Trainable Region of Interest Prediction: Hard Attention Framework for Hardware-Efficient Event-Based Computer Vision Neural Networks on Neuromorphic Processors

Cina Arjmand
cinaarjmand@gmail.com

Department of Electrical and Information Technology
Lund University

Supervisor: Mattias Borg, Liang Liu

Examiner: Erik Lind

December 15, 2023

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Neuromorphic processors are a promising new type of hardware for optimizing neural network computation using biologically-inspired principles. They can effectively leverage information sparsity such as in images from event-based cameras, and are well-adapted to processing event-based data in an energy-efficient fashion. However, neuromorphic processors struggle to process high resolution event-based data due to the computational cost of high resolution processing. This work introduces the Trainable Region of Interest Prediction (TRIP) framework for attaining hardware-efficient processing of event-based vision on a neuromorphic processor. TRIP uses active region-of-interest (ROI) generation to perform hard attention by cropping selected regions of input images, automatically filtering out unnecessary information and learning to process only the most important information in an image. TRIP is implemented on several neural networks tested on various event-based datasets. It leverages extensive hardware-optimization to maximize efficiency with respects to hardware-related metrics such as power, memory utilization, latency, number of network parameters, and area. The algorithm is implemented and benchmarked on the SENECA neuromorphic processor. The algorithms employing the TRIP framework exhibit intelligent ROI selection behavior and the capability to dynamically adjust ROI size and position to fit various targets, while obtaining or in some cases improving over state-of-the-art accuracy. Utilizing lower resolution input reduces the computation requirements of TRIP by 46× compared to state-of-the-art solutions. The embedded hardware implementation of TRIP more than doubles the speed and energy efficiency of classification on the DVS Gesture recognition dataset compared to a baseline network, and generally outperforms other state-of-the-art neuromorphic processors benchmarked using DVS Gesture.

Popular Science Summary

Artificial intelligence (AI) is being pervasively adopted across all facets of life, but one thing holding back the expansion of AI capabilities is its massive energy requirements. Servers used to deliver AI services consume more electricity than entire countries[1], and the cost is felt not only in the wallets of companies but on the planet as well. While the things that AI can do are extremely impressive, the current cost of powering AI is simply unsustainable.

Humans are looking for ways to make more powerful AI systems fit on smaller devices that consume less energy. Not only will this be cheaper for both the planet and those paying for cloud servers, it will enable AI systems like robots and self-driving cars to be faster, more sophisticated, and more effective. When an AI-powered system like a self-driving car can perform its advanced object detection and decision-making efficiently within the device itself, it does not have to send huge amounts of data back and forth to a cloud server. Instead, it can operate locally and be self-contained within itself. This is faster, potentially more energy efficient, and eliminates the risk of sensitive data being leaked. Furthermore, the less space and energy that an advanced AI takes up on a device, the more advanced AI we can fit on a single device. Super energy-efficient AI can enable us to cram a multitude of sophisticated functions into a single wearable or implantable device. The brain of a robot will be able to fit even more neurons in it and still run on the same battery. More efficient AI will enable a future that looks even more like science fiction.

The challenge of making AI more efficient require re-thinking both the hardware that we build to run it and the algorithms we design for the software. In this thesis, an algorithm is implemented which mimics the way that human eyes analyze a scene. Instead of taking in an entire scene at once (kind of like an insect eye) as most computer vision AI does, the algorithm in this thesis figures out where the most interesting place to look in a picture is, and then focuses only on that part. This is called a hard attention algorithm, and doing this turns out to be much more effective than doing it the insect way. Additionally, this algorithm is implemented on a special type of hardware known as a neuromorphic processor, which is inspired by how neurons in the human brain communicate with one another. The reason why neuromorphic processors try to mimic the human brain is the same reason why the algorithm tries to mimic human vision: the human brain

[1] https://doi.org/10.1016/j.joule.2023.09.004

is simply incredibly energy efficient! Sure, AI might be able to beat you at a board game. However, your brain only consumes as much power as a single lightbulb, while the AI needs something like the electricity consumption of an entire household. For that reason, many researchers believe that there is something to learn about how to make AI more energy-efficient by studying our own efficient and intelligent brains. Naturally, computers and AI are very different from humans and biological neurons, so human biology is not a blueprint but rather a source of potential inspiration and ideas.

The hard attention algorithm used in this work is designed for maximizing the efficiency of the algorithm on hardware. This means making it is as fast as possible while requiring as little power, memory, and area as possible. The algorithm itself drastically improves effectiveness over state-of-the-art solutions for computer vision using a special kind of camera known as an event-based camera (the design of which is also inspired by human vision in the interest of greater efficiency). It utilizes a whole set of hardware optimization techniques, and is implemented on a neuromorphic processor to produce an incredibly efficient computer vision algorithm for detecting gestures performed by a human. When a person waves their arms in front of the camera, the system automatically filters out information from the most interesting region (usually the arm) to minimize the amount of information that it needs to process. It automatically adjusts its own focus to filter out irrelevant objects or to focus on bigger or smaller objects in the camera's view. It detects the gesture with high accuracy, high speed, and with minimal memory and energy usage.

By overcoming the limitations typically faced by embedded AI, the solution presented in this work offers potential solutions to many of the problems currently faced in research. One of these is the high computational cost of processing high resolution images, which makes it impossible for neuromorphic systems to process data from event-based cameras with high resolution. High resolution data is important to obtain high accuracy, but because of the current hardware inefficiencies, state-of-the-art solutions are forced to resort to low-resolution processing instead. By focusing processing towards small, low-resolution areas, the hard attention framework introduced in this thesis could be very useful for high resolution processing of event-based data on neuromorphic systems.

Table of Contents

1 Introduction **1**
- 1.1 Motivation 1
- 1.2 Research goal 2

2 Theory **5**
- 2.1 SENECA Neuromorphic Processor 5
- 2.2 Event-based Vision 6
- 2.3 Hard Attention 7

3 Method **11**
- 3.1 Trainable Region of Interest Prediction 11
- 3.2 Additional Hardware-aware Algorithm Optimizations 15
- 3.3 Datasets Used for Algorithm Benchmarking 16
- 3.4 Hardware Implementation and Benchmarking 18

4 Results and Discussion **21**
- 4.1 DVS Gesture Dataset 21
- 4.2 Marshalling Signals Dataset 22
- 4.3 Synthetic N-MNIST-based Dataset 23
- 4.4 Hardware Implementation: Benchmarking Results 23

5 Outlook and Conclusion **27**

References **29**

List of Figures

2.1 Overview of SENECA cores. Right: a collection of multiple SENECA cores. Left: expanded image of one seneca core, containing RISC-V controller, eight NPEs, and periphiral circuitry including memory and network on chip (NoC). 7

2.2 The same scene recorded with (a) an RGB camera and (b) an event-based camera. The event-based camera only produces an output when light intensity changes, so the moving person is captured while the stationary background is not. 8

3.1 Overview of TRIP framework showing the data processing steps from input to output. Downsampled, low-resolution events are provided as input to the ROI prediction network to produce ROI parameters. ROI parameters are used by the ROI generation block to generate a low-resolution cropped ROI for processing by the classificaiton network. H_t is the output values of the ReLU RNN, and P_t is the output values of processed events from timebin t. 12

4.1 Visualization of ROI receptive fields from three different gesture samples of the DVS Gesture dataset. The receptive field is visualized as a yellow square superimposed above the event image. 22

4.2 ROI receptive field visualization for sample gestures of the Marshalling Signals dataset performed from different camera distances. 23

4.3 ROI visualization on three different N-MNIST samples (left to right: digit 7, digit 3, and digit 0). 24

4.4 Overview of per-core utilization of TRIP implementation, including latency, memory usage, and energy of each core. 25

List of Tables

4.1 Performance comparison of TRIP against other state-of-the-art solutions on DVS Gesture dataset. 22
4.2 Performance comparison of TRIP against state-of-the-art networks on the Marshalling Signals dataset. 23
4.3 Performance comparison of TRIP against baseline networks on the synthetic N-MNIST dataset. 24
4.4 Comparison of TRIP and SENECA baseline with state-of-the-art neuromorphic implementations on DVS Gesture dataset. 25

Chapter 1

Introduction

Artifical neural networks trained by deep learning have enabled artificial intelligence (AI) systems to reach the point of near-human, human, or superhuman performance across all manner of tasks. They are deployed in autonomous driving systems, in language models that analyze and generate text, in wearable smart devices, in neural implants, in aerial and underwater robots, in warehouses to move boxes, in our homes to vacuum our floors, and in countless other facets of life. As the rate of AI adoption increases, and AI systems powered by neural networks grow more ubiquitous and entrenched in society, certain limitations of these systems also grow ever more apparent.

