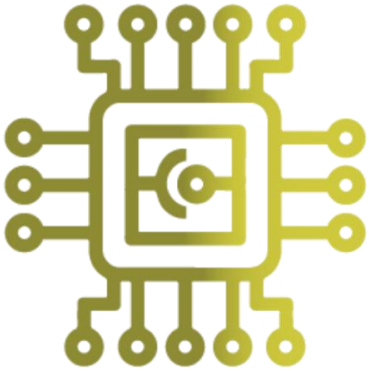

CONVOLVE

Seamless design of smart edge processors

GRANT AGREEMENT NUMBER: 101070374

Deliverable D4.2

Hardware Requirements for Artificial and Spiking Neural Networks

Title of the deliverable	Hardware Requirements for Artificial and Spiking Neural Networks
WP contributing to the deliverable	WP 4
Task contributing to the deliverable	Task 4.2
Dissemination level	RE - Restricted to a group specified by the consortium
Due submission date	03/04/2023
Actual submission date	28/04/2023
Author(s)	Jim Garside (MAN), Mounir Ghogho (UIR), Friedemann Zenke (FMI), Edward Jones (MAN), Simon Davidson (MAN)
Internal reviewers	Sander Stuijk (TUE) Andre Guntoro (BOS)

Document Version	Date	Change
V0.1	19.03.2023	Initial document draft
V0.2	04.04.2023	Formatting a bit more & fleshing out
V0.3	12.04.2023	Some reorganisation for accepting contributions
V0.3.2	14.04.2023	Riccardo's suggestions incorporated: post meeting notes added (Believed) reconciled with other input at this version.
V1.0	28/04/2023	Final formatting
v.1.1	16/05/2023	Re-editing in progress
v.2.0	31/05/2023	Re-edit complete
V3.0	30/06/2023	Second re-edit complete

TECHNICAL REQUIREMENTS FOR HARDWARE ACCELERATORS FOR NEURAL NETWORKS

Table of Contents

1 Purpose of this Document

This document is intended to provide technical requirements of Neural Network (NN) accelerators for implementation in WP2. These are derived from two classes of source: amalgamated input from industrial partners and 'local' research from WP itself (Figure 1).

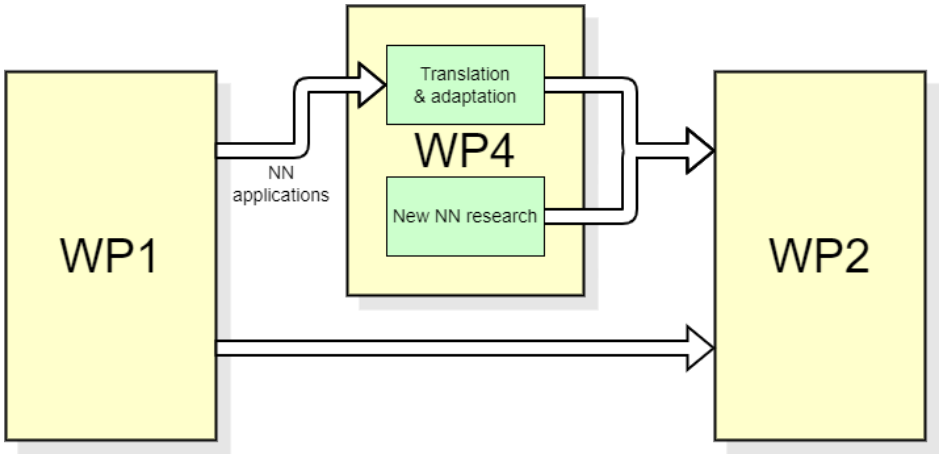

Figure 1 How WP4 fits with its dependent work packages, focusing on the hardware development

The scope of this document includes:

- A summary of industrial partners' application interest areas
- A summary of industrial partners' current and future technical interest areas
- Identifying common functions where there would be good "bang for the buck" with hardware acceleration
- Identifying new functions for support in less-developed areas — particularly in SNNs — and their commonality with the functions above

Some of these topics are already well defined; others, particularly the developments in Spiking Neural Networks (SNNs), are still evolving.

2 Context

Artificial Neural Networks (ANNs) are used increasingly in numerous applications which deal with imprecise inputs. Major strides in Neural Network (NN) applications have been made recently, largely due to the availability of increasing processing power – with a corresponding cost in energy. Producing dedicated hardware accelerators to provide much of the core functionality of existing algorithms will give correspondingly higher energy efficiency. A particularly clear initial example would be a more appropriate selection of data types.

Beyond accelerating existing algorithms, the prospect of new processing paradigms suggests even greater energy savings may be achievable. It is estimated that the energy-efficiency of biological brains may be around *five* orders of magnitude greater than current ANNs and so reaching a fraction of this could produce significant power-consumption benefits. This inspires the development of Spiking Neural Networks (SNNs).

The first function of WP4 is to amalgamate the requirements of existing NN applications to identify a practical subset of operations which may benefit a wide range of needs. Industrial partners have supplied example applications and these have been analysed with their component functions consolidated into practical components. This may also involve some adaptation of the original algorithms to allow implementation on realistic, low-power processing elements. Flexibility of the resultant hardware is desirable, providing this does not seriously compromise its overall efficiency. The outcome will need to be matched to the system requirements, encompassing architecture as well as processing, notably reducing costly data transport by moving much of the repetitive work close to its appropriate memory when feasible and supporting operand movement efficiently when this is not practicable.

The second function of WP4 is to devise more energy-efficient alternative approaches to processing. Synthetic 'Neural Networks' come in at least two 'flavours': both 'conventional' Artificial Neural Networks (ANNs) and Spiking Neural Networks (SNNs) [18] are considered here. SNNs share some characteristics with ANNs – notably in their complex interconnectivity and imprecise (low precision) computing needs – but have some fundamental differences in that they possess *state* that evolves through time, as opposed to working on static (or 'staticized') data. An analogy would be that SNNs are procedural programming to ANNs functional programming, which potentially makes SNNs more appropriate for real-time applications. It is possible to translate existing ANNs into SNNs with equivalent function, although the results are not always efficient. WP4 is attempting to identify more suitable paradigms which are closer to the biological 'prototype'! Ideally, the same hardware accelerators would adapt to support both ANN and SNN frameworks.

A related objective is to determine appropriate data types for implementation based on commonality within the set of use-cases. Since NN operations are inherently approximate so the cost of computation, and of both data storage and transfer, can be significantly reduced by using appropriate representations. Such types can apply to both ANNs and SNNs and synergy of function is sought here.

A final purpose applicable to both ANNs and SNNs is to investigate *architectures* to support Dynamic Neural Networks (DyNNs) [11] — NNs which are configurable to degrade gracefully, either in active area or time, such that the processing (energy) demand can be decreased whilst the precision of the processing remains 'good enough' for the users' applications. This will require

network configurability and may need continuous 'profiling' of operation to relate effort to performance. A related issue is the ability of an NN to 'learn' continuously, rather than being (expensively) 'trained' once, at its inception. This shares some characteristics with the DyNN requirements in performance monitoring and feedback. Biological networks (unlike current ANNs) are continually adapting.

3 Users' requirements

3.1 WP1 applications

The first function of this document may be summarised as requirements from WP1 having been translated into NN terms for input to WP2. Industrial partners' NN applications' requirements are summarised in Table 1, expressing the current interest for ANNs and SNNs in each application area from the point of view of the industrial partner. The table also addresses the perceived importance of DyNNs for energy saving and dynamic adaptability ('learning') in each case. The ability of an NN to 'learn' in parallel with inference should enhance its adaptivity to changing conditions, both in the environment it is processing and in its internal configuration: the latter of these can be a significant contributor to both fault tolerance and provide guidance for dynamic (re)configuration.

Partner	Application	ANN	SNN	DyNN	Learning
BOS	Siren detection and tracking	Yes	Yes	Yes	Maybe
GNA	Deep noise suppression	Yes	Yes	Yes	Yes
GNA	Speech quality prediction	Yes	Yes	No	No
TASE	On-board computer vision	Yes	Yes	No	Yes
VIN	Traffic analysis in live video surveillance	Yes	No	Maybe	Maybe

TABLE 1 APPLICATIONS REQUIREMENTS WITH RESPECT TO NEURAL NETWORK IMPLEMENTATION

Table 2 enhances the requirements from Table 1 with some further characteristics which are applicable to any NN. The terminology used here includes:

Pruning: the removal of 'insignificant' connections either post-training or, iteratively, during training. This reduces the connectivity of the network which can save effort.

Quantisation: the reduction in the precision of value representation.

Binarisation: quantisation to the Boolean level where connections simply exist or not.

Distillation: iteration of training whereby an initially trained NN is used to train a smaller NN rather than resorting to the original data.

Partner	Application	Pruning	Quantisation	Binarisation	Distillation
BOS	Siren detection and tracking	Yes	Yes	Yes	
GNA	Deep noise suppression	Yes	Yes	Maybe	Yes
GNA	Speech quality prediction	Yes	Yes	Yes	Yes
TASE	On-board computer vision	Yes	Yes	Maybe	Maybe
VIN	Traffic analysis in live surveillance video	Yes	Yes	Maybe	?

