Less is More: The Neuromorphic Engineering Advantage

Paul Kirkland

A thesis submitted for the degree of

Doctor of Philosophy

Neuromorphic Sensor Signal Processing Lab

Centre for Signal and Image Processing

Department of Electronic and Electrical Engineering

University of Strathclyde

Glasgow

2021

Declaration

Signed: Paul Kirkland
Date: 1st December 2020

Dedication

In Memory of George Edward Matich

To my industrial supervisor and friend, without whom I would not have started, or been so well guided through this journey. I hope to inspire and facilitate the Neuromorphic research field and new researchers much like you did with me.

Acknowledgement

I would like to express my very great appreciation to my PhD supervisor Prof John J. Soraghan, for his wise guidance, support, and advice throughout my whole PhD. Thanks to him the whole journey has been made more enjoyable. Giving me space and room to conduct my research and gain experience, while being approachable and accessible whenever I had any concerns or issues. I wish him all the best on his impending retirement, which I'm 99% sure is not correlated with my PhD completion.

I would also like to thank my second supervisor Prof Stephan Weiss, as although my research took me outside the remit of his research, he was always friendly and approachable should I even require assistance.

A special thanks to my third supervisor Dr Gaetano Di Caterina, who was very often my first port of call for any problem or issue. Thanks for your support and guidance throughout my PhD your advice has been a valuable resource.

I would like to offer my special thanks to George Matich and Leonardo MW ltd for sponsoring my PhD. Alongside the funding from the CeSIP and the EEE Department. Without this valuable funding source, my PhD dream could not have been realised.

In particular, I would also like to thank my industrial supervisor George Matich, his advice, motivation and passion helped me to believe in a subject area I didn't even know existed before starting.

I would like to thank my lovely wife Madeleine (Pops) for her abundance of love and support throughout this research journey. This is especially so during the long nine months of working from home thanks to the Covid-19 lockdown restrictions. Thank you for helping me relax, unwind and leave the stresses of work at work, even if it often seemed as though I did not.

Lastly, I would like to thank two of my best friends Andreas Aßmann and Lukasz Zapotoczny as we collectively guided our own EngD and PhD journeys. You have played an important role in the ups and downs of the PhD process, supporting or celebrating as required.

Abstract

Artificial Neural Networks (ANN) have helped to revolutionise the world of Computer Vision (CV) with modern interpretations of the ANN based on visual cortex creating Convolutional Neural Network (CNN) and the research movement of Deep Learning (DL). Another more biologically inspired movement is that of Neuromorphic Engineering with its spiking neuron model and Spiking Neural Network (SNN). Recently, research has merged large parts of these two research fields allowing Neuromorphic Engineering to gain more momentum, creating a paradigm shift in the approach to CV. This provides the reality of having asynchronous, low latency and low computational power approach available when utilising the SNN.

A novel solution to both semantic segmentation and a framework in which to utilise it is developed. The Perception-Understanding-Action (PUA) framework aims to add a contextual understanding through semantic segmentation, with a low latency and computational SNN, entitled the Spiking Segmentation Network (SpikeSEG). This framework aims to improve the low latency and reactive Perception-Action Cycles used in many constrained robotics tasks. By adding understanding, a low latency approach aims to add no noticeable latency to the system, exploiting the asynchronous advantage that is available when using Neuromorphic Vision Sensors (NVS). The framework allows an end-to-end spiking system to be realised where latency and computational power are limiting factors.

Further to semantic segmentation, a novel method for instance segmentation is also proposed with the Hierarchical Unravelling of Linked Kernels with Similarity Matching through Active Spike Hashing (HULK-SMASH) algorithm. This solves the difficult problem of unsupervised class instance clustering, deciphering between separate instances of classes on a per sequence and sequence to sequence basis. The algorithm allows each instance within the classification layers to be traced during the decoding back to the pixel space, allowing a pixel-wise instance mapping of each class instance. The algorithm is successfully able to identify the same person within a neuromorphic vision face dataset, while also being able to successfully recover tracking of instance after complete occlusion.

Additionally, a novel solution to the non-typical imaging problem, temporal imaging, is presented. This method of 3D imaging makes use of minimal spatial data from a single-point sensor, but with a high-resolution of temporal data captured in a time-of-flight (ToF) manner. To produce images from this method the novel network Spiking-Single Point Imager (Spike-SPI) is required to solve the inverse retrieval problem of creating a 3D depth map from only the temporal data, inferring the spatial locations based on previous temporal sequences it has seen. This network makes use of both encoder-decoder networks and CNNs and their training methods to train the system. These are then converted to an SNN to allow asynchronous, lower latency and lower computational processing. Spike-SPI was able to outperform the current state of the art in 3D depth estimation, losing no accuracy in the CNN to SNN conversion process while gaining the aforementioned benefits.

Contents

List of acronyms

1D	One dimension
2D	Two dimension
3D	Three dimension
AER	Address-Event-Representation
AI	Artificial Intelligence
ANN	Artificial Neural Network
ASH	Active Spike Hashing
CCTV	Closed Circuit Television
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CV	Computer Vision
DARPA	Defence Advanced Research Projects Agencies
DL	Deep Learning
DVS	Dynamic Vision Sensor
FCN	Fully Convolutional Networks
GPU	Graphical Processing Unit
HBP	Human Brain Project
HH	Hodgkin-Huxley
HULK	Hierarchical Unravelling of Linked Kernels
IF	Integrated and Fire
IoU	Intersection over Union
IRF	Instrument Response Function
IZ	Izhikevich
LCD	Liquid Crystal Display
LiDAR	Light Detection and Ranging
LIF	Leaky-Integrated and Fire
LTD	Long-Term Depression
LTP	Long-Term Potentiation
MCP	McCulloch Pitts Neuron

ML	Machine Learning
MLP	Multi-Layer Perceptron
NEF	Neural Engineering Framework
NM	Neuromorphic
NN	Neural Networks
NVS	Neuromorphic Vision Sensor
PA	Perception Action
PDF	Probability Density Function
PEAT	Pre-Emptive and Adaptive Thresholding
PSTH	Peri-Stimulus-Time Histogram
PUA	Perception Understanding Action
ReLU	Rectied Linear Unit
R-SNR	Reconstruction Signal-to-Noise Ratio
R-STDP	Reinforcement Spike-Timing-Dependent-Plasticity
RM	Retinamorphic
RMSE	Root Mean Square Error
SAM	Spiking Activity Map
SCNN	Spiking Convolutional Neural Network
SMASH	Similarity Matching through Active Spike Hashing
SNN	Spiking Neural Network
SNR	Signal-to-Noise Ratio
SP-SPAD	Single Point, Single Photon Avalanche Diode
STDP	Spike-Timing-Dependent-Plasticity
SWaP	Size, Weight and Power
SyNAPSE	Systems of Neuromorphic Adaptive Plastic Scalable Electronics
TCSPC	Time Correlated Single Photon Counting
ToF	Time of Flight
TTFS	Time-To-First-Spike
UAV	Unidentified Aerial Vehicle
VLSI	Very Large Scale Integration

List of figures

List of tables

Chapter 1

Introduction

We are living in an unprecedented time for Artificial Intelligence (AI). AI has developed rapidly due to advancements in algorithms, computing power and big data. The most prominent AI field is that of Deep Learning (DL) [1]. This is especially true in the realm of Computer Vision (CV), where the majority of the state of the art systems involve DL implementations [2].

For AI to perceive the world, CV must act as an interface between the computer and the world. The process of making sense of the world, based on the light that is scattered from objects into our eyes, is an ability that comes naturally to the vast majority of people. When in fact our conscious mind perceives the world as a series of patterns of motions, encoding the objects and their locations subconsciously. This lack of awareness is tied conceptually to the sentiment "The eyes look, but the brain sees". This suggests to give the "Computer" its "Vision", there needs to be both an eye-like sensor and a brain-like processor.

This often leads to the notion of CV needing to be more than just perception, with a level of cognition or understanding also required. It is this cognition or understanding (brain-like processing) that DL through the use of Artificial Neural Networks (ANN) [3] tries to achieve, acting as the computers visual cortex. It is the ANN's dramatic success in pattern recognition tasks and in particular the Convolutional Neural Network (CNN) [4] within the visual domain, that has seen the surge in DL applications. The CNN has fuelled great strides in a variety of computer vision problems, such as image recognition (AlexNet and ResNet) [5, 6], object detection (Faster R-CNN and RFCN) [7, 8], object tracking (FCNT and MD Net) [9, 10] and semantic and instance segmentation (FCN, SegNet, U-Net and Mask R-CNN) [11–14].

1.1 Research Motivation

With the advancements in DL through the use of CNNs has also come the need for more power and more data to train these systems [15]. The majority of the methods developed within neural networks stem from biologically inspired approaches [3, 16, 17]. However, as the field has progressed it has started to lose this biological aspect and with it some of the unique benefits of the biological system it was inspired by, especially in terms of Size, Weight and Power (SWaP) [18, 19]. However, there is a more biologically inspired approach to CV with a Retinamorphic (RM) [20], eye-like sensing and Neuromorphic (NM) [21], brain-like processing. Together with this sensing and processing approach form what is typically known as NM Engineering [22]. This NM engineering sensing and processing method are biologically inspired, such that it is spiking based, utilising a spiking variation of the neural network the SNN. This approach has several motivating factors over the traditional approaches but can be refined into a theoretical and engineering basis.

Concerning the theoretical motivation with the spiking approach, bio-inspired learning (weight modification) permits the ability of processing temporal data in a more intuitive and localised manner [23], in other words, a comparison of the relative timing of the spiking activity between pairs of directly connected neurons. This is better known as spike-timing-dependent plasticity (STDP) learning [24]. This type of learning is not permissible with the aforementioned DL approaches due to the ANN/CNN utilised not emitting spikes. This change in the neuron model to an SNN enables the local weight modification of adjacent pairs of neurons. This temporal sparsity coupled with the spatial locality of learning results in a less computational complex method for processing [25,26]. This similar spiking method of sensing allowing even greater returns in lowering computational overhead [25]. All the while receiving further benefits such as mitigating data redundancy, online learning and reduced latency through asynchronousity [27].

Concerning the engineering motivation, the main advantage of this biological spiking approach is the spikes. That is the realisation that it is the spiking events that consume energy and using fewer spikes, that have higher information content, reduces overall energy consumption [22, 28, 29]. This same advantage occurs in both the sensing and processing hardware [22, 28, 30]. Permitting low energy spiking hardware based on the property that spikes are sparse in time [22]. This allowed the theoretical motivation to become the engineering motivation, with mitigating data redundancy, online learning and reduced latency and computational complexity through asynchronousity being attainable. That is in stark contrast to the present DL approaches which rely on training of deep traditional networks on energy-intensive high-end graphic cards, often forgoing the precise temporal information, to reduce complexity [15].

1.2 Summary of Original Contributions

The work conducted in this research has led to several novel contributions in the field of NM engineering:

1. Design of a novel encoder/decoder architecture for use with a Retinamorphic sensor and Neuromorphic processing. The network makes use of both the spiking manner of the input and processing to implement a semantic segmentation network SpikeSEG. The network also employs a biologically inspired unsupervised learning mechanism in STDP for feature extraction. The network introduces a novel spiking implementation for upsampling within the decoder section of the network, exploiting the spatial-temporal sparsity of both the sensor input and processing method.

   The SpikeSEG network also has novel adaptive neuron thresholding and neuron pruning algorithms. These novel algorithms respectively help with additional sensor input noise associated with NM sensors and non-converging neurons during training.(Chapter 4)

2. Application of the SpikeSEG network within a novel spiking end to end tracking framework, the Perception Understanding Action (PUA) framework. Utilising the temporal advantage of the spiking perception method, through to the SpikeSEG network adding contextual understanding (processing) to the system, before continuing the low latency approach through to the action (control). This results in advancement over the simple Perception-Action Cycle systems through the ability to segment the information to focus on particular areas of objects of interest. (Chapter 4)

3. Design of a novel spatial-temporal feature matching algorithm called the Hierarchical Unravelling of Linked Kernels (HULK) and Similarity Matching through Active Spike Hashing (SMASH). To the best of the author's knowledge, this work presents the first spiking instance segmentation method using unsupervised learning. This algorithm makes use of the STDP learned features and uses both feature and temporal correlations to identify if instances are grouped or not. The approach can not only identify individual

instances of a class but can also be utilised to group inter-class instances e.g. the same person in a face dataset. (Chapter 5)

4. Design of a novel network, Spiking Single Pixel Imager (Spike-SPI) to translate 1D depth data from a single point LiDAR sensor and convert it into a 3D depth map. Spike-SPI makes uses of the traditional back-propagation trained CNN that is then converted into a spiking version. This network is a variational encoder/decoder translating the depth histogram features into spatial features with depth. The network is designed to exploit the unique features of the different components of the system. Utilising the feature extraction capabilities of Convolutional Neural Networks (CNN), the high dimensional compressed latent space representations of an Encoder-Decoder network structure and the asynchronous processing capabilities of an SNN. Furthermore, the convolutional decoder can exploit the strong spatially local correlation present in natural images to form both qualitative and quantitative better images. The network can outperform the original ANN in performance and matches the pre-conversion CNN results however, using significantly less computational overhead. (Chapter 6)

1.3 Organisation of the Thesis

The remainder of this thesis is organised as follows. Chapters 2 and 3 give a review of the relevant literature providing further background and insight into the techniques used throughout the thesis. Chapter 2 introduces the neural network leading into the CNN and encoder/decoder network structure. With a description of typical use cases for this network type in image-based segmentation tasks. Chapter 3 presents a review of the changes made to ANNs to get to SNNs. This also covers the main benefits and developments that the introduction of spikes has in terms of algorithms and learning strategies. Chapter 3 also provides an insight into the general field of NM engineering with the corresponding current developments in hardware and software. Chapter 4 through 6 present the novel contributions of the thesis. Chapter 4 introduces the novel PUA framework with the main research focus being the understanding or processing element. This understanding comes from the SpikeSEG network a convolutional SNN (SCNN) encoder/decoder network designed to allow semantic segmentation filtering out the unwanted information from the perception stage through to control. This provides a framework to maintain the useful features of NM engineering especially the sparsity and temporal aspects. Chapter 5 builds further on the SpikeSEG semantic segmentation network with the introduction of the Hierarchical Unravelling of Linked Kernels with Similarity Matching through Active Spike Hashing (HULK SMASH) algorithm. The HULK section of the algorithm reworks the decoding section of the SpikeSEG network with a class instance tracking method that allowing the decoding process to be unravelled linking the classification instance to be mapped to the original pixel space in an instance-based and not semantic-based manner. The SMASH section is then presented the seemingly difficult task of differentiating between instances that are grouped or individual, typically a regression process. However, without regression SMASH can differentiate between instance that form the same objects or different ones with a feature similarity and spatial-temporal proximity matching process. Chapter 6 introduces a novel spiking network, Spiking Single Pixel Imaging (SpikeSPI) that processes 1D depth data from a time of flight sensor and through a variational encoder/decoder network output a 3D depth map. This chapter presents a different approach to spiking networks from the two prior chapters with a converted network based on a data-driven approach due to the complexity of the ill-posed spatial reconstruction problem. This further highlights the variety in NM engineering approaches which allow a variety of solution that can be tailored to the task at hand. The Conclusion and Future work, Chapter 7, then summarises the finding in the thesis and highlights areas that could be continued in subsequent research.

1.4 Publications

1. Xing, Y., Kirkland, P., Di Caterina, G., Soraghan, J. and Matich, G., 2018, October. Real-time embedded intelligence system: emotion recognition on Raspberry Pi with Intel NCS. In International Conference on Artificial Neural Networks (pp. 801-808). Springer, Cham.

2. Kirkland, P., Di Caterina, G., Soraghan, J., Andreopoulos, Y. and Matich, G., 2019, September. UAV detection: a STDP trained deep convolutional spiking neural network retina-neuromorphic approach. In International Conference on Artificial Neural Networks (pp. 724-736). Springer, Cham.

3. Kirkland, P., Di Caterina, G., Soraghan, J., Thomas, K., Matich, G., 2020, January. Neuromorphic engineering: taking AI to the edge. In Polaris Innovation Journal

4. Kirkland, P., Di Caterina, G., Soraghan, J. and Matich, G., 2020, July. SpikeSEG: Spiking segmentation via STDP saliency mapping. In 2020

International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE.

5. Kirkland, P., Kapitany, V., Lyons, A., Soraghan, J., Turpin, A., Faccio, D. and Di Caterina, G., 2020, September. Imaging from temporal data via spiking convolutional neural networks. In Emerging Imaging and Sensing Technologies for Security and Defence V; and Advanced Manufacturing Technologies for Micro-and Nanosystems in Security and Defence III (Vol. 11540, p. 115400J). International Society for Optics and Photonics.

6. Kirkland, P., Di Caterina, G., Soraghan, J. and Matich, G., 2020, September. Neuromorphic technologies for defence and security. In Emerging Imaging and Sensing Technologies for Security and Defence V; and Advanced Manufacturing Technologies for Micro-and Nanosystems in Security and Defence III (Vol. 11540, p. 115400V). International Society for Optics and Photonics.

7. Kirkland, P., Di Caterina, G., Soraghan, J. and Matich, G., 2020, Perception Understanding Action: Adding Understanding to the Perception Action Cycle With Spiking Segmentation. Frontiers of Neurorobotics. 14:568319. doi: 10.3389/fnbot.2020.568319

Chapter 2

Neural Networks

2.1 Introduction

Since the CNN success of 2012 [5], within the ImageNet Challenge [31] (an image classification challenge with 1000 different classes of image e.g. Dog Breeds, Cars, Birds) Neural Networks (NN) and especially CNNs have grown massively in popularity and have started a new resurgence in AI called “Deep Learning”. This surge has seen the number of publications growing year on year, while also being added to most engineering conferences as a topic. The Neural Network itself though is not a new idea with its roots based on the research of McCulloch and Pitts in the 1940s [16]. This research was based loosely on the model of the brain, using a series of connected neurons and the pathways between them to complete tasks such as classification and recognition. The learning mechanism most commonly used in DL is backpropagation. Backpropagation was derived by several researchers in the 1960s [32] before being implemented onto a computer in the 1970s [33]. However, it wasn't until 1974 that it was first applied to neural networks [34]. It was only with the computational power of Graphics Processing Units (GPUs) in the late 2000s that saw the rise of the neural network, especially the CNN [35]. With the ability to process between 4 and 70 times faster than Central Processing Unit (CPU) methods [32, 36], exploiting its massively parallel processing structure. This ability to accelerate the learning process enables NNs to outclass humans on the ImageNet challenge in 2015 [37]. Highlighting that in the fields where enough labelled data exists, and where you can train a network big enough, computers can now determine the fine differences between 120 dog breeds, as demonstrated within the ImageNet dataset. Since this amazing feat within image classification, the CNN has shown adaptability to manage other image-based tasks, with a particular focus of this thesis being segmentation. For the segmentation task one particular network architecture that of the encoder/decoder, seen in its most common and efficient form as SegNet [12]. This network was able to provide a key milestone in segmentation with no fully connected layers and pooling indices providing state of the art results in 2017 and setting a trend for years to come.

In this chapter, a review of the neural network and its multiple forms is provided. Section 2.2 covers the ANN, from neuron to training methods. Section 2.3 introduces the CNN describing the new layers introduced in this network type, including how this leads to the notion of DL seen in Section 2.4. Section 2.5 describes the research background of the encoder/decoder network architecture that leads to the popular segmentation approach, before the conclusion summaries the chapter in section 2.6.

2.2 The ANN

Researchers have long been inspired by the human brain. This inspiration leads to the idea of an artificial neural network. The idea of ANNs began as a model of how neurons in the brain function, which at the time wasn't seen in AI but more connectionism as it used connected circuits to simulate cognition or intellectual abilities. This line of thinking starts with the McCulloch-Pitts neuron [16], a simple electrical circuit by neurophysiologist Warren McCulloch and logician Walter Pitts, as seen in Figure 2.1. Figure 2.1 depicts the linear threshold gate (a.k.a McCulloch Pitts (MCP) Neuron), where the neuron takes a sum of the inputs (x_i) and then returns an output (y) where $x, y \in 0, 1$ so boolean input and outputs.

$$y = \sum_{i=1}^{n} x_i \tag{2.1}$$

This ideology was further developed by Donald Hebb, in his book "The Organization of Behaviour" [23]. In this book, he proposes that neural pathways strength over successive use. So long as there is locality and causality, in that the two neurons have a connection, and the firing of the first neuron in some part causes the firing of the second. These two concepts set the main theme of neural networks at the time: Threshold Logic and Hebbian Learning. This is also essentially the start point of SNNs which will be further explored within the Neuromorphic Engineering literature review in chapter 3.

For the ANN the next breakthrough comes from tackling the McCulloch-

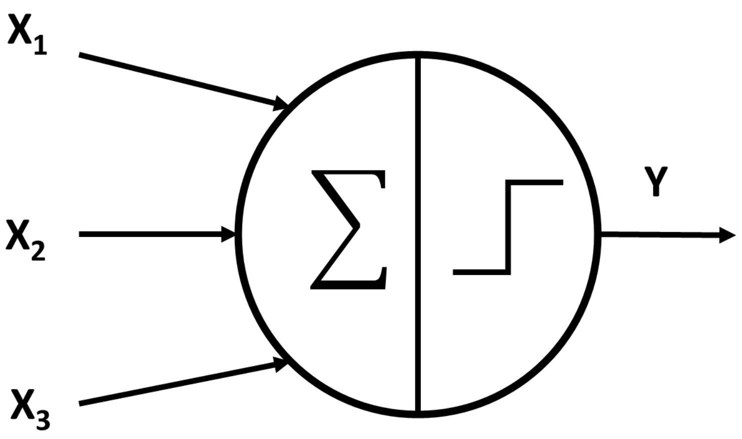


Fig. 2.1: Illustration of the McCulloch-Pitts Neuron.

Pitts neuron's biggest problems. One is a mechanism for learning, another being non-boolean inputs and finally a weighting for each input. Frank Rosenblatt in 1958 proposed the idea for the Perceptron [3] as seen in Figure 2.2. Rosenblatt implemented this idea within custom hardware to show its ability to learn to classify shapes correctly with a 20x20 (image like) input. This is essentially the first textquoteLearning within a Machine or to put it simply the dawn of Machine Learning (ML). The perceptron was able to complete this classification task by stacking multiple perceptron neurons within a layer, with the number of perceptrons equalling the number of different output classes you could have. The training makes use of the individual weights now allocated on the input connections.

$$y = \sum_{i=0}^{n} w_i x_i \text{ or } y = \sum_{i=1}^{n} w_i x_i + b \text{ where the bias is separated} \tag{2.2}$$

where the weights are w_i, inputs are x_i, the output is (y) and the bias is $b = w_0 x_0$. Describing the weighted sum of inputs which cause a 1 or 0 as the output, if above or below the threshold. The training itself follows the simple algorithm of increasing the weights if the answer was correct and decreasing if incorrect.

This perceptron model in its multi-layer format is essentially the neural network that we use today. The only real difference is the substitution of the threshold function with a variable function parameter more commonly known as an activation function. It was partially this thresholding function (and its inability to be differentiated) combined with lack of viable training methods for multilayer perceptrons that essentially lead to the perceptron being called a dead-end in

.

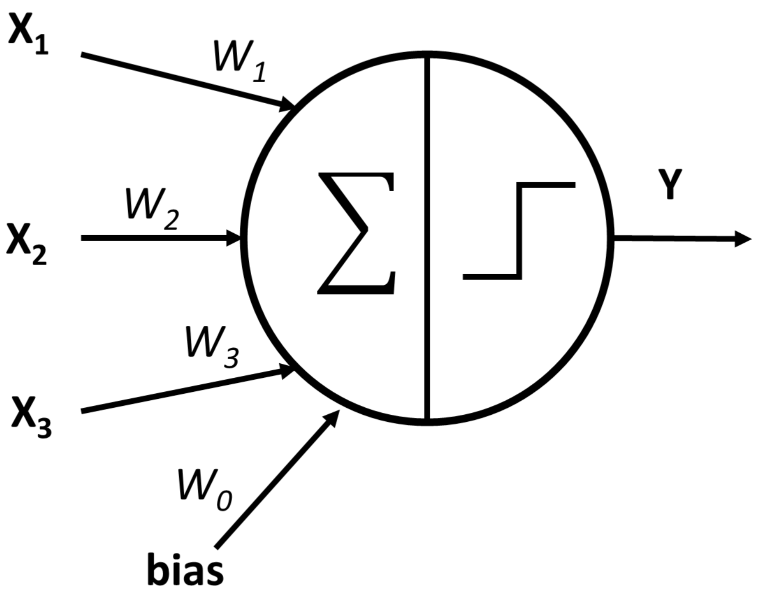


Fig. 2.2: Illustration of the Perceptron Neuron.

research [38].

It would take until 1986 for NNs to gain any real traction again with the paper "Learning representations by back-propagating errors" by Rumelhart et.al [39]. Backpropagation was the major re-discovery of a concept that was already in existence since the 1960s that helped neural nets to get out of their premature grave. Although publications had mentioned or proposed similar ideas, Paul Werbos did propose that backpropagation could be used for NNs within his PhD Thesis in 1974 [34]. Werbos did not publish the results until 1982 for several reasons, one of which being the lack of interest in the field at that current point in time. Rumelhart even cites two further publication suggestion the use of backpropagation by David Parker [40] and Yann LeCun [41]. However, it is the clear and concise manner in which the idea is stated, coupled with the recent re-ignition of funding into the subject that saw it gain in popularity. This newfound ability to train multilayer neural networks is what starts the new wave of research into what is now known as "Deep Learning" (DL). However, it was 1989 saw what is the fundamental discovery of NNs with the finding (conveniently also the paper name) that could mathematically prove "Multilayer feedforward networks are universal approximators" [42].

2.3 Convolutional Neural Networks

One method for reducing the amount of computational overhead also found success with Yann LeCun's paper "Backpropagation Applied to Handwritten Zip Code Recognition". This concept of weight sharing (the name at the time for the convolution process) allows a more computationally efficient way to process 2D inputs such as an image. This efficient processing manner was also of biological inspiration based on work studying the visual cortex. Hubel and Wiesel [17, 43] proposed a biological model for the structuring of the visual cortex over a series of papers in the 1960s. These papers introduced two distinct cell types, "S-cells" (Simple Cells) and "C-cells" (Complex Cells). These structures and the source papers went on to serve as the direct inspiration for Fukushima's new model: the Neocognitron [4]. The Fukushima Neocognitron, as Hubel and Wiesel describe, divides the neural network layers into two distinct types: the S-cell layers and the C-cell layers as seen in Figure 2.3. Each coloured box within Figure 2.3 highlights a layer of the Neocognitron with an S- and C-cell combined in each box. Each progressive layer acts as a larger receptive field for the input image, with layer 1 (green) doing edge detection. Layers 2 and 3 (orange and blue) are extracting higher-order features before layer 4 (red) performs the recognition.

This division of the layers bears a strong relation to the layers seen in the modern CNN: the convolution layer and the max-pooling layer. The input connections to the S-cell were plastic and could have their weights varied or even suppressed altogether. The S-cells acted as input to the C-cells. These connections were static and could not be varied. The work division was such that the S-cell layers were responsible for forming weights that would recognize patterns while the C-cell layers would ensure that the recognition was possible even after a shift of position or distortion in shape. Essentially the C-cells were performing a blurring of the input from the S-cell layers. Each cell-plane also contained an inhibitory cell to suppress irrelevant outputs, which is not seen in modern ANNs or CNN, but does features with SNN as will be shown in Chapter 3. It is from the research of Fukushima that influences the overall structure of the modern interpretation of the CNN starts to form. This is first seen as previously mentioned in the "Backpropagation Applied to Handwritten Zip Code Recognition" paper from Yann LeCun [44], in which information is directly extracted from images was for the first time used along with backpropagation with this weight sharing network structure. This later gets refined by LeCun into the most common form of CNN called "LeNet-5" [45] with the iconic image shown in Figure 2.4. This iconic image highlights the new ability of the NN to be able to extract its features

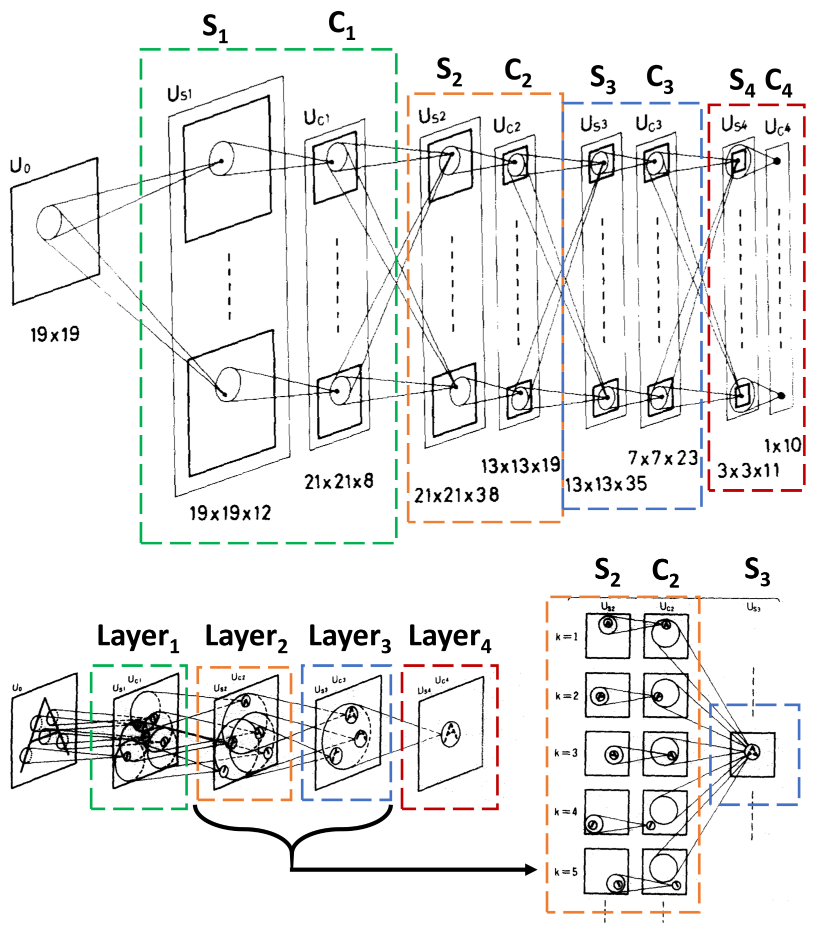


Fig. 2.3: Adaptation of an Illustration of the Neocognitron [4]. Using coloured dashed boxes to better highlight the same layer in each part of the image.

within the Convolutions and Subsampling (later renamed pooling) stages, before a more typical ANN type processing and classification in the full connection layers. The same paper from LeCun also introduces the world to the MNIST dataset a common benchmark still used to this date as a baseline for many new models and approaches.

2.3.1 Convolution Layer

Convolution is one of the main building blocks of a CNN. The term convolution refers to the mathematical combination of two functions to produce a third function. It merges two sets of information. In the case of a CNN, the convolution is often a misnomer as strictly speaking the operation is not always a convolution

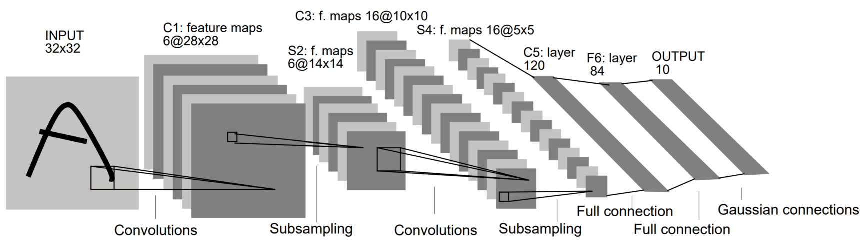


Fig. 2.4: Illustration of the LeNet-5 network [44].

but instead a correlation, due to the kernel not being flipped both horizontally and vertically before being applied. However, in practical terms the two operations would be equivalent in CNNs if the weights are initialised the same, the training would just result in flipped versions of the kernels [46]. The term convolution is then more to emphasise the link to traditional signal processing and image filtering techniques. Many machine learning libraries implement cross-correlation but call it convolution [46]. In this research, we follow this convention of calling both operations convolution and specify whether we mean to flip the kernel or not in contexts where kernel flipping is relevant. This convolutional operation is performed on the input data with the use of a kernel (or filter, these terms are used interchangeably) to then produce a feature map. We execute this step by sliding the filter over the input. At every location, matrix multiplication is performed and sums the result onto the feature map. This is illustrated in Figure 2.5, which depicts a Kernel, K passing over an input, I and the resulting output.

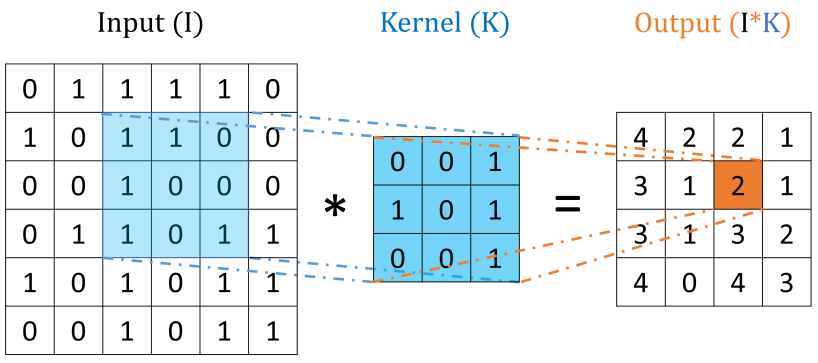


Fig. 2.5: Illustration of the convolution layer operation

More formally this expression of the convolution operation, $I * K$ as seen in Figure

2.5, for when the kernel is flipped (so actual convolution) is stated as

$$(I * K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} K_{m,n} \cdot I_{i-m,j-n} + b \tag{2.3}$$

While the non flipped variation (correlation) is

$$(I \star K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} K_{m,n} \cdot I_{i+m,j+n} + b \tag{2.4}$$

wherein both cases the filter or kernel K is of dimension $k1k2$ (however usually square so in this case $k1 = k2$) has m by n as the iterators and b is the bias. Equation 2.4 then turns into the more general

$$x^l_{ij} = \sum_m \sum_n w^l_{m,n}\, o^{l-1}_{i+m,\, j+n} + b^l \tag{2.5}$$

where x is the input with i and j as iterators. $w^l_{m,n}$ is the weight matrix of neurons from layer l with $l-1$. $o^l_{i,j}$ is the output vector at layer l given by $f(x^l_{i,j})$, where $f()$ is the activation function applied (simply swap the signs of the m and n variables to have the equivalent flipped kernel version based on Equation 2.3). Equation 2.5 can then be used to calculate the output of any convolution process at any layer or point. The Convolution layers of a CNN are used as a means of feature extraction. With multiple layers allowing non-linear combinations of these features to build hierarchically.