1.1 Motivation

The impressive capabilities of AI systems require staggering amounts of computations to be performed. These computations translate to large amounts of power consumed by the hardware on which neural networks run, usually in data centers accessed via cloud services. Cloud computing allows advanced AI to be used within a system such as a self-driving car while offloading the computations to a remote server. For the car to use its AI, requests are sent back and forth between the car and a neural network located on a processor somewhere far away from the actual car. This way, car designers need not consider the limitations of the car itself when selecting hardware to power the neural network. If not for cloud servers, a robot vacuum cleaner would have to carry specialized hardware which might drain its battery faster, complicate product design, impact manufacturing considerations, and likely exacerbate the development of an efficient form factor. Therefore, offloading computations to the cloud is convenient, since the current bulky, energy-expensive state of neural network computation makes nimble AI deployment difficult to achieve locally, on the devices themselves.

However, there are several reasons why researchers and companies alike are seeking ways to move AI off the cloud, and embed it within entirely self-contained devices. This is known as edge computing, where neural networks are computed locally "on the edge" rather than "on the cloud". First, the transfer of large amounts of data back and forth from the cloud is highly costly in terms of energy. Second, sending data has a huge impact on the speed of a system. In fact, for computer vision applications, it is often the main latency bottleneck affecting the speed at which a neural network can perform inference. Thirdly, reliance on data transfer and storage opens up privacy concerns. A significant amount of companies at some point fall victim to data breaches or attacks, which exposes potentially sensitive data of its users. Another important consideration particularly relevant for companies delivering AI-powered solutions is the cost of either running or renting cloud services. Data centers are very expensive to run, and the cost of renting cloud services can easily dominate operating costs if not carefully managed.

If AI systems can be designed to be more hardware-efficient than they are today, they could be run locally on devices without needing to send any data back and forth to the cloud. This would avoid the risk of sensitive data exposure and increase the speed at which AI can operate. It would remove the cost/operating overhead of cloud services. It has the potential to make systems more energy efficient than what they are today, which opens up greater possibilities for how complicated a single system can get.

Edge AI applications today are limited by how efficient algorithms and hardware are when running together. The challenge of making AI algorithms run efficiently on hardware necessitates making improvements and design considerations in both software design and the hardware design. Importantly, software must be designed to run efficiently on a given hardware in order to best leverage that hardware's capabilities. Equally importantly, design of hardware must be informed by how the design can most efficiently accelerate the algorithms that need to be run on it. Therefore, hardware design and software design must inform and be informed by one another. They must be developed and optimized together in order to achieve hardware-efficient algorithms and thereby hardware-efficient AI systems.

1.2 Research goal

This thesis work explores methods of making deep learning neural networks more hardware-efficient, *i.e.* faster and less area-, memory- and energy-consuming to run on a processor. In particular, this work deals with a combination of three different paradigms being used in edge AI research and industry: 1) neuromorphic computing, 2) event-based vision, and 3) hard attention algorithms.

Neuromorphic processors are a specialized form of hardware designed to optimize neural network computations by performing in-memory computation. This work uses the SENECA digital neuromorphic processor developed by researchers at IMEC.

Event-based vision is visual processing performed using event-based cameras: an alternative technology to RGB cameras. Event-based cameras are developed to be more energy efficient than RGB by using less information to represent a scene or object.

Hard attention algorithms are used to train computer vision neural networks to automatically select a small region(s) of an image that are important for understanding a scene; the important region is then cropped in order to limit the amount of information that the network needs to process. It allows the computational cost of image processing to be reduced due to the smaller dimension of data being processed. Neuromorphic processors, in-memory computing, SENECA, event-based vision, and hard attention algorithms are explained in further detail in the next chapter.

Hard attention algorithms are proven to be able to reduce the computational cost of image processing. In particular, they enable high resolution processing which is beneficial for high classification accuracy without a prohibitive increase in processing requirements. However, little to no work exists that seeks to combine hard attention methods with event-based vision and neuromorphic processing in order to leverage the combined hardware efficiency of all three paradigms together. For limited neuromorphic systems, the hardware cost of processing high-resolution input is oftentimes prohibitively high [29] as evidenced by the lack of state-of-the-art processors that can perform high resolution processing efficiently. As such, neuromorphic processors cannot leverage the high resolution event-based data that is currently available from state-of-the-art event camera technology.

The goal of this thesis work can be summarized in two parts: 1) to develop a novel design of a hard attention framework with the goal of maximizing the hardware efficiency of a neural network performing classification on event-based images. 2) to realize a neuromorphic hardware implementation of said network with improved efficiency over state-of-the-art through hardware-software co-optimization.

The framework developed in this thesis in order to meet this goal is called TRIP: Trainable Region of Interest Prediction, and is described in further detail in Chapter 3. Chapter 3 also contains details of the methods used to develop and benchmark the framework on different datasets and in the hardware implementation. Chapter 4 details the results of experiments and benchmarking, and Chapter 5 provides an outlook on future directions for expanding the work outlined within the previous chapters.

Chapter 2

Theory

2.1 SENECA Neuromorphic Processor

SENECA is a Scalable Energy-efficient NEuromorphic Computer Architecture [33]. A neuromorphic architecture differs from the type of computer architectures used in most computing devices. The computers and processors that are currently used for computing neural networks are based on what is called von-Neumann architecture. For example, a graphics processing unit (GPU) which is the most commonly used hardware for AI, is an example of a von-Neumann architecture. In a von-Neumann architecture, memory and computation are separated. To perform multiplication using neural network weights, for instance, the correct weight has to be loaded from the memory into the computation. Once the result of the multiplication is computed, the output value must be stored back in the memory as the updated neuron state. The movement of data from memory to computing and vice versa is the main bottleneck in von-Neumann processing which limits processing speed and energy efficiency. Neuromorphic architectures typically mitigate this issue by co-localizing computing and memory through so-called in-memory computing: by performing computation directly on the memory storing a neuron state, the state is updated directly without need for data movement. In practice, true co-localization of memory and computation is difficult to realize in digital architectures since memory and computation is always defined by separate circuits. Analog solutions based on memristor crossbar arrays enable "true" in-memory computations, but digital neuromorphic solutions technically implement *near-memory* computation by minimizing the transfer overhead between memory and computation as much as possible through design and layout considerations.

In von-Neumann architectures such as GPUs, multiple GPU cores perform processing in parallel using the same shared data memory between cores, and there is no direct data transfer from core to core. This limits scalability; adding more cores is prevented by the data transfer bottleneck between shared memory and cores imposed by the limited memory bandwidth. Where the SENECA processor architecture differs is that each core contains its own data memory, and data is transfered between cores by implementing sequential processing blocks across multiple cores. This allows arbitrary scaling not limited by memory bandwidth [33]. In this way, SENECA circumvents the memory bottleneck of von Neumann systems by implementing a form of near-memory computing on the architectural level.

2.1.1 Event-based Processing

SENECA is a flexible and reconfigurable processor architecture which can implement various algorithms for neural network computing. In this work, neural networks on SENECA are implemented using integrate and fire neurons. In integrate and fire, the process of updating the neurons of a neural network is divided into two steps: 1) integrate and 2) fire. In integrate, incoming input to a neuron is used to update the neuron's state/memory potential. The fire step only occurs once the memory potential of the neuron exceeds a pre-defined threshold value, upon which the neuron fires one output signal. The output signal is called an event or a spike. Because a neuron is only updated when it receives an event(s), and neurons only produce events when it is time for them to fire, the integrate and fire model is an example of an event-based processing system. The advantage of using an event-based system is that only non-zero events are propagated through the neural layers. In non-event-based systems, all neuron states must be computed at each timestep even for neurons with no activity. By only computing non-zero events, an event-based system can perform less computations when updating neuron states. The number of non-zero events processed by a system is measured using the metric effective multiply-and-accumulate operations (effective MACs). A higher number of effective MACs directly translates to a greater hardware utilization as a result of more memory and energy being consumed for processing [32]. The ratio of effective MACs to overall size of a network is a measure of the activation density of a network. A network with a low number of effective MACs in relation to its size is said to have low activation density or high activation sparsity. The higher the sparsity in an event-based system, the less events have to be processed and the less computations have to be performed. The event-based system in this work encodes events using an adress-event representation (AER), in which each event is made up of a digital signal containing the magnitude of the event and the address of the source neuron that produced the event.

2.1.2 SENECA Architecture

SENECA consists of multiple cores. The architecture is scalable in the sense that it can be expanded to an arbitrary number of cores. Each core contains eight SIMD Neuron Processing Elements (NPEs). The NPEs are configured by a RISC-V controller which also manages input/output of events to and from the core, and address decoding/encoding. The RISC-V programs each NPE with the same microkernel, and the NPEs execute the same microkernel on 8 adjacent memory addresses at once.

