Jensen Huang, which predicts that the performance of GPUs will more than double every two years [26]. For battery-powered devices that rely on a lot of dedicated hardware to be efficient, advancements in powerhouse technology alone will not cut it. As technology powered by machine learning enters cars, watches, tablets and other mobile systems, power consumption in such environments will be critical to the success of those systems.

The goal of this thesis is to find ways that are inspired by biology to compute more efficiently than current computer systems. Neuromorphic engineering, taking inspiration from biological systems, rethinks computation from the ground up, as our brains do not separate memory from processing but combine the two principles in each and every neuron. We could build an end-to-end neuromorphic pipeline that exists in parallel to existing systems, but due to expensive sensors and processing hardware that is still in a research stage, this is not a straightforward feat. Alternatively we can replace parts of the conventional machine learning pipeline and examine how neuromorphic equivalents can contribute to saving power. These replacements are neuromorphic cameras, algorithms and hardware.

Neuromorphic cameras use a novel sampling theorem to imitate the asynchronous firing pattern of the mammal retina to avoid capturing redundant information and therefore save energy. This class of vision sensors has promising applications in machine vision that need to record a visual scene as efficiently as possible, while still exhibiting superior dynamic range and temporal resolution. We raise the question to what extent we can make use of event camera properties in neuromorphic algorithms to compute more efficiently than conventional architectures that use image-capturing cameras.

Mobile devices with their optimised processing hardware should be able to profit directly from such efficient sensors and algorithms. They already integrate a growing amount of sensors for specific tasks and specialised hardware that takes up an increasing share of silicon area. Neuromorphic sensors and algorithms can be added to this mix to help reduce power consumption further. The integration should be seamless to ensure adoption for devices that are ubiquitous in our lives.

The use of low-power conventional hardware as used in mobile devices can be seen as

intermediate step, but neuromorphic computing will eventually make use of dedicated hardware to unlock its full potential. Much like conventional neural networks do not seem as powerful when executed on a CPU, bio-inspired algorithms will benefit from hardware that boasts artificial neurons. The recent past has shown that the co-design of algorithms and hardware is more important than ever.

We explore algorithms that use asynchronous, event-based computation on both conventional hardware that is widely available today and on neuromorphic hardware that is emerging. Neuromorphic computing has the potential to follow a similar success story as deep learning and outperform current AI and machine learning architectures when it comes to power efficiency. Artificial general intelligence using 20 Watts is the ultimate goal to be as efficient as our brain, but until then it is still a long way to go. The advancements in neuromorphic technology will at the very least help devices to use less power and hopefully make technology function more human-like. Starting from the basic elements of neurons, the goal is to facilitate the successful merging of human and machine.

In the remainder of our introduction, we provide an overview about components of a neuromorphic system that can gradually replace and complement current technology. The combination of new sensors, algorithms and hardware will help to enable applications that are inherently low-power and might help us find new ways of biologically plausible learning.

1.2 Rethinking the Way our Cameras See

We want machines to be able to see like us, and in that effort have created cameras. The field of modern computer vision is based on the common output format of those sensors: frames. However, the way we humans perceive the world with our eyes is very different. Most importantly, we do it with a fraction of the energy needed by a conventional camera [27]. The field of neuromorphic vision tries to understand how our visual system processes information, in order to give modern cameras that same efficiency and it looks like a substantial shift in technology for machine vision.

We are so focused on working with data that modern cameras provide, that little thought is given about how to capture a scene more efficiently in the first place. Current cameras acquire frames by reading the brightness value of all pixels at the same time at a fixed time interval, the frame rate, regardless of whether the recorded information has actually changed. A single frame acts as a photo; as soon as we stack multiple of them per second it becomes a motion picture. This synchronous mechanism makes acquisition and processing predictable. But it comes with a price, namely the recording of redundant data. And not too little of it. As shown in Figure 1.2, redundant information about the background is

captured even though it does not change from frame to frame, when at the same time, high velocity scene activity results in motion blur.

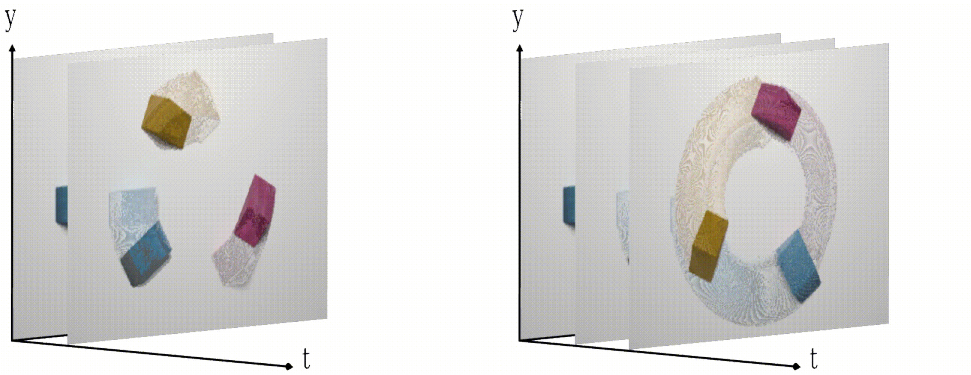

Fig. 1.2 *Image blur can occur in a frame depending on the exposure time.*

1.2.1 Taking Inspiration from the Human Visual System

The human retina has evolved to encode information extremely efficiently. Narrowing down the stimuli of about 125 million light sensitive photoreceptors to just 1 million ganglion cells which relay information to the rest of the brain, the retina compresses a visual scene into its most essential parts. Photoreceptor outputs are bundled into receptive fields of different sizes for each retinal ganglion cell as shown in Figure 1.3. The way a receptive field in the retina is organised into center and surround allows ganglion cells to transmit information about spatial contrast, encoded as the differences of firing rates of cells in the center and surround. Retinal ganglion cells are furthermore capable of firing independently of each other, thus decoupling the activity of receptive fields from each other. Even if not triggered by external stimulus, a retinal ganglion cell will have a spontaneous firing rate, resulting in millions of spikes per second that travel along the optic nerve. It is thought that in order to prevent the retinal image from fading and thus be able to see the non-moving objects, our eyes perform unintentional rapid jumps called micro-saccades. This movement only happens once or twice per second, so in between micro-saccades, our vision system probably relies on motion. To put it in a nutshell, our retina acts as a pre-processor for our visual system, extracting contrast as an important stream of information that then travels along the optical nerve to the visual cortex. In the cortex it is processed for higher-level conscious processing of the visual scene.

Inspired by the efficiency and complexity of the human visual system, Misha Mahowald developed a new artificial stereo vision system in the late 80s [28]. She was one of Carver Mead's students, a scientist at Caltech who spawned the field of Neuromorphic Engineering at that time. In his lab, Misha built what would become the first silicon retina in the

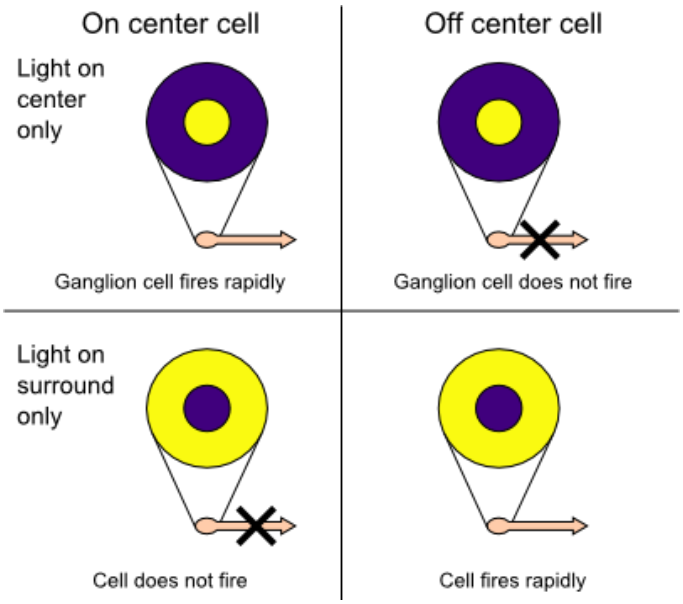

Fig. 1.3 *Center-surround receptive fields in the mammalian retina.*

early 90s [29]. It was based on the same principle of center-surround receptive fields in the human retina, which emit spikes independently of each other depending on the contrast pattern observed.

Although Misha drafted the beginning of a new imaging sensor, the design did not provide a practical implementation at first. In response, the neuromorphic community simplified the problem by dropping the principle of center-surround pixels [30]. Instead of encoding spatial contrast across multiple pixels which needed sophisticated circuits, the problem could be alleviated by realising a circuit that could encode *temporal* contrast for single pixels. That way, pixels could still operate individually as processing units just as receptive fields in the retina do and report any deviations in illuminance over time. While the first silicon retinas where fully analog [31, 32], it would take until 2008 when the first refined temporal contrast sensors was published based on digital architecture [33], the event cameras as they are known today.

1.2.2 A Paradigm Shift in Signal Acquisition

The new architecture led to a paradigm shift in signal acquisition, illustrated in Figure 1.4. Standard cameras capture absolute illuminance at the same time for all pixels driven by a clock and encoded as frames. One fundamental approach to dealing with temporal redundancy in classical videos is frame difference encoding. This simplest form of video compression includes transmitting only pixel values that exceed a defined intensity change threshold from frame to frame after an initial key-frame. Frame differencing is naturally

performed in post-processing, when the data has already been recorded. Trying to take inspiration from the way our eyes encode information, event cameras capture changes in illuminance over time for individual pixels corresponding to one retinal ganglion cell and its receptive field.

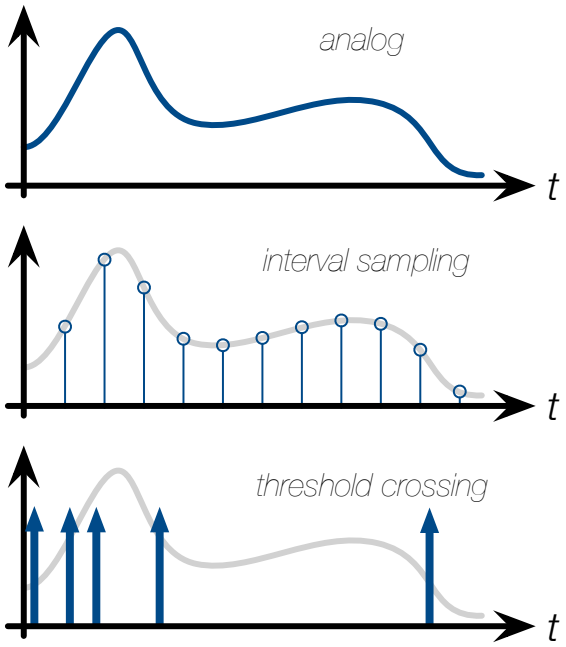

Fig. 1.4 *Different sampling theorems. The 'real world' is a collection of analog signals, which in order to store and digitise it we transform into numbers. Digital signal acquisition relies on regular sampling along the time axis. An alternative approach is level or threshold crossing, where the signal is sampled whenever it surpasses a threshold on the y-axis.*

If light increases or decreases by a certain percentage, one pixel will trigger what's called an event, which is the technical equivalent of a cell's action potential. One event will contain information about a timestamp, x/y coordinates and a polarity depending on the sign of the change. Pixels can trigger completely independently of each other, resulting in an overall event rate that is directly driven by the activity of the scene. It also means that if nothing moves in front of a static event camera, no new information is available hence no pixels fire apart from some noise. The absence of accurate measurements of absolute lighting information is a direct result of recording change information. This information can be refreshed by moving the event camera itself, much like a saccade.

Because of the considerable size of the circuit that enables temporal contrast for each pixel, it didn't leave much room for the photo diode to capture incoming photons. The ratio of a pixel's light sensitive area versus the total area is called fill factor and amounted

to 9.4% for the first event camera [33]. Modern CMOS (Complementary Metal Oxide Semiconductor) technology will enable a fill factor of above 90% at a fabrication process of 180 nm. With a reduced fill factor the photon yield will be low, which will in turn drive image noise. This was thus a major obstacle for event camera mass production early on. Nevertheless already this first camera was able to record contrast changes under moonlight conditions. New generations of event cameras use backside illumination in order to decouple the processing circuit for each pixel from the photo diode, by flipping the silicon wafer during manufacturing [34]. Most of today's smartphone cameras already use backside illumination in order to maximise illumination yield at the expense of fabrication cost.

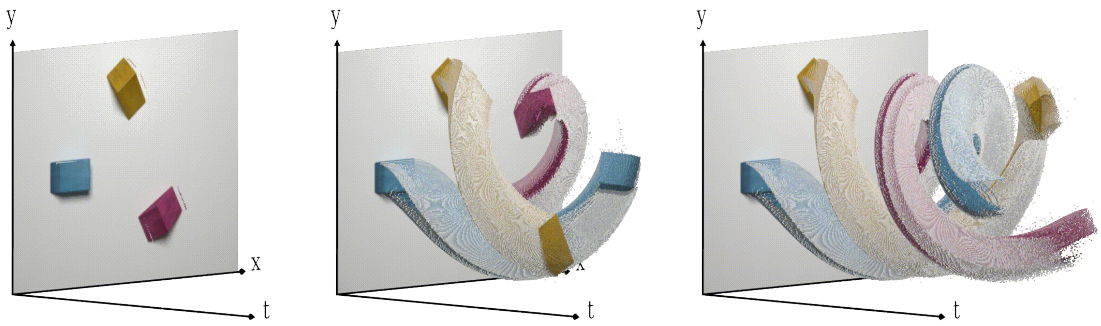

Fig. 1.5 *An event-camera will only record change in brightness and encode it as events in x, y and time. Colour is artificial in this visualisation. Note the fine-grained resolution on the t-axis in comparison with the frame animation in Figure 1.2. Thanks to Alexandre Marcireau for the data. Visualisation has been created using Rainmaker*.*

1.2.3 A Novel Sensor for Machine Vision

Overall an event camera has three major advantages compared to conventional cameras: since pixel exposure times are decoupled of each other, very bright and very dark parts can be captured at the same time, resulting in a dynamic range of up to 125dB. The decoupled, asynchronous nature furthermore frees bandwidth so that changes for one pixel can be recorded at a temporal resolution and latency of microseconds. This makes it possible to track objects with very high speed and without blur as exemplified in Figure 1.5. The third advantage is low power consumption due to the sparse output of events, which makes the camera suitable for mobile and embedded applications. As long as nothing in front of the camera moves, no redundant data is recorded by the sensor which reduces computational load overall. It also relieves the need for huge raw data files. Current drawbacks for most commercially event cameras available today are actually further downstream, namely

*https://github.com/neuromorphic-paris/command_line_tools

the lack of hardware and algorithms that properly exploit the sparse nature of an event camera's data. Rethinking even the most basic computer vision algorithms without frames takes a considerable effort.

Over the years, event cameras have seen drastic improvements in spatial resolution and signal to noise ratio. The main generations of cameras are Dynamic Vision Sensor (DVS) [33], Asynchronous Time-based Image Sensor (ATIS) [35] and the Dynamic and Active pixel Vision Sensor (DAVIS) [36]. Examples of companies that produce commercially available event cameras are Samsung [37], Prophesee [38], Celepixel and Insigthness [36]. Most commercially available event cameras are still large in size, but small form factor version have been developed too. Event cameras for mobile applications include a small embedded DVS system [39] and a small ATIS which can be connected via mini-Universal Serial Bus (USB) [40], which is explained in more detail in Chapter 3. The first commercially available single-chip neuromorphic vision system for mobile and Internet of Things (IoT) applications is called Speck[†], which combines a DVS and the Dynap-se neuromorphic Convolutional Neural Network (CNN) processor. The rise of the event camera has been relatively slow, as larger gains in power efficiency are being made by focusing on the processing of image data further downstream, notably on a GPU. This trend however is also likely to saturate at some point and will make it worth to further explore and employ this novel image sensor [41].

1.3 Event-based Computer Vision and Applications

1.3.1 A Temporal Component to Understand Visual Input

In 1975, Swedish perceptual psychologist Gunnar Johannson writes:
The eye is often compared to the camera, but there is one enormous difference between the two. In all ordinary cameras a shutter 'freezes' the image [...]. In all animals, however, the eye operates without a shutter. Why, then, is the world we see through our eyes not a complete blur? [42]

Animals integrate visual information over time into a continuous, conscious stream. Johannson proposed that an object can be recognised purely by its motion, based on the idea of continuous input [43]. Figure 1.6 shows one of his examples and more can be found online[‡]. Humans can indeed easily recognise objects that are represented by several simple dots moving, suggesting that timing is important for our visual system. At the same time, research has also shown that it is harder for humans to correctly identify a point-light walker when it is positioned upside down, suggesting that a spatial component is in fact

[†] https://www.speck.ai/
[‡] A video of a Johannson experiment can be found under https://youtu.be/rEVB6kW9p6k

also necessary to detect an object [44].

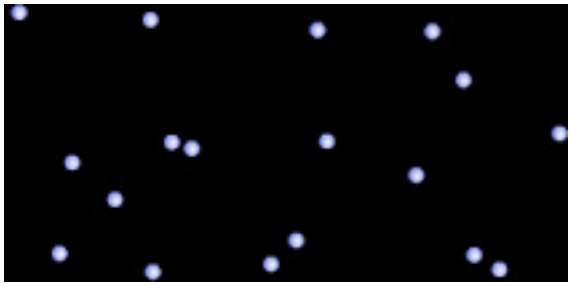

Fig. 1.6 *The points themselves make it hard to identify the object behind it. Can you guess what it is once the points start to move? Check* https://tinyurl.com/y3yehcur

1.3.2 The Era of Deep Learning

The success of deep learning using neural networks has proven triumphantly that a computer can recognise objects at a superhuman level by analysing purely spatial compositions (such as an image). To understand more subtle concepts such as actions, intents or emotions in an effort to be more *intelligent*, there has to happen some form of aggregation of information over time. And to make that possible on an embedded system, computation using frames that carry a lot of redundant information seems counter-intuitive. Recent research focuses on making neural networks more efficient by using techniques such as pruning [45, 46], quantization [47, 48, 49, 50], knowledge distilling [51] or finding functionally similar but smaller sub-networks [52]. Taking quantization to the extreme leads to binary weights, which significantly reduces model size and inference time [53, 54]. Neural networks have successfully been optimised to use fewer multiplier-accumulator operations and parameters by designing novel network architectures that exploit computation or memory efficient operations such as depthwise separable convolutions to fit on mobile devices [55, 56, 57, 58]. The most prominent examples of this endeavour are the MobileNet architectures, which is a family of computer vision neural network models designed to maximise accuracy while being mindful of the restricted resources for an on-device or embedded application [59, 60, 61].

These measures take effect late in the pipeline, still recording and processing redundant data. If we work with highly sparse data such as from event cameras in order to save power, we need a form of temporal component in our models.

1.3.3 Event-based Processing

The field of event-based computer vision has grown rapidly over the last years, as event cameras have the potential to displace standard cameras in wearable technology, robotics and mobile applications. Applications that are currently available specifically on mobile systems include drowsiness driving detection systems [62], proximity sensing for handheld devices [63], motion detection [64] or gesture recognition as described in Chapter 3. Mobile autonomous robots can learn to cooperate [65] and drones learn autonomous flight [66, 67, 68].

Rooted in classical computer vision, a lot of work focuses on accumulating events into *bins* of fixed or variable length depending on scene activity [69], thus artificially creating frames from a sparse output signal. This group of algorithms leverages existing advances, most notably analog artificial neural networks to use them for optical flow [70, 71], depth prediction [72], high dynamic range image reconstruction from events [73, 74, 75], or Simultaneous Localisation and Mapping (SLAM) [76, 66]. Other applications include image deblurring [77], star tracking [78] or object segmentation [79, 80].

This approach also allows the use of GPUs when spatially sparse frames or volumes are processed using Artificial Neural Network (ANN)s, which results in processing a lot of redundant information because of the high temporal data precision. Aiming to increase the speed and power efficiency with which inference can be done, certain computations can be skipped using dedicated hardware [81, 82, 83, 84].

Another approach tries to get the best of both worlds, frames and events, by mixing them together, which works well with hybrid sensors such as the DAVIS but is computationally very demanding [86, 69, 87]. Methods that can possibly achieve the lowest latency and power consumption work on an event-by-event basis. This approach advocates short, incremental calculations triggered by each event, and requires rethinking computer vision algorithms from the ground up. The downside of this method is that they cannot make use of dedicated hardware such as GPUs at the moment, being restricted to highly parallel CPUs in most cases. There are exemptions of Field-Programmable Gate Array (FPGA) implementations to speed up the low-latency processing [88, 89, 90, 91]. Exemplary applications of event-by-event methods include event stream classification [85, 92], optical flow [93, 94, 95, 96], corner detection [97, 98, 99], pose estimation [100] and tracking [101]. It lead to the emergence of new features called time surfaces [85, 78, 102], which are spatio-temporal features generated for each event as shown in Figure 1.7. They resemble local patches of temporal gradients and have been employed in bag-of-features methods such as Hierarchy of Time Surfaces (HOTS) [85], which creates a hierarchy of time surfaces with different time constants. Other bag-of-features methods build on time surfaces as

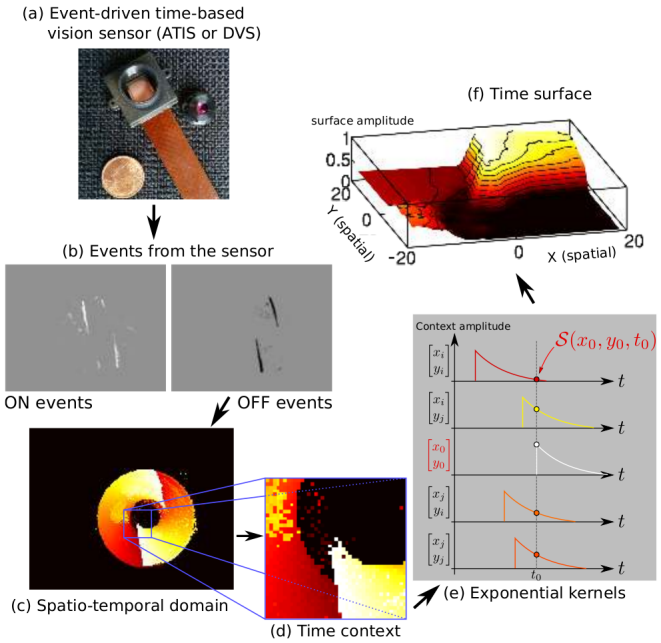

Fig. 1.7 *Time surface generation from the spatio-temporal context of events. A time surface resembles a local image patch that is generated for each event. (a,b) An event camera such as the ATIS or DVS records motion in a visual scene, represented by ON and OFF events over time. (c) If we represent the accumulated events in one image, brighter pixels signify more recent events. (d) focusing on a spatial region of interest we generate a local time surface by applying an exponential kernel (e) to the timing of events in the neighbourhood. Image taken from Lagorce et al. [85].*

well [103, 92].

1.4 Spiking Neural Networks

1.4.1 Sparse Data Representations

A more biologically-inspired possibility is to turn to a Spiking Neural Network (SNN) for learning tasks. Labelled as the 3rd generation neural network archetype [104], data is represented in binary form, where a neuron can either spike or not. Every neuron has an internal state such as membrane potential, threshold and decay times. Neurons in an SNN do not fire automatically when new input is presented, but only when enough spikes per time unit accumulate so as to push the membrane potential across its threshold. The spike emitted will then be propagated forward, subject to synapse weights and delays. The sparse nature of communication offers the potential to encode and transmit information

in a significantly more energy efficient manner.

Figure 1.8 shows the principal difference between a neuron unit in an ANN and in an SNN. Whereas the input for an ANN is typically a tensor with high data precision, but low temporal resolution, the input for an SNN are binary flags of spikes with comparatively high temporal precision in the order of µs. The unit in the SNN integrates all of the incoming spikes, which affect the internal parameters such as membrane potential. The unit in the ANN merely computes the linear combination for inputs on all synapses and outputs that. Although there are ANN units with memory characteristics and internal states such as recurrent or long short-term memory units [105, 106], they typically operate on much wider time frames such as a few words in a sentence or a video frame that is captured every 25 ms.

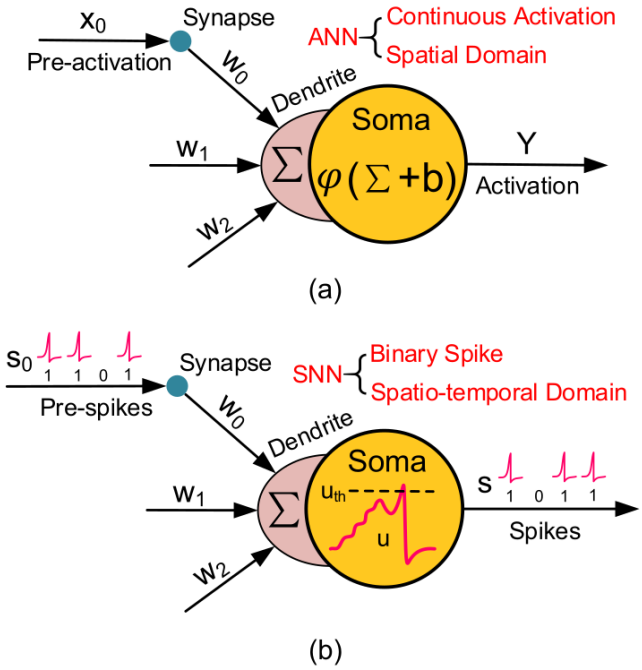

Fig. 1.8 *Basic neuron model in (a) ANNs and (b) SNNs. Picture taken from Deng et al. [107].*

The development in SNNs has focused to a great extent on vision tasks such as image classification or object detection, largely driven by the need to compare new work to existing classical architectures. However, SNNs are likely not going to outperform ANNs in every aspect, but rather fill a niche. What this niche is, is an interesting research question at the moment. Some groups have developed spiking sorting algorithms [108], or spike time encoded addressable memory [109]. Certainly the ability for near-sensor feature extraction, ultra-low power neural network inference, local continual learning [110]

or constraint satisfaction problems [111, 112] are tasks where SNNs can already excel. The stateful, recurrent architecture of Recurrent Neural Network (RNN) also seems suitable to be mapped to SNNs [113, 114].

There are other areas of artificial intelligence that are little explored when it comes to employing SNNs, such as in reinforcement learning [115] or attention-based models. Deep reinforcement learning uses deep learning to model complex value functions for continuous high-dimensional state spaces that allows an agent to perform actions even though while training it only encountered a small subset of states during trial and error learning [116, 117]. Deep reinforcement learning suffers from high sensitivity to noisy, incomplete, and misleading input data and SNNs with their inherent stochastic nature could provide some robustness to that [115]. In the same vein, ANNs are notoriously sensitive to malevolent adversarial attacks. Sharmin et al. demonstrate that SNNs tend to show more resiliency compared to ANN under black box attack scenario, which could help deploy them in real-world scenarios [118].

So far SNNs have not proven that they perform better in general. The rise of attention-based deep neural network architectures called transformers [119, 120] make it clear that time and recurrent architectures are not a necessity when computing on sequences. They allow for parallel training on multiple tokens at the same time but need lots of parameters. Transformers are causing a stir in deep learning and are being used not only in natural language processing but also vision and audio tasks. However, they work with highly dense training data such as images and regularly sampled audio files. A crucial point that neuromorphic computing relies on is sparsity. This is, after all, the strength of event-based sensing and the principle of threshold crossing. No change in the input signal means no data recorded. Nevertheless there are different training methods how an SNN can extract features from input data.

1.4.2 Training Spiking Neural Networks

Training SNNs follows one of 3 major pathways: converting the weights of pre-trained ANN [121, 122], supervised learning using backpropagation with spikes [123, 124, 125, 126] or local learning rules based on Spike-Time Dependent Plasticity (STDP) [127, 128] or local errors [129]. The most straightforward path to create an SNN is to convert an ANN which had previously been trained on a GPU. The idea is to trade a small impact in performance for reduced latency and power efficiency. Continuous values are hereby transformed into rate-coded schemes [122]. Alternatively, the network can also be converted using a temporal coding scheme [130], which we explore in more detail in Chapter 4. Converted SNNs benefit from a large ecosystem available for GPU-based training of ANNs and certain training mechanisms such as batch normalisation [131] or dropout [132].

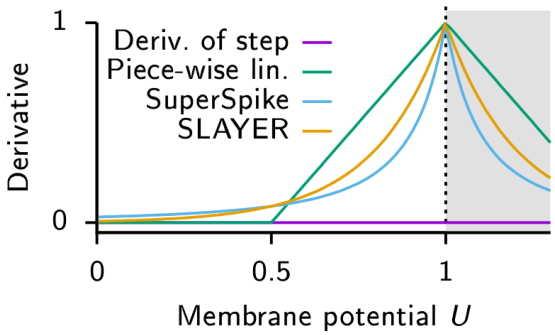

Fig. 1.9 *Commonly used derivatives as a replacement for spike activation to provide a differentiable signal when training spiking neural networks. The step function has zero derivative (violet) everywhere except at 0 where it is ill defined. Examples of replacement derivatives which have been used to train SNNs are in Green: Piece-wise linear [133, 134, 135]. Blue: Derivative of a fast sigmoid [136]. Orange: Exponential [124]. Figure taken from Neftci et al. [137].*

In order to facilitate learning in an SNN directly, we can apply methods from classical neural network training, such as backpropagation through time [138], to our SNN. Since the activation of a single spike, which resembles a Dirac impulse, is not differentiable, methods resort to smoothing spike activation itself [135, 136, 133, 137] as shown in Figure 1.9. A recent method has also adapted backpropagation to spikes without approximations [126]. Training methods using a global error signal achieve very good results, but are not very plausible to happen in the brain. SNNs that have been trained directly with backpropagation have yet to achieve the accuracy of converted SNNs when it comes to deeper networks, but the end-to-end training also speeds up the overall time needed for one network propagation and therefore reduces latency [139].

Local learning rules strive for biological plausibility without sacrificing performance too much. *DECOLLE* [110] and *e-prop* [114] are two recent examples of those algorithms that can both be implemented in neuromorphic hardware. Lastly, unsupervised feature extraction using local learning rules such as STDP [140, 141] relies purely on the timing between pre- and postsynaptic spike. It is biological plausible since it is without need for a global error signal, but has yet to reach ANN performance. The introduction of a third factor such as a global reward signal to complement the learning rule [142, 143] seems like a promising path forward. Overall, event-based vision promises efficient processing for naturally sparse inputs. An asynchronous network such as an SNN however needs asynchronous hardware to fully exploit its advantages.

1.5 Low-power Hardware for Mobile Systems

When power efficiency for a computing system is of utmost importance, dedicated hardware plays a crucial role. A System on Chip (SoC), soldered onto the mainboard of a mobile device, bundles multiple components into single chip to save space, cost and power consumption. It combines CPU, GPU and neural processing unit among other vital parts to act as integral computation unit of the device. ARM cores with their Reduced Instruction Set Computer (RISC) architecture achieve a much better performance per Watt ratio as opposed to x86 cores which use complex instruction sets and have become the *de facto* industry standard for CPUs in mobile phones and tablets. Apple announcing in 2020 that it is going to switch to ARM-based architecture for their latest laptop series products underlines the importance of low-power computer architecture in the mid-term future [144]. The SoC also includes dedicated graphics processors, as demand for high-fidelity, multi-user games has continuously increased over the years. Although GPUs have been diverted as AI accelerators on desktops, where power consumption does not play such an important role, mobile systems cannot afford the overhead of computing with double precision and off-chip memory. Neural processing unit, AI accelerator or Machine Learning processor are all terms that describe hardware which is specifically optimised for neural network operations. These processors are the newest class of dedicated silicon within a SoC and take up a growing percentage of the overall chip as shown in Figure 1.10. They make their way into mobile devices to enable biometric security features such as face or fingerprint unlocks, predictive text, voice assistants, content providers, system optimisation, navigation, health monitoring, intelligent cameras and more.

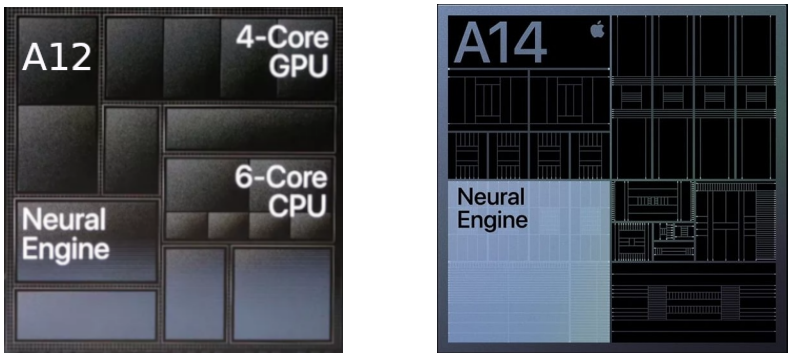

Fig. 1.10 *Two of Apple's SoCs, from 2018 on the left and 2020 on the right. The amount of silicon space dedicated to neural network accelerators is growing from year to year, up to a quarter in the latest version.*

Specifically photo and video capturing is an important feature for consumers, with phone manufacturers embedding no less than 4 cameras into their phones in 2020 [145]. Because

of the high power consumption involved in capturing and processing photos and videos in comparison to other embedded sensors [146], Vision Processing Units are yet another AI accelerator designed to improve performance of specific machine vision tasks that use embedded cameras. They are different from GPUs since they may include direct interfaces to the cameras, process using on-chip buffers and low precision fixed point arithmetic for image processing. Intel's Movidius Vision Processing Units [147] targets mobile devices, the IoT and the digital camera market. Qualcomm introduced the Darwin Neural Processing Unit already in 2015, which is a highly configurable neuromorphic hardware co-processor based on SNNs implemented with digital logic [148]. In the same year Google introduced the first generation of its Tensor Processing Unit (TPU), another neural network accelerator to speed up training and inference and has since made it available for third party use [149, 150]. In 2018 they announced the Edge TPU as part of their Coral line[§], which is a smaller and low-power version specifically for inference on power-constrained devices. An integrated version of the Edge TPU is already used in Google's own mobile phone series, the Google Pixel 4 [151].