Typically when used in a multilayer network the standard notation for the equation is as follows (this will also be the convention used throughout this document). The output, y_{cur} is for any given neuron in the current layer, cur. Let there be $n+1$ inputs (where n = number of neurons in the previous layer, pre) with signals x_0 through x_n and weights $w_{cur,0}$ through $w_{cur,n}$. Usually, the x_0 input is assigned the value +1, which makes it a bias input with the weight of that input being the bias value, $w_{cur,0} = b_{cur}$. This leaves only n actual inputs to the neuron: from x_1 to x_n. The summed total is passed through the activation function φ

$$y_{cur} = \varphi\left(\sum_{pre=1}^{n} w_{cur,pre} x_{pre} + b_{cur}\right) \tag{2.6}$$

This equation shows that the weights are indexed with the subscript order, current layer comma previous layer. Where the layer-wise indexing will also be in reverse alphabetical order e.g. k, j, i [47]. Leading to the equation takes the form

$$y_i = \varphi\left(\sum_{j=1}^{n} w_{i,j}x_j + b_i\right) \tag{2.7}$$

2.3.2 Subsample/Pooling Layers

The subsampling, or more commonly know as the pooling layer serves multiple functions, one is to progressively reduce the spatial size of the representation (image or latent space) to reduce the number of parameters and computation in the network. At the pooling layer, forward propagation results in an $N \times N$ receptive field being reduced to a single value. This pooling of the receptive field serves as the second function, that is to add a small amount of translational invariance to the network, so small translational movements do not change the output [46]. Back-propagation through the pooling layers of the network then computes the error which is acquired by this single value. No learning takes place on the pooling layers [45]. Pooling units typically use functions like max-pooling [48] or average pooling [4]. To keep track of the pooled value its index is noted during the forward pass and used for gradient routing during backpropagation. This gradient routing is either directly assigned to where it came from (Max Pooling) or multiplied by $\frac{1}{N \times N}$ and assigned to the whole receptive field (Average Pooling) [46]. The image in Figure 2.6 shows the difference between this max and average pooling on an example input with both pooling examples labelled. In modern DL network, max-poolinghas proven the more popular option, helping to provide an amount of translation invariance, help with over-fitting and reducing computational complexity by reducing the number of parameters to learn. It must also be noted that there has also been research into omitting the pooling layers and just using a strided convolution to achieve the same effect [49]. This can often result in no loss in accuracy while reducing the number of parameters / operations required.

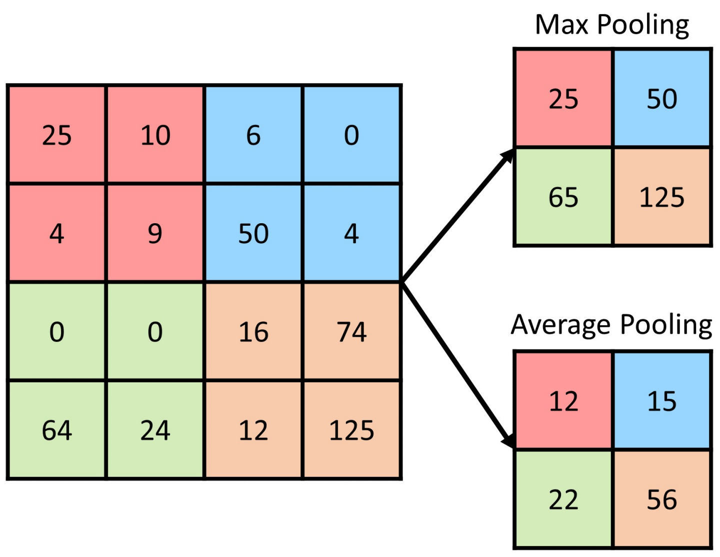
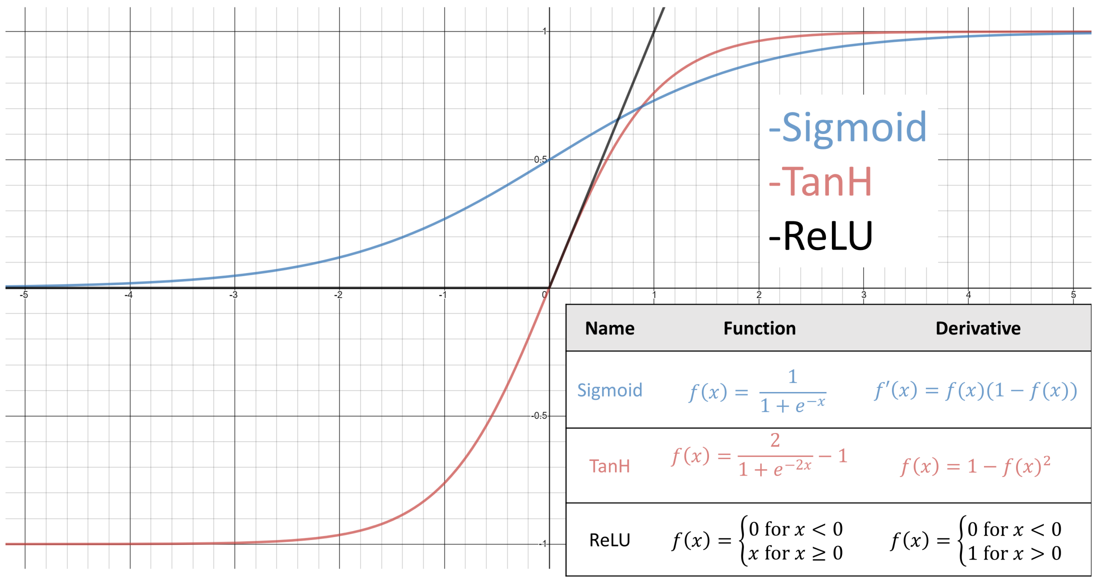


Fig. 2.6: Illustration of the pooling operation for both max and average pooling

2.4 Deep Learning

Despite the success of the CNNs and LeCun's research, NNs once again fell out of favour in the late 1990s. The deep learning phenomenon hit its tipping point in 2012 with "ImageNet Classification with deep convolutional neural networks" by Alex Krizhevsky and Ilya Sutsekever [5]. This paper as part of an entry into the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in which they almost halved the error rate of the nearest competitor. This was the standout moment of years of research and the coining of the term "Deep Learning" in 2006. This DL phase of neural networks with the deep belief network [50], helped to prove that many layers could be trained if the weights are initialised correctly. The next milestone for DL comes from the 2012 paper "Acoustic Modeling Using Deep Belief Networks", by Abdel-rahman Mohamed and George E. Dahl [51] which builds upon much of the work used in the prior Deep Belief Networks research to achieve a new state of the art within the TIMIT Acoustic-Phonetic Continuous Speech Corpus, a telephone speech recognition task. However, it was not an algorithm change or initialisation of weights, or even the state of the art results that made this paper stand out. Instead, they tackled the issue that DL had, that had been somewhat overlooked: pure computational power. To learn the millions of weights typical in deep models, the limitations of weak CPU parallelism had been highlighted. Instead, Mohamed and Dahl replaced the CPU with the massively parallel computing power of the GPU. This key change is highlighted in another paper around the time [36] suggesting a speed increase of up to 70 times. But processing power alone was unable to force NNs into the spotlight. Instead, two further algorithm changes help to form the modern interpretation of DL.



Name	Function	Derivative
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH	$f(x) = \frac{2}{1+e^{-2x}} - 1$	$f(x) = 1 - f(x)^2$
ReLU	$f(x) = \begin{cases} 0 \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$	$f(x) = \begin{cases} 0 \text{ for } x < 0 \\ 1 \text{ for } x > 0 \end{cases}$

Fig. 2.7: Comparison of activation functions used within CNNs

The first being the Rectified Linear Unit (ReLU) activation function, as shown in Figure 2.7 alongside other common activation functions of NNs. Three separate papers [52–54], from the leading names in NNs: LeCun, Hinton and Bengio all concluding that the ReLU activation function seemed to have the best performance. A considerable change from the previous sigmoid function used. With the ReLU representing a sparse representation for the derivative, that being 0 or 1. Not only helped the vanishing gradient problem of back-propagation [55], but also provides a simpler function to compute: so less computational overhead.

The second algorithmic change is the introduction of "Dropout" [56]. Dropout was initially set out to prevent over-fitting during training. However, this method of randomly selected some neurons to "switch off" during the training process, also helps to make the network more diverse. The approach mimics the essence of ensemble learning, that being learning many different ways to solve the same problem then combine them. This however would typically be done by training multiple instances of the neural network then using all the network together to solve the task. This is an impractical way due to the computational expenses to train a single network. Instead, by randomly not allowing the network to use certain neurons it essentially forces the training process to learn similar representations through multiple means. Current DL implementation has since stopped using dropout in favour of other regularisation techniques such as batch normalisation. As network architecture moved away from dense connectivity the need for dropout also fades.

The aforementioned tipping point is a combination of not only the ReLU activations and dropout, but a combination of years of research and fine-tuning. Bringing together a CNN with convolution and pooling layers, along with back-propagation and efficient GPU implementation. Not to mention the sheer amount of data and processing power now at the disposal of researchers. Together they combine to create a network called Alex-Net [5] and a paper that has now been cited almost 70,000 times in just 8 years. This number just exemplifies the impact this paper had, but the runaway success that DL has become. Since 2012 DL has gone on to be the state of the art in numerous fields. One particular example is computer vision, where DL has changed the way research is carried out, as illustrated in Figure 2.8. This figure shows how NNs went from one of many classification options to the ML community, to solve the whole problem space. Removing the need for expertly hand-crafted features that serve as an input to the classifier to learn. DL allows back-propagation to help solve the classification and feature extraction problem simultaneously. Removing the need for hand-crafted features then classification (in which ANNs could be used). This is in essence the change that enabled huge strides in image classification [5,6], object detection [7,8] and especially segmentation [11–14, 57].

2.4.1 ReLU

To fully utilise the backpropagation of errors to train deep neural networks, an activation function that possesses both properties of linear and non-linear function is required. With ReLU the activation function is essentially two linear relationships joined at the y-intercept, as was seen in Figure 2.7. This property allows it to built the non-linear relationships in data required in DL, while also providing a series of benefits compared to the other standard activation functions such as Sigmoid and TanH. As such, it is important to take a moment to review some of the benefits of the approach, first highlighted by Xavier Glorot et al. [54] in their milestone 2011 paper on using ReLU titled “Deep Sparse Rectifier Neural Networks”.
The other benefits brought about from ReLU are:

- Computational Simplicity - Easy computational implementation thanks

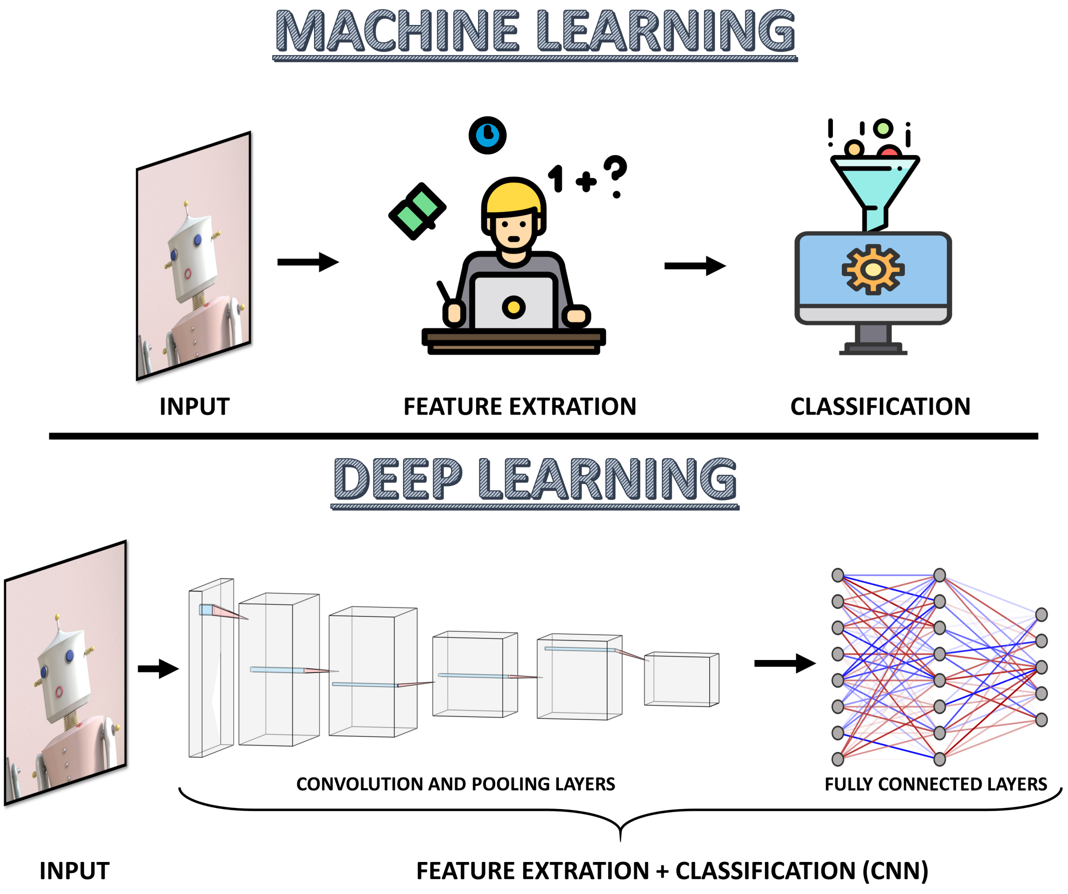


Fig. 2.8: Image highlighting the changes in how a task like image classification is completed, from the traditional ML approach to the new found DL approach

to the use of a max() function. Unlike sigmoid and tanh that require exponential calculations.

- Representational Sparsity - The output of the function can be zero, unlike the sigmoid and tanh counterparts. As both these other function just approximate a zero output, while anything negative with ReLU is simply zero. This is known as a sparse representation and is a very desirable property in hardware when doing representational learning as it can accelerate learning and simplify the model.

- Linear Behaviour - As previously mentioned the ReLU is just two linear activations joined at the y-intercept. Typically, a NN is easier to optimise when its behaviour appears to be linear. The key to this linear behaviour is that networks trained with this activation function almost completely avoid the problem of vanishing gradients, as the gradients as fixed to 1 or 0.

- Train Deep Networks - This implementation and adoption of the ReLU activation function meant that it became possible to exploit improvements in hardware. This then allows successful training of deep multi-layered networks with a non-linear activation function using back-propagation. Removing the need for cumbersome training schemes such as layer-wise training and unlabeled pre-training.

2.5 Segmentation

Once CNNs had established their dominance in the image classification realm, research then focused on other tasks they could aim to solve. This brings about the Fully Convolutional Networks (FCNs), who owe their name to their architecture [11], which is built only from locally connected convolution, pooling and upsampling layers. This architecture removes the fully connected dense layers seen at the end of a CNN used for classification. This helps to reduce the number of parameters and computational overhead. Also, going fully convolutional allows the network to work regardless of the original image size, due to not requiring any fixed number of units at any stage, given that all the network connections are local.

This network arises from the need to solve a particular CV task, that of segmentation. In CV terms segmentation is a more challenging task than Image Classification (Classify the object (Recognize the object class) within an image) or Object Detection (Classify and detect the object(s) within an image with bounding box(es) bounded the object(s)). With the insight of the FCN approach being to take advantage of these existing CNNs as powerful visual models that can learn hierarchies of features.

Segmentation builds on the object detection solution where there is the need to know the class, position and size of each object. Segmentation comes in three main variants Semantic, Instance and Panoptic. Semantic Segmentation: Classify the object class for each pixel within an image. That means there is a label for each pixel. Instance Segmentation: Classify the prediction of object instances and their per-pixel segmentation mask. That means there are class and object labels for each pixel of known objects. Panoptic Segmentation: This is essentially the combination of semantic and instance segmentation. That means there is now a class and object label for every pixel. The image in Figure 2.9 shows an example of the difference between semantic, instance and panoptic segmentation.

To obtain a segmentation map (output), segmentation networks usually have 2 parts: downsampling to capture semantic/contextual information (typically seen

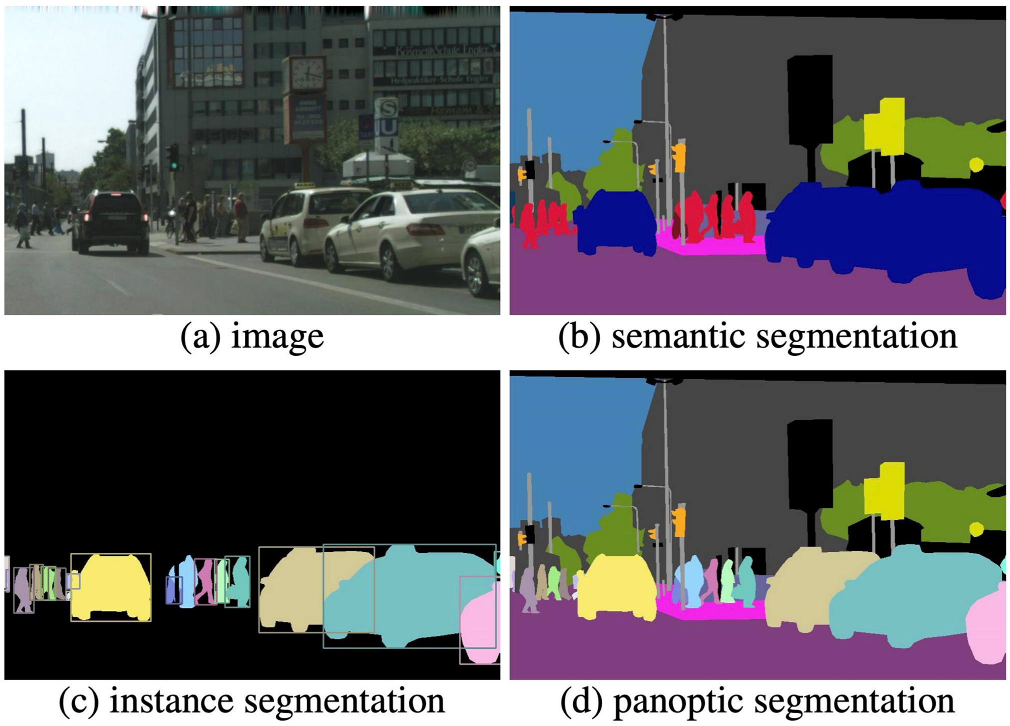

(a) image
(b) semantic segmentation
(c) instance segmentation
(d) panoptic segmentation

Fig. 2.9: An example of the different types of segmentation for a given image (a). : (b) semantic segmentation (per-pixel class labels), (c) instance segmentation (per-object mask class label), and (d) panoptic segmentation (per-pixel class+instance labels) [58]

as encoding) and upsampling to recover spatial information (typically seen as decoding). The encoding path is used to extract and interpret the context of what is in the image. While the decoding path is used to enable precise localization as to where in the image something is.

The first milestone for FCN comes from the paper aptly named "Fully Convolutional Networks for Semantic Segmentation" by Long et al [11]. Which applies the FCN to the task of semantic segmentation. They showed that by transforming the existing and well-known classification models like AlexNet [5] (and the more modern variations that occurred between 2012 and 2015, VGG [59], GoogLeNet [60], and ResNet [6]) into fully convolutional networks. That is, by replacing the fully connected layers with convolutional ones, they could output spatial maps (heatmaps) instead of classification scores. Figure 2.10 helps to illustrate this process with a collection of images from their paper [11, 57]. By transforming the fully connected layers: (a) into convolutional ones, (b) the network changes from classification network to outputting spatial maps. Then once this is combined with interpolation layers and a spatial loss (c) it can produce an efficient end-to-end pixel-wise learning network.

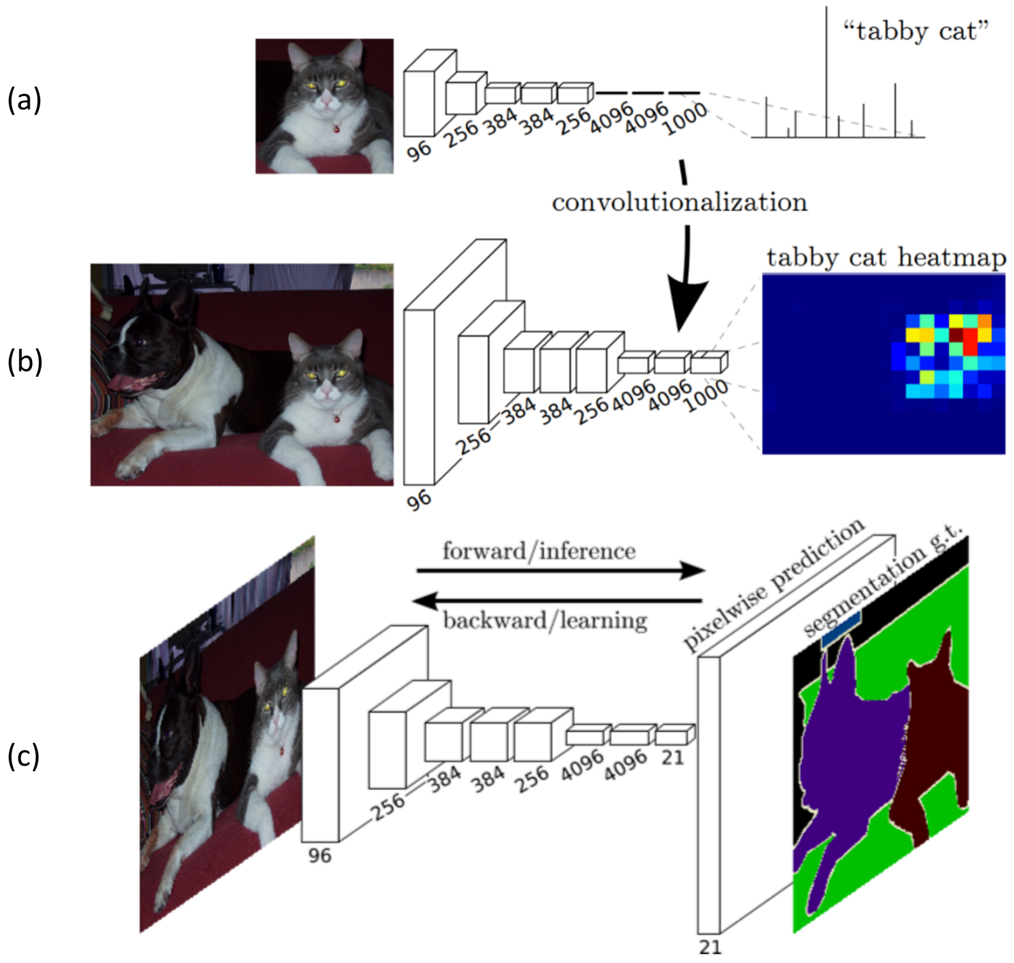


Fig. 2.10: Breakdown of the fully convolutional network: (a) only convolutions for classification (b) using the convolutions to give feature heatmap (c) full system with a pixel-wise prediction for segmentation. [11, 57]

The spatial maps (or heatmaps as shown in Figure 2.10) are upsampled using a process called fractionally strided convolution (originally called deconvolution [61], but now commonly know as transposed convolution) to produce dense pixel-based class labelled outputs. This work is considered a milestone, although it didn't introduce any novel ideas. It instead is another paper that helped to showcase and highlight a combination of ideas clearly and concisely while producing better than the state of the art results with less computational overhead. The key finding was to highlight that FCNs can be trained end-to-end for this problem, efficiently learning how to make dense predictions for semantic segmentation with inputs of arbitrary sizes.

The issue with the FCN approach used in Long et al [11] is that by propagating through several alternated convolution and pooling layers, the resolution of the output feature maps is lower resolution, which results in rough object boundaries. To combat this issue a variety of FCN-based approaches propose alternative solutions to the upsampling (encoding) stage. Eigen et al. [62] proposed a network to progressively refine the lower resolution prediction through a series of scaled sub-convolutional networks. This work also progressed the idea to solve depth prediction, surface normal estimation along with semantic labelling. Noh et al. [63] purpose a network entitled DeconvNet, which uses the same encoding structure as Long et al [11]. The decoding structure, however, is now a deep transposed convolutional network itself that is mirrored to the encoding. To do this the transposed convolutional network uses both the transposed convolutions [61, 64] and unpooling [64, 65] processes. DeconvNet works by reposing the semantic segmentation as an instance-wise segmentation problem. Where the top 50 out of 2000 region proposals (object bounding boxes), are detected by an object detection approach, EdgeBox [66]. Then, DeconvNet [63] is applied to each proposal where aggregation of the output proposals is mapped back to the original image. Utilising this instance wise approach allows various scales can be handled more effectively, producing state of the art results. However, the architecture of DeconvNet is called fully convolutional it does utilise 1x1 convolutions (so essentially a fully connected layer) at the intersection between the down and upsampling stages.

Removing these densely connected layers in the middle of DeconvNet is what gives SegNet [12] its large computational performance gain. Using an equivalent network (VGG16) without the 1x1 convolutions results in one-tenth of the parameters being used. Segnet also introduces a pixel-wise softmax at the last layer, which allows an easier end to end training regime. The difference between the two networks DeconvNet and SegNet are shown in Figure 2.11 (a) and (b) respectively. As is also shown in Figure 2.11 (b), SegNet removes the transposed convolutional layers using the pooling indices instead to upsample, then using convolution layers to rebuild the features of the image in the decoder stage.

SegNet is often associated with the popularisation of the use of pooling indices within the newly named convolutional encoder-decoder networks, replacing the name FCN. However, the original case seems to stem from Ranzato et al [67] called switch upsampling, then further implementations by Zeiler et al. [64, 65] and use within DeconvNet [63]. This in part could be down to the popularity differences between the papers, with SegNet accumulating almost twice as many citations as all the others just mentioned combined. It is also interesting to note that the first iteration of SegNet was submitted to the Conference on Computer Vision and Pattern Recognition (CVPR) 2015. The same conference as the original

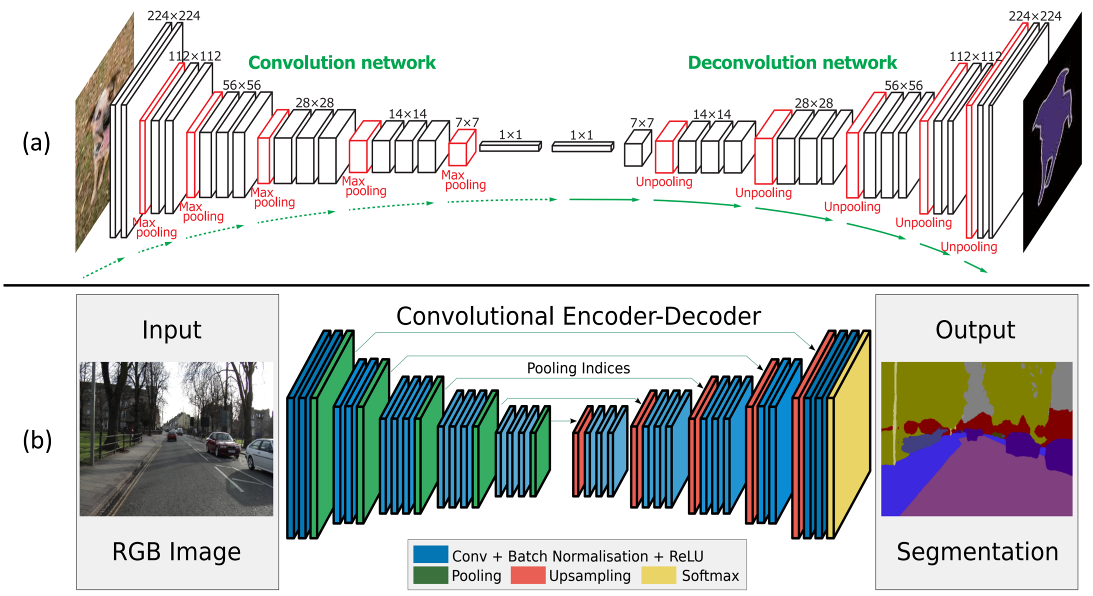


Fig. 2.11: An example the two segmentation networks (a) Deconvnet [63] and (b) SegNet [12].

FCN paper [11], suggesting that SegNet as seen in the Journal Transactions on Pattern Analysis and Machine Intelligence in 2017 [12] is not a progression of the FCN network but instead a parallel research effort. However, the original SegNet-2015 [68] is quite different from the 2017 version [12], as the original uses a stacked encoder-decoder that is trained similar to an auto-encoder in a layer-wise manner. With the 2017 version seemingly benefiting from hindsight and the popularisation of fully convolutional deep networks such as DeconvNet. The efficiency of SegNet's approach to decoding using max-pooling indices is compared with the decoding of FCN within their paper [12] and shown within Figure 2.12. The image highlights the reduction in computation required to do the decoding (upsampling) process.

Another popular alternative to the idea of pooling indices comes within U-Net. The U-Net was developed by Olaf Ronneberger et al. [13] for Bio-Medical Image Segmentation. However, it can be successfully applied to many other segmentation tasks and remains one of the most popular segmentation networks, mostly due to its successful implementation in biomedical imaging. Instead of using pooling indices U-Net shares the features maps between the encoding and decoding sides of the network, as illustrated in Figure 2.13.

This copy and crop connection between the encoding and decoding would later go on to be known as long skip connections. These connections allow half of the feature maps from the encoding layer to mix with the decoded upsampled ones, by

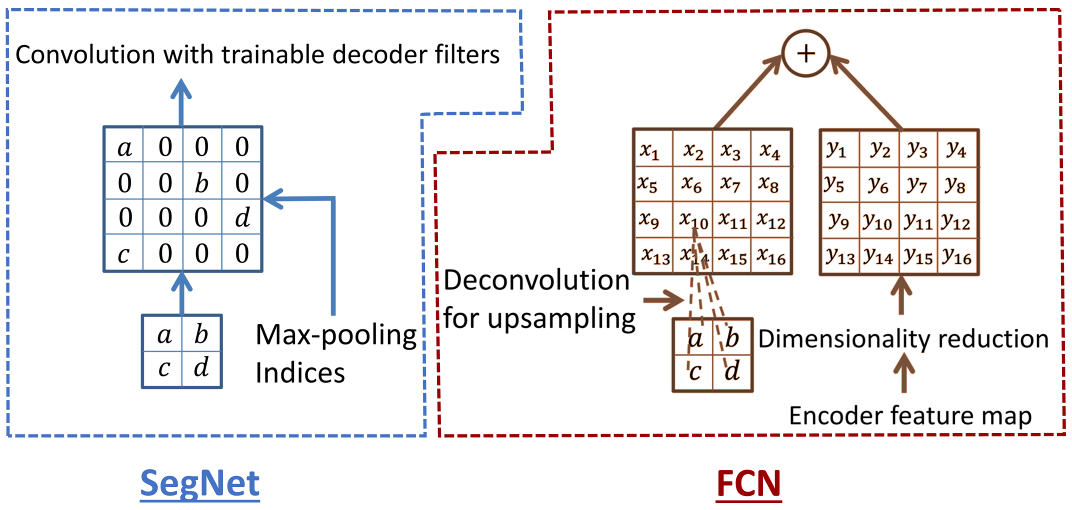

Convolution with trainable decoder filters
a 0 0 0
0 0 b 0
0 0 0 d
c 0 0 0
a b
c d
Max-pooling Indices
SegNet
Deconvolution for upsampling
x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 y11 y12 y13 y14 y15 y16
+
Dimensionality reduction
Encoder feature map
FCN

Fig. 2.12: An example of the two decoding methods of SegNet (as labelled) and FCN.

concatenating the two together. This allows fine-grained details to be recovered in the prediction that might otherwise be lost in the 'up-convolution' (transposed convolutional) stage. This helps to give the localization information from the encoding path to the decoding path. This network utilises the deconvolution method to undo the max pooling. The skip connections also allow a convenient path for the backpropagation of errors into the early convolution layers. This is seen within the UNet within Figure 2.13, that to reach the first convolution layer the error would have needed to backpropagate through 22 other layers. Meanwhile, the skip connections of U-Net would allow part of the error to propagate back through only 4 layers, avoiding the vanishing gradient problem often associated with deeper networks. However this was not a direct discovery from this research but actually, an insight gained later through further investigation [69] and the popularisation of networks that used skip connections like ResNet [6].

2.5.1 Upsampling

Typical upsampling techniques used two main methods, or unlearned and learned parameters. The unlearned parameters techniques commonly seen are nearest neighbour, bi-linear interpolation and zero-padding, which are all more traditional techniques, not specific to NNs. An example of these methods is shown in the top half of Figure 2.14. The other technique shown in Figure 2.14 is that previously mentioned in SegNet [12] of unpooling. In which pooling indices are stored from

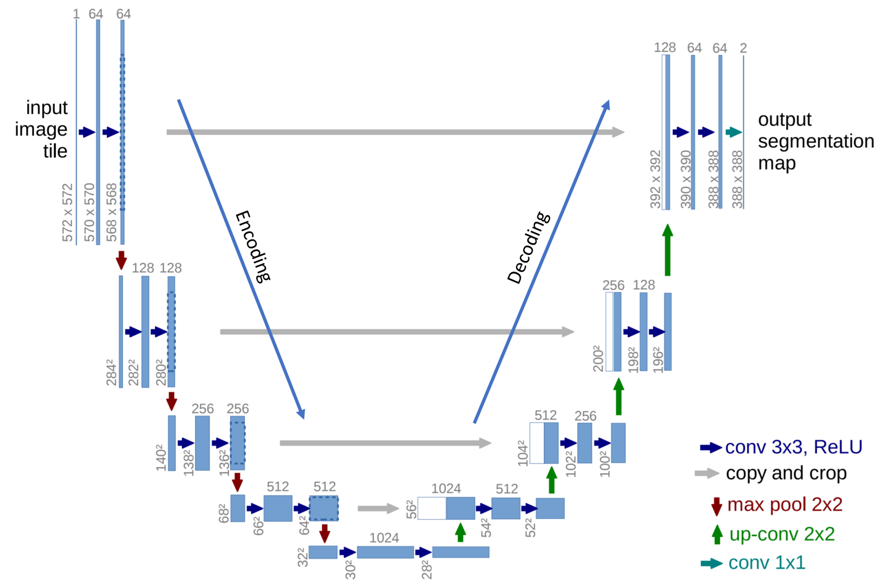


Fig. 2.13: An example of the U-Net architecture, highlighting the connections between the encoding and decoding layers [13].

the decoding layer and used to upsample the encoding layer as shown. In Nearest Neighbours, as the name suggests, an input pixel value is taken and copied to the K-Nearest Neighbours where K depends on the expected output. In Bi-Linear Interpolation, the 4 nearest pixel values of the input pixel are taken. To which a weighted average based on the distance of the four nearest cells is processed, smoothing the output. In Zero Padding, we copy the value of the input pixel at the corresponding position in the output image and filling zeros in the remaining positions.

The learned parameters technique is that of transposed convolutions, the same technique mentioned previously used within FCN and DeconvNet. Transposed Convolutions are used to upsample the input feature map to a desired output feature map using some learnable parameters (the weights of the kernel). This is illustrated in Figure 2.15, where the 2x2 input is convolved with the kernel in a transposed manner. This results in the 4 intermediate outputs to be produced which are then summed to form the larger 3x3 output of the process. As can be seen within the image the process is similar to that of conventional convolution, hence the multiple naming conventions. The difference between this method and the unlearned parameters is that the kernel can go through the same weight learning process as regular convolution kernels with back-propagation.