TABLE 2 EXTENSION OF TABLE 1: FINER DETAILS

The following section defines some more specific requirements for hardware development and is a summary of the requirements listed in deliverable D1.1, included here for ease of reference.

3.2 Deep noise suppression & speech quality prediction (GN Audio)

GN Audio put forward two related applications as use-cases. Though similar, there are specific constraints for each and so these are listed separately.

3.2.1. Deep Noise Suppression

Outline of application

Deep Noise Suppression (DNS) or *Speech Enhancement* aims to improve the quality of both Tx (uplink) and Rx (downlink) speech signals by reducing background noise, thereby improving their quality or intelligibility. This is a challenging task, due to the vast number of complex acoustic situations that may arise in the real world, such as the presence of an undesired speaker (referred to as a "jammer") close to the main user's microphone. Thus, Speech Enhancement can be seen as an umbrella term for more specific denoising tasks. It is considered a "hot topic" in the wider communication and computer industry, with large companies and academia dedicating massive resources to it[1] although not necessarily focusing on edge processing.
Requirements come mainly in terms of power, overall processing requirements, latency for I/O audio processing.

Application requirements

The requirements presented here for Speech Enhancement are largely based on those derived and stated in Deliverable 1.1[2] (using an ANN) and are summarized in Table 3. Although the initial implementation should be an ANN, GN Audio are open to the possibility of re-implementing the network as an SNN.

[1]https://arxiv.org/pdf/2202.13288.pdf
[2]WP1_Deliverables_v2

For online learning approaches the following should be supported:

- Effective backpropagation
- Teacher-student network updates

Metric	Unit	Value
Latency	I/O latency RTF[3]	< 2 ms < 0.8
Power Consumption	mW / MMACs Overall	< 0.1 < 0.2 W
Processing needs	MMACs	> 4350
Memory	MB	> 75.5
Memory BW	MB/s	> 72 MB/s
Quality	VISQOL[4] Mean-opinion score (MOS)	> 4
Precision		Float32, Float16, Float8(?), Int8

TABLE 3: LIST OF REQUIREMENTS FOR SPEECH ENHANCEMENT USE-CASE

Hardware implications

The processing and memory bandwidth requirements here are not severe. The size of the memory implies the need for SDRAM which is comparatively power-hungry. It will be difficult to accommodate an on-chip SRAM of this size; it may be possible if the requirement were reduced, for example by using smaller variable types. In any case the energy requirement will be dominated by data movement which will need to be carefully managed; this may be an application that may benefit from optimisation using the techniques of Dynamic Neural Networks (DyNN). Hardware developed to support the principles of DyNN is likely to be of wide applicability.

The desire to explore spiking networks for this application motivates the development of a pipeline to support SNNs. Ideally, this should re-use as much as possible from the ANN design, to maximise productivity. The principal point of distinction between an ANN and an SNN is that in the latter time plays a fundamental role, both in the decay of state variables (such as the membrane potential) and in the modelling of inter-neuron delays. In the case of the ANN time plays a much less important, although there is an implied time base in ANNs networks that employ recurrent connections (such as GRU and LSTM). For the SNN the inclusion of interneuron delays in the spike delivery system can consume significant on-chip resource, but also renders the SNN a potentially more natural fit to process data that is inherently timed.

The inclusion of learning as a requirement adds a new dimension to the hardware development. Although generic 'back propagation' is stated as a required learning algorithm, alternative learning schemes might be possible such as Backpropagation Through Time (BPTT) and Forward Propagation Through Time (FPTT). For the spiking network techniques such as surrogate gradients and eProp would be the best known and hence the first to be considered. Before developing hardware to support these alternatives, they would need to be modelled at the algorithmic level and assessed by the WP team in consultation with GNA to ensure that they meet performance goals as well as energy constraints.

[3]https://devopedia.org/speech-recognition
[4]https://arxiv.org/pdf/2004.09584

3.3.1 Speech Quality Prediction

Outline of application

The field of speech quality prediction can be divided into full-reference (also known as 'intrusive'), which requires a clean reference signal to compare against, and reference-less (also termed 'non-intrusive'), which operates on the given signal only. While there exist several full-reference metrics based on DSP or perceptual models (PESQ, POLQA, ViSQOL etc.) that correlate well with a human-attributed MOS, these are often computationally expensive and it might be beneficial to "approximate" them using an optimized, ANN-based implementation running on an accelerator. However, due to the lack of reference signals in real-world scenarios, most of the focus will be on reference-less methods.

Reference-less methods often rely on statistical models, machine learning algorithms and blind signal processing techniques to estimate speech quality. These approaches can analyse speech signals and extract relevant features, such as spectral or temporal characteristics, to predict the perceived quality without requiring a reference signal.

Application requirements

The requirements for Speech Quality Prediction are based on numbers from deliverable 1.1[5]

Metric	Unit	Value
Latency	I/O latency RTF	< 1 ms < 0.5
Power Consumption	mW / MMACs Overall	< 0.1 < 0.01 W
Processing needs	MMACs	> 5.3
Memory	MB	> 0.6
Memory BW	MB/s	> 17 MB/s
Precision		Float32, Float16, Float8(?), Int8

TABLE 4: AS IN TABLE 3, MMACS REFERS TO MILLION MULTIPLY-ACCUMULATES PER SECONDS, MB STANDS FOR MEGABYTE (1 MB = 1024 KBYTE) AND RTF STANDS FOR REAL-TIME FACTOR.

Hardware implications

The application here looks readily feasible, the challenge being the extreme power consumption requirement, which would be dominated by data movement and therefore primarily the memory architecture. The memory requirement can be met using SRAM in contemporary technology although the address and data buses are likely to contribute significantly to energy costs.

The non-standard data types also offer the opportunity to develop custom hardware to compute with greater efficiency that would be achieved using generic hardware such as a GPU.

3.4 Acoustic scene analysis, focused on siren detection and tracking (Bosch)

Outline of application

Within the scope of this use-case, we would like to extract information about emergency vehicles from an acoustic scene. In a first step, the presence of an emergency vehicle should be detected based on siren sounds and, in a second step, its spatial position should be reconstructed.

[5] WP1_Deliverables_v2

For detection, the following network is proposed:

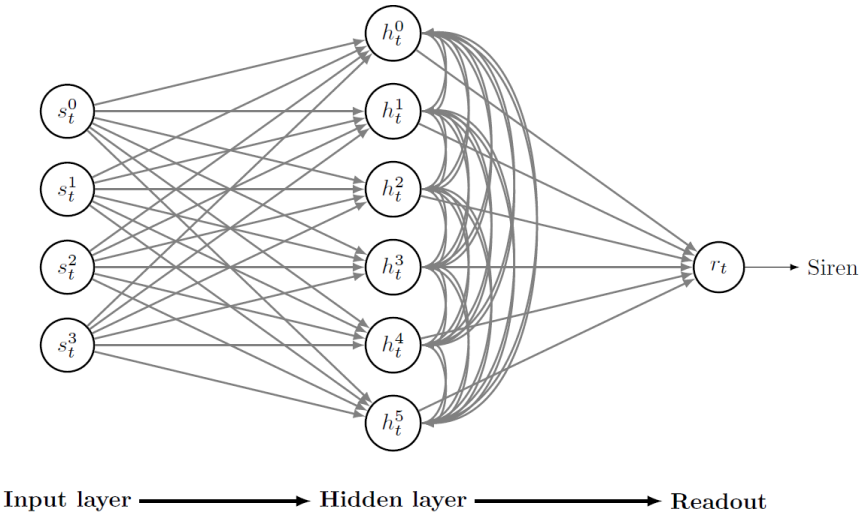

Figure 2 Network for detecting a siren given audio Mel-Spectrogram data as input

The key components of this architecture are:

- an input layer implemented by a Mel-Spectrogram of the audio signal,
- a hidden layer (set of hidden layers) composed of GRUs or LSTMs,
- and a readout comprising of a single linear unit with sigmoidal activation.

For siren tracking the following network has been proposed:

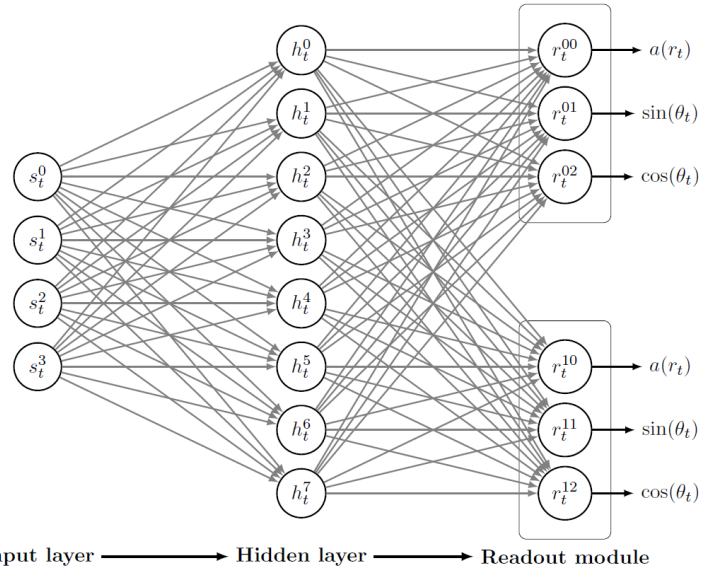

Figure 3 Network to track a siren given input data consisting of multiple channels of Mel-Spectrogram data

The key components of these tracking networks are:

- an input layer implemented by a Mel-Spectrogram of multi-channel audio signals,
- a hidden layer composed of either GRUs or LSTMs,
- a set of readout modules each comprising of three linear units with sigmoidal and tanh activation functions to constrain the range of the prediction to their physical plausible range.