1.5.1 Neuromorphic Hardware

Companies increasingly focus on a tight integration between hardware and software by designing the chips themselves to get a competitive advantage, as the race for more compute accelerates. Neuromorphic chips are posed to claim a share of the dedicated computing space for machine learning related tasks. A decisive factor will be whether hardware and algorithms together can exploit the sparsity inherent to some sensors, such as event cameras, to accomplish the same task using less energy. The hardware mimics the natural biological structures of a mammal nervous system, trying to imitate the power-efficient brain as a whole by rebuilding its basic components, the neurons, in silicon. This network of silicon neurons is the matching hardware to SNNs in the software domain. At the moment, neuromorphic chip design targets three main areas of research: 1. the exploration of new, asynchronous, bio-inspired algorithms for computation 2. helping neuroscientists understand the brain by being able to use billions of artificial neurons in scaled-up systems [152] 3. low-power applications more generally.

Neuromorphic hardware is either based on a fully digital design or alternatively brings analog components into the mix. Analog circuitry emulates the behaviour of neurons directly [153, 24], which means that a neuron's membrane potential and spike behaviour on chip can be followed on an oscilloscope if so desired. The routing of spikes however still uses digital circuitry and an asynchronous communication protocol [154]. Examples of this approach are CAVIAR [155], BrainScaleS [156], DYNAPs [157, 158] and Neurogrid [159].

[§] https://www.coral.ai/

Their major advantage is power efficiency by using transistors in a sub-threshold regime, and the operation in real-time independently of the model size or complexity. The drawback is a large silicon area per neuron, reducing the number of neurons per chip overall, as well as sensitivity to temperature, noise and *mismatch*, which is a term to summarise production variations of transistors across the silicon wafer. This class of neuromorphic hardware is used when low power is of utmost importance, but boundaries are pushed even further by exploring new materials to replace the relatively large analog circuits. A new electrical 2-terminal component called memristor [160, 161] can be used as an artificial synapse with adjustable weights [162, 163]. Memristors can be packed extremely densely in so-called crossbar arrays, to which the weights of a neural network can be mapped directly to perform in-place computation.

Digital chips abstract away some of the downsides of analog logic at the expense of power consumption. Examples of fully digital chips are SpiNNaker v1 and v2 [164, 165], TrueNorth [166] and Loihi [167]. They contain many processing cores distributed across the chip, where each core simulates a bundle of neurons and stores their membrane potential and variables related to learning in memory. Their advantage is deterministic neuron behaviour and fully-fledged learning capabilities. Fully digital architectures are used to help drive the exploration of novel spike-based algorithms, as they provide more reliable systems.

1.5.2 Hardware Benchmarking and Scalability

Research currently encounters a growing interest in benchmarking the power consumption of machine learning models [168] and making efficiency an evaluation criterion alongside accuracy as a related measure [169, 107]. Some preliminary work has explored the advantages of event-based sensing over classical frame-based methods in terms of power consumption during motion tracking [1], object tracking [170] or resilience to difficult lighting conditions [30]. Figure 1.11 shows the time to solution plotted over energy spent in comparison to Loihi for different tasks. This is an important task to carve out the areas where neuromorphic computing can excel and to guide future research. To put a price on energy used for machine learning models will also play a vital role in the effort to produce computing systems that are not completely detached from the limits of what biology can do.

An important question of hardware is the question of scalability. Neuromorphic hardware developed in research labs has seen a steady increase in amount of artificial neurons and learning capabilities over the past decade. Some large-scale neuromorphic systems that contain hundreds of chips in parallel now make available for computation an amount of artificial neurons that is comparable to the brain of a mouse [172, 173, 174]. Industry

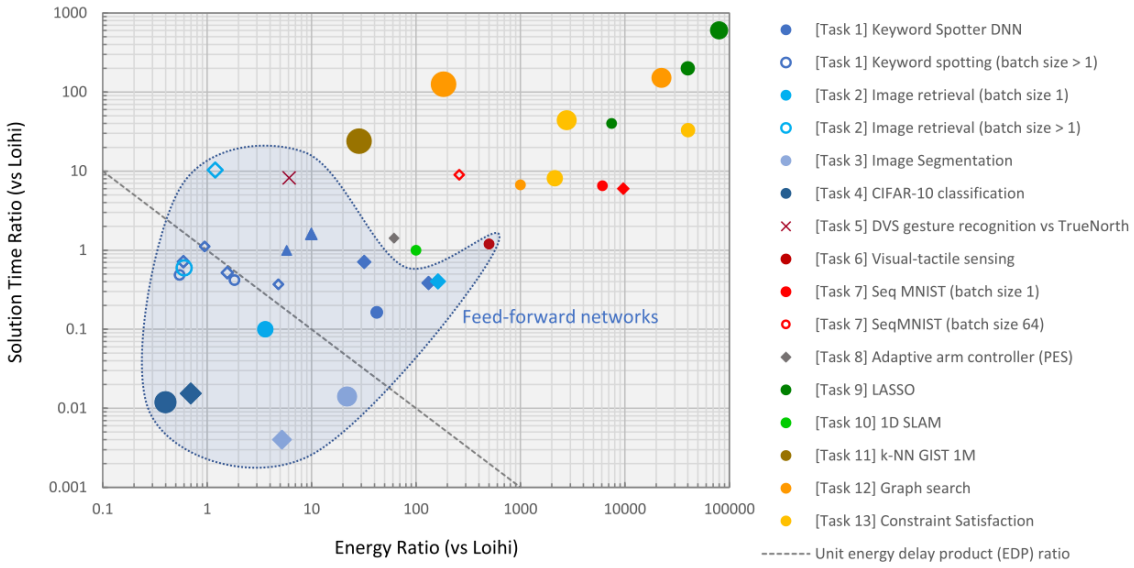

Fig. 1.11 *This plot compares the latest neuromorphic hardware against other architectures in terms of energy and latency for certain tasks. Whereas Loihi is really good at solving constraint satisfaction problems, large scale nearest neighbour search or RNN, the situation for feed forward networks is not so clear yet. Plot taken from Davies et al. [171].*

currently releases ultra-low power programmable chips for edge computing such as GrAI One [175] or the Akida Neural Processor [176]. As the spiking ecosystem evolves, it will define the area of where it will excel over and succumb to classical AI accelerators.

1.6 Thesis Outline

In this thesis we look at neuromorphic algorithms that are combined with both von Neumann and neuromorphic hardware to compute more efficiently than conventional systems.

In Chapter 2 we make use of event camera properties such as high temporal precision and robustness to different lighting conditions to explore the generation of spatio-temporal features that take into account fine-grained temporal signatures. We use the spatio-temporal signature of eye blinks that can be captured well with event cameras for event-based face detection and tracking. We do not rely on frame representations as as an alternative to conventional, frame-based methods. We show that when exploiting the sparse nature of the camera, we can use less power than gold-standard alternatives.

In Chapter 3 we turn to hardware that is optimised for battery-powered systems . Much

of the efficiency in mobile systems comes from several dedicated chips that are designed to execute specific tasks. Due to the use of conventional cameras however, such systems are not suited for always-on sensing. Always-on sensing is too costly when done using conventional cameras, especially for tasks that happen infrequently such as gesture recognition. This is unfortunate as it it deprives a growing population of elderly and visually impaired users of an intuitive interface. We present an Android framework that enables always-on sensing using an event camera, by avoiding computation in the absence of new visual information. Our framework connects the world of event-based computer vision to mobile devices that are powered by conventional hardware.

Neuromorphic computing is designed to work with artificial neurons and spikes. To leverage its full potential, we explore SNNs on neuromorphic hardware in Chapter 4. This hardware enables us to execute execute asynchronous algorithms efficiently. The chapter shines light on how it is possible to compute using the precise timing of spikes on neuromorphic hardware and how pre-trained networks can be ported for power-efficient inference on Loihi. This platform has superb support for power benchmarking, which plays a vital role in evaluating the strengths and weaknesses of spiking hardware over other specialised hardware.

Chapter 5 concludes our work and puts it into a bigger perspective. We draw parallels to the ascent of deep learning and how it is similarly backed by the development of dedicated hardware and give an outlook of developments in neuromorphic computing yet to come.

Chapter 2

Event-based Processing: Face Detection and Tracking

We start by looking at algorithms for event-based processing and introduce the first purely event-based method for face detection. It uses the high temporal resolution properties of an event-based camera to detect the presence of a face in a scene using eye blinks. Eye blinks are a unique and stable natural dynamic temporal signature of human faces across population that can be fully captured by event-based sensors. We show that eye blinks have a unique temporal signature over time that can be easily detected by correlating the acquired local activity with a generic temporal model of eye blinks that has been generated from a wide population of users. In a second stage once a face has been located it becomes possible to apply a probabilistic framework to track its spatial location for each incoming event while using eye blinks to correct for drift and tracking errors. Results are shown for several indoor and outdoor experiments as exemplified in Figure 2.1. We also released an annotated data set that can be used for future work on the topic.

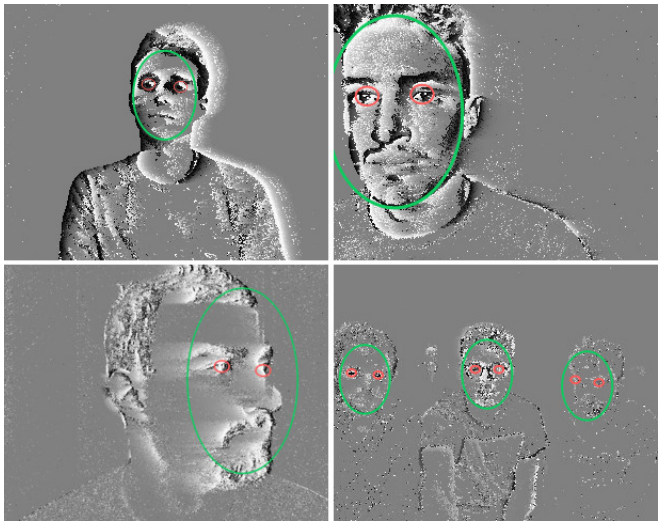

Fig. 2.1 *Event-based face tracking in different scenes. From left to right, top to bottom:* **a***) indoors* **b***) varying scale* **c***) with one eye occluded* **d***) multiple faces at the same time.*

2.1 Introduction

The method exploits the dynamic properties of human faces to detect, track and update multiple faces in an unknown scene. Although face detection and tracking are considered practically solved in classic computer vision, it is important to emphasise that current performances of conventional frame based techniques come at a high operating computational cost after days of training on large databases of static images. Event-based cameras record changes in illumination at high temporal resolutions (in the range of 1 µs to 1 ms) and are therefore able to acquire the dynamics of moving targets present in a scene [33]. In this work we will rely on eye blink detection to determine the presence of a face in a scene to in a second stage initialise the position of a Bayesian tracker. The permanent computation of eye blinks will allow to correct tracking drifts and reduce localisation errors over time. Blinks produce a unique space-time signature that is temporally stable across populations and can be reliably used to detect the position of eyes in an unknown scene. This work extends the sate-of-art by:

- Implementing a low-power human eye-blink detection that exploits the high temporal precision provided by event-based cameras.
- Tracking of multiple faces simultaneously at µs precision, once they have been detected.

The developed methodology is entirely event-based as every event output by the camera is processed into an incremental, non-redundant scheme rather than creating frames from events to recycle existing image-based methodology. We also show that the method is inherently robust to scale changes of faces by continuously inferring the scale from the distance of the two eyes of the tracked face from detected eye blinks. The method is compared to existing image-based face detection techniques [177, 178, 179, 180]. It is also tested on a range of scenarios to show its robustness in different conditions: indoor and outdoor scenes to test for the change in lighting conditions; a scenario with a face moving close and moving away to test for the change of scale, a setup of varying pose and finally a scenario where multiple faces are detected and tracked simultaneously. Comparisons with existing frame-based methods are also provided.

2.1.1 ATIS

In this work, we use the ATIS [35] event camera as it also provides events that encode absolute luminance information in an asynchronous manner. Apart from the regular change detection events known from of a DVS, this camera also sends a pair of spikes with an artificial polarity encoding for the grey-level information. The interval between the pair of grey-level spikes at the same pixel is indirectly proportional to the light intensity, meaning

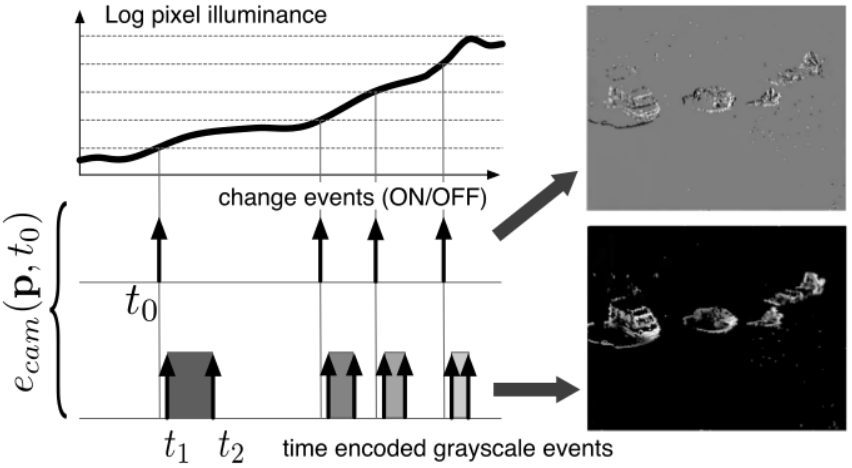

Fig. 2.2 *Working principle of an ATIS and two types of events. 1) change event of type ON is generated at t_0 as voltage generated by incoming light crosses a voltage threshold. 2) time $t_2 - t_1$ to receive a certain amount of light is converted into an absolute grey-level value, emitted at t_2 used for ground truth in this work.*

that areas of low exposure will have a long integration time. Grey-level information from an event camera allows for direct and easier comparisons with the frame-based world. To be able to handle the many different file formats available when using event cameras including the ATIS, we also made available a python library that can handle multiple formats*.

2.1.2 Face Detection

State-of-the-art face detection relies on neural networks that are trained on large databases of face images, to cite the latest from a wide literature, readers should refer to [181, 178, 182, 183]. Neural Networks usually rely on intensive computation that necessitates dedicated hardware co-processors (usually GPU) to enable real-time operations [184]. Currently dedicated chips such as Google's TPU or Apple's Neural Engine have become an essential tool for frame-based vision. They are designed to accelerate matrix multiplications at the core of neural networks inference. However the computation costs associated to these computations are extremely high.

Dedicated blink detection using conventional frame-based techniques operate on a single frame. To constrain the region of interest, a face detection algorithm is used beforehand [185]. In an event-based approach, the computational scheme can be inverted as

*The library is available under https://github.com/neuromorphic-paris/loris and a description in Appendix A.1

detecting blinks is the mechanism that drives face detection.

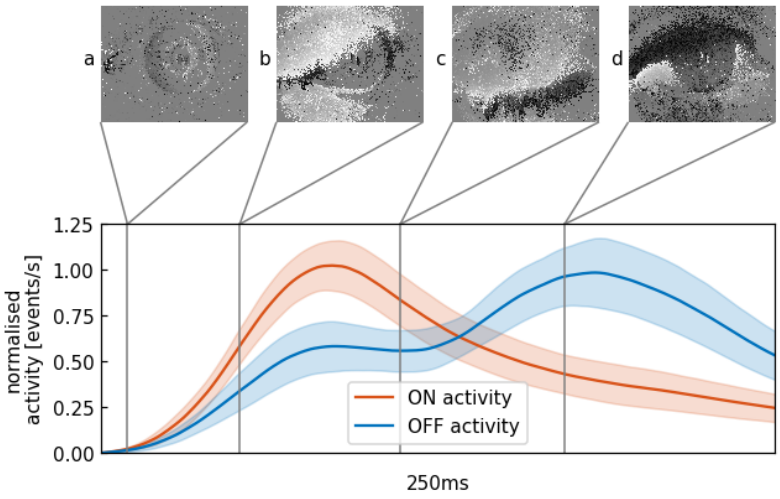

Fig. 2.3 *Mean and variance of the continuous activity profile of averaged blinks in the outdoor data set with a decay constant of 50 ms. a) minimal movement of the pupil, almost no change is recorded. b) eye lid is closing within 100 ms, lots of ON-events (in white) are generated. c) eye is in a closed state and a minimum of events is generated. d) opening of the eye lid is accompanied by the generation of mainly OFF-events (in black).*

2.1.3 Human Eye Blinks

Humans blink synchronously in correlation to their cognitive activities and more often than required to keep the surface of the eye hydrated and lubricated. Neuroscience research suggests that blinks are actively involved in the release of attention [186]. Generally, the observed eye blinking rates in adults depend on the subject's activity and level of focus. Rates can range from $3\,blinks/min$ when reading to up to $30\,blinks/min$ during conversation (Table 2.1). Fatigue significantly influences blinking behaviours, increasing both rate and duration [187]. In this work we will not consider these boundary cases that will the be subject of further work on non-intrusive driver monitoring [188, 189]. A typical blink duration is between $100 - 150\,ms$ [190]. It shortens with increasing physical workload, increased focus or airborne particles related to air pollution [191].

To illustrate what happens during an event-based recording of an eye blink, Figure 2.3 shows different stages of the eye lid closure and opening. If the eye is in a static state, few events will be generated (a). The closure of the eye lid happens within 100 ms and generates a substantial amount of ON events, followed by a slower opening of the eye (c,d) and the generation of primarily OFF events. From this observation, we devise a method

Activity (blinks/min)	[192]	[187]
reading	4.5	3-7
at rest	17	-
communicating	26	-
not reading	-	15-30

Table 2.1: Mean blinking rates according to [192] and [187].

to build a temporal signature of a blink. This signature can then be used to signal the presence of a single eye or pair of eyes in the field of view that can then be interpreted as the presence of a face.

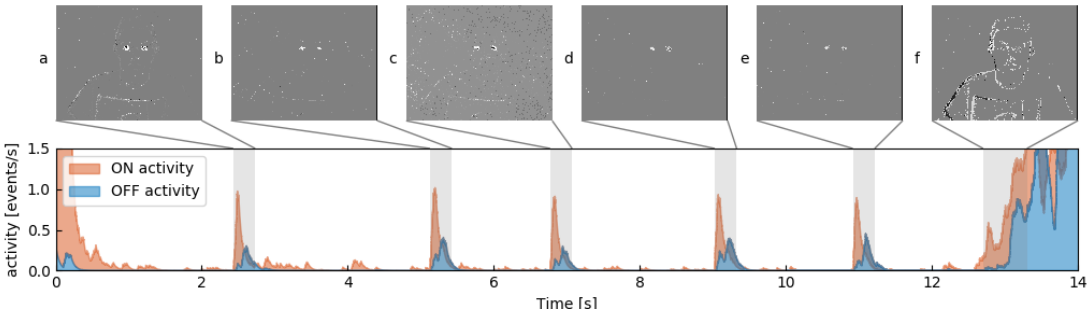

Fig. 2.4 *Showing ON (red) and OFF (blue) activity for one tile which lines up with one of the subject's eyes. Multiple snapshots of accumulated events for 250 ms are shown, which corresponds to the grey areas.* ***a-e)*** *Blinks. Subject is blinking.* ***f)*** *Subject moves as a whole and a relatively high number of events is generated.*

2.2 Methods

2.2.1 Temporal Signature of an Eye Blink

Eye blinks can be represented as a temporal signature. To build a canonical eye blink signature $A(t_i)$ of a blink, we convert events acquired from the sensor into temporal activity. For each incoming event $ev = (x_i, y_i, t_i, p_i)$, we update $A(t_i)$ as follows:

$$A(t_i) = \begin{cases} A_{ON}(t_i) = A_{ON}(t_u)e^{-\frac{t_i - t_u}{\tau}} + \frac{1}{scale} & \text{if } p_i\text{=ON} \\ A_{OFF}(t_i) = A_{OFF}(t_v)e^{-\frac{t_i - t_v}{\tau}} + \frac{1}{scale} & \text{if } p_i\text{=OFF} \end{cases} \tag{2.1}$$

where t_u and t_v are the times an ON or OFF event occurred before t_i. The respective activity function is increased by $\frac{1}{scale}$ each time t_n an event ON or OFF is registered

(light increasing or decreasing). The quantity *scale* initialised to 1 acts as a corrective factor to account for a possible change in scale, as a face that is closer to the camera will inevitably trigger more events. Figure 2.4 shows the two activity profiles where 5 profiles of a subject's blinks are shown, as well as much higher activities at the beginning and the end of the sequence when the subject moves as a whole. From a set of manually annotated blinks we build such an activity model function as shown in Figure 2.3 where red and blue curve respectively represent the ON and OFF parts of the profile.

Our algorithm detects blinks by checking whether the combination of local ON- and OFF-activities correlates with the canonical model of a blink that had previously been learned from annotated material. To compute the local activity, the overall input focal plane is divided into one grid of $n \times n$ tiles, overlapped with a second similar grid made of $(n-1) \times (n-1)$ tiles. Each of these are rectangular patches, given the event-camera's resolution of 304×240 pixels. The second grid is shifted by half the tile width and height to allow for redundant coverage of the focal plane. In this work we set experimentally $n = 16$ as it corresponds to the best compromise between performance and the available computational power of the used hardware.

2.2.1.1 Blink Model Generation

A total of $M = 120$ blinks from 6 subjects are manually annotated from the acquired data to build the generic model of an eye blink shown in Figure 2.3. Each blink, extracted within a time window of 250ms is used to compute an activity function as defined in Equation 2.1. The blink model is then obtained as the average of these activity functions:

$$B(t) = \begin{cases} B_{ON}(t) = \sum_{k=1}^{M} \frac{A_{ON}(t)}{M}, & \text{if } p_i\text{=ON} \\ B_{OFF}(t) = \sum_{k=1}^{M} \frac{A_{OFF}(t)}{M}, & \text{if } p_i\text{=OFF} \end{cases} \tag{2.2}$$

To provide some robustness and invariance to scale and time changes to the blink model, we also define N, the number of events per unit of time and normalised by the scale factor. This number provides the number of samples necessary to calculate the cross-correlation to detect blink as explained in section 2.2.1.2.2.1.2. Formally, $N = \lfloor \frac{\#\text{events}}{T.scale} \rfloor$, where $\lfloor . \rfloor$ is the floor function giving the largest integer smaller than $\frac{\#\text{events}}{T.scale}$.

Finally, we used two different models for indoor and outdoor scenes, as experiments showed that the ratio between ON and OFF events change substantially according to the lighting conditions. Although the camera is supposed to be invariant to absolute illumination, this is practically not the case due to hardware limitations of early camera generation that we used for this work.

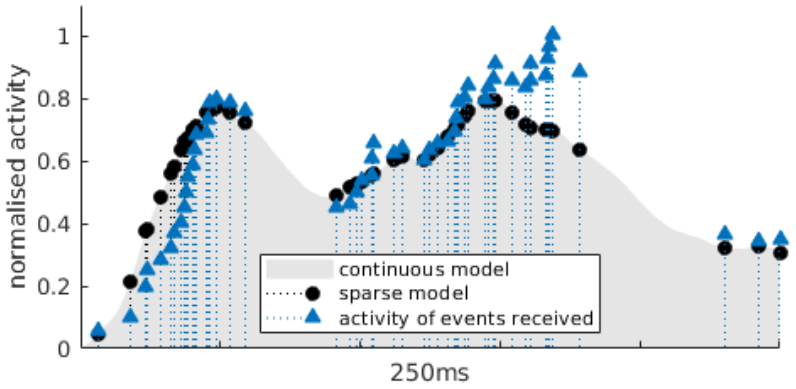

Fig. 2.5 *Example of the samples used to calculate the sparse cross-correlation for the OFF activity of an actual blink. The grey area represents B_{OFF}, the activity model for OFF events (in that particular example, it is previously built for outdoor data sets). Blue triangles correspond to the activity $A(t_i)$ for which events have been received in the current time window. Black dots are the $B_{OFF}(t_i)$, the value of activity in the model at the same times-tamps as incoming events.*

2.2.1.2 Sparse Cross-correlation

When streaming data from the camera, the most recent activity within a $T = 250\,\text{ms}$ time window is taken into account in each tile to calculate the similarity score, here the cross-correlation score, for the ON and OFF activities. This cross-correlation is only computed if the number of recent events exceeds an amount N defined experimentally according to the hardware used. The cross-correlation score between the incoming stream of events and the model is given by:

$$C(t_k) = \alpha C_{ON}(t_k) + (1-\alpha)C_{OFF}(t_k), \tag{2.3}$$

where

$$C_p(t_k) = \sum_{i=0}^{N} A_p(t_i)B_p(t_i - t_k), \tag{2.4}$$

with $p \in \{ON, OFF\}$. The ON and OFF parts of the correlation score are weighted by a parameter α set experimentally that tunes the contribution of the ON vs OFF events. This is a necessary step as due to the camera manual parameter settings, the amount of ON and OFF events are usually not balanced. For all experiments, α is set to $\frac{2}{3}$.

It is important for implementation reasons to compute the correlation as it appears in Equation 2.4. While it is possible to calculate the value of the model $B(t-t_k)$ at anytime t, samples for A are only known for the set of times $\{t_i\}$, from the events. This is illustrated as an example by Figure 2.5, for an arbitrary time t_k, where triangles outline the samples

of the activity for calculated events at t_i and the circles show the samples calculated at the same time t_i with the model. If $C(t_i)$ exceeds a certain threshold, we create what we call a blink candidate event for the tile in which the event that triggered the correlation occurred. Such a candidate is represented as the n-tuple $eb = (r, c, t)$, where (r, c) are the row and column coordinates of the grid tile and t is the timestamp. We do this since we correlate activity for tiles individually and only in a next step combine possible candidates to a blink.

2.2.1.3 Blink Detection

To detect the synchronous blinks generated by two eyes, blink candidates across grids generated by the cross-correlation are tested against additional constraints for verification. As a human blink has certain physiological constraints in terms of timing, we check for temporal and spatial coherence of candidates in order to find true positives. The maximum temporal difference between candidates will be denoted as ΔT_{max} and is set experimentally to 50 ms, the maximum horizontal spatial disparity ΔH_{max} is set to half the sensor width and the maximum vertical difference ΔV_{max} is set to a fifth of the sensor height. Following these constraints we will not detect blinks that happen extremely close to the camera or stem from substantially rotated faces. Algorithm 1 summarises the set of constraints to validate the detection of a blink. The scale factor here refers to a face that has already been detected.

Algorithm 1: Blink detection

Inputs: A pair of consecutive blink candidate events $eb_u = (r_u, c_u, t_u)$ and $eb_v = (r_v, c_v, t_v)$ with $t_u > t_v$
if *(*$t_u - t_v < \Delta T_{max}$*) AND (*$|r_u - r_v| < \Delta V_{max} \times scale$*) AND (*$|c_c - c_v| < \Delta H_{max} \times scale$*)* **then**
    if *face is a new face* **then**
        return 2 trackers with $scale = 1$
    else
        return 2 trackers with previous *scale*
    end
end

2.2.2 Gaussian Tracker

Once a blink is detected with sufficient confidence, a tracker is initiated at the detected location. We use trackers such as the ones presented in [101] that rely on bivariate normal distributions to locally model the spatial distribution of events. For each event, every

tracker is assigned a score that represents the probability of the event to belong to the tracker:

$$p(u) = \frac{1}{2\pi}|\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(u-\mu)^T \Sigma^{-1}(u-\mu)} \tag{2.5}$$

where $u = [x, y]^T$ is the pixel location of the event, is covariance matrix Σ that defines the shape and size of the tracker. The tracker with the highest probability is updated according to the activity of pixels and also according to the estimated distance between the spatial locations of the detected eyes.

2.2.3 Global Algorithm

The detection and tracking blocks combined operations are summarized by following algorithm:

Algorithm 2: Event-based face detection and tracking algorithm

```
for each event ev(x, y, t, p) do
    if at least one face has been detected then
        update best blob tracker for ev as in (2.5)
        update scale of face for which tracker has moved according to tracker distance
    end
    update activity according to (2.1)
    correlate activity with model blink as in (2.3)
    run Algorithm 1 to check for a blink
end
```

2.3 Experiments and Results

We evaluated the algorithm's performance by running cross-validation on a total of 48 recordings from 10 different subjects, comprising 248 blinks. Our annotated dataset is publicly available to encourage further research in this direction[†]. The recordings are divided into 5 sets of experiments to assess the method's performances under realistic constraints encountered in natural scenarios. The event-based camera is kept static and test the following scenarios of sequences of:

- indoor and outdoor sequences showing a single subject moving in front of the camera,
- a single face moving back and forth w.r.t. the camera to test the robustness of scale change,

[†]Dataset is available under https://tinyurl.com/face-detection-dataset

- several subjects discussing, facing the camera to test for multi-detection,
- a single face changing its orientation w.r.t. the camera to test for occlusion resilience.

The presented algorithm has been implemented in C++ and runs in real-time on an Intel Core i5-7200U laptop CPU. We are quantitatively assessing the proposed method's accuracy by comparing it with state of the art and gold standard face detection algorithms from frame-based computer-vision. As these approaches require frames, we are generating grey-levels from the camera when this mode is available. The Viola-Jones (VJ) algorithm [177] provides the gold standard face detector but we also considered the Faster Region Based Convolutional Neural Network (FRCNN) [193] and the Single Shot Detector (SSD) network [179] that have been trained on the Wider Face[194] data set. This allows us to compare the performances of the event-based blink detection and tracking with state-of-the-art face detectors based on deep learning. Finally, we also tested a conventional approach that combines a CNN with a correlation filter presented in [180]. It is referred to as Correlation Filter (CF) for the rest of the chapter. This technique relies on creating frames by summing the activities of pixels within a predefined time window.

An important statement to keep in mind is that the proposed blink detection and face tracking technique requires reliable detection. We do not actually need to detect all blinks since a single detection is already sufficient to initiate the trackers. Additional blink detections are used to correct a trackers' potential drifts regularly.

2.3.1 Indoor and Outdoor Face Detection

The indoor data set consists of recordings in controlled lighting conditions. Figure 2.6 shows tracking results. The algorithm starts tracking as soon as a single blink is detected (a). Whereas tracking accuracy on the frame-based implementation is constant (25 frames per second (fps)), our algorithm is updated event-by-event depending on the movements in the scene. If the subject stays still, the amount of computation is drastically reduced as there is a significantly lower number of events. Head movement causes the tracker to update within µs (b).

Subjects in the outdoor experiments were asked to step from side to side in front of a camera placed in a courtyard under natural lighting conditions. They were asked to gaze into a general direction, partly engaged in a conversation with the person who recorded the video. Table 2.2 shows that results are similar to indoor conditions. The slight difference is due to the non-idealities of the sensor (same camera parameters as in the indoor experiment). It is important to emphasise that event-based cameras still lack an automatic tuning system of their parameters that hopefully will be developed for the

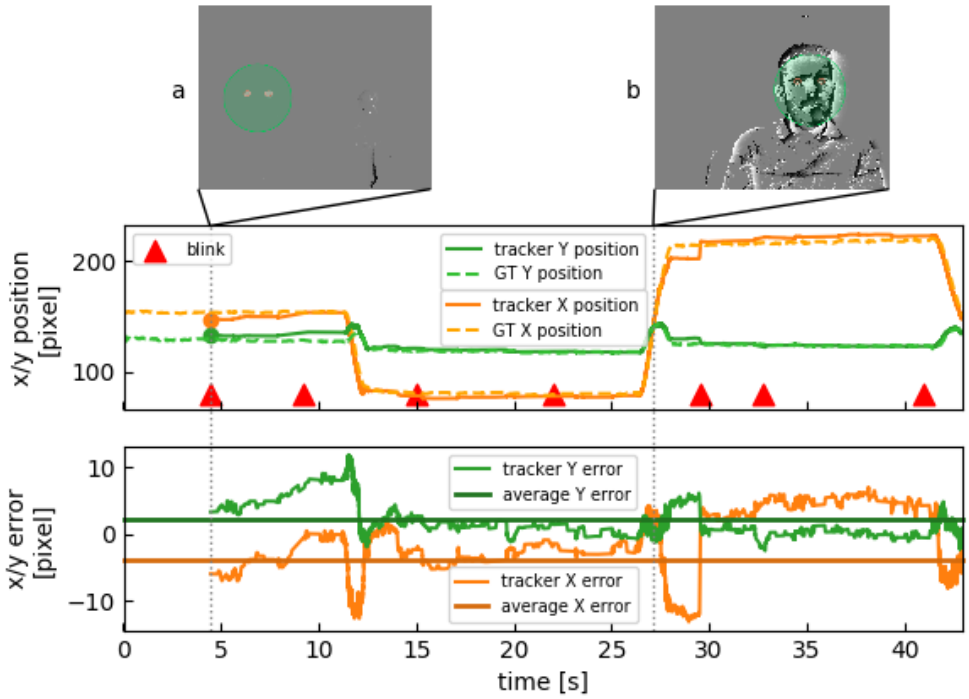

Fig. 2.6 *face tracking of one subject over 45s. a) subject stays still and eyes are being detected. Movement in the background to the right does not disrupt detection. b) when the subject moves, several events are generated*

future generations of a cameras.