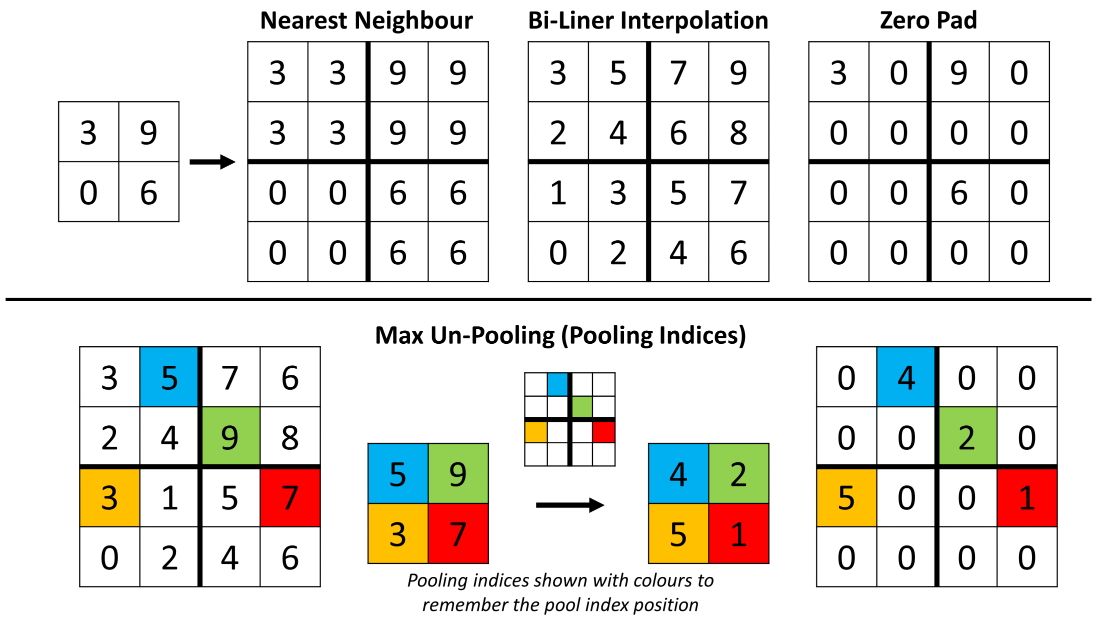


Fig. 2.14: An example of the upsampling methods. Three traditional methods; Nearest Neighbour, Bi-Linear Interpolation and Zero Pad

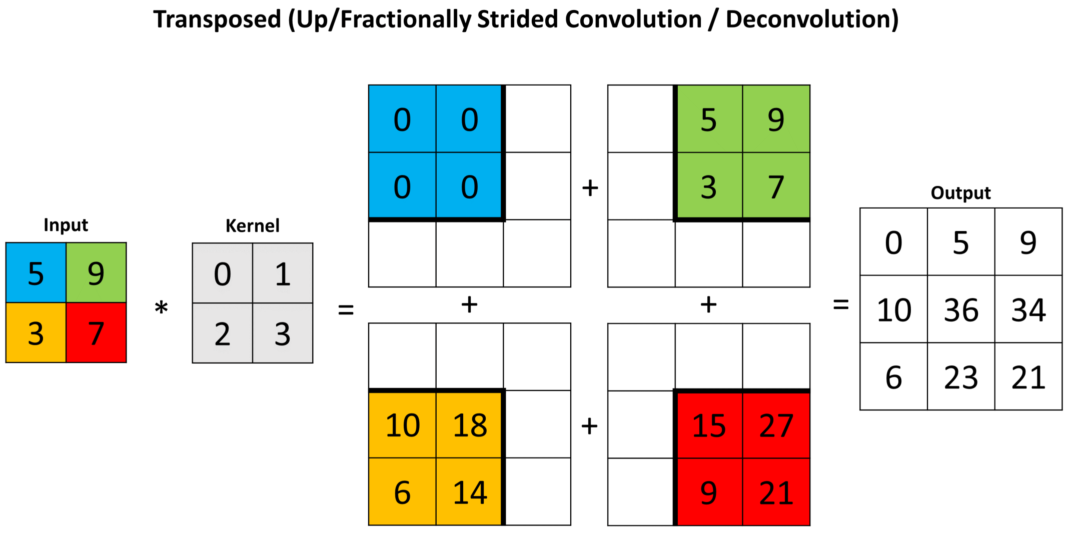


Fig. 2.15: An example of the U-Net architecture, highlighting the connections between the encoding and decoding layers.

2.6 Conclusion

This chapter serves as a review and historical background as to how CNNs came to be. From biological inspiration with the perceptron, through the age of deep learning. Highlighting the challenges that the ANN had to go through to become successful. This can help to put into context the task in which SNNs have to do now with NM engineering, as discussed in Chapter 3. The review also allows an insight into an architecture type used throughout all three contribution chapters of this thesis, that being the encoder-decoder network. These networks also help to popularise new upsampling algorithms such as the pooling indices and transposed convolution which feature within the research contributions of this thesis. This review also highlights a meta element of the research history of NNs. That is many of the algorithmic developments do not align with the milestones. Instead, the milestones are research paper in which the algorithm was well implemented and explained. This ethos is reflected within the contributions chapters of this thesis, where intuition, understanding and interpretability are driving factors in the research.

Chapter 3

Neuromorphic Engineering

3.1 Introduction

Much like the neural network, neuromorphic engineering stems from the implementation of neural circuits in electronics. Both fields share a similarly rich history, building on the work of the McCulloch-Pitts Neuron [16], the perceptron [3] and the Neocognitron [4] (retina simulation). Neuromorphic engineering as a field began from the research of using the Very-large-scale integration (VLSI) of transistors, with a focus on the non-linear characteristic of the transistor. This also brought about a collaboration of some of the most prominent scientists of the time, Max Delbrück, John Hopfield, Carver Mead, and Richard Feynman [70]. This leads to Carver Mead defining the term "Neuromorphic Engineering" to describe a new field of engineering whose design principles and architecture are biologically inspired [21]. This inspiration of the retina and its graded synaptic transmission property, similar to that of an analogue transistor, leads to the analogue neuromorphic circuits mimicking the protein channels in neurons and the world's first neural-inspired chips including an artificial retina and cochlea [21].

The desired goal of Neuromorphic engineering is to uncover the question, "how does our brain compute?". In an attempt to answer this question, one naturally has to consider the biological nervous systems. However, to understand this there appear to be two approaches, to mimic or not to mimic. The attempt to mimic, with a neuromorphic approach, include Boahen's neuromorphic circuit at Stanford University and their Braindrop (formerly Neurogrid) processor [71], the mathematical spiking neuron model of Izhikevich [72] as well as the large scale modelling of Eliasmith [73]. All of these approaches also make use of the spiking variant of the NN, SNNs due to the spiking neurons having closer links to biology plausible systems. The approach that abandons mimicry of the biological systems, instead of taking loose inspiration, to algorithmically solve problems is that of the Neural Network approaches found in Chapter 2. There are however research endeavours that fit in-between due to the abundance of digital computer structures and the rise in popularity and success of deep learning. These methods help to bridge the gap between the biologically plausible (NM engineering) and implausible (Deep Learning) domains. Examples such as conversion from NN to SNN, [74–78] and utilising DL and NNs for Neuromorphic Sensors for applications in classification [79, 80], image recognition [81, 82] and optical flow [83, 84] highlight the popularity of research in this middle ground. It also allows certain characteristic of NM engineering like low power to be exploited with the training ease of back-propagation from DL.

In this chapter, a review of the biologically inspired elements of NM engineering is provided, from neuron types in section 3.2 to neural coding algorithms 3.3. This is followed by an overview of SNN learning algorithms in section 3.4 a review of current neuromorphic resources available, hardware and software in sections 3.5. This is followed by a section covering the perception action cycle in section 3.6, then a conclusion to summarise the literature review in section 3.7.

3.2 Spiking Neuron Models

The fundamental processing units in the central nervous system are neurons. The neuron receives one or more inputs and combines them to produce an output. This was previously discussed in Chapter 2 with the ANN and perceptron. However, for neuromorphic engineering a different type of neuron is used, a spiking neuron.

The spiking neuron-like its artificial counterpart contains information regarding the neuronal and synaptic states. Additionally, the spiking neuron also incorporates the concept of time into the neuron model. As such the neuron no longer needs to adhere to the synchronous propagation cycle (as seen in the ANN), instead of propagating information - neuron firing a spike - when the membrane potential reaches its desired threshold. This then enables an asynchronous mode of processing and communication to be achieved by the SNN. Furthermore, the spiking neuron allows the expensive memory access operations of ANNs to be negated due to computations and memory being exclusively local [85], again inspired by biology.

Of the spiking neurons model available the three most common represent the full spectrum of approaches at hand. These range from the low computational overhead, but low biological plausibility/meaningfulness of the Integrate and Fire

(IF) neuron model [86]. All the way upto the highly complex Hodgkin-Huxley (HH) neuron model [87], one of the most important models in computational neuroscience and has a high level of biological plausibility/meaningfulness but at the cost of a high computational burden. The final neuron model is that of the Izhikevich (IZ) neuron model [72], which was designed to permit the majority of biological features of the Hodgkin-Huxley neuron, but at a computation cost closer to the IF neuron. This spectrum of possibilities was illustrated by Izhikevich himself within the following Figure 3.1 [72]. Figure 3.1 helps to highlight the range of neuron performance parameters and where the three mentioned neuron models lie within this, where IF and their leaky (LIF) counterpart are computationally are the same. The HH neuron due to its complexity is then rarely used, as the majority of models that require that level of complexity of neuron can make use of the IZ neuron. Though most typically in neuromorphic engineering application it is often the IF and LIF neurons that are utilised. This is in part due to being less than half the computational complexity of an IZ neuron, but also importantly just having fewer hyper-parameters to optimise for. It has already been shown with the field of DL, that a simple neuron such as the perceptron can be used to create a highly complex learning system. Often the extra hyper-parameters just come at the cost of extra complexity when training, a task that is not well solved within SNNs.

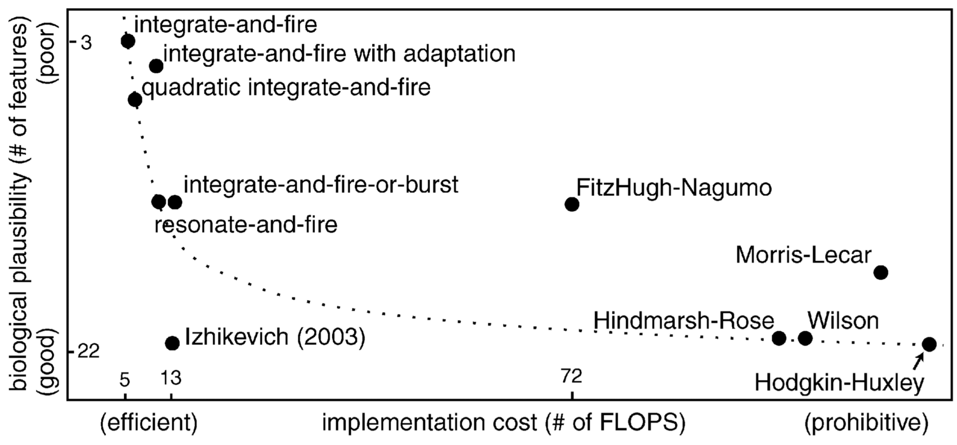


Fig. 3.1: Illustration of a graph that shows the computational complexity, in terms of floating-point operations versus the biologically plausibility through the number of features present in biological neurons. [72]

3.2.1 Integrate-and-Fire (IF) neuron model

The IF neuron represents one of the earliest models of nervous system stimulation, used to compare experimental data collected from a frog in 1907 by Louis Lapicque [86]. Lapicque states that neuron is represented in time by

$$I_m(t) = C_m \frac{dV_m(t)}{dt} \tag{3.1}$$

where I is the current flowing through the cell membrane, C is the membrane capacitance and V is the membrane voltage. Equation 3.1, which is just the derivative of the capacitance formula, $Q = CV$, with respect to time. This indicates that with a current flowing through the cell membrane, the voltage will increase until such time it reaches its threshold, V_{th} at which point a spike is fired and the membrane voltage is reset. To stop the model from increasing its firing frequency linearly with the input a refractory period t_r is also set for the neurons. This lead to the firing frequency being calculated as

$$f(I) = \frac{I}{C_m V_{th} + I_m t_r} \tag{3.2}$$

However, a shortcoming of this capacitance only model of the neuron means it has no time-dependent memory. For each sub-threshold increase in the membrane potential, the potential will remain at that point until the membrane voltage is increase past the threshold at which it resets. Although this shortcoming can be overcome in implementation, due to the IF model representing the least computationally complex version, resulting in it still being a popular method. If the implementation of the IF neuron is paired with a synchronous digital input, which could be a buffer to collect spiking activity (as the processing utilises time multiplexing) or if the input is frame-based. Then this natural synchronisation point is used to reset the neurons, so long as enough time is given at the end of the processing for all the spiking activity to complete.

Typically when using this neuron type within an SNN structure, it takes on a similar formulation to the standard NN model and perceptron model seen in Chapter 2 with equations 2.2, 2.6, 2.7. However, the formulation has now expanded to take into context the temporal nature of the neuron and the spike element of the input. Essentially the weighted sum of the spiking inputs to that neuron from the previous time step plus the membrane potential from the previous time step [88].

$$V_i(t) = V_i(t-1) + \sum_j w_{j,i} S_j(t-1) \tag{3.3}$$

where V_i is the internal potential of the ith neuron at time step t, $W_{j,i}$ is the synaptic weight between the jth presynaptic (previous layer) neuron and the ith neuron, and S_j is the spike train of the jth presynaptic neuron. Where $S_j(t-1) = 1$ if the neuron has fired at the previous time step, other it is 0. Neurons will send a spike if the potential V_i exceeds its threshold, V_{thr}, after which V_i is reset such that

$$\text{If } V_i(t) \geq V_{thr}, \text{ then } V_i(t) = 0 \text{ and } S_i(t) = 1 \tag{3.4}$$

Leaky Integrate-and-Fire(LIF) neuron model

To address some of the previously mentioned shortcomings of time-dependent memory Lapicque [86] also developed the LIF neuron model. A comparison of the two model types is visualised in Figure 3.2, which shows the two neuron models as electronic circuit diagrams. This diagram represents the neuron cell and how it operates under the simplifications of the IF and LIF models.

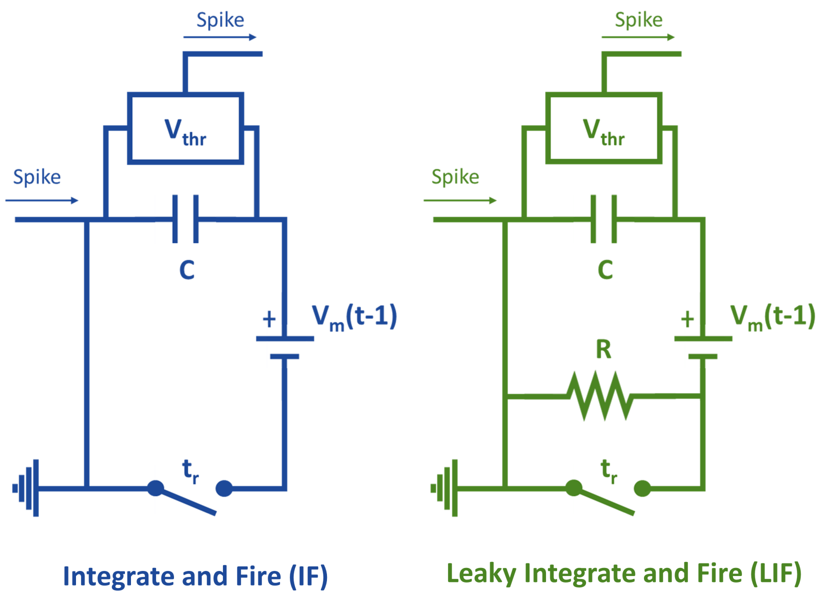


Fig. 3.2: Illustration of the circuits for the IF and LIF neuron model.

From Figure 3.2 the addition of a resistor is seen within the LIF neuron model allowing the leakage of voltage from the capacitor. This additional component then furthers the previous equation 3.1, to include the change in voltage $V_m(t)$ over the resistor R_m seen as

$$I_m(t) = C_m \frac{dV_m(t)}{dt} + \frac{V_m(t)}{R_m} \tag{3.5}$$

This modification also changes the standard NN formulation with the addition of the λ component to represent the leakage [89].

$$V_i(t) = V_i(t-1) + \sum_j w_{j,i} S_j(t-1) - \lambda \tag{3.6}$$

To better visualise the differences this makes within an implementation, an illustration of the spiking response of the different models is shown in Figure 3.3. Where both the membrane potential of the IF and LIF neuron are shown with the same spiking input response.

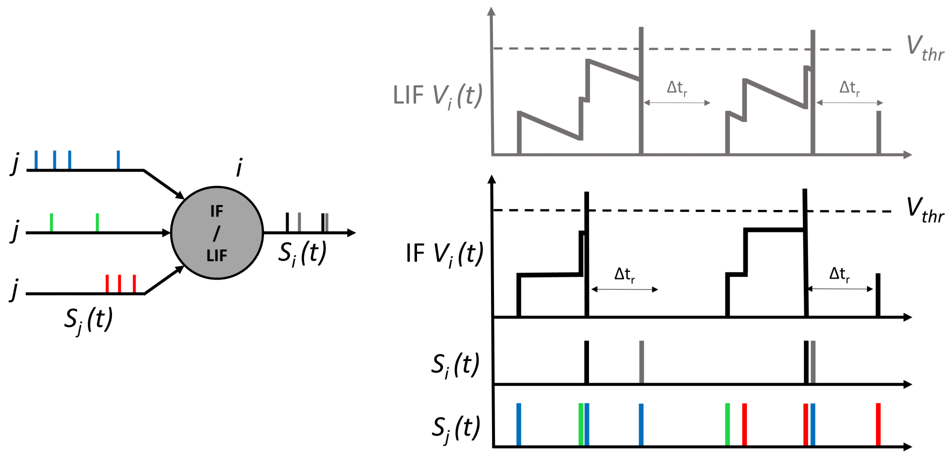


Fig. 3.3: Spiking response $s_i(t)$ and membrane potential $v_i(t)$ of both the IF and LIF neuron model to the same input stimulus $s_j(t)$.

Figure 3.3 depicts when a spike inputs the neuron ($S_j(t) = 1$), the synaptic weight w_{ij} associated with this spike will be integrated on the membrane (in this example all weights are not shown and are the same for ease of illustration). If the membrane potential V_i becomes greater than a threshold V_{thr}, the neuron fires ($S_i(t) = 1$) and resets its membrane potential. However, if the refractory time t_r is not reached (the amount of time since the last output spike is smaller than t_r), the neuron does not fire, regardless of the membrane potential being above the threshold. Also, Figure 3.3 highlights the different membrane potential responses between the IF and LIF neurons. Where the LIF neuron displays leakage between input spikes, decreasing the membrane potential over time. Meanwhile, the IF neuron potential remains constant until the next stimulus.

Both the IF and LIF neurons are displayed together in this section due to the commonality of the term "Integrate and Fire" being used to describe the LIF neuron [90]. This is in part due to the popularity of the LIF neuron model, its use in neuromorphic hardware [91–94] and the overall similarity between the two methods.

3.2.2 Hodgkin-Huxley neuron model

The Hodgkin-Huxley (HH) neuron model [87] presents itself as one of the most biophysically meaning models of a neuron. This is in stark contrast to the previously described IF and LIF neurons, whose biophysiological properties were traded out for computation simplicity. The HH neuron was designed from an experimental study of the squid giant axon, partly due to the ability to insert a micropipette into the axon and perform voltage-clamp experiments, a technique that had been devised in the 1930s by Cole and Curtis [95]. This method allowed insight into the response of the neuron from external current stimulation while allowing to measure the effects across the different ionic channel and leakage currents. The Hodgkin-Huxley model can be understood with the help of the electrical circuit diagram of the cell membrane as shown in Figure 3.4

The membrane of the cell acts as a capacitor and separates the interior of the cell from the extracellular liquid. When an external stimulus current I(t) is injected into the cell, it either acts to charge the capacitor, C_m, or leak through the channels in the cell membrane. Each channel type is represented in Figure 3.4 by a variable resistor. The sodium and potassium channels are represented by R_{Na} and R_k respectively, while the remaining channel represents any leakage in the cell, R_L. Because of active ion transport through the cell membrane, the ion concentration inside the cell is different from that in the extracellular liquid. The Nernst potential generated by the difference in ion concentration (due to the channels allowing this transportation) is represented by batteries in Figure 3.4. As there is a different potential across each ion type, there are separate batteries for sodium, potassium, and the leak channel, with battery voltages V_{Na}, V_K and V_L, respectively.

Understanding the circuit diagram shown in Figure 3.4 allows the derivation of the HH equation. Where first the total applied current, $I(t)$ is made up of the sum of the current, resistive $I_R(t)$ and capacitive $I_C(t)$, within the circuit.

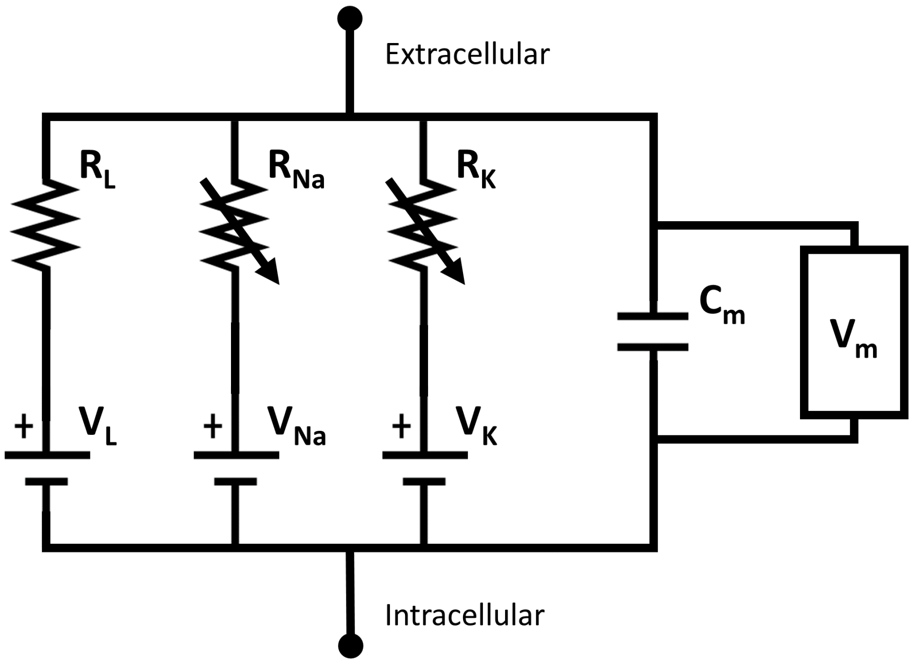


Fig. 3.4: Circuit model of the Hodgkin-Huxley Neuron.

$$I(t) = I_C(t) + \sum_R I_R(t) \tag{3.7}$$

Then through use of the capacitor definition, $C = \frac{q}{V}$, where q is the charge and V is the voltage. We can find the charging current $I_C = C\frac{dV}{dt}$ to give

$$C_m \frac{dV_C(t)}{dt} = I(t) - \sum_R I_R(t) \tag{3.8}$$

The equation leads to the breakthrough of Hodgkin and Huxley with their neuron model. They successfully measure the resistance change of a channel as a function of time and voltage. Specifically, they introduced additional gating terms m, n and h that models the probability that a channel is open at a given time. The effective conductance of sodium channels is given as $\frac{1}{R_{Na}} = g_{Na}m^3h$, where m describes the channel opening and h its blocking. The conductance of potassium is $\frac{1}{R_K} = g_K n^4$, where n describes the channel opening.

$$\sum_R I_R(t) = g_{Na}m^3h(v - V_{Na}) + g_K n^4(v - V_K) + g_L(v - V_L) \tag{3.9}$$

where the three variables m, n and h are derived from a series of differential equations [87, 90].

3.2.3 Izhikevich neuron model

The Izhikevich neuron model [72] represents an ideal compromise between a biologically meaningful neuron model and one that can be implemented without too much computational overhead. It achieves this because, unlike the Hodgkin-Huxley model, the Izhikevich model does not account for the biophysics of neurons. Instead, it uses mathematical equations to compute a wide range of neuronal spiking patterns. Importantly though the output of this model is realistic and biologically plausible. The time evolution of the membrane potential V_m is described by the differential equations:

$$I(t) = \frac{dV_m}{dt} - 0.04V_m^2 - 5V_m - 140 + u \tag{3.10}$$

$$\frac{du}{dt} = a\left(bV_m - u\right) \tag{3.11}$$

where the post spike relationship is

$$\text{If } V_m \geq 30\text{mV then } v \leftarrow c, u \leftarrow u + d \tag{3.12}$$

where u is the recovery variable, I external current input to cell. The remaining variables a, b, c and d are used to determine the spiking or bursting behaviour of the neuron. a describes the recovery time scale of u, b describes the sensitivity of u to sub-threshold fluctuations of V_m. c describes the post spiking reset value of V_m and d describes the post spiking reset value of u. Though this model is considerably easier to implement compared to the HH neuron, it still has 5 variables to set. This means there is still a large hyper-parameter space to optimise, especially compared to the IF and LIF neurons. This often leads to the IZ neuron being unused, especially in more complex NNs.

3.3 Neural Coding algorithms

SNNs utilise a binary spike train to relay information. To understand what information is being passed by the neurons, an understanding of how this binary spiking was encoded is required. However, this spiking neuron coding debate is not a solved problem in neuroscience [96–98]. There are currently a number of neural coding implementations available, typically falling into two categories, rate-based and temporal-based. A collection of the most common implementation from each of these categories is shown in Figure 3.5, with each implementation discussed in the following section. The majority of SNNs use rate coding, that is the information is being inferred from the rate at which neurons fire (how often over a given observation time). This rate encoding is typically produced through the use of a Poisson spike train, where the exact timing of the spikes is random but the overall frequency is constant. The alternative, temporal coding is gaining more popularity as some neurological systems react with such low latency rate encoding would be impossible [99]. This implementation makes use of the exact timing of spiking neuron to allow low latency processing in stark contrast to the rate encoding. Nevertheless, a combination of both implementations is used [100] where a combination of both fast reactive temporal information is then backed up with the slower rate based processing, to exploit both low latency and contextual understanding that is given over time.

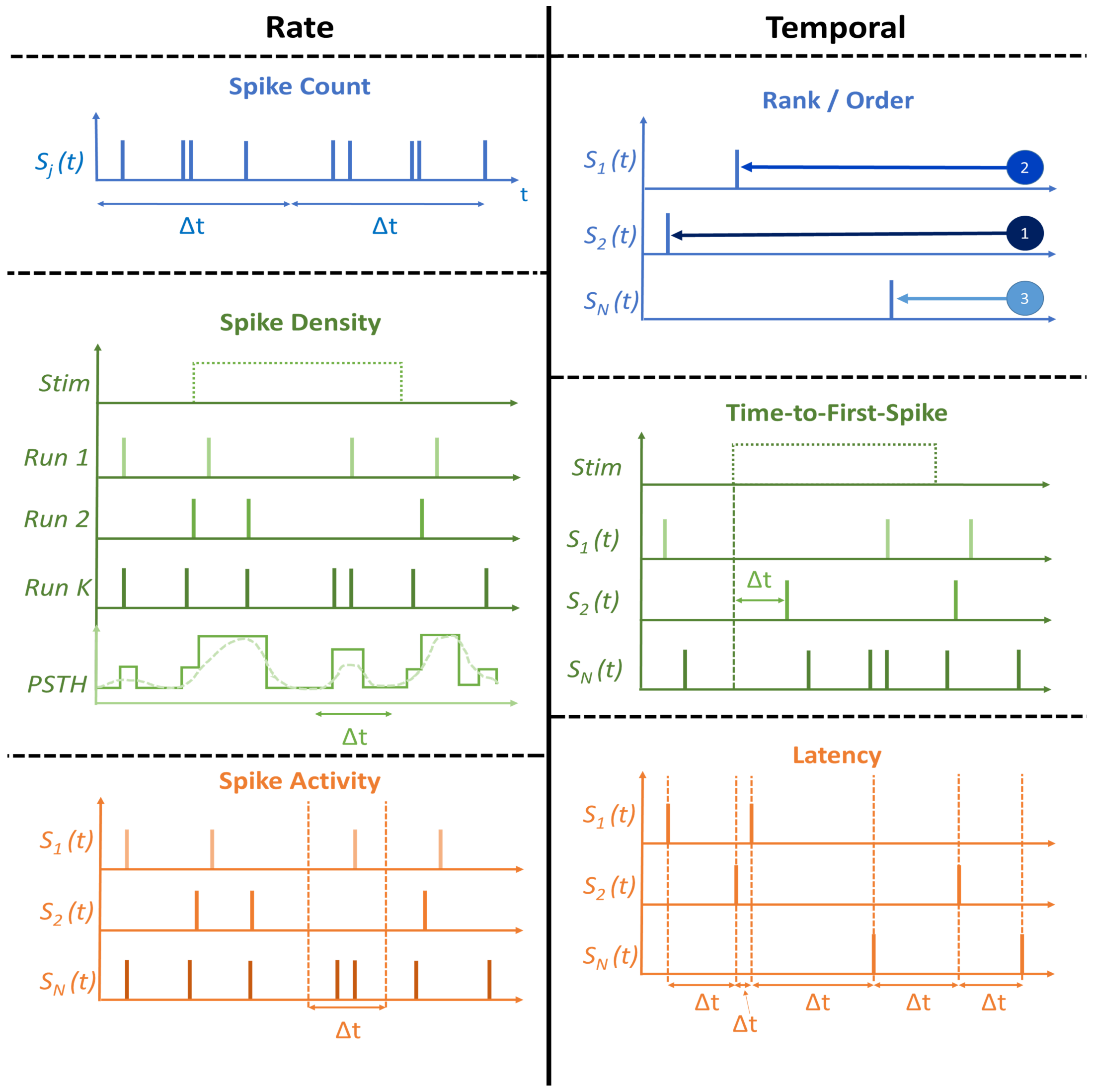


Fig. 3.5: Illustration of three popular coding schemes for rate-based and temporal-based neuron coding.

3.3.1 Rate coding

The term “Rate Coding” is typically used when referring to not just one type of mean firing rate, but instead is used to group this concept of using firing rates for encoding. This section will explore three of the most common and popular examples, Spike Counter (averaging of a single neuron over time), Spike Density (averaging of a single neuron over multiple instances) and Spiking Population Activity (averaging of multiple neurons over time). The following three subsections will reconsider the three concepts.

Spike Counter

Spike Counter [90] refers to the temporal averaging of the spike train, to calculate the number of spikes that have occurred over a period of time, Δt. This is shown within the top left of Figure 3.5, where the spike train of the jth neuron is calculated by summing the number of spikes that occurred in the given time Δt, then dividing by Δt to work our the frequency

$$Count(t) = \frac{n_{sp(\Delta t)}}{\Delta t} \tag{3.13}$$

The length of the observation time Δt is a hyper-parameter and varies depending on the task and the level of stimulation received by the neuron. Typically this would involve ensuring at least several spikes occur within the period. The periods of 100ms to 500ms are typical for this type of scheme and has experimental backing within sensory and motor system evaluation and classification [90]. However, due to the long time window, it is inconceivable that the full brain uses this neural coding scheme.

Spike Density

Spike Density [90] refers to a process in which the spike counter method is used on the same neuron over repeated neuron stimuli. The repetition of the stimulus then gives the neurons Peri-Stimulus-Time Histogram (PSTH), as was illustrated in Figure 3.5 under the Spike Density heading. Figure 3.5 highlights that the time t indicates the start of the recording, with the time described at Δt. The number of spike occurrences (density) is then the sum of all the spikes that occur within time $t; t + \Delta t$ over the number n_K of runs K. This value is then divided by both the time and number of runs to give the spike density PSTH

$$Density(t) = \frac{1}{\Delta t}\frac{n_K(t; t+\Delta t)}{K} \tag{3.14}$$

Spiking Population Activity

In a similar vein to the *Density*, the *Population Activity* [90] measures the spiking activity across multiple neurons. However, instead of monitoring the same neuron, under the same stimulus, to find correlations, the Population Activity looks at a group of neurons, that respond to the same stimulus similarly. This is then similar to that of cortical columns in the visual cortex [17]. In an ideal situation, all the input and output connections of the neurons within the population should be identical. Figure 3.5 illustrates the population activity with spike trains from N number of inputs from neurons in another population. Similar to the density, the spike count of each input is summed over the time Δt then divided by the time and the number of input connection.

$$Activity(t) = \frac{1}{\Delta t}\frac{n_{\text{act}}(t; t+\Delta t)}{N} \tag{3.15}$$

Population coding is a popular rate coding method due to the reduced levels of variability of the neuron while allowing the simultaneous representation of numerous stimulus attributes. Population coding also allows for a vast latency reduction compared to other rate coding methods. This is due to the reflection of changes across multiple neurons being almost instantaneous post stimuli compared to singular neurons. [101, 102]. The drawback with this methodology is that it requires all neuron to be identical including the same connection, which is less biologically plausible and more challenging to implement in hardware and software.

3.3.2 Temporal Coding

This section covers the alternative approach to Rate Coding, known as Temporal or Spike Coding. Three popular coding schemes are shown in Figure 3.5 on the right-hand side. They are Rank / Order Coding (ranking the neurons in order in which they spiked), Time To First Spike (TTFS) (selecting the first neuron after a given stimulus) and Latency Coding (using the inter-spike interval to note the exact timing of a set of spikes relative to each other).

Rank / Order

The top right panel of Figure 3.5 corresponds to *Rank / Order* coding. Information about a stimulus is encoded in the rank (first to last) or order in which a neuron (or population of neurons) emit their first spikes. For this encoding method to work the neurons must rarely produce more than one spike per stimuli. This work is underpinned by an experiment by Thorpe et al. [99] in which a binary classification task is set in which human subjects were told to release a button if they detected an animal in a photograph. The photograph only appeared for 20ms which reduced the amount of stimulation. The experiment helped to show that the processing required to identify the animal and release the button occurred in less than 150 ms. This research counters the rate based argument as in order to reach high order cortical areas, the visual information would go through at least 10 processing stages and given the constraints on real neurons firing rates. They proposed that a sparse order based temporal coding must be utilised to get such low latency. Due to the low latency of this approach rank order coding has been utilised for fast object classification mimicking the visual system where different orders encode for different classes. In tasks like object recognition, the performance of artificial networks are still poor when compared to humans. It could be due to this lack of temporal information that a performance disparity still exists. This method of coding has also shown successful results in more recent work with spiking convolutional NNs [88, 103], respectively show state of the art performance for image recognition tasks.

Time To First Spike

Time-to-First Spike, illustrated in Figure 3.5, is the time taken between the initiation of a stimulus and the first spike occurring. In practice, information can be encoded within the time it takes for the stimulus to produce a spike, Δt. This is formulated as any numerical input variable $x_i \in \mathbb{R}$ by its firing time $T_{input} - x_i c$ of any input neuron a_i, where $c > 0$ is some constant and T_{input} is the arrival time of the input (independent of x) [25, 104]. This would allow the magnitude of external inputs to be encoded, so that the input neuron could fire later or earlier, relative to the start of the stimulus. This simple encoding scheme would permit low latency processing of a rich amount of information, which was shown to be able to encode touch signal from the fingers [105]. This methodology has also been used recently to provide a method of allowing back-propagation of errors from temporal differences [106, 107]

Latency

Finally, *Latency* coding shown in Figure 3.5, can be seen as an extension of the Time to First Spike method by relative time difference not just to the first spike, and instead between all consecutive spikes. Precise spike timing has been shown to play an important role in the nervous system [108]. The spike timing also plays an important role in many of the temporal based learning rules. Borst and Theunissen have also shown that latency coding can be used to encode detailed information [109]. Latency coding is typically utilised within feed-forward networks only as recurrency can mess with the precise timing of information. However, attempts to maintain the precise spike timing in recurrent networks have been explored within the idea of reservoir computation [110, 111].

3.4 SNN learning algorithms

As detailed in the previous section, several options are available when designing an SNN, from the neuron model choice to the coding scheme it utilises. However, as was seen in the previous Neural Network chapter neuron types, architectures etc allow only so much progress. For a network to be useful in more complex applications it has to have a robust method to learn from the data it is presented. That is, there need to be some rules that permit synaptic weights changes to alter the SNN over time. This was the key turning point in ANN history, with the back-propagation for learning discovery. It was this discovery that took ANNs from an interesting experiment to state of the art in many fields. However, back-propagation does not translate as well to the SNN, due to the discrete asynchronous nature of the events produced and the intrinsic spiking nature of the neurons are not inherently differentiable. This leads to the open problem of learning within SNNs. Where the closer ties to biology, meaning that it can be difficult to develop intuitive algorithms since the exact mechanisms of how the brain learns with spiking neurons is still an open question. The following sections present the three most popular option relevant to the field of computer vision. That is the unsupervised learning methods using Hebbian rules and spike timing. Supervised rules using a variety of methods to allow back-propagation of errors through the SNNs and lastly the ANN-SNN conversion route, where a pre-trained ANN is converted to a network of a spiking nature.

3.4.1 Unsupervised Learning

The notion of unsupervised learning within SNNs is based on biological plausibility. In that neurons can only receive information from their local neighbours, meaning they have no notion of the task at hand, or how to solve the task. Learning simply involves an adaptation according to local activity. The most well known of these unsupervised learning rules is Hebb's Rule (Hebbian Learning) [23]. This describes how the synaptic connections should be modified, and as such, has inspired many unsupervised approaches [112].