Application requirements

Here, we summarize the requirements for the siren detection and tracking use-case based on the numbers reported in deliverable 1.1.

Latency	Prediction	< 100 ms
Power consumption	Total	< 100 mW
Precision		Float32, Float16, Int8

TABLE 5: REQUIREMENTS FOR THE SIREN DETECTION AND TRACKING USE-CASE.

It has been proposed that binary quantization of activations will be explored in the CONVOLVE project for this use-case.

Hardware implications

This represents a relatively small network, although the hidden layer consists of recurrent units (GRU and LSTM units) rather than simple sum-of-products neurons. The majority of the computation for this network is the generation of Mel-Spectrogram data from time series audio data. While a neural network might be trained to perform this task, it is unlikely that such a solution would be better than a dedicated accelerator using the standard algorithm. This may be available as an off-the-shelf component and it will not be necessary to design one from scratch. Thus we can focus on the neural network itself, which contains no non-standard components. A generic ANN pipeline design would be sufficient for the neural network component of this use-case.

The proposed use of binary activations reduces the size of the data bus carrying the activations (rendering them equivalent to spikes) and also simplifies the compute, since the MAC operation is replaced with a simpler addition operation. In this case the ability to use components intended for a purely spiking network (single bit activations corresponding to spike-or-no-spike) there is opportunity to re-use components in the binary activation ANN pipeline.

3.5 On-board Computer Vision (Thales)

Outline of application

The sheer volume of data that can be gathered by a satellite in low-Earth orbit is huge and processing that data is currently done almost exclusively on the ground. This process is constrained by the ability of the satellite to store the images until the satellite passes over a point on the Earth's surface where the data can be downloaded. This delay corresponds to a potentially crucial time lag before any meaningful inference can be drawn and acted upon.

Figure 4, below, shows the simplification that could be achieved by processing the data on the satellite itself:

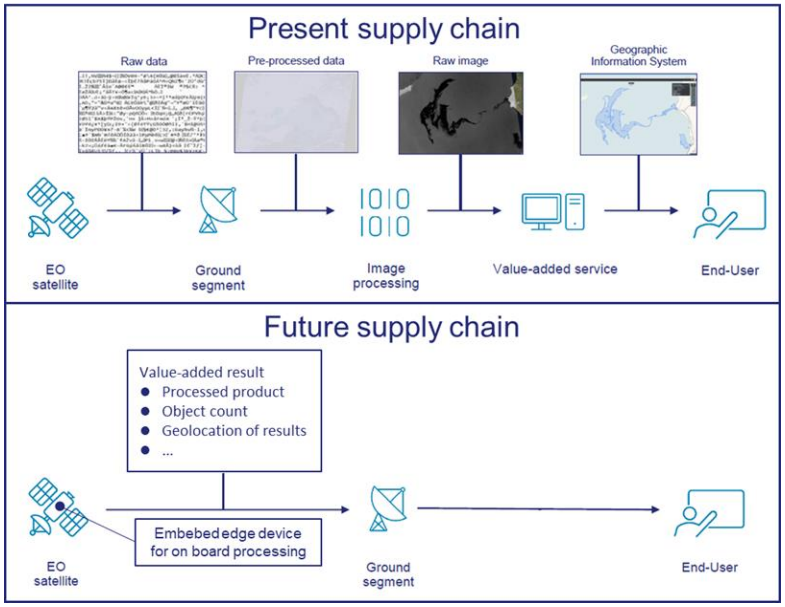

Figure 4 Illustration of the potential to improve both the responsiveness and data bandwidth for space-bound image processing

Application requirements

The end goal is to locate the image processing on the satellite itself, leading to significant reductions in latency. The camera will take images in stripes and build up an internal image over time. Since the image width is large (but not stated here for commercial reasons) this will require significant buffering before processing can begin.

The current solution employs conventional GPU technology to post process the images after download. Specifically, a Nvidia Tesla M60, with 8 GB of memory and a max power consumption of 300W is used. The neural network in the current system is approximately fifty layers deep and contains a range of layer types, including CNNs, recurrent elements and densely connected layers. It also makes use of a range of different activation functions. These capabilities would need to be provided for any on-board solution.

Hardware implications

The current solution is an ANN, but Thales are keen to explore the possibility of a spiking network. Thus an initial ANN pipeline would be most appropriate, but with a modular design that will allow alternative implementations of components to be used to configure the design and a spike network. The range of connectivity types (CNN and dense) with both local and shared memory storage for weight/kernel information must be supported, as well as the required list of neuraon activation functions.

Low power is critical to moving this processing to a satellite, where power management is at a premium. An early area of investigation is the memory requirement and the best way to support the amount of intermediate storage required for partially-captured images.

3.6 Video-based traffic analysis (Vinotion)

Outline of application

ViSense is an edge-based video analysis system that reads, analyzes and processes real-time video from a standard CCTV/RGB camera (24/7 measurements) for surveillance, traffic management, incident detection, crowd management and various other traffic cases.

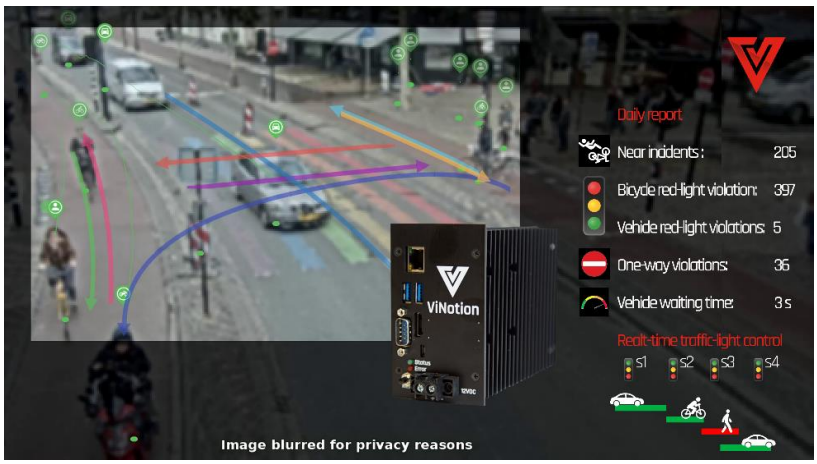

Figure 5 Visualisation of the use case

Application requirements

The current product runs 1 full HD video stream with 2 x 512x512 ANN template at 4 fps including tracking at 30 fps with more than 100 objects on an Nvidia Jetson TX2 at a typical consumption of 15 W. Most of the resources of the TX2 system are currently reserved for a single ANN performing object detection and classification (mainly GPU resources) and for a tracking algorithm (combination of GPU and CPU resources). Video decoding is currently offloaded to a dedicated on-board hardware.

Current HW elements are:

- 256-core NVIDIA Pascal™ GPU architecture with 256 NVIDIA CUDA cores
- Dual-Core NVIDIA Denver 2 64-Bit CPU
  Quad-Core ARM® Cortex®-A57 MPCore
- 8GB 128-bit LPDDR4 Memory
  1866 MHx - 59.7 GB/s
- 32GB eMMC 5.1
- Accelerators for H264 encoding and decoding.

The current solution consumes approximately 20 watts.

Hardware implications

In CONVOLVE we would focus on the ANN portion of the system, which is a 2x512x512 network. From table 1 and 2, we note that Vinotion are focused on ANNs not SNNs, with potential interest in Dynamic NNs and binarised networks. The basic network requires would appear to map to the type of hardware-based ANN pipeline that would also be appropriate for other use-cases on CONVOLVE.
Analysis of the required weight precision and degree of pruning of connections would need to be performed to establish the savings that can be made to the fetch from weight memory.

3.7 Summary of application requirements

There is a wide spectrum of requirements listed in this section. Unsurprisingly, the most common factor is the demand for low – sometimes *extremely* low – energy. The processing needs are typically not hugely challenging for hardware implementation. There are particular needs for data movement both to/from memory and I/O and this is the biggest challenge.
In terms of processing it is clear that there is interest in less precise data types (i.e. fewer bits to move) and this should yield a noticeable contribution. Nevertheless computing units situated 'in' (near) memories are appropriate for most of these applications.

Flexibility of processing is clearly required, either by multiple processing units or by a high degree of reconfigurability in operation, whilst still fitting within the overall architecture. Network demands include both ANNs (with DyNN functionality) and SNNs (as a proof of concept at this stage). Data storage that should be supported includes both local co-efficient (for CNNs and small dense networks), and more remote access to larger shared memory (for fully connected layers of neurons).

3.8 SSN development

Beyond the requirements derived from pre-existing applications, WP4 is researching new developments in SNN representations and implementation. These requirements will seek to exploit commonality with structures supporting ANNs, with a priority given to the minimisation of data movement.