2.3.2 Face Scale Changes

In 3 recordings the scale of a person's face varies by a factor of more than 5 between the smallest to the largest detected occurrence. Subjects were instructed to approach the camera within 25 cm from their initial position to then move away from the camera after 10 s to about 150 cm. Figure 2.7 shows tracking data for such a recording. The first blink is detected after 3 s at around 1 m in front of the camera (a). The subject then moves very close to the camera and to the left so that not even the whole face bounding box is seen anymore (b). Since the eyes are still visible, this is not a problem for the tracker. However, ground truth had to be partly manually annotated for this part of the recording, as two of the frame-based methods failed to detect the face that was too close to the camera. The subject then moves backwards and to the right, followed by further re-detections (c).

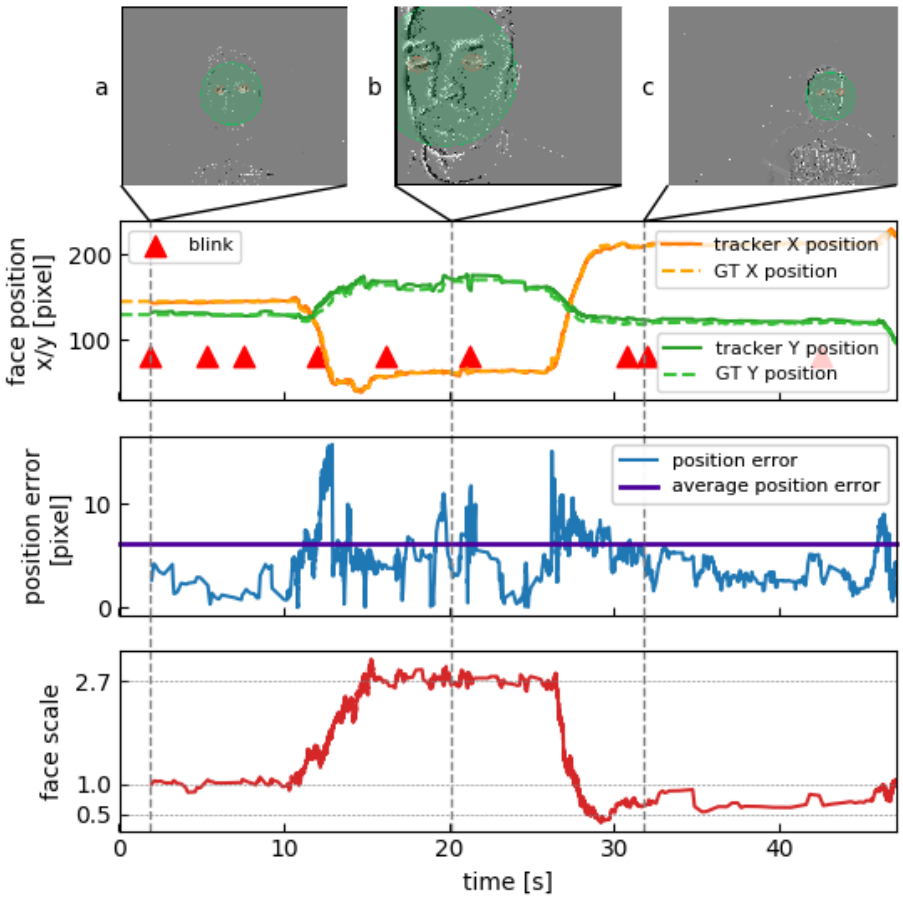

Fig. 2.7 *Verifying robustness to scale. a) first blink is detected at initial location. Scale value of 1 is assigned. b) Subject gets within 25cm of the camera, resulting in a three-fold scale change. c) Subject veers away to about 150cm, the face is now 35% smaller than in a)*

2.3.3 Multiple Faces Detection

We recorded 3 sets of 3 subjects sitting at a desk talking to each other. No instructions where given to the subjects. Figure 2.8 shows tracking results for the recording. The three subjects stay relatively still, but will look at each other from time to time as they are engaged in a conversation or sometimes focus on a screen in front of them. Lower detection rates (see Table 2.2) are caused by an increased pose variation, however this does not result in an increase of the tracking errors due to the absence of drift.

2.3.4 Pose Variation Sequences

The subjects in these sequences are rotating their head from one side to the other until only one eye is visible in the scene. Experiments show that the presence of a single eye

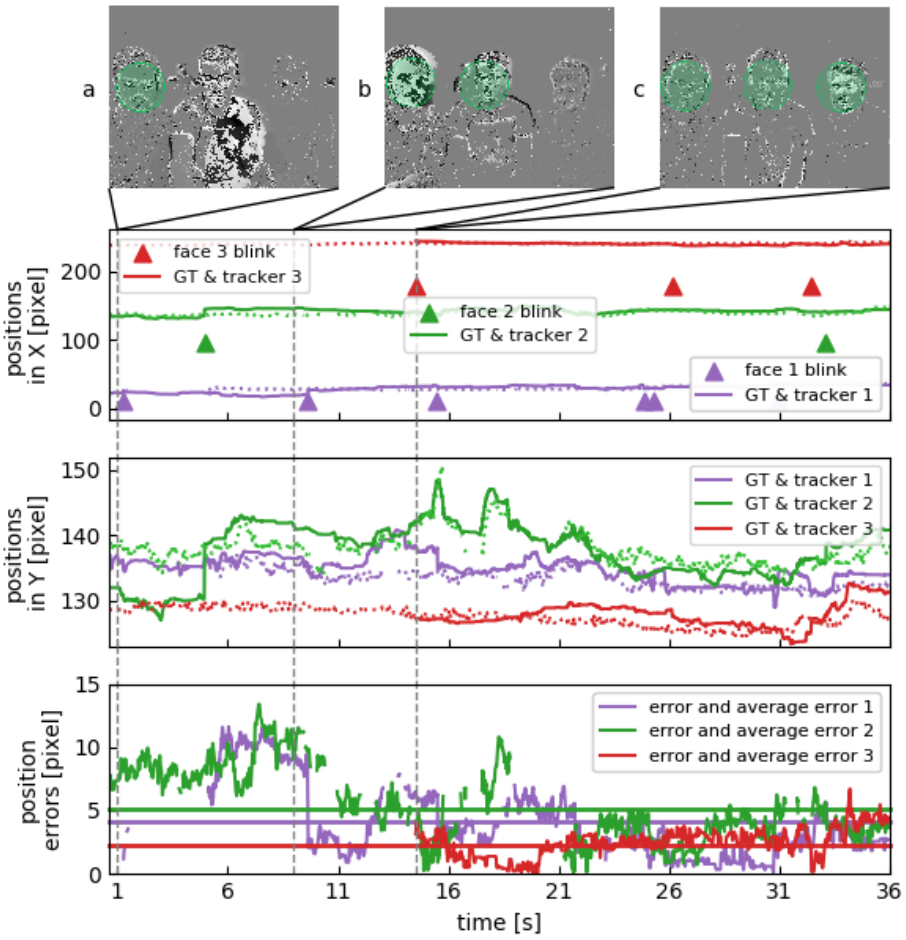

Fig. 2.8 *Multiple face tracking in parallel. Face positions in X and Y show three subjects sitting next to each other, their heads are roughly on the same height. a) subject to the left blinks at first. b) subject in the centre blinks next, considerably varying their face orientation when looking at the other two. c) third subject stays relatively still.*

does not affect the performances of the algorithm (see Figure 2.9). These experiments have been carried out with an event-based camera that has a resolution of 640×480 pixels. While this camera provides better temporal accuracy and spatial resolution, it does not provide grey-level events measurements.

Although we fed frames from the change detection events (which do not contain absolute grey-level information but are binary) to the frame-based methods, none of them could detect a face. This can be expected as the networks have been trained on grey-level images instead.

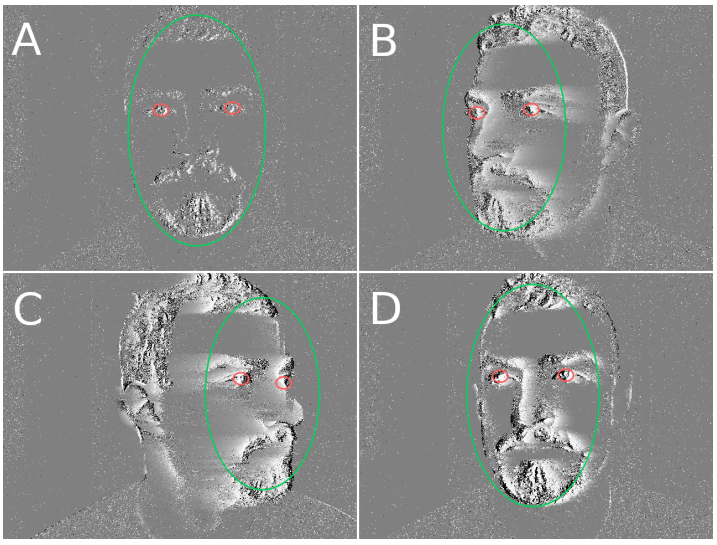

Fig. 2.9 *Pose variation experiment.* ***A****) Face tracker is initialised after blink.* ***B****) subject turns to the left.* ***C-D****) One eye is occluded, but tracker is able to recover.*

2.3.5 Summary

Table 2.2 summarises the relative accuracy of the detection and the tracking performances of the presented method, in comparison to VJ [177], FRCNN [193], SSD [179] and CF [180]. We also compiled a video that shows visual results [‡]. We set the correlation threshold to a value that is guaranteed to prohibit false positive detections, in order to (re-)initialise trackers at correct positions. The ratio of detected blinks is highest in controlled indoor conditions and detection rates in outdoor conditions are only slightly inferior. We attribute this fact to the aforementioned hardware limitations of earlier camera generations that are sensitive to lighting conditions. A lower detection rate for multiple subjects is mostly due to occluded blinks when subjects turn to speak to each other.

The tracking errors are the deviations from the frame-based bounding box centre, normalised by the bounding box's width. The normalisation provides a scale invariance so that errors estimated for a large bounding box from a close-up face have the same meaning as errors for a small bounding box of a face further away.

VJ, FRCNN and SSD re-detect faces at every frame and discard face positions in previous frames, resulting in slightly erratic tracking over time. They do however give visually convincing results when it comes to accuracy, as they can detect a face right from the start of the recording and at greater pose variation given the complex model of a neural network. CF uses a tracker that updates its position at every frame that is created from

[‡]The result video is available under https://youtu.be/F5UzXQsr5Es

	# of recordings	blinks detected (%)	error VJ (%)	error (%) FRCNN	error SSD (%)	error CF (%)
indoor	21	68.4	5.92	9.42	9.21	10.51
outdoor	21	52.3	7.6	14.57	15.08	14.88
scale	3	62.6	4.8	10.17	10.22	17.6
multiple	3	36.8	15	16.15	14.61	n/a
total	48	59	7.68	11.77	11.52	12.82

Table 2.2: Summary of results for detection and tracking for 4 sets of experiments. % of blinks detected relates to the total number of blinks in a recording. Tracking errors are Euclidean distances in pixel between the proposed and respective method's bounding boxes, normalised by the frame-based bounding box width and height in order to account for different scales.

binning the change detection events, rather than working on grey-level frames. The tracker update at each frame based on the previous position ensures a certain spatial consistency and smoothness when tracking, at the temporal resolution of the frame rate. However, since a correlation filter was initially designed for classic (grey-level) images, it relies on visual information of the object to track to be present at all time, which is not necessarily the case for an event-camera.

The CF technique from [180] requires the camera to move constantly in order to obtain visual information from the scene to maintain the tracking, as the algorithms uses rate-coded frames. This required us to modify their algorithm since in our data, tracked subjects can stop w.r.t. to the camera, hence they became invisible. We added a mechanism to the correlation filter that freezes the tracker's position when the object disappears. We use a maximum threshold of the peak-to-sidelobe ratio [195], which measures the strength of a correlation peak and can therefore be used to detect occlusions or tracking failure while being able to continue the online update when the subject reappears. This results in delays in tracking whenever an object starts to move again and results in tracking penalties. CF has further limitations at tracking at high scale variance and cannot track multiple objects of the same nature at the same time.

2.4 Discussion

We introduced a method able to perform face detection and tracking using the output of an event-based camera. We have shown that these sensors can detect eye blinks in real time. This detection can then be used to initialise a tracker and avoid drifts. The approach

makes use of dynamical properties of human faces rather than relying on an approach that only uses static information of faces and therefore only their spatial structure. The face's location is updated at µs precision once the trackers have been initialised, which corresponds to the native temporal resolution of the camera. Tracking and re-detection are robust to more than a five-fold scale, corresponding to a distance in front of the camera ranging from 25 cm to 1.50 m. A blink provides robust temporal signatures as its overall duration changes little from subject to subject. The amount of events received and therefore the resulting activity amplitude varies only substantially when lighting of the scene is extremely different (i.e. indoor office lighting vs bright outdoor sunlight). The model generated from an initial set of manually annotated blinks has proven to be robust to those changes across a wide set of sequences. The algorithm mechanism is also robust to eye occlusions and can still operate when face moves from side to side allowing only a single blink to be detected. In the most severe cases of occlusion, the tracker manages to reset correctly at the next detected blink.

The occlusion problem could be further mitigated by using additional trackers to track more facial features such as the mouth or the nose and by linking them to build a part-based model of a face as it has been tested successfully in [196]. The blink detection approach is simple and yet robust enough for the technique to handle up to several faces simultaneously. We expect to be able to improve detection accuracy by learning the dynamics of blinks via techniques such as HOTS [85] or Histogram of Averaged Time Surfaces (HATS) [92]. At the same time with increasingly efficient event-based cameras providing higher spatial resolution, the algorithm is expected to increase its performance and range of operations. We estimated the power consumption of the compared algorithms to provide numbers in terms of efficiency:

- The presented event-based algorithm runs in real-time using 70% of the resources of a single core of an Intel i5-7200U CPU for mobile Desktops, averaging to 5.5 W of power consumption while handling a temporal precision of 1 µs [197].
- The OpenCV implementation of VJ is able to operate at 24 of the 25 fps in real-time, using a full core at 7.5 W [197]).
- The FRCNN Caffe implementation running on the GPU uses 175 W on average on a Nvidia Tesla K40c with 4-5 fps.
- The SSD implementation in Tensorflow runs in real-time, using 106 W on average on the same GPU model.

These numbers show that spike-based computation can outperform conventional approaches in terms of power efficiency, even when executed on a regular desktop. Specialised hardware has the potential to further amplify those advantages.

Chapter 3

A Framework for Event-based Computer Vision on a Mobile Phone

In this chapter we examine how mobile systems can benefit from the event-driven nature of event-based algorithms. The optimised, RISC-based hardware can be seen as an intermediate step to dedicated neuromorphic hardware. The key to saving energy is, as in the previous chapter, to exploit the fact that camera output is directly driven by visual scene activity, contrary to conventional sensors that are clocked. Downstream processing can therefore be prevented in the absence of visual stimuli. This opens up the possibility for power-efficient always-on sensing on a mobile phone. For tasks that happen infrequently when interacting with the phone, we see an increasing return of investment when employing event-based algorithms. Gesture recognition is an example application that needs always-on sensing and conventional cameras have played a central role in facilitating such tasks. But they cannot record continuously, as the amount of redundant information recorded is costly to process. To overcome this limitation, we present an Android framework that connects efficient event-based sensors and algorithms to resource constrained mobile devices for a new generation of low-power human-computer interfaces. The mobile framework allows us to stream events in real-time and opens up the possibilities for always-on and on-demand sensing on mobile phones. To combine the asynchronous event camera output with synchronous hardware used in mobile phones, we look at how buffering events and processing them in batches can benefit on-device computer vision applications. We evaluate our framework in terms of latency and throughput and show examples of computer vision tasks that involve both event-by-event and pre-trained neural network methods applied to a dataset of mid-air hand gestures.

3.1 Introduction

Mobile handheld devices are indispensable technology nowadays. An increasing range of their functionality is powered by machine learning models and in particular neural networks that are trained offline and deployed for inference. To be able to perform on-device inference instead of computing in the cloud is important for a number of reasons [198]:

- Since there is no round-trip to a server, latency is greatly reduced.
- No data needs to leave the device, which avoids issues regarding the user's privacy.
- An internet connection is not required, which is beneficial to autonomy.
- Network connections are power hungry and should therefore be avoided if possible.

The number of parameters in neural network models grows rapidly every year. To be able to employ them on mobile devices that have constraint power budgets, we have seen the emergence of specialised neural network accelerator hardware and different approaches to reduce model size, number of floating point operations as well as latency [199, 51, 48]. But no matter the optimisations performed, neural networks still have to crunch a lot of redundant data, preventing mobile devices from continuously making use of them. For computer vision applications, this is because normally the visual scene is recorded using a conventional camera with a fixed frame rate, which is independent of the scene being recorded. Multiple cameras that are embedded into phones today not only serve to take pictures, but also facilitate tasks such as recognition of faces, gestures, objects, activities, and landmarks. Since image capturing and neural network inference are expensive, these tasks are often triggered by less computationally demanding sensors such as accelerometers or gyroscopes instead. In today's systems with a tight power budget it is essential to intelligently manage high fidelity sensors and processing to reduce power consumption as much as possible. But this can lead to latency issues or inaccurate triggering of the demanding processing in question and therefore consumes energy that could otherwise be saved.

Event-based computer vision tackles the need for efficiency by using a novel image sensor. It employs event cameras [33, 35, 200, 201, 36], which are emerging, biologically-inspired vision sensors that can operate in an always-on fashion using very little power. Their pixels are fully asynchronous and only ever triggered by a change in log illuminance. The amount of events that are output is thus directly driven by activity in the visual scene and can range from a few to hundreds of thousands events per second. Power consumption is coupled to the amount of events recorded, which gives event cameras an edge for applications that might happen infrequently over time.

Previous work has shown that mobile devices can profit from event cameras for low-power tasks such as visual activity detection [202], face detection [1], gesture recognition [2, 40], sensor fusion [203] or image deblurring [77, 204]. Event cameras have successfully been employed on robotic platforms, which have similar limited power constraints [205, 206, 207, 66]. Apart from the low power consumption, applications can also profit from high temporal resolution and good low-level light capture.

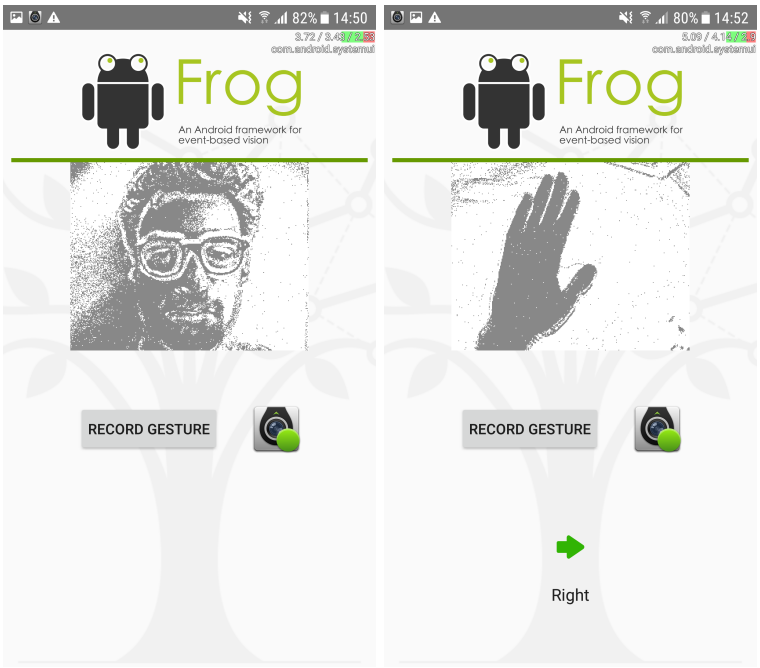

Fig. 3.1 *Screenshots of our Android app.* ***Left:*** *showing the live view of a connected event camera that renders in real-time.* ***Right:*** *Capturing a hand gesture being performed that signifies* Right.

In this work we use a prototype device for our experiments consisting of an off-the-shelf mobile phone to which we connect a small form-factor ATIS [35] event camera via mini-USB connection as shown in Figure 3.2. We mount the event camera on a printed frame such that it faces the user. The device is self-contained and does not need any external cabling. Once the camera is connected, our Android framework is able to stream data from the event camera in real-time, enabling on-device processing. We make it straightforward for a user to deploy their own code using Android's Native Development Kit (NDK) or to use pre-trained neural network models in combination with the Tensorflow Lite library. We give details about app architecture and how the different modules within depend on each other. We also provide examples of computer vision tasks that show the applicability of event cameras on mobile devices and benchmark throughput as well as latency to motivate further exploration in that direction. Overall, our contributions are as follows:

- A publicly available mobile framework to stream events from an event camera in real-time*.
- Real-time application of event-driven algorithms or pre-trained neural networks on

*The framework *Frog* is available under https://github.com/neuromorphic-paris/frog and the description in Appendix A.3.

events on a mobile phone.

- A self-contained device that uses a variable trigger in the form of an event camera for always-on computer vision applications.

Processing event by event from our camera can potentially achieve the lowest latency, as new information is integrated as soon as it is available. This kind of processing which does not use conventional frames needs rethinking computer vision algorithms from the ground up and has seen promising applications in event stream classification or detection [85, 208, 92, 1]. Since we execute event-based algorithms on conventional von Neumann as opposed to specialised neuromorphic hardware, the event-by-event approach of asynchronous input does come with an overhead when repeatedly updating a state up to hundreds of thousands of times per second. von Neumann hardware is designed to compute on bulks of memory, and not for fine-grained parallelism. We therefore examine the effect of buffering events, to be able to process them in batches. Depending on the algorithm, this can alleviate some of the computational burden, but also incurs latency. Buffering events means balancing a power/latency trade-off that depends on the number of input events per second. On one end of the spectrum, an input event rate of hundreds of thousands events per second for an active visual scene and a buffer size of 1 is likely to overwhelm a device such as a mobile phone with many updates. On the other end, a large buffer size when there are only few input events will not trigger any update at all. Depending on the application, we show how an acceptable trade-off can look like, to bring event-based computer vision to power-constrained mobile devices.

3.2 Mobile Device and Event Camera

Our device prototype as shown in Figure 3.2 consists of a Samsung Galaxy S6 smartphone and a small form-factor event camera. Small form-factor event cameras such as the embedded DVS [39] have a lower spatial resolution and optimised power consumption in comparison to normal event cameras since they target battery-powered devices. Our low-power version of an ATIS [35] has a spatial resolution of 304×240 pixels, is fixed on a 3d-printed external frame and connected to the device via the mini USB port. The camera die of size $5000 \times 5000 \mu m^2$ with a fill factor of 30% was fabricated using a UMC 180 nm process. Power consumption for the chip depends directly on scene activity, where one pixel draws 300 nW under static conditions or 900 nW under high activity. The readout of events is facilitated using an FPGA and draws 30 mW for high activity of all pixels. Input/output communication for the USB connections needs further 20 mW. The camera is embedded in a printed case on top of the mobile phone, to be able to directly face the user. In order to ensure flexibility and compactness, a stacked design of two printed circuit

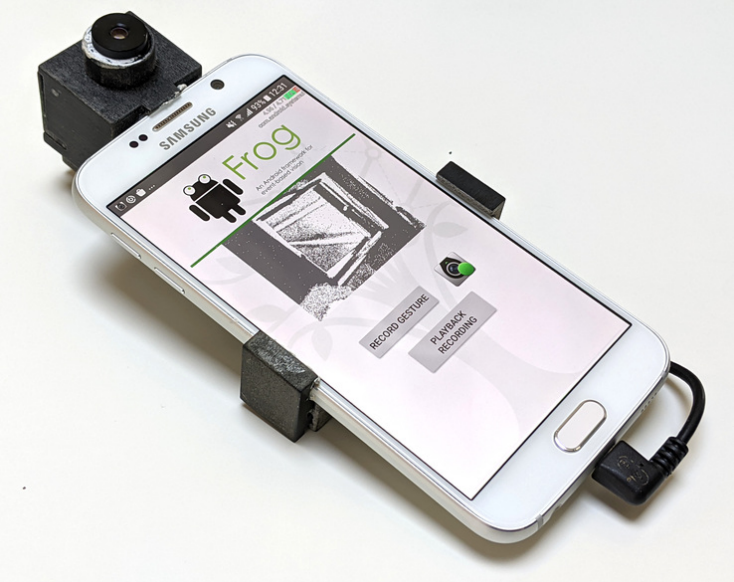

Fig. 3.2 *Prototype device, consisting of a Samsung Galaxy S6 and a small form-factor ATIS connected via mini-USB port. The camera is held in place with a 3D-printed frame that attaches to the phone.*

boards was chosen as depicted in Figure 3.3. In theory also other event cameras can be connected via USB as long as drivers are open source, although standalone cameras will need an external power supply.

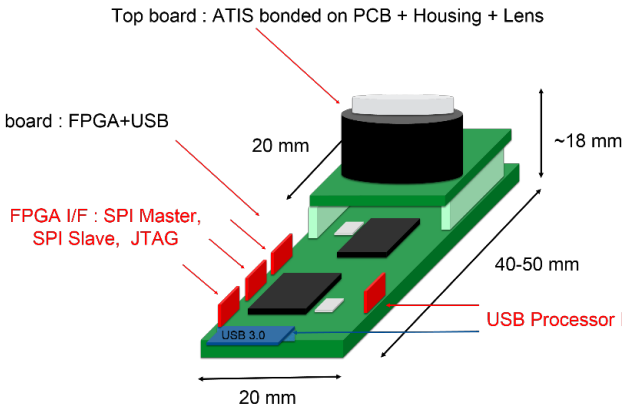

Fig. 3.3 *Small form-factor event camera assembly. the stacked PCB is located within the housing on top of the phone, as shown in Figure 3.2.*

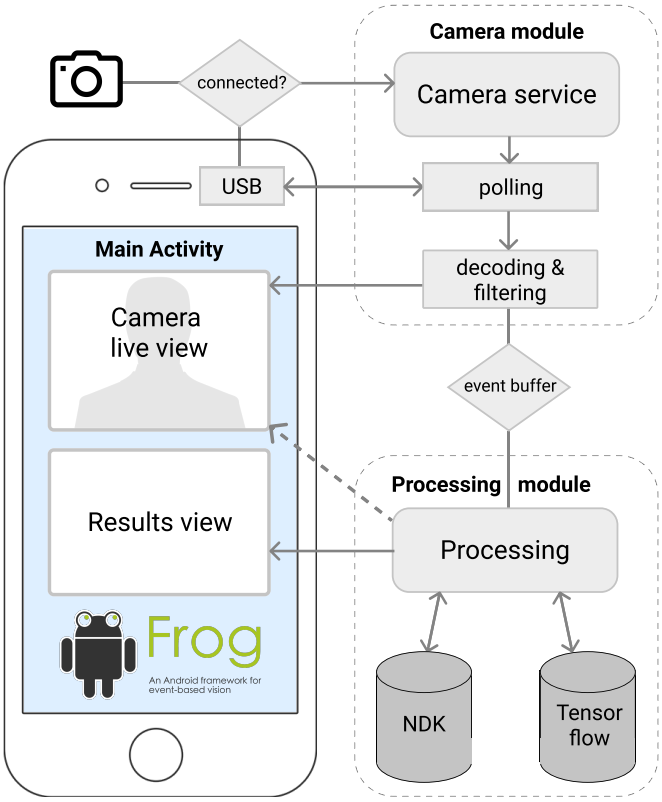

Fig. 3.4 *Application software architecture. Based on Android, we make use of a* ***Camera module*** *to handle the streaming of events from a camera and a* ***Processing module*** *that is able to run different algorithms depending on the backend. Both modules update Views in the* ***Main Activity****.*

3.3 Android Application Framework

Our proposed Android app facilitates the readout and processing of events from an event camera in a power-efficient way. We split the streaming of data packets via USB from the rendering and processing of user-defined code into separate modules, which are outlined in Figure 3.4:

1. The main activity which renders the user interface.
2. The camera module that deals with transferring events from the camera as well as accumulating them in a buffer.
3. The processing module that can be called on demand to execute algorithms on the events in the buffer.

From a functional standpoint, as soon as the event camera is connected, a live view will

start rendering the camera output on the screen so that the user can have visual feedback of how they interact with the device as shown in Figure 3.1 on the left. Data received from the camera is checked for isolated noise events and constantly firing pixels, which are filtered and discarded to not unnecessarily strain the downstream processing. In this phase, there is no computationally heavy processing necessary.

Whenever the user gives the signal to start processing the events with a pre-defined algorithm by pressing a button on screen, the app accumulates events in an event buffer, where they await further processing. The live view continues uninterrupted. As soon as the buffer of a specific size is full, the processing routine will be called in a separate worker thread. The event buffer size can be adjusted, which indirectly determines how often the processing routine is called. If the buffer size is too small, the amount of computations per second might overwhelm the phone's CPU. If the buffer size is too large, results are presented to the main activity very infrequently and might impact user experience. The right buffer size causes events to be processed in batches, balancing computational cost and result latency. The processing routine then returns a result depending on the algorithm used, such as a specific classification outcome or optical flow speeds. The result that is presented to the main activity can be displayed in a text box or used as an overlay for the live camera view. In the following part we will describe the 3 modules that the app consists of in more detail.

3.3.1 Main Activity

The main activity is responsible for the app life cycle and for rendering the user interface, bundling together the camera live view as well as results view. It is also responsible for handling the necessary permissions for USB devices, which a user has to agree to when they connect a new device. Any continuously ongoing processing such as USB polling has to be done in background threads, as otherwise the user experience would suffer if the interface started to lag or stall. The live camera view is rendered at the native frame rate of the phone, which is 60 Hz for the Samsung Galaxy S6. For efficiency reasons, the live camera view renders a binary bitmap at the native resolution of the event camera, which is then scaled up to display view size. The results view will update whenever the processing routine in the processing module returns a new result.

3.3.2 Camera Module and Event Buffer

This module deals with receiving the events from the event camera via USB and pre-processing them. As soon as such an event camera is connected, a *camera service* as part of the main thread will be started, which deals with the camera initialisation and handles two background threads for *polling* and *decoding*. The event camera and its FPGA need

2-3 seconds to power up, after which the camera's biases are set and it is switched into readout mode. The *polling* thread managed by the camera service is periodically querying the USB interface for new data packets, about every 1 ms. A packet can be anything from 0 to 16 kB, depending on the scene activity and therefore the event camera's output. Those packets are placed in a packet buffer, so that the polling can continue uninterrupted. The *decoding* thread managed by the camera service takes a USB packet from the packet buffer whenever available, and converts the binary blob into a number of events. The user can decide to apply simple refractory periods for each pixel to prevent excessive firing of pixels, or to apply additional filtering to remove noisy events. The same thread also directly updates the bitmap used by the Camera live view, at potentially much higher rate than the display refresh rate. If the user has triggered algorithm execution, the filtered events that were used to update the bitmap are then accumulated in an event buffer of size N. This buffer will act as a gate for downstream processing and will only trigger a computation when the buffer is full.

3.3.3 Processing Module

This module is responsible for the execution of algorithms using the batch of events that is passed from the event buffer. A third background thread is started whenever the processing routine is triggered. The routine can make use of different backends to make computation as efficient as possible. One option is the deployment of user code in C++, which can be executed natively on the phone using Android's NDK. The process routine can then call those native functions via the Java Native Interface, which take one or more events as parameter, to efficiently compute and return a result for that same batch. Calling a function through the Java Native Interface incurs an overhead, but the efficiency of native code execution often makes it worth wile. It should be noted that this backend uses a single background thread only.

Another option is to make use of a TensorFlow Lite backend [209], which is a framework for neural network inference for edge devices with hardware support on Android platforms. A neural network that has been trained offline can be processed to suit the deployment on an Android phone, by fusing and dropping as many operations as possible or quantizing weights to reduce computational effort and latency. The compressed network can be bundled with the Android app. Given an input, such a condensed network returns a similar result as the full precision network up to an error margin. A neural net that has been trained on events takes an accumulated event frame as input, so the event buffer size N will typically be higher in this setting. The neural network output and result can also be reported to the main activity to update the result and/or the live view.

3.4 Performance Measurement Methods

We benchmark the components of our system that contribute to the overall latency from the point when the event camera emits an event until a result is computed and measure the amount of events that can be handled per second. These components are the Camera module including its event buffer and the Processing module.