More generally unsupervised learning rules take one of the following forms of synaptic plasticity: Unprompted decaying or growing of weights in the light of any activity [113]; Weight changes purely due to the effects of postsynaptic spikes alone [114]; Weight changes purely due to the effects of presynaptic spikes alone, the case for short-term synaptic plasticity [115]; Weight changes caused by presynaptic and postsynaptic spikes, the case in Hebbian learning [116]; and lastly, a modification to any of the previous method, but dependent on the current value of the synaptic weight [112].

Similar to the early perceptron models of neural networks, the modelling of the firing of neurons within an SNN have the advantage that Hebbian learning [23] rules are applicable. However, these rules do not deal with the exact timing of spikes, instead referring to neurons firing together. As this issue does not arise for these early neuron models, the question of how to incorporate the exact timing of spikes into Hebbian learning was unanswered. Furthermore, due to the continuous nature of most neural networks, Hebb's rule applied to the firing rates, which further reduced the need for further development. It would take a study that highlighted the correlation between the synaptic weight changes and the pre and postsynaptic timing [117]. Generally, if a presynaptic spike occurs shortly before the postsynaptic spike, this would result in a strengthening of the synaptic connection (potentiation). Meanwhile, if the presynaptic spike arrives shortly after the postsynaptic spike, this would result in weakening of the connection (depression) [24]. This process has been termed Spike-Timing-Dependent-Plasticity (STDP) and is seen as a temporally asymmetric extension of Hebbian learning induced by exact spike timing [118]. Similar to other forms of synaptic plasticity, it is believed that this could be the underlying learning and information storage processing in the brain [118, 119]. As mentioned with STDP, reoccurring presynaptic spikes arriving a few milliseconds before postsynaptic spikes typically leads to Long-Term Potentiation (LTP). Whereas, reoccurring spikes arriving after postsynaptic spikes typically leads to Long-Term Depression

(LTD). This change plotted as a function of the relative pre and postsynaptic spike timings are called the STDP function or learning window as shown in Figure 3.6. Figure 3.6 highlights the results from experimental work [24], showing how spike timing can directly contribute to synaptic weight changes. The weight change

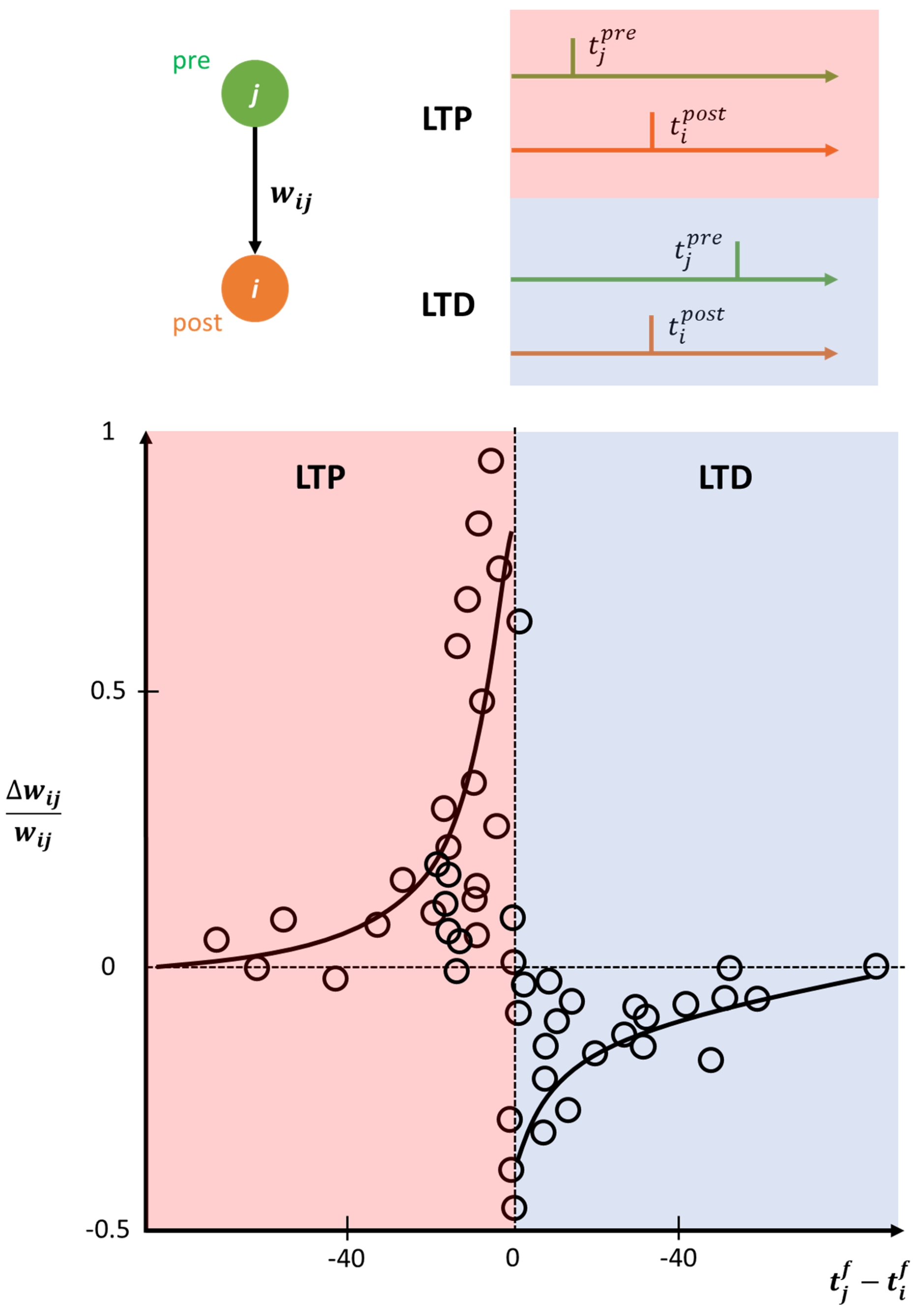


Fig. 3.6: A plot of the STDP function showing the change of synaptic weights as a function of the relative timing of pre and postsynaptic spikes after 60 spike pairings, highlighting LTP and LTD. Adapted and redrawn from Bi and Poo 1998

Δw_j of any synapse from a presynaptic neuron j is dependent on the relative timing between the pre and postsynaptic spike. Let the presynaptic spikes time at synapse j be t_j^f where $f \in \mathbb{R}$ counts the presynaptic spikes. Similarly, t_i^n with $n \in \mathbb{R}$ labels the firing times of the postsynaptic neuron. The total weight change Δw_j is then the combination of the pre and postsynaptic spikes [120, 121]

$$\Delta w_j = \sum_{f=1}^{N} \sum_{n=1}^{N} W\left(t_i^n - t_j^f\right) \tag{3.16}$$

where $W(\Delta t)$ is one of the STDP functions learning window as illustrated in Figure 3.6.

A typical choice for the STDP function $W(\Delta t)$ is

$$W(\Delta t) = \begin{cases} A^+ e^{\frac{-\Delta t}{\tau^+}} & \Delta t > 0 \\ A^- e^{\frac{\Delta t}{\tau^-}} & \Delta t < 0 \end{cases} \tag{3.17}$$

with this function being derived from both experimental data [122] and models [123]. Where parameters A^+ and A^- may or may not depend on the current synaptic weight w_j and the time constants are on the order of τ^+ $\tau^- = \pm 10$ms.

3.4.2 Supervised Learning

In contrast to unsupervised learning, which learns solely from raw data, the principal idea behind supervised learning is that precisely labelled training data is available for learning. Supervised methods for learning have shown great success with ANNs since being used on the perceptron [3] and have engrossed the research communities of all things NN ever since. While there is some evidence that this of learning is carried out in the brain [124, 125]. The exact mechanisms of how such learning is portrayed through spike timing, and dynamic weight adaptation, are still ongoing research areas in themselves. This can be problematic from the viewpoint of SNNs. As they often rely on guidance and inspiration from biology and neuroscience when designing new algorithms, mainly due to the complicated dynamics at play with the variety of spiking neurons.

With unsupervised SNN learning struggling to find a method to achieve high accuracy training, the task of finding an efficient supervised SNN learning rule remains hopeful. This hope takes the form of Several SNN-specific learning algorithms [126–131]. Among those are notable examples; ReSuMe [127], SPAN [130], and Chronotron [129]. The proposed supervised learning methods resolve the spiking neuron problem in a number of ways. From STDP based supervision training [127, 128], to empirically computing the weights of the network [131], while others deal directly with the spike by approximate its behaviour, in order to generate conventional errors [130].

One particular supervised method by Bohte et al. SpikeProp [126] discovered a way to back-propagate the gradient in SNNs, with a long list of recent methods utilising its application. [106, 132–135]. The difficulty with back-propagating errors with spiking neurons is the nature of the spike itself, in that spikes are undifferentiatable. However, a variety of ideas have been suggested to overcome this issue. Lee et al. [132] approximate the spike timings, while Zenke and Ganguli [135] proposed to replace the spike problem with a surrogate function to serve as the derivative. While these methods do allows for better integration with neuromorphic sensor inputs compared to the conversion approach, they do so at the expense of losing the exact spike timing. These spike timings contain valuable information that should be utilised and not discarded. Mostafa et al. [106] exploit these time coding aspects inherent in SNNs to enable errors to be calculated from these temporal differences. This allows the networks to be able to operate with less latency and computational overhead compared to the rate based method. However, this method only works on single-layer networks reducing its usefulness. Huh and Sejnowski [134], show that designing differentiable models can enable SNN networks to use gradient evaluation. Wu et al. [133] proposed a new framework to allow back-propagating the gradient along both the network depth and time dimensions. This method allows the training to take into account both the time and feature data in error propagation. One last supervised method to mention, based on the same surrogate proposal as SuperSpike [135], but showing great promise is Spike Layer Error Reassignment (SLAYER) [136]. This method approximates the derivative of the spike function based on a temporal credit assignment policy of previous spiking activity for back-propagating error to preceding layers. This method allows multi-layer networks to be designed and can utilise both weights and axonal delays as learning mechanisms. Overall, all of the supervised methods still suffer from the traditional deep learning issue for requiring large dataset, that then introduce biases into the network and make then less applicable to the multi-purpose general system, a problem less seen in unsupervised approaches. They also often can't match the performance results of converted SNNs on non-neuromorphic inputs.

3.4.3 Conversion of ANN to SNN

One method of training SNNs, which allows the bypassing of the SNNs inability to have easily conceived errors backpropagated through the network is ANN to SNN conversion. In this method, any type of ANN, from deep network to convolutional ones, can be trained as normal with gradient descent methods. This trained network is then adapted to work as an SNN by changing the weights and parameters of the neurons and connections. The ultimate goal of this method is to have an identical input to output result of the network. However, for an SNN this also means including the input and output encoding methods and not just the network itself.

The use of early conversion methods pre-date the modern surge in DL, and as such have this state of the art architecture as a reason to covert. Instead, the early methods were developed for event-based sensor processing, using hand-crafted convolution kernels on the spiking inputs [137,138]. This approach would go on to be the norm with the majority of the conversion approaches, that is converting the ANN (rate code) into an SNN (rate code), i.e. Converting the numerical value of the ANN to an equivalent rate. To convert these values which are not present in ANNs, hyperparameters such as the weights require rescaling to account for new parameters of the SNN neurons such as leakage and refractory periods (there can be more or fewer parameters dependent on the choice of the neuron, eg IF, LIF or Izhikevich). One slightly alternative approach early on was proposed using the Neural Engineering Framework [139] for the conversion of restricted Boltzmann machines [73].

The conversion to SNNs means the spiking component of the network does not get fully utilised and as such means, some of the benefits like temporal causality and online learning are lost. However, with this loss also comes gain in the conversion method can utilise the majority of the research into the state of the art methods from DL. Allowing a relatively straightforward approach to get highly accurate spiking networks [140]. Currently, most of the state of the art approaches within SNNs come from conversion approaches showing little to no loss in accuracy compared to the ANN counterpart while providing benefits in computational overhead and power [75, 141, 142]. With some methods even going as far as to provide boundaries of the expected performance of the converted SNN [78].

One factor in the performance change between SNNs and ANNs come within the utilisation of the max-pooling function as described in Section 2.3.2, as the non-linear mechanism behind it is difficult to implement in a spiking approach [143]. One method to overcome this is to replace the max-pooling with average pooling [75, 141, 144]. However, this alteration comes at the cost of a reduction in accuracy for these models. Alternative methods involve using: output units contain gating functions that only let spikes from the maximally firing neuron pass, and all other spikes are discarded; Removing the max-pooling layers and utilising strided convolutions, which approximates this pooling without the computational overhead [49]. Lastly, max-pooling can be implemented with latency codes [145], but this is not directly compatible with rate codes typically used for conversion. Another issue that faced earlier iterations of ANN to SNN conversion was the effect of negative activations with the ANN, as SNN activations are always positive. However, this issue became less prevalent due to the introduction of the ReLU activation function as seen in Section 2.4.1, which has no negative activations and a linear positive activation [144].

The conversion of an ANN to SNN is not always a method that results in a more efficient model with fewer computations. As the conversion and weight normalisation process may in fact cause more spikes to be produced, resulting in less energy efficiency. This trade-off between accuracy and latency [75], can be mitigated somewhat by promoting the SNN to reach a required performance level with the least amount of latency [146]. This issue is in part due to the actual conversion process inefficiencies itself, in that rate coding requires multiple spikes to represent the ANN activation value. This can lead to a situation where more spiking operations are required than the equivalent ANN multiply-additions. However, even in this less than ideal situation, the spiking operations are cheaper to process than the matrix multiplications. A point that is only magnified when implemented on efficient NM hardware. This helps to explain the push towards this conversion method as it ultimately leads to more energy-efficient implementations of the equivalent DL ANN network.

As part of the research conducted within this thesis, the Nengo DL simulator [147] was utilised to perform ANN to SNN conversion. This software simulator makes use of the Tensorflow [148] and Keras [149] to either train or take a trained model and convert it into an SNN. The Nengo DL simulator allows the transformation of a Keras or Tensorflow model to be converted into a Nengo model. This can be run as a typical non-spiking rate-based model initially to test and ensure the network is operating correctly. For the best performance results post-conversion, it is recommended to use the ReLU activation function in the original model. This helps to reduce the difference when converting the model to use a spiking version of the ReLU function also more typically seen as an IF neuron. At this point of the conversion process the aforementioned optimisation of the newly introduced parameters is required: Presentation Time (simulation time), Synaptic Smoothing and Firing Rates. The converted networks performance can be drastically reduced if these parameters are not set correctly. The Presentation time can be adjusted based on the typical firing rates of the neurons, e.g. 1 spike every 10 time steps (given the simulator time step of 1ms) would mean the firing rate is approx 100Hz, therefore a presentation time of longer than 10 time steps would be required to get useful results. Synaptic scaling acts as a method to smooth out the spiking response of the neurons in the simulation, typically a spike would only exist in one timestep of the simulation. This smoothing acts as a low pass filter applied to the output of the neuron. Intuitively, we can think of this as computing a running average of each neuron's activity over a short window of time (rather than just looking at the spikes on the last timestep). As a result of this low pass filter though the network would require more simulation time to reach the same performance. Lastly, the firing rate, or more specifically the scaling of the firing rate. This parameter helps to control the number of spikes from any given neuron within one timestep of the simulation. Where the maximal value is the true numerical equivalent to the pre-converted network, however, this result is the least efficient implementation. The ideal value is the highest accuracy with the least number of spikes per timestep. The firing rate is also a parameter that can be optimised during the training process if the network is imported into Nengo for training. By adding a loss function between the network layers to target a specific firing rate. This can be seen as an L2 regularisation step, to some pre-determined regularisation point. This pre-optimisation step is also a highly effective method to allow conversion to LIF neurons.

As mentioned the conversion and weight normalisation can result in more spikes being produced, therefore being less energy efficient. The common trade-off between latency and accuracy in SNNs [75] is often tackled by training SNNs to achieve a target performance level with the minimal latency [146]. Nevertheless, the rate to rate conversion process is not particularly efficient in terms of spikes being produced, due to multiple spikes being needed to represent one ANN activation value. In the worst-case conversion, the resultant network might require more spiking activity than the ANN needed multiplies and additions. However, these spiking operations are considerably cheaper than then ANN matrix multiplications and can be implemented on highly efficient neuromorphic hardware.

3.5 Neuromorphic Engineering Resources

Neuromorphic Engineering is an emerging research area, building upon the decades of progress made by the early neuromorphic researchers. However, the engineering aspect of this research has drawn more focus on application, and not necessarily brain research. This has helped to create a paradigm shift to a new form of sensing and processing, mainly that of an asynchronous, massively parallel, event-driven nature. This section aims to give an overview of the development in all the resources available: hardware - processing and sensors, and software that helps to enable this new computing paradigm.

3.5.1 Neuromorphic Processing

One of the main turning points in neuromorphic processing came in 2008, as part of the Defence Advanced Research Projects Agencies (DARPA) Systems of Neuromorphic Adaptive Plastic Scalable Electronics (SyNAPSE) development program. This brought about the inception of the IBM TrueNorth system [92] in 2014, a digitally implemented neuromorphic chip, that aims to run large scale networks with very low power consumption. TrueNorth comprises an implemented crossbar array with limited weight values representation and time-multiplexed neuron updates. However, this implementation allows one million neurons and 256 million synapses to be networked across the 4096 neurosynaptic cores. This also saw the first real commercial industrial entity enter the neuromorphic processing realm. Prior to this the Spiking Neural Network Architecture (SpiNNaker) [150] project had already started, though this was mainly aimed at academic research in neuroscience, computer scientist and roboticists. This is due to the SpiNNaker project being funded with the help of the Human Brain Project (HBP) [151] that funded by the European Union. SpiNNaker makes use of general-purpose arm cores tightly connected to local memory, on a single chip. It is highly reconfigurable due to the implementation of any neuron model as software running on the cores. This comes at the sacrifice of hardware acceleration to allow the maximal amount of model flexibility. Currently, SpiNNaker has connected over one million arm core together, which allows modelling of over a billion spiking neurons with biologically realistic connectivity (1 to 10 thousand synapses per neuron) all with a 1 ms time step for simulation. It is estimated that this could help to simulation 1% of the human brain [150]. However, with the introduction of SpiNNaker 2 they estimate they could simulate the whole brain, with their 10 million core machine [152].

An alternative solution to SpiNNaker from the HBP [151] is BrainScaleS [153]. This mixed-analogue-digital waferscale neuromorphic system allows a high amount of interconnectivity with 40 million synapses for up to 180 thousand neurons. BrainScaleS is a collection of HiCANN neurocores placed on highly interconnected wafers. The system aims to target the emulation of a precise biological neural network at faster than real-time speeds. Braindrop [154] is another mixed-analogue-digital design, built as a successor to the NeuroGrid [91] system. Braindrop makes use of the previously mentioned Neural Engineering Framework (NEF) [139] as the theoretical underpinning for its high level of abstractions used. This helps to alleviate the issues often found due to the diversity within analogue neurons. Braindrop makes use of the NEF programming to help automate the implementation of the non-linear dynamic system used for processing. Meaning specific user knowledge on implementation onto hardware is not required.

Recently, another industrial group has shown interest in neuromorphic processing chips. Intel with their introduction of the Loihi processer [94] has put a step in the balance of this commercial/academic development. Loihi is a digital processor that provides a flexible neuron implementation while catering for large scale SNN evaluation. This processor aims to bridge the gap between academic researchers and commercial products, while also tapping into the current trend of bio-mimicry and deep learning. This system allows on-chip learning with a variety of learning rules, a range of neuron models and a number of information coding protocols. Allowing it to emulate a variety of different algorithms. A Loihi chip has 128-neuromorphic cores implemented, along with 130,000 artificial CUBA leaky-integrate-and-fire neurons and 130 million synapses. The system design also promotes a highly scalable architecture meaning multiple chips can be utilised at one time [155]. Another noteworthy addition to the neuromorphic processor family is the introduction of the DYNAP [156, 157] series of chips. The DYNAP-SEL DYNAP-SE2 and DYNAP-CNN [158]. Both the SEL and SE2 feature 1,000 adaptive exponential integrate-and-fire analogue spiking neurons. The SE2 features 65k enhanced synapses with configurable delay, weight and short term plasticity. While the SEL has up to 80,000 re-configurable synaptic connections, with 8,000 of those synapses having integrated spike-based learning rules. Finally, the DYNAP-CNN features 1 million spiking ReLU neurons per chip for implementing Spiking CNNs. The chip also features a direct interface for neuromorphic sensors, mainly the dynamic vision sensors [158].

3.5.2 Neuromorphic Sensors

Neuromorphic sensors are another area that attracted attention from both academics and industries. The new sensing paradigm aims to take on the conventional sensor which can typically generate large volumes of redundant data and as a result tend to consume excessive power [22]. Recent example of neuromorphic sensors are silicon retinas (event-based cameras / neuromorphic vision sensors) [159–162], silicon cochlea (neuromorphic audio sensor) [163, 164], electronic nose system (neuromorphic olfactory sensor) [165] and robotics skin (neuromorphic tactile sensor) [166]. By far the most popular of these sensors is the neuromorphic vision sensor (NVS). The NVS and almost all other sensors make use of the Address Event Representation (AER) [167], a standardised method to evaluate sensor outputs. These camera-like devices are bio-inspired vision sensors that attempt to emulate the functioning of biological retinas. They differ from conventional cameras in that, they don't record all the information the sensor sees at set intervals. Instead, these sensors produce an output only when a change is detected. This in turn means they are capturing the luminosity at a set point in time, meaning a continuous temporal derivative of luminosity is output. Whenever this happens, an event $e = [x, y, ts, p]$ is created, indicating the x and y position along with the time ts at which the change has been detected and its polarity, where $p \in 1, -1$ is a positive or negative change in brightness. This change in operation not only increases the sparsity of the signal but allows for it to output asynchronously. Resulting in microsecond temporal resolution and considerably lower power consumption and bandwidth.

The NVS is able to deliver 1 to 3 orders of magnitude increase in output rate (33 ms traditional to 15 μ s Event-Based) [161]. This allows the sensor to have a much higher temporal resolution (in essence a 66000 frames per second super slow-motion camera for up to 800 pixels, as compared to real-world frames per second closer to 1-2,000) but without the caveat of the extra processing required for the pixels that didn't change. In other words, the sensor now has a dynamic relationship with the scene, as illustrated in Figure 3.7. Figure 3.7 highlights the ability to change the integration time of the events captured to create a frame (for visual representation and training). The top row shows a slow-moving Unidentified Aerial Vehicle (UAV), where a higher integration time is required to collect enough event to represent the UAV. While the bottom row illustrates the removal of motion blur, in a fast-moving UAV collision, by decreasing the integration time. The integration times can also be overlapped allowing a combination of both a longer integration time to capture events and the fine temporal resolution changes in the scene.

Another feature of the NVS is a high dynamic range, rated at >140dB versus the >60dB of traditional cameras [161, 162]. This allows the event-based camera to see in a wide variety of lighting conditions, from quickly changing brightness conditions, to low light ones, where traditional cameras would not be able to detect anything. This feature is highlighted in Figure 3.8. It can be seen that the

NVS camera can capture the shape of the UAV in a well-lit situation Figure3.8 (b) and (c) and is also able to capture the shape in the low light situation when the outline of the UAV is indistinguishable in Figure 3.8 (e) and (f). The images in Figure 3.8 (c) and (f) show a typical post-processing median filtering of the images to give better sensor noise suppression.

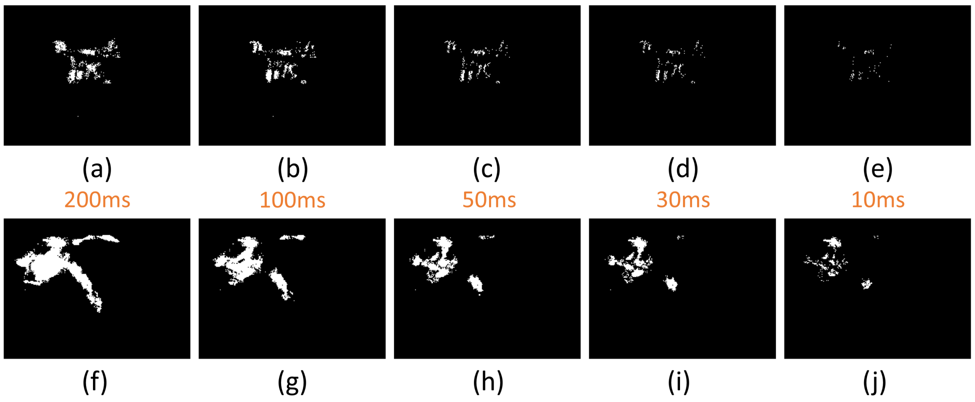


Fig. 3.7: NVS filtered events captured in a range of time frame. The top row showing a low speed scene and the bottom row showing a fast moving scene

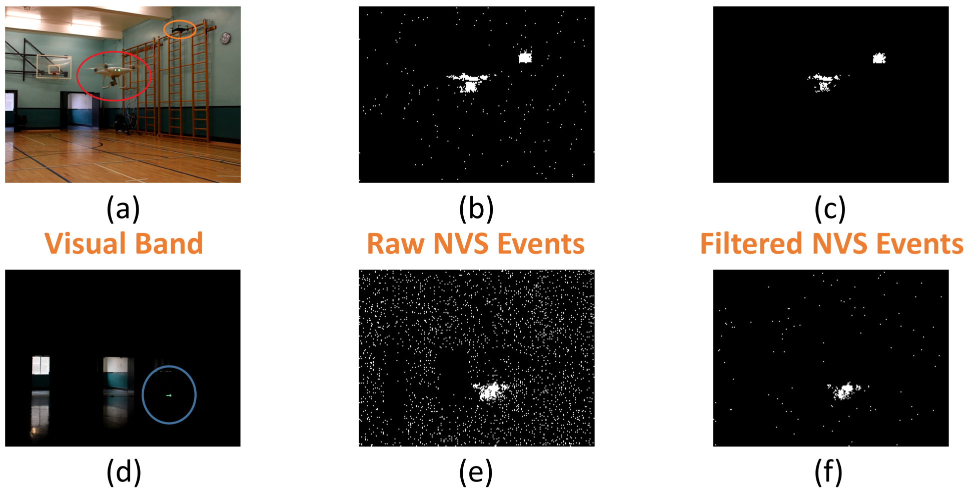


Fig. 3.8: High Dynamic Range within the NVS to capture stark lighting differences. The top row is the indoor well lit scene, with the bottom row showing a low light scene.

3.5.3 Software

Neuromorphic simulation software unlike the ANN equivalent has yet to settle on and develop a selection of do-it-all software tools. ANNs are able to make use of the large communities and software tools designed and developed by large companies such as Google (Tensorflow [148]) and Facebook (Pytorch [168]). However, as mentioned earlier in this chapter, the scope of the research within the neuromorphic community is much more diverse, so finding a useful one size fits all solution is less likely. As discussed previously, SNNs have a variety of research groups from neuroscience to engineering, with each requiring a different set of tools and features. Several prominent SNN simulation packages are, NEURON [169, 170], NEST [171], BRIAN [172], and ANNarchy [173] which all focus on biologically relevant simulations and allow complex models of single neurons, all the way up to the network level. For even more complex neuron models with higher-level functionalities platforms such as NEURON [169, 170] and Genesis [174] are used.

Neuromorphic frameworks such as NeuCube [175] and Nengo [176] shift the focus of the software onto high-level behaviours of SNNs. Both NeuCube and Nengo support rate-based coding implementations, with Nengo providing further support for simulations at multiple levels, spikes, rates and high-level behaviours, while NeuCube implements a spatial-temporal 3D spiking reservoir module. Nengo is often used to simulate high-level functionality of brains or brain regions, as a cognitive modelling toolbox implementing the Neural Engineering Framework [177] rather than a machine learning framework. However, with the addition of Nego DL [147] as part of the open-source project, supporting a Tensorflow backend. The ability to construct or convert models as previously mentioned as now implementable. Recently, a greater emphasis on frameworks that can utilise the tools of the deep learning community. The following three methods all have PyTorch implementations. BindsNet [178] is a general purpose framework designed for fast SNN simulations mainly developed for conducting AI experiments. Although it is based on PyTorch is internal network design language is different. The remaining two frameworks are more specific, with SpykeTorch [179] being used for temporal encodings and SLAYER [136] being used for rate encodings. SpykeTorch is optimized specifically for convolutional SNNs with at most one spike per neuron. SpykeTorch offers utilities for building hierarchical feedforward SNNs with deep or shallow structures and learning rules such as STDP and R-STDP [24, 120, 180–182]. SpykeTorch only supports time-to-first-spike information coding and provides a non-leaky integrate and fire neuron model with at most one spike per stimulus. The SLAYER framework was ported to PyTorch though originates in C++. This framework re-imagines the back-propagation algorithm for use within SNNs, through the distribution of the credit of the error back through time and the network layers, as the neurons current state depends on its previous. Through this implementation, the network can simultaneously learn both synaptic weights and axonal delays.

In addition to the largest software simulation packages, a number of prominent library add ons are also available, which often build upon already existing frameworks to provide the ability to implement SNNs. The largest of these libraries is PyNN [183], which is a simulator independent and can be used to run models in simulators like NEST and BRIAN. The PyNN provides a set of standard neuron, synapse and synaptic plasticity models, which can be integrated into all the different supported simulators. It also provides a set of commonly used connectivity algorithms with the flexibility to easily provide your connectivity in a simulator-independent way.

3.6 Perception-Action Cycle

The Perception-Action (PA) cycle is seen as the circular flow of information between an agent and its environment. This cycle is often a means of sensing guiding the actions in search of a goal. As each action consequently changes the environment which is then processed by the agent's sensory system, thus leading to the generation of further actions. These actions cause new changes that are sensory analysed and lead to a new action, and so the cycle continues. This exact sentiment is why the perception-action cycle is seen as the fundamental logic of the nervous system [184]. Meaning the individual perception and action processes are functionally intertwined. This leads to the conclusion that perception is a means of action and action is a means of perception. The brain as such has evolved for governing motor activity with the ability to transform sensory patterns into patterns of motor coordination also known as Perception-action coupling [184].

The PA cycle can describe simple functionality such as how a plant would direct its leaves toward the sun to absorb the most light. With the perception being of the amount of light, and the action being to turn the leaves towards this source [185, 186]. It can also be thought of in simplistic engineering terms and as "If Then" statement: If perception, Then action. However, it is through cognition that the PA cycle leads to complex decision making processes within the brain. Cognition is defined as "the mental action or process of acquiring knowledge and understanding through thought, experience, and the senses". Neuromorphic Engineering with SNN has the unique ability to be able to utilise this PA cycle due to the low latency asynchronous nature of the SNN, and also its inherent ability to be able to form dynamic systems used for action control. This notion leads to one of the breakthrough moments in neuromorphic engineering and spiking network in the creation of SPAUN (Semantic Pointer Architecture Unified Network) [187]. A computer modelling 2.5 million neurons that can recognise numbers, remember them, figure out numeric sequences, and even write them down with a robotic arm. The task for SPAUN is shown in Figure 3.9, which depicts the system observing a

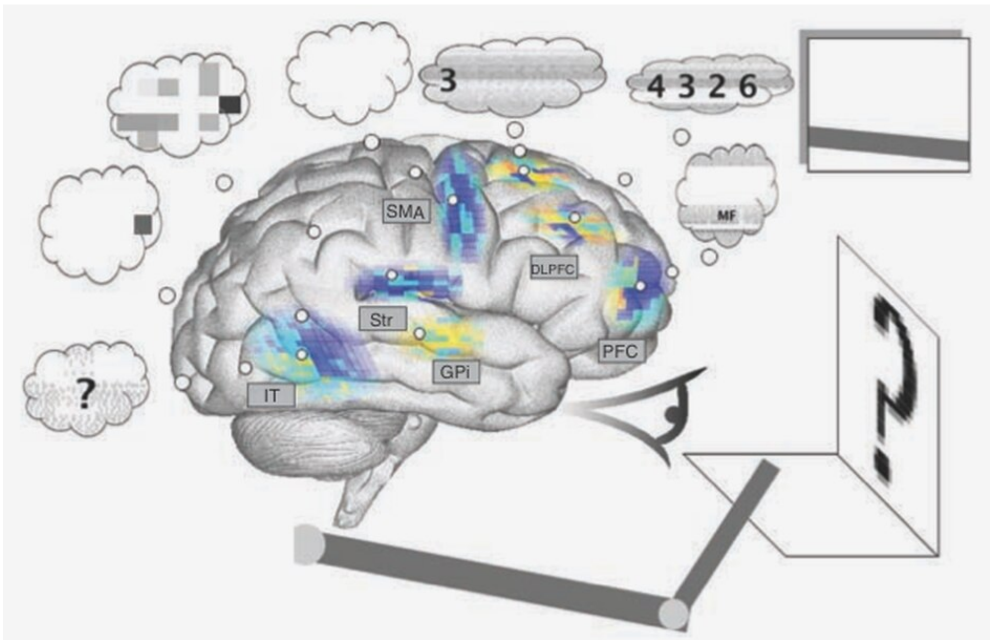


Fig. 3.9: A snapshot of the simulation movie of SPAUN in action. The image shows the input image on the right shown with the question mark. The output is drawn on the surface below SPAUN's arm and the neuron activity is mapped to relevant cortical areas within SPAUN's brain. The through bubbles show the cognition process. [187]

series of numbers, which it has to guess the next in the sequence then draw the number. The perception system is able to sense the differences between white and black pixels. This then leads to a multilayer cognition process, which starts with understanding, which is essential to solving an optical character recognition-like task. It then uses reasoning to determine the next value in the sequence and then plan the appropriate sequence of movements to draw that number. The action system is then the actual process of controlling the arm to enable it to correctly draw the number. The SPAUN system was able to show off how an end to end functioning system could operate using only spiking neurons. This type of system could lead to the “Killer Application” that neuromorphic engineering is still searching for, but as yet no updated version or competition to this system has been convinced.

3.7 Conclusion

This chapter detailed the complex underpinning that forms the field of neuromorphic engineering. Giving insight into the world of neuroscience through to the electrical engineering involved to create this interdisciplinary subject. This review covered the low-level functions of the singular neuron up to the high-level learning rules along with how to implement and simulate a variety of each. Throughout the chapter, it is clear to see how this third generation of the neural network is considerably more complex than the first and second. Although, it is also interesting to see how the surge in popularity for neural networks, in general, has helped to further popularise this field. With the promise of unique features such as asynchronous sensing, processing and control, along with low latency, low power and online learning to name but a few.

Chapter 4

Perception Understanding Action Framework

4.1 Introduction

Understanding and reasoning is a fundamental process in most biological perception-action cycles as seen in section 3.6. It is through our visual perception, that basic decision-making processes like “friend or foe” and “edible or inedible” can be realised, which ultimately is key to progression or survival. Adding some level of understanding into this cycle could help to deliver a robust robotic system that could perform more complex variations of the simple following and tracking tasks. Computer Vision (CV) has made this understanding a reality for robotics systems, with traditional CV methods providing simple feature extraction at low latency, or modern deep learning-based Convolutional Neural Networks (CNN) providing state of the art results in almost every task with high precision and accuracy, but at the cost of higher latency and computation throughput. This often leaves the CNN out of the reach of the small robotic system world due to its lower power and computational specifications.

Modern research looks towards biological inspirations to help solve these tasks, by bringing forward neuromorphic robotics, which seeks to merge the computational advantages of the system such as the NVS and neuromorphic processors, combined with SNNs which can allow for processing and control system structures. Typically a robotic system in this domain might aim to reach a Perception, Cognition, Action cycle, while the simpler approach of Understanding as a step towards cognition could be realised in an easier way. The Perception-Understanding-Action (PUA) cycle is seen as a stepping stone towards this goal.