4 Current Research and State-of-the-art in Neuromorphic Hardware

4.1. Hardware-focused dynamic neural networks (UIR)

Dynamic Neural Networks (DyNNs) have the advantage of being adaptable to diverse hardware constraints and changing computational budgets over time. They allow tangible reduction of computations and resource usage. However, current deep learning hardware and libraries are mostly optimized for static models, which are not automatically transferable to DyNNs impeding the practical benefit of such architectures. As an example, many DyNN techniques for spatial data utilise sparse computation (e.g. sparse convolutions), which are still inefficiently implemented on modern hardware due to the memory access bottleneck [11].

Many studies rely on injecting sparsity into computation as in convolutions [31, 28]. This approach is known to yield limited speed-up on modern hardware due to its irregular memory access and computation pattern: vector accelerators such as GPUs work most efficiently when fed with a stream of contiguous values that keep the data paths 100% occupied. Sparsity and irregular data

patterns reduce the efficiency of this computing model because it is difficult to keep the data paths fed with data and the memories providing the data cannot supply it at 100% efficiency without implementing fine-grained access to individual data values, which has an overhead in terms of addressing.

In the case of dynamic pruning, where some channels are dynamically disabled and given that all weights are loaded to the accelerator before any processing occurs, the accelerator is filled with weights that are never used. Hence, the more pruning, the more memory I/O bandwidth is wasted because of unused loaded weights. Overall, runtime calculations in terms of MACs will be reduced, but the gains in terms of performance and energy efficiency will be limited by the overhead in I/O access to the memory. This problem also applies to the early exit or skip layers strategies, since devices that are optimised for static neural models will potentially load the weights of the consecutive layers proactively, while some layers might be skipped or never used [16].

To tackle this issue, one avenue is to redesign the network topologies and learning algorithms to integrate sparsity [24] or reflect from the hardware side by proposing different routing strategies of features maps; e.g. storing feature map batches within memory buffers to reduce off-chip accesses [14]. Another more promising avenue is to codesign algorithms and hardware for accelerating DyNNs [32]. In this context, reconfigurable arrays play a major role thanks to their flexibility and reconfigurability. Various papers focused on one aspect of dynamicity to deploy such as work by Huang et al. with deformable convolutions [15] and Paul et al. with early-exit networks [22].

Based on the conception of early-exiting networks, it is conceivable that deeper layers in such types of networks will be less frequently used than the early ones. Thus, by exploiting this property, and due to the limited FPGA memory, Paul et al. suggested in [16] storing the first convolutional layers' parameters in the on-chip BRAM while keeping the remaining set of parameters and results in the off-chip DDR memory. A DMA is set to transfer data between the external DDR and the on-chip buffers and a controller is configured to generate signals for the different implicated modules (e.g. convolution operation, activation function, pooling).

Moreover, since convolution operators are the most resource-demanding, a convolution engine was designed through 16 processing elements responsible for MAC operations. As for scheduling, computations are parallelised; the DMA loads fragments of input feature maps as well as kernel weights into the on-chip buffers for convolution. Once the feature map is output, the DMA fetches another fragment of the input with a new set of kernels.

Gao et al. proposed in [33] a potential solution, coined DPACS, from both the algorithm and hardware perspectives. From the algorithm standpoint, DPACS uses a dynamic pruning block (DPBlock) that produces a fine-grained spatial mask and a coarse-grained channel mask dynamically through trained 1x1 convolution and fully connected layers respectively. The two masks are designed to be shared among all the layers within a DPBlock helping to amortise the mask generation overhead and facilitate efficient hardware processing across the shared layers. As for the hardware implementation, a network-specific dynamic data flow engine is designed to process the irregular data flow arising from the pruning operations with a flexible pipeline using an elastic line buffer. Finally, the coarse-grained channel mask is designed in such a way that it can be precomputed using only the sparse output of an earlier DPBlock, and that the channel masking groups will fit the weight storage layout in memory. The number of IO memory accesses can thus be reduced to only those necessary for loading needed weights.

While FPGAs provide enough flexibility for hardware/software codesign in the context of DyNNs some studies criticized using FPGAs for runtime applications because of the overheads of hardware configuration. Bouzidi et al. proposed an alternative that capitalized on the supported Dynamic Voltage and Frequency Scaling (DVFS) features of traditional CPU and GPU hardware, where the DVFS settings are jointly optimized with the early-exiting features to minimize energy consumption [4].

In conclusion, few works in the literature address the potential implementations of DyNN on hardware devices, which opens interesting investigation paths in state-of-the-art research.

4.2. Hardware-focused Spiking Neural Networks

A key feature of biological neural networks is the sparsity of connectivity and sparsity of activity, where sparsity means that only a small subset of elements are non-zero. Accelerating arbitrary sparse connectivity and the efficient processing of sparse-in-time events is a desirable feature of any hardware aimed at accelerating neural network models.

4.2.1 Neuromorphic Hardware

Since the work of Rosenblatt in the 1960s, implementing neural network models more directly in hardware has been an area of research interest [25]. The rationale behind this has been that efficiency and performance gains may be manifest by reducing the number of levels of abstraction between algorithm and physical computing substrate. Broadly, the acceleration of ANNs has been carried out by expressing ANN models as linear algebra and exploiting hardware for accelerating matrix-vector operations, GPUs.

Though this approach can be applied to spiking models, the sparsity of activity in space and time that is common in spiking models can make this inefficient. Neuromorphic computing systems seek to accelerate such sparse operations by mimicking the structure and connectivity patterns seen in biology at a hardware level.

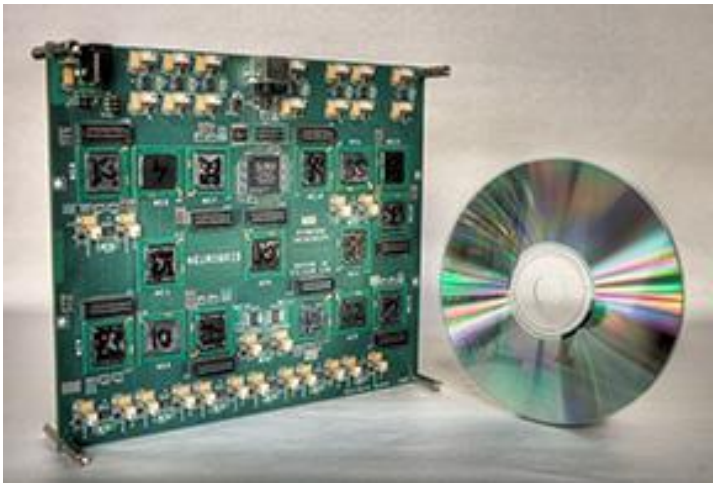

Figure 6 Neurogrid analogue neuromorphic processing board, developed by Stanford University. It models neurons at the level of ion channels

Work on neuromorphic hardware systems that emulated the behaviour of biological neurons was pioneered by Mead in the 1980s and 1990s (Mead, 1990). Direct descendants of this work are the analogue and mixed signal approaches seen in projects like Neurogrid [3] and BrainScaleS [20, 27]. The analogue electronics approach has the potential to be significantly lower power than an equivalent digital network. However, there are several limitations in this approach.

First, the manufactured characteristics of analogue hardware vary much more than those of digital. Since the functionality can be distorted by such variations, analogue circuits are more difficult to use where identical and consistent behaviour between chips is required. Second, since this variability becomes more pronounced at smaller geometries, analogue circuits are not represented at the more advanced technology nodes, such as 22nm and below. This limits their competitiveness. Third, while digital circuits can be time-multiplexed to save on area, analogue circuits do not typically make use of this technique. Instead physical neurons are laid out one-to-one corresponding to the target network. This saves on data movement (and hence energy) since the weights can be located where they are used, but ultimately limits the size of network that can be implemented in an analogue process. It is thus no surprise that the majority of commercial neuromorphic products are digital.

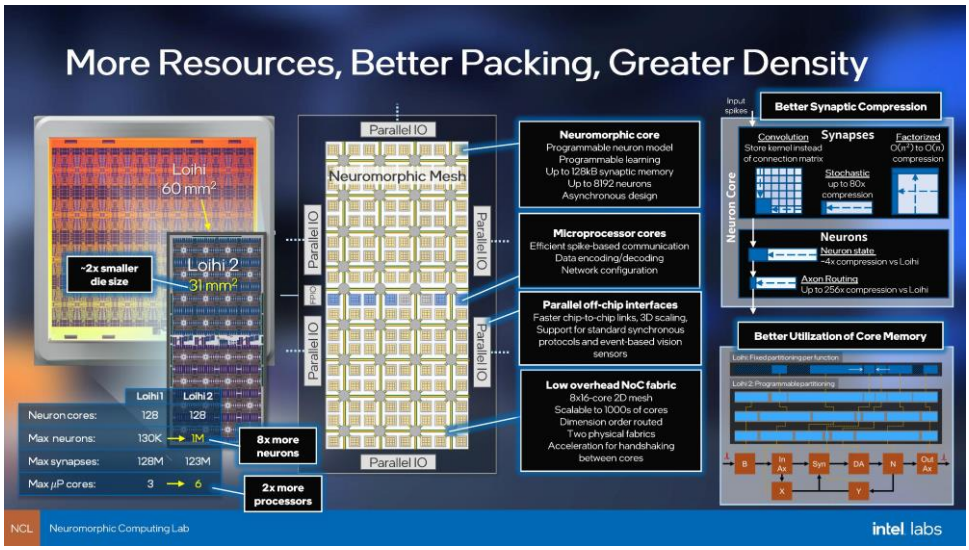

Figure 7 Marketing image of the new Loihi-2 neuromorphic processor

Low-level digital neuromorphic approaches include TrueNorth [1], Loihi [6] and Loihi 2 [10]. SpiNNaker [8] and SpiNNaker2 [9]. are examples of systems that define neuron models at a higher level of abstraction to permit research and rapid iteration at the neuron and network level. The SpiNNaker architecture is treated in more detail next, but is distinct in that it is a many-core system that emulates neurons in software, in contrast to the Loihi and TrueNorth chips that implement neurons directly in hardware. Recent smaller start-ups such as GrAIMatter Labs are now marketing a spiking neuromorphic processor, NeuronFlow [21] that emphasises the sparseness inherent in the spikes, the connections and the events as methods to lower the power consumption of their image processing hardware.