2.2 Event-based Vision

In an event-based camera, each pixel independently and asynchronously produces an output proportional to the change of brightness at that pixel [10]. Because the

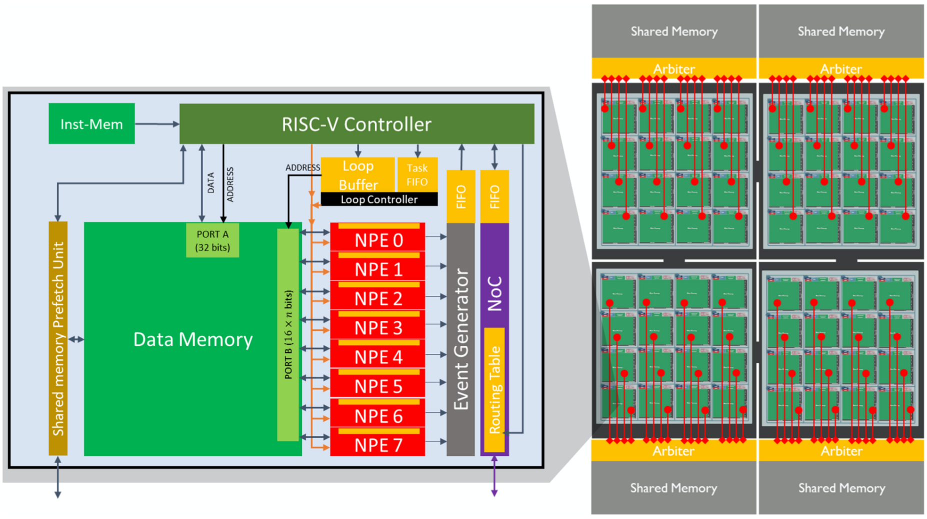


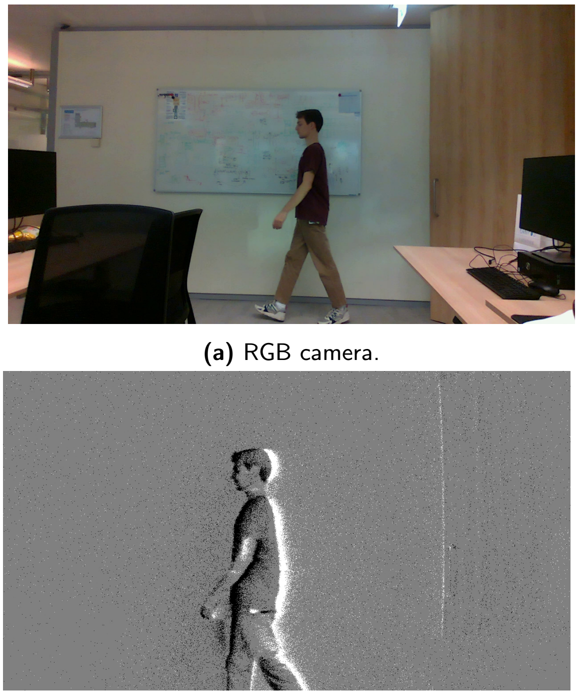
Figure 2.1: Overview of SENECA cores. Right: a collection of multiple SENECA cores. Left: expanded image of one seneca core, containing RISC-V controller, eight NPEs, and periphiral circuitry including memory and network on chip (NoC).

event-based camera only responds to changes in brightness, stationary or unchanging objects in the camera's field of view do not produce any output, see Figure 2.2. As a result, an event-based camera produces less information than an RGB camera because information from the background or from non-active regions is filtered out [6]. Assuming that the camera is only interested in objects that are not stationary/unchanging, the event-based camera is a more efficient way of processing important information about a scene since less pixels need to be processed. In particular, an event-based system is capable of leveraging the information sparsity in an event-based image to reduce the number of computations it has to perform during processing [36, 24].

An event-based image has two output channels, one for negative polarity and one for positive polarity, reflecting either an increase or decrease in relative brightness. Event-based images can be binary (magnitude of an event is always one) or graded. Graded events can represent arbitrary magnitudes of brightness as the event value which is sent in the AER format.

2.3 Hard Attention

Hard attention is the name given to top-down methods of selectively processing cropped regions of an input image [13]. They belong to a class of methods known as attention methods, in which different parts of an input are given different weighted values to represent their relative importance or saliency. High priority or attention is directed towards the most important or salient regions. In soft attention, the



(a) RGB camera.

(b) Event-based camera.

Figure 2.2: The same scene recorded with (a) an RGB camera and (b) an event-based camera. The event-based camera only produces an output when light intensity changes, so the moving person is captured while the stationary background is not.

entirety of the input dimension is processed in order to map the saliency of every data point. In hard attention, only a part of the input which is deemed important enough is ever selected for further processing while the rest is simply not processed at all.

The challenge in hard attention is finding a reliable strategy for determining a region of interest (ROI) without knowing a priori where the ROI will be. Early methods for saliency detection in computer vision focused on low-level image features such as differences in pixel brightness to make informed guesses about where important information might be [16]. The earliest top-down strategies attempted to frame the problem of saliency detection as a problem of selecting the right action of cropping, *i.e.* "deciding where to look" [20]. This idea was expanded upon through models that used reinforcement learning to define actions based on placing a limited bandwidth sensor on various locations across an input image. Through reinforcement learning, the agent placing the sensor is trained using a reward proportional to the likelihood of correctly classifying the image based on the input from the selected region. The recurrent attention model (RAM) utilizes recurrent neural networks to maintain an integrated history of previous predictions, in order to iteratively refine predictions over time based on data from previous attempts

[23]. Many other reinforcement learning models elaborate these concepts using increasingly sophisticated architectures and various tasks [3, 8, 27], including hard attention models for video footage [5] and for extremely high-resolution images containing small ROIs [19].

Hard attention algorithms have multiple beneficial properties. First and foremost, reducing the input size that needs to be processed by later network layers drastically reduces the number of computations that need be performed by the network, resulting in a more efficient algorithm. Using higher resolution input resolution improves accuracy on classification tasks, but computational cost increases quadratically when using convolutional networks [31, 27]. Furthermore, it has been shown that in cases where an input image has a low ROI-to-image ratio, classification accuracy can be degraded as a result [19]. Using hard attention can improve accuracy by filtering out unnecessary information that can interfere with classification [23, 12, 3]. Hard attention also offers a natural interpretability of neural networks' decision-making process, as by visualizing the ROI used by the network one can interpret what information was used by the network in its decision [8]. Additionally, hard attention methods have been shown to provide resistance against adverserial noise attacks [34, 11, 21].

Reinforcement learning (RL) is a well-suited method for training neural networks that need to make decisions by selecting actions, such as deciding where to crop an input image. RL methods are very effective at accurate ROI prediction, but they are difficult to train due to the complexity of RL training. These training complexities also translate to difficulties in embedded hardware implementation, due to the difficulty of applying hardware-aware optimization techniques to reinforcement learning models. The reason why most models rely on reinforcement learning is due to a lack of an effective differentiable mechanism for ROI selection. If the ROI selection is not differentiable, as is the case in all the existing RL hard attention models, then the entire model is not end-to-end differentiable and thus not trainable using the much simpler backpropagation algorithm commonly used in machine learning.

The Deep Recurrent Attention Writer (DRAW) utilizes a differentiable ROI generation mechanism using Gaussian kernels [12]. Scalar output values of a neuron layer are translated into the center location and distance between the mean values of a $N \times N$ grid of Gaussian kernels. By controlling the distance between the mean points of the Gaussian functions in the set of Gaussian kernels, the ROI generation can control the size of its "receptive field", *i.e.* what portion of the input image it takes as input to its downsampling, without changing the output resolution of the ROI. This downsampling operation which is capable of controlling the size of the region being downsampled is effectively a differentiable cropping operation. Neuromorphic DRAW (N-DRAW) uses this cropping technique with a recurrent neural network to predict ROIs on event-based image datasets with the goal of improving accuracy by selectively processing the most important information in an image [4]. This is the only other known work which applies hard attention to event-based images. The target of N-DRAW is not to minimize hardware usage, but rather to improve accuracy on a number of synthetic event-based datasets. This goal of TRIP is to go further, developing a more sophisticated model for maximum hardware efficiency and high performance on real and challenging datasets. Additionally, this work targets a real embedded hardware implementation with its own challenges.

Chapter 3

Method

Achieving the goal of this work necessitates development of software algorithm and hardware implementation in tandem: performing co-optimization of software with respects to hardware performance and vice versa. In order to develop an algorithm with the target of improving hardware efficiency, it is essential to understand and measure the implications of algorithm design choices in terms of their effect on hardware metrics such as latency, computational complexity, number of computations, and power consumption. This chapter introduces the Trainable Region of Interest Prediction framework and describes the multiple hardware-optimizations that were made to the algorithm prior to implementation, the methods employed to realize an efficient hardware implementation, and the benchmarking procedures.