This chapter presents both a novel method for spiking semantic segmentation and a PUA framework for it to be utilised. Image segmentation, as reviewed in Section 2.5 as part of *Understanding* is seen as a critical low-level visual routine for robot perception. As with a semantic understanding of the world, the robot can perform actions on a more contextual basis. This chapter introduces the spiking segmentation network, SpikeSEG built using a biologically plausible means of learning, STDP [24](section 3.4.1). This approach to understanding aims to exploit the low latency and low computational benefits of Neuromorphic Engineering as discussed in Chapter 3. Leveraging this spiking event-based nature of the full PUA pipeline.

The remainder of the chapter is organized as follows. Section 4.2 reviews related research topics covering each of the PUA framework individual sections. Section 4.3 presents the methodology, with an insight to each of the proposed system components. The results are detailed in section 4.4, a discussion of the research is provided in section 4.5 and section 4.6 provides the conclusion.

4.2 Related Work

The allure of low latency object recognition and localisation has brought the attractive features of the NVS (mainly the DVS [159]) to the forefront of research. Early low latency control examples, such as the Pencil Balancer [188] and the Robotic Goalie [189], help to highlight the latency advantages that an NVS can provide. Exploiting the sparse and asynchronous output of the sensor allow successful applications to these low latency reactive tasks. However, both systems fall short of fully capitalising on the event-driven asynchronous output, through a processing and control regime of similar nature.

The concept of exploiting the NVS low latency continues into object tracking. Low latency tracking relies upon robust feature detection, with geometric shapes being ideal features to detect. A number of methods have been implemented successfully, such as geometric constraints [190] along with advanced corner detection methods, for example, Harris [191] and FAST [192]. The use of more complex features such as Gaussians, Gabors and other hand-crafted kernels [193] provides a pathway to modern Convolutional Neural Network feature extraction approaches [194], that implements a correlation filter from the learned features of the CNN. This allows a multi-level approach whereby correlations of intermediate layers can also be performed to improve the inherent latency disadvantage of the CNN approach, albeit with an accuracy trade-off.

Spiking Neural Networks have seen success with NVS data used for object detection and classification [195–197]. Recent work has implemented Spiking Convolutional Neural Networks [88, 198] with NVS-like data created using a difference of Gaussian filter, suggesting the combination of SNNs and Deep Learning could yield successful results [199]. SNNs have also been utilised for tracking with an NVS through implementations inspired by the Hough Transform [200–202], to be able to detect and track lines and circles. Spiking Neural Networks can also be utilised to implement control systems, from simple altitude control [203] to an adaptive robotic arm controller [204]. Ultimately the majority of research only utilises one aspect of the SNN, either processing or control. Even though SNNs have been shown to implement a full perception cognition action cycle with Spaun [73], underpinning the ideology of a fully spike-based neuromorphic system similar to that proposed with the Perception Understanding Action framework in this chapter.

4.3 Perception-Understanding-Action Framework

The Perception-Understanding-Action framework specifies how the system will utilise the asynchronous event-driven nature of the neuromorphic spiking domain, and it is illustrated in Figure 4.1. In the Perception block, the NVS is used to sparsely and asynchronously encode the luminosity changes within the scene. In the Understanding block, inputs are understood through the use of the Encoder-Decoder SCNN (SpikeSEG [205]) contextualising and building an understanding of the scene through semantic segmentation. In the Action block, the segmented output is used to provide input to the spike counters at the edge of the field of view, allowing a simplistic semantic tracking controller to be realised. This control output would then be able to influence motors or actuators to facilitate an asynchronous end to end neuromorphic system.

This system has the potential to provide a low latency competitor to the Perception-Action robotic system where the sensor input is directly fed to the controller while providing an upgraded feature representation to the more complex line and edge detection-based approaches. The system can even provide benefits or replace some computer vision-based robotic tasks which utilise CNNs for complex feature extraction while providing lower latency and computational overhead. Furthermore, compared to the CNN, the SCNN provides a more readily understandable processing stage, where features are sparse and more visually interpretable.

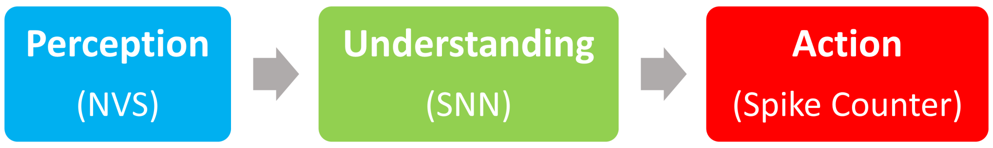


Fig. 4.1: Perception Understanding Action Framework flowchart.

4.3.1 Perception

A key element in producing a low latency system with low computational overhead is to have a sensor that can exploit the sparse and asynchronous computational elements of an SNN while still giving a detailed recording of the scene. Perception using NVS has become a promising solution. A NVS, for example, the Dynamic Vision Sensor (DVS) [159, 161], mimics the biological retina to generate spikes in the order of microseconds, in response to the pixel-level changes of brightness caused by motion. The NVS offers significant advantages over standard frame-based cameras, with no motion blur, a high dynamic range, and latency in the order of microseconds [206] as previously seen in section 3.5.3. Hence, the NVS is suitable for working under poor light conditions and on high-speed mobile platforms. There has been considerable research detailing the advantages of using an NVS (or similar) approach in various vision tasks, such as high-speed target tracking [192, 193] and object recognition [88]. Moreover, since a pixel of an NVS is a silicon retinal neuron represented by an asynchronously generated spiking impulse, this can be directly fed into an SNN as input spikes for implementing target detecting and tracking in a faster and more neuromorphic approach. This also allows two methods for perception a live asynchronous update, or the ability to accumulate spikes over a longer period to gain more information as shown in Figure 4.2. Figure 4.2 shows a conventional image from the CalTech-101 dataset [31] along with a NVS [159, 161] with an example of a live asynchronous and accumulated stream of data from the N-CalTech100 dataset [207].

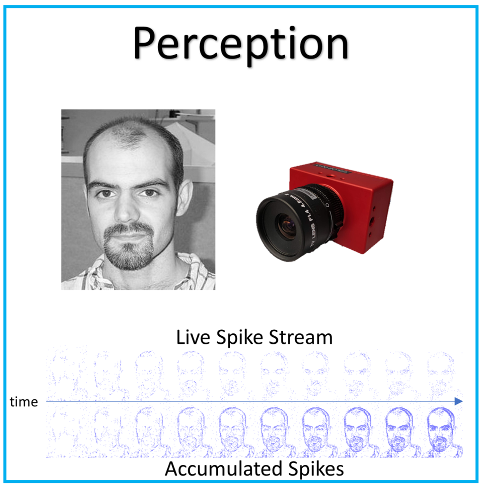


Fig. 4.2: Perception block showing the sensing process.

N-CalTech Dataset

The Neuromorphic-Caltech101 (N-Caltech101) dataset [207] is a neuromorphic spiking vision version of the original frame-based Caltech101 dataset [31]. The Caltech dataset contained both a "Faces" and "Faces Easy" class, with each consisting of different versions of the same images. The "Faces" class has been removed from N-Caltech101 to avoid confusion, leaving 100 object classes plus a background class. The N-Caltech101 dataset was captured by mounting a NVS on a motorized pan-tilt unit and having the sensor move while it views Caltech101 examples on an LCD monitor as shown in the video below. Each image is captured with a triangular three saccade movement, diagonally down to the right, diagonally up to the right, then from right to left horizontally. These movements result in a total of 300ms of asynchronous NVS data per image. A full description of the dataset and how it was created can be found in the paper "Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades" [207] The N-CalTech dataset was used throughout the experimental procedure of this chapter and chapter 5. Mostly using the "Faces" and "Motorbike" classes although the use of the "Stegosaurus, Watch, Cup, In-line Skate, Revolver, Camera, Stop Sign and Windsor Chair" make up the 10 classes used within this research. Figure 4.3 illustrates an example of each of the 10 classes mentioned from the dataset.

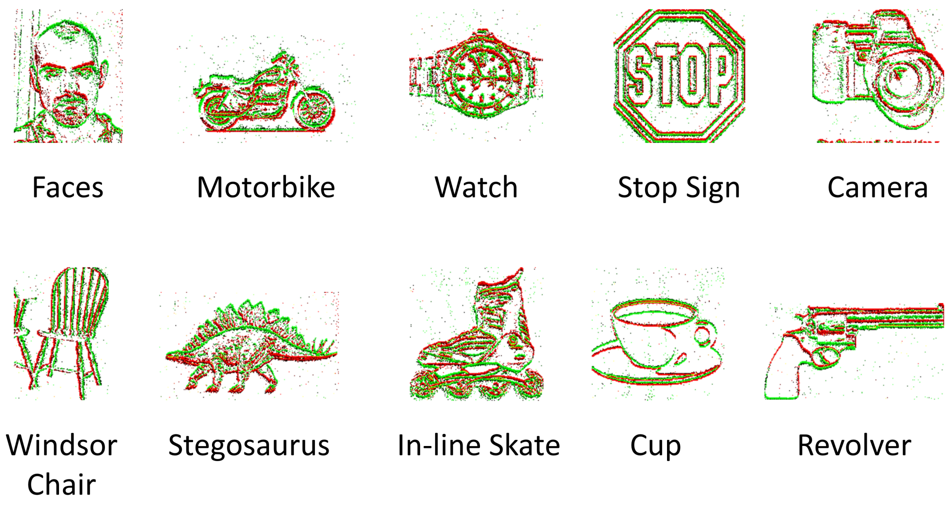


Fig. 4.3: Examples from the N-Caltech Dataset.

4.3.2 Understanding through Spiking Segmentation

The Understanding of this system is inferred from the semantic segmentation operation carried out by the SpikeSEG network [205], seen in Figure 4.4 within the Understanding block. The SpikeSEG segmentation network has received a number of improvements and upgrades along with its integration within the PUA framework.

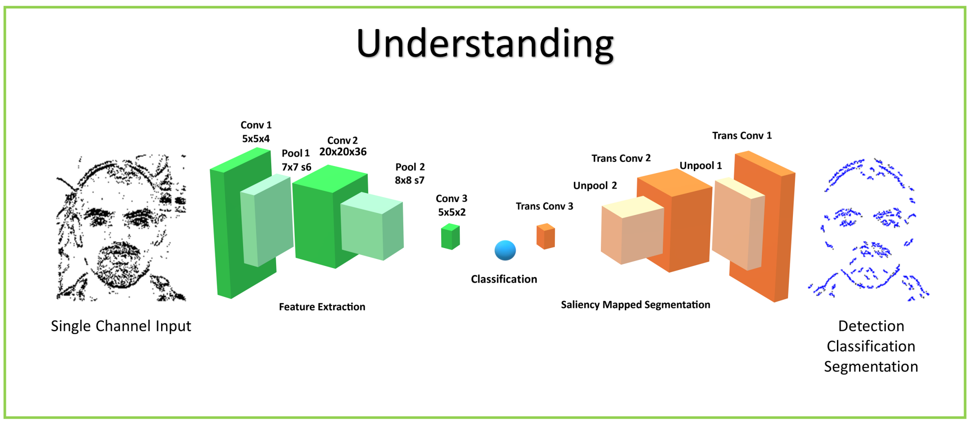


Fig. 4.4: Understanding block showing the semantic segmentation process with the SpikeSEG network.

Network Architecture

The network architecture illustrated within Figure 4.4 (Understanding) is made up of two main sections seen in green and orange, that relate to the encoding and decoding layers respectively. The network is split into these two sections where training only occurs on the encoding side, while the weights are tied to the mirrored decoding layers. This allows an IF neuron, trained with a layer-wise STDP mechanism, and featuring adaptive thresholding and pruning to help with feature extraction. This can then be used to help compress the representation of the input to allow the decoding layer to segment the image based on the middle pseudo classification layers.

This encoding-decoding structure symbolises a feature extraction followed by a shape generation process. The learning of the encoding process aims to extract common spatial structures as useful features, then decode those learned features over to the shape generation process, unravelling the latent space classification representation but with a reduction in spike due to the max-pooling process. As seen in Figure 4.4 the network has 9 computational layers *(Conv1-Pool1-Conv2-Pool2-Conv3-TransConv3-UnPool2-TransConv2-UnPool1-TransConv1)* . Between the Conv3 and TransConv3 layers, there is a user-defined attention inhibition mechanism detailed in section 4.3.2, which can operate in two manners: No Inhibition, which allows semantic segmentation of all recognised classes from the pseudo classification layer; or With Inhibition, which only allows one class to propagate forward to the decoding layers. This attention not only provides a reduction in the amount of computation but also simplifies the input to the controller.

Encoding

The encoding part of this system is derived from a basic SCNN with a simplified STDP learning mechanism [88]. To allow the network to better suit the framework and encoding decoding structure, several modifications were made. As the structure of the network is now fully convolutional there is no longer a requirement for a global pooling layer for classification. Instead, the final convolution layer is utilised as a mock classifier by mapping the number of known classes to the number of kernels used for feature learning, similar to that seen with FCN [11] in section 2.5. This method is also used to help the interpretability of the system as having one kernel per classes allows for better visualisation of the network features. Through the use of a modified STDP rule and adaptive neuron thresholding shown in sections 4.3.2 and 4.3.2. The encoder aims to capture the reoccurring features that are most salient through the event stream inputs. The input events are fed into the network via a temporal buffering stage, to allow for a more plausible current computing solution such as on the Intel Loihi Neuromorphic chip [94], while ideally, they would just be a constant stream. To internally mimic the continuous data, 10ms of event data is buffered into 10 steps, representing 1ms each (this value of 10ms is chosen based on empirical testing of the N-Caltech Dataset); this input data stream is shown in Figure 4.5. Fig 4.5, also illustrates what 1ms of data looks like over the 10ms (a) and how it looks if accumulated over 10ms (b). Figure 4.5 demonstrates how added noise affects the input stream, repeating the images in Fig 4.5 (a) and (b) with noise in 1ms steps in (c) and accumulated over 10ms in (d). For each time step in the encoding processing, a spike activity map Sk_{mt} is also produced, where m is the feature map and t is the time step. This allows an account of the exact spatial time location of each active pixel used in the encoding processing, which helps allow the decoder to map these active areas back into the pixel space.

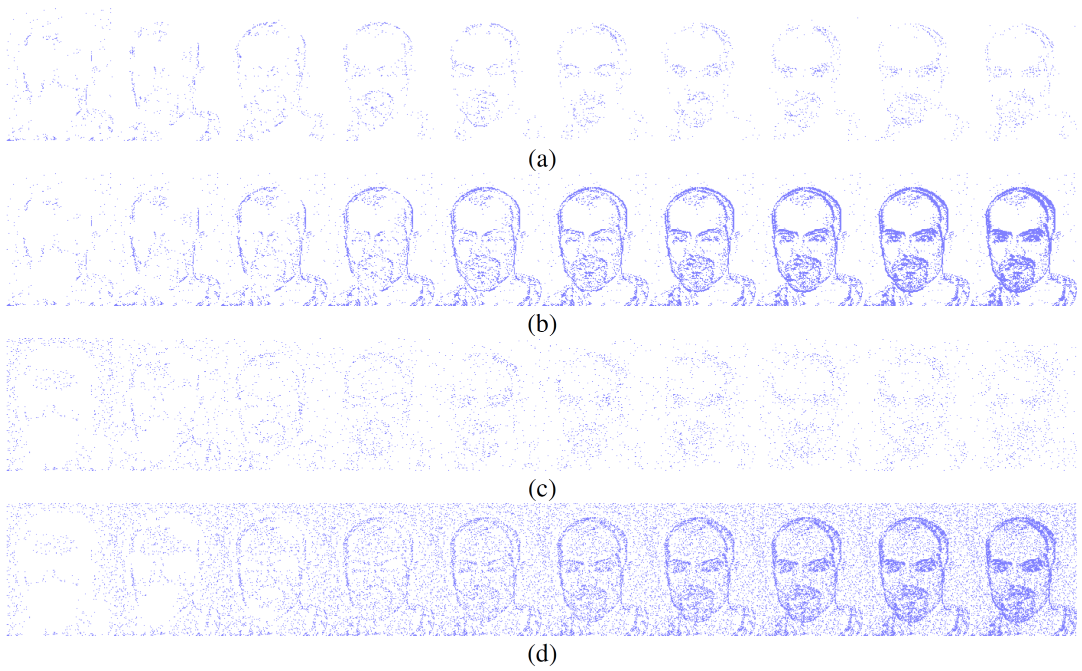


Fig. 4.5: Input event streams from N-Caltech Dataset “Face”, with (a-b) showing a 10ms clip over 10 steps going from left to right. (a) showing the input to the network per step and (b) showing the accumulated inputs for easier visualisation. (c-d) show a 10ms clip over 10 steps with additive noise to show how extra noise affects the input stream, with (c) showing per step and (d) showing accumulated.

Adaptive Neuron Thresholding

The adaptive neuron thresholding used within this system makes use of both a Pre-Emptive and Adaptive Thresholding (PEAT) approach. The pre-emptive approach acts as a form of homoeostasis called synaptic scaling [208], where typically chronically high activity in a neural network is dealt with by a negative feedback loop, to reduce the overall firing rate of the network. However, in this instance, it is used to manage the activity level coming from the input sensor into the network. This mechanism then adapts the thresholds of all the neurons within the network in a layer-wise manner. This ensures a high amount of spiking activity within the sensor doesn't cause erroneous propagations through the network. This pre-emptive approach alone is successful in stopping the progression of less structured noise features within the first convolution layer and structured noise when synaptic scaling is applied to all layers [205, 209]. However, along with the structured noise filtering process, this homoeostasis rule also accidentally removes some of the less common desired features from propagating as discrimination between these and noise from input spike count is impossible with this method alone. To correct this an adaptive thresholding [26, 198, 210, 211] approach is then combined with the pre-emptive one to form the PEAT process. Adaptive thresholding has been utilised in many approaches to SNNs, however, it is not instantaneous processing and usually works over multiple time steps. A combination of the pre-emptive approach with the adaptive allows the network to instantaneously react to high activity, reducing the errors propagating then adapting to look for the more subtle features that might have been lost. This adaptation can be seen as an intrinsic layer-wise synaptic scaling (a layer-wise spike counter) that is added to the overall extrinsic layer-wise synaptic scaling of the pre-emptive thresholding.

This algorithm is able to function both in training and during inference, with the threshold V_{thr} being dependent on both the number of spikes incoming from the sensor S_{in} and the spikes with the layers of the network S_l

$$V_{thr}(S_{in}, S_l) = \begin{cases} \frac{K_l}{4} & \text{for} \quad S_{in} < S_{in(min)} \\ \left.\begin{matrix} c + mV_{thr} + h^- & \text{for } S_l < H_l \\ c + mV_{thr} + h^0 & \text{for } S_l = H_l \\ c + mV_{thr} + h^+ & \text{for } S_l > H_l \end{matrix}\right\} & \text{for} \quad S_{in(min)} < S_{in} < S_{in(max)} \\ \frac{K_l}{2} & \text{for} \quad S_{in} > S_{in(max)} \end{cases} \tag{4.1}$$

where m is the gradient of the linear relationship between V_{thr} and S_{in}, with c being an initial offset. h is the homoeostasis offset determined to be either positive, negative or zero dependent on the layer-wise spike count S_l when compared to the set homoeostasis value H_l. While K_l is the convolution kernel size within that layer. Equation 4.1 follows a piecewise function such that V_{thr} is described as $\{V_{thr} \in \mathbb{N} \mid \frac{K_l}{4} < V_{thr} < \frac{K_l}{2}\}$. When the spike input rate S_{in} is within a normal range, the function is then defined by the bounded linear relationship with the homoeostasis offset. The values of h^-, h^0, h^+ and H_l are set through empirical testing by monitoring the range of $S_{l_{max-min}}$ and $S_{in_{max-min}}$ values from the N-Caltech dataset.

Once training is complete and the features within the convolution kernels are known, the thresholding changes to take into account the size of the active region of the feature, as the range of threshold values might now be smaller than in the training stage. The following modification changes to the V_{thr} function to account for these new outer bounds of the threshold

$$V_{thr}(S_{in}, S_l) = \begin{cases} \frac{F_{min}}{2} & \text{for} \quad S_{in} < S_{in(min)} \\ \left.\begin{array}{ll} c + mV_{thr} + h^- & \text{for } S_l < H_l \\ c + mV_{thr} + h & \text{for } S_l = H_l \\ c + mV_{thr} + h^+ & \text{for } S_l > H_l \end{array}\right\} & \text{for} \quad S_{in(min)} < S_{in} < S_{in(max)} \\ F_{min} & \text{for} \quad S_{in} > S_{in(max)} \end{cases} \tag{4.2}$$

where F_{min} is the smallest feature size within that layer. This parameter change ensures the threshold value does not exceed the smallest feature size, which would result in that neuron being unable to reach firing potential. In both cases, the training and testing input spike count S_{in} value affects the threshold for each input spike buffer, while the layer-wise spike count S_l is averaged over 10 inputs.

This allows layer-wise adaptability dependent on the amount of spiking within the previous layer. The algorithm now permits a high volume of spiking activity at the input to be initially pre-emptively dealt with, ensuring a large amount of spiking activity does not reach the controller, causing an undesired response. Then adapting the thresholds to allow sufficient spiking activity ensures a smoother and more robust controller output of the system. The key element of this method is to ensure a more robust and predictable outcome when a noisy, corrupt or adversarial input is received. With this being more of a concern due to the system be asynchronous end to end, a high volume incoherent input could directly lead to a wild or undesired response from the controller. This approach errs on the side of caution with the sudden increase in input spikes being inhibited first, and then excited to the desired level, in contrast to a typical intrinsic response of allowing the activity, and then inhibiting to the desired response.

Changes to STDP training with active pruning

A simplified unsupervised STDP rule [88, 212] seen in Equation 3.17 is used throughout the training process, based on the original STDP rule [24]. The simplification removes the exponential term from the equation as such

$$\Delta w_{ij} = \begin{cases} a^{+} w_{ij} \left(1 - w_{ij}\right), & \text{if } t_j - t_i \leq 0 \\ a^{-} w_{ij} \left(1 - w_{ij}\right), & \text{if } t_j - t_i > 0 \end{cases} \tag{4.3}$$

Furthermore, the algorithm now includes a Winner Take All (WTA) [213] approach to STDP. Meaning that it operates by only allowing one neuron (feature) in a neuronal map (feature map) to fire per time constant; this is viewed as an intra-map competition. This WTA approach then moves onto the inter-map inhibition, only allowing one spike to occur in any given spatial region, typically the size of the convolution kernel, throughout all the maps, similar to the hypercolumn of the Neocognitron [4] as discussed in section 2.3. As a result of these inhibition measures, two features can tend towards representing the same feature until such point where one becomes more active, while the other gets inhibited to the point of infrequent or no use. At this stage the feature representation has become obsolete and can be pruned or reset, allowing the opportunity to form another more useful feature. This method can be seen as an alternative to the weight decay method [132, 214], to tackle the issue of weight decay of neurons with relevant, but not frequently active features. To capture this information the layer-wise training method makes use of the training layers convergence values [88]

$$C_l = \sum_k \sum_i \frac{w_{i,k}(1 - w_{i,k})}{n_{w_{i,k}}} \tag{4.4}$$

where C_l is the convergence score for the layer l and $w_{i,k}$ is the ith synaptic weight of the kth convolution kernel. The $n_{w_{i,k}}$ is the number of individual weights contained within the layer (independent of the features), calculated by kernel size K_{xy} (e.g. 3×3 or 5×5) and the number of kernels used in the previous k_{pre} and current layers k_{cur}, $n_{w_{i,k}} = K_{xy} \times k_{pre} \times k_{cur}$. The pruning function makes use of the convergence score that is typically used to indicate when training is complete, as the convergence tends to zero due to the weights tending to 0 or 1.

Noticing that the layer-wise convergence is just a sum across all the kernels allows a modification to calculate the convergence across each kernel within that layer with respect to all previous maps.

$$C_{k_{cur}} = \sum_{k_{pre}} \sum_{i} \frac{w_{i,k_{pre}}(1 - w_{i,k_{pre}})}{n_{w_{i,k_{pre}}}} \tag{4.5}$$

This new terms $C_{k_{cur}}$ allows monitoring of each kernel during the learning process, instead of each layer. Meaning rather than just knowing if the layer has stopped learning, in order to move onto the next, each kernel can be monitored to see when learning has stopped. This is useful as previously mentioned, the obsolete kernels that started off learning similar features then stopped learning, have a high spike activity rate but don't converge to the optimal values of 0 and 1 for the weight. The high spiking activity is due to the kernel maintaining the high starting weight value which are values drawn from a normal distribution with the mean of $\mu = 0.8$ and a standard deviation of $\sigma = 0.05$. These kernels do not learn features that allow them to spike quick enough to receive weight updates from the STDP WTA rule, therefore remain close to the initial value. As the kernel had already started a divergence then convergence to a particular feature, once under-active it then attempts to converge to another commonly occurring feature. However, the kernel often converges to a useless feature representation that is unhelpful to the final result of the network. This pruning method, rather than simply removing the kernel, gives it the chance to learn a new feature from scratch by resetting the kernels weights. Thus allowing the best chance of convergence to a useful feature. This pruning process takes place once the convergence value of the layer C_l drops below the original starting value. As initially, the weights are diverging from the mean weight initialisation, before returning to the original convergence value on the way to zero. Once this milestone has been reached the pruning function is activated

$$Prune_{k_{cur}}(C_{k_{cur}}, C_l, S_k) = \begin{cases} 1 & \text{for} \quad C_{k_{cur}} > \bar{C}_l + 1\sigma_{C_l} \quad \text{and} \quad S_k > \bar{S}_l + 3\sigma_{S_l} \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

where $\bar{C}_l$ is the mean convergence for that layer, σ_{C_l} is the standard deviation of that layers values, S_k is the spike activity within an individual kernel. $\bar{S}_l$ is the mean spike count of that layer and σ_{S_l} is the standard deviation. If a kernel value has a convergence score higher than 1 STD from the mean, while having a spiking activity 3 STD higher than the mean spike rate in that layer, the kernel is reset with the initial weight distribution. Since many of the kernels are already converging to useful features this newly reset kernel will convergence to a new unrepresented feature. This method is able to be implemented during the training process due to the speed at which the unhelpful kernels converge to an unsuitable value.

Latent Space Inhibition for Attention

In order to have the network change its focus or attention, the classification layer also acts as an attention inhibition layer for this mechanism as shown in Figure 4.4. Although it is not formally a layer in the network as it operated between the Conv3 and Trans Conv 3 layers in Figure 4.4. It operates by inhibiting other neurons in that layer if a specific neurons feature is chosen to be the attention. Operating with the same principles as the intra-map inhibition within the encoder, though now the spatial region is the whole latent space. This is an external mechanism to the network as otherwise, the network will give equal attention to the full scene and semantically segments all known objects within a scene. This allows a simplification of the output of the network fed to the controller, allowing the attention of the system to be narrowed to that particular pseudo-class. This inhibition can also work autonomously where the pseudo-class with the most activity is the attention of the network, allowing the network to switch attention to known classes based on their prevalence within the scene.

Decoding

The Decoding Process makes use of the same unpooling and transposed convolutions upsampling methods [11, 12, 59, 65] as seen in section 2.5.1. This allows the pixels in the latent classification space to be mapped back into the original pixel space. However, unlike the previously mentioned methods, no learning mechanism is used within the decoding half of the network. Instead, by utilising a similar method to tied weights of auto-encoders [50] and switches [67] allows the weights of the encoding layers to map directly to the decoding layers such that $W_{j,i(encoding)} = W_{i,j(decoding)}$, meaning the encoding and decoding weights are identical. The decoding half of the network can utilise the encoding weights and maintain the temporal sequence of the input through the use of a spike activity map (SAM). This SAM acts similarly to the pooling indices described in section 2.5.1. However, it works on the convolutional layers to index the active spikes during the encoding process, as illustrated in Figure 4.6. Figure 4.6 highlights where the SAM process occurs within the encoding (green box) and decoding (orange box) sequence, along with the unpooling and transposed convolutions. The other part of the SAM process illustrated in Figure 4.6, is how the temporal continuity is maintained. To map the correct encoding time step t^e to its equivalent time on the decoding stage t^d, the processing time t^p between the equivalent encoding and decoding convolution process must be considered.

Since the encoding neurons emit at most one spike per buffered time input, the SAM is used to keep track of the first spike times (in processing time-step scale) of the neurons. Every stimulus in the SAM is represented by SAM_{x,y,m,t^e} where x and y is the spatial location within the mth feature map and in the tth time step. This then converts to the SAM used in the decoding section where SAM_{x,y,m,t^d} is used after considering the time to process, where a spike in the decoding layer depends on

$$SAM(x,y,m,t^e,t^p) = \begin{cases} 1 & SAM_{x,y,m,t^d} = SAM_{x,y,m,t^{e+p}} \\ 0 & otherwise \end{cases} \tag{4.7}$$

For this particular network, SpikeSEG, the range of processing times are as follows: 9 time steps between Conv-1 and TConv-1, while only 5 steps between Conv-2 and TConv-2 and 1 step between Conv-3 and TConv-3. So, if a spike occurs at time step 2 within Conv-1, the temporal check will only allow TConv-1 to allow a spike at that location at time step 11.

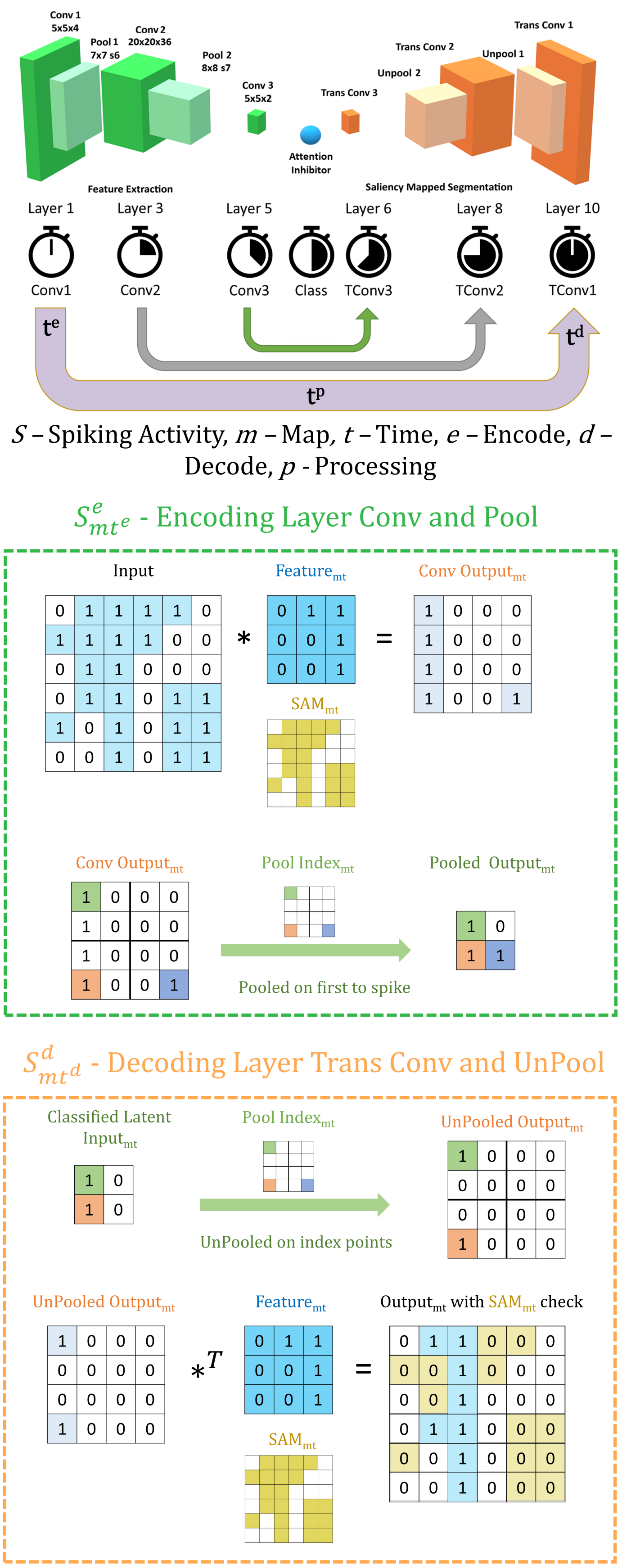

Conv 1 5x5x4
Pool 1 7x7 s6
Conv 2 20x20x36
Pool 2 8x8 s7
Conv 3 5x5x2
Attention Inhibitor
Trans Conv 3
Unpool 2
Trans Conv 2
Unpool 1
Trans Conv 1
Feature Extraction
Saliency Mapped Segmentation
Layer 1
Layer 3
Layer 5
Layer 6
Layer 8
Layer 10
Conv1
Conv2
Conv3
Class
TConv3
TConv2
TConv1
t^e
t^d
t^p
S – Spiking Activity, m – Map, t – Time, e – Encode, d – Decode, p - Processing
$S^e_{mt^e}$ - Encoding Layer Conv and Pool
Input
Feature_{mt}
Conv Output_{mt}
SAM_{mt}
Pool Index_{mt}
$\text{Pooled Output}_{mt}$
Pooled on first to spike
$S^d_{mt^d}$ - Decoding Layer Trans Conv and UnPool
Classified Latent Input_{mt}
$\text{UnPooled Output}_{mt}$
UnPooled on index points
$*^T$
Output_{mt} with SAM_{mt} check

Fig. 4.6: Decoding using transposed convolutions with spike activity mapping (SAM), resulting in active pixel saliency mapping

4.3.3 Action

In the third part of the PUA framework seen in Figure 4.1, the Action part of the system with its spiking controller is directly influenced by the attention mechanism, as when no attention is chosen the controller acts on all the segmented data being output by the SpikeSEG network. This could cause unwanted control output if the scene contained more than one known class, as unknown classes should still be removed by the process. Once a class has been chosen as the attention, the segmentation output is reduced to only that class, as illustrated in Figure 4.7, which allows for a simple spike counter controller to produce a more robust and reliable output. This is due to the segmentation output only containing information relating to the attention of the network, the controller's task is just to keep this in the centre of the field of view. The simplicity of the controller also allows it to take advantage of the asynchronous event-driven system to provide low latency tracking updates, a key element of the system. However, if there was more than one instance of a class in a scene there is no way to separate the two instances, so tracking would be based on all instances of a class. Nevertheless, this system would improve the purely spiking activity tracking systems by adding some semantic context to the activity, while the simplified spike counter in this instance allows class-based tracking could be enhanced with more complex spike tracking such as dynamic neural fields [215] or through adding integral and derivative values to the controller to smooth out the proportional spiking signal.

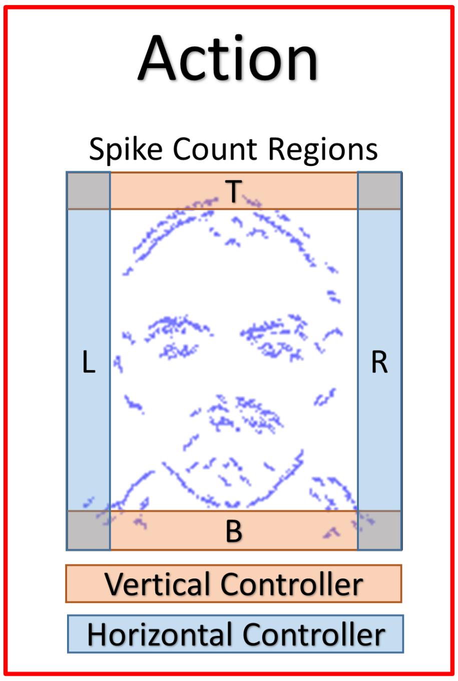


Fig. 4.7: Action block showing the tracking method

4.4 Results

In this section, a series of experiments on individual and multi event-stream recordings are presented. The metric used in this chapter is the Intersection over Union [216] (IoU, also known as the Jaccard Index) to grade the segmentation, which guides the control system of the network and ultimately, with user choice, the attention of the system. The IoU metric was used due to the availability of the bounding box annotations within the subset of the N-Caltech dataset [207]. Allowing the ability to compare the overlapping bounding box regions of the ground truth in a proposed system. The feasibility of the attention-based tracking is also encapsulated within the IoU value, though due to the small saccade movements of the camera within the N-Caltech dataset, it is infeasible to use this to highlight spike-based tracking. This is due to two issues throughout the movements. The first is the IoU value only receives a small change as the displacement is often less than 10 pixels. The second is that the occurrence of segmented spike activity in the controlled regions is limited due to the tight field of view around the class in the scene. This results in the testing of the Perception and Understanding system only with this data. To ensure testing of the full Perception, Understanding and Action system, two further experiments were carried out. The first with multi-input streams on a large input space and the second using our own captured DVS data of a desk ornament with a hand-held sensor. Secondly, the results sections show how the system is more robust and interpretable than alternative models, with the use of the Pre-Empt and Adapt Thresholding and the contour like sparse features within the weights of SpikeSEG.