3.4.1 Camera Latency

At first we want to measure how quickly we can transfer, decode and filter events sent from our event camera connected via USB. Depending on scene activity, the event camera can generate up to hundreds of thousands of events per second. We denote the rate of events/s recorded by the camera as R. It serves a proxy for activity in the visual scene. The latency incurred by the camera module is the time it takes to transfer, decode and filter a USB packet of events: $\lambda_{\text{cam}} = (t_{\text{transfer}} + t_{\text{decoding}} + t_{\text{filter}})N_{packet}^{-1}$, where N_{packet} is the number of events in a USB packet. The accumulated latency per second for the camera module is:

$$\mathcal{L}_{\text{cam}}(R) = R\lambda_{\text{cam}} \tag{3.1}$$

3.4.2 Buffering Latency

After the events have been decoded, they are placed in the event buffer. Performing computation on each event individually incurs a large overhead when looking up and dereferencing functions [210]. We can therefore accumulate multiple events in our event buffer of size N to then process them as a batch. This typically saves overhead costs of creating new threads and updating states with every event, but the buffer size has to be chosen depending on the application and R. The accumulation of events causes latency per event that is the inverse of the input stream event rate, $\lambda_{\text{buffer}} = \text{R}^{-1}$. Bigger buffer sizes will cause longer latency and vice versa. The cost of moving data to and from the buffer is factored into camera and processing module respectively. To calculate the latency that is accumulated per second, we write:

$$\mathcal{L}_{\text{buffer}}(R, N) = N\lambda_{\text{buffer}}s^{-1} \tag{3.2}$$

3.4.3 Execution Latency

We measure the latency for a certain algorithm A as a function of buffer size, $\lambda_{\text{exec}} = A(N)$. If this function exhibits sublinear behaviour, the algorithm benefits from batching operations. To calculate accumulated latency per second, we multiply by the number of executions per second:

$$\mathcal{L}_{\text{exec}}(A, R, N) = \frac{R}{N}\lambda_{\text{exec}} \tag{3.3}$$

Together, these terms provide us with a tool to measure latency:

$$\mathcal{L}(A, R, N) = \mathcal{L}_{\text{cam}} + \mathcal{L}_{\text{buffer}} + \mathcal{L}_{\text{exec}} \tag{3.4}$$

$\mathcal{L}(A, R, N)$ computes a dimensionless output that tells us how many seconds of latency is accumulated per second from the moment that an event originates at the camera to the point when an algorithm returns a result. Everything at or under a value of 1 will be able to run in real-time.

3.5 Experiments and Results

We benchmark the amount of events that we can handle in real-time from our event camera within the camera module, calculate buffer latency for different input event rates R and implement 3 different computer vision algorithms on a mobile phone with the help of our framework. In the following experiments, we study event buffer size and its effect on latency for gesture recognition, computation of optical flow and image reconstruction from events. Optical flow computation is a relatively lightweight algorithm, which gives low-level information about direction and speed of events and which can benefit from batching operations. With gesture recognition we want to exploit the event camera's natural suitability as a motion detector to extract higher-level information and make use of an event-based learned model. The frame reconstruction is an example of a comparatively inexpensive neural network model that has been trained on events directly and that can make use of the TensorFlow backend. It also serves as a connection between purely event-based and conventional machine learning applications.

3.5.1 Measuring Throughput of Camera Module and Event Buffer Latency

For a scene where a user is holding the phone in front of them, we observe 0.91 ms of latency on average to transfer a USB packet that encodes 1024 events. The decoding of such a packet including filtering and setting the shared bitmap live view takes another 0.73 ms on a separate thread on average. The filtering is done to alleviate computational burden on our test device, where we remove about two thirds of events from the input stream. For that we use a refractory period of 1 ms, a spatiotemporal filter of 1 pixel and 1 ms and also remove 2 constantly firing pixels completely. This results in

$$\lambda_{\text{cam}} = 1.6 \pm 0.3 \mu s/Event \tag{3.5}$$

of latency for transferring, decoding and filtering events from the camera. This translates to a maximum event rate of 624.39 kEvents/s that we can sustain for real-time live view using a single CPU thread.

For reasons of reproducibility and comparability, we benchmark all downstream components using a pre-recorded dataset. We use gesture recordings from the Navgesture database [40] which have been acquired with the same camera as ours. It contains 1342 recordings of 6 different mid-air hand gestures performed. About 1 billion events are distributed over 47 minutes of recording time, which when distributed equally corresponds to an average R of 365.6 kEvents/s. Applying the same filtering as in the previous camera module experiment, this leaves us with an average input R of 113.9 kEvents/s.

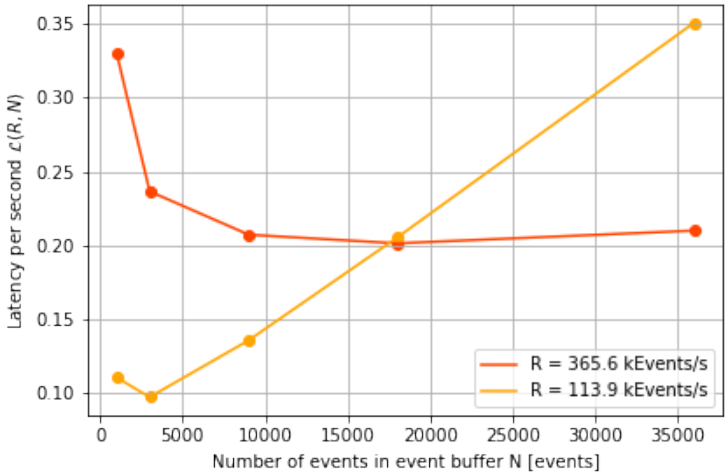

Fig. 3.5 *Accumulated latency per second for different event rates when computing event-based, aperture robust optical flow [96]. For high event rates (orange line) the overhead of calling a function repeatedly when the buffer size is low dominates the overall latency. For lower event rates (yellow line), buffer size can be considerably lower while still being able to compute in real-time. High buffer size combined with fewer input events means that events are spending a lot of time in the buffer which increases latency again.*

3.5.2 Aperture Robust Event-based Optical Flow

We implement and benchmark event-based aperture-robust flow as in Alkolkar et al [96]. Standard event-based flow as in Benosman et al. [93] provides directions that are perpendicular to the surface formed by the events, which does not necessarily correspond to the true direction of motion. Alkolkar et al. propose an algorithm that corrects the optical flow over a spatial region. It can be divided into three steps: At first, local optical flow for each event is computed via a least-squares minimisation of a plane, as described in [94]. In the second step, different spatial scales are evaluated over which the mean magnitude of local flows is maximised. In the third step, the mean direction of local flows is calculated for the previously found optimal spatial scale.

This algorithm, especially the second step, can directly benefit from batching operations, as multiple spatial scales can be evaluated more efficiently. Figure 3.5 shows the effect of accumulated latency per second when computing the corrected flow. The algorithm computes in real-time even for high event rates of $> 365\,\text{kEvents/s}$. We observe a drop in accumulated latency for batch sizes at around 5000. For larger batch sizes, the effect of buffer latency starts to dominate, which is especially apparent for lower input event rates. Independent of the buffer size, we achieve correct flow measurements that indicate the direction of the gesture performed, as shown in an example in Figure 3.6.

3.5.3 Event-by-event Gesture Recognition

We implement and benchmark an event-based gesture recognition algorithm including background suppression for mobile phones [40]. The algorithm was trained using the NavGesture dataset [40] so that a user can perform one of 6 gestures: Up, Down, Left, Right, Select and Home. An overview of the algorithm is shown in Figure 3.7. As soon as the user presses a button, two seconds of events are recorded, at the end of which a predicted gesture is displayed in the result view. The processing happens in two stages. During the two seconds of input events, the algorithm computes a spatio-temporal descriptor called a *time surface* [85] for each event, which will be used as features during classification. The time surface represents the spatio-temporal context of an incoming event by linearly decaying events in its surroundings and encodes both static information

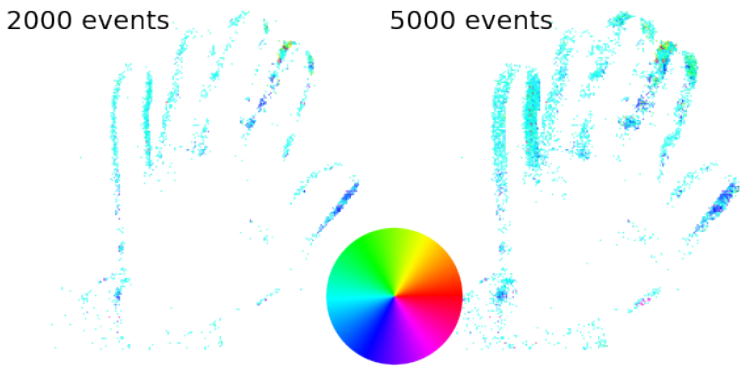

Fig. 3.6 *Aperture-robust event-based optical flow [96] computed on a recording of a person performing a mid-air hand gesture. The events are colour-coded to represent the direction of computed flow.* ***Left:*** *2000 events are taken into account when computing flow, which provides a thin outline but correctly detects the direction.* ***Right:*** *5000 events are accumulated for visualisation, equally achieving good results in terms of direction sensitivity. The motion looks blurry due to the longer time window of events.*

such as shape and dynamic information such as trajectory and speed. The time surface is then matched against learned time surfaces prototypes using a HOTS architecture [85], triggering activation for the closest matching one. This process happens continuously for the duration of the two seconds.

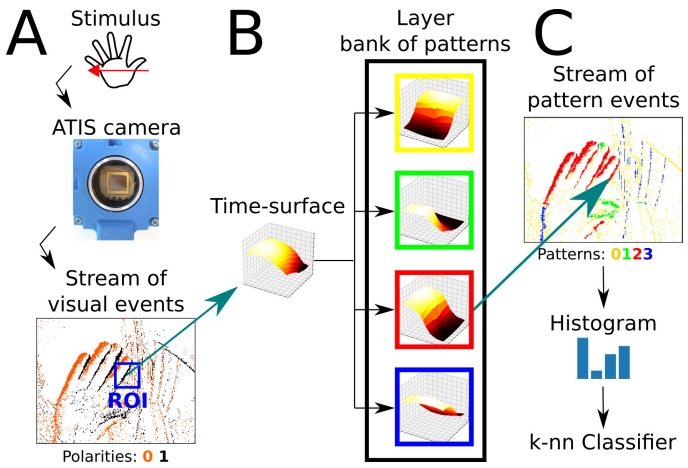

Fig. 3.7 *(A) A stimulus is presented in front of a neuromorphic camera, which encodes it as a stream of events. (B) A time-surface can be extracted from this stream. (C) This time-surface is matched against known patterns, called prototypes. The number of occurrences of each prototype can be used as a feature for classification in the form of a histogram. Figure adapted from Maro et al. [40]*

Figure 3.8 shows how the event buffer size impacts accumulated latency per second. This algorithm as currently implemented does not profit from batched operation, so we can see that latency is relatively stable. We do notice a slight overhead when buffer size approaches 1, and also see the impact of buffer latency $\mathcal{L}_{\text{buffer}}$ towards the other end. Overall, the feature generation can happen in real-time for event-rates at about 150 kEvents/s and less.

After the last feature has been generated, the second processing stage is triggered. The number of occurrences of all prototypes over a period of time is compiled into a histogram which is used as the gesture signature. The classification is done using k-Nearest-Neighbours. Here there is no option to break down or buffer the computation, so we just provide measurements of mean latency for the second processing stage of classification in Table 3.1 when no event filtering is applied. The *home* gesture with its dynamic back and forth motion causes many more events to be recorded, which has a significant impact on the time to prediction.

The filtering of input events has an impact on algorithm performance, so we plot the classification accuracy over amount of filtered events in Figure 3.9. 100% of events

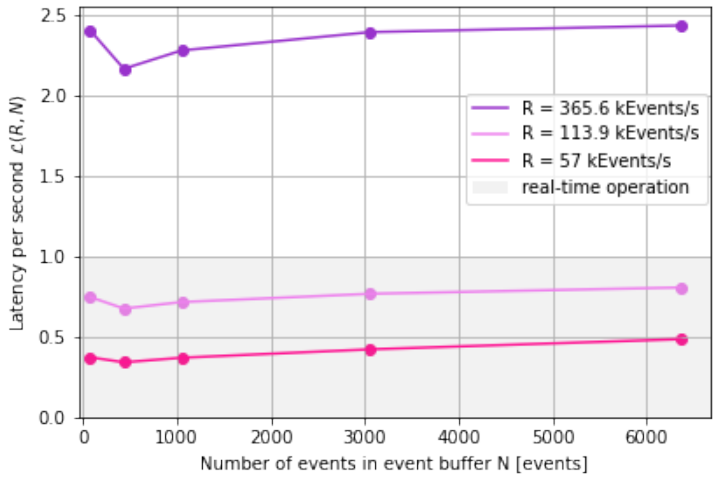

Fig. 3.8 *Accumulated latency per second when computing HOTS [85] features for classification. Features can be generated in real-time for input event rates of about 150 kEvents/s and beneath. From the measured stability in accumulated latency over batch size we can conclude that the algorithm, making use of a single thread and the NDK backend, does not benefit from batching.*

correspond to all events from the Navgesture database. We show that we can filter about half of all events without a drastic drop in accuracy.

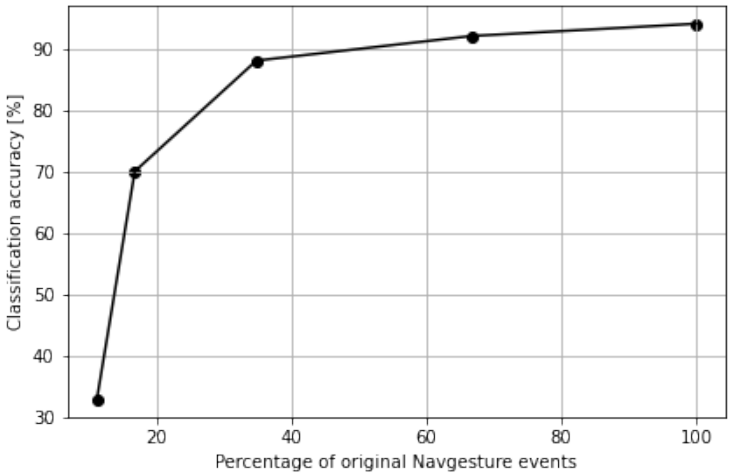

Fig. 3.9 *Event-by-event gesture classification results on NavGesture-sit [40]. By using spatiotemporal filters and refractory periods, we can reduce the amount of events and therefore computational cost considerably, without impacting classification accuracy too much.*

Table 3.1: Classification latency for 6 different gestures from the Navgesture database [40]. Mean latency is calculated over 5 trials each.

	Up	**Down**	**Left**	**Right**	**Select**	**Home**
mean latency [ms]	94.2	49	78.3	41	46.8	2825.6

Fig. 3.10 *Gray-level frame reconstruction from events using a pre-trained FireNet [211] model that has been converted to TensorFlow Lite. The two pictures differ in terms of number of events that have been used as input to the network.* ***Left:*** *3192 events are used to create a voxel grid. The reconstructed frame exhibits strong smearing artifacts.* ***Right:*** *12768 events are fed to the network, which increases latency, but also improves the visual results, for example details in the face or the door to the right.*

3.5.4 Leveraging Pre-trained Neural Networks for Image Reconstruction

We convert the pre-trained model published by Scheerlinck et al. for fast image reconstruction from events [211] to a TensorFlow Lite model that we can execute on the phone. This network has 38k parameters and uses voxel grids as input, which are accumulated and weighted accumulations of events into frames [212]. The aim is to show that we can reconstruct change detection events from the NavGesture database depending on scene activity, potentially allowing processing of conventional computer vision pipelines triggered by our inexpensive gate. It might also serve as a way to render a visually appealing live view to the user.

Depending on the buffer size and unlike for the previous algorithms, we do observe results of different visual quality that depends on the event buffer size. Figure 3.10 shows the difference between the accumulation of 3192 events fed to the network that results in an image that exhibits strong smearing artefacts and lack of detail. The accumulation of 4 times the amount of events, 12768, results in much more consistent results.

Voxel grids as a representation for events inherit some of the downsides of conventional

frames, namely an abundant amount of redundant information. This is costly to process, and therefore we need to process events in higher buffer sizes. Figure 3.11 shows the effect of buffer size on accumulated latency. Using the Tensorflow Lite backend, we can make use of special hardware acceleration, owing to the success of deep learning. It is therefore not a direct comparison to the previous two algorithms that make use of the NDK backend. A low buffer size of < 7000 triggers the processing routine very often, swamping it with additional, redundant input data that is generated from the voxel grids. When R is high, this is not sustainable for real-time operation, even with the use of dedicated hardware acceleration. A sweet spot seems to be at around a buffer size of 15000 events.

Until now we have worked under the assumption that events are evenly distributed over time, which is not the case for event camera recordings. Therefore we also want to show how many event voxel grids are generated over time for an example recording, shown in Figure 3.12. The bump in number of frames per second is caused by the gesture being performed, as the beginning and ending of the recording have little events that occur. Depending on the buffer size, this can trigger computation that refreshes the result more often than what the display is capable of rendering. Such computation could therefore be clipped to save energy.

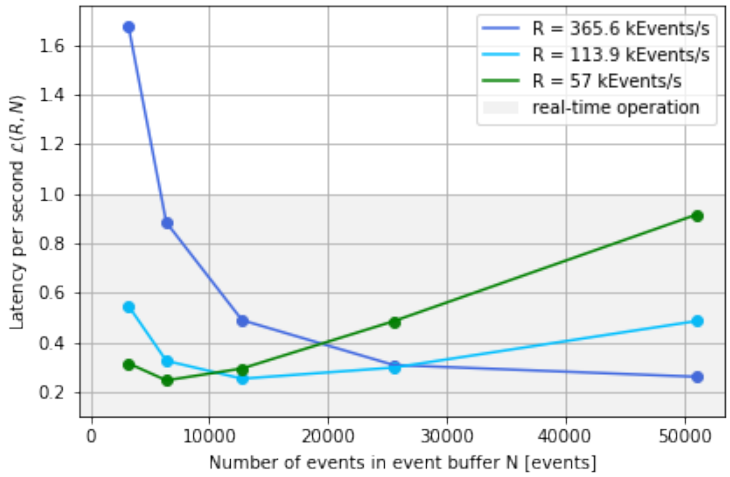

Fig. 3.11 *Latency per second $\mathcal{L}(R, N)$ over number of events in event buffer for gray-level frame reconstruction from events using the FireNet [211] neural network architecture and TensorFlow Lite backend. Using a low buffer size will trigger this expensive operation very often for high R, which is not permanently sustainable. If buffer size on the other hand is too big, no updates at all will be computed and events spend time waiting.*

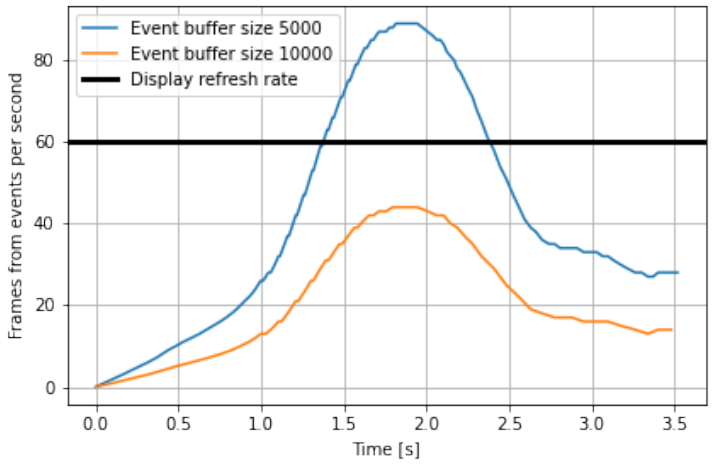

Fig. 3.12 *Event frames per second for an example gesture recording of 3.5 s and two event buffer sizes of 5000 and 10000 events. The number of events and therefore frames depends directly on the visual scene activity and is thus highest when the gesture is performed. Display refresh rate is also shown for reference.*

3.6 Discussion

This work presents a first step to integrate event cameras into mobile devices. Continuous processing of frames from conventional cameras is very costly on battery-powered devices and therefore only triggered when absolutely necessary. By swapping the conventional camera for our event camera that naturally acts as a motion detector, we can reduce computational load when there is no new information in the visual scene, and at the same time reduce latency to a minimum when computing results for fast motion. This approach is complementary to previous efforts of shrinking model sizes or algorithm foot prints.

We show that we can process input in real-time depending on different scenarios of visual activity. The algorithms we tested are subject to a trade-off between computational demand and latency. It is worth mentioning that we process event-based data via the NDK backend on a single CPU thread on conventional von Neumann hardware, which is not designed for the level of fine-grained parallelism needed in some event-based approaches. Nevertheless, our results on optical flow and gesture recognition that can be computed in real-time show the efficient nature of event-based computation. The TensorFlow Lite backend on the other hand makes use of special hardware acceleration such as the GPU to be able to reach sustainable throughput rates even though a lot of redundant information is generated in this case. In reality, our input event rate R will change continually. Even if an algorithm accumulates more than a second of latency per second, computations can

be skipped or allowed to catch up over time if input event rate drops again.

To increase the efficiency even further, one option would be to dynamically adjust event buffer size so as to minimise $\mathcal{L}(A, R, N)$. Work in this direction was done by Tapia et al. [213], although events are discarded if there are too many. Another option would be to cap computation at any rate higher than the display refresh rate. It would also be desirable to make use of a more efficient connection than USB to connect the event camera directly to the mainboard, such as Mobile Industry Processor Interface (MIPI) buses which are designed for low-power applications. Not only could the camera be integrated into the phone, but dedicated neuromorphic hardware which is specialised to execute spiking neural networks could help leverage the full potential of power-efficient computation. Event-based algorithms can make use of spiking neural networks that do not rely on the creation of voxel grids or other frame representations from events.

Connecting the world of event-based vision to a mobile device enables a range of potential applications such as face recognition, eye tracking, image deblurring, super slow-motion recordings or voice activity detection [202]. Our work opens up the route to always-on sensing on battery-powered devices that make direct use of a vision sensor and that do not have to rely on low fidelity sensors to trigger expensive computation. Neuromorphic cameras and algorithms can make current conventional systems more efficient by reacting to changes in the visual scene. The computation however is still done using von Neumann hardware. Even specialised neural network accelerators in the form of GPU-derived hardware do not exactly meet neuromorphic's demand for artificial neurons that can perform asynchronous computation. In the next chapter, we explore whether neuromorphic hardware can further reduce power costs to justify yet another piece of specialised hardware in the mix.

Chapter 4

Neural Computation on Neuromorphic Hardware Using Precise Timing

Conventional hardware, even when optimised for mobile applications, is not designed to work with event-based data. Current hardware for computer vision systems is tailored to images in the form of GPUs or other neural network accelerators. That is why we resorted to batching operations in the previous chapter to maximise throughput overall. To take computing efficiency a step further and away from conventional hardware architectures, we explore the full potential of event-based algorithms on dedicated hardware. Neuromorphic hardware starts replicating the brain from its essential components and employs thousands of spiking artificial neurons per chip. Even though we can imitate the biological hardware as such up to a certain level of abstraction, it is not entirely clear how the brain encodes information. We know of at least two major coding schemes in the brain, temporal coding and rate coding. Since we want to compute as efficiently as possible on neuromorphic hardware, we also want to use the most efficient coding scheme. Temporal neural encoding schemes generally have lower latency and employ fewer spikes than the currently dominant rate coding schemes, verified by observations of precise spike timing in biological systems. Using temporal encoding to compute with spiking neurons on neuromorphic hardware has the potential to show advantages in terms of power consumption over conventional hardware.

Spike Time Computation Kernel (STICK) is a framework that allows us to encode values and compute arbitrary mathematical systems using temporal coding in spiking neural networks. It encodes information in the interval between two spikes and provides networks for mathematical operations such as addition or exponentiation. We implement and extend STICK on Loihi, a fully digital spiking neural network processor. Since synaptic weight precision is typically greatly reduced on neuromorphic hardware, we make use of the time axis as primary means of encoding information. We show that using temporal coding on Loihi, we can combine small functional neuron blocks into larger, more complex SNNs to accurately compute arbitrary mathematical functions. We also show that we can achieve state-of-the-art classification accuracy when converting a pre-trained feed-forward artificial neural network. We compare to existing rate coding frameworks on the same hardware and

provide evidence that temporal coding can have advantages in terms of power consumption, throughput, and latency for similar numerical accuracy. Our work opens up the route for an efficient, Turing-complete computer system that might be based on neurons only, in domains such as scientific, frugal, or massively parallel computing.

4.1 Introduction

Computer programs make use of elementary operations, branching and external memory [214]. The retrieval and writing of instructions to and from memory, known as the von Neumann bottleneck, not only constricts the data traffic between processor and memory, it is also an intellectual bottleneck that has tied us to instruction-at-a-time thinking [215]. Since Moore's law and the proposed growth of transistors per area unit is slowing down as we encounter physical limitations [20], alternatives to the classic von Neumann architecture become increasingly interesting to explore [216, 217, 218].

Biological neural networks such as the brain process information significantly more efficiently than current artificial neural networks on any computer system today, using only 20 W [219]. Instead of separating the processing unit from memory which stores instructions, the brain uses its neurons for both computation and storage at the same time. Spike-based computation in conjunction with an incredible amount of recurrent connections and parallel structures makes neural information processing efficient and has the potential to outperform classical computing [220, 108]. Spiking neural networks try to emulate some of these principles to find ways to compute much more efficiently than non spike-based, classical computation. They need specialized *neuromorphic* hardware which comprises artificial neurons and synapses in silicon. Silicon as a replacement substrate to biological neurons can process and transmit signals significantly faster than what a biological system is capable of [221, 164, 167], considering the mobility of electrons in silicon is about 10^7 times that of ions in solution, therefore potentially even outperforming the brain.

Even if we emulate a brain-like system with all its components, it is not clear how information is actually encoded and decoded. Multiple neural coding schemes have been suggested to be active in the brain at the same time and their relevance for computation and learning is a topic of intense debate within the neuroscience community. When we transfer analog values into the spiking domain to compute using neurons in the hope of increased efficiency, rate coding is a dominant encoding scheme. Rate coding can be used for general purpose computation [222] and has also been used to convert ANNs into SNNs for efficient inference [122, 223] following the success of deep learning. Rate coding integrates spikes over discrete time intervals in order to correlate the firing rate of a neuron

to the strength of a stimulus. This strategy is seemingly easy to implement but relies on a large number spikes, which can be very costly on a neuromorphic system.

Spiking neurons can also adopt a *temporal coding* scheme where the relative timing of spikes is considered to carry meaningful information despite the presence of noise and have been shown to be powerful computational models [104]. Precise timing is known to be an important paradigm in biological systems [224, 225, 226] and especially so in the human brain [227, 228, 229, 230]. For instance, the information carried in spike timing can be leveraged for sound localization [231, 232, 233], fast visual processing [234] and arbitrary nonlinear function approximation via an encoding scheme that relies on the inter spike interval of spike pairs [235, 236]. Taking advantage of spike timing can increase the efficiency of a spiking neural network by requiring significantly fewer neurons and spikes for computation. We implement and extend the Spike Time Computation Kernel framework, in short STICK [237], on neuromorphic hardware called Loihi [167], which provides us with a temporal encoding scheme and networks for mathematical operations. Given the brain's incredible power efficiency, biologically-inspired architectures might one day offer a viable alternative to the classical von Neumann architecture, and various spike-based computing models are already being actively explored in that regard [238, 174].

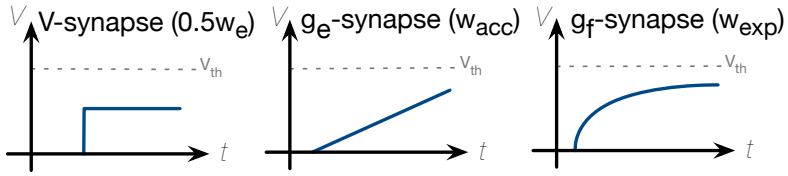

Fig. 4.1 *The effect of three different STICK synapses V, g_e and g_f and their respective weights on neuron membrane potential over time. The V-synapse injects an instantaneous current, g_e injects constant current and g_f injects an exponentially decaying current.*

4.2 STICK

The Spike Time Computation Kernel [237] encodes numbers in spike intervals and provides 3 different synapses that differ in how current is integrated into the target neuron over time. Neurons act both as local computation as well as storage units. The timing and weight of current that is injected during the time between two input spikes determines the *calculation*, where it is integrated into the target neuron's membrane potential. By turning input spikes into a membrane potential, the target neuron also acts as *storage* unit, given a large enough decay constant. Combined with the asynchronous and highly sparse nature of spike-based communication, Loihi provides us with hardware that can

unfold STICK's full potential in the hope to save power and to show the potential of a Turing complete machine on neuromorphic hardware. The neuronal units in STICK use the following model:

$$\begin{cases} \tau_m \dfrac{\mathrm{d}V}{\mathrm{d}t} = g_e + gate\ g_f \\ \dfrac{\mathrm{d}g_e}{\mathrm{d}t} = 0 \\ \tau_f \dfrac{\mathrm{d}g_f}{\mathrm{d}t} = -g_f \end{cases} \tag{4.1}$$

where V is the neuron's membrane potential, g_e represents a constant input current and g_f represents exponential current dynamics that are turned on and off by $gate \in \{0, 1\}$. τ_m is set to a much slower ($\times 1000$) time constant than other time constants such as τ_f in the system, resulting in a comparatively small leakage current. The effects of V, g_e and g_f synapses are illustrated in Figure 4.1. They have respective weights w_{e}, w_{acc} and w_{exp}.

Notably, on a discrete time axis, the precision of the encoded number and its spike interval depends on the largest chosen interval. The system's precision is inversely proportional to its speed, as increased resolution of a time interval needs more simulated time steps and more time steps to encode the same value result in slower computation. The spiking neural models in STICK also take into account a delay for the time it takes for a neuron to fire, called T_{neu}. In order to encode a real value $x \in [0, 1]$ into a time interval ΔT, STICK uses the following linear function:

$$\Delta T = f(x) = T_{\min} + x(T_{\max} - T_{\min}) \tag{4.2}$$

where $T_{\min}$ is a minimum time interval to encode the value 0 and $T_{\max}$ encodes the value 1.

4.3 Loihi

4.3.1 Hardware

Loihi is a neuromorphic research chip that implements spiking neural networks in a fully digital architecture[167]. It features 131,072 artificial neurons and 130 million synapses across 128 neuromorphic cores per chip and connections can be routed in between chips to scale up the total amount of neurons. Using Intel's 14 nm process, it achieves high-speed asynchronous computation. Continuous functions of membrane potentials are approximated in discrete time steps whereby all neurons maintain a timestamp synchronised throughout the entire system. For every neuron that enters a firing state, a spike message is sent across the mesh to the receiving cores. Those spike messages are communicated

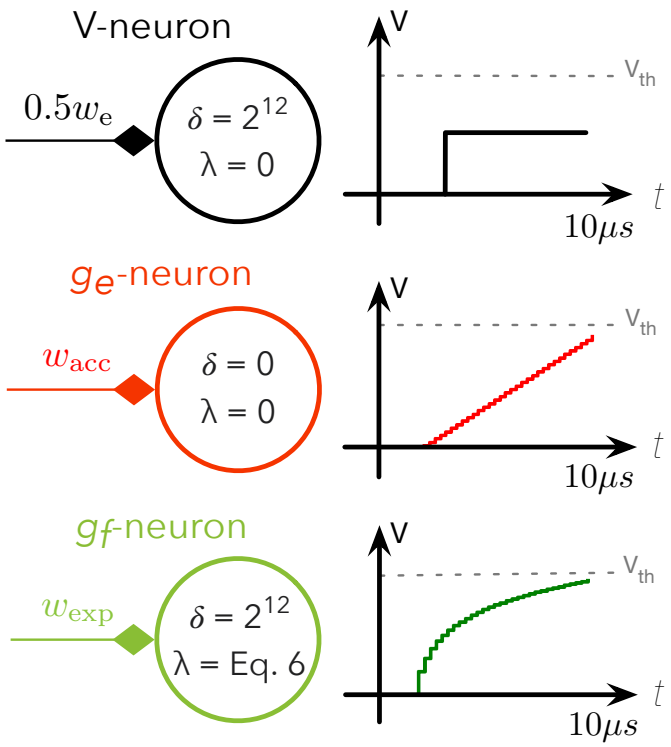

Fig. 4.2 *Three different Loihi neurons V, g_e and g_f that implement the behaviour of the respective STICK synapses. The parameters δ and λ refer to the equations on Loihi for current accumulation and membrane potential decay (Equation 4.3 and Equation 4.4).*

independently across cores, although all cores send out messages within the same time step.

Neurons on Loihi will accumulate current from attached synapses and decay it using the following model given by the hardware [239]:

$$i(t) = i(t-1)(2^{12} - \delta)2^{-12} + 2^{6+\eta} \sum_j w_j s_j(t) \tag{4.3}$$

where $\delta \in [0..2^{12}]$ is a current decay factor and $\eta \in [0..7]$ is a weight exponent scaling factor. These factors and exponents are used to approximate exponential decays in a digital system with limited resolution. w is the weight and $s \in \{0, 1\}$ the spike indicator for a connected synapse j. The summed input current is then integrated into the neuron's membrane potential, calculated as follows:

$$V(t) = V(t-1)(2^{12} - \lambda)2^{-12} + i(t) \tag{4.4}$$

where $\lambda \in [0..2^{12}]$ is a voltage decay factor. The decay is again approximated on a digital system. As soon as $V(t)$ reaches or exceeds the membrane voltage threshold

$V_{\text{spike}} = V_{\text{th}} \times 2^6$, the neuron emits a spike and V is reset to zero. Loihi also supports axonal delays σ, which schedule all outgoing spikes to arrive at a future time step and thus determine a maximum transmission delay between two neurons.