3.1 Trainable Region of Interest Prediction

The solution developed in this thesis work has been named Trainable Region of Interest Prediction (TRIP). It is a hard attention framework designed to maximize hardware efficiency by implementing hard attention to reduce input resolution during image processing, and to do so with minimal latency and computation overhead. It is developed and optimized for event-based processing on neuromorphic hardware.

The processing pipeline is divided into three steps: 1) ROI prediction, 2) ROI generation, and 3) classification. An overview of the TRIP framework is visualized in Figure 3.1.

The ROI prediction is the initial step in the framework pipeline. A downsampled, low-resolution version of the full-resolution input image is provided as input to the ROI prediction network. The ROI prediction network consists of a neural network which outputs three neuron states. These neuron states are used to encode the center location and receptive field of a region of interest (ROI) on the full-resolution input image. A fixed resolution, cropped ROI is generated using the receptive field as input in the ROI generation step. The low-resolution ROI is then provided as input to the classification step, which consists of another neural network that performs classification on the ROI. The entire pipeline is differentiable and trained end-to-end using backpropagation and a standard cross-entropy loss function.

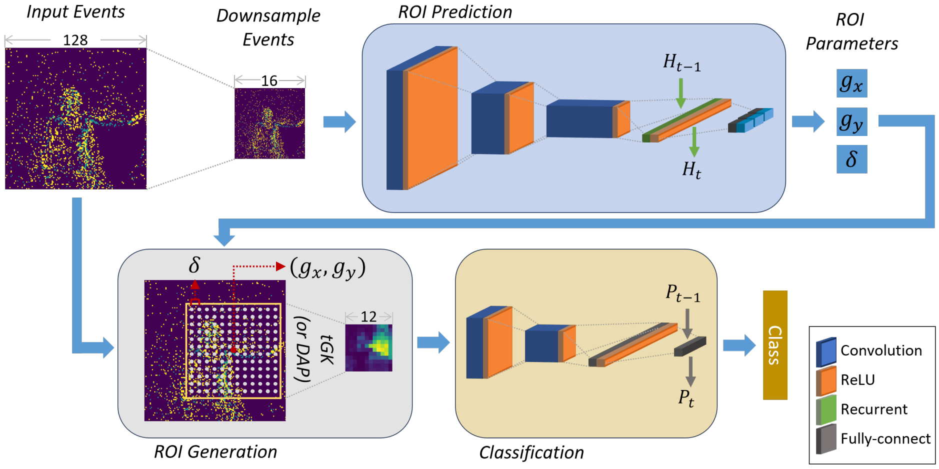


Figure 3.1: Overview of TRIP framework showing the data processing steps from input to output. Downsampled, low-resolution events are provided as input to the ROI prediction network to produce ROI parameters. ROI parameters are used by the ROI generation block to generate a low-resolution cropped ROI for processing by the classificaiton network. H_t is the output values of the ReLU RNN, and P_t is the output values of processed events from timebin t.

3.1.1 ROI Prediction Using Truncated Gaussian Kernels

The specific architecture of the ROI prediction may vary from task to task, but it sequentially processes the downsampled 16×16 input with multiple neural layers, the last of which contains three neurons. The scalar value output of these three neurons are referred to as the ROI parameters. They are used to define the location and width of the receptive field, *i.e.* what portion of the input image will be used as input for the ROI generation. Depending on the ROI generation method used, the parameters are decoded differently. In this work, two ROI generation method are mainly utilized: 1) truncated Gaussian kernels and 2) dynamic average pooling. During the initial training of the TRIP network, the truncated Gaussian kernels are used. Since the Gaussian kernels are differentiable with smooth gradients, they enable effective training by ensuring fast and reliable convergence on high classification accuracy. Once the model is trained to high accuracy, the ROI generation method can be substituted for a more hardware-friendly dynamic average pooling operation that is faster and cheaper to compute in the embedded hardware implementation.

When utilizing truncated Gaussian kernels, the three ROI parameters $\hat{g_x}, \hat{g_y}, \hat{\delta}$ are processed in the ROI generation:

$$g_x = \frac{A}{2} \cdot (tanh(\hat{g_x}) + 1) \tag{3.1}$$

$$g_y = \frac{B}{2} \cdot (tanh(\hat{g_y}) + 1) \tag{3.2}$$

$$\delta = S \cdot (sigmoid(\hat{\delta}) + 1) \tag{3.3}$$

to produce (g_x, g_y) which encode the center coordinates of the grid of $N \times N$ Gaussian kernels that constitute the Receptive field. The equations result in the center coordinates intializing in the center of the input image width and height A and B when $\hat{g_x}, \hat{g_y}$ are equal to zero. As $\hat{g_x}, \hat{g_y}$ increase in magnitude, the output coordinates (g_x, g_y) can span across the entire input dimension. The stride parameter δ allows control of the distance between each of the $N \times N$ Gaussian kernels in the grid, thereby controlling the distance between sampling points in the downsampling operation. In this manner, it is possible to control the scope of the receptive field at the input without changing the output size of the ROI generation, *i.e.* the ROI dimension remains fixed at $N \times N$. The scalar value $\hat{\delta}$ is multiplied by a scaling factor S to enable more rapid and dynamic adaptation of receptive field size.

The position of each Gaussian kernel in the $N \times N$ grid is defined by computing the mean value in the x-dimension μ_x^i of a Gaussian function centered at each point in the grid:

$$\mu_x^i = g_x + (i - \frac{N}{2} - 0.5) \cdot \delta, \ i \in [0, N-1] \tag{3.4}$$

The mean y-position μ_y^j is computed in the same way, and i and j describe the column and row of the Gaussian kernel within the $N \times N$ grid.

Using the computed position, the truncated Gaussian kernel's weight on column i for pixels n in the x-dimension $F_x^i[n]$ (y-dimension weight is computed similarly) is defined:

$$F_x^i[n] = \begin{cases} \exp(\frac{(n-\mu_x^i)^2}{2\sigma}) & \text{for } n \in [\mu_x^i - \frac{\theta}{2},\ \mu_x^i + \frac{\theta}{2}] \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

The variance σ of the Gaussian function is a pre-defined and fixed value. The parameter θ determines where the Gaussian distribution is truncated. In this case θ is equal to 10, which means the Gaussian kernel is computed for the nearest 10×10 neighboring pixels of each of the $N \times N$ points.

With the truncated Gaussian kernel weights in both the x- and y- dimensions $\mathbf{F_x}$ and $\mathbf{F_y}$ computed, the value at each output coordinate i, j of the $N \times N \times 2$ ROI (third dimension with size 2 is for the 2 polarity channels of the event image) $v_{(x_i,y_j)}$ is computed,

$$v_{(x_i,y_j)} = \mathbf{F_x^i} \cdot \mathbf{I} \cdot \mathbf{F_y^j} \tag{3.6}$$

where $\mathbf{I}$ is one channel of the full resolution input event image. The computation is repeated for both channels of the input image.

3.1.2 Dynamic Average Pooling

As mentioned, the Gaussian kernels provide fast convergence during training due to differentiability and smooth gradient. Utilizing a different smooth function in place of the Gaussian function also provides the same benefits, such as the product of two complementary error functions. However, the downside of the Gaussian kernel is that it is expensive in terms of the computations that have to be performed in hardware, due to the exponential function and multiplications involved. Furthermore, the Gaussian kernels in the grid have the possibility of overlapping, which means that a given input event may need to be processed more than once in the manner described in the previous section.

In order to retain the desirable characteristics of the truncated Gaussian kernels - namely the ability to define and control receptive field width - while reducing the computational cost and complexity of ROI generation, a dynamic average pooling (DAP) operation is utilized. The dynamic average pooling resembles a standard, non-overlapping, average pooling operation, but with a non-static (dynamic) kernel size to ensure fixed output resolution. The kernel size is computed based on the width of the receptive field, which in the $x-$dimension (and similarly for the $y-$dimension) is defined by the maximum and minimum coordinates x_{max} and x_{min},

$$x_{max} = g_x + (\frac{N}{2} - 0.5) \cdot \delta + \frac{\theta}{2} \tag{3.7}$$

$$x_{min} = g_x - (\frac{N}{2} + 0.5) \cdot \delta - \frac{\theta}{2} \tag{3.8}$$

computed using the same decoded scalar values as in equations 3.1, 3.2, and 3.3. The kernel size k_{DAP} is then defined,

$$k_{DAP} = (x_{max} - x_{min})/N. \tag{3.9}$$

Because of the $\theta \times \theta$ truncation of the Gaussian kernels, the DAP receptive field is the same as when using truncated Gaussian kernels. This enables a network pre-trained with the truncated Gaussian kernels be fine-tuned with the DAP by freezing the parameters of the ROI prediction network and re-training the classification network on the new DAP-generated ROI. This fine-tuning procedure fully recovers any accuracy loss caused by the substitution of the ROI generation method.