4.2.2 The Human Brain Project: BrainScaleS and SpiNNaker

Considerable recent development work in simulating and researching SNNs has been performed as part of the Human Brain Project (HBP), a major EU-funded project currently in its closing phases. This included two hardware development streams: BrainScaleS (Heidelberg) [20] which built analogue neuron models and SpiNNaker [8] where neuron models are implemented in software with a degree of digital hardware acceleration. As part of the latter work the SpiNNaker2 chip was constructed as a partnership between Manchester and TU Dresden [13]; this is now being moved towards commercialisation. Experience from the chip's predecessor

was incorporated in the form of hardware acceleration for some of the 'bottlenecks' in the form of mathematical accelerators and improved networking and communications.
There is much more to be done here, in particular supporting data types which are better adapted to neuromorphic models, probabilistic computing and the adaptation of spike routeing to accommodate network robustness, dynamic switching (for DyNN support). The most common code executed on SpiNNaker is the interrupt-driven response to arriving packets (containing spikes from neurons resident on other processors in the machine). Each such spike arrival triggers a DMA from shared memory to retrieve a row of the synaptic matrix. The returning row then triggers the unpacking of the contiguous data to extract weight, delay and target neuron index information for the sparse synaptic connections that are targeted by the spike. This data transformation process could be accelerated with a configurable hardware block that would significantly increase the energy efficiency of simulation. This work would be applicable to both ANNs or SNNs.

Figure 8 A 48-node SpiNNaker board, release in 2011, consists of 48 nodes each containing 18 ARM 968 cores and proprietary routing technology

While SpiNNaker supports some forms of learning in hardware, including neuromodulated Spike-Timing Dependent Plasticity (STDP) [7] and e-prop [2], these methods exploit the fact that neuron and synapse modelling is carried out in software with hardware acceleration being dedicated towards the processing of spikes; the SpiNNaker architecture provides no hardware acceleration for learning. From the research that has been carried out using SpiNNaker as a platform, it has become clear that support for learning is highly desirable. This may be achieved through real-time activity monitoring to facilitate synaptic adaptation.

The CONVOLVE project will investigate novel learning algorithms for spiking networks. We will consider their hardware implementations if time permits, but will assume that learning is handled in software as a default position.

4.3 Hardware constraints from online learning algorithms

For Backpropagation Through Time (BPTT) in SNNs it is desirable to make both the forward pass and the backward pass sparse. Normally, surrogate gradients have a dense backward pass, but it can be implemented sparsely without loss of accuracy [23]. This could be implemented in hardware as a custom CUDA-like kernel. However, it still has backward locking and cannot be run online.

One approach to achieving online learning is to use an algorithm with even smaller memory demands than BPTT, which can be done with approximate Real Time Recurrent Learning (RTRL)[34]. The resulting accuracy is not as high as BPTT, but it has the same time complexity without the requirement of storing the activation history. Modern approximations of RTRL rely on synaptic eligibility traces to solve the temporal credit assignment problem, which usually necessitates maintaining at least one additional dynamic state variable for each synapse. Depending on the precise loss function used in SNNs, these traces often need to be updated at every time-step. Since these variables are used during learning, they typically must be full precision, although reduced bit-width implementations have yet to be explored in detail and may be feasible. Specifically, synaptic eligibility traces, as used in SuperSpike [35] and e-Prop [2] create extra demand on both memory and computing and thus would offer substantial savings if some of this complexity could be accelerated at the hardware level.

Similarly, continual learning algorithms often require additional state variables to store importance values for each model parameter, similar to conventional optimizers such as Adam [36], which implement per-parameter learning rates. While these variables do not need updating at every time step, they often need to be read at every step, thereby creating additional memory traffic.

4.4 Other Related Research Areas

4.4.1 Representations

In ANNs, weights and activation values can take varying precisions at inference, but typically high precision is necessary for the back propagation of gradients, to accurately assign credit in the model. Two key research questions arise in relation with data representations and SNN models:

1. What precision and numerical representations are necessary at the implementation level of SNN models?
2. How can populations of spiking neurons efficiently and robustly encode information?

Question 1 is very similar to questions that have been asked on the route to quantising ANNs. If one wishes to implement a spiking model, how should the available bits best be used? For instance, a subsidiary question that may be asked is how much precision do I need in synaptic weights to maintain the same functionality as a 'full precision' model? A further consideration here is how much movement of numerical values is necessary and can computations be abstracted, and populations of neurons modelled by hardware units without requiring access to, or modelling, low-level details. One example of a function that might be accelerated, which goes beyond modelling individual neurons, is the function of finding the top 'k' integers in a block of

memory. This function is analogous to a k-winner-take-all (k-WTA) basic operation, but could remove the need to simulate neuron state evolution in detail.

Question 2 looks at a higher level of abstraction, at the network function scale and looks more to understand functional links between ANNs and SNNs. This question also relates closely to understanding mappings between ANNs and SNNs, practically seen in ANN-to-SNN conversion (see D4.1). Because of the high noise environment of the brain, it seems unlikely that the spiking patterns of biological neural networks use dense numeric binary codes like the integer or floating-point ones seen in traditional architectures, more promising avenues are the high dimensional vectors seen in vector symbolic architectures (VSAs) [17, 26].

5 Derived Hardware Requirements

In this section we present a list of areas where custom hardware can have significant impact on the project goal of low-power computation using neural networks (either ANNs or SNNs). When asked about their interest in neural networks, all of the industrial partners expressed a preference for ANNs and were interested in trying out techniques relating dynamic Neural Networks. Therefore the initial focus of the hardware development should be to accelerate these types of network. Most of the industrial partners have expressed at least an interest in spiking neural networks but they are not sure how to harness these relatively immature compute models for their applications.

The immediate 'line of attack' is to look at existing (& evolving) applications algorithms and solutions, to profile them and identify particular bottlenecks. These are then decomposed into their common components, paying attention to data types (and where they can be compromised) and the functions that conventional processors struggle to support. Highly repetitive ('vector') operations may be identified as candidates for processing 'in memory' – e.g. by a programmable coprocessor or Coarse Grain Reconfigurable Array (CGRA).

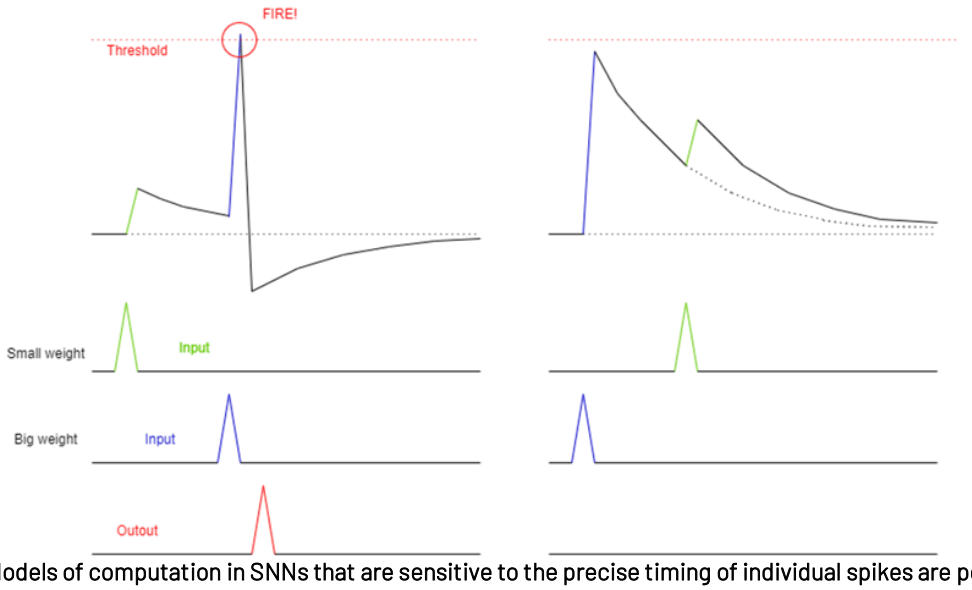

Figure 9 Models of computation in SNNs that are sensitive to the precise timing of individual spikes are potentially more energy efficient than the more common rate-coded SNNs

Design principles for SNNs are less well developed than those for ANNs and have not attracted as much industrial attention, to date. Most current SNN implementations use *spike rates* to convey information, although it is clear from processing latency alone that biological systems (largely) do not. Harnessing the temporal information in spike streams increases the information density in the representation. Consequently only a fraction of the number of spike events would be required

and thus a fraction of the calculation effort, along with the lowered latency. This should be investigated with high-level models, initially of simple functions, with the objective of mapping the results to accelerator hardware.