Within these experiments the step time for any processing is now linked to the input time step, meaning internal propagation of spikes take one step (or 1 ms) per layer, resulting in an 11 ms lag to get the segmented results. This allows for better visualisation of the asynchronous manner of the processing and control for each step. However, this does not reflect the actual processing time of the network which, given its complexity compared to similar models ran on neuromorphic hardware, would most likely be able to execute this task in real-time for the 1 ms step, meaning a full pass through the network per input step. However, testing in this manner would not fully highlight the asynchronous advantage, especially within a dynamic environment.

One further note is that throughout all the testing the features of Convolution Layer 1 are pre-set to best-found features for initial edge detection, which results in a horizontal, a vertical and two diagonal lines which can be seen later in the Interpretability Section 4.4.3 within Figure 4.18.

4.4.1 Perception to Understanding with Segmentation

The first experiment uses two subset classes from the N-Caltech dataset [207] are used to evaluate the Understanding section of the system, "Face" and "Motorbike" similar to testing seen in Kheradpisheh et. al. [88]. These tests are used as a proof of concept, to see if the feature extraction and segmentation capabilities of the SpikeSEG network can successfully classify and segment the two distinct classes. Testing was carried out using 200 samples from the Face and Motorbike classes with another 200 used for testing. Where each of the images is the 300ms NVS data stream was broken into its 10ms chucks as described in the encoding part of section 4.3.2. This means a total of 6000 data streams are used in the training process with the same number also used for testing.

On this single stream input which only contains a singular class with variable amounts of background noise and clutter, the network is able to gain an accuracy of 96.8% within the pseudo classification layer. Further to this, it was able to achieve an 81% mean IoU score over each of the 10ms buffered input, results are also shown in Table 4.1. The segmentation also maintains an IoU value of above 70% when tested on the full 300ms data stream, which given the variable amount of spiking activity throughout the three saccade movements. For reference, segmentation values of 50% are acceptable, 70% are good and 90% very precise [217] and if the full input size is used for IoU the average output is approximately 57%. The same testing scenario was also used to test the performance of the network without the adaptive threshold and pruning when the network only utilised the pre-emptive thresholding [205]. The newer method results in a marked improvement on the testing using the older method seen within [205] of 92% and 67% for accuracy and IoU. This increase in performance can be attributed to an improved feature creation allowing a more detailed representation, as will be detailed in section 4.4.3, allowing an improvement in both the accuracy and segmentation.

To aid in the understanding of how the SpikeSEG network allows semantic segmentation to be realised a breakdown of the output from the convolutional layers is shown for the Face and Motorbike examples in Figure 4.8. This Figure illustrates the journey from the NVS input (a), through the latent space feature extraction process of (b) and (c). Where the coloured pixels then represent different features and their location within the original pixel space. The classifier is shown in Figure 4.8 (d) shows the different coloured classification features used for the Face (blue) and the Motorbike (red) and the location of this classification region. Figure 4.8 (e) and (f) then show the decoding process of upsampling from the latent classification space back into the pixel space again. With the coloured regions now being the sparse version of their encoding counterparts, due to the segmentation only using the most salient features to reduce noise. Lastly, Figure 4.8 (g) highlights the output of both the Face and Motorbike segmentations with a precise and concise representation of the class. The output segmentation of the network is then shown overlapped onto the input image in Figure 4.9, with (a) showing the pixel related to the Face class in blue and (b) showing the pixels related to the Motorbike class in red. Figure 4.9 further highlights the features that have and have not been captured, due to the network removing all but

Table. 4.1: Results from each of the experimental setup, listing both the accuracy and intersection over union

Dataset	Classification Accuracy (%)	Intersection over Union (%)
N-CalTech (2 Class)	96.8	81
N-CalTech (5 Class)	86	76
N-CalTech (10 Class)	75	71
Multistream N-CalTech	96.8	81
Multistream N-CalTech with Noise	95.1	79
Panda	94	75

the most salient features used in the classification process and as such removing the background noise and clutter also. With the successful classification and segmentation of the input data stream, it is important to note that the output of the network has maintained full temporal continuity as is highlighted in Figure 4.10. This illustration shows that the saccade movements of the N-Caltech101 dataset, as mentioned in section 4.3.1 are maintained through to the output of the SpikeSEG network. Figure 4.10 (a) showing a downward and right movement, (b) showing an upward and right movement and (c) showing horizontal movement to the left. With each of the three images showing an accumulated image of the 100ms allocated to that saccade movement. Figure 4.10 is also useful in highlighting the difficulty in using the NVS data for classification as each of the three images appear to focus on different regions of the face, dependent on the feature saliency due to that particular movement direction. This is particularly noticeable in Figure 4.10 (c) where the top of the head is under-represented when compared to (a) and (b). However, this result is expected as when the saccade movement is horizontal the difference change in horizontal features will be at its minimum.

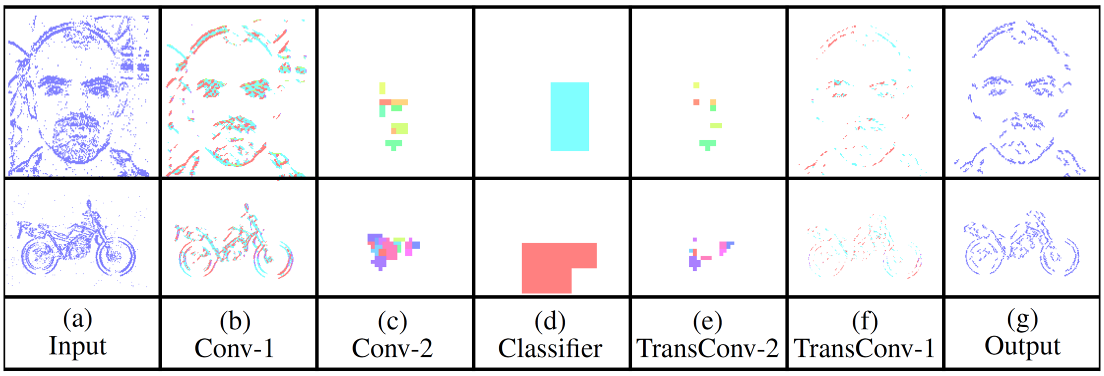


Fig. 4.8: Segmentation performance of the network on a Face and Motorbike example, highlighting the encoding transition into the latent space used for pseudo classification (a-d), then retracing of chosen features back to pixel level (d-g), with the output at each of the named convolution layers.

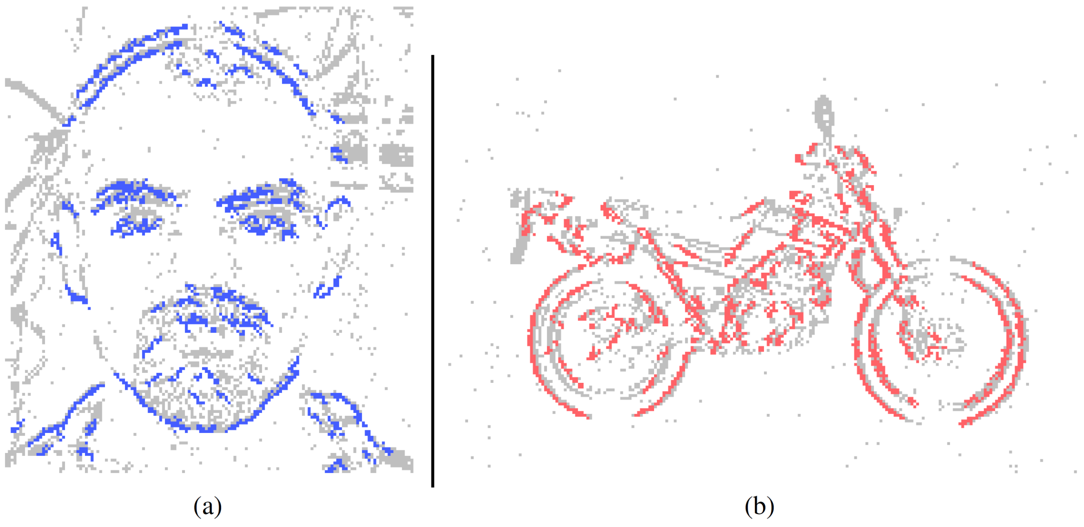


Fig. 4.9: Segmentation overlays for the (a) Face and (b) Motorbike class from the N-CalTech dataset

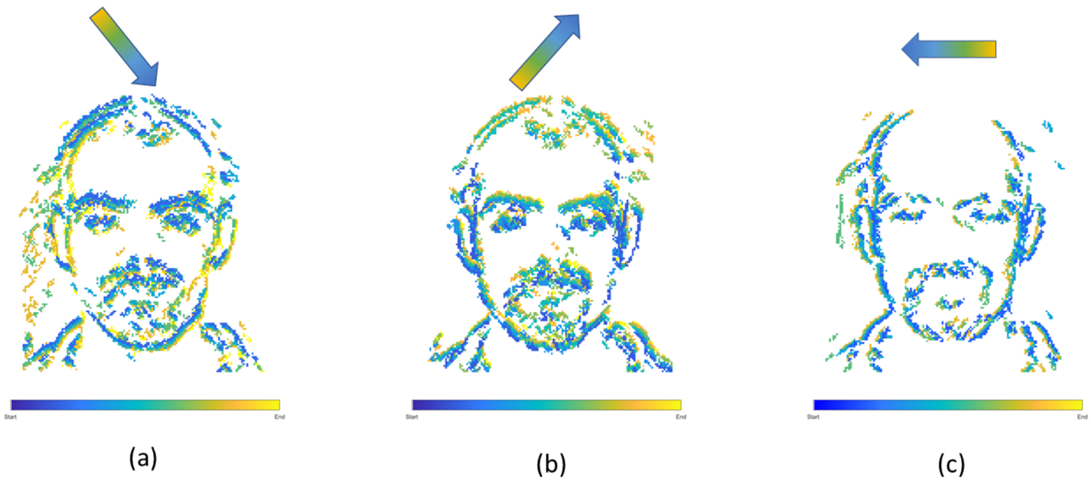


Fig. 4.10: Overlapped Segmentation output over the complete event stream, showing the triangle of movements over the three saccades, (a) first movement, (b) second movement, (c) third movement.

N-Caltech Dataset Extended

To further evaluate the scalability of the model, 2 additional experiments were carried out with 5 and 10 classes. This allowed testing the model with 2, 5 and 10 classes within the same experimental parameters, comprising of 16 features per class in the second convolution layer and 1 per class in the third convolution layer, with active thresholding and pruning. 16 features were found to be a suitable value for the number of features through prior empirical testing, where more features gave no further improvement, while fewer features were unable to capture the variation of some classes. The further classes added are Inline Skate, Watch and Stop Sign for the 5 class, while Camera, Windsor Chair, Revolver, Stegosaurus and Cup are added for the 10 class experiment.

These classes are chosen due to low variability in image spatial structure, which was investigated through testing of each of the 100 classes available. This involved attempting to train the SpikeSEG SNN with one classification layer to attain the best classes to continue with testing. Another contributing factor in the choice was the intra-class variance, the amount of difference per instance in a class has and the proximity to inter class variance, how similar it looks to another class. Some classes have vastly different shapes per image, and considering the Neuromorphic dataset essentially only captures the edges of the shapes and has no colour information this makes the task considerably more difficult, especially in an unsupervised context. Since the network is only looking for natural spatial structural similarity, classes that have a large intra-class variance compared to the overall inter-class relationship [218] were avoided. Furthermore, some class types are very similar, with multiple chair classes, along with similar plants and animal classes. So, the final 10 classes are the best selection of objects that were able to create distinct features and converged to classification when tested individually, that would avoid significant overlap with an already selected class.

With this in mind and due to some of the additional classes having a smaller number of sequences, the number of training and testing instances was changed to suit, at 20 training and 10 testing images per class. Overall the network was able to achieve classification accuracies of 86% and 75% and IoU values of 76% and 71% for the 5 and 10 classes respectively, results are shown in Table 4.1. The decrease in overall accuracy with additional classes can be partially attributed to having fewer training example per class but was also to be expected as the features built in the second convolution layer tend to get more similar. This similarity will be visually detailed in section 4.4.3 with the Interpretability showing the different features learned in the convolution layers. With this closer similarity of layer-wise features, an example of how the active pruning mechanism is shown in Figure 4.11, where a number of the features within the second convolution layer have a slower convergence rate while maintaining a high spike activity. This typically suggests the feature is not very discriminative and is an ideal candidate for being reset to learn a new feature. Figure 4.11, shows the original features just prior to reaching the pruning checkpoint within (a), then indicates which features are chosen to prune with the feature being reset to random initialisation within (b), then finally resulting in new features shown in (c).

Drawing insight from the results, within the 5 class experiment the inter-class variance was high. However, once the 10 classes were added these inter- and intra-class variances appear to overlap. Resulting in many of the classes relying on similar features constructed from circles, with Motorbike, Cup, Camera, Watch, Stop Sign and Face at times producing features are that indistinguishable from one another. It was also noted that as the number of classes increased the difference between the average number of features in a kernel per class (that is ones that can be recognised as belonging to a particular class) leads to a higher likelihood that the class with the highest average feature number will be the most active. Within the last experiment with the 10 classes, this was prevalent within the Revolver class as it had an average feature count in convolution layer 2 of around 200, while the average for the camera was 110. This results in a higher chance that the revolver was classified by mistake ultimately bringing the overall accuracy down.

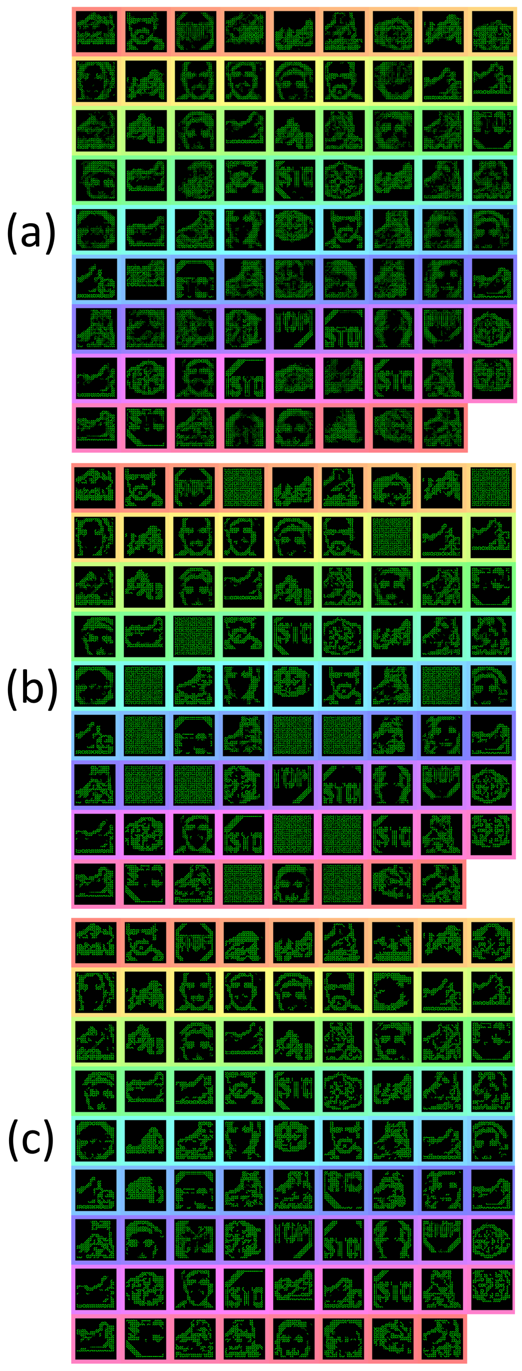


Fig. 4.11: Features from the second convolution layer during training highlighting the pruning process. (a) highlights the features prior to pruning, (b) shows which features were reset to initial parameters and (c) shows the newly learned features.

4.4.2 Perception, Understanding and Action

This section is split into two parts both further testing the full PUA system, the first continues using the N-Caltech Dataset, however with multiple simultaneous inputs. The second part makes use of recorded data of desk ornament from a hand-held NVS to provide a further example of how the system works within another test environment and how the action part of the system deals with a moving class.

N-Caltech Mutli-Stream Input

Building upon the results gathered from the successful process in section 4.4.1, this experiment looks at how the system would deal with multiple input streams. This allows the network to demonstrate the segmentation ability in the face of multiple distractors such as other classes and the addition of spatial-temporal Gaussian noise with an average peak signal to noise ratio (SNR) of 18dB. The input spatial size is a 3x3 grid based upon the size of the central input stream, with any of the four corners being able to display another input stream as illustrated in Figure 4.12. These input streams illustrated in Figure 4.12 (a) and (b), consist of 1 face and 2 motorbikes for the known classes and 2 Garfield streams for the unknown (Garfield in another class of the N-Caltech dataset, but is never used for training due to the low number of samples). Figure 4.12 (b) demonstrating the effect of the added noise on the input of (a), partially masking the classes. For testing each stream is presented for 300ms (dictated by the recording length in the dataset) then some of the locations are changed and the next stream is played.

To help visualise the results of this experiment, a snapshot of the processing in each layer as time progresses is shown in Figure 4.13. Figure 4.13 displays both this asynchronous throughput of activity and how the network reduces the numbers of computations, even when presented with noise and distractors, with the time axis showing an accumulation of spikes to ease with visibility. Figure 4.13 shows that by Conv 1 the added noise is mostly removed as it lacks any real structured shape, but the distractor Garfield remains and progresses onto Pool 1 and Conv 2. During Conv 2 though, due to its low saliency with any of the learned features for the classes of Face or Motorbike the distractor is removed from the processing pipeline. This leaves only the two known classes, which then progress onto the Conv 3 layer and then through the decoder layers to the output, where they are successful segmented. When testing the multi-stream input without any noise the accuracy and IoU value is identical to the single-stream instance at 96.8% and 81%. With added noise, this value sees a slight reduction to 95.1%

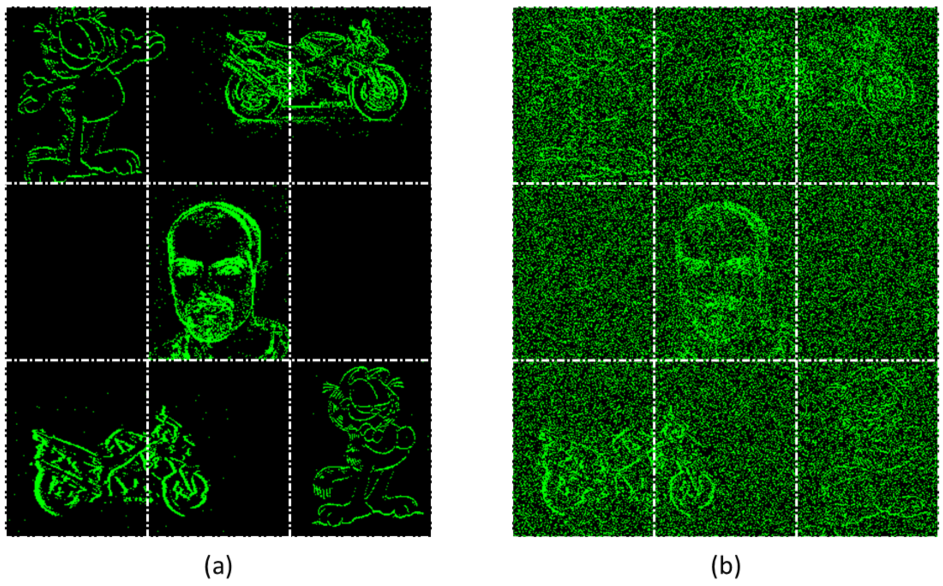


Fig. 4.12: Example of input for the Multi-Stream Input without noise (a) and with noise (b), both with extra gridlines indicating the 3x3 grid which determines the initial location of the inputs.

and 79% for accuracy and IoU, these results are also shown in Table 4.1. The decreases being attributed to the noisy pixels directly contacting or occurring within the class boundary, as the network has no real way to discern this noise from actual data. This is clearly shown within the segmented output comparisons shown in Figure 4.14, where the noiseless output (a) and the noisy (b) show considerable difference in their respective segmentations with far more diagonal lines present in the noisy output (b) in comparison to (a). This outcome could have been predicted and will be highlighted in section 4.4.3 as the first layer of the network has a larger feature representation for the diagonal line when compared to the horizontal and vertical lines, with more pixels allocated to representing the diagonal lines rather than horizontal and vertical, due to the larger variety of edges this feature had to represent. This implies that with the same threshold the diagonal feature is more likely to be activated than the horizontal and vertical.

With the segmentation successfully output, the spiking controller now has less spiking activity so should find it easier to be able to track a given class. The tracking starts once the user has made a selection of which class is to become the attention of the network. Experimentally this was tested by selecting the attention after two successful multi-class segmentation examples where the stream inputs were repositioned. Figure 4.15 displays the outputs of the three inputs (a), (b) and (c) with their subsequent paths to segmentation. Figure 4.15 shows

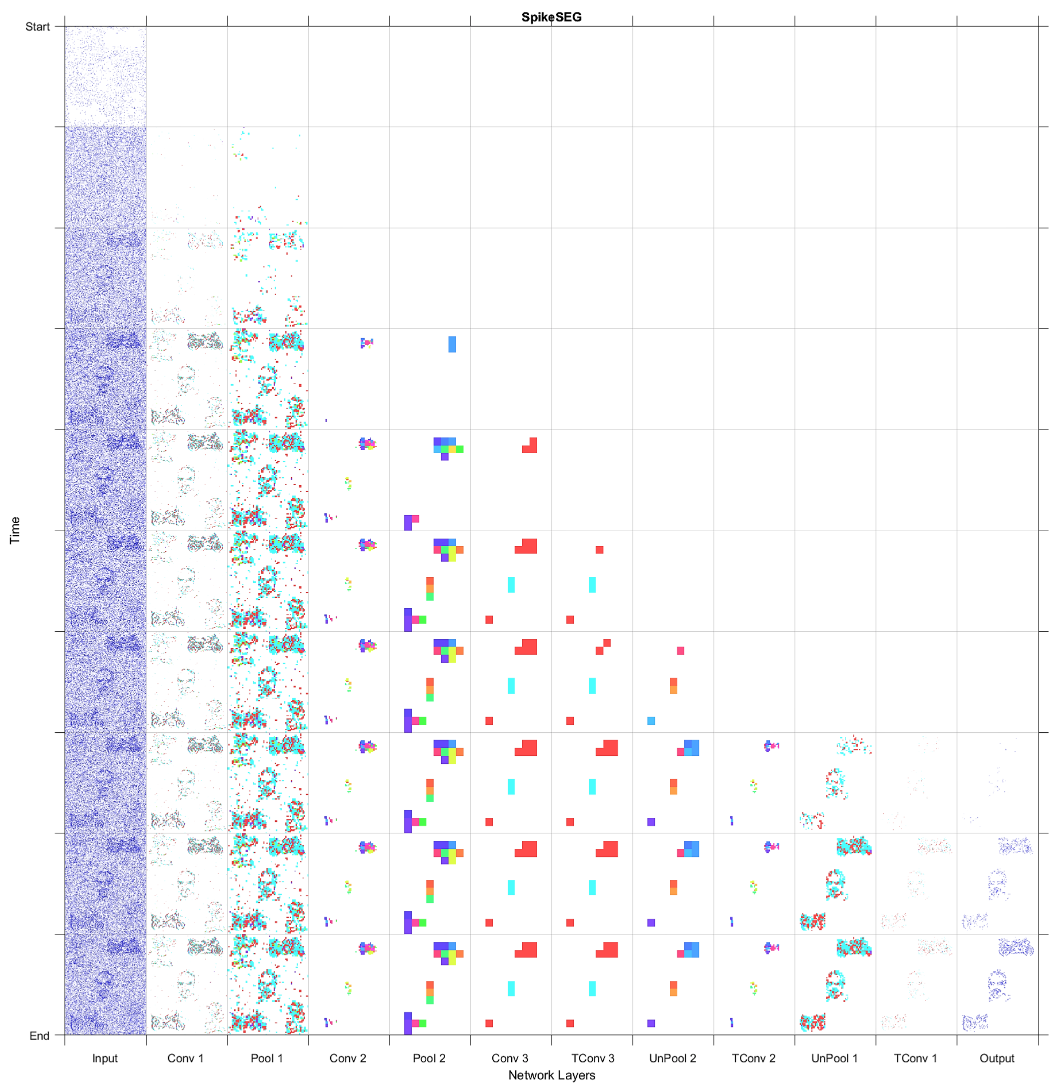


Fig. 4.13: Full Layer-wise spiking activity for the system, showing the progression of spikes through the network encoding then decoding section into the segmentation output

that for inputs (a) and (b) the network is correctly segmenting the input and displaying an output with a highlighted segment displayed in the 3x3 grid. It is only in Fig 4.15 (c) that the guided attention mechanism is triggered causing the inhibition of the other class in the propagation between layer Conv 3 and T-Conv 3. This feature is highlighted with the red circle showing which neurons are now no longer represented in the subsequent layer and thus no longer computed out to the segmentation, highlighting part of the efficiency in SNN. The output section of the diagram in Figure 4.15 (c) highlights the attention of the network being drawn to the face located on the bottom left of the grid, which in the spiking controller would result in an output of left and down to ensure the face is located

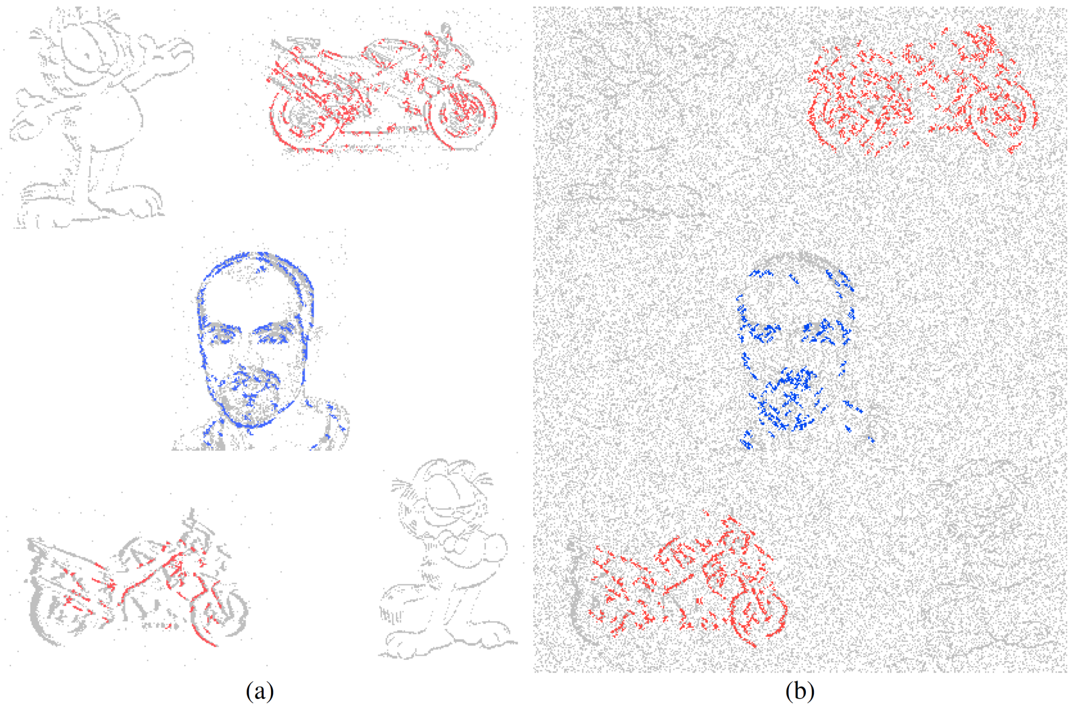


Fig. 4.14: Segmentation overlays for the (a) Multi-Stream Input and (b) Multi-Stream Input with noise, including the classes Face, Motorbike and Garfield from the N-CalTech dataset)

within the central region. The arrow within Figure 4.15 (c) also indicates the movement of the track update, which is based on the central region as within the previous two sequences the multi-class attention doesn't give a control output. This attention-based tracking update is delivered within 34 ms or 34 input steps, which with the 11 ms processing lag with each layer to propagate through the network results in a 31 ms delay within the 300 ms input stream. This may seem like a considerable amount of time, but as shown in both Figures 4.5 and 4.13 due to the way the N-Caltech dataset was recorded, the first 30 ms of the recording contains very little information due to the lack of movement with the main concentration of spiking activity during the middle of each of the saccade movements. To test this the first 30 ms of events were removed from all the input streams which result in a reduction in track update to 15 ms and with the offset of 11 ms to progress through the network means a 4-5 ms latency to get from input to a control output if the processing could be done in real-time. However, even this latency is mainly from the initial delay in spiking activity within the network first layer, suggesting once running the latency would decrease. This would make it a highly competitive alternative or efficient middle ground between highly precise CNNs and low latency edge detection systems. To further investigate the efficiency of the SNN versus an equivalent CNN, the total number of average calculations completed by the SNN is approx 9% of the total calculations from an equivalent CNN. This is when comparing the number of convolution and pooling operations, as a traditional CNN will still propagate forward the calculations from each convolution even if the about of information is minimal. The SNN benefits here due to the sparse nature of both the features and the SNN thresholding processing.

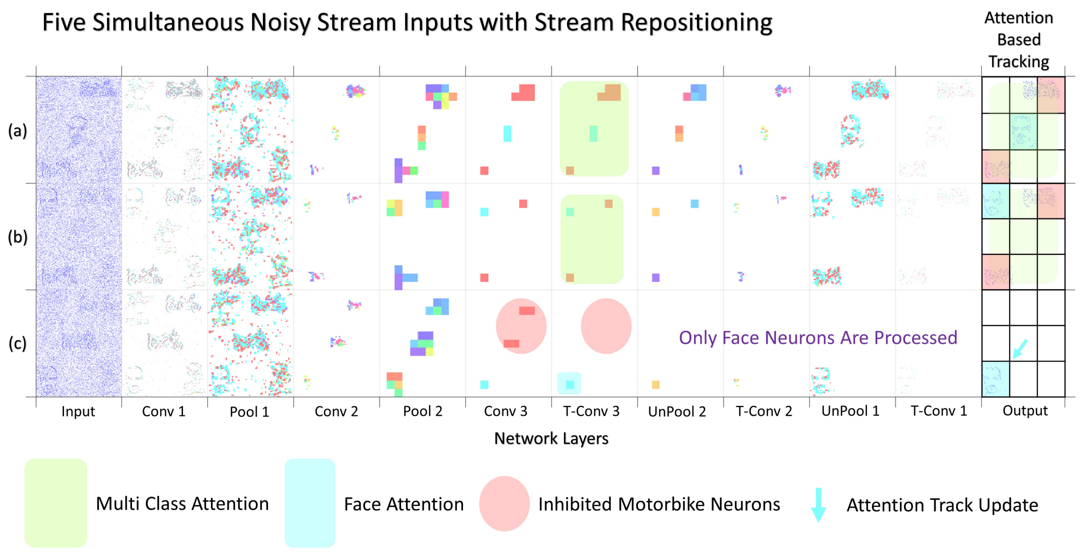


Fig. 4.15: Image showing three separate multi-input data streams. (a) and (b) both representing the full system layer-wise computations when no attention is selected, while (c) shows the layer-wise computation after the Face class has been selected as the attention of the network, thus enabling a simplification of the output and activating the action part of the system with a tracking controller update.

Tracking from Handheld NVS

For this section, the SpikeSEG network was retrained to be able to identify a panda desk ornament and aims to better highlight the control and tracking aspects of the PUA system. The input stream recorded from a NVS has the panda start on the far left in the field of view then the camera pans to the left resulting in the panda being on the far right, with an example of the input images shown in Figure 4.16 (a). The results detail how well the segmentation would work within this example, with the extra complexity of 3D shapes and natural indoor lighting conditions. Overall the results of the 1-second test stream, show that only 60ms (6%) of streaming footage failed to produce a segmentation output. This also occurs at the points where the least amount of movement of the camera happens, the turning points, subsequently producing fewer output spikes. Nevertheless, this results in no actual loss in tracking accuracy as the panda object stayed within the previous segmentations IoU bounding box. Furthermore, the IoU for this test stream was 75%, shown in Table 4.1, perhaps lower than expected given the high level of accuracy within the classification/segmentation process. This is illustrated in Figure 4.16 (a) where the middle section of the panda is not well resolved by the sensor, meaning on occasion the segmentation output was only of the top or bottom section. Figure 4.16 (b), (c) and (d) also show the full system process for the two different control outputs of moving right (c) and left (d), that is when the segmentation area enters the proximity of the spike counter at the edges of the output image. Within Figure 4.16 (d) there is also an example of how the system overcame a background object that could have affected simpler approaches, as originally the input image had a background object on the right-hand side of the image. Due to the feature extraction and segmentation, the background object was unable to influence the controller which without the Understanding-based segmentation would have had spiking activity in both left and right spike counters.

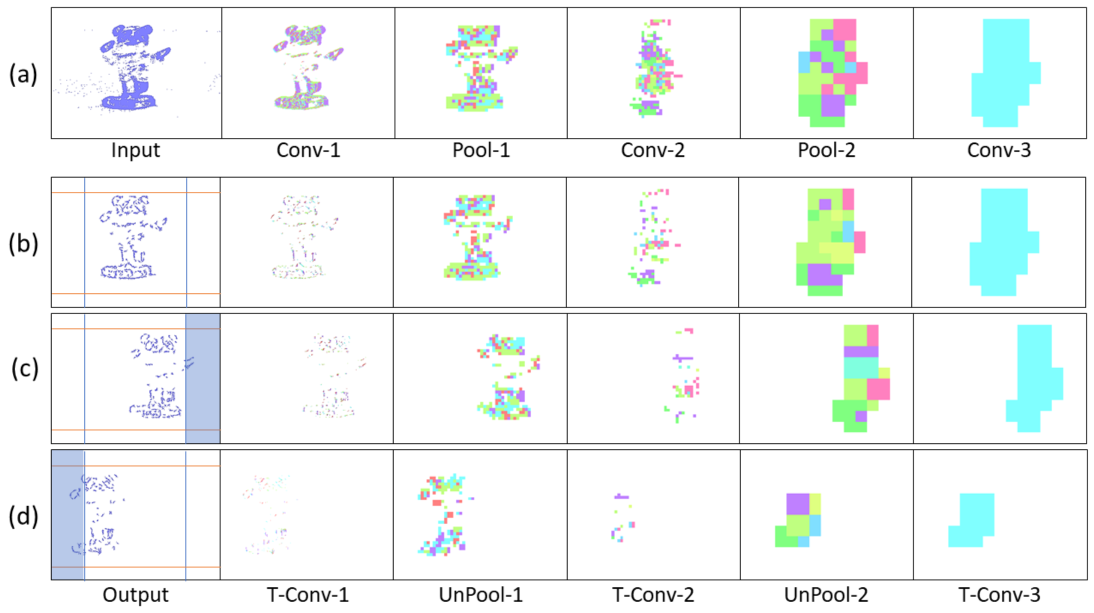


Fig. 4.16: (a) Panda Input Image, (b) Panda Image showing the boundary regions of the controller, (c) reaching rightmost boundary triggering a control action, (d) Panda reaching leftmost boundary triggering a control action.

4.4.3 Robustness and Interpretability

This section highlights two key features of utilising an SNN approach for this framework, the first is system robustness, especially that pertaining to Perception and Understanding (the sensor and processing) and how that affects the Actions (control) of the system. The second feature is that of interpretability something that is not often not associated with CNN type approaches.