3.2 Additional Hardware-aware Algorithm Optimizations

The use of truncated Gaussian kernels and DAP for ROI generation is motivated by the goal of maximizing the hardware efficiency of the algorithm. However, several other optimizations were also performed during algorithm development. The TRIP algorithm was optimized through sparsity-aware training, gradual layer-by-layer quantization-aware training, and parallelization of the subnetworks. These methods are described in detail in the following subsections.

3.2.1 Sparsity-aware Training

In an event-based processor, the number of computations is directly related to the number of events processed by each layer. By reducing the number of events, the number of computations and thereby the latency and energy can be reduced. Each event corresponds to an activation of a layer, *i.e.* the number of activations corresponds to the number of non-zero values in the output of the activation function of a layer. It is possible to train a network to minimize its activations using sparsity-aware training. An $L1$ regularization loss term proportional to the sum of the number of activations a_l in a layer l, for all L layers of a network,

$$L1 = w_L \cdot \sum_{L}^{l} w_l \cdot a_l \tag{3.10}$$

is weighted using layer-wise and overall weights w_l and w_L and added to the cross-entropy loss function used for classification [39]. This way, the network is incetivized to minimize its activations while obtaining high classification accuracy.

3.2.2 Quantization-aware Training

Weights and biases of neural network layers occupy memory on a processor. In the case of the SENECA neuromorphic processor the width of each parameter is 16 bits, and weights and biases are encoded using the brainfloat16 format. However, a layer with a large size such as a 576 × 256 FC layer will require 2.4 Mbit of memory to store its weights in 16 bit format, which exceeds SENECA's 2 Mbit capacity. As such, it is necessary to employ weight quantization to reduce the memory size occupied by each layer.

A 16 bit value can be quantized to a 4 bit integer value by using a shared power-of-two scaling factor. For a given layer, a shared scaling factor s is computed based on the distribution of the weights. Each 16 bit value is converted to the 4 bit integer which when multiplied by 2^s is nearest to its original value. The advantage of using this method is that weights can be stored using only the 4 bit integer values, added together during integration, and only multiplied in the firing step once integration is completed. The shared term occupies only one space in memory and is only multiplied once per neuron in the firing step. This reduces the memory utilization by 4× with only minimal loss in precision. It also has the added benefit of speeding up integration due to the lower number of bits used in the addition operations.

Quantizing all of the layers at once is very likely to cause significant accuracy loss. To avoid this issue, gradual, layer-by-layer quantization can be employed. Using this method, the parameters of the first layer of the network is quantized and fixed (*i.e.* not updated during training). Then the remaining layers in the network are fine-tuned (trained) on the output of the quantized layer until the accuracy loss has been recovered. Then, the next layer after the first one is quantized and fixed, and the process is repeated using the remaining layers. This is repeated step-by-step until each layer in the network is quantized. This process allows for full quantization with minimal accuracy loss.

3.2.3 Parallelization

In order to improve the latency of the algorithm, the ROI prediction step can be parallelized with the ROI generation and classification steps. By performing ROI generation and classification on the previous timestep's ROI prediction output, the latency can be almost halved since the ROI generation and classification do not need to wait for the ROI prediction to finish before processing. This means the ROI prediction processes the timebin of timestep $t + 1$ while the ROI generation and classification processes the timebin of timestep t. Although the information reaching the ROI generation and classification is one timestep out of sync with the ROI prediction, it was verified that this did not result in any accuracy loss. This is likely due to the fact that the latency of the hardware implementation is small in relation to the speed at which the ROI changes.

3.3 Datasets Used for Algorithm Benchmarking

In order to benchmark the algorithm performance in a way which allows for comparison with other reported methods in literature, the network architecture designed in this step is trained and tested on various datasets widely adopted in the research field. For the task of event-based classification, we selected one of the most commonly used datasets: DVS Gesture [2]. Because it is widely used in event-based vision research, it enables benchmarking against many other algorithms. Furthermore, DVS Gesture has also been used to benchmark multiple state-of-the-art neuromorphic processors such as Intel Loihi [7] and IBM TrueNorth [1]. Therefore, using DVS Gesture not only enables benchmarking of the TRIP algorithm, but also enables benchmarking of the hardware implementation on neuromorphic hardware against comparable state-of-the-art solutions. Another property of the gesture recognition dataset which makes it beneficial for testing the TRIP method is that gesture recognition inherently contains a low ROI-to-image ratio, since a gesture is typically performed by one or more body parts while areas like the head, body, or background are not involved. This makes the problem of gesture recognition well-suited for a hard attention approach.

Other widely used event-based datasets are either very similar to DVS Gesture, *i.e.* also gesture recognition datasets with similar number of classes and resolutions, or synthetically generated by displaying RGB datasets like MNIST or Caltech101 in front of an event-based camera. Real (*i.e.* not synthetically generated from non-event-based datasets) datasets are more accurately representative of real world applications of event-based vision, and using two very similar datasets is not highly informative. For these reasons, the second dataset selected for benchmarking the algorithm is the Marshalling Signals dataset [25], which is interesting for the task of testing ROI prediction since it contains samples recorded at multiple different distances. This will allow analysis of the ROI prediction's ability to adjust to informative regions that vary in size across samples. Additionally, Marshalling Signals is a higher resolution dataset which makes it interesting for studying the computational cost reduction of hard attention.

The third dataset used for benchmarking is synthetically generated and specially constructed to test the TRIP solution. In order to analyze the effect of visual noise on the ROI prediction, a special high resolution, noisy dataset is constructed based on the neuromorphic MNIST (N-MNIST) [26] dataset.

3.3.1 DVS Gesture Dataset and Training Details

DVS Gesture is a dataset recorded using a 128 × 128 resolution DVS128 event-based camera. The dataset contains recordings of different people performing various hand and arm gestures such as rolling their arms in front of them, waving one of their hands, playing air drums, or waving their arms in circular motions. There are 11 gesture classes in total, and the dataset consists of 1176 training samples and 288 testing samples.

Events were collected into 32 timebins per sample using the SpikingJelly preprocessing package [9] which constructs timebins/frames with a roughly equal number of events per frame. Additionally, the training dataset was augmented by peforming random rotation, scaling, and spatial translation of each sample. Each sample is downsampled to 16 × 16 using maxpooling before being fed to the first network layer.

The ROI prediction network consists of a three-layer CNN with increasing channel sizes 32, 64, and 128. Each convolutional layer has a kernel size of 3, a zero-padding of 1, and is followed by a maxpooling layer, a batch normalization layer, and a ReLU activation function. The CNN is followed by a ReLU RNN with size 256 and a fully connected (FC) layer with 3 neurons. The classification network consists of a two-layer CNN with identical parameters to the first two layers of the ROI predicition network. This is followed by two FC layers with sizes 256 and 11.

The entire network is trained for 1000 epochs using backpropagation, cross-entropy loss, a learning rate of 0.0001, and an Adam optimizer.

3.3.2 Marshalling Signals Dataset and Training Details

The Marshalling Signals dataset is developed by IMEC in 2022. It is a gesture recognition dataset with 346 × 224 resolution and contains gestures being performed at eight different distances from the camera. As a result, the size of the person performing the gesture in comparison to the overall size of the frame can be significantly smaller or bigger from sample to sample. The dataset contains 10 classes, 11,040 training samples, and 930 testing samples. Each sample is recorded at one of eight evenly spaced distances between 1.5 and 4.5 m from the camera. Each sample is collected in a single 960 ms timebin. The network used is the same as the one used for DVS Gesture. Because of the higher resolution of the dataset, the intial maxpooling downsample results in a 43×28 input resolution to the ROI prediction network. The network is trained for 25 epochs with a learning rate of 0.0001 and an Adam optimizer.

3.3.3 N-MNIST-based Synthetic Dataset and Training Details

N-MNIST is a handwritten digit recognition dataset generated by recording the MNIST dataset [18] with an event-based camera. It contains 60,000 training samples and 10,000 testing samples with a 34×34 resolution. The dataset used in this benchmarking experiment was generated by placing N-MNIST digits on randomly selected x, y-coordinates of an empty 128 × 128 event image. The digits are also randomly scaled by a factor between 1 and 2, and structured noise is added to the image by randomly sampling eight random 8 × 8 crops from eight randomly selected digits in the dataset. The crops are placed in random locations across the image. The network used is similar to the ones used with DVS Gesture and Marshalling Signals datasets, but with only two layers in each CNN (the last layer removed).