The training of NNs is an aspect in need of significant improvement. In ANNs this is a very expensive overhead with the backpropagation of errors as the network parameters change to minimise the error in each layer [37]. Stochastic arithmetic for applying 'small' fractional changes as weights are altered is an interesting contemporary methodology for reducing the size of stored variables and, consequently, the energy costs in their transfer. A similar technique can be applied to SNNs, but is equally costly.

Various approaches to learning have been proposed. The project will focus on back-propagation through time and surrogate gradient methods, which are the most popular and successful training methods for time-based signals in the ANN and SNN domains, respectively. These approaches require a record of each neuron's recent history, which is combined with output error information to compute necessary changes to the synaptic weight in each connection to reduce the overall error.

5.1 Data representation

Data representation is an area of particular interest. Both ANNs and SNNs are 'programmed' by tuning the strength of the synaptic weights. Most NN models use far fewer bits to represent these weights than a conventional processor 32- or 64-bit word; indeed weights can be reduced to single bits, although intermediate formats such as Google's (Brain Floating Point) 'bfloat16' [4] have been proposed as a good trade-off between storage efficiency and useful information content. They still demand support for floating point calculations. The matrix of interconnection weights is typically the largest data block, where storing and fetching (as well as modifying and rewriting in learning systems) these values imposes a considerable energy cost.

Even shorter, 'minifloats' – such as 8-bit values – are used in some computations and companded values (akin to A-law/μ-law used in audio processing) may be appropriate and computed with directly or on-the-fly translated into a form suitable for a more conventional processor by 'smart' DMA.

An issue with shortened value representations, particularly when a network is learning, is the dynamic range available; a 'small' alteration to a value may be rounded away and have no effect. Rather than lengthen the storage it is proposed to facilitate stochastic (probabilistic) arithmetic which will approximate the desired effects whilst retaining short-form storage. This requires the generation of many (adequately) 'random' numbers which can be done at reasonable cost in dedicated hardware whereas is expensive in software.

Providing customised data representations tuned to the particular application to be more energy efficient than more general-purpose neuromorphic processors that are not optimised for any specific task. CONVOLVE provides the framework (in WP6) to search the space of possible data representations to select those that provide the optimum trade-off between accuracy and energy efficiency. To do this, there needs to be a range of data representations from which to choose, along with their hardware implementations.

5.2 Custom Operators

The aim of this work is to translate algorithmic advances on sparsely connected and sparsely active neural networks using low-bit-width or binary activations and dynamic execution as in DynNNs. To do this we need accelerators with comprehensive support for their underlying operations. Specifically, we need dedicated support for efficient sparse matrix-vector multiplications in which both the matrix and the vector may be sparse. Similarly, we need support for conditional evaluation as needed in the case of DyNN, whereby threads commonly used on GPUs may be too limited. Finally, we require accelerators that can deal seamlessly with mixed precision arithmetic to reap the benefits of binary activation functions or spikes. For example, such support would allow simulation of an analogue membrane potential of a spiking neuron while integrating binary inputs through sparse presynaptic spikes.

These accelerators should be tailored to one or more of the target applications and should be implemented either as tightly coupled instructions within the RISC-V processor or as standalone execution engines on the main bus.

Requested operators include:

1. Exponentiation
2. Complex thresholding functions (for ANNs)
3. Matrix and vector operations with non-standard bit-widths and mixed precision operations, stochastic arithmetic
4. Colour space conversion
5. Projective transform
6. 'Branch predictor'-like hardware in support of early exit in DyNNs

Since time and resource on the project is limited, it is appropriate to focus on those functions that are beneficial to the most use-cases and would result in the largest reduction in energy consumption when executing the target algorithms.

5.3 Data transport

Data movement is likely to represent the most significant cost, energetically speaking. Reducing operand sizes will help with this but will not provide all the desired savings. Locating processing close to the memory will also provide savings; unfortunately some applications require so many operands that more distant 'backing store' becomes essential.
The dominant feature in an ANN is supplying the Multiply-Accumulate (MAC) units, which are successively multiplying 'activation' vectors with appropriate weights for each 'neuron'. Storing the weights in a sparse matrix representation can reduce the number of MAC cycles in each vector and, importantly, the number of memory read operations. This might be performed with a 'smart' addressing unit for streaming only selected activations to the MAC. The sparse weight representation is similar to that currently used for SSN weights, where activations arrive individually, spread over time and the MAC is degenerate to a simpler accumulator.

In some problems, such as CNNs, a small set of weights is used repeatedly and these can be held conveniently in local storage. In applications requiring large matrices the storage becomes more remote. Here further packing of operands is important to reduce long bus switching. It may also be feasible to filter of sort the weights at source, supplying only a selected subset; this technique fits well with the DyNN strategy.

This sub-project should define, implement and integrate a flexible bus-based DMA engine to support these operations. The accelerator should be able to queue up a list of DMA transfers

freeing up the processor from these tasks. Data will be retrieved from one memory and, post processing, will be deposited into a second memory, local to where it will be used.

To support learning, the weights must be processed and rewritten. Thus the data unpacking operation should be reversible. Implementing the inverse operation (taking unpacked data and compressing it into a set of contiguous words ready for transfer back to main memory) should also be developed if learning is to be supported in hardware.

The pack operation that forms part of the support for learning would be the appropriate place to implement hardware in support of *stochastic computation*: specifically, the dithered rounding operation required to truncate a synaptic weight before it is written back to memory. This requires a source of random numbers to decide if the value should be rounded up and down during truncation. Such a source need not conform to very stringent constraints on the true randomness of the values generated, but some care must be taken as to its complexity to avoid systematic bias during learning.

5.4 Streaming co-processing for ANNs and SNNs

There are three basic types of neural networks that would cover the majority of the requirements for the applications required for CONVOLVE:

1. Convolutional Neural Network (CNN)
2. Generic Artificial/Deep Network (ANN/DNN)
3. Spiking Neural Network (SNN)

These differ either in the way that the weights are stored or the order in which the computation proceeds. For CNNs, the kernel weights can typically be stored local to where they are used and the fetching of the activations from the previous layer can be scheduled to maximise re-use (and hence minimise energy cost).

For generic ANNs, the weight matrix can be larger, sparser and less regular than the CNN case. It may be stored in a more distant (even external) memory. But the scheduling of the fetches for the activations can still be regularised to reduce the number of accesses to activation memory. The SNN could also be a convolutional network, but typically has a sparse, irregular weight matrix. Since an SNN is event-driven, the pattern of accesses to the weight memory is typically irregular and difficult to predict.

An SNN may also include the management of delays to simulate the non-zero transmission time of spikes. These are important in biologically plausible neural networks and may have significance in more abstract computation models but (outside audio processing) the use of transmission delays in spiking models is limited.

Despite these differences, there are sufficient areas of commonality that it should be possible to develop a component-based toolkit of parts allowing networks of all three types to be constructed and optimised, using the toolchain developed in WP6 and the compiler developed in WP5. These components should include configurable datapath widths to support a range of data types, allowing highly optimised processing for each application.

An accelerator that automates the implementation of these neural networks types would offer a significant saving in energy over a software implementation of the same functionality. Modules that implement specialised layer functions for ANNs such as Max Pooling, normalisation and K-

winner takes all would also be desirable since these implement costly sorting functions that are slow in software but could be much faster and more energy efficient as hardware implementations.

Figure 10 shows a possible, generic ANN accelerator, which also has some commonality with SNN requirements.

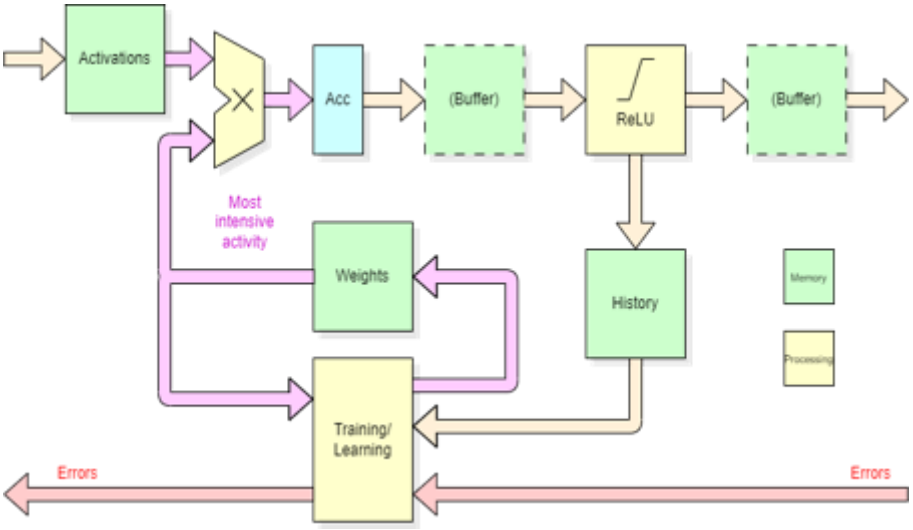

Figure 10 Accelerator pipeline implementing one layer of an ANN

The building blocks of a spiking neural network have commonality with those of the ANN, but the control flow will be different and the size of the buffers will be different as a consequence (for example, for the ANN the output of an entire layer will be stored locally so that it can be referenced multiple times. This is not required for the SNN, which is event-driven and discards each spike after it has been processed). The equivalent diagram for the SNN is shown below (Figure 11). In both of these pipelines learning may also be implemented in software running on the RISC-V processor or may be accelerated using custom hardware.