4.3.2 Neuron Models Implement STICK Synapses

We model STICK's neuron model and 3 synapse types described in Equation 4.1 and Figure 4.1 with the help of 3 different neuron models on Loihi that specify how current from synaptic inputs is integrated. The three neuron types V, g_e and g_f are depicted in Figure 4.2. The V and g_f-neurons have instantaneous current decay $\delta = 2^{12}$, whereas the current from incoming synapses at g_e-neurons does not decay, therefore $\delta = 0$ in Equation 4.3. Neither V nor g_e neuron have voltage leakage, therefore $\lambda = 0$ in Equation 4.4. In contrast, the membrane voltage of the g_f neuron decays exponentially over time following

$$V(t) = i_0 \, e^{(t_0 - t)/\lambda} \tag{4.5}$$

where i_0 is the input current at time t_0 of spike arrival. The exponential voltage decay on Loihi is approximated in Equation 4.4. We choose the voltage decay factor λ to scale with T_{max} in the following way:

$$\lambda = \frac{2^{12}}{T_{\text{max}}} \tag{4.6}$$

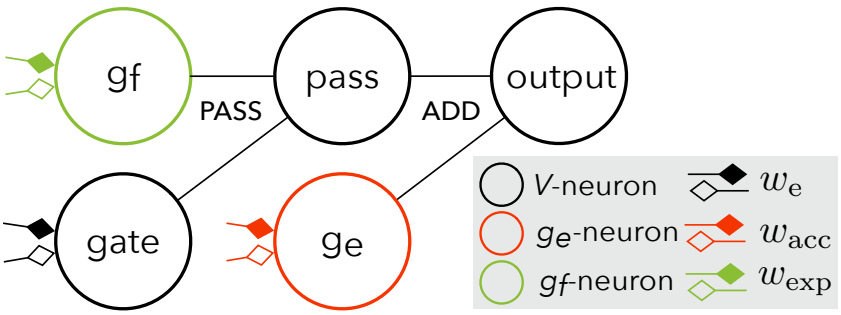

Fig. 4.3 *Dendritic tree for Loihi multicompartment neuron. It integrates voltage from a g_e- and a g_f-neuron into into a common* output *via voltage join operations* ***PASS*** *and* ***ADD*** *defined in Equations 4.7 and 4.8. The former rule makes sure that voltage from the g_f-neuron is only ever integrated when* gate *is firing and the second rule adds the constant current from a g_e-neuron to it. Coloured synapses define excitatory, empty synapses define inhibitory connections.*

Some networks need more complex current dynamics than what any single neuron could provide. We therefore connect together all three neuron types within a binary dendritic tree to form a multicompartment neuron, as shown in Figure 4.3. The goal is to combine linear and nonlinear currents from g_e and g_f neurons in a single neuron membrane potential.

The dendritic tree makes it possible to define rules on how voltage from child neurons is integrated into the parent neuron. Current integration from a g_f-neuron to the output is controlled via a *gate* and a **PASS** operation. This is formally defined as:

$$dV_{\text{pass}} = \begin{cases} i_{\text{pass}} + V_{g_f} & \text{if } s_{\text{gate}}{=}1 \\ 0 & \text{if } s_{\text{gate}}{=}0 \end{cases} \tag{4.7}$$

where i is input current, V is voltage and s_{gate} an indicator whether the neuron has spiked at the current timestamp. g_f neuron voltage will thus only be integrated when *gate* is firing. A second rule adds together *pass* and g_e neuron voltages via an **ADD** operation, formally defined as:

$$dV_{\text{output}} = V_{\text{pass}} + V_{g_e} \tag{4.8}$$

We can thus combine both exponentially decaying as well as constant membrane voltage characteristics at the *output*, which in turn connects to other neurons.

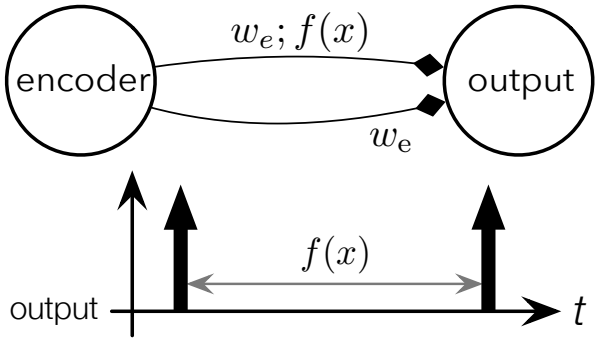

Fig. 4.4 *Delay encoder network that maps a real value x to a time interval ΔT. We define $f(x)$ in Equation 4.2.*

4.3.3 Value Encoding Using Delays

In our implementation on Loihi, we use T_{min} and T_{max} from Equation 4.2, which correspond to the number of simulated time steps for a given time interval. We set $T_{\text{min}} = 1$ time step to represent an interval that encodes the value 0 and use $T_{\text{max}} = 2^p$, where $p \in \mathbb{N}$ to be able to tune the trade-off between network speed and precision, since longer time intervals between spikes will necessarily slow down the overall system, but also be able to provide a more fine-grained resolution on the time axis. We model $T_{\text{neu}} = 1$ time steps, as it takes 1 time step to emit and propagate one spike on Loihi.

In an encoder network that uses V-neurons only, we vary the axon delay σ of an *encoder* neuron to represent our value $f(x) = \Delta T$ following Equation 4.2, as shown in Figure 4.4.

4.4 Composing Networks For Computation Using STICK

Now we turn to the process of combining the previously proposed neuron models into functional blocks, and how multiple blocks can be combined to bigger networks which might perform more complicated operations. We start with a straightforward routing network, the *Router*, pictured in Figure 4.5. A pair of spikes that are received at *input* will be routed to two separate outputs *first* and *second*. This block is often used to feed inputs to different paths in other networks.

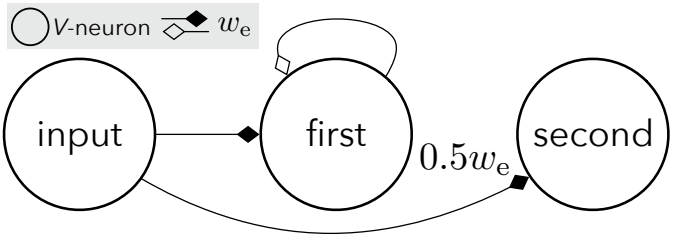

Fig. 4.5 *The router network routes two consecutive spikes at* input *into different outputs* first *and* second.

4.4.1 Storing Values

4.4.1.1 Inverting Memory

Figure 4.6 shows a simple network that can store an input spike interval x and output an interval corresponding to the input's inverse $1 - \hat{x}$. A single spike from *recall* will trigger a spike pair at *output* with an interval $\Delta T = T_{\max}(1 - x)$.

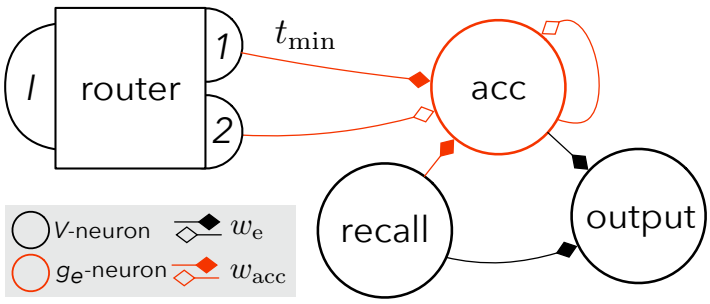

Fig. 4.6 *Inverting memory. The first input spike triggers the accumulation at the membrane of the* acc *neuron, which is stopped by the second input spike. The* recall *neuron will trigger a readout of an inverted value.*

4.4.1.2 Non-inverting Memory

A *Memory* network to help route two input spikes to different *accumulation* neurons. The *Memory* block can store a value as membrane potential of *acc2* and output it whenever *recall* is triggered. Figure 4.7 pictures the network that uses V and g_e neurons.

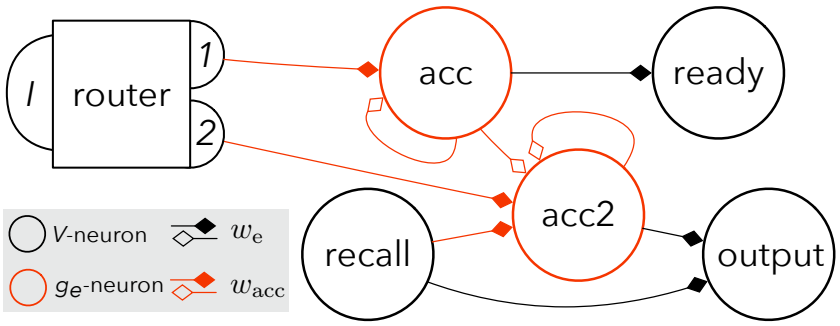

Fig. 4.7 *Memory block. This network stores an input spike interval as membrane potential of the* acc2 *neuron. The first input spike will immediately trigger a constant input current for accumulation neuron* acc*. The second input spike causes the membrane potential of* acc2 *to rise. As soon as* acc *reaches V_{th}, it signals to* ready *that the output is ready for readout and inhibits* acc2*, which now stores Δt that represents the encoded value x. A single spike from* recall *is enough to reproduce the original spike pair at* output*.*

4.4.1.3 Signed Memory

Using the *memory* block, we can create a network shown in Figure 4.8 that can store spike intervals for positive and negative numbers by adding further neurons that represent the number's sign. Depending on which input path a spike pair is received, we then interpret the value as positive or negative.

4.4.1.4 Synchroniser

Figure 4.9 shows a network that can synchronise n inputs, so that the first spike of all outputs will be aligned. Making use of N *memory* blocks, the last block to receive its input will trigger a *sync* neuron to start the synchronous readout.

4.4.2 Branching Operations Minimum and Maximum

Using a combination of excitatory and inhibitory connections shown in the Minimum network in Figure 4.10, we can detect the smaller of two synchronised inputs as soon as the second spike thereof arrives. The *output* neuron will emit the smaller input and the

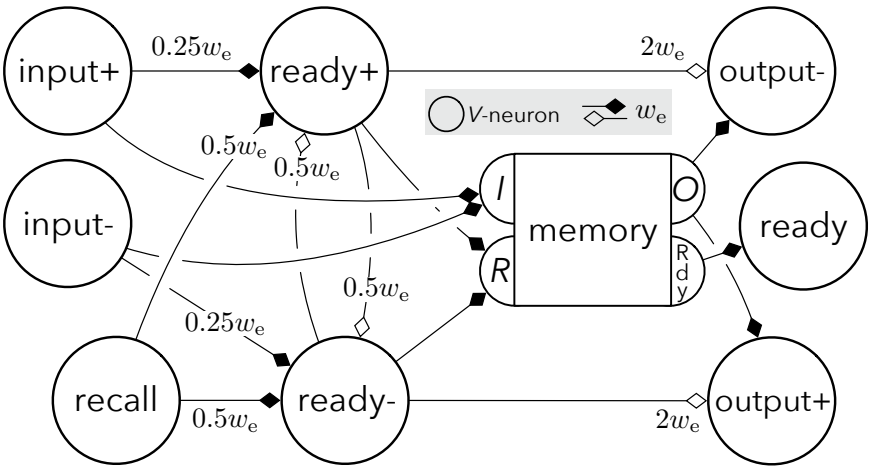

Fig. 4.8 *Signed memory. This network employs additional neurons on top of the memory network to signify positive or negative stored values.*

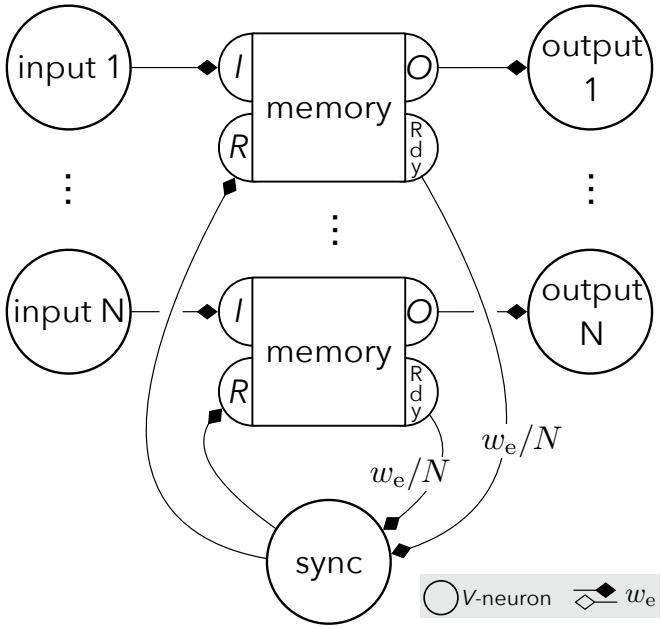

Fig. 4.9 *Synchroniser. This network can store N inputs that are received at arbitrary times in* memory *units and triggers the synchronous readout of all cells as soon as the last input is received.*

presence of a spike at either *smaller1* or *smaller2* will mark which input was the smaller one.

The Maximum network shown in Figure 4.10 will indicate the larger of two synchronised inputs as soon as the smaller input has fired its second spike. This will be indicated on

either *larger1* or *larger2*. The *output* neuron then mirrors the larger of the two inputs afterwards.

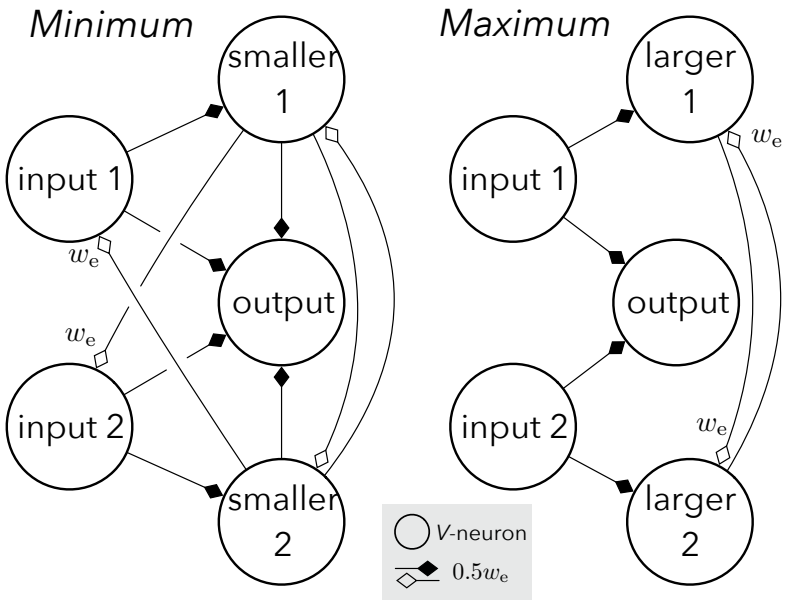

Fig. 4.10 *Minimum and maximum branching operations for two inputs. The first complete input will signify in both networks which of the two inputs is the smaller one. For these networks to work, the inputs have to be synchronized.*

4.4.3 Linear Operations

4.4.3.1 Subtractor

Figure 4.11 shows a network that computes the difference between two synchronised inputs. The difference between the two spike time intervals will be another interval, but may be output to either *output+* or *output-*, depending on the sign. We also use an additional *zero* neuron that fires if the inputs are equal.

4.4.3.2 Linear Combination

We can compute the linear combination of n different inputs given the coefficients $\alpha_0, \ldots, \alpha_n$ using the network shown in Figure 4.12. Each input will either be directed to a positive or negative input path, depending on its sign. Neuron *acc1+* accumulates values for all positive inputs, whereas *acc1-* does the same for all negative inputs. The difference between the two accumulated values is then synchronised and subtracted. A *start* neuron indicates the result being ready for readout.

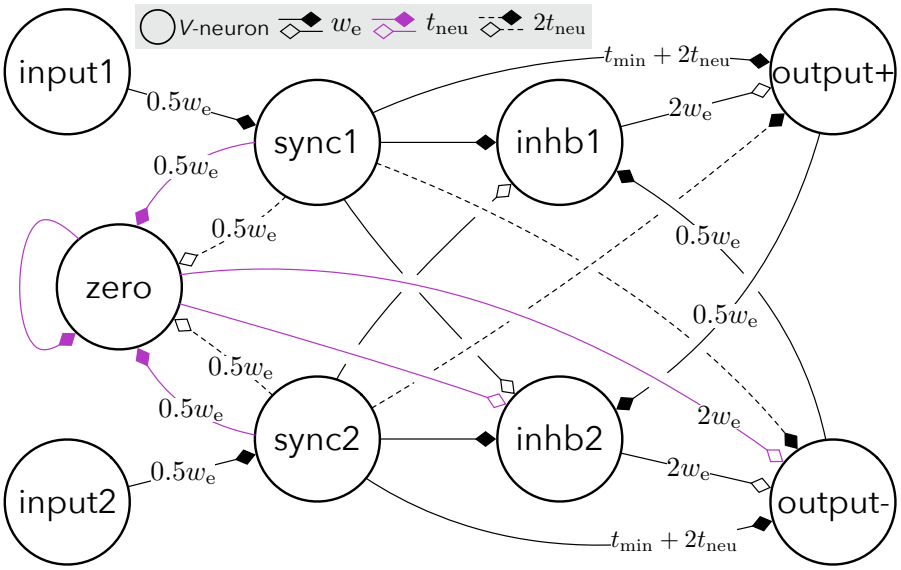

Fig. 4.11 *Subtractor. Connections related to the* zero *neuron that have a delay of T_{neu} are colour-coded for better visibility.*

4.4.4 Nonlinear Operations

4.4.4.1 Logarithm

We might also exploit the more complex characteristics of a multicompartment neuron to compute the natural logarithm of an input $f(x)$, as shown in Figure 4.13. The first input spike triggers g_e current characteristics, whereas the second input spike triggers the nonlinear g_f neuron. That way the network can output a spike interval as follows:

$$\hat{x} = \ln \frac{f(x)}{T_{\max}} \tag{4.9}$$

4.4.4.2 Exponential

When we exchange the sequence of nonlinear and linear current accumulation in the Natural Logarithm network, we are able to calculate the exponential of a given value x. The network is shown in Figure 4.14. The first input spike connects to *decay* and *gate* compartments and starts nonlinear accumulation. The second input spike stops the *gate* from spiking, thus inhibiting any further influence of the *decay* compartment on *multi*. It also triggers the first spike on *output*, which outputs a spike pair corresponding to:

$$\hat{x} = e^{\Delta t / T_{\max}} \tag{4.10}$$

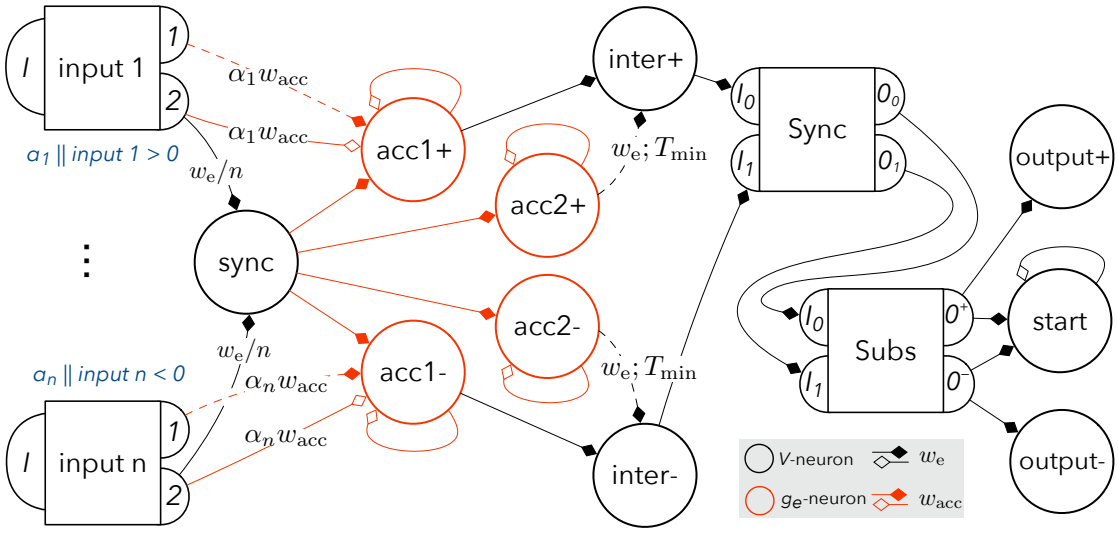

Fig. 4.12 *Linear combination. Amount of neurons is* $6n + 40$*, where* n *is the number of inputs. The overall network consists of accumulation parts for positive and negative coefficients plus a synchroniser and subtractor sub network.*

4.4.4.3 Multiplier

We can combine Natural Logarithm and Exponential networks to provide the product of two inputs x_1, x_2 as follows:

$$\hat{x} = x_1\, x_2 = \exp\left(\ln x_1 + \ln x_2\right) \tag{4.11}$$

On arrival of the very last spike of the two inputs, both values x_1 and x_2 have been loaded onto the membrane potentials of *acc log1* and *acc log2* respectively. The *sync* neuron triggers accumulation on *exp*. Its *gate* has to be deactivated after a time corresponding to the sum of the natural logarithm of the two inputs in order to obtain their product.

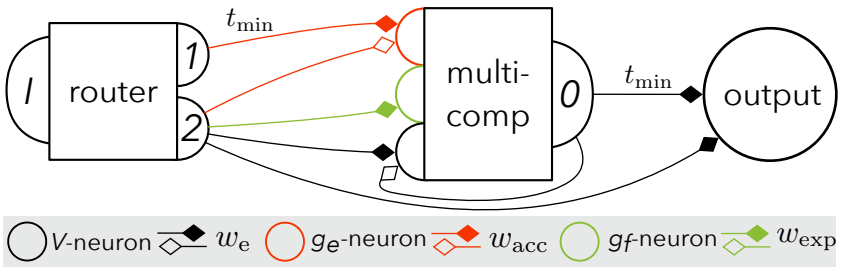

Fig. 4.13 *Natural logarithm. The first spike to arrive will trigger accumulation on neuron* multi*, which will be stopped by the second spike. The second input spike will also immediately trigger the first spike at* output*.*

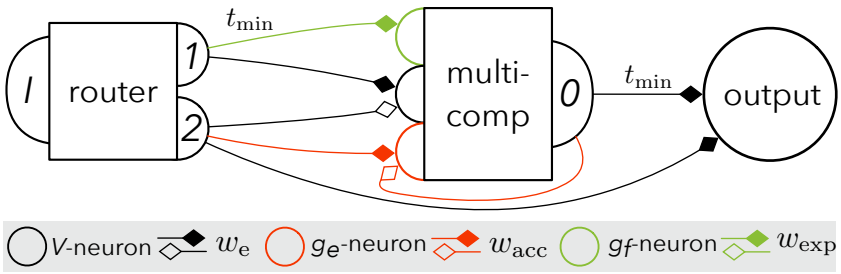

Fig. 4.14 *Exponential. The first spike to arrive will trigger nonlinear current accumulation on neuron* multi*, which will be replaced by linear current integration as soon as the second spike arrives. The second input spike also triggers the first* output *spike.*

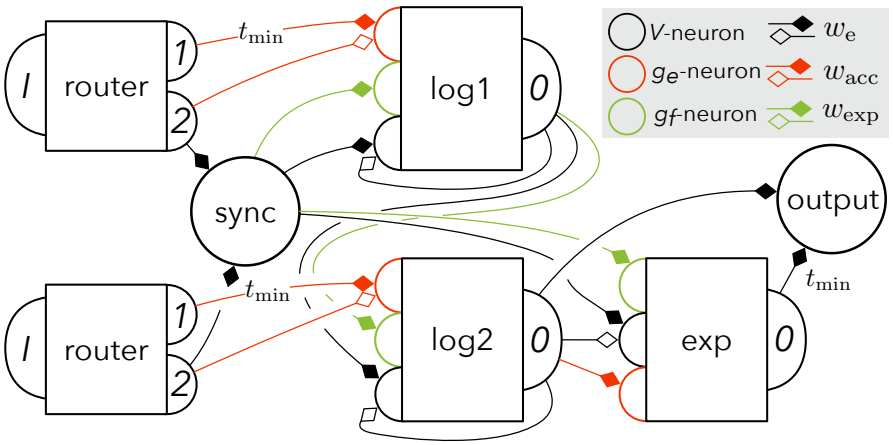

Fig. 4.15 *Multiplier. The two inputs are accumulated in logarithm networks* log1 *and* log2*. Their sum is then fed to an exponential unit* exp*, which outputs the product of the original inputs.*

We implement all the above networks, which we re-parameterize for the sake of our implementation, on Loihi. Overall, the networks provided by STICK allow us to compute arbitrary mathematical systems in an asynchronous manner.

4.4.5 ANN-SNN Network Conversion

To build a spike-based computer that is able to solve increasingly complex task, we also investigate the conversion of trained neural network graphs onto Loihi using the STICK framework. After all, neural network inference of converted models on neuromorphic hardware bears the promise to decrease power cost of deep learning models. Most current ANN to SNN conversion techniques are based on rate coding [240, 122, 241, 223, 242], which is straightforward to implement, robust to firing errors and propagates a signal

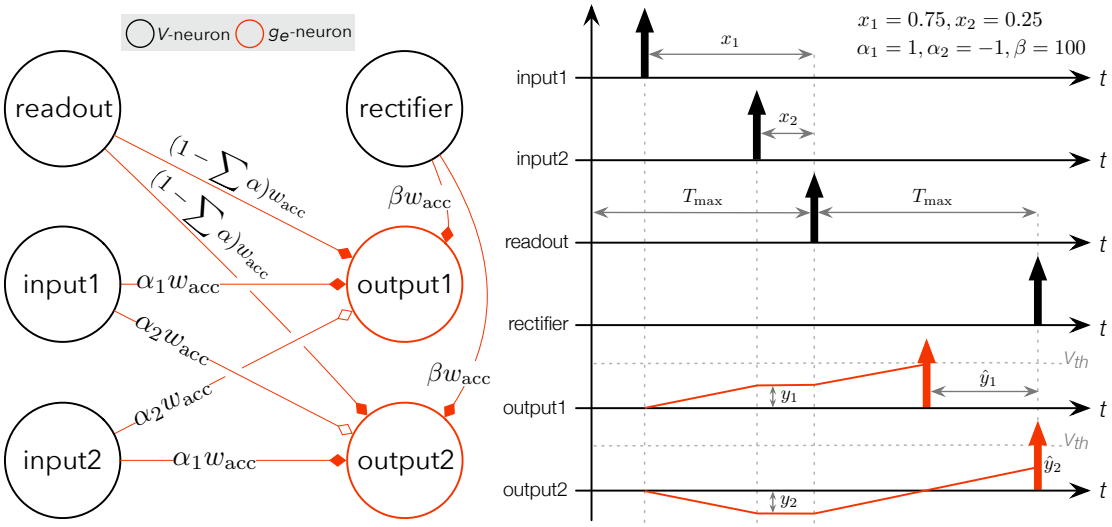

Fig. 4.16 *Conversion of 2 ANN units to an SNN on Loihi, using V and g_e-neurons.* ***Left:*** *The weights in the ANN can be directly used for synaptic weight coefficients in the SNN, in this example $\alpha_{\{1,2\}}$. Inputs $x_{\{1,2\}}$ are encoded using latency coding. The* readout *neuron ensures that at the end of T_{max} time steps for one layer, all input currents for a neuron are balanced by injecting a current that is the negative sum of all inputs plus w_{acc}. The readout time converts the value that is encoded in the membrane potential into a spike interval for the next layer. The* rectifier *injects a large current with $\beta \gg \alpha$ to force a neuron to spike if it hasn't yet at the last time step of a layer. The neuron* output1 *computes $\hat{y}_1 = max(0, \alpha_1 x_1 + \alpha_2 x_2)$, whereas* output2 *computes $\hat{y}_2 = max(0, \alpha_2 x_1 + \alpha_1 x_2)$.* ***Right:*** *Chronogram of the same network, with example inputs $x_1 = 0.75, x_2 = 0.25$ and weights $a_1 = 1, a_2 = -1$. Whereas* output1 *outputs the expected $1 \times 0.75 - 1 \times 0.25 = 0.5$,* output2 *with $1 \times 0.25 - 1 \times 0.75 = -0.5$ is forced to spike early by injecting a high current at time step $2T_{max}$. This spike coincides with the* readout *of the next layer (not shown here), where their effects will cancel out because the output is 0. For this diagram T_{max} is assumed to be large so that transmission delays T_{neu} are negligible.*

fairly quickly through the network. It can however exhibit issues with propagating the signal to deeper layers, which scales unfavourably for larger networks. Rate coding can therefore 'starve' downstream parts of a network, while still employing millions of spikes per inference [243, 122].

In contrast, conversion frameworks based on temporal coding have made significant advances as of late, at much fewer spikes in comparison to rate coded SNNs [244, 245, 246]. Time To First Spike (TTFS) uses the least amount of spikes, leading to great energy

efficiency and possibly achieving very low latency. It is however very sensitive to the order of inputs and needs large refractory periods to ensure a single spike per unit. Most TTFS methods also use dynamic neuron membrane thresholds to prevent early firing [130, 247]. Such mechanisms can be used in SNN simulators, but are not directly transferable to neuromorphic hardware or are at least very costly to execute. Using STICK, we rely on the structure of the network itself, using neurons and synapses only without the need for dynamic threshold adaptation.

Figure 4.16 shows the network diagram and chronogram of an SNN that has been converted from two ANN units. We can make direct use of the ANN's weights to connect the units in our SNN. Every layer computes over $T_{\max}$ time steps, so that we can choose a desired trade-off between accuracy and latency. In addition, every layer will have 2 additional neurons *readout* and *rectifier* which trigger the readout for the next layer and ensure that no negative membrane potentials are decoded. Because ANNs typically have very large numbers of neurons, we choose an optimised approach for this task. Since layers (including the inputs) in an ANN are synchronised, we employ the same principle in our converted SNN blocks and align all input spikes in a layer along the time step of the *readout* neuron for that layer. The *readout* neuron connects to every neuron in its layer with a negative sum of all input weights for that neuron to balance the input current, plus 1 to trigger the readout. Thus, for a minor cost in synapses (1-2 additional per neuron in comparison to rate coding), we now have a reference spike for each layer that triggers the readout at pre-defined times and converts the membrane potential that reflects the stored value at that point again in to a spike interval: $u(t) \propto x, t \in nT_{\max}$, where $n \in \mathbb{N}$. The main difference to other TTFS methods is the balancing second spike that acts as a counterweight, which provides us with better accuracy. The fact that we have counterweights allows us to use the same membrane threshold for all accumulating compartments and readout with a pre-defined input current. Conversion methods based on similar temporal coding schemes [248, 249, 250] have achieved very good accuracy, albeit not yet on neuromorphic hardware.

In summary this section presented the tools to cast arithmetic operations into spiking neural networks on Loihi. This is the most crucial part of any computing system. In the following results section, we will look at how SNNs that use temporal coding perform when compared against rate-coded networks.

4.5 Experiments and Results

We begin by showing that the numerical precision of networks using precise timing on Loihi is more than sufficient to compute complex dynamic systems and extensive converted

graphs of neural networks. Being able to manipulate numbers is a key element for computing machines, alongside branching operations.

4.5.1 Computing Dynamic Systems

The higher-level network composition for first order, second order and Lorenz system are conceptually equal to the ones in [237], but is now based on our implemented networks on Loihi. The basic building blocks are shown in section 4.4. We compare the output to another system of spiking neurons with the same parameters on the same hardware using population coding, using support of the Nengo framework [251] with Loihi as a backend. Neural Engineering Framework (NEF) and by extension Nengo [252, 251] take the mean firing frequency of a homogeneous population of neurons instead of relying on a single rate-coded neuron for computation. Population coding can be considered more biologically plausible due to the added stability [253] and the ability to quickly respond to changes in the input information [254], but, while computationally powerful, relies on a large number of neurons as well as spikes.

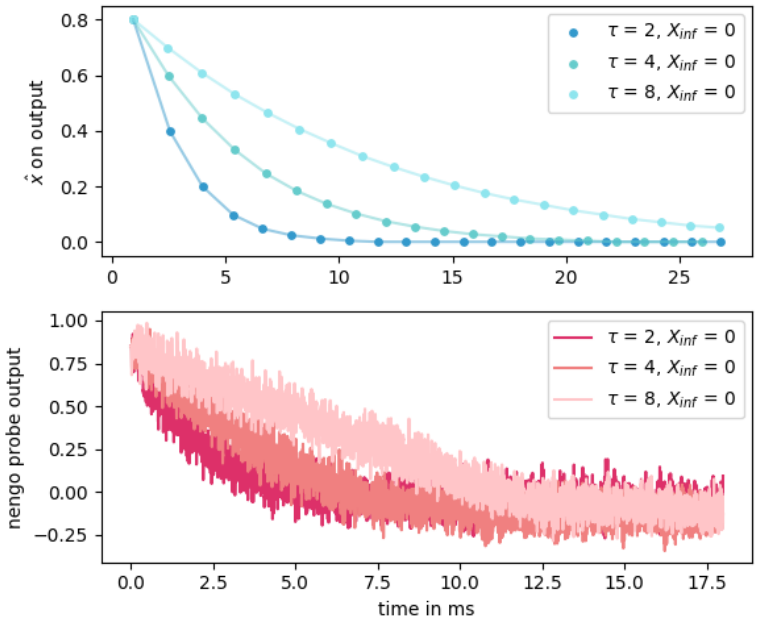

Fig. 4.17 *Outputs of a first order system networks using precise timing and population coding for different* $\tau \in \{2, 4, 8\}$. *Both networks use 117 neurons.* ***Above:*** *our network with* $T_{max} = 2^6$. ***Below:*** *Nengo network across 30000 time steps.*

4.5.1.1 First Order System

We demonstrate general purpose computation capabilities by calculating a first order system

$$\tau \frac{\mathrm{d}X}{\mathrm{d}t} + X(t) = X_{\mathrm{inf}} \tag{4.12}$$

We combine an encoder that outputs X_{inf}, a linear combinator that computes $\mathrm{d}X/\mathrm{d}t$ and an integrator that computes X from its derivative. The composite network comprises 117 neurons.