Baseline networks used for comparison were constructed using the same number of layers as TRIP. The baseline networks do not implement hard attention, and are used to compare the added effect of using hard attention against a similar non-attention network. The baselines were tested using inputs downsampled to different resolutions, 16 × 16, 32 × 32, and 64 × 64, and TRIP was tested using 16×16, 32×32. The networks consist of four subsequent convolutional layers with increasing channel dimensions 16, 32, 64, 128, followed by a size 128 (for 16 × 16 input) or size 512 (for 32 × 32 input) RNN and two FC layers. The networks are trained for five epochs with a learning rate of 0.0006 and an Adam optimizer.

3.4 Hardware Implementation and Benchmarking

The TRIP network described in section 3.3.1 was implemented on the SENECA neuromorphic processor using the software development kit (SDK) of the SENECA platform to program the RISC-V controller and NPE microkernels in C. The implementation was simulated using a full, gate-level simulation of the entire elaborated SENECA chip. Latency measurements, memory utilization, and area were extracted from the simulation and elaboration results. The power utilized by each core was measured using Cadence Joules, and the energy computed using the latency and power of each core.

3.4.1 Event-CNNs

The entire network is fully event-based, and convolutional layers are implemented as depth-first event-CNNs. The event-CNN performs convolution, maxpooling, and batch normalization before activation in order to maximize sparsity in the output. The events in the CNN are processed using depth-first scheduling, which parses events in a pre-defined spacial order from top left to bottom right of the input layer, allowing it to finish processing one point in X, Y space without having to return in the same frame. This allows the convolutional layer to update states and fire more rapidly, and for subsequent layers to immediately consume the produced events. This results in lower latency and greater parallelization due to events being produced faster, and also less memory utilization since the memory can be released when the convolution moves from one spatial location to another.

3.4.2 Event-grouping

The weight quantization procedure described in section 3.2.2 is combined with event-grouping to maximize efficiency. Since 4-bit quantization allows each word in the memory to store four weights, four incoming events that share a destination address can be processed at once while only reading one weight address. The four events are combined into one so a neuron state is read only once and updated four times. This results on average in a halving of the energy and latency per synaptic operation. This works under the assumption that the timing between the group events do not differ significantly enough to result in discrepancy when processed simultaneously.

3.4.3 Baseline Network

In addition to TRIP, a baseline network was implemented in the same manner. The baseline network is implemented in order to enable comparison against a network with similar parameters as TRIP that does not use hard attention. The baseline network consists of five sequential convolutional layers of increasing channel sizes 32, 64, 128, 128, and 128. The other parameters are the same as in the convolutional layers of the TRIP network, including the order of maxpooling, batchnormalization, and ReLU activation. The convolutional layers are followed by a recurrent layer with size 256 and an FC layer with size 11. The input dimension to the network is 32×32, as a result of a maxpooling downsampling operation of the full resolution input image. The same hardware-optimizations applied to TRIP are also applied to the baseline, *i.e.* quantization-aware training, sparsity-aware training, event-CNNs, depth-first convolution, and spike grouping.

Chapter 4

Results and Discussion

4.1 DVS Gesture Dataset

The benchmarking results of TRIP on DVS Gesture compared to other state-of-the-art methods from literature is shown in Table 4.1. The mean accuracy is similar to other state of the art networks, between 97% and 98%. A difference of 0.3% accuracy corresponds to only a single test sample, and therefore all models except the simple LSTM in the first row have negligible difference in error. However, TRIP requires 46× less computations than the other models to achieve the same accuracy. TRIP also requires less parameters than all networks except ConvLIAF. ConvLIAF utilizes LIAF blocks which require less number of parameters than recurrent and convolutional blocks [35]. The ConvLIAF network has roughly half the number of parameters as TRIP, but 65× more computations due to relying on a higher input resolution. TRIPs ability to process lower input resolution allows for simpler network architectures with less parameters to be used for the same task when compared to similar convolutional and recurrent architectures.

A visualization of the receptive field used in TRIPs ROI generation is shown in Figure 4.1. Across samples, TRIP locates the receptive field at the location where the gesture is taking place. For instance, in the Left Hand Clockwise gesture, the person performing the gesture is rotating their arm in a circular motion, and the receptive field follows the motion of the arm. In the Right Hand Wave gesture, the receptive field is focused on the right arm, moving slightly as the arm waves back and forth. For stationary gestures such as the Arm Roll, the receptive field remains stationary and fixed on the arms performing the gesture. These visualizations suggest that the network is learning to focus its receptive field on the part of the body performing the gesture, most likely because this is the most informative region for classification.

Architecture	Input Resolution	Param	Effective MACs (Single Timebin)	Accuracy [%] (mean ± std)	Accuracy [%] (Maximum)
LSTM [14]	32×32	7.4M	3.9M	–	86.8
AlexNet+LSTM[15]	128×128	8.3M	601.3M	–	97.7
CNN+EGRU [30]	128×128	4.8M	80.6M	97.3± 0.4	97.8
ConvLIAF [35]	32×32	**0.22M**	113.3M	–	97.6
TRIP (This work)	16×16+12×12	0.46M	**1.75M**	**97.6 ± 0.5**	**98.6**

Table 4.1: Performance comparison of TRIP against other state-of-the-art solutions on DVS Gesture dataset.

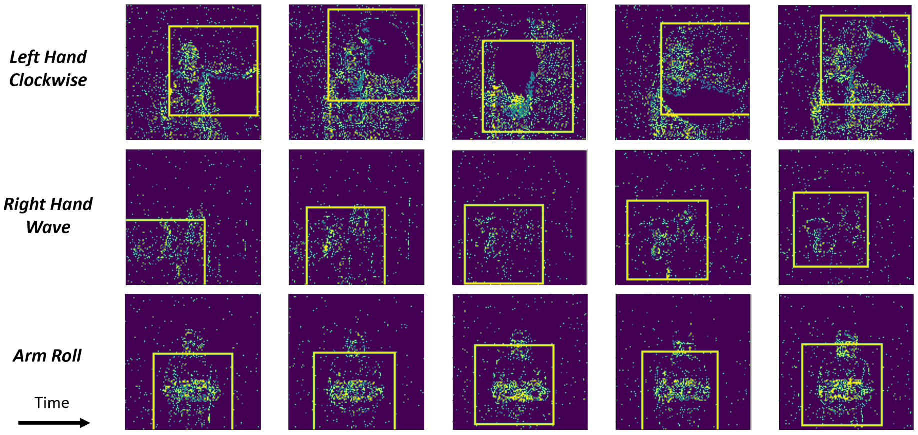


Figure 4.1: Visualization of ROI receptive fields from three different gesture samples of the DVS Gesture dataset. The receptive field is visualized as a yellow square superimposed above the event image.

4.2 Marshalling Signals Dataset

Table 4.2 shows the accuracy, number of parameters, and FLOPs of TRIP compared to the previously reported results of ResNet18 and EfficientNet-B1. Since [25] did not report activation sparsity or effective MACs, we use network floating point operations (FLOPs) as the metric for computational cost, without considering activation sparsity. FLOPs measure the number of computations performed by a network and is proportional to layer sizes and numbers. The TRIP network has 18× less FLOPs than the best performing competitor EfficientNet-B1 while slightly outpeforming in accuracy by 1%. Visualizing the ROI receptive field shows the network's capability to adjust the ROI size in response to different gestures' distances, increasing or decreasing to match the size of the region encompassed by the gesture in a given sample.

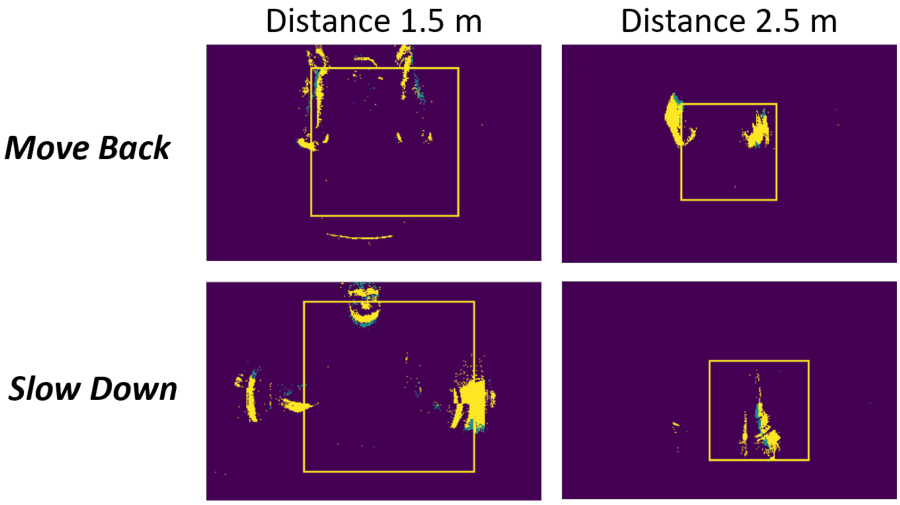


Figure 4.2: ROI receptive field visualization for sample gestures of the Marshalling Signals dataset performed from different camera distances.