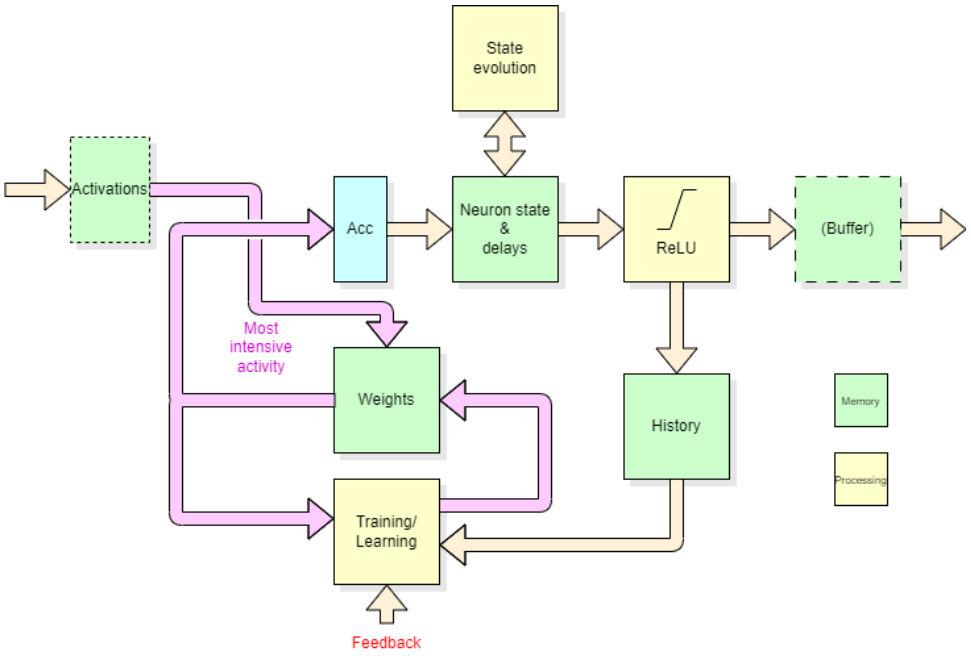

Figure 11 Accelerator pipeline implementing one layer of a spiking neural network (SNN)

6 Placement of accelerators

There are multiple architectural opportunities for the placement of hardware accelerators to support NNs of various types.

Unit	Operations	Example(s)
1	Operating on register variables within the processor	Exponentiation etc.
2	ISA-driven SIMD/vector accelerator; processing of 'novel' data types	Akin to Intel SSE/ARM Neon
3	Bus-based units providing scalar-type functions	Pseudorandom numbers
4	'In memory' vector processors, potentially with private workspace	Semi-autonomous processing such as periodic updates to data structures
5	I/O movement and transformation	Packing/unpacking structures using DMA; expanding I/O

TABLE 6: POTENTIAL ARCHITECTURAL PLACEMENT OF ACCELERATORS

In this classification #1 and #2 would be 'L0' accelerators, the former exploiting the existing RISC-V datapath and the latter augmenting this with its own (probably wider) registers and ALUs; #3 covers simple 'L1' accelerators which may not need bus mastery whilst #5 are more complex 'L1' devices which provide data transfers. The 'in memory' devices (#4) are the most complex and are 'L2' accelerators; they may benefit from multiple I/O buses to exploit private SRAM (e.g. for CNN support), effectively multiport (interleaved block) SRAM or streaming external data to provide the appropriate operand bandwidth.

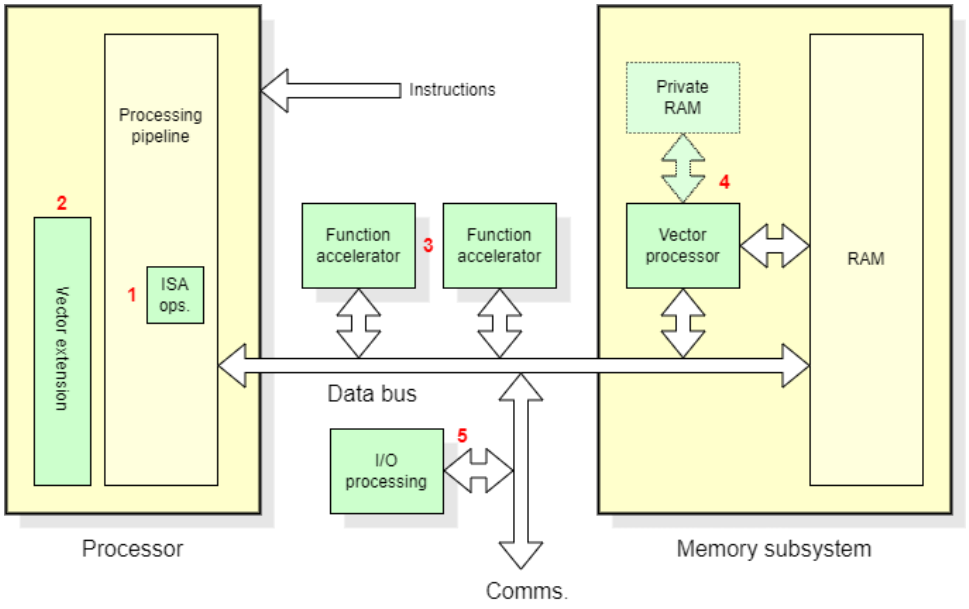

FIGURE 12: ARCHITECTURAL OPPORTUNITIES FOR CUSTOM ACCELERATORS

7 Proposed Hardware Development

There are numerous opportunities for hardware acceleration for supporting NNs of different types, covering the use cases selected by CONVOLVE as well as those identified by partners from previous development of neuromorphic hardware. These vary in architectural location and in complexity. More suggestions have been raised than it will be feasible to complete in WP2 so some prioritisation is suggested here based on available time and resources, the potential complexity of implementation, and the generality of implementation.

Block Name	Purpose	Scale	Usage
Adders, MACs	Component in other hardware	Small	Can be used in all other hardware or as a RISC-V extension instruction
Exponentiation Unit	Component in other hardware	Small/medium	RISC-V extension instruction
Threshold function: RELU	ANN threshold function	Small	RISC-V extension instruction or component in other hardware
Threshold function: look-up table/interpolation	Flexible threshold function for ANNs	Small/medium	RISC-V extension instruction or used in other hardware block
Stochastic rounding unit	Supports lower-bit precision storage for learning applications	Medium	Learning ANNs or SNNs with large data storage, to benefit from lower storage precision
Vector ALU supporting MAC and other packed ops	Compute engine for ANNs/SNNs	Medium	Flexible scalable component as the basis for larger pipeline
DMA with sparse matrix unpack and pack	Fetch engine for weights	Medium	Building block of pipeline to operate on sparse matrices for ANNs/SNNs
K-winner-take-all unit	Used in threshold functions. Select top K from a list of values	Medium	Used in SNNs for fixed-weight coding, and rank-order coding.
MaxPooling	Dimensionality reduction in deep ANNs	Small/medium	Standard layer for ANNs
Normalisation	Rescale list of values	Small/medium	Standard layer for ANNs
ANN accelerator pipeline	Processes layers of an ANN	Large	Co-processor on the RISC-V bus
SNN accelerator pipeline	Processes layers of an SNN	Large	Co-processor on the RISC-V bus

Table 7 List of proposed hardware components and sub-projects for development in WP2 in support of Neural Networks

7.1 Summary

The four use-cases selection by CONVOLVE were described in detail in document D1.1 and summarised in this document, section 3, for convenience. It is reasonable to assume that technology and algorithms that are developed in CONVOLVE that will help to achieve the two-orders of magnitude energy efficiency will be more widely applicable.

All four use-cases have been designed with ANNs in mind and it is reasonable to focus our efforts in developing neural network hardware that is targeted on ANNs (with and without support for Dynamic Neural Networks) to help to deliver the point demos in month 18. However, the essential components for such an ANN pipeline would also be appropriate for a spiking network and so with some forethought we can maximise the flexibility and applicability of the hardware developed in the WP4/WP2 crossover to cover as wide of range of future applications as possible. Hence, we identify a range of different memory management models, threshold functions and data formats and must ensure that components based on these will work together in whatever combination is required for the task in hand.

The hardware proposed here is challenging in places but the designs should be realisable in the timeframe of CONVOLVE, with the first versions of the major components delivered in time for the point demos in month eighteen and the complete and optimised set, incorporating feedback from their deployment in implementing the use-cases, by month 33.