Robustness

The added robustness of the PUA approach comes from the Understanding section within the PEAT mechanism. As mentioned in section 4.3.2, the buffering of input spikes allows a spike counter to be implemented, allowing a pre-emptive rather than reactive approach to the thresholding within the network. Permitting synaptic scaling homoeostasis to increase the threshold values on all layers, ensuring noisy or adversarial inputs are mitigated first. Subsequently, if the spike level persists the threshold levels using intrinsic homoeostasis may be adapted. An example of this system at work is illustrated within Figure 4.17, with (a) showing a multi-stream input with no noise, then the input is corrupt with noise in (b), (c) and (d) showing the resulting effects of the noise throughout the system with and without the PEAT mechanism active. The PUA framework implements a regime that giving no output to the controller is better than an incorrect output. As such, it then will suppress any noisy input to the point of no output in the first instance, before adapting the thresholding to allow a correct output. This is opposed to the alternative approach is which to simply just adapt the output to a correct one while it was outputting incorrect values. This robustness feature is highlighted in the output of Fig 4.17 (b) which is incorrect and if passed to the controller could cause an undesired response, meanwhile in Fig 4.17 (c) the PEAT is seen to allow the network to threshold the noise level resulting in no segmentation output. Incidentally, Figure 4.17 (d) could be the adaptive outcomes of both approaches (b) and (c), it is just intermediate control output suppression that adds an extra level of robustness to the system.

Interpretability

The interpretability of a system is often overlooked when values of accuracy or precision appear to be high. But understanding or gaining some insight into how the system got to an answer could be a valuable advantage for SCNN compared to conventional CNNs. As SCNNs trained using STDP happens to produce a sparse

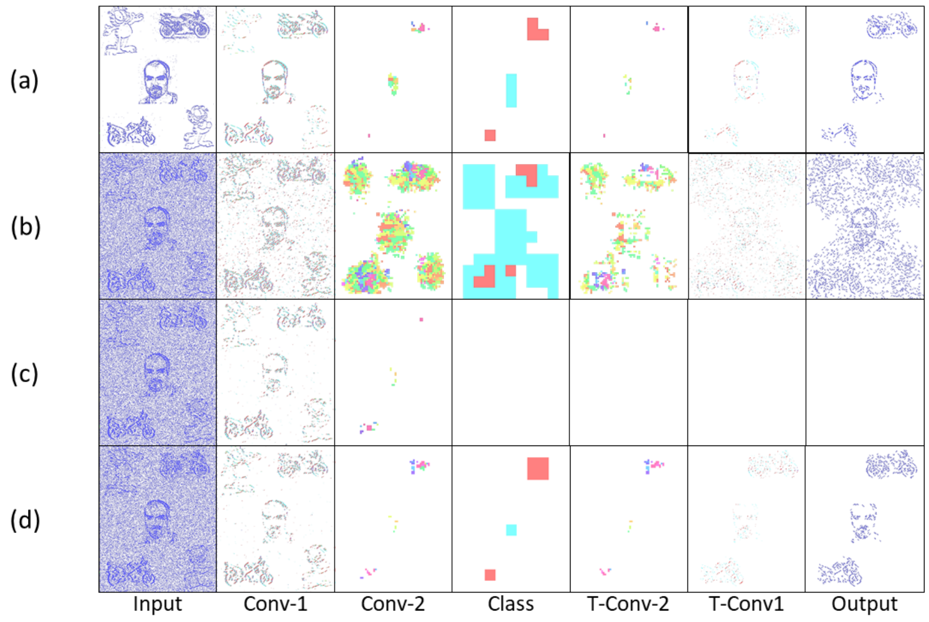


Fig. 4.17: Highlighting the Robust noise suppression with the Pre-Empt then Adapt Thresholding mechanism.

feature variation of typical CNN outputs, the SCNN results in features that are more akin to those from contour matching papers [219] while CNNs typically take on features that resemble textures [220]. These texture maps are often hard to interpret, although modern approaches have found ways to highlight the most salient parts of the input with reference to these texture maps. Nevertheless, it is still often difficult to predict how the system might react to an unknown input. The features that were learned for the testing of the N-Caltech dataset used within this work is shown in Figure 4.18. Figure 4.18 (a), illustrates the differences between the previous version of the model and the current implementation with PEAT and pruning improving the feature extraction, using the same Conv-1 features representing simple edge detection structure of horizontal, vertical and two diagonal lines. Figure 4.18 (a) then shows the mapping of those features onto the weights of the Conv-2 resulting in the features that resemble shapes and objects before the classification stage in Conv-3 (Class Features). It can be seen that half of the 36 features in Conv-2 relate to the Face class and the other half to the Motorbike, with these features helping to build up the classification layers with two features either Face or Motorbike. Figure 4.18 also shows a selection of features from both the Five Class (b) and Ten Class (c) experiments. Top half showing the Conv -2 features and the bottom showing the Class Features. Figure 4.18 (b) Classes shown in Class Features order are Face, Motorbike, In-line Skate, Stop Sign and Watch. Figure 4.18 (c) Classes shown in Class Features order are

Stegosaurus, Watch, Cup, In-line Skate, Windsor Chair, Motorbike, Revolver, Camera, Face and Stop Sign.

This image helps to explain what the network has learned and how it appears to be looking for contour like shapes to help it distinguish between inputs. Along with this insight into how the network operates, it also allows the user to perhaps understand why the network might not always give the correct answers, due to certain feature not be represented in the learned features. Similar to how if creating a system using hand-crafted contours features, you would understand the limitation this allows, this method could provide a similar understanding to be had. This could allow manual manipulation of features or manual pruning throughout the training if the user happens to have expert knowledge of the task, bringing neural networks closer to known computer vision-based techniques, which could provide an interesting overlap, especially in the robotics domain.

In order to perceive how the additional classes affect the interpretability of the system Figure 4.18 (b) and (c) highlights a sub-selection of the features within the 5 and 10 class models. This highlights how the interpretability is still there for some of the features while others have become more difficult to understand, perhaps due to overlapping features from two classes. Overall, Figure 4.18 (b) and (c) highlight how reviewing of the features within a Spiking Neural Network can help to gain understanding about parts of the network, with the classification layers features representing each of the 5 and 10 classes. The visualisations help to explain why certain classes might struggle versus others due to similar sub-classification features.

4.4.4 Comparison with CNN

This presents the differences between the spiking and non-spiking versions of this network architecture. This baseline acts to show the advantage that can be gained with a SNN, especially in the context of processing Neuromorphic data over a CNN. Firstly testing the accuracy between the CNN and SNN, the SNN beating the equivalent CNN by at least 20% for each of the 2, 5 and 10 class tests, as seen in Table 4.2. Where the CNN was trained using a supervised method and having a fully connected softmax output layer. These results are in line with expectations of the performance of some CNNs with Neuromorphic data [221], where the sparse input data can cause issues with learning useful latent representations. This results in a VGG16 [59] network only achieving 78% compared to 91.7% on the original Caltech 101 dataset [31], where the state of the art in this dataset is set at around 97% [222]. This allows an idea of the increase in difficulty the CNN type networks appear to have when using neuromorphic data. Another conventionally compared metric within the deep learning community when comparing two models is the number of parameters. This is a good indicator of the complexity and amount of computational throughput required to process. In this aspect, the network as shown in Figure 4.4 has the same number of parameters in both the SNN and CNN version as the two networks would be identical. With the number of feature maps in the final classification convolution layer making a significant difference in the number of parameters, 2 class with 26,121 parameters, 5 class with 138,189 parameters and 10 classes with 296,274 parameters, with these results being collated in Table 4.2

Typically along with the number of parameters, the number of actual calculations is stated for deep learning networks, as this is a better indicator of actual computational power required to use the network. This metric normally comes in the form of number of multiplies and accumulates (additions) operations (MAC). The MAC operation is a common step that computes the product of two numbers and adds that product to an accumulator, essentially the operation carried out many hundreds or thousands of times in a neural network. As the network size increases, this value increases linearly following this MAC equation

$$MAC_{layer} = K_{cur}^2 C_{in} H_{out} W_{out} C_{out} \tag{4.8}$$

where the number of MACs for each layer is the product of the Kernel size, K. The number of kernel maps or channels from the previous layer or input, C_{in}. The height and width of the output after the convolution layer, H_{out} W_{out}, so also depends on padding methodology. Lastly, the MAC also depends on the number of feature maps in the next layer C_{out}, so then in this fully convolutional network where the last convolution acts as a classifier, the MACs depend on the number of output classes. From the testing of 2, 5 and 10 classes within this network, MACs of 44 million, 106 million and 211 million respectively were calculated as the number typically run with an unoptimised CNN version of the network, also shown in Table 1. The equivalent number of MACs required by an SNN would be directly the number of spikes that need to be processed, as these are the only parts of the kernels that are being multiplied then accumulated. This results in a massive reduction in computational cost as with the same 2, 5 and 10 classes the number of MACs required by the SNNs are 8.2 thousand, 8.1 thousand and 7.8 thousand respectively, shown in Table 4.2. Where as the number of classes is scaled up the number of calculation reduced thanks to the inhibitive nature of the SNN. However, typically the number should be equivalent if only the class number

Table. 4.2: Comparison of the calculations required by the CNN and SNN for the same network.

No. Classes	Accuracy CNN—SNN (%)	Parameters	CNN MACs	SNN Spikes	Quotient (CNN/SNN)
2	72.5 — 96.8	26k	44M	8.2k	5.4k
5	56.3 — 86	138k	106.5M	8.1k	13.1k
10	48.1 —75	296k	210.7M	7.8k	27k

change and would increase if it was to see an increase in feature maps used. This CNN to SNN reduction is then over 5000 times for the 2 class, over 13000 times for 5 classes and over 27000 times for the 10 class, shown in Table 4.2. This massive reduction in the number of MACs highlights how much more efficient the SNN can be. Even with optimisation for the CNNs that would potentially save 99%, this would still leave a 500 to 3000 times increase over the SNN. This exploitation for sparse sensing and processing is what leads to the potentially massive savings in terms of Size, Weight and Power (SWaP), which is an important factor in many deployed systems. The reduction in MAC processing unit able to reduce the silicone size of processing systems, or the vast reduction in processing instances would reduce the overall power consumption of the system. Ultimately the SNN is able to process images with an average spike count per layer being: Layer 1 - 2534 spikes to capture all the edge information. Layer 2 - 72 spikes representing shapes or complex patterns and Layer 3 - 2.87 classification spikes, typically to cover the full original pixel space of the identified class.

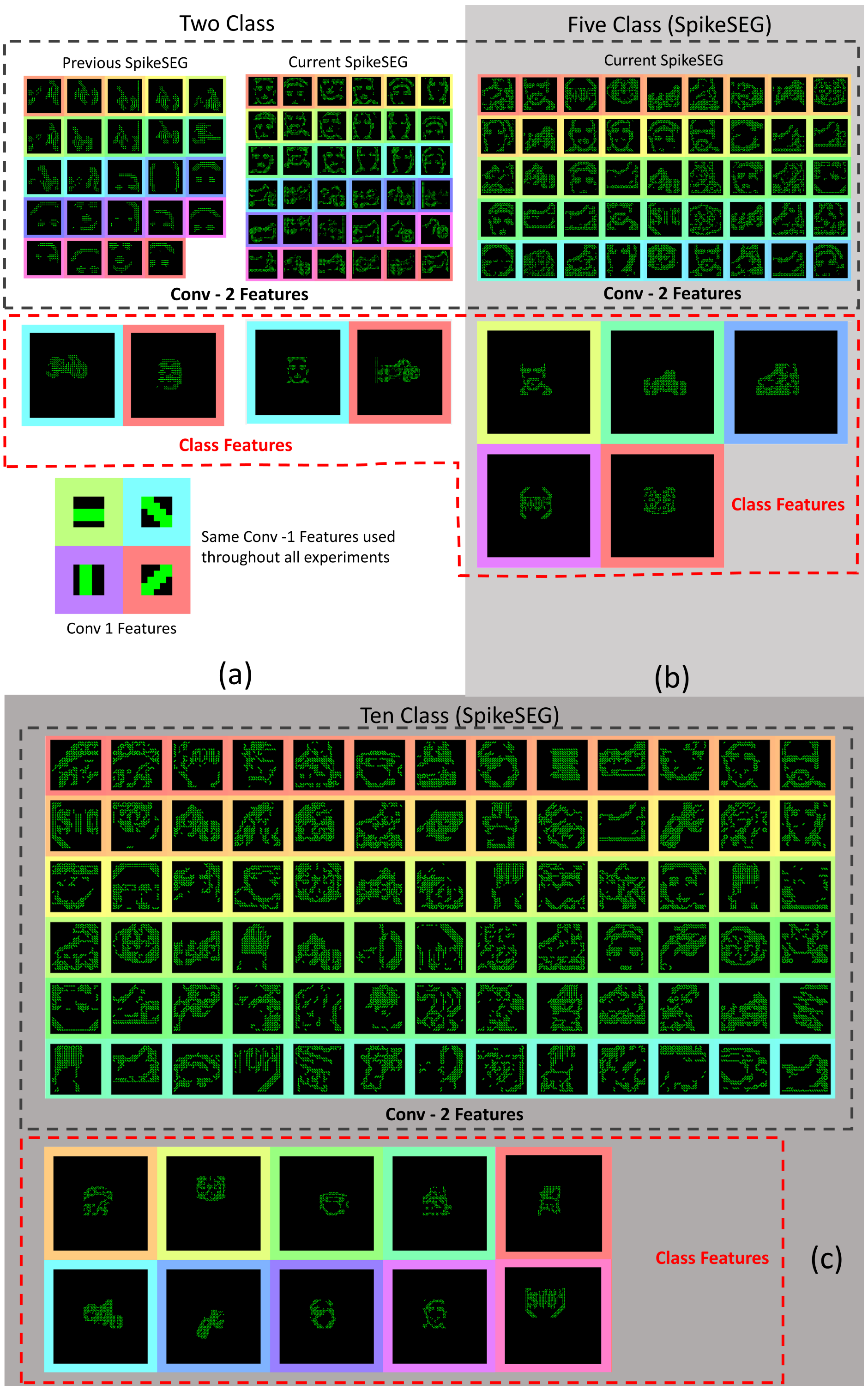


Fig. 4.18: (a) Features map representations of the convolution layers, with colouring to match the latent space representation from the two class experiment, showing prior and current results of the Conv -2 features and Class features.

4.5 Discussion

The understanding method shown in this work details an unsupervised STDP approach. To fully utilise the spiking nature of the processing it is paired with the perception method of a spiking input sensor. Together this perception understanding pair can successfully semantically segment up to 10 classes of the N-Caltech dataset. The output of this process is a spiking grid indicating the location of the class within the scene, which can be interpreted by the action system to allow the objects to be followed if attempting to leave the field of view.

The full PUA process is completed in a fully spiking and convolutional manner. This ensures all calculations are either spiking or spike counting. Allowing the network to maintain the temporal and processing advantages, along with the asynchronicity associated with neuromorphic vision sensors, from input to output. However, this method of processing is not without its drawbacks, as there is an overall decrease in accuracy associated with this addition of extra classes. That perhaps indicates the limitation of this unsupervised method in terms of problem scaling. For instances with the 100 classes available within the N-Caltech dataset, the system would only be able to learn the most common features that occur within each class, but only if they present a large enough variance. That is, it will only learn common class features as long as they look different enough from the other classes. That is essentially what can be seen happening with the 5 and 10 class experiments visualised in Figure 4.18 and Figure 4.11 (c). Figure 4.11 (c) highlights that even with a high intra-class variance the kernels sometimes learn differentiating features from all other classes, while other times learns features that are an amalgamation of two or more classes. The 5 class experiment displays this most prominently with the Bike and In-Line Skate classes, as there are similarities between the outline shape of both objects.

Nevertheless, this ability to find the most common features that express the highest variance from others is both the limitation and strength of this STDP approach. Limiting in that this approach might not scale to larger datasets, but a strength in that it made the network asynchronous, adaptable, computational sparse and visually interpretable. This highlights that the STDP method used might not be suitable for all problems, but serves as an indication of the benefits if the problem is appropriate. This work demonstrates that STDP alone can be used to find the most common features of a dataset. Which in turn, can be used to successfully perform image classification and semantic segmentation. However, a further learning rule to help focus on more discriminative features such as Reinforcement-STDP (R-STDP) [182, 223, 224] would be a useful extension.

This could help in tackling the main issue of inter-class to intra-class variance differentiation. This could allow not only the most common feature to be discovered but the most common discriminative feature.

4.6 Conclusion

We proposed a new spiking-based system, the Perception Understanding Action Framework, which aims to exploit the low latency and sparse characteristic of the NVS in a fully neuromorphic asynchronous event-driven pipeline. Using the understanding gained through the SpikeSEG segmentation, the network is able to detect, classify and segment classes with high accuracy and precision. Then from this understanding, the system makes a more informed decision about what action is to be taken. In this context, the framework was able to show a semantic class tracking ability that combines the feature extraction capability of CNNs and low latency and computation throughput of line and corner detection methods. The framework also explores the unique benefits that can be gained through utilising SNNs with interpretability and robustness, with the use of thresholding algorithms and sparse feature extractions. The PUA framework also shows off the unique attention mechanism, emphasizing how simple local inhibition rules when combined with an encoder-decoder structure; can help reduce the computation overhead of the semantic segmentation process. This research highlighted the series of benefits when utilising a fully neuromorphic approach with a pragmatic engineering and robotics outlook, looking at the biologically inspired mechanisms, features and benefits, then combining them with modern deep learning-based structures.

Chapter 5

Spiking Instance Segmentation with STDP

5.1 Introduction

In Chapter 4 an unsupervised version of semantic segmentation was realised using only the STDP rules and a fully convolutional encoding-decoding network structure. This semantic segmentation builds upon the image classification and localisation inferred from the active regions within the SCNN structure. However, this is unable to achieve object detection and as such is unable to discriminate between multiple instances of a class within a scene.

This chapter presents both the Hierarchical Unravelling of Linked Kernels (HULK) and Similarity Matching through Active Spike Hashing (SMASH) methods. Aimed to help identify these individual instances of a class by looking at the internal spiking activity of the network, that is “What” features spike and “When”. This highlights the STDP unsupervised methods cannot only be used to cluster similar looking images into pseudo-classes as seen in Chapter 4 but also help to further differentiate between those classes into individual objects. All while being interpretable enough to gain understanding as to how it achieves this. This novel method aims to utilise the information that the network has already extracted within the latent space. This idea stems from the realisation that visually the features learned within the network appear to represent different subgroups from the classes of the dataset.

This chapter is organised as follows. In section 5.2 A systematic overview of how the HULK SMASH algorithm works is presented. Section 5.2.1 details how the spiking activity is recorded for use in similarity checking, while section 5.2.2 explains how this similarity measure is determined. Section 5.3 details the results of the experimental work carried out and its useful implementations. Section 5.4 gives a guide of how intuition and understanding is gained through visualisation. Lastly, section 5.3 concludes the chapter.

5.2 Temporal Spike Matching

The idea behind the HULK SMASH method is to allow object detection and instance segmentation, using the already learned features from the STDP learning as described in Chapter 4. However, instead of using a regression process to indicate which classification instance are connected, as seen with DL models reviewed in Section 2.5, the proximity and temporal-featural (time- and feature-based) similarity is used. As illustrated in Figure 5.1 this method of temporal spike matching for instance segmentation can be broken down into two main parts, the *Intra-* and *Inter-Sequence Processing*, in other words, what is happening internally and externally throughout the process. The intra-sequence is where most of the processing happens, while the inter allows the processing to link to other instance of the process running e.g. to compare object instance to see if they are the same object. This can be seen in a block diagram of the proposed method in Figure 5.1, within the green and red dot-dashed boxes.

Figure 5.1 highlights that the *intra-sequence Process* starts at the intersection of the SpikeSEG network from Chapter 4. The first step of the process is to individually process each of the spiking instances from the classification layer of the SpikeSEG network. This is in contrast to the original network which just grouped all instance based on class, while this method takes each instance within each class. This process of decoding each instance individually is what is referred to as the HULK process. Once each classification spike has been decoded back into the original pixel space, this can then be feed to the SMASH process. This process in parallel calculates the bounding box of each instance S_{BB} in the pixel space by taking the max and min value in the x and y coordinates. It then uses this bounding box as a *Proximity Score* check with every other instance bounding box $S_{BB'}$ within the *intra-sequence* with the Jaccard Index [216] $J(S_{BB}, S_{BB'})$. Meanwhile, the other process is performing the *Active Spike Hashing (ASH)* part of the SMASH process. Which stores 2D featural-temporal data S_{ci} from the 4D spatial-featural-temporal decoding class instance processing (X, Y, Map and Time similar to that of the SAM seen in Section 4.3.2). This then passes onto the *Similarity Matching (SM)* phase, where a similarity score is given to the featural-temporal data of every other instance $S_{ci'}$ within the *intra-sequence* again with the Jaccard index $J(S_{ci}, S_{ci'})$. The combination of both the Similarity and Proximity Score gives the SMASH score $SMASH_{ci,BB}$. This score indicates if the S_{ci} and S_{BB} of that particular class instance match any other $S_{ci'}$ and $S_{BB'}$. Resulting in an outcome of whether the instance is part of the same *Object* $SMASH_{co}$ or a different one $SMASH_{co'}$. This intra-sequence process is run in parallel once all the instances have their ASH and bounding box process complete. This is run for every input sequence of data so in this case the same 10ms of NVS N-Caltech101 [207] data is used throughout Chapter 4 and described in Section 4.3.1. This allows both object detection and instance segmentation to be performed on this spiking input image and could be used to further the PUA framework of Chapter 4.

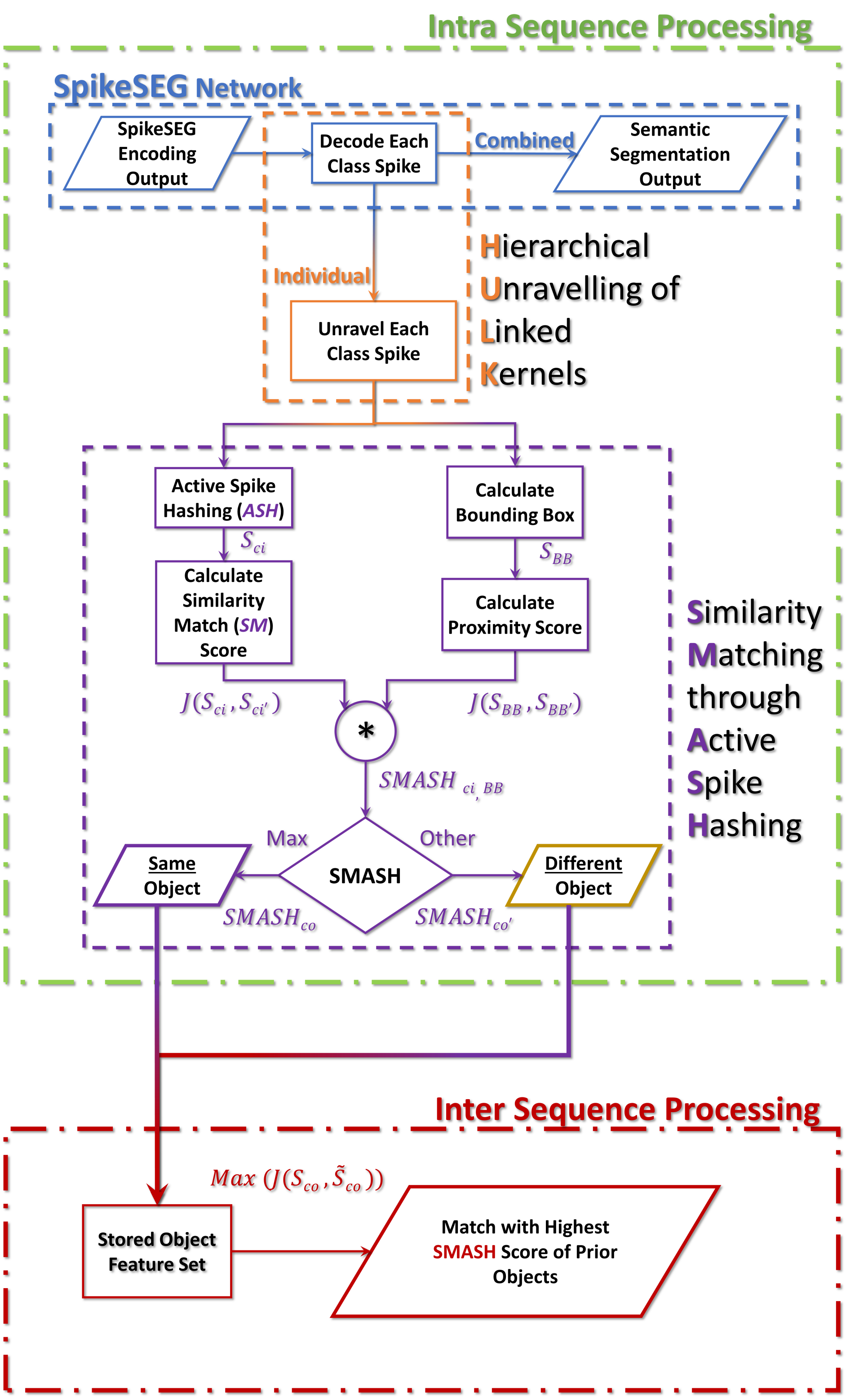

Intra Sequence Processing
SpikeSEG Network
SpikeSEG Encoding Output
Decode Each Class Spike
Combined
Semantic Segmentation Output
Individual
Unravel Each Class Spike
Hierarchical Unravelling of Linked Kernels
Active Spike Hashing (ASH)
S_{ci}
Calculate Similarity Match (SM) Score
Calculate Bounding Box
S_{BB}
Calculate Proximity Score
$J(S_{ci}, S_{ci'})$
$J(S_{BB}, S_{BB'})$
*
$SMASH_{ci,BB}$
Max
Other
Same Object
SMASH
Different Object
$SMASH_{co}$
$SMASH_{co'}$
Similarity Matching through Active Spike Hashing
Inter Sequence Processing
$Max(J(S_{co}, \tilde{S}_{co}))$
Stored Object Feature Set
Match with Highest SMASH Score of Prior Objects

Fig. 5.1: Block Diagram of the HULK SMASH system and where it intersects with the SpikeSEG Network

5.2.1 Hierarchical Unravelling of Linked Kernels

The Hierarchical Unravelling of Linked Kernels (HULK) process, permits spiking activity from the classification convolution layer, to be tracked as it propagates through the decoding layers of the network. This tracked propagation allows a record of each subsequent layer's spatial (x, y), featural (m) and temporal (t) spiking activity. This process runs in parallel with the transposed convolution process, which itself is mapping each active spike in a layer to the subsequent layer. However, after each layer transform a record of each spike that is permitted under the SAM of the SpikeSEG network is stored. This allows a causal hierarchical map to be created tracing each classification spike back to the original pixel space. This process is illustrated within Figure 5.2, with a decoded sequence shown called "Accumulated Decoded Spiking Activity" and the individual class spike breakdown from this are shown within Instance A through D. Each instance, in this case, is representing a single spiking pixel from the classification layer. Whereas in Chapter 4, all the instance belonging to one class are treated as the same entity and decoded. Figure 5.2 is then able to highlight how the class instance is broken down into its individual instances, where each instance provides enough information to recreate the face in the output pixel space. Although, it is clear that some of the instances favour certain features over others. It is through this process of unravelling the classification spiking activity that permits the ensuing similarity matching process.

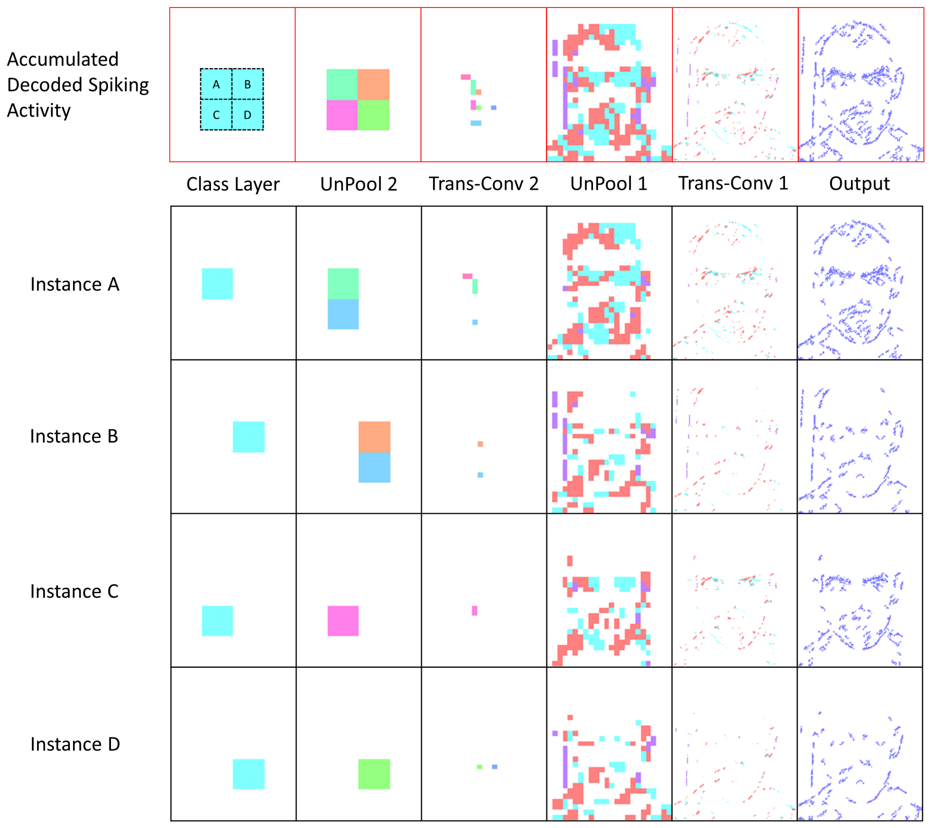


Fig. 5.2: Illustration of the individual class spikes being unravelled, indicating activity caused by these class spikes

5.2.2 Similarity Matching through Active Spike Hashing

The Active Spike Hashing (ASH) is the process of taking the recorded spiking activity and implementing an efficient and effective way to store the sparse 4 dimensional spatial (x, y), featural (m) and temporal (t) values. This is done by realising that the convolution structure of the SCNN SpikeSEG, is already dealing with the translational invariance and spatial dimension, together with the bounding box proximity score. It can also be noted that position within the scene is not a useful evaluation metric of the similarity between two objects. The ASH process then results in a 2D featural-temporal hashing of the spiking activity. The featural data then has the same total number of features as the network, in this case, is 41. This value comes from the use of a network with a similar structure as the Face-Motorbike Network used in Chapter 4, previously shown in Figure 4.4, except with only one classification layer to start. The other 40 features come from the 36 from Trans-Conv2 and 4 from Trans-Conv1.

A further memory reduction to the ASH process is permitted by storing the values as binary terms. As the number of spikes recorded within each map per timestep is more a measure of the total spiking activity, rather than the featural-temporal features we evaluate with the similarity measure.

An illustration of the ASH process where the spiking activity is assigned its feature map and time-stamp index (f,t) can be seen in Figure 5.3. Figure 5.3 builds upon the HULK process previously seen in Figure 5.2 with the indices of the spiking activity now visible in Figure 5.3 for each convolutional layer. Each active neuron is being assigned an f and t index on a class-instance basis. Allowing a 2D matrix of each instance to be realised where from Figure 5.3, Instance A would have 1s in the first column (time) forth row (feature map) for the classification layer activity. Meanwhile, the green coloured features of Instance A's Trans-Conv 2, are stored with row 19 and column 3 and 5. The ASH process typically reduces the size of the tensor by 98%, reducing the memory overhead considerably.

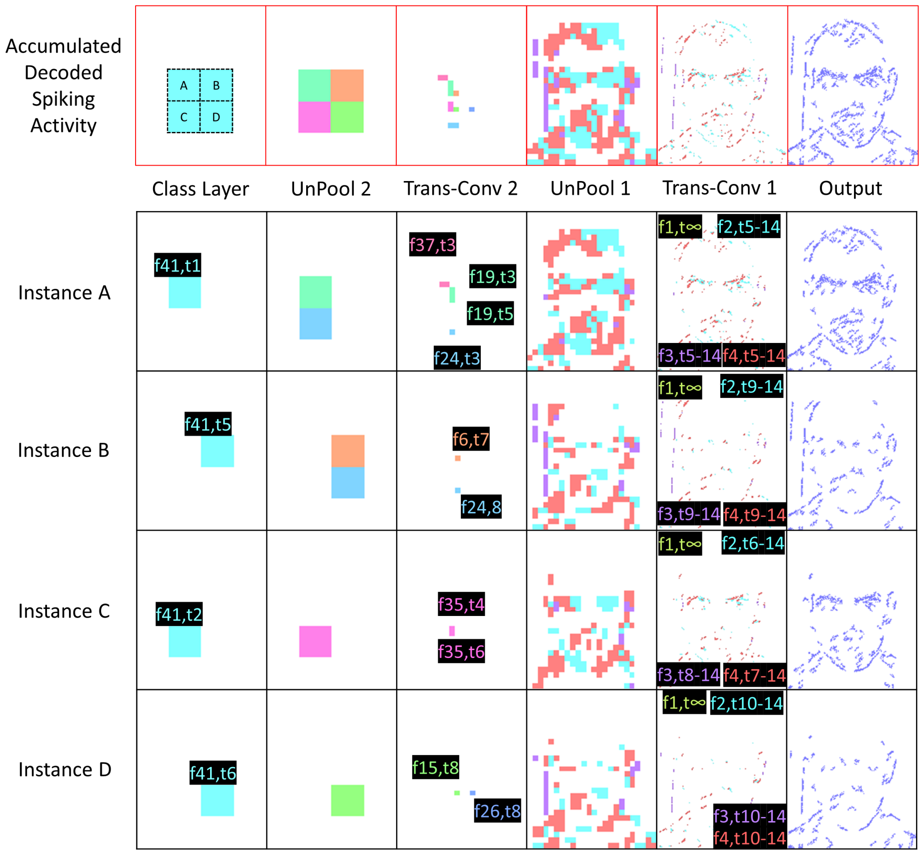


Fig. 5.3: Further explanation of the Active Spike Hashing mechanism from the HULK process, feature map and time given to each active neurons

Once the spiking activity is hashed it is ready for the Similarity Matching section of SMASH. The Jaccard similarity coefficient was once again utilised, as previously seen in Section 4.4 within the intersection over union used to calculate the segmentation bounding box overlap. However, within this instance, the measure is used to see how much similarity there is from each instance in the intra-sequence process. Comparing the featural-temporal similarity of each instances spiking activity, compared to the overall amount of spiking activity [216].

$$J(S_{Ci}, S_{Ci'}) = \frac{|S_{Ci} \cap S_{Ci'}|}{|S_{Ci} \cup S_{Ci'}|} \tag{5.1}$$

where S_{Ci} is the spiking activity of that class instance and $S_{Ci'}$, represents the spiking activity of any other class instance for that sequence. However, due to the ASH process storing binary values of the activities, this equation can be simplified down to a logical calculation performed with only ORs and ANDs. This allows the quick comparison of the number of spikes that feature in both the current instance *and* the comparison, divided by the number of spikes that feature in the current instance *or* the comparison.

$$J(S_{Ci}, S_{Ci'}) = \frac{S_{Ci} \land S_{Ci'}}{S_{Ci} \lor S_{Ci'}} \tag{5.2}$$

To complete the intra-sequence of the SMASH process the bounding box IoU score must be calculated as seen previously in section 4.4, with the modifier being that it is now computing the IoU for each class instance bounding box Ci_{BB} against the other instances bounding boxes $Ci_{BB'}$, instead of for each class in total

$$J(Ci_{BB}, Ci_{BB'}) = \frac{|Ci_{BB} \cap Ci_{BB'}|}{|Ci_{BB} \cup Ci_{BB'}|} \tag{5.3}$$

Multiplication of the similarity score, with the IoU, results in the novel proposed SMASH score for each instance

$$\text{SMASH}(Ci, Ci') = J(S_{Ci}, S_{Ci'}) \times J(Ci_{BB}, Ci_{BB'}) \tag{5.4}$$

Once the SMASH score is calculated for each instance, the maximum of each instance is assigned to a class object Co is then

$$S_{Co} = \arg\max_{Ci}(\text{SMASH}(Ci, Ci')) \tag{5.5}$$

This maximum SMASH score is based on the pairing Ci, Ci', where each object S_{Co} is compiled from any overlapping parings e.g. for 5 instances the argmax presents the following sets of Ci, Ci' pairs, (1-3), (2-4), (3-5), (4-2), (5-3). The values 2 and 4 are assigned to one object, while 1, 3 and 5, as 3 and 5 are matched pairs and 1 is associated with 3, so becomes part of that object. It is these class objects S_{Co}, that are used within the inter-sequence processing. This permits class objects from preceding sequences $\tilde{S_{Co}}$, to be compared with current ones. Again looking for similarity with the binary Jaccard coefficient

$$J(S_{Co}, \tilde{S_{Co}}) = \arg\max_{S_{Co}} \frac{S_{Co} \wedge \tilde{S_{Co}}}{S_{Co} \vee \tilde{S_{Co}}} \tag{5.6}$$

These class objects within the inter-sequence then allow sequence to sequence continuity, thus allowing tracking of objects permitted that they maintain a level of feature similarity.

As the original SpikeSEG network outputs a semantic segmentation output, the HULK and SMASH processes are supplementary to this process in transforming the semantic regions into instance-based objects. Based on the block diagram seen in Figure 5.1, the pseudo-code for the SMASH process is provided within Algorithm 1. This pseudo-code allows further insight to the method for comparison both intra- and inter-sequence and serves to compliment the block diagram seen in Figure 5.1, but with the addition on the internal functions used to calculate the values.

Algorithm 1 SMASH

```
1:  procedure INTRA-SEQUENCE($a, b$)
2:      for each input sequence do
3:          for each spiking instance in the classification layer do
4:              perform HULK for each instance
5:              compute max and min, x and y values
6:              compute the $[X, Y, W, H]$ bounding boxes
7:              perform ASH to create $S_{Ci}$
8:              return $S_{Ci}$
9:          for each hashed instance, $S_{Ci}$ do
10:             compute Similarity score against other instances, $S_{Ci'}$ as in (5.2)
11:             compute Bounding Box IoU score against other instances $S_{Ci'}$ as in
    (5.3)
12:             compute the SMASH score as in (5.4)
13:             assign max SMASH score instance pair to object $S_{Co}$ as in (5.5)
14:             return $Listof S_{Co}$
15: procedure INTER-SEQUENCE($a, b$)
16:     for each class object $S_{Co}$ do
17:         compute Similarity score against previous class objects $\tilde{S_{Co}}$ as in (5.6)
18:         return $max(J(S_{Co}, \tilde{S_{Co}}))$
```

From the pseudo-code it can be determined that the SMASH method has two conditions before it concludes that instances are of the same object:

1. That the two instances must have overlapping bounding boxes
2. That the two instances must share a featural-temporal data

Without either of these, the multiplication of the similarity score and intersection over union will result in 0 and determine that the two instances are different objects. All instances that combine into the same object are then stored as one set of features and used at the inter-sequence stage to track objects over multiple input sequences.

The HULK and SMASH processes are able to build upon the semantic segmentation abilities of the SpikeSEG network, allowing a further progression of the Understanding model of the PUA system as seen in chapter 4.3.2. Giving object detection and instance segmentation capability, without the use of a regression sequence. Instead, this method opts for locality and similarity of the most salient features, utilising the information already available from the SCNN process.

5.3 Experimental Results

In the evaluation of the performance of the novel SMASH process presented in this chapter, a number of tests have been carried out. The tests include:

1. Semantic to instance segmentation with object detection
2. Sequence to sequence tracking of objects with object occlusion and recovery
3. Similarity matching for intra-class grouping (feature detection)

These tests are all carried out with the N-Caltech Dataset [207] as seen previously in chapter 4 and section 4.3.1 mostly utilising the Face category. This Face subset provides a subtle yet distinct amount of intra-class variance allowing the successful extraction of multiple diverse face life features within the second convolution layer. Which then provides a robust and general face detection within the classification layer. The training was carried out using 10 of the different people within the subset, with testing being carried out on the same 10 people however with different input sequences used for each. A total of 3 input sequences are used in training, each containing 300ms of spiking activity, so 30 buffered inputs (3 of each of the 10 people), for a total of 900 input sequences. Testing was carried out using 2 input sequences, so 600 buffered inputs. The network parameters are set to be the same as within chapter 4, with the only difference being 36 features available for the one class present. This was to limit the number of external factors and focus the testing on the intra-classification abilities rather than inter classification which was discussed within chapter 4. To further validate the SMASH method the 5 and 10 class SpikeSEG network from chapter 4 was utilised. These tests helps to validate the SMASH method in a more complex feature similarity environment. Where each class has less representation in layer Conv 2, as these networks utilised 16 features times the number of classes, 80 and 160 for the 5 and 10 classes respectively.

5.3.1 Semantic to Instance Segmentation with Object Detection

A series of multiple input streams are given as input to the SpikeSEG network, illustrated in Figure 5.4. The network then creates both the semantic segmentation output and via the HULK-SMASH process, an object detection/instance segmentation output. The input sequence is shown in Figure 5.4 (a), with 5 input stream presented at the same times, 3 with Faces and 2 without. Figure 5.4 (b) highlights the semantic segmentation output of the SpikeSEG network, while Figure 5.4 (c) demonstrates the HULK process extracting each instance from the semantic representation. Figure 5.4 (d) illustrates how the SMASH process groups instances that score above 0 into class instance objects. Lastly Figure 5.4 (e) presents the final instance segmentation output, thanks to the object detection separating the semantic representation.

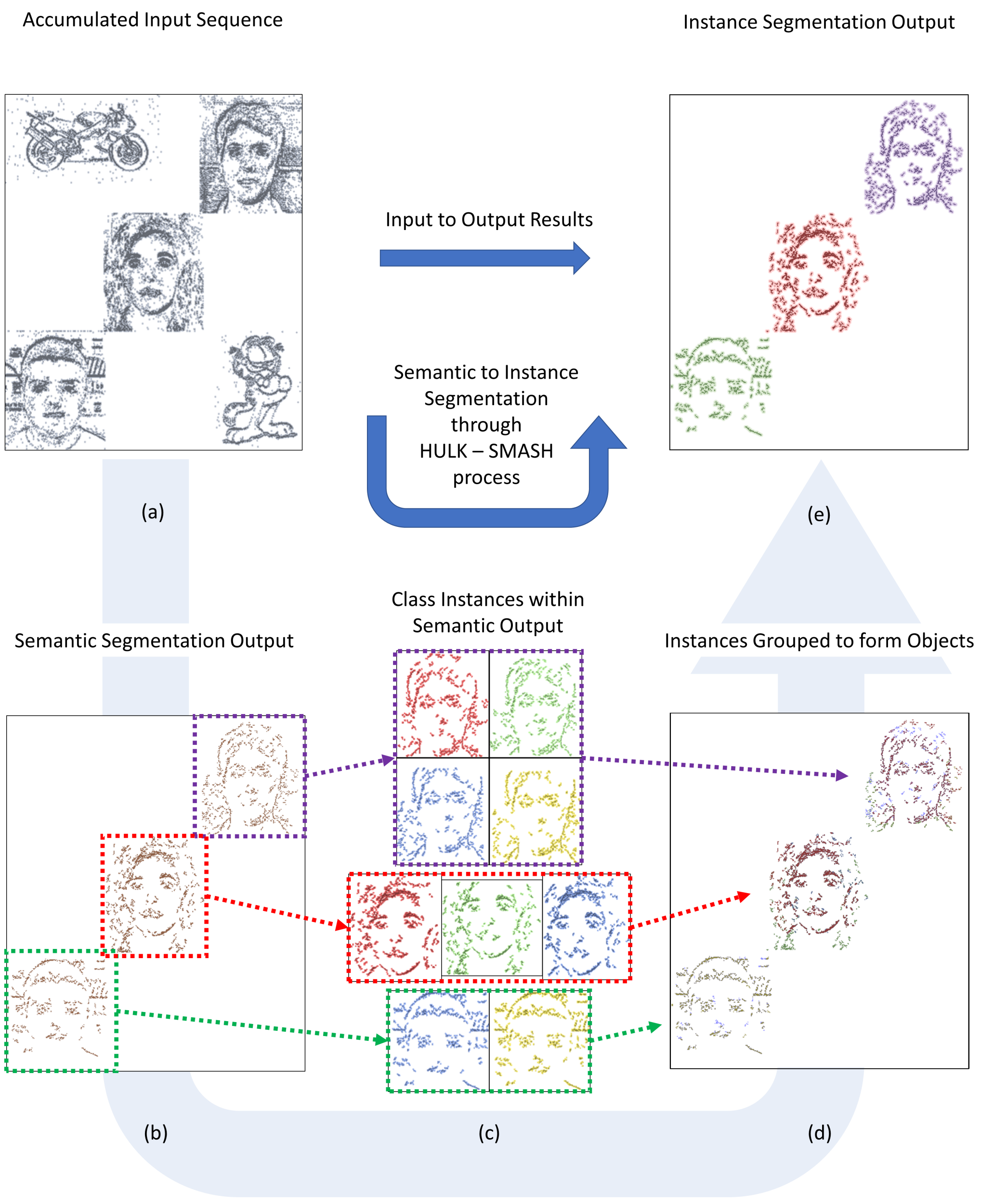

Accumulated Input Sequence
Instance Segmentation Output
Input to Output Results
Semantic to Instance Segmentation through HULK – SMASH process
(a)
(e)
Class Instances within Semantic Output
Semantic Segmentation Output
Instances Grouped to form Objects
(b)
(c)
(d)

Fig. 5.4: Processing from Semantic Segmentation to Instance Segmentation: (a) the accumulated input sequence, (b) the semantic segmentation output, (c) the class instances from the semantic segmentation, (d) the grouped class objects and (e) the instance segmentation output

Within this scenario shown in Figure 5.4, the separate input streams are not overlapping, consequently meaning the bounding boxes between input stream also do not overlap. This means the SMASH score between the input streams will always be 0 regardless of the similarity score. This is partially the intention of

Table. 5.1: Results from the experimental testing split into 4 sections covering the Object detection, detection with occlusion, occlusion recovery and object self matching

	Object Detection Accuracy (%)		
	per Input	**per Sequence**	
Multistream N-CalTech (Face)	97.33	100	
Multistream N-CalTech (5 Class)	95.17	100	
Multistream N-CalTech (10 Class)	91.67	100	

	Detection Accuracy with rate of Occlusion (%)		
	5%	**25%**	**50%**
Multistream N-CalTech (Face)	97.33	62	28

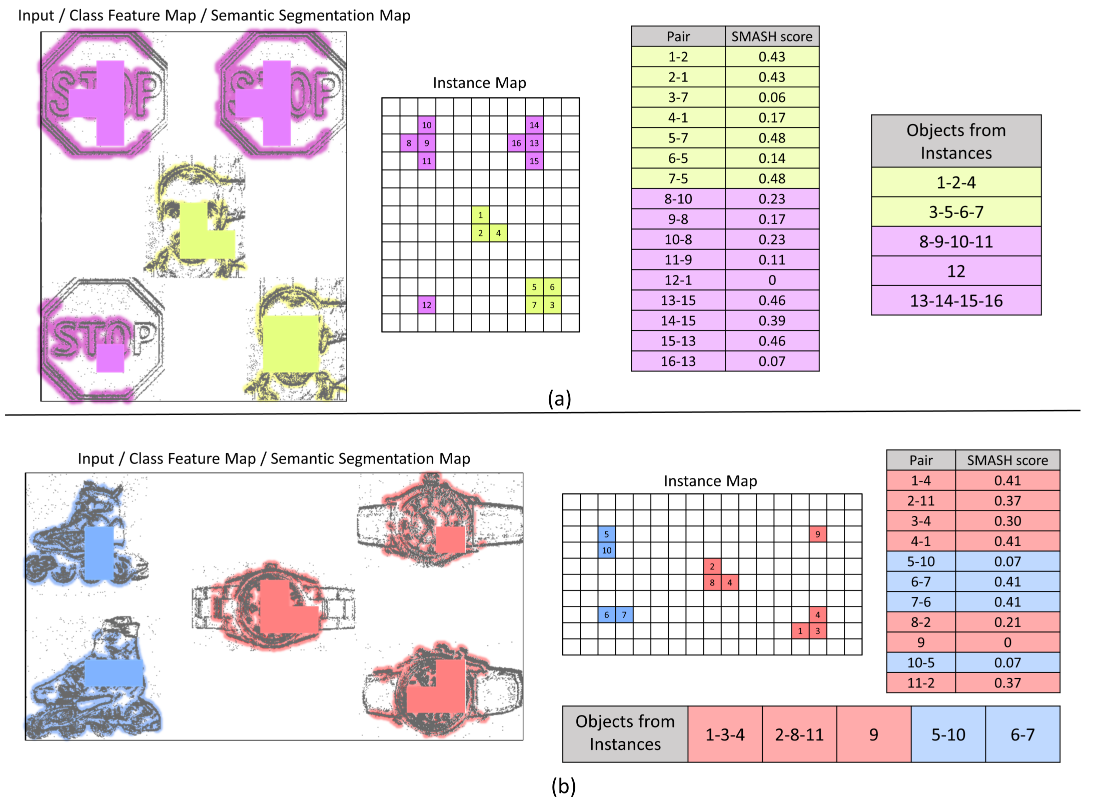
	Occlusion Recovery Rate (%)		
	without noise	**with noise**	
Multistream N-CalTech (Face)	100	100	

	Object Self Matching		
	Top 1	**Top 3**	**Top 10**
Multistream N-CalTech (Face)	95	100	100

the SMASH process, allowing objects that don't share spatial locality to lead to false object grouping. As such the performance of the object detection to identify that the individual class instances belonging to one object is being evaluated, similar to the process depicted in Figures 5.2 and 5.3. Meaning even though the network is classifying all the Faces as just that, Faces, the SMASH process is able to identify that these do no share the same featural content. The object detection accuracy to identify three separate distinct faces is 100% over the complete input sequence, results are tabulated within Table 5.1. However, over each input buffer, this drops to 97.33% as there are a few cases in which the similarity is 0 due to a low input spiking rate producing the minimal number of spikes for activation. This is typical of the inputs from the start and end of the saccade movement within the N-Caltech dataset [207].

5.3.2 Object Detection and Segmentation in a Multi-Class Environment

To further evaluate the SMASH method the multi-class SpikeSEG networks from section 4.4 were tested. This allowed testing of the object detection and instance segmentation from the 5 and 10 class semantic segmentation environment. Together with the increased number of classes present in each segment, there was also a reduction in the overall number of features available within the second convolution layer, Conv 2. This presents a scenario where the features are more generalised and therefore are less opportunity to learn features that help to diversify. Despite this, the SMASH process was still able to deliver high accuracy in determining how many object instances of a class there was, with 95.17% within the 5 class testing and 91.67% for the 10 class test, collated in Table 5.1. To help visualise the results from this experiment two examples of successful instance segmentations outputs from the 10 class test are shown in Figure 5.5, which shows for both (a) and (b), the input overlaid with the classification layer feature map and the semantic segmentation output respectively. Figure 5.5 also indicates how the class instances are assigned in the "Instance Map", where the numerical assignment is displayed to help illustrate the SMASH score also seen within the image. Where the Pairs are shown in numerical order with associated SMASH score then how that is grouped into objects.


Input / Class Feature Map / Semantic Segmentation Map
Instance Map
(a)
Input / Class Feature Map / Semantic Segmentation Map
Instance Map
(b)

Pair	SMASH score
1-2	0.43
2-1	0.43
3-7	0.06
4-1	0.17
5-7	0.48
6-5	0.14
7-5	0.48
8-10	0.23
9-8	0.17
10-8	0.23
11-9	0.11
12-1	0
13-15	0.46
14-15	0.39
15-13	0.46
16-13	0.07

Objects from Instances
1-2-4
3-5-6-7
8-9-10-11
12
13-14-15-16

Pair	SMASH score
1-4	0.41
2-11	0.37
3-4	0.30
4-1	0.41
5-10	0.07
6-7	0.41
7-6	0.41
8-2	0.21
9	0
10-5	0.07
11-2	0.37

Objects from Instances	1-3-4	2-8-11	9	5-10	6-7

Fig. 5.5: Illustrating the process of going from semantic segmentation to instance wise object construction, displaying the instance mapping and pairing with SMASH scores that help to construct the class instance objects for two examples (a) and (b). Respectively showing the process for the classes Face and Stop Sign, and Watch and In-line Skate.

The classes other than Face presented a different problem to the Face class, as most of the other classes (excluding Stop Sign) had a higher feature count in the second convolution layer Conv 2. This presented a problem in which the timing of the occurrence of the features was more important than the fact it had occurred which during the Face class only testing, wasn't an issue due to the sparsity at which features appeared. This testing highlights the importance of the temporal component of the similarity matching as simple feature-wise similarity matching alone is unable to determine class instances from one another. This extra temporal dimension can be seen a being able to leverage the saliency (timing) of features within individual class instances.

5.3.3 Object Occlusion

To help test the HULK-SMASHs ability to be able to perform sequence to sequence tracking of objects, further evaluation of the object detection capability is tested within three more challenging scenarios, where the class instance will now overlap. Taking the same multi-stream approach as the previous test, however, now the central stream of the image will be positioned such that occlusion of 5%, 25% and 50% will occur. Partial occlusions test the feature extraction and similarity matching performance when there is a bounding box overlap, meaning the SMASH score is now the active measure of the combined similarity and locality. The three occlusion states are illustrated in Figure 5.6 were (a), (b) and (c) depicted the occlusion covering for 5 %, 25% and 50% respectively as the stream of data in the centre is repositioned to create the occlusion.

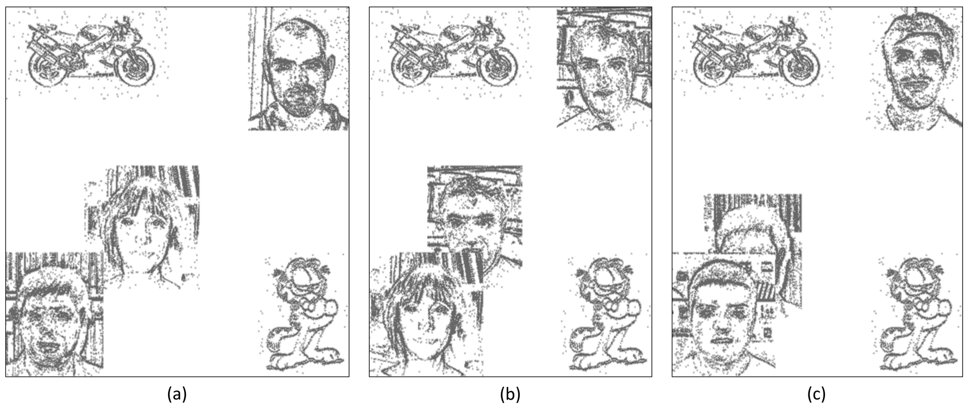


Fig. 5.6: Input occlusion example illustrating: (a) 5% occlusion of center image, (b) 25% occlusion of center image, and (c) 50% occlusion of center image

With the 5% occlusion, the performance was unchanged at 97.33% also shown in Table 5.1, which was to be expected considering the example shown in Figure 5.6 (a) shows no overlapping of the facial features. With all images in the dataset having the faces centred in the image. While the 25% occlusion reduced the performance down to 62% and with 50% occlusion getting 28% also both in Table 5.1 when testing over each input buffer. A more detailed example from the 25% occlusion testing is shown in Figure 5.7. The figure highlights the process from the input in (a), the class latent space neurons that were active in (b), the HULK end process of the active class pixels represented back in the pixel space with (c), and finally, the instance segmentation output (d), with three successful instances of each face detected. The only concern with this approach is the overlapped section between the bottom two images has a region that is assigned to two objects. This results in an area of uncertainty and even with a process where the two objects could compete for the pixels, the misclassification appears to exist in the feature detection domain, not the HULK-SMASH process. However, it must be noted that the failures within this testing phase are exclusively from the failure to find enough features to give a positive classification, not the incorrect assignment of instances. This element is highlighted in Figure 5.8, where the classification process has failed to result in classification for the centre image, however, the bottom left and top right faces are correctly classified and segmented without overlap onto the unclassified region.

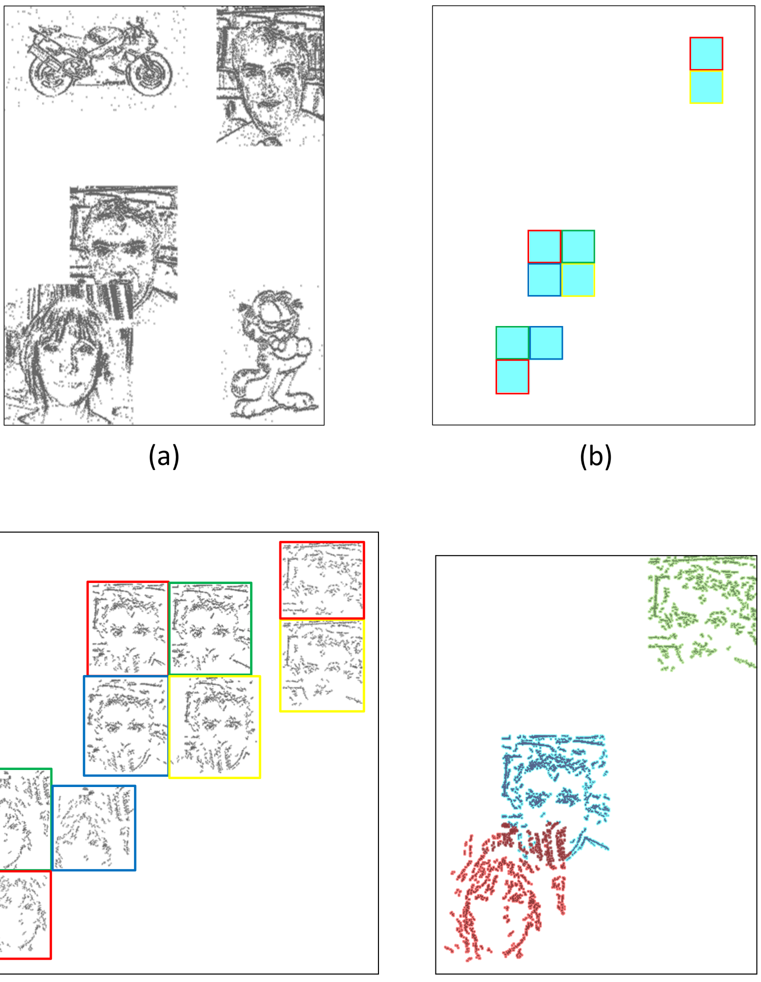


Fig. 5.7: Input 25% occlusion example illustrating: (a) The input sequence, (b) The latent classification space, (c) The class instance breakdown and (d) The instance segmentation output

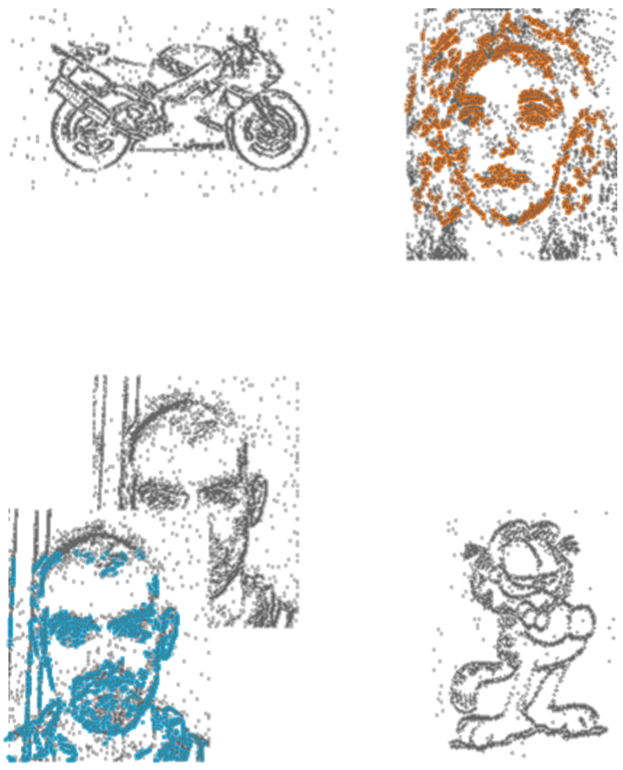


Fig. 5.8: Segmentation mapped over the input sequence, with a failure to identify the central image, thus failure to include in instance segmentation, however successful segmentation of the other two faces

5.3.4 Object Occlusion Recovery

As shown in the previous section, the object detection and instance segmentation method can identify features belonging to a particular instance of that class on an input by input basis, known as intra-sequence detection. The experimental set up for this section allows testing of object detection across inputs multiple sequences known as inter-sequence detection. This test then allows sequence to sequence tracking, then partial and full occlusion and subsequent recovery. The inputs are the full 300 ms sequence broken into the usual 10 ms steps as seen in previous sections. This places two inputs at the top right and bottom left, then has then move diagonally across towards each other for each buffered input. The results of the test from a few selected instances are shown in Figure 5.9. Figure 5.9 (a) depicts the start position of the two different sequences and their respective instance segmentation and bounding boxes (red and black). Figure 5.9 (b-e) show the transition of the respective segmentations diagonally across with (c) and (d) showing the occlusion due to overlap where the black object completely occludes the red object. Figure 5.9 (e) then highlights the detection picking up the red object again, from finding a high similarity with the red object that existed prior to the occlusion.

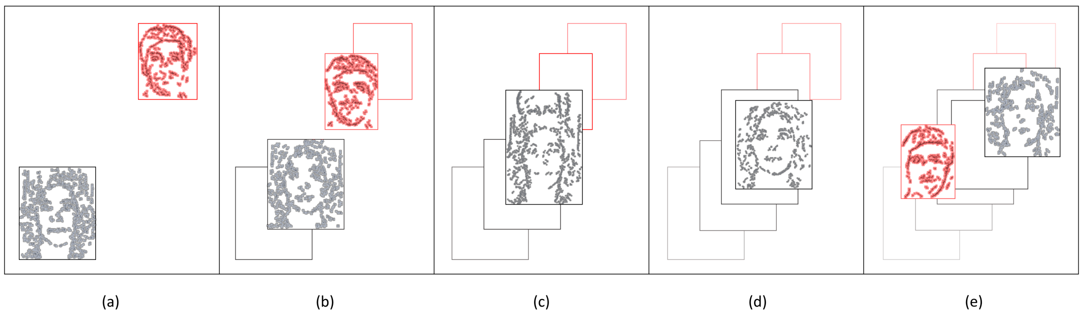


Fig. 5.9: Segmentation mapped over the input sequence, with a failure to identify the central image, thus failure to include in instance segmentation, however successful segmentation of the other two faces

The experimental results show the inter-sequence similarity matching is able to recover the occluded input sequence. Such that across all the test sequences the similarity check managed to recover the occluded objects 100% of the time, as seen in Table 5.1. This feat is more impressive due to the occluded object being a dynamic sequence itself. Meaning the post occlusion object is not just a replica of pre occlusion one. This occlusion recovery was also tested again with a noisy input, that is added noise to both the input sequence and to the general background. This noisy sequence is shown in Figure 5.10, where Figure 5.10 (a) and (e) shown the pre and post occlusion states, presenting that the recovery works correctly, while Figure 5.10 (b), (c) and (d) show some of the issues the occlusion can cause. Figure 5.10 (b) highlights the failure of the object detection in finding extra objects that don't exist. Figure 5.10 (c) correctly illustrates the full occlusion state where only the female face object is present. Figure 5.10 (d) displays the point at which partial occlusion is still causing only one object to be present, due to the similarity of class instances being close across the 9 instances in this case. Some of the 9 instances include multiple features of both objects, similar to what is shown in Figure 5.10 (b). However, in this case, there was at least one instance that matched another within both objects. The object in Figure 5.10 (d) is also blue as it scored the highest similarity with that object on the inter-sequence process. Despite some of the mentioned inaccuracies associated with the added noise, the system was able to re-identify occluded objects 100% of the time, as shown in Figure 5.10 with this result recorded with the others in Table 5.1.

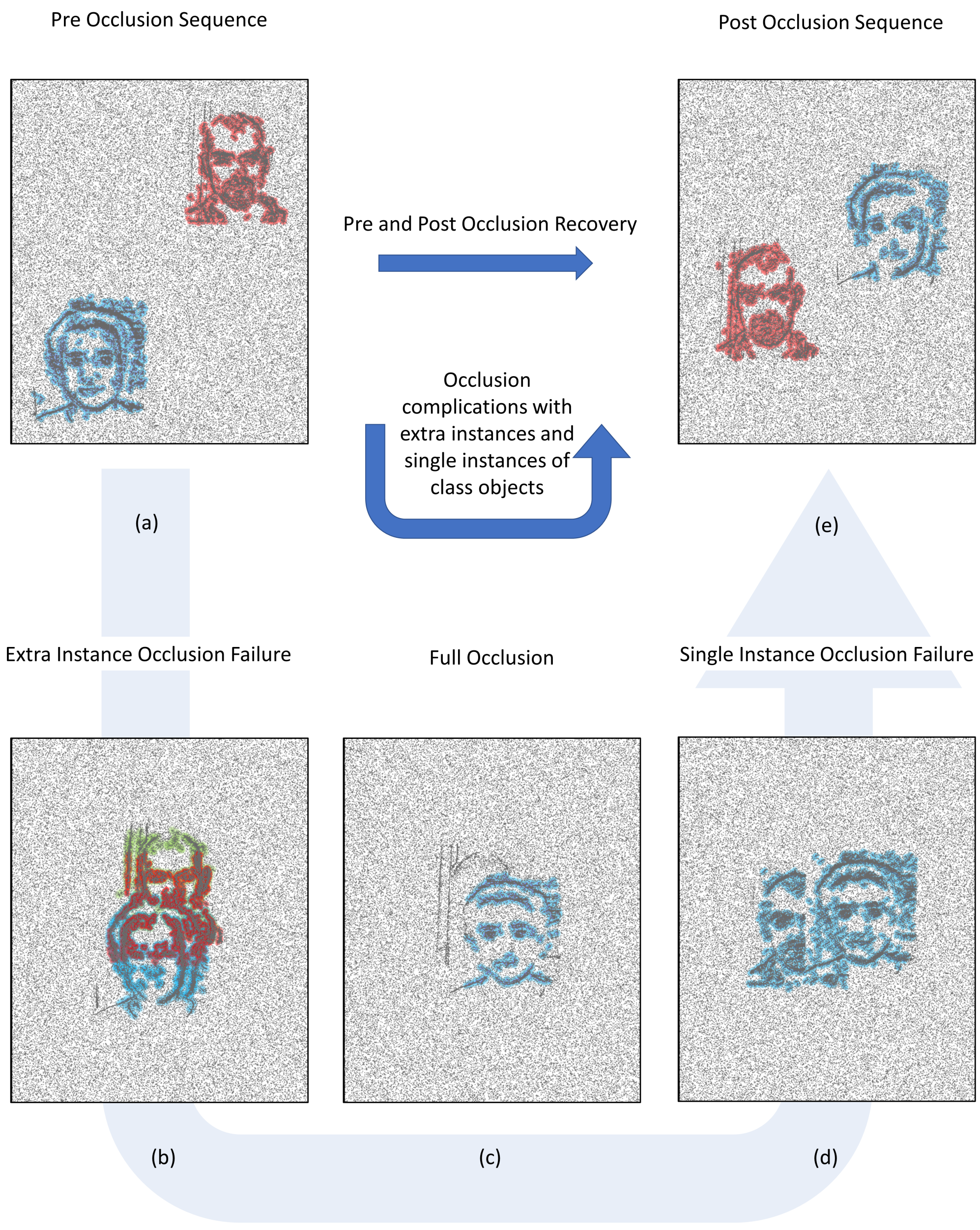

Pre Occlusion Sequence
Post Occlusion Sequence
Pre and Post Occlusion Recovery
Occlusion complications with extra instances and single instances of class objects
(a)
(e)
Extra Instance Occlusion Failure
Full Occlusion
Single Instance Occlusion Failure
(b)
(c)
(d)

Fig. 5.10: Occlusion recovery with noisy input. (a) Pre Occlusion state, (b) Occlusion causing extra objects to appear, (c) Full occlusion with only one object present, (d) Partial occlusion grouping of objects and (e) Post occlusion state

5.4 Intuition and Understanding through Visualisation

Building upon the visualisation ideas of Chapter 4, the principle behind how object detection and instance segmentation are resolved can be better understood through visualisation. Through visualisations of the weights as they are mapped back into the pixel space through the subsequent layers are shown in Figure 5.11. Figure 5.11 (a) showing the features of Conv 1, which are the unlearned pre-determined edge detection features, (b) showing the features of layer Conv 2 (Trans-Conv 2) and (c) highlighting the classification layer feature which depicts a Face. Figure 5.11 (b) contains the bulk of the information used for the similarity matches process, thus contains the key to interpreting how the network can differentiate between different people in the dataset.

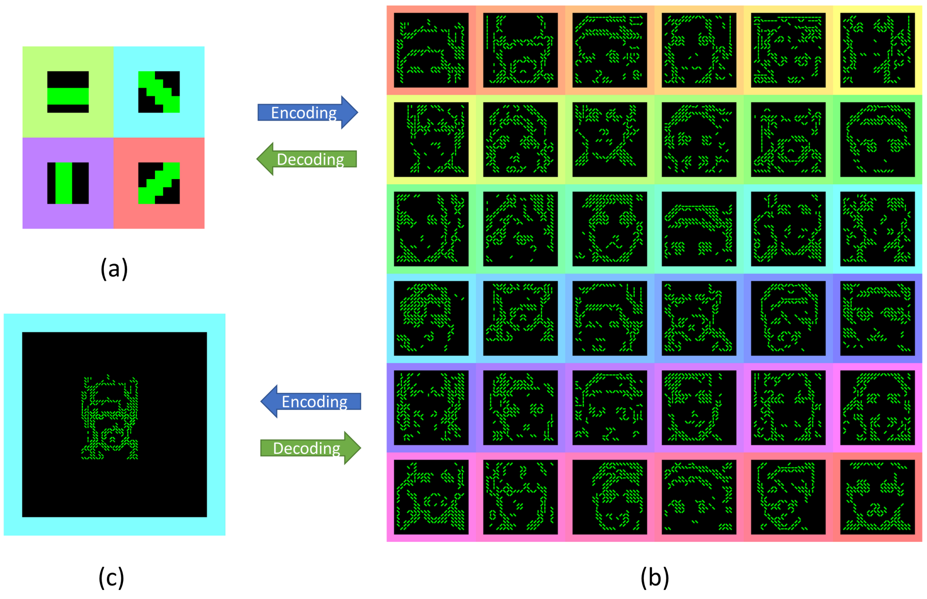


Fig. 5.11: The set of features from the trained network, (a) Conv 1, (b) Conv 2 and (c) Conv3

Mapping the features of the network from Conv 2 layer onto some original images from the Caltech dataset presents an insight into how this layer is differentiating between the different people in the dataset, as each person has unique and distinctive enough features to be learned through repeat occurrence. This ability to learn sub-classification feature clustering further highlights comments made in section 4.5 about the ability to classify up to 10 different objects due to the variance between the class (inter-class) being higher than the variance of one particular class (intra-class). However, in this case, it is now the intra sub-class variance that allows the unique featural-temporal identifiers of each face to be captured. This is highlighted in Figure 5.12 where 4 distinctive features that appear to represent a particular person from the dataset are overlaid onto the non-spiking version of the original image (for easier visualisation) it matches closest. In which the features of Figure 5.12 appear to pick out the typical facial features of eyes nose mouth and hair.