Architecture	Param	FLOPs	Accuracy [%]
ResNet18 [25]	11.7M	1810M	74.6
EfficientNet-B1 [25]	7.794M	690M	82.6
TRIP (This work)	**4.13M**	**37.0M**	**83.6**

Table 4.2: Performance comparison of TRIP against state-of-the-art networks on the Marshalling Signals dataset.

4.3 Synthetic N-MNIST-based Dataset

The results in Table 4.3 show that TRIP allows network FLOPs to be reduced from 32×32 to 16×16 without sacrificing accuracy. For the baseline network, this reduction in resolution results in unacceptable accuracy loss ($> 20\%$), while for TRIP accuracy is only reduced by less than one percent. Additionally, the number of FLOPs required to obtain 96% accuracy is nearly halved.

Visualization of the samples in Figure 4.3 shows the network is capable of consistently focusing the receptive field on the digit even in the presence of structured noise. This suggests the network is not merely being triggered by presence of activity, but is in fact capable of discerning informative regions from uninformative ones.

4.4 Hardware Implementation: Benchmarking Results

The hardware implementation of TRIP is compared with other neuromorphic hardware implementations in Table 4.4. Since different works vary in whether they

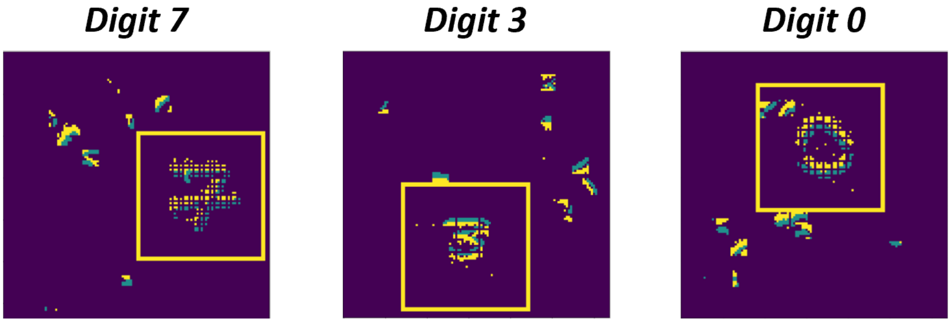


Figure 4.3: ROI visualization on three different N-MNIST samples (left to right: digit 7, digit 3, and digit 0).

Architecture	Param	FLOPs	Accuracy [%]
			(mean ± std)
Baseline (16x16)	0.31M	6.0M	71.8± 2.3
Baseline (32x32)	0.67M	24.4M	93.0± 0.6
Baseline (64x64)	0.67M	57.4M	96.2± 0.9
TRIP (16x16)	0.30M	16.0M	95.4 ± 0.4
TRIP (32x32)	0.65M	28.0M	96.1 ± 0.3

Table 4.3: Performance comparison of TRIP against baseline networks on the synthetic N-MNIST dataset.

report metrics for single or multiple timebins, TRIP and the SENECA baseline are evaluated on both metrics to enable comparison against all of the reported works. The TRIP implementation consumes less area and energy than all the other reported solutions, while also decreasing the error by 50% or more. One implementation outperforms TRIP in latency by 3 ms. This is a result of a very high degree of network parallelism in the implementation, which sacrifices a large amount of area (7× more than TRIP) in order to minimize latency. While the Loihi implementation succeeds in reducing the latency significantly, this trade-off limits the scalability of the solution, in particular for higher resolution inputs.

The baseline network on SENECA consumes less area than TRIP, due to requiring one less core as a result of not including the DAP block and having one less FC layer. However, the latency is 3× higher and energy 2.5× higher compared to TRIP, due to larger input dimensions of the convolutional layers increasing the number of events that have to be processed.

The memory utilization, energy, and latency of each individual core the TRIP implementation is visualized in Figure 4.4. The figure also visualizes the parallelization of the three components classification, ROI generation, and ROI prediction. The most significant energy is consumed by the earlier convolutional layers in either network, due to having the largest input dimensions and thereby the largest number of events to process.

			Single Timebin			Multiple Timebins		
Hardware	Solutions	Area [mm^2]	Latency [ms]	E_{inf} [uJ]	Accuracy [%]	Latency [ms]	E_{inf} [uJ]	Accuracy [%]
Loihi [7]	SCNN [29]	>8.20	11	–	89.6	–	–	–
Loihi [7]	SCNN [22]	24.19	–	–	–	**22.0**	2731	96.2
TrueNorth [1]	SCNN [2]	383.8	–	–	**91.8**	104.6	18702	94.6
SENECA [33]	Event-CNN	**3.29**	–	–	–	78.9	1069.2	97.3
SENECA [33]	TRIP	4.23	**2.7**	**35.86**	91.1	25.8	**430.32**	**98.3**

Table 4.4: Comparison of TRIP and SENECA baseline with state-of-the-art neuromorphic implementations on DVS Gesture dataset.

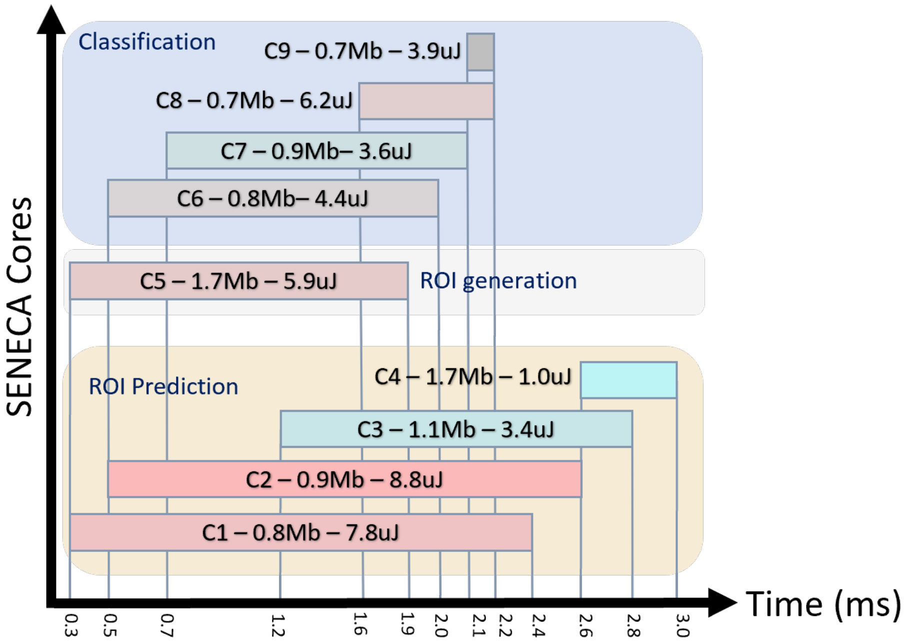


Figure 4.4: Overview of per-core utilization of TRIP implementation, including latency, memory usage, and energy of each core.

Chapter 5

Outlook and Conclusion

The TRIP framework was successfully applied to improving the hardware efficiency of event-based classification on a neuromorphic processor. Compared to the higher resolution baseline network, energy, latency, and accuracy could be improved. The combined hardware-optimizations resulted in the implementation outperforming state-of-the-art neuromorphic chips in all hardware metrics. The only exception is one implementation that was able to achieve a minor improvement in latency by using scaling-inhibitive and highly area-consuming parallelization techniques. Algorithmic analysis revealed major efficiency improvements over state-of-the-art baseline networks across multiple datasets, highlighting the effectiveness of the TRIP approach to hard attention in both software and hardware. Through its substantial efficiency improvements, TRIP delivers an effective method of enabling high resolution processing on neuromorphic processors.

In the future, the method outlined in this work could be expanded and applied to high resolution event-based object detection datasets. Currently the Prophesee automotive dataset promises to enable efficient event-based sensing for applications such as autonomous driving [28]. However, huge challenges with processing the high resolution data limit the ability of networks to process the dataset. Difficulties in both achieving high accuracy and limiting the computational cost are holding back the devlopment of efficient systems that can leverage high resolution event-based datasets. The TRIP framework offers a potential solution to both these problems, but would need to be reworked for the object detection task.