References

[1] Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., Imam, N., Nakamura, Y., Datta, P., Nam, G.J., Taba, B., Beakes, M., Brezzo, B., Kuang, J.B., Manohar, R., Risk, W.P., Jackson, B., Modha, D.S., 2015. 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 34, 1537–1557. https://doi.org/10.1109/TCAD.2015.2474396

[2] Bellec, G., Scherr, F., Subramoney, A. et al., A solution to the learning dilemma for recurrent networks of spiking neurons. Nat Commun 11, 3625 (2020). https://doi.org/10.1038/s41467-020-17236-y

[3] Benjamin, B.V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A.R., Bussat, J.M., Alvarez-Icaza, R., Arthur, J.V., Merolla, P.A., Boahen, K., 2014. Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations. Proceedings of the IEEE 102, 699–716. https://doi.org/10.1109/JPROC.2014.2313565

[4] Bouzidi, H., Odema, M., Ouarnoughi, H., Faruque, M.A.A., Niar, S., 2022. HADAS: Hardware-aware dynamic neural architecture search for edge performance scaling'. https://doi.org/10.48550/arXiv.2212.03354.

[5] Burgess, N., Milanovic, J., Stephens, N., Monachopoulos, K. and Mansell,D., "Bfloat16 Processing for Neural Networks," *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, Kyoto, Japan, 2019, pp. 88-91, doi: 10.1109/ARITH.2019.00022.

[6] Davies, M., Srinivasa, N., Lin, T.H., Chinya, G., Cao, Y., Choday, S.H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C.K., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., Weng, Y.H., Wild, A., Yang, Y., Wang, H., 2018. Loihi: A Neuromorphic

Manycore Processor with On-Chip Learning. IEEE Micro 38, 82–99. https://doi.org/10.1109/MM.2018.112130359

[7] Diehl, Peter U., and Matthew Cook. "Efficient implementation of STDP rules on SpiNNaker neuromorphic hardware." *2014 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2014.

[8]Furber, S.B., Galluppi, F., Temple, S., Plana, L.A., 2014. The SpiNNaker Project. Proceedings of the IEEE 102, 652–665. https://doi.org/10.1109/JPROC.2014.2304638

[9] Mayr, C., Hoeppner, S., Furber, S., 2019. SpiNNaker 2: A 10 Million Core Processor System for Brain Simulation and Machine Learning. https://doi.org/10.48550/arXiv.1911.02385

[10] Orchard, G., Frady, E.P., Rubin, D.B.D., Sanborn, S., Shrestha, S.B., Sommer, F.T., Davies, M., 2021. Efficient Neuromorphic Signal Processing with Loihi 2, in: 2021 IEEE Workshop on Signal Processing Systems (SiPS). Presented at the 2021 IEEE Workshop on Signal Processing Systems (SiPS), pp. 254–259. https://doi.org/10.1109/SiPS52927.2021.00053

[11] Han, Y., Huang, G., Song, S., Yang, L., Wang, H. and Wang, Y., 2021. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(11), pp.7436-7456.

[12] Han, Y., Huang, G., Song, S., Yang, L., Wang, H., Wang, Y., 2022. Dynamic Neural Networks: A Survey'. IEEE Trans. Pattern Anal. Mach. Intell 44, 7436-7456,. https://doi.org/10.1109/TPAMI.2021.3117837.

[13] Höppner, S., Yan, Y., Dixius, A., Scholze, S., Partzsch, J., Stolba, M., ... & Mayr, C., 2021. The SpiNNaker 2 processing element architecture for hybrid digital neuromorphic computing. *arXiv preprint arXiv:2103.08392*.

[14] Hua, W., Zhou, Y., Sa, C., Zhang, Z., Suh, G.E., 2019. Boosting the performance of CNN accelerators with dynamic fine-grained channel gating', in: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture. ACM, Columbus OH USA, pp. 139–150. https://doi.org/10.1145/3352460.3358283.

[15]Huang, Q., 2021. CoDeNet: Efficient deployment of input-adaptive object detection on embedded FPGAs', in: The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Virtual Event. ACM, Feb, USA, pp. 206–216. https://doi.org/10.1145/3431920.3439295.

[16] Huang, W. *et al.*, "FPGA-Based High-Throughput CNN Hardware Accelerator With High Computing Resource Utilization Ratio," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 8, pp. 4069-4083, Aug. 2022, doi: 10.1109/TNNLS.2021.3055814.

[17]Kleyko, D., Rachkovskij, D.A., Osipov, E., Rahimi, A., 2023. A Survey on Hyperdimensional Computing aka Vector Symbolic Architectures, Part I: Models and Data Transformations. ACM Comput. Surv. 55, 1–40. https://doi.org/10.1145/3538531

[18] Maass, W., 1997. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, *10*(9), pp.1659-1671.

[19] Mead, C., 1990. Neuromorphic electronic systems. Proceedings of the IEEE 78, 1629–1636. https://doi.org/10.1109/5.58356

[20] Meier, K., "A mixed-signal universal neuromorphic computing system," 2015 IEEE International Electron Devices Meeting (IEDM), Washington, DC, USA, 2015, pp. 4.6.1-4.6.4, doi: 10.1109/IEDM.2015.7409627.

[21][Moreira, O., et al. "*NeuronFlow*: A hybrid neuromorphic–dataflow processor architecture for AI workloads." *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2020.

[22] Paul, D., Singh, J., Mathew, J., 2019. Hardware-software co-design approach for deep learning inference', in: 2019 7th International Conference on Smart Computing & Communications (ICSCC), Jun. pp. 1–5. https://doi.org/10.1109/ICSCC.2019.8843626.

[23] Perez-Nieves, Nicolas, and Dan Goodman. 2021. 'Sparse Spiking Gradient Descent'. In Advances in Neural Information Processing Systems, 34:11795–808. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2021/hash/61f2585b0ebcf1f532c4d1ec9a7d51aa-Abstract.html.

[24] Ren, M., Pokrovsky, A., Yang, B., Urtasun, R., 2018. *SBNet*: Sparse blocks network for fast inference', in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 8711–8720.

[25] Rosenblatt, F., "Perceptron Simulation Experiments," in Proceedings of the IRE, vol. 48, no. 3, pp. 301-309, March 1960, doi: 10.1109/JRPROC.1960.287598.

[26] Schlegel, K., Neubert, P., Protzel, P., 2022. A comparison of vector symbolic architectures. Artif Intell Rev 55, 4523–4555. https://doi.org/10.1007/s10462-021-10110-3

[27] Schmitt, S., Klähn, J., Bellec, G., Grübl, A., Güttler, M., Hartel, A., Hartmann, S., Husmann, D., Husmann, K., Jeltsch, S., Karasenko, V., Kleider, M., Koke, C., Kononov, A., Mauch, C., Müller, E., Müller, P., Partzsch, J., Petrovici, M.A., Schiefer, S., Scholze, S., Thanasoulis, V., Vogginger, B., Legenstein, R., Maass, W., Mayr, C., Schüffny, R., Schemmel, J., Meier, K., 2017. Neuromorphic hardware in the loop: Training a deep spiking network on the BrainScaleS wafer-scale system, in: 2017 International Joint Conference on Neural Networks (IJCNN). Presented at the 2017 International Joint Conference on Neural Networks (IJCNN), pp. 2227–2234. https://doi.org/10.1109/IJCNN.2017.7966125

[28] Verelst, T., Tuytelaars, T., 2020. Dynamic convolutions: Exploiting spatial sparsity for faster inference', in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2320–2329.

[29] Wang, Y., Huang, R., Song, S., Huang, Z., Huang, G., 2022. Not all images are worth 16x16 words: Dynamic transformers for efficient image recognition', in: Advances in Neural Information Processing Systems.

[30] Wiegand, Thomas, et al. "Overview of the H. 264/AVC video coding standard." *IEEE Transactions on circuits and systems for video technology* 13.7 (2003): 560-576.

[31] Xie, Z., Zhang, Z., Zhu, X., Huang, G., Lin, S., 2020. Spatially adaptive inference with stochastic feature sampling and interpolation', in: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.-M. (Eds.), Computer Vision – ECCV 2020. Springer International Publishing, Cham, pp. 531–548. https://doi.org/10.1007/978-3-030-58452-8_31.

[32] Yang, Y., 2019. *Synetgy*: Algorithm-hardware co-design for ConvNet accelerators on embedded FPGAs', in: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Feb. pp. 23–32. https://doi.org/10.1145/3289602.3293902.

[33] Yizhao G., Baoheng Z., Xiaojuan Q., and Hayden K.-H., "DPACS: Hardware Accelerated Dynamic Neural Network Pruning through Algorithm-Architecture Co-design", In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS 2023). doi: 10.1145/3575693.3575728

[34] Zenke, F., and Neftci, E., O., 2021. 'Brain-Inspired Learning on Neuromorphic Substrates'. Proceedings of the IEEE 109 (5): 935–50. https://doi.org/10.1109/JPROC.2020.3045625.]

[35] Zenke, F., Ganguli, S., SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks. *Neural Comput* 2018; 30 (6): 1514–1541. doi: https://doi.org/10.1162/neco_a_01086

[36] Zhang, Z., "Improved adam optimizer for deep neural networks." *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)*. Ieee, 2018.

[37] LeCun, Y., et al. "A theoretical framework for back-propagation." *Proceedings of the 1988 connectionist models summer school*. Vol. 1. 1988.