We evaluate the amount of time steps it takes for the system to reach a steady state and the error for each output value. The network is capable to reliably compute different function parameters τ, shown in Figure 4.17 on top. Using the same amount of neurons in population coding to compute the same system, the signal is much noisier as shown in Figure 4.17 at the bottom.

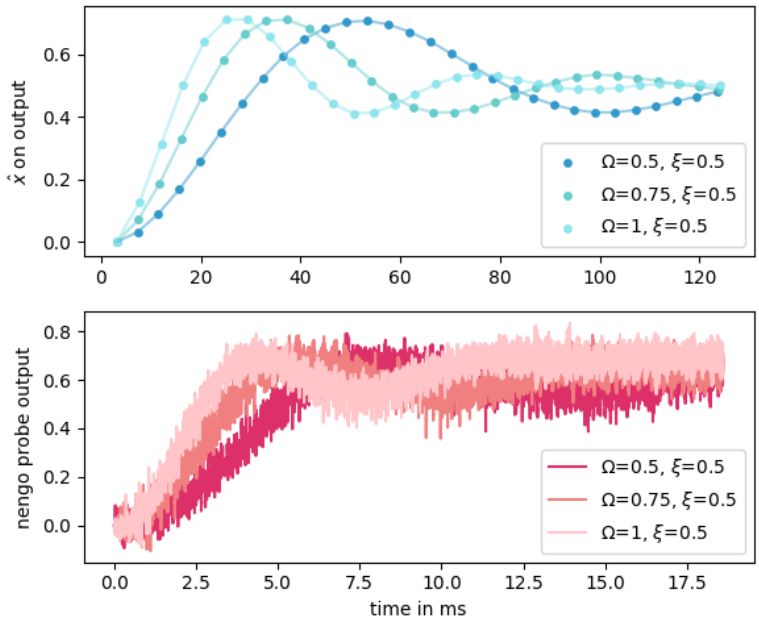

Fig. 4.18 *Outputs of second order system networks using precise timing and population coding for different parameters Ω and ξ. Both networks use 185 neurons.* ***Above:*** *STICK network with $T_{max} = 2^8$.* ***Below:*** *Nengo network across 30000 time steps.*

4.5.1.2 Second Order System

In order to increase the complexity, we also compute a second order system

$$\frac{1}{\Omega^2}\frac{\mathrm{d}^2X}{\mathrm{d}t^2} + \frac{\xi}{\Omega}\frac{\mathrm{d}X}{\mathrm{d}t} + X(t) = X_{\text{inf}} \tag{4.13}$$

The results of which can be seen in Figure 4.18. Again our network using precise timing is able to compute a much cleaner output as a population-coded network, using the same amount of 185 neurons. Our network does need more time steps for the error to stay low. It takes 126 ms of spiking time for a T_{max} of 2^8, whereas the Nengo network can be simulated within 18 ms.

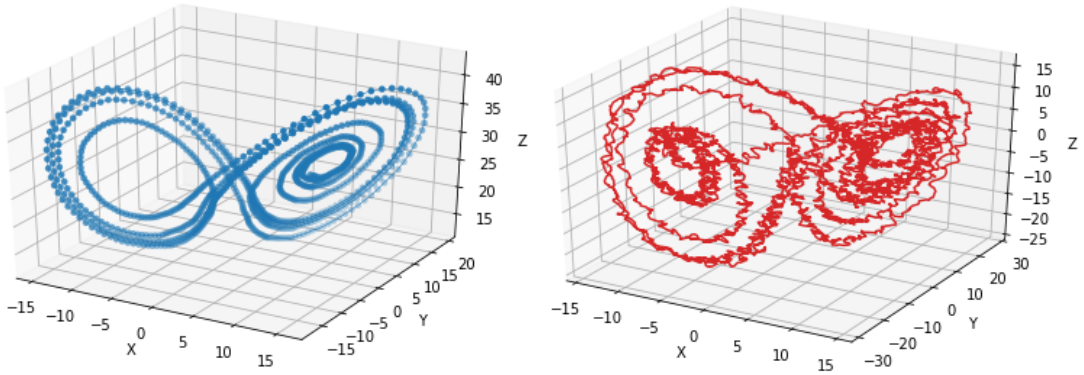

Fig. 4.19 *Outputs in X, Y and Z for Lorenz system networks computed with precise timing and population coding. Both networks use parameters $\rho = 28$, $\sigma = 10$, $\beta = 8/3$.* ***Left:*** *our network uses 598 neurons and takes 20 ms to compute using $T_{max} = 2^{10}$.* ***Right:*** *Nengo network computed across 100000 time steps.*

4.5.1.3 Lorenz System

So far we have only used linear buildings blocks such as linear combinations and subtraction to differentiate and integrate over time. We also implement a Multiplier network (shown in Figure 4.15), which combines nonlinear networks to provide the product of two inputs. To show the fine granularity that our non-linear systems are capable of, we compute an example of a system of ordinary differential equations that relies on a multiplicative factor, known as the *Lorenz attractor*. It is defined as follows:

$$\frac{dX}{dt} = \sigma(Y(t) - X(t)), \tag{4.14}$$

$$\frac{dY}{dt} = \rho X(t) - Y(t) - X(t)Z(t), \tag{4.15}$$

$$\frac{dZ}{dt} = X(t)Y(t) - \beta Z(t), \tag{4.16}$$

The system has chaotic solutions given certain parameters and we choose $\rho = 28$, $\sigma = 10$, $\beta = 8/3$. As can be seen in Figure 4.19, we can faithfully replicate the chaotic behaviour using just 598 neurons. The Nengo network on Loihi exhibits strong stochastic behaviour and has troubles with synaptic weight quantisation on Loihi, leading to a largely reduced output precision even though it uses 3000 neurons. Using fewer neurons in this population coding implementation fails to replicate the Lorenz system completely.

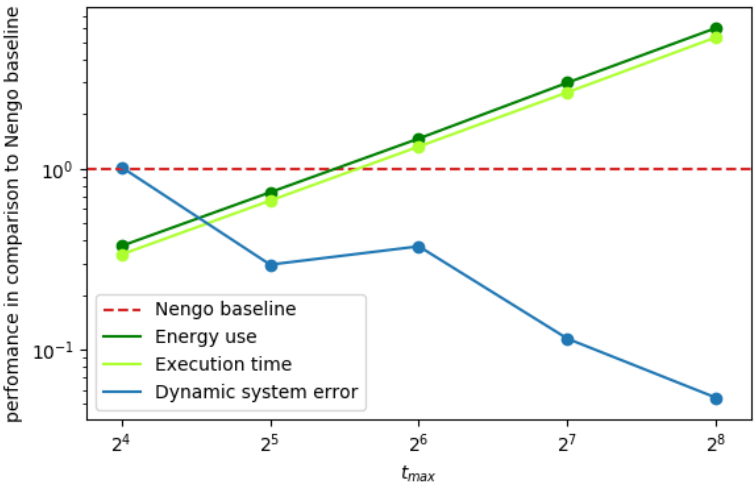

Fig. 4.20 *Performance comparison between proposed networks and Nengo implementation for 3 dynamic systems. For lower T_{max}, precise timing is able to compute faster and using less energy, with comparable error performance. Number of neurons is always equal.*

4.5.1.4 Dynamic System Performance

We benchmark the performance of dynamic systems shown in this section that use precise timing and compare it to a baseline of a Nengo implementation using population coding on Loihi. By varying $T_{\max}$, we can see the effects of speed-up versus average error of the system, shown in Figure 4.20. Decreasing $T_{\max}$ will calculate the same system faster, but will do so with a more coarse-grained resolution on the time axis, which causes the error to increase. Overall, networks using precise timing do have a region in parameter space

where they can compute both faster and more precise, given the same amount of neurons. Increasing computation precision takes both more time and energy.

4.5.2 Converting Pre-trained ANNs

We also benchmark the conversion of an ANN which had previously been trained on a GPU to an SNN using the MNIST image classification task. We show that we can successfully deploy converted models on constrained neuromorphic hardware and provide evidence that STICK's temporal encoding has several advantages in comparison to the dominant rate-coding conversion scheme. The conversion framework as well as the pre-trained model will be made publicly available* and example code is available in Appendix A.4.

4.5.2.1 Training and Converting the Graph

For MNIST, we choose to convert a convolutional architecture with several channels, followed by max-pooling layers. The converted network is illustrated in Figure 4.22 and is trained with a fixed learning rate of 0.001, the ADAM optimiser [255] and dropout regularisation [132]. We also apply important loss penalties on all activations and weights to prevent outliers, which narrows the parameter distributions. When converting the trained weights for the SNN, previous work [240, 122, 242] has demonstrated the importance of data-based weight normalisation, when jointly scaling the weight and bias parameters in each layer. In heavily-tailed parameter distributions, we can focus on a given percentage of the distribution mass to help increase the dynamic range of weights we can map onto Loihi. For our conversion, we choose the 99.9^{th} percentile of activations as a target for our scaling process. Biases are encoded using delays, as exemplified in Figure 4.4.

With the help of our ANN-SNN conversion framework, we can recreate convolutional, fully-connected and max-pooling layers. Convolutional as well as fully connected layers are comprised of one g_e-neuron for each unit in the ANN plus one V encoder neuron that provides the biases for every output channel. For convolutional layers, we implemented the possibility for different kernel sizes, strides, padding and groups. Max-pooling can be easily achieved by connecting groups of neurons from the previous layer using V-neurons.

4.5.2.2 Classification Accuracy

In Table 4.1 we list literature results and our results that have been implemented and benchmarked on spiking hardware, to allow for a fairer comparison. We achieve near state-of-the-art classification accuracy at a cost of one spike per neuron. Figure 4.21 shows the trade-off between more accurate versus faster computation, which is controlled using

*https://github.com/biphasic/Quartz

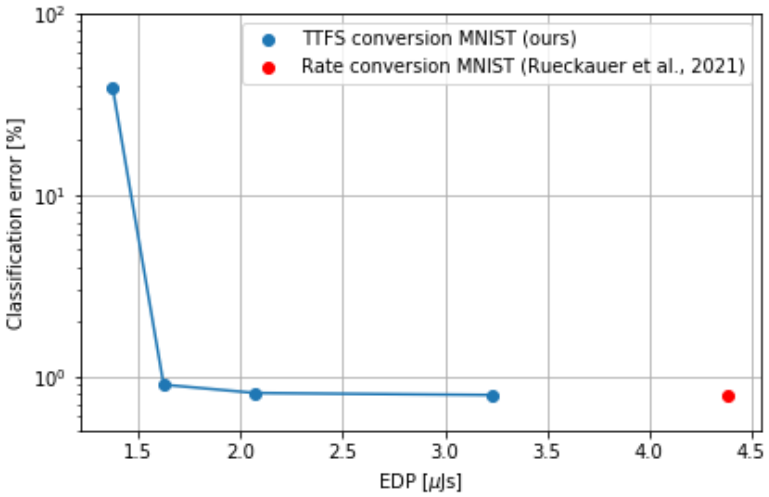

Fig. 4.21 *Classification error plotted over Energy Delay Product (EDP) for MNIST in comparison to Rueckauer et al. [242] measured on the same platform.*

Table 4.1: Comparison of accuracy and performance to other SNNs for a classification task on MNIST. Sorted after publication date.

Method	**SNN error [%]**	**# spikes**	**# neurons**	**spiking hardware**
Rate [256]	1.3	-	1306	Simulated 28 nm
TTFS [257]	3.02	135	1394	FPGA
Rate [258]	10	-	316	Simulated 65 nm
Rate [259]	1.4	130k	2330	10nm FinFET
TTFS [260]	3.1	162	1000	Simulated 0.35 µm
Rate [261]	1.3	-	8266	Loihi
Rate [242]	0.79	-	4k	Loihi
TTFS **(ours)**	0.9	5422	5422	Loihi

the $T_{\max}$ parameter in our network. While our trained ANN exhibited a classification accuracy error of 0.73%, we achieve a range of errors in our SNN depending on the latency from 38.79% to 0.79%. We observe a sweet spot of 0.9% classification accuracy error with a delay of 6.3 ms when using $T_{\max} = 2^4$. For this parameter setting, the first spike in the last layer is received after 67 time steps on average, whereas the full presentation takes 90 time steps.

Table 4.2: Breakdown of static and dynamic power consumption per MNIST classification inference for neuromorphic cores and Lakemont x86 CPUs as well as latency on Loihi. Setting $T_{\max} = 2^4$ results in 0.9% classification error for our converted network.

Power consumption [mW]	x86	Neuron	Total
Static	0.14	8.7	8.84
Dynamic	23.4	8.4	31.8
Total	23.54	17.1	40.64
Latency [ms]		6.3	
Time steps per inference		90	
Energy per inference [µJ]		256.7	
EDP [µJs]		1.62	

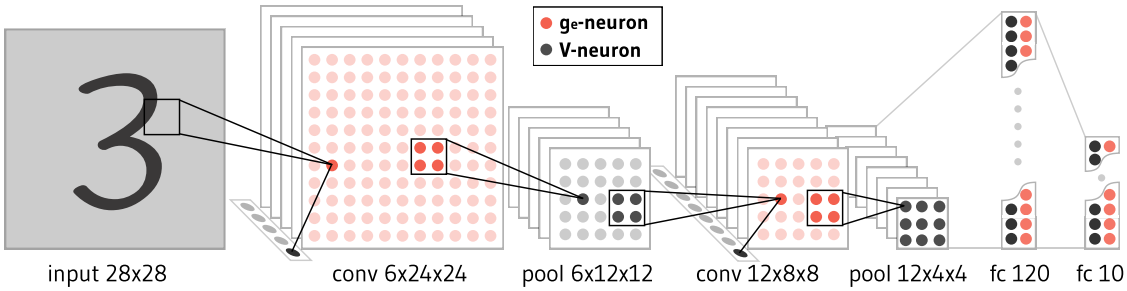

Fig. 4.22 *Converted spiking neural network architecture with convolutional (conv), max-pooling (pool) and fully connected (fc) layers based on V and g_e STICK neurons. The input is encoded using latency coding.*

4.5.2.3 Power Measurements

NxSDK is a library to configure networks on Loihi and it provides tools that break down power draw for workloads into dynamic and static components. This is done for neuromorphic cores, which store neuron states and emit spikes, and the x86 Lakemont CPUs which are responsible for executing user scripts. Static power consumption is independent of the workload and depends to a great extent on the number of active components and the manufacturing process. As such it is not relevant for comparing algorithm performance, because we typically want to benchmark workloads. Dynamic energy is consumed by switching transistors to update neuron states and route spikes. It does not depend on the time it takes to compute one algorithmic time step. A complete table that breaks down static and dynamic power consumption across neuromorphic cores and Lakemont CPUs is provided in Table 4.2. Our converted SNN model that uses $T_{\max} = 2^4$ time steps per layer occupies 18 of the 128 neuromorphic cores on one

chip and consumes 256.7 µJ of energy per inference sample for both neuromorphic cores and Lakemont CPUs, of which 200.34 µJ is dynamic energy. A preferred metric when benchmarking neuromorphic algorithm performance on hardware is the product of energy consumption and latency, the Energy Delay Product (EDP) [262]. For our experiment it amounts to 1.62 µJs. All measurements were obtained using NxSDK 0.9.8 on a Nahuku board with 32 chips. For comparison, the best-in-class rate-coded model on Loihi has an EDP of 4.38 µJs at a classification error of 0.79 %, and a GPU that computes using single batches achieves an EDP of 222 µJs for an error of 0.73 % [242].

4.6 Discussion

In this paper we show that temporal coding can have advantages over rate-coding in terms of power consumption and numerical accuracy when performing computations using artificial neurons on neuromorphic hardware. Nowadays the search for alternative mechanisms of computing becomes increasingly viable, as von Neumann computing is encountering physical limits. Apart from quantum computing, neuromorphic computing is one such alternative avenue. The dominant rate-coding scheme often means employing a large number of neurons and spikes, which can scale unfavourably for bigger networks. The Spike Time Computation Kernel [237] framework provides us with an alternative based on temporal coding. By means of concrete examples, we compare the performance of temporal versus rate encoding in terms of power, latency and numerical accuracy. The digital architecture of Loihi allows us to compute fully deterministic results.

For a set of general purpose dynamical system tasks that include nonlinear dynamics, we show that temporal coding can compute much more accurately in comparison to population coding, using an equal amount of neurons. NEF has been designed with biological plausibility in mind, so the results are noisy. The accuracy of the approximation depends not only on the neural properties but also on the functions being computed [263]. An increase in population size of neurons in Nengo normally leads to better and less noisier results using a normal computer as a backend, but the quantisation of weights and time on Loihi seems to pose a problem for that framework, which normally relies on fine-grained neuron tuning curves available on conventional hardware. Increasing the time step resolution in Nengo as suggested in [264] did not have a positive effect on the network's performance but only lead to longer execution times in our experiments. STICK does not rely as much on synaptic weight resolution and can therefore output less noisy results on this digital neuromorphic hardware, even if that means that the output is much coarser over time.

Not only do we evaluate static blocks that can be combined to compute more complex

systems, we also look at converting graphs of learned neural networks for efficient inference on Loihi. Again, nowadays rate coding is the dominant conversion technique from ANNs to SNNs. We show that SNNs that use temporal coding have a number of advantages or are at least on par with rate coded networks:

- In order to propagate spikes to the deeper layers, neurons do not need to undergo a strong transient response resulting in uneven firing rates across time.
- Value signals that are passed from layer to layer do not deteriorate or starve in deeper layers since all values are encoded using the same number of spikes using TTFS encoding. Only the timing changes.
- Naturally suitable for max-pooling.
- Number of spikes is drastically reduced and not dependent on the number of layers.
- Equal capabilities to use bias conversion or batch normalisation folding.
- The input signal propagates through the network like a wave and each layer's activities are decoupled from each other. That means that the input signal does not have to be presented for an extended time to arrive in later layers.
- Ability to represent negative values when spikes of an interval originate from different neuron sources and can therefore encode a *negative interval*. This typically helps with the last layer of a network that outputs logits.
- No *soft reset* of membrane spike threshold needed to achieve good results.

Rate-coded converted networks present a large number of input spikes for extended duration at the first layer, which causes a strong transient response in early layers to propagate the signal through the network, but might still not be enough to reach deep layers because neurons that output low values spike less frequently. Layers in a converted SNN using our method sequentially process the input, following the functional principle of an ANN. The spiking activity in different layers can thus be decoupled from that of another layer once the spikes have been received or transmitted. That allows us to feed new input at the beginning of the network while later layers are still computing a previous input, considerably increasing throughput. We achieve a form of *temporal batching* which is independent of network depth. A further consequence of the decoupled layers is that we can choose different time constants for each layer. Since errors in early layers that necessarily occur because of quantisation will accumulate for downstream layers, we could choose a larger $T_{\max}$ in the first layer and gradually reduce it in deeper layers. We are on par with other conversion methods when it comes to the number of neurons used for the conversion, apart from 2 support neurons per layer. These support neurons *readout*

and *rectifier* are responsible for an additional amount of synapses in the network, as they connect to every neuron. This however suits Loihi's architecture, which is designed to have many more synapses than neurons.

Massa et al. [261] and Rueckauer et al. [242] allow for the most direct comparison for image classification performance on the same hardware. Whereas Massa et al. only provide classification results, Rueckauer et al. provide power benchmarks for static and dynamic power combined without breaking down the individual components. Our EDP when using the same metrics is a factor of 2-3 lower than the best highly optimised rate-coded network, which we assume is due to the lower amount of spikes used. We provide a detailed breakdown of power consumption into static and dynamic components, in the hope that future work will be able to compare in greater detail.

One of the limitations of our architecture is the need to encode zeros. Since we compute using intervals and counterweights, a zero has to be represented by two spikes from input and *readout* neurons arriving at the same time so that their weights cancel each other out. A rate-coded architecture can omit to send spikes when the value is zero. The sparsity of frame-based datasets and therefore the opportunity to exploit this circumstance is highly variable, with MNIST containing 80.7% zeros, whereas CIFAR10 only has 0.25% zeros. In either case, even for a dataset such as MNIST, we still use orders of magnitudes fewer spikes than rate-coded techniques. Another limitation is the robustness to external noise injected into the network. A few spikes that are dropped or inserted might jeopardise the whole network execution.

In terms of possible further improvements, we can think of using g_f neurons for logarithmic activation functions in a converted network, which has been shown to increase classification accuracy [265]. Such computations are normally very costly on conventional hardware, but could potentially be cheaper on Loihi. Furthermore we could imagine exploring the combination of temporal and rate coding approaches in SNNs using a Time Difference Encoder [266] that translates a spike time interval into a spike rate, to try to combine the best of both worlds. Additional SNN blocks for recurrent or skip connections could be envisioned. The asynchronous nature of neuromorphic hardware could hereby help to reduce latency and improve accuracy when considering other rollout schemes than classical sequential rollout [267].

Spiking neural networks are an interesting avenue to explore for scientific computing but have until now scaled unfavourably when using rate-coding schemes [268, 269]. The ability to perform elementary computations, together with branching networks previously shown by STICK, paves the way for a frugal general-purpose computing machine. Together with other works based on STICK [270, 109], we hope that our implementation on neuromorphic hardware aids the development of precise timing frameworks.

Chapter 5

Conclusion

This thesis explores the components of a neuromorphic system and how they can help current technology to compute more efficiently. When we compare such a system to currently dominant technology and deep learning, we can reconstruct a similar trajectory of its success story. It was only the combination of algorithms which had existed since the 70s, large amounts of data available via the internet and the right hardware that enabled its success. Hooker [271] introduced the idea of a hardware lottery, claiming that a research idea wins because it is suited to the available software and hardware at that time, and not because the idea is superior to alternative research directions. When GPUs were widely available as highly parallel graphics processors, the research community drilled down a certain path of algorithm exploration because it worked well [272]. And indeed, groups have shown impressive results in various fields that use deep learning ranging from computer vision over natural language processing to speech recognition, but progress across application areas is strongly reliant on increases in computing power. Extrapolating forward this reliance reveals that progress along current lines is rapidly becoming economically, technically, and environmentally unsustainable which is detrimental to the deployment on battery-powered devices [273, 169]. It is these factors that drive the exploration of novel sensors, computational principles, and hardware. Neuromorphic engineering offers such alternatives, which have yet to prove successful. In order for neuromorphic systems to have a clear advantage over currently optimised systems, we argue that a full end-to-end pipeline is necessary to guarantee computation at significantly lower power. This pipeline consists of an event-based sensor and its output, which is in turn processed using an asynchronous algorithm on spiking hardware.

In chapter 2 we showed how an event-by-event algorithm can detect and track faces using less power than gold standard frame-based alternatives. The event camera provides us with a fine-grained temporal dimension that we can make use of in our features, which works well when detecting a spatio-temporal event such as an eye blink. Our work opens up the route for always-on detection capabilities on power-constrained systems such as robots or mobile devices that need to be able to detect the presence of a human face. The event-by-event driven nature allows the algorithm to have minimal latency when it comes to tracking the user's head, but in practice, this is often not needed. Much like

we down-sampled the signal received from the event camera in chapter 3, we could apply similar methods of spatio-temporal filtering in this case to record less data. This would allow us to reduce power consumption even further, which is arguably the main drawback of current conventional camera systems. We could extend our face detection algorithm to not only tell where and when a user blinks, but also to track their gaze. An event sensor with adequate spatio-temporal resolution would be able to track fine-grained eye movement to follow where a user is looking [274], potentially allowing the control of a device by otherwise physically handicapped people.

Our features were handcrafted, even though machine learning teaches us that we should let the data speak for itself. This is in part because we had a limited amount of data at hand for modern day standards to feed to our algorithm. Since there was no publicly available dataset for event-based face tracking available, we recorded our own database that we made available [1] in order to train and test our algorithm. Neuromorphic computing being an emerging field, this is a fairly common issue, although the situation is improving and increasingly large event-based vision datasets [275, 276] are available. Some works have resorted to generating the output of event cameras by converting videos from conventional cameras to events [277, 278] to be able to leverage already existing datasets or simulated them from scratch [279] to dramatically increase the amount of data available. What's interesting in both the datasets we use in chapters 2 and 3 is that they are real-world recordings of people [1] and gestures [2], contrary to several monitor recording datasets executed with an event camera that provide spiking versions of images [280, 281]. Artificial temporal correlation between events simply does not reflect real-world applications when dealing with neuromorphic algorithms because it ignores time as a potentially important feature and as such might be of limited value. To motivate further research activity with event-based datasets, we made publicly available a python library to facilitate the download of such datasets*. Part of its documentation is available in Appendix A.2.

We continued to use spatio-temporal features in chapter 3, where we connect event-based gesture recognition and other computer vision tasks to mobile phones that embed optimised hardware. We benchmark our algorithms on a database for mid-air hand gestures, since gesture recognition is poised to play a major role for future touchless user interfaces [282, 283, 284]. Currently such interfaces are not commonly seen on mobile systems, as frame-based or even active sensing approaches such as radar [285] are very costly to process. A passive sensor that is directly coupled to scene activity such as an event camera can afford to stay on for extended periods, thus making interaction with mobile technology accessible and intuitive. Our Android framework for mobile devices makes it easy to parse events from a small embedded event camera and our prototype

*The library *Tonic* is available under https://github.com/neuromorphs/tonic

device is able to execute different algorithms in real-time with low latency. Our event camera uses a relatively low spatial resolution of 304×240 pixels, but the event cameras are maturing fast, going from 128×128 pixels to full high definition resolution in a few years time. Even though it is desirable to have the option, event-based computer vision on mobile platforms might not need very high spatial resolution for every task, which is costly to process even on asynchronous hardware. Instead, a sensor with the adequate spatial and temporal resolution should be used that is suitable for the task at hand. In the case of on-demand frame reconstruction, an event camera will need higher spatial than temporal resolution, whereas in the case of gesture recognition, the reverse might be true. We already see the number of specialised sensors in an embedded device proliferating and event cameras could be a worthy addition to that.

From our setup it becomes immediately apparent that the integration of an external sensor is not a straightforward feat on a mobile consumer device such as a phone. The fact that the camera is connected via a USB cable results in higher power usage and bandwidth issues that are not present to such an extent with integrated sensors. This stresses the need for a tight integration between sensor and underlying processing hardware. The embedding of an event camera into a device such as a phone or tablet is likely to happen in the mid-term future, but faces hurdles since such devices are manufactured in vertically integrated processes that make the addition of single components a complicated task. Showing always-on visual detection or recognition pipelines on mobile devices that on average uses a fraction of the power of a conventional system will speed up industry adoption.

On our prototype device, we also made use of the Tensorflow Lite backend, which in turn uses neural network accelerator hardware for efficient inference. The issue is, however, that when we convert events into frame representations, we lose a lot of the advantages of event cameras. As mentioned in the beginning of this conclusion, the hardware lottery gave us GPUs to work with, but they are designed for a different kind of data. GPUs are a great workhorse when it comes to parallelising compute-intense tasks, but they fail to exploit high sparsity in signals. That is why sparse computation on GPUs is something that the research community is actively looking into at the moment [286, 287]. NVIDIA announced in 2020 that their latest generation of tensor cores would be able to transform dense matrices into sparse matrices using a transformation called *4:2 sparsity*, where the cost of computation is reduced by half. This accelerates inference up to a factor of 2 for a minor hit of accuracy [288], but such a feature requires supplementary hardware to check for zeros in the data.
On neuromorphic hardware, the sparse input directly drives asynchronous transistor switching activity, without the need for additional checks. The difference is essentially the

lack of input in comparison to lots of zeros of input in the case of GPUs. Even though GPUs and TensorFlow Lite are making amends to reduce the need to process unnecessary zeros, neuromorphic computing tackles different application scenarios. Sparsity in signals from an event-based sensor reaches levels of 99% depending on scene activity and is therefore much higher than what a *4:2 sparsity* could achieve to shrink. In the end neuromorphic computing that can exploit the *absence* of new input information will have a head start in certain applications for power-critical systems that track spurious events at high speeds. This is where GPUs that apply sparsity checks to avoid computation in a later step will not be able to compete.

SNNs are built to exploit sparsity. But it is not a straightforward task to train them using hardware that was not built for that. Currently, supervised training algorithms based on backpropagation show the best performance when it comes to detection or classification tasks [124, 137, 126]. Backpropagation as currently used is not biologically plausible [289] and thus goes against the grain of neuromorphic computing, but it seems as if some of the constraints that have been thought essential in backpropagation can be relaxed to bring it in line with biology. One of the constraints that can be relaxed is known as the *weight transport problem*, which means that forward and backward passes need symmetric weights. By using a fixed random matrix in the backward pass, it was shown that the network is still able to learn well [290, 291]. Another constraint is that errors that are passed backwards are signed and that rate-coded neurons would not easily be able to represent a negative value [289]. Interestingly, we show in chapter 4 that by using a reference timing spike, we can easily encode negative activations, even though it is not clear how the reference timing would be triggered in the brain. Overall, some works have focused on local approximations of a global error signals [114, 110], which is a promising avenue forward to biologically plausible learning.

From a practical perspective, it is relevant for researchers and practitioners alike how easy it is to train an SNN once data and potentially annotations are available. Currently, certainly due to the relative novelty of the field, there is no default tool chain that one can turn to. Deep learning has spawned and evolved over a multitude of open source frameworks and neuromorphic computing will need a similar ecosystem to ease development, training and exploration. Nengo [251] developed by Applied Brain Research comes closest to such an open neuromorphic ecosystem at the moment, but still employs restrictive licensing that prevents other companies from using their code. Furthermore its theoretical basis NEF is based on population coding rather than precise timing, which can be costly in certain cases as shown in Chapter 4. A crucial addition in neuromorphic training tools will be the option to execute code on neuromorphic hardware and not just CPU or GPU backends. Nengo does already support training and inference on different hardware backends.

We now turn to explore in more detail what neuromorphic hardware is capable of at the moment. An essential feature is the ability to compute using *time*. That means that asynchronous hardware such as TrueNorth [166] or Loihi [167] will compute using algorithmic time steps, which might execute faster or slower when measured in wall clock time depending on the workload. Computation using time has recently seen interesting applications on Loihi such as large scale nearest neighbour search [174], dynamic programming [292] or stochastic constraint optimisation [112]. Chapter 4 explores spiking neural networks on Loihi using a temporal instead of the currently prevalent rate encoding scheme and we show that using STICK [237] we can implement an efficient Turing complete system. Computation and memory are combined in every neuron that integrate input currents over time.

Spiking general purpose computation might one day be used as a resilient and fault-tolerant way to compute reliably and using little power. Currently, there is little to no error correction in place to detect faulty behaviour in conventional hardware systems. During production, areas of faulty transistors might be re-routed or powered down completely to retain a downgraded version of a chip that will still be able to sell, but after it has left the fab, no further checks or modifications will be executed. In extremely harsh environments such as in space or in radioactively contaminated areas this leads to systems having to carry redundant computer systems in case parts of it are irreversibly damaged. SNNs and neuromorphic hardware might exploit neuroplasticity mechanisms to realise efficient fault-tolerant and reconfigurable systems [293, 294], where neuroplasticity refers to the ability of the brain to adapt and reconfigure during lifetime operation. Additional support neurons in the system, much like the glia cells in the brain, could detect if a synapse is faulty and will increase probability of healthy synapses to transmit a spike in order to keep up the firing rate of target neuron. This can happen while the system is running and might offer the option for a *degradation* in system performance if parts are damaged rather than catastrophic hardware failures [295].

Spiking computing might also form a part of heterogeneous high performance computing due to its potential to implement large scale computations with a small power footprint [269]. Such a large scale system that computes with time will have its own areas of applications where it outperforms conventional computing. One such example that is straightforward to visualise is the search for the shortest path in a graph such as a road network [296]. Whereas conventional systems have to perform computation on every node and slowly integrate over path lengths [297], a neuromorphic system can just send out a wave front of spikes and automatically stop once a spike is received at the destination node. Such a graph search spiking system scales sub-linearly, in contrast to conventional systems.

Apart from using spikes for general purpose computation, we also explored the conversion

from feed-forward ANNs to SNNs using our temporal encoding scheme. We demonstrated a functional advantage as well as lower power usage over rate-coded networks in that respect. The comparison within the neuromorphic world (rate versus temporal coding) is a clear win for temporal coding in the scenarios tested, and both methods can beat EDP for inference on a GPU, for the specific case of a batch size of 1 [242]. Since GPUs are designed for high throughput, a single batch burns more power on them than on a neuromorphic chip for this task. As elucidated in Figure 1.11 however, neuromorphic hardware cannot compete with the execution of feed-forward neural nets on GPUs for higher batch sizes. This leads us to the following conclusions:

1. Use cases for neuromorphic chips to process static dense inputs such as images are very limited. This makes sense when we consider that GPUs have essentially been designed to display a sequence of static images at display frame rate.

2. Even though feed-forward network inference on neuromorphic hardware might not be ideal, the situation with recurrent architectures looks more promising. This is actually in favour of the biological model: our brain is a network of neurons with feedback connections. RNNs use the notion of time for sequence learning, which is quite costly to compute on GPUs but comes almost for free on neuromorphic hardware.

3. When dealing with one input at a time, neuromorphic hardware can compete with GPUs. This has interesting consequences for real-world applications. It is currently very costly for a machine learning model to be continually updated on the fly given an infrequent novel input over time. All this has to happen while at the same time preventing the model to completely forget things it learned in the past, which is called *catastrophic forgetting*. Continual learning is a field that seems promising to explore on neuromorphic hardware, to incorporate newly learned information into a model using little energy. This could become a critical differentiation feature in a market of neuromorphic edge devices, where a model should ideally continue to learn on demand once deployed with as little resources as possible. Some recent work has shown promising results of continual model updates on Loihi inspired by the olfactory circuit [298].

Overall, ANN-SNN conversion methods might be able to exploit some training tricks that are currently only available to ANNs, but this will only be an intermediate step. As neuromorphic hardware incorporates different learning capabilities and software training tool chains mature, SNNs will be trained natively using a backend that suits the task at hand. The sparing use of spikes in TTFS coding seems like a promising way forward for problems that make use of *time* on neuromorphic hardware. But as mentioned earlier regarding artificial time dimensions in synthetic event datasets, we have to be prudent not

to introduce time as a factor into computational problems that cannot make any use of it.

Spiking neural networks have been inspired by their biological counterpart, so one might ask the question at what point we will be able to connect the synthetic hardware to the different substrate that is our bodies given that the similarity between artificial and biological neurons will make interfacing easier [7]. Brain machine interfaces that record from dense multi-electrode arrays on the cortex currently suffer from high power consumption, increased heat dissipation, low channel count as well as an enormous amount of raw data to put through. By pre-processing signals close to where they originate, less energy has to be spent downstream to transmit information and tell different neurons apart. Mixed-signal neuromorphic processing units promise low-power sensory-processing and edge-distributed computation on hardware platforms, making use of the threshold crossing sampling theorem. In Haessig et al. [3] we present a local computation primitive of a spatio-temporal signal classifier that does on-sensor spike sorting in real-time. This work represents a first step towards the design of a large-scale neuromorphic processing system, and together with high-bandwidth systems that provide high channel count [299] will bring us closer to brain machine interfaces.

So far we have seen that a tight integration of hardware and software is more important than ever. This is an increasingly pressing topic for industry, for example when developing a product for the mobile or IoT world today. Already a simple consumer product has multiple different chips built in which take care of power delivery, battery life, storage controllers, sensor electronics, signal processors and more. Today, small companies do not have the financial power available that is needed to design their own chips. But a solution with off-the-shelf hardware will often not be competitive anymore when it comes to power efficiency. If companies need custom hardware, they are left with few choices. In most cases, companies will have to make do with off-the-shelf chips that come as close as possible to what they need. This could change however with the development of open-hardware that would allow smaller players to crowdsource and license designs that are specific to their needs. RISC-V is a computer architecture that was originally designed to support research and education by providing an open standard [300]. Recently this architecture has seen considerable traction and is poised to trigger an open-hardware revolution, much like what GNU/Linux did to the software world. The democratisation in hardware design follows similar reasoning in the sense that the way computer chips are designed has now become ubiquitous and common knowledge. Chips will be able to be customised to specific needs and therefore optimise power, space or cost. Due to the open design, they will also be less susceptible to security flaws [218]. Companies have already started putting their chip designs on GitHub [301]. As neuromorphic computing finds its niche in the vast

computing sector, it too will be subject of interest to educators, tinkerers and hackers of all sorts. Neuromorphic architectures based on RISC-V are already available [302] and the space will hopefully continue to grow.

Much like we have seen the breakout of memory into separate chips or the emergence of GPUs as dedicated hardware for high-throughput, parallel computation, we will see more specialised hardware arriving on the scene. Steve Furber paints a picture of heterogeneity in future processors, where specialised hardware accelerators will be coordinated by general-purpose cores [303]. Neuromorphic technology has the potential to form a part of this modern day computing, for tasks that use *time* to compute, that have sparse input signals, need to be fault-tolerant or extremely low-power. Within the neuromorphic sector we will see further proliferation into specialised variants of spiking hardware for inference, ultra-low power memristive architectures and more general-purpose spiking hardware [304]. When it comes to the software stack on top, responsible for training and deployment, we will hopefully see it converge to some common frameworks used by industry and research labs alike. Mobile platforms such as Mars robots, drones, brain-machine interfaces or satellites are just a few examples that can benefit from a highly specialised approach to computing. Today's handsets handle a wider range of workloads than ever before and as we humans rely more and more on our technology, the technology also meets us halfway and becomes a bit more like us. Such is the course of human-made inventions.

Appendix A

Authored Software Packages

A.1 Loris

This package was conceived because of the need for easy data parsing of files from neuromorphic cameras*. There existed already various standards for cameras such as **AEDAT**, or **DAT** but support to read files into Python was scarce and mainly based on scripts that circulated within labs. These scripts were mostly split between C++ and Matlab and a single point of entry to read multiple file types into Python was lacking. Loris is an easy to install versioned package and available on PyPi. It can read and write different file formats from neuromorphic cameras such as `.aedat4`, `.dat`, `.es` or `.csv`. Loris automatically deducts sensor size, total number of events and event type, such as DVS events or ATIS events that contain grey-level information too. A simple example of how to parse a file can be seen in Listing 1.

```
import loris
my_file = loris.read_file("/path/to/my-file.dat")
events = my_file["events"]

for event in events:
    print("ts:", event.t, "x:", event.x, "y:", event.y, "p:", event.p)
```

Listing 1: Python example code for Loris that shows how to read a file generated from a neuromorphic camera, for example a .dat file and afterwards loops over all events.

The package combines simple to understand parsing logic in Python with a fast C++ backend that allows to read files extremely fast. The frontend Python code relies on a `loris extension` module to read and write files in the Eventstream format. To support reading from `aedat` version 4, Alexandre Marcireau added support for another Python

*The source code is available at https://github.com/neuromorphic-paris/loris

package that is based on Rust and therefore similarly fast.

A.2 Tonic

Tonic provides publicly available spike-based datasets and data transformations based on PyTorch[†]. It is inspired by the PyTorch Vision package that provides image and video datasets in a similar easy manner and has received multiple contributions from the community. The goal is to provide researchers with an easy tool to work with different datasets to benchmark their algorithms. The optional transformations provide an additional way to filter, modify and batch the events before they are read. A simple example to read events from the NMNIST dataset, denoise them and create a time surface for each event can be executed with just a few lines, as shown in Listing 2 below. The installation is straightforward as it is available on PyPi.

```
import tonic
import tonic.transforms as T
transform = T.Compose([T.Denoise(time_filter=10000),
                       T.ToTimesurface(surface_dimensions=(7,7),
                       ↪  tau=5e3),])

testset = tonic.datasets.NMNIST(save_to='./data',
                                train=False,
                                transform=transform)

testloader = tonic.datasets.DataLoader(testset, shuffle=True)
for surfaces, target in iter(testloader):
    print("{} surfaces for target {}".format(len(surfaces), target))
```

Listing 2: Example code of loading data points of N-MNIST with a default batch size of 1 and applying two transformations to it. The first one drops all events that are temporally and spatially isolated, therefore denoising the recording. The second transformation creates time surfaces for each event.

[†]The source code is available at https://github.com/neuromorphs/tonic

Datasets

All datasets are subclasses of `torch.utils.data.Dataset`, that means that they have `_getitem_` and `_len_` methods implemented. Hence, they can all be passed to a `torch.utils.data.DataLoader` which can load multiple samples in parallel using workers provided in `torch.multiprocessing`. For example:

```
dataset = tonic.datasets.NMNIST(save_to='./data', train=False)
dataloader = tonic.datasets.DataLoader(dataset, shuffle=True,
↪  num_workers=4)
```

All the datasets have almost similar API. They all have two common arguments: `transform` and `target_transform` to transform the input and target respectively. Currently Tonic provides support for the following datasets:

- DVS gestures [305]
- N-CALTECH 101 [280]
- N-CARS [92]
- N-MNIST [280]
- NavGesture-sit and NavGesture-walk [40]
- POKER DVS [306]

Transforms

Transforms are common event transformations. They can be chained together using `Compose`. Additionally, there is the `tonic.functional module`. Functional transforms give fine-grained control over the transformations. This is useful if you have to build a more complex transformation pipeline. Tonic provides the following transformations, where community contributions are marked with a symbol(†):

Functional transformations

Crop Crops the sensor size to a smaller sensor and removes events outsize of the target sensor and maps.

Denoise Cycles through all events and drops it if there is no other event within a time of `time_filter` and a spatial neighbourhood of 1.

DropEvents† Drops events with a certain probability.

FlipLR Mirrors x coordinates of events and images (if present).

FlipPolarity† Changes polarities 1 to -1 and polarities [-1, 0] to 1.

FlipUD Mirrors y coordinates of events and images (if present).

RefractoryPeriod Cycles through all events and drops event if within refractory period for that pixel.

SpatialJitter Blurs x and y coordinates of events. Integer or subpixel precision possible.

TimeJitter Blurs timestamps of events. Will clip negative timestamps by default.

TimeReversal† Will reverse the timestamps of events with a certain probability.

TimeSkew† Scale and/or offset all timestamps.

UniformNoise Inject noise events.

Event representations

ToAveragedTimesurface† Creates Averaged Time Surfaces as in [92].

ToRatecodedFrame Bins events to frames of different time length.

ToSparseTensor Turn event array (N,E) into sparse Tensor (B,T,W,H).

ToTimesurface Create Time surfaces for each event as in [85].

Target transforms

ToOneHotEncoding Transforms one or more targets into a one hot encoding scheme.

Repeat Copies target n times. Useful to transform sample labels into sequences.

A.3 Frog

Fig. A.1 *The Frog logo to the left and Tonic logo to the right.*

Frog is an Android framework packaged as app which lets you use an event-camera that is connected via USB to the phone[‡]. It is then possible to parse the events with custom C++ code or use the Tarsier toolbox. To install the framework, it's easiest to use Android Studio, for superb support of both Java and C++ code and the Gradle build tool.

Frog was born in an effort to reconcile event cameras with mobile phones to showcase a prototype interface. Until an event camera is eventually connected via a MIPI interface directly to the motherboard of a mobile device, the USB connection serves as a plug and play replacement. Frog takes care of managing the permissions and life cycle of when a camera is connected and eventually disconnected. It will poll the camera for new event packets, which will be displayed real time in a live preview. This is possible since the event handling is done in C++, which allows for minimal lag. Apart from the live preview, the Frog framework will also pass the events to some custom C++ code which can be inserted by the user for algorithmic processing. We tested a gesture recognition algorithm based on HOTS [85] and NavGestures [2], with a 93% recognition rate. Frog currently supports the ATIS camera.

Fig. A.2 *App screenshots from left to right: Camera disconnected, camera starting, gesture being recognised.*

[‡]The source code is available at https://github.com/neuromorphic-paris/frog

A.4 Quartz

Quartz is an ANN to SNN conversion framework to facilitate efficient inference on neuromorphic hardware[§]. Contrary to the majority of conversion frameworks currently available that are based on rate coding, Quartz uses precise timing of spikes. A number is encoded in the inter spike interval (ISI), which allows to drastically reduce the number of spikes overall in comparison to rate coding. Quartz currently supports the following layer types:

- Dense
- Convolutional 2D
- Maxpooling

With this setup, simple CNNs can be encoded efficiently. After training your network with the help of a GPU, one selects the same architecture within Quartz and passes the weights and biases. Quartz will take care of quantization. A simple example is shown in Listing 3 below.

[§]The source code is available at https://github.com/biphasic/Quartz

```
import quartz
from quartz import layers

t_max = 2**8
input_dims = (1,28,28)
pool_kernel_size = [2,2]
batch_size = 100

# load weights and biases of ANN model
loihi_model = quartz.Network(t_max=t_max, layers=[
    layers.InputLayer(dims=input_dims),
    layers.ConvPool2D(weights=weights[0], biases=biases[0],
    ↪  pool_kernel_size=pool_kernel_size),
    layers.ConvPool2D(weights=weights[1], biases=biases[1],
    ↪  pool_kernel_size=pool_kernel_size),
    layers.Conv2D(weights=weights[2], biases=biases[2]),
    layers.Dense(weights=weights[3], biases=biases[3]),
    layers.Dense(weights=weights[4], biases=biases[4]),
])

# load inputs, for example an image
loihi_output = loihi_model(inputs)
```

Listing 3: Example code of how to convert an ANN model to an equivalent spiking model. Simply by passing the input to the model together with the number of steps per image for batch sizes larger than 1 will execute inference on Loihi and return the results.

Bibliography

[1] G. Lenz, S. H. Ieng, and R. B. Benosman, "Event-based face detection and tracking using the dynamics of eye blinks," *Frontiers in Neuroscience*, vol. 14, p. 587, 2020. v, 19, 40, 42, 84

[2] J.-M. Maro, G. Lenz, C. Reeves, and R. Benosman, "Event-based visual gesture recognition with background suppression running on a smart-phone," in *2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)*. IEEE, 2019, pp. 1–1. v, 40, 84, 95

[3] G. Haessig, D. G. Lesta, G. Lenz, R. Benosman, and P. Dudek, "A mixed-signal spatio-temporal signal classifier for on-sensor spike sorting," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5. v, 89

[4] S. Bamford and J. Danaher, "Transfer of personality to a synthetic human ('mind uploading') and the social construction of identity," *Journal of Consciousness Studies*, vol. 24, no. 11-12, pp. 6–30, 2017. 1

[5] W. M. Grill, S. E. Norman, R. V. Bellamkonda *et al.*, "Implanted neural interfaces: biochallenges and engineered solutions," *Annual review of biomedical engineering*, vol. 11, no. 1, pp. 1–24, 2009. 1

[6] N. Lago and A. Cester, "Flexible and organic neural interfaces: A review," *Applied Sciences*, vol. 7, no. 12, p. 1292, 2017. 1

[7] F. D. Broccard, S. Joshi, J. Wang, and G. Cauwenberghs, "Neuromorphic neural interfaces: from neurophysiological inspiration to biohybrid coupling with nervous systems," *Journal of neural engineering*, vol. 14, no. 4, p. 041002, 2017. 1, 89

[8] F. Corradi and G. Indiveri, "A neuromorphic event-based neural recording system for smart brain-machine-interfaces," *IEEE transactions on biomedical circuits and systems*, vol. 9, no. 5, pp. 699–709, 2015. 1

[9] A. Clark, "Natural-born cyborgs?" in *International Conference on Cognitive Technology*. Springer, 2001, pp. 17–24. 1

[10] W. Barfield, *Cyber-humans: Our future with machines*. Springer, 2015. 1

[11] R. Rosenberger, "An experiential account of phantom vibration syndrome," *Computers in Human Behavior*, vol. 52, pp. 124–131, 2015. 1

[12] Ofcom, "Adults' media use and attitudes report 2020," accessed: 2020-06-30. [Online]. Available: https://www.ofcom.org.uk/___data/assets/pdf_file/0033/196458/adults-media-use-and-attitudes-2020-full-chart-pack.pdf 1

[13] J. James, "Leapfrogging in mobile telephony: A measure for comparing country performance," *Technological Forecasting and Social Change*, vol. 76, no. 7, pp. 991–998, 2009. 1

[14] M. W. Fong, "Technology leapfrogging for developing countries," in *Encyclopedia of Information Science and Technology, Second Edition*. IGI Global, 2009, pp. 3707–3713. 1

[15] D. Amodei and D. Hernandez, "Neuromorphic computing gets ready for the (really) big time," 2018, accessed: 2020-09-25. [Online]. Available: https://openai.com/blog/ai-and-compute/ 2

[16] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," *arXiv preprint arXiv:1906.02243*, 2019. 2

[17] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020. 2

[18] Bloomberg, "Tesla's newest promises break the laws of batteries," 2017, accessed: 2020-11-30. [Online]. Available: https://www.bloomberg.com/news/articles/2017-11-24/tesla-s-newest-promises-break-the-laws-of-batteries 3

[19] O. Lopez-Fernandez, D. J. Kuss, L. Romo, Y. Morvan, L. Kern, P. Graziani, A. Rousseau, H.-J. Rumpf, A. Bischof, A.-K. Gässler *et al.*, "Self-reported dependence on mobile phones in young adults: A european cross-cultural empirical survey," *Journal of behavioral addictions*, vol. 6, no. 2, pp. 168–177, 2017. 3

[20] M. M. Waldrop, "The chips are down for moore's law," *Nature News*, vol. 530, no. 7589, p. 144, 2016. 3, 58

[21] F.-L. Yang, D.-H. Lee, H.-Y. Chen, C.-Y. Chang, S.-D. Liu, C.-C. Huang, T.-X. Chung, H.-W. Chen, C.-C. Huang, Y.-H. Liu *et al.*, "5nm-gate nanowire finfet," in *Digest of Technical Papers. 2004 Symposium on VLSI Technology, 2004.* IEEE, 2004, pp. 196–197. 3

[22] Y.-C. Huang, M.-H. Chiang, S.-J. Wang, and J. G. Fossum, "Gaafet versus pragmatic finfet at the 5nm si-based cmos technology node," *IEEE Journal of the Electron Devices Society*, vol. 5, no. 3, pp. 164–169, 2017. 3

[23] M. Mahowald and R. Douglas, "A silicon neuron," *Nature*, vol. 354, no. 6354, pp. 515–518, 1991. 3

[24] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. Van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud *et al.*, "Neuromorphic silicon neuron circuits," *Frontiers in neuroscience*, vol. 5, p. 73, 2011. 3, 18

[25] D. Monroe, "Neuromorphic computing gets ready for the (really) big time," 2014. 3

[26] T. S. Perry, "Move over, moore's law. make way for huang's law [spectral lines]," *IEEE Spectrum*, vol. 55, no. 5, pp. 7–7, 2018. 4

[27] W. Maass, C. H. Papadimitriou, S. Vempala, and R. Legenstein, "Brain computation: a computer science perspective," in *Computing and Software Science*. Springer, 2019, pp. 184–199. 5

[28] M. A. Mahowald and T. Delbrück, "Cooperative stereo matching using static and dynamic image features," in *Analog VLSI implementation of neural systems*. Springer, 1989, pp. 213–238. 6

[29] M. Mahowald, "The silicon retina," in *An Analog VLSI System for Stereoscopic Vision*. Springer, 1994, pp. 4–65. 7

[30] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis *et al.*, "Event-based vision: A survey," *arXiv preprint arXiv:1904.08405*, 2019. 7, 19

[31] T. Delbruck, "Silicon retina with correlation-based, velocity-tuned pixels," *IEEE Transactions on neural networks*, vol. 4, no. 3, pp. 529–541, 1993. 7

[32] S. Kameda and T. Yagi, "A silicon retina calculating high-precision spatial and temporal derivatives," in *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, vol. 1. IEEE, 2001, pp. 201–205. 7

[33] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128× 128 120 db 15μs latency asynchronous temporal contrast vision sensor," *IEEE journal of solid-state circuits*, vol. 43, no. 2, pp. 566–576, 2008. 7, 9, 10, 24, 40

[34] G. Taverni, D. P. Moeys, C. Li, C. Cavaco, V. Motsnyi, D. S. S. Bello, and T. Delbruck, "Front and back illuminated dynamic and active pixel vision sensors compari-

son," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 5, pp. 677–681, 2018. 9

[35] C. Posch, D. Matolin, and R. Wohlgenannt, "A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, 2010. 10, 24, 40, 41, 42

[36] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240× 180 130 db 3 μs latency global shutter spatiotemporal vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, 2014. 10, 40

[37] B. Son, Y. Suh, S. Kim, H. Jung, J.-S. Kim, C. Shin, K. Park, K. Lee, J. Park, J. Woo *et al.*, "4.1 a 640× 480 dynamic vision sensor with a 9μm pixel and 300meps address-event representation," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 66–67. 10

[38] T. Finateu, A. Niwa, D. Matolin, K. Tsuchimoto, A. Mascheroni, E. Reynaud, P. Mostafalu, F. Brady, L. Chotard, F. LeGoff *et al.*, "5.10 a 1280× 720 back-illuminated stacked temporal contrast event-based vision sensor with 4.86 μm pixels, 1.066 geps readout, programmable event-rate controller and compressive data-formatting pipeline," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 112–114. 10

[39] J. Conradt, "On-board real-time optic-flow for miniature event-based vision sensors," in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2015, pp. 1858–1863. 10, 42

[40] J.-M. Maro, S.-H. Ieng, and R. Benosman, "Event-based gesture recognition with dynamic background suppression using smartphone computational capabilities," *Frontiers in Neuroscience*, vol. 14, p. 275, 2020. 10, 40, 49, 50, 51, 52, 53, 93

[41] T. Delbruck, "The slow but steady rise of the event camera," 2020, accessed: 2020-09-30. [Online]. Available: https://www.eetimes.com/the-slow-but-steady-rise-of-the-event-camera/ 10

[42] G. Johansson, "Visual motion perception," *Scientific American*, vol. 232, no. 6, pp. 76–89, 1975. 10

[43] ——, "Visual perception of biological motion and a model for its analysis," *Perception & psychophysics*, vol. 14, no. 2, pp. 201–211, 1973. 10

[44] T. F. Shipley, "The effect of object and event orientation on perception of biological motion," *Psychological science*, vol. 14, no. 4, pp. 377–380, 2003. 11

[45] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015. 11

[46] S. Changpinyo, M. Sandler, and A. Zhmoginov, "The power of sparsity in convolutional neural networks," *arXiv preprint arXiv:1702.06257*, 2017. 11

[47] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International conference on machine learning*, 2016, pp. 2849–2858. 11

[48] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713. 11, 40

[49] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016. 11

[50] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," *arXiv preprint arXiv:1902.08153*, 2019. 11

[51] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015. 11, 40

[52] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018. 11

[53] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542. 11

[54] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131. 11

[55] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016. 11

[56] A. Gholami, K. Kwon, B. Wu, Z. Tai, X. Yue, P. Jin, S. Zhao, and K. Keutzer, "Squeezenext: Hardware-aware neural network design," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 1638–1647. 11

[57] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856. 11

[58] L. Sifre and S. Mallat, "Rigid-motion scattering for image classification," *Ph. D. thesis*, 2014. 11

[59] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017. 11

[60] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520. 11

[61] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1314–1324. 11

[62] G. Chen, L. Hong, J. Dong, P. Liu, J. Conradt, and A. Knoll, "Eddd: Event-based drowsiness driving detection through facial motion analysis with neuromorphic vision sensor," *IEEE Sensors Journal*, vol. 20, no. 11, pp. 6170–6181, 2020. 12

[63] J.-Y. Won, H. Ryu, T. Delbruck, J. H. Lee, and J. Hu, "Proximity sensing based on a dynamic vision sensor for mobile devices," *IEEE Transactions on industrial electronics*, vol. 62, no. 1, pp. 536–544, 2014. 12

[64] V. Vasco, A. Glover, E. Mueggler, D. Scaramuzza, L. Natale, and C. Bartolozzi, "Independent motion detection with event-driven cameras," in *2017 18th International Conference on Advanced Robotics (ICAR)*. IEEE, 2017, pp. 530–536. 12

[65] N. Waniek, J. Biedermann, and J. Conradt, "Cooperative slam on small mobile robots," in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2015, pp. 1810–1815. 12

[66] A. R. Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Ultimate slam? combining events, images, and imu for robust visual slam in hdr and high-speed scenarios," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 994–1001, 2018. 12, 40

[67] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza, "Are we ready for autonomous drone racing? the uzh-fpv drone racing dataset," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6713–6719. 12

[68] B. J. Pijnacker Hordijk, K. Y. Scheper, and G. C. De Croon, "Vertical landing for micro air vehicles using event-based optical flow," *Journal of Field Robotics*, vol. 35, no. 1, pp. 69–90, 2018. 12

[69] D. P. Moeys, F. Corradi, E. Kerr, P. Vance, G. Das, D. Neil, D. Kerr, and T. Delbrück, "Steering a predator robot using a mixed frame/event-driven convolutional neural network," in *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*. IEEE, 2016, pp. 1–8. 12

[70] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "Ev-flownet: Self-supervised optical flow estimation for event-based cameras," *arXiv preprint arXiv:1802.06898*, 2018. 12

[71] ——, "Unsupervised event-based learning of optical flow, depth, and egomotion," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 989–997. 12

[72] D. Gehrig, M. Rüegg, M. Gehrig, J. H. Carrio, and D. Scaramuzza, "Combining events and frames using recurrent asynchronous multimodal networks for monocular depth prediction," *arXiv preprint arXiv:2102.09320*, 2021. 12

[73] H. Kim, S. Leutenegger, and A. J. Davison, "Real-time 3d reconstruction and 6-dof tracking with an event camera," in *European Conference on Computer Vision*. Springer, 2016, pp. 349–364. 12

[74] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, "Events-to-video: Bringing modern computer vision to event cameras," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3857–3866. 12

[75] ——, "High speed and high dynamic range video with an event camera," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019. 12

[76] H. Kim, A. Handa, R. Benosman, S. Ieng, and A. Davison, "Simultaneous mosaicing and tracking with an event camera," in *BMVC 2014-Proceedings of the British Machine Vision Conference 2014*, 2014. 12

[77] L. Pan, C. Scheerlinck, X. Yu, R. Hartley, M. Liu, and Y. Dai, "Bringing a blurry frame alive at high frame-rate with an event camera," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6820–6829. 12, 40

[78] S. Afshar, A. P. Nicholson, A. van Schaik, and G. Cohen, "Event-based object detection and tracking for space situational awareness," *arXiv preprint arXiv:1911.08730*, 2019. 12

[79] A. Mitrokhin, Z. Hua, C. Fermuller, and Y. Aloimonos, "Learning visual motion segmentation using event surfaces," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 414–14 423. 12

[80] E. Perot, P. de Tournemire, D. Nitti, J. Masci, and A. Sironi, "Learning to detect objects with a 1 megapixel event camera," *arXiv preprint arXiv:2009.13436*, 2020. 12

[81] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I.-A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S.-C. 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*, vol. 30, no. 3, pp. 644–656, 2018. 12

[82] P. Spilger, E. Müller, A. Emmel, A. Leibfried, C. Mauch, C. Pehle, J. Weis, O. Breitwieser, S. Billaudelle, S. Schmitt *et al.*, "hxtorch: Pytorch for anns on brainscales-2," *arXiv preprint arXiv:2006.13138*, 2020. 12

[83] M. R. Azghadi, C. Lammie, J. K. Eshraghian, M. Payvand, E. Donati, B. Linares-Barranco, and G. Indiveri, "Hardware implementation of deep network accelerators towards healthcare and biomedical applications," *arXiv preprint arXiv:2007.05657*, 2020. 12

[84] M. Parsa, J. P. Mitchell, C. D. Schuman, R. M. Patton, T. E. Potok, and K. Roy, "Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design," *Frontiers in Neuroscience*, vol. 14, p. 667, 2020. 12

[85] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman, "Hots: a hierarchy of event-based time-surfaces for pattern recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 7, pp. 1346–1359, 2016. 12, 13, 38, 42, 50, 51, 52, 94, 95

[86] H. Liu, D. P. Moeys, G. Das, D. Neil, S.-C. Liu, and T. Delbrück, "Combined frame- and event-based detection and tracking," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2016, pp. 2511–2514. 12

[87] D. Tedaldi, G. Gallego, E. Mueggler, and D. Scaramuzza, "Feature detection and tracking with the dynamic and active-pixel vision sensor (davis)," in *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*. IEEE, 2016, pp. 1–7. 12

[88] C. Zamarreño-Ramos, A. Linares-Barranco, T. Serrano-Gotarredona, and B. Linares-Barranco, "Multicasting mesh aer: A scalable assembly approach for reconfigurable neuromorphic structured aer systems. application to convnets," *IEEE transactions on biomedical circuits and systems*, vol. 7, no. 1, pp. 82–102, 2012. 12

[89] M. Ambroise, T. Levi, Y. Bornat, and S. Saighi, "Biorealistic spiking neural network on fpga," in *2013 47th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2013, pp. 1–6. 12

[90] L. A. Camuñas-Mesa, Y. L. Domínguez-Cordero, A. Linares-Barranco, T. Serrano-Gotarredona, and B. Linares-Barranco, "A configurable event-driven convolutional node with rate saturation mechanism for modular convnet systems implementation," *Frontiers in neuroscience*, vol. 12, p. 63, 2018. 12

[91] R. Tapiador-Morales, J.-M. Maro, A. Jimenez-Fernandez, G. Jimenez-Moreno, R. Benosman, and A. Linares-Barranco, "Event-based gesture recognition through a hierarchy of time-surfaces for fpga," *Sensors*, vol. 20, no. 12, p. 3404, 2020. 12

[92] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman, "Hats: Histograms of averaged time surfaces for robust event-based object classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1731–1740. 12, 13, 38, 42, 93, 94

[93] R. Benosman, S.-H. Ieng, C. Clercq, C. Bartolozzi, and M. Srinivasan, "Asynchronous frameless event-based optical flow," *Neural Networks*, vol. 27, pp. 32–37, 2012. 12, 49

[94] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi, "Event-based visual flow," *IEEE transactions on neural networks and learning systems*, vol. 25, no. 2, pp. 407–417, 2013. 12, 49

[95] M. B. Milde, O. J. Bertrand, R. Benosmanz, M. Egelhaaf, and E. Chicca, "Bioinspired event-driven collision avoidance algorithm based on optic flow," in *2015 International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*. IEEE, 2015, pp. 1–7. 12

[96] H. Akolkar, S. H. Ieng, and R. Benosman, "Real-time high speed motion prediction using fast aperture-robust event-driven visual flow," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. 12, 49, 50

[97] X. Clady, S.-H. Ieng, and R. Benosman, "Asynchronous event-based corner detection and matching," *Neural Networks*, vol. 66, pp. 91–106, 2015. 12

[98] E. Mueggler, C. Bartolozzi, and D. Scaramuzza, "Fast event-based corner detection." in *British Machine Vision Conference (BMVC)*, no. CONF, 2017. 12

[99] J. Manderscheid, A. Sironi, N. Bourdis, D. Migliore, and V. Lepetit, "Speed invariant time surface for learning to detect corner points with event-based cameras," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 245–10 254. 12

[100] D. Reverter Valeiras, G. Orchard, S.-H. Ieng, and R. B. Benosman, "Neuromorphic event-based 3d pose estimation," *Frontiers in neuroscience*, vol. 9, p. 522, 2016. 12

[101] X. Lagorce, C. Meyer, S.-H. Ieng, D. Filliat, and R. Benosman, "Asynchronous event-based multikernel algorithm for high-speed visual features tracking," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 8, pp. 1710–1720, 2014. 12, 30

[102] S. Afshar, T. J. Hamilton, J. Tapson, A. van Schaik, and G. Cohen, "Investigation of event-based surfaces for high-speed detection, unsupervised feature extraction, and object recognition," *Frontiers in neuroscience*, vol. 12, p. 1047, 2019. 12

[103] X. Clady, J.-M. Maro, S. Barré, and R. B. Benosman, "A motion-based feature for event-based pattern recognition," *Frontiers in neuroscience*, vol. 10, p. 594, 2017. 13

[104] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997. 13, 59

[105] J. J. Hopfield, "Hopfield network," *Scholarpedia*, vol. 2, no. 5, p. 1977, 2007. 14

[106] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 14

[107] L. Deng, Y. Wu, X. Hu, L. Liang, Y. Ding, G. Li, G. Zhao, P. Li, and Y. Xie, "Rethinking the performance comparison between snns and anns," *Neural Networks*, vol. 121, pp. 294–307, 2020. 14, 19

[108] S. J. Verzi, F. Rothganger, O. D. Parekh, T.-T. Quach, N. E. Miner, C. M. Vineyard, C. D. James, and J. B. Aimone, "Computing with spikes: The advantage of fine-grained timing," *Neural computation*, vol. 30, no. 10, pp. 2660–2690, 2018. 14, 58

[109] J. V. Monaco, M. M. Vindiola, and R. Benosman, "Steam: Spike time encoded addressable memory," *unpublished*, 2018. 14, 82

[110] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity dynamics for deep continuous local learning (decolle)," *Frontiers in Neuroscience*, vol. 14, p. 424, 2020. 14, 16, 86

[111] G. A. Fonseca Guerra and S. B. Furber, "Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems," *Frontiers in neuroscience*, vol. 11, p. 714, 2017. 15

[112] C. Yakopcic, N. Rahman, T. Atahary, T. M. Taha, and S. Douglass, "Solving constraint satisfaction problems using the loihi spiking neuromorphic processor," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1079–1084. 15, 87

[113] B. Yin, F. Corradi, and S. M. Bohté, "Effective and efficient computation with multiple-timescale spiking recurrent neural networks," in *International Conference on Neuromorphic Systems 2020*, 2020, pp. 1–8. 15

[114] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, "A solution to the learning dilemma for recurrent networks of spiking neurons," *bioRxiv*, p. 738385, 2020. 15, 16, 86

[115] D. Patel, H. Hazan, D. J. Saunders, H. T. Siegelmann, and R. Kozma, "Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to atari breakout game," *Neural Networks*, vol. 120, pp. 108–115, 2019. 15

[116] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *arXiv preprint arXiv:1509.06461*, 2015. 15

[117] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016. 15

[118] S. Sharmin, P. Panda, S. S. Sarwar, C. Lee, W. Ponghiran, and K. Roy, "A comprehensive analysis on adversarial robustness of spiking neural networks," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8. 15

[119] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, pp. 5998–6008, 2017. 15

[120] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser *et al.*, "Rethinking attention with performers," *arXiv preprint arXiv:2009.14794*, 2020. 15

[121] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: opportunities and challenges," *Frontiers in neuroscience*, vol. 12, p. 774, 2018. 15

[122] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in neuroscience*, vol. 11, p. 682, 2017. 15, 58, 70, 71, 77

[123] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in neuroscience*, vol. 10, p. 508, 2016. 15

[124] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems*, 2018, pp. 1412–1421. 15, 16, 86

[125] S. R. Kheradpisheh and T. Masquelier, "S4nn: temporal backpropagation for spiking neural networks with one spike per neuron," *arXiv preprint arXiv:1910.09495*, 2019. 15

[126] T. C. Wunderlich and C. Pehle, "Eventprop: Backpropagation for exact gradients in spiking neural networks," *arXiv preprint arXiv:2009.08378*, 2020. 15, 16, 86

[127] G.-q. Bi and M.-m. Poo, "Synaptic modification by correlated activity: Hebb's postulate revisited," *Annual review of neuroscience*, vol. 24, no. 1, pp. 139–166, 2001. 15

[128] P. J. Sjostrom, E. A. Rancz, A. Roth, and M. Hausser, "Dendritic excitability and synaptic plasticity," *Physiological reviews*, vol. 88, no. 2, pp. 769–840, 2008. 15

[129] H. Mostafa, V. Ramesh, and G. Cauwenberghs, "Deep supervised learning using local errors," *Frontiers in neuroscience*, vol. 12, p. 608, 2018. 15

[130] B. Rueckauer and S.-C. Liu, "Conversion of analog to spiking neural networks using sparse temporal coding," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5. 15, 72

[131] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015. 15

[132] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014. 15, 77

[133] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," in *Advances in Neural Information Processing Systems*, 2018, pp. 787–797. 16

[134] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the national academy of sciences*, vol. 113, no. 41, pp. 11 441–11 446, 2016. 16

[135] S. M. Bohte, "Error-backpropagation in networks of fractionally predictive spiking neurons," in *International Conference on Artificial Neural Networks*. Springer, 2011, pp. 60–68. 16

[136] F. Zenke and S. Ganguli, "Superspike: Supervised learning in multilayer spiking neural networks," *Neural computation*, vol. 30, no. 6, pp. 1514–1541, 2018. 16

[137] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, pp. 61–63, 2019. 16, 86

[138] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990. 16

[139] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019. 16

[140] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS Comput Biol*, vol. 3, no. 2, p. e31, 2007. 16

[141] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "Stdp-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, vol. 99, pp. 56–67, 2018. 16

[142] N. Frémaux and W. Gerstner, "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules," *Frontiers in neural circuits*, vol. 9, p. 85, 2016. 16

[143] W. Gerstner, M. Lehmann, V. Liakoni, D. Corneil, and J. Brea, "Eligibility traces and plasticity on behavioral time scales: experimental support of neohebbian three-factor learning rules," *Frontiers in neural circuits*, vol. 12, p. 53, 2018. 16

[144] "Apple Silicon (Arm) Macs," accessed: 2020-07-30. [Online]. Available: https://www.macrumors.com/guide/apple-silicon/ 17

[145] "Hands on: Huawei p40 review," accessed: 2020-07-30. [Online]. Available: https://www.techradar.com/reviews/huawei-p40 17

[146] A. Carroll, G. Heiser *et al.*, "An analysis of power consumption in a smartphone." in *USENIX annual technical conference*, vol. 14. Boston, MA, 2010, pp. 21–21. 18

[147] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, and V. Toma, "Always-on vision processing unit for mobile applications," *IEEE Micro*, vol. 35, no. 2, pp. 56–66, 2015. 18

[148] D. Ma, J. Shen, Z. Gu, M. Zhang, X. Zhu, X. Xu, Q. Xu, Y. Shen, and G. Pan, "Darwin: A neuromorphic hardware co-processor based on spiking neural networks," *Journal of Systems Architecture*, vol. 77, pp. 43–51, 2017. 18

[149] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 1–12. 18

[150] N. Jouppi, C. Young, N. Patil, and D. Patterson, "Motivation for and evaluation of the first tensor processing unit," *IEEE Micro*, vol. 38, no. 3, pp. 10–19, 2018. 18

[151] "Introducing the next generation of on-device vision models: Mobilenetv3 and mobilenetedgetpu," 2019, accessed: 2020-07-30. [Online]. Available: https://ai.googleblog.com/2019/11/introducing-next-generation-on-device.html 18

[152] S. J. van Albada, A. G. Rowley, J. Senk, M. Hopkins, M. Schmidt, A. B. Stokes, D. R. Lester, M. Diesmann, and S. B. Furber, "Performance comparison of the digital neuromorphic hardware spinnaker and the neural network simulation software nest for a full-scale cortical microcircuit model," *Frontiers in neuroscience*, vol. 12, p. 291, 2018. 18

[153] R. Douglas, M. Mahowald, and C. Mead, "Neuromorphic analogue vlsi," *Annual review of neuroscience*, vol. 18, no. 1, pp. 255–281, 1995. 18

[154] K. A. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 5, pp. 416–434, 2000. 18

[155] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodríguez, L. Camuñas-Mesa, R. Berner, M. Rivas-Pérez, T. Delbruck *et al.*, "Caviar: A 45k neuron, 5m synapse, 12g connects/s aer hardware sensory–processing–learning–actuating system for high-speed visual object recognition and tracking," *IEEE Transactions on Neural networks*, vol. 20, no. 9, pp. 1417–1438, 2009. 18

[156] J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, 2010, pp. 1947–1950. 18

[157] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses," *Frontiers in neuroscience*, vol. 9, p. 141, 2015. 18

[158] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps)," *IEEE transactions on biomedical circuits and systems*, vol. 12, no. 1, pp. 106–122, 2017. 18

[159] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014. 18

[160] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971. 19

[161] A. Chanthbouala, V. Garcia, R. O. Cherifi, K. Bouzehouane, S. Fusil, X. Moya, S. Xavier, H. Yamada, C. Deranlot, N. D. Mathur *et al.*, "A ferroelectric memristor," *Nature materials*, vol. 11, no. 10, pp. 860–864, 2012. 19

[162] S. Saïghi, C. G. Mayr, T. Serrano-Gotarredona, H. Schmidt, G. Lecerf, J. Tomas, J. Grollier, S. Boyn, A. F. Vincent, D. Querlioz *et al.*, "Plasticity in memristive devices for spiking neural networks," *Frontiers in neuroscience*, vol. 9, p. 51, 2015. 19

[163] S. Boyn, J. Grollier, G. Lecerf, B. Xu, N. Locatelli, S. Fusil, S. Girod, C. Carrétéro, K. Garcia, S. Xavier *et al.*, "Learning through ferroelectric domain dynamics in solid-state synapses," *Nature communications*, vol. 8, no. 1, pp. 1–7, 2017. 19

[164] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014. 19, 58

[165] C. Mayr, S. Hoeppner, and S. Furber, "Spinnaker 2: A 10 million core processor system for brain simulation and machine learning," *arXiv preprint arXiv:1911.02385*, 2019. 19

[166] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 34, no. 10, pp. 1537–1557, 2015. 19, 87

[167] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018. 19, 58, 59, 60, 87

[168] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75–88, 2019. 19

[169] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green ai," *arXiv preprint arXiv:1907.10597*, 2019. 19, 83

[170] B. Ramesh, A. Ussa, L. Della Vedova, H. Yang, and G. Orchard, "Low-power dynamic object detection and classification with freely moving event cameras," *Frontiers in Neuroscience*, vol. 14, p. 135, 2020. 19

[171] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, and S. R. Risbud, "Advancing neuromorphic computing with loihi: A survey of results and outlook," *Proceedings of the IEEE*, 2021. 20

[172] "Human brain supercomputer with 1 million processors switched on for first time," accessed: 2020-07-30. [Online]. Available: https://www.manchester.ac.uk/discover/news/human-brain-supercomputer-with-1million-processors-switched-on-for-first-time 19

[173] C. S. Thakur, J. L. Molin, G. Cauwenberghs, G. Indiveri, K. Kumar, N. Qiao, J. Schemmel, R. Wang, E. Chicca, J. Olson Hasler *et al.*, "Large-scale neuromorphic spiking array processors: A quest to mimic the brain," *Frontiers in neuroscience*, vol. 12, p. 891, 2018. 19

[174] E. P. Frady, G. Orchard, D. Florey, N. Imam, R. Liu, J. Mishra, J. Tse, A. Wild, F. T. Sommer, and M. Davies, "Neuromorphic nearest-neighbor search using intel's pohoiki springs," *arXiv preprint arXiv: 2004.12691*, 2020. 19, 59, 87

[175] O. Moreira, A. Yousefzadeh, F. Chersi, G. Cinserin, R.-J. Zwartenkot, A. Kapoor, P. Qiao, P. Kievits, M. Khoei, L. Rouillard *et al.*, "Neuronflow: a neuromorphic processor architecture for live ai applications," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 840–845. 20

[176] P. A. van der Made and A. S. Mankar, "Neural processor based accelerator system and method," Jan. 26 2017, uS Patent App. 15/218,075. 20

[177] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, no. 2, pp. 137–154, 2004. 24, 32, 36

[178] H. Jiang and E. Learned-Miller, "Face detection with the faster r-cnn," in *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*. IEEE, 2017, pp. 650–657. 24, 25

[179] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37. 24, 32, 36

[180] H. Li and L. Shi, "Robust event-based object tracking combining correlation filter and cnn representation," *Frontiers in neurorobotics*, vol. 13, 2019. 24, 32, 36, 37

[181] S. Yang, P. Luo, C. C. Loy, and X. Tang, "Faceness-net: Face detection through deep facial part responses," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 8, pp. 1845–1859, 2017. 25

[182] X. Sun, P. Wu, and S. C. Hoi, "Face detection using deep learning: An improved faster rcnn approach," *Neurocomputing*, vol. 299, pp. 42–50, 2018. 25

[183] V. Bazarevsky, Y. Kartynnik, A. Vakunov, K. Raveendran, and M. Grundmann, "Blazeface: Sub-millisecond neural face detection on mobile gpus," *arXiv preprint arXiv:1907.05047*, 2019. 25

[184] J. Ren, N. Kehtarnavaz, and L. Estevez, "Real-time optimization of viola-jones face detection for mobile platforms," in *Circuits and Systems Workshop: System-on-Chip-Design, Applications, Integration, and Software, 2008 IEEE Dallas*. IEEE, 2008, pp. 1–4. 25

[185] M. Noman, T. Bin, M. Ahad, and A. Rahman, "Mobile-based eye-blink detection performance analysis on android platform," *Frontiers in ICT*, vol. 5, p. 4, 2018. 25

[186] T. Nakano, M. Kato, Y. Morito, S. Itoi, and S. Kitazawa, "Blink-related momentary activation of the default mode network while viewing videos," *Proceedings of the National Academy of Sciences*, vol. 110, no. 2, pp. 702–706, 2013. 26

[187] J. A. Stern, D. Boyer, and D. Schroeder, "Blink rate: a possible measure of fatigue," *Human factors*, vol. 36, no. 2, pp. 285–297, 1994. 26, 27

[188] Q. Wang, J. Yang, M. Ren, and Y. Zheng, "Driver fatigue detection: a survey," in *2006 6th world congress on intelligent control and automation*, vol. 2. IEEE, 2006, pp. 8587–8591. 26

[189] H. Häkkänen, H. Summala, M. Partinen, M. Tiihonen, and J. Silvo, "Blink duration as an indicator of driver sleepiness in professional bus drivers," *Sleep*, vol. 22, no. 6, pp. 798–802, 1999. 26

[190] S. Benedetto, M. Pedrotti, L. Minin, T. Baccino, A. Re, and R. Montanari, "Driver workload and eye blink duration," *Transportation research part F: traffic psychology and behaviour*, vol. 14, no. 3, pp. 199–208, 2011. 26

[191] J. C. Walker, M. Kendal-Reed, M. J. Utell, and W. S. Cain, "Human breathing and eye blink rate responses to airborne chemicals." *Environmental health perspectives*, vol. 109, no. suppl 4, pp. 507–512, 2001. 26

[192] A. R. Bentivoglio, S. B. Bressman, E. Cassetta, D. Carretta, P. Tonali, and A. Albanese, "Analysis of blink rate patterns in normal subjects," *Movement disorders*, vol. 12, no. 6, pp. 1028–1034, 1997. 27

[193] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99. 32, 36

[194] S. Yang, P. Luo, C.-C. Loy, and X. Tang, "Wider face: A face detection benchmark," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5525–5533. 32

[195] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2010, pp. 2544–2550. 37

[196] D. R. Valeiras, X. Lagorce, X. Clady, C. Bartolozzi, S.-H. Ieng, and R. Benosman, "An asynchronous neuromorphic event-driven visual part-based shape tracking," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 12, pp. 3045–3059, 2015. 38

[197] IntelCorporation, "7th Generation Intel ® Processor Family and 8th Generation Intel ® Processor Family for U Quad Core Platforms Specification Data sheet," *Tech reports*, 2017. 38

[198] "Tensorflow lite guide," 2021, accessed: 2021-03-05. [Online]. Available: https://www.tensorflow.org/lite/guide 39

[199] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114. 40

[200] T. Serrano-Gotarredona and B. Linares-Barranco, "A 128 ×128 1.5% contrast sensitivity 0.9% fpn 3 μs latency 4 mw asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 3, pp. 827–838, 2013. 40

[201] R. Berner, C. Brandli, M. Yang, S.-C. Liu, and T. Delbruck, "A 240× 180 10mw 12us latency sparse-output vision sensor for mobile applications," in *2013 Symposium on VLSI Circuits*. IEEE, 2013, pp. C186–C187. 40

[202] A. Savran, R. Tavarone, B. Higy, L. Badino, and C. Bartolozzi, "Energy and computation efficient audio-visual voice activity detection driven by event-cameras," in *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*. IEEE, 2018, pp. 333–340. 40, 56

[203] E. Ceolini, G. Taverni, L. Khacef, M. Payvand, and E. Donati, "Sensor fusion using emg and vision for hand gesture classification in mobile applications," in *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2019, pp. 1–4. 40

[204] Z. Jiang, Y. Zhang, D. Zou, J. Ren, J. Lv, and Y. Liu, "Learning event-based motion deblurring," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 3320–3329. 40

[205] F. Galluppi, C. Denk, M. C. Meiner, T. C. Stewart, L. A. Plana, C. Eliasmith, S. Furber, and J. Conradt, "Event-based neural computing on an autonomous mobile platform," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 2862–2867. 40

[206] E. Mueggler, G. Gallego, and D. Scaramuzza, "Continuous-time trajectory estimation for event-based vision sensors," University of Zurich, Tech. Rep., 2015. 40

[207] A. Censi and D. Scaramuzza, "Low-latency event-based visual odometry," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 703–710. 40

[208] G. Haessig and R. Benosman, "A sparse coding multi-scale precise-timing machine learning algorithm for neuromorphic event-based sensors," in *Micro-and Nanotechnology Sensors, Systems, and Applications X*, vol. 10639. International Society for Optics and Photonics, 2018, p. 106391U. 42

[209] J. Lee, N. Chirkov, E. Ignasheva, Y. Pisarchyk, M. Shieh, F. Riccardi, R. Sarokin, A. Kulik, and M. Grundmann, "On-device neural net inference with mobile gpus," *arXiv preprint arXiv:1907.01989*, 2019. 46

[210] A. Marcireau, S.-H. Ieng, and R. Benosman, "Sepia, tarsier, and chameleon: A modular c++ framework for event-based computer vision," *Frontiers in neuroscience*, vol. 13, p. 1338, 2020. 47

[211] C. Scheerlinck, H. Rebecq, D. Gehrig, N. Barnes, R. Mahony, and D. Scaramuzza, "Fast image reconstruction with an event camera," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020, pp. 156–163. 53, 54

[212] D. Gehrig, A. Loquercio, K. G. Derpanis, and D. Scaramuzza, "End-to-end learning of representations for asynchronous event-based data," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 5633–5643. 53

[213] R. Tapia, A. G. Eguluz, J. Martınez-de Dios, and A. Ollero, "Asap: Adaptive scheme for asynchronous processing of event-based vision algorithms," in *2020 IEEE ICRA Workshop on Unconventional Sensors in Robotics. IEEE*, 2020. 56

[214] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014. 58

[215] J. Backus, "Can programming be liberated from the von neumann style? a functional style and its algebra of programs," *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, 1978. 58

[216] R. K. Cavin, P. Lugli, and V. V. Zhirnov, "Science and engineering beyond moore's law," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1720–1749, 2012. 58

[217] K. Berggren, Q. Xia, K. K. Likharev, D. B. Strukov, H. Jiang, T. Mikolajick, D. Querlioz, M. Salinga, J. R. Erickson, S. Pi *et al.*, "Roadmap on emerging hardware and technology for machine learning," *Nanotechnology*, vol. 32, no. 1, p. 012002, 2020. 58

[218] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Communications of the ACM*, vol. 62, no. 2, pp. 48–60, 2019. 58, 89

[219] D. Drubach, *The brain explained*. Prentice Hall, 2000. 58

[220] J. B. Aimone, O. Parekh, and W. Severa, "Neural computing for scientific computing applications: more than just machine learning," in *Proceedings of the Neuromorphic Computing Symposium*, 2017, pp. 1–6. 58

[221] S. Furber and S. Temple, "Neural systems engineering," in *Computational intelligence: A compendium.* Springer, 2008, pp. 763–796. 58

[222] D. F. Goodman and R. Brette, "Brian: a simulator for spiking neural networks in python," *Frontiers in neuroinformatics*, vol. 2, p. 5, 2008. 58

[223] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: Vgg and residual architectures," *Frontiers in neuroscience*, vol. 13, p. 95, 2019. 58, 70

[224] R. VanRullen, R. Guyonneau, and S. J. Thorpe, "Spike times make sense," *Trends in neurosciences*, vol. 28, no. 1, pp. 1–4, 2005. 59

[225] J. Putney, R. Conn, and S. Sponberg, "Precise timing is ubiquitous, consistent, and coordinated across a comprehensive, spike-resolved flight motor program," *Proceedings of the National Academy of Sciences*, vol. 116, no. 52, pp. 26 951–26 960, 2019. 59

[226] J. Luo, S. Macias, T. V. Ness, G. T. Einevoll, K. Zhang, and C. F. Moss, "Neural timing of stimulus events with microsecond precision," *PLoS biology*, vol. 16, no. 10, p. e2006422, 2018. 59

[227] M. J. Berry, D. K. Warland, and M. Meister, "The structure and precision of retinal spike trains," *Proceedings of the National Academy of Sciences*, vol. 94, no. 10, pp. 5411–5416, 1997. 59

[228] P. Reinagel and R. C. Reid, "Temporal coding of visual information in the thalamus," *Journal of Neuroscience*, vol. 20, no. 14, pp. 5392–5400, 2000. 59

[229] G. Buzsáki, R. Llinas, W. Singer, A. Berthoz, and Y. Christen, *Temporal coding in the brain.* Springer Science & Business Media, 2012. 59

[230] G. T. Buracas, A. M. Zador, M. R. DeWeese, and T. D. Albright, "Efficient discrimination of temporal patterns by motion-sensitive neurons in primate visual cortex," *Neuron*, vol. 20, no. 5, pp. 959–969, 1998. 59

[231] C. Carr and M. Konishi, "A circuit for detection of interaural time differences in the brain stem of the barn owl," *Journal of Neuroscience*, vol. 10, no. 10, pp. 3227–3246, 1990. 59

[232] W. Gerstner, R. Kempter, J. L. Van Hemmen, and H. Wagner, "A neuronal learning rule for sub-millisecond temporal coding," *Nature*, vol. 383, no. 6595, pp. 76–78, 1996. 59

[233] G. Haessig, M. B. Milde, P. V. Aceituno, O. Oubari, J. C. Knight, A. van Schaik, R. B. Benosman, and G. Indiveri, "Event-based computation for touch localization based on precise spike timing," *Frontiers in Neuroscience*, vol. 14, 2020. 59

[234] S. Thorpe, A. Delorme, and R. Van Rullen, "Spike-based strategies for rapid processing," *Neural networks*, vol. 14, no. 6-7, pp. 715–725, 2001. 59

[235] N. Iannella and A. D. Back, "A spiking neural network architecture for nonlinear function approximation," *Neural networks*, vol. 14, no. 6-7, pp. 933–939, 2001. 59

[236] W. Maass, "Fast sigmoidal networks via spiking neurons," *Neural Computation*, vol. 9, no. 2, pp. 279–304, 1997. 59

[237] X. Lagorce and R. Benosman, "Stick: spike time interval computational kernel, a framework for general purpose computation using neurons, precise timing, delays, and synchrony," *Neural computation*, vol. 27, no. 11, pp. 2261–2317, 2015. 59, 73, 80, 87

[238] X. Wang, T. Song, F. Gong, and P. Zheng, "On the computational power of spiking neural p systems with self-organization," *Scientific reports*, vol. 6, p. 27624, 2016. 59

[239] C.-K. Lin, A. Wild, G. N. Chinya, T.-H. Lin, M. Davies, and H. Wang, "Mapping spiking neural networks onto a manycore neuromorphic architecture," *ACM SIGPLAN Notices*, vol. 53, no. 4, pp. 78–89, 2018. 61

[240] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*. ieee, 2015, pp. 1–8. 70, 77

[241] Y. Hu, H. Tang, Y. Wang, and G. Pan, "Spiking deep residual network," *arXiv preprint arXiv:1805.01352*, 2018. 70

[242] B. Rueckauer, C. Bybee, R. Goettsche, Y. Singh, J. Mishra, and A. Wild, "Nxtf: An api and compiler for deep spiking neural networks on intel loihi," *arXiv preprint arXiv:2101.04261*, 2021. 70, 77, 78, 80, 82, 88

[243] B. Rueckauer, I.-A. Lungu, Y. Hu, and M. Pfeiffer, "Theory and tools for the conversion of analog to spiking convolutional neural networks," *arXiv preprint arXiv:1612.04052*, 2016. 71

[244] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, and T. Masquelier, "Spyketorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron," *Frontiers in neuroscience*, vol. 13, 2019. 71

[245] J. Göltz, A. Baumbach, S. Billaudelle, O. Breitwieser, D. Dold, L. Kriener, A. F. Kungl, W. Senn, J. Schemmel, K. Meier *et al.*, "Fast and deep neuromorphic learning with time-to-first-spike coding," *arXiv preprint arXiv:1912.11443*, 2019. 71

[246] K. T. N. Chu, Y. Tavva, J. Wu, M. Zhang, H. Li, T. E. Carlson *et al.*, "You only spike once: Improving energy-efficient neuromorphic inference to ann-level accuracy," *arXiv preprint arXiv:2006.09982*, 2020. 71

[247] S. Park, S. Kim, B. Na, and S. Yoon, "T2fsnn: Deep spiking neural networks with time-to-first-spike coding," *arXiv preprint arXiv:2003.11741*, 2020. 72

[248] B. Han and K. Roy, "Deep spiking neural network: Energy efficiency through time based coding," in *European Conference on Computer Vision*, 2020. 72

[249] C. Stöckl and W. Maass, "Recognizing images with at most one spike per neuron," *arXiv preprint arXiv:2001.01682*, 2019. 72

[250] ——, "Classifying images with few spikes per neuron," *arXiv preprint arXiv:2002.00860*, 2020. 72

[251] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. Voelker, and C. Eliasmith, "Nengo: a python tool for building large-scale functional brain models," *Frontiers in neuroinformatics*, vol. 7, p. 48, 2014. 73, 86

[252] C. Eliasmith, "A unified approach to building and controlling spiking attractor networks," *Neural computation*, vol. 17, no. 6, pp. 1276–1314, 2005. 73

[253] J. S. Montijn, G. T. Meijer, C. S. Lansink, and C. M. Pennartz, "Population-level neural codes are robust to single-neuron variability from a multidimensional coding perspective," *Cell reports*, vol. 16, no. 9, pp. 2486–2498, 2016. 73

[254] P. Berens, A. S. Ecker, R. J. Cotton, W. J. Ma, M. Bethge, and A. S. Tolias, "A fast and simple population code for orientation in primate v1," *Journal of Neuroscience*, vol. 32, no. 31, pp. 10 618–10 626, 2012. 73

[255] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. 77

[256] S. Yin, S. K. Venkataramanaiah, G. K. Chen, R. Krishnamurthy, Y. Cao, C. Chakrabarti, and J.-s. Seo, "Algorithm and hardware design of discrete-time spiking neural networks based on back propagation with binary activations," in *2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2017, pp. 1–5. 78

[257] H. Mostafa, B. U. Pedroni, S. Sheik, and G. Cauwenberghs, "Fast classification using sparsely active spiking networks," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2017, pp. 1–4. 78

[258] N. Zheng and P. Mazumder, "A low-power hardware architecture for on-line supervised learning in multi-layer spiking neural networks," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018, pp. 1–5. 78

[259] G. K. Chen, R. Kumar, H. E. Sumbul, P. C. Knag, and R. K. Krishnamurthy, "A 4096-neuron 1m-synapse 3.8-pj/sop spiking neural network with on-chip stdp learning and sparse weights in 10-nm finfet cmos," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 992–1002, 2018. 78

[260] S. Oh, D. Kwon, G. Yeom, W.-M. Kang, S. Lee, S. Y. Woo, J. S. Kim, M. K. Park, and J.-H. Lee, "Hardware implementation of spiking neural networks using time-to-first-spike encoding," *arXiv preprint arXiv:2006.05033*, 2020. 78

[261] R. Massa, A. Marchisio, M. Martina, and M. Shafique, "An efficient spiking neural network for recognizing gestures with a dvs camera on the loihi neuromorphic processor," *arXiv preprint arXiv:2006.09985*, 2020. 78, 82

[262] M. Davies, "Benchmarks for progress in neuromorphic computing," *Nature Machine Intelligence*, vol. 1, no. 9, pp. 386–388, 2019. 80

[263] T. C. Stewart, "A technical overview of the neural engineering framework," *University of Waterloo*, 2012. 80

[264] "Nengo examples: Communication channel," accessed: 2020-06-30. [Online]. Available: https://www.nengo.ai/nengo-loihi/examples/communication-channel.html 80

[265] Y. Liu, J. Zhang, C. Gao, J. Qu, and L. Ji, "Natural-logarithm-rectified activation function in convolutional neural networks," in *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*. IEEE, 2019, pp. 2000–2008. 82

[266] G. D'Angelo, E. Janotte, T. Schoepe, J. O'keeffe, M. B. Milde, E. Chicca, and C. Bartolozzi, "Event-based eccentric motion detection exploiting time difference encoding," *Frontiers in Neuroscience*, vol. 14, p. 451, 2020. 82

[267] V. Fischer, J. Köhler, and T. Pfeil, "The streaming rollout of deep networks-towards fully model-parallel execution," in *Advances in Neural Information Processing Systems*, 2018, pp. 4039–4050. 82

[268] W. Severa, O. Parekh, K. D. Carlson, C. D. James, and J. B. Aimone, "Spiking network algorithms for scientific computing," in *2016 IEEE international conference on rebooting computing (ICRC)*. IEEE, 2016, pp. 1–8. 82

[269] S. G. Cardwell, C. Vineyard, W. Severa, F. S. Chance, F. Rothganger, F. Wang, S. Musuvathy, C. Teeter, and J. B. Aimone, "Truly heterogeneous hpc: Co-design to achieve what science needs from hpc," in *Smoky Mountains Computational Sciences and Engineering Conference*. Springer, 2020, pp. 349–365. 82, 87

[270] J. V. Monaco and R. B. Benosman, "General purpose computation with spiking neural networks: Programming, design principles, and patterns," in *Proceedings of the Neuro-inspired Computational Elements Workshop*, 2020, pp. 1–9. 82

[271] S. Hooker, "The hardware lottery," *arXiv preprint arXiv:2009.06489*, 2020. 83

[272] J. Bhattacharya and M. Packalen, "Stagnation and scientific incentives," National Bureau of Economic Research, Tech. Rep., 2020. 83

[273] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," *arXiv preprint arXiv:2007.05558*, 2020. 83

[274] A. N. Angelopoulos, J. N. Martel, A. P. Kohli, J. Conradt, and G. Wetzstein, "Event based, near eye gaze tracking beyond 10,000 hz," *arXiv preprint arXiv:2004.03577*, 2020. 84

[275] A. Z. Zhu, D. Thakur, T. Özaslan, B. Pfrommer, V. Kumar, and K. Daniilidis, "The multivehicle stereo event camera dataset: An event camera dataset for 3d perception," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2032–2039, 2018. 84

[276] P. de Tournemire, D. Nitti, E. Perot, D. Migliore, and A. Sironi, "A large scale event-based detection dataset for automotive," *arXiv*, pp. arXiv–2001, 2020. 84

[277] D. Gehrig, M. Gehrig, J. Hidalgo-Carrió, and D. Scaramuzza, "Video to events: Recycling video datasets for event cameras," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 3586–3595. 84

[278] A. Z. Zhu, Z. Wang, K. Khant, and K. Daniilidis, "Eventgan: Leveraging large scale image datasets for event cameras," *arXiv preprint arXiv:1912.01584*, 2019. 84

[279] H. Rebecq, D. Gehrig, and D. Scaramuzza, "Esim: an open event camera simulator," in *Conference on Robot Learning*, 2018, pp. 969–982. 84

[280] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in neuroscience*, vol. 9, p. 437, 2015. 84, 93

[281] H. Li, H. Liu, X. Ji, G. Li, and L. Shi, "Cifar10-dvs: an event-stream dataset for object classification," *Frontiers in neuroscience*, vol. 11, p. 309, 2017. 84

[282] M. Bhuiyan, R. Picking *et al.*, "A gesture controlled user interface for inclusive design and evaluative study of its usability," *Journal of software engineering and applications*, vol. 4, no. 09, p. 513, 2011. 84

[283] K. M. Gerling, K. K. Dergousoff, R. L. Mandryk *et al.*, "Is movement better? comparing sedentary and motion-based game controls for older adults," in *Proceedings-Graphics Interface*. Canadian Information Processing Society, 2013, pp. 133–140. 84

[284] L. Hakobyan, J. Lumsden, D. O'Sullivan, and H. Bartlett, "Mobile assistive technologies for the visually impaired," *Survey of ophthalmology*, vol. 58, no. 6, pp. 513–528, 2013. 84

[285] "Google project soli," 2021, accessed: 2021-04-05. [Online]. Available: https://atap.google.com/soli/ 84

[286] S. Gray, A. Radford, and D. P. Kingma, "Gpu kernels for block-sparse weights," *arXiv preprint arXiv:1711.09224*, vol. 3, 2017. 85

[287] T. Gale, M. Zaharia, C. Young, and E. Elsen, "Sparse gpu kernels for deep learning," *arXiv preprint arXiv:2006.10901*, 2020. 85

[288] D. Salvator, "How sparsity adds umph to ai inference," 2020, accessed: 2020-10-05. [Online]. Available: https://blogs.nvidia.com/blog/2020/05/14/sparsity-ai-inference/ 85

[289] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton, "Backpropagation and the brain," *Nature Reviews Neuroscience*, vol. 21, no. 6, pp. 335–346, 2020. 86

[290] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature communications*, vol. 7, no. 1, pp. 1–10, 2016. 86

[291] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, "Event-driven random back-propagation: Enabling neuromorphic deep learning machines," *Frontiers in neuroscience*, vol. 11, p. 324, 2017. 86

[292] J. B. Aimone, O. Parekh, C. A. Phillips, A. Pinar, W. Severa, and H. Xu, "Dynamic programming with spiking neural computing," in *Proceedings of the International Conference on Neuromorphic Systems*, 2019, pp. 1–9. 87

[293] J. Liu, J. Harkin, L. P. Maguire, L. J. McDaid, and J. J. Wade, "Spanner: A self-repairing spiking neural network hardware architecture," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 4, pp. 1287–1300, 2017. 87

[294] A. P. Johnson, J. Liu, A. G. Millard, S. Karim, A. M. Tyrrell, J. Harkin, J. Timmis, L. J. McDaid, and D. M. Halliday, "Homeostatic fault tolerance in spiking neural networks: a dynamic hardware perspective," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 2, pp. 687–699, 2017. 87

[295] C. D. Schuman, J. P. Mitchell, J. T. Johnston, M. Parsa, B. Kay, P. Date, and R. M. Patton, "Resilience and robustness of spiking neural networks for neuromorphic systems," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–10. 87

[296] J. B. Aimone, Y. Ho, O. Parekh, C. A. Phillips, A. Pinar, W. Severa, and Y. Wang, "Provable neuromorphic advantages for computing shortest paths," in *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, 2020, pp. 497–499. 87

[297] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959. 87

[298] N. Imam and T. A. Cleland, "Rapid online learning and robust recall in a neuromorphic olfactory circuit," *Nature Machine Intelligence*, vol. 2, no. 3, pp. 181–191, 2020. 88

[299] E. Musk *et al.*, "An integrated brain-machine interface platform with thousands of channels," *Journal of medical Internet research*, vol. 21, no. 10, p. e16194, 2019. 89

[300] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic, "The risc-v instruction set manual, volume i: Base user-level isa," *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62*, vol. 116, 2011. 89

[301] "Eh2 swerv risc-v core 1.2 from western digital," 2020, accessed: 2020-12-05. [Online]. Available: https://github.com/chipsalliance/Cores-SweRV-EH2 89

[302] A. Zelensky, A. Alepko, V. Dubovskov, and V. Kuptsov, "Heterogeneous neuromorphic processor based on risc-v architecture for real-time robotics tasks," in *Artificial Intelligence and Machine Learning in Defense Applications II*, vol. 11543. International Society for Optics and Photonics, 2020, p. 115430L. 90

[303] S. Furber, "Microprocessors: the engines of the digital age," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 473, no. 2199, p. 20160893, 2017. 90

[304] J. B. Aimone, "A roadmap for reaching the potential of brain-derived computing," *Advanced Intelligent Systems*, p. 2000191, 2020. 90

[305] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7243–7252. 93

[306] T. Serrano-Gotarredona and B. Linares-Barranco, "Poker-dvs and mnist-dvs. their history, how they were made, and other details," *Frontiers in neuroscience*, vol. 9, p. 481, 2015. 93