Additionally, the observations made in this work can inform future development of in-sensor processing capabilities for event-based cameras. Developers of event-based cameras such as Prophesee are expanding to include increasingly sophisticated pre-processing on the visual front-end to improve the overall efficiency of the entire system containing camera and processor hardware [37, 38, 17]. Framed in this context, TRIP highlights the usefulness of in-sensor pre-processing capabilities for average pooling operations and similar downsampling operations. Despite maximized hardware optimization, the ROI generation step using dynamic average pooling still consumed significant latency and power. An integrated, in-sensor pre-processing capability could increase the efficiency of this step and significantly improve the overall efficiency of the entire pipeline.

References

[1] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon 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*, 34(10):1537–1557, 2015.

[2] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7243–7252, 2017.

[3] Jimmy Lei Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. *International Conference on Learning Representations*, 32, 2015.

[4] Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. Attention mechanisms for object recognition with event-based cameras. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1127–1136. IEEE, 2019.

[5] Yuning Chai. Patchwork: A patch-wise attention network for efficient object detection and segmentation in video streams. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3415–3424, 2019.

[6] Gregory Cohen, Saeed Afshar, Garrick Orchard, Jonathan Tapson, Ryad Benosman, and Andre van Schaik. Spatial and temporal downsampling in event-based visual classification. *IEEE Transactions on Neural Networks and Learning Systems*, 29(10):5030–5044, 2018.

[7] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, 38(1):82–99, 2018.

[8] Gamaleldin Elsayed, Simon Kornblith, and Quoc V Le. Saccader: Improving accuracy of hard attention models for vision. *Advances in Neural Information Processing Systems*, 32, 2019.

[9] Wei Fang, Yanqi Chen, Jianhao Ding, Zhaofei Yu, Timothée Masquelier, Ding Chen, Liwei Huang, Huihui Zhou, Guoqi Li, and Yonghong Tian. Spikingjelly: An open-source machine learning infrastructure platform for spike-based intelligence. *Science Advances*, 9(40):eadi1480, 2023.

[10] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J Davison, Jörg Conradt, Kostas Daniilidis, et al. Event-based vision: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(1):154–180, 2020.

[11] Jonathan M Gant, Andrzej Banburski, and Arturo Deza. Evaluating the adversarial robustness of a foveated texture transform module in a cnn. In *SVRHM 2021 Workshop@ NeurIPS*, 2021.

[12] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. In *International conference on machine learning*, pages 1462–1471. PMLR, 2015.

[13] Meng-Hao Guo, Tian-Xing Xu, Jiang-Jiang Liu, Zheng-Ning Liu, Peng-Tao Jiang, Tai-Jiang Mu, Song-Hai Zhang, Ralph R Martin, Ming-Ming Cheng, and Shi-Min Hu. Attention mechanisms in computer vision: A survey. *Computational visual media*, 8(3):331–368, 2022.

[14] Weihua He, YuJie Wu, Lei Deng, Guoqi Li, Haoyu Wang, Yang Tian, Wei Ding, Wenhui Wang, and Yuan Xie. Comparing snns and rnns on neuromorphic vision datasets: Similarities and differences. *Neural Networks*, 132:108–120, 2020.

[15] Simone Undri Innocenti, Federico Becattini, Federico Pernici, and Alberto Del Bimbo. Temporal binary representation for event-based action recognition. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 10426–10432. IEEE, 2021.

[16] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, 1998.

[17] Xabier Iturbe, Nassim Abderrahmane, Jaume Abella, Sergi Alcaide, Eric Beyne, Henri-Pierre Charles, Christelle Charpin-Nicolle, Lars Chittka, Angélica Dávila, Arne Erdmann, et al. Nimbleai: Towards neuromorphic sensing-processing 3d-integrated chips. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2023.

[18] Daniel Keysers. Comparison and combination of state-of-the-art techniques for handwritten character recognition: topping the mnist benchmark. *arXiv preprint arXiv:0710.2231*, 2007.

[19] Fanjie Kong and Ricardo Henao. Efficient classification of very large images with tiny objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2384–2394, 2022.

[20] Hugo Larochelle and Geoffrey E Hinton. Learning to combine foveal glimpses with a third-order boltzmann machine. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[21] Yan Luo, Xavier Boix, Gemma Roig, Tomaso Poggio, and Qi Zhao. Foveation-based mechanisms alleviate adversarial examples. *arXiv preprint arXiv:1511.06292*, 2015.

[22] Riccardo Massa, Alberto Marchisio, Maurizio Martina, and Muhammad Shafique. An efficient spiking neural network for recognizing gestures with a dvs camera on the loihi neuromorphic processor. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2020.

[23] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. *Advances in neural information processing systems*, 2014.

[24] Orlando Moreira, Amirreza Yousefzadeh, Fabian Chersi, Gokturk Cinserin, Rik-Jan Zwartenkot, Ajay Kapoor, Peng Qiao, Peter Kievits, Mina Khoei, Louis Rouillard, et al. Neuronflow: a neuromorphic processor architecture for live ai applications. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 840–845. IEEE, 2020.

[25] Leon Müller, Manolis Sifalakis, Sherif Eissa, Amirreza Yousefzadeh, Paul Detterer, Sander Stuijk, and Federico Corradi. Aircraft marshaling signals dataset of fmcw radar and event-based camera for sensor fusion. In *2023 IEEE Radar Conference (RadarConf23)*, pages 01–06. IEEE, 2023.

[26] Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in neuroscience*, 9:437, 2015.

[27] Athanasios Papadopoulos, Pawel Korus, and Nasir Memon. Hard-attention for scalable image classification. *Advances in Neural Information Processing Systems*, 34:14694–14707, 2021.

[28] Etienne Perot, Pierre De Tournemire, Davide Nitti, Jonathan Masci, and Amos Sironi. Learning to detect objects with a 1 megapixel event camera. *Advances in Neural Information Processing Systems*, 33:16639–16652, 2020.

[29] Bodo Rueckauer, Connor Bybee, Ralf Goettsche, Yashwardhan Singh, Joyesh Mishra, and Andreas Wild. Nxtf: An api and compiler for deep spiking neural networks on intel loihi. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 18(3):1–22, 2022.

[30] Anand Subramoney, Khaleelulla Khan Nazeer, Mark Schöne, Christian Mayr, and David Kappel. Efficient recurrent architectures through activity sparsity and sparse back-propagation through time. In *The Eleventh International Conference on Learning Representations*, 2023.

[31] M Tan. rethinking model scaling for convolutional neural networks. arxiv. 2019 doi: 10.48550. *arXiv: 1905.11946*, 2019.

[32] Guangzhi Tang, Ali Safa, Kevin Shidqi, Paul Detterer, Stefano Traferro, Mario Konijnenburg, Manolis Sifalakis, Gert-Jan van Schaik, and Amirreza Yousefzadeh. Open the box of digital neuromorphic processor: Towards effective algorithm-hardware co-design. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2023.

[33] Guangzhi Tang, Kanishkan Vadivel, Yingfu Xu, Refik Bilgic, Kevin Shidqi, Paul Detterer, Stefano Traferro, Mario Konijnenburg, Manolis Sifalakis, Gert-Jan van Schaik, et al. Seneca: building a fully digital neuromorphic processor, design trade-offs and challenges. *Frontiers in Neuroscience*, 17, 2023.

[34] Manish Reddy Vuyyuru, Andrzej Banburski, Nishka Pant, and Tomaso Poggio. Biologically inspired mechanisms for adversarial robustness. *Advances in Neural Information Processing Systems*, 33:2135–2146, 2020.

[35] Zhenzhi Wu, Hehui Zhang, Yihan Lin, Guoqi Li, Meng Wang, and Ye Tang. Liaf-net: Leaky integrate and analog fire network for lightweight and efficient spatiotemporal information processing. *IEEE Transactions on Neural Networks and Learning Systems*, 33(11):6249–6262, 2022.

[36] Jason Yik, Soikat Hasan Ahmed, Zergham Ahmed, Brian Anderson, Andreas G Andreou, Chiara Bartolozzi, Arindam Basu, Douwe den Blanken, Petrut Bogdan, Sander Bohte, et al. Neurobench: Advancing neuromorphic computing through collaborative, fair and representative benchmarking. *arXiv preprint arXiv:2304.04640*, 2023.

[37] Feichi Zhou and Yang Chai. Near-sensor and in-sensor computing. *Nature Electronics*, 3(11):664–671, 2020.

[38] Yue Zhou, Jiawei Fu, Zirui Chen, Fuwei Zhuge, Yasai Wang, Jianmin Yan, Sijie Ma, Lin Xu, Huanmei Yuan, Mansun Chan, et al. Computational event-driven vision sensors for in-sensor spiking neural networks. *Nature Electronics*, pages 1–9, 2023.

[39] Zeqi Zhu, Arash Pourtaherian, Luc Waeijen, Egor Bondarev, and Orlando Moreira. Star: Sparse thresholded activation under partial-regularization for activation sparsity exploration. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 4554–4563. IEEE, 2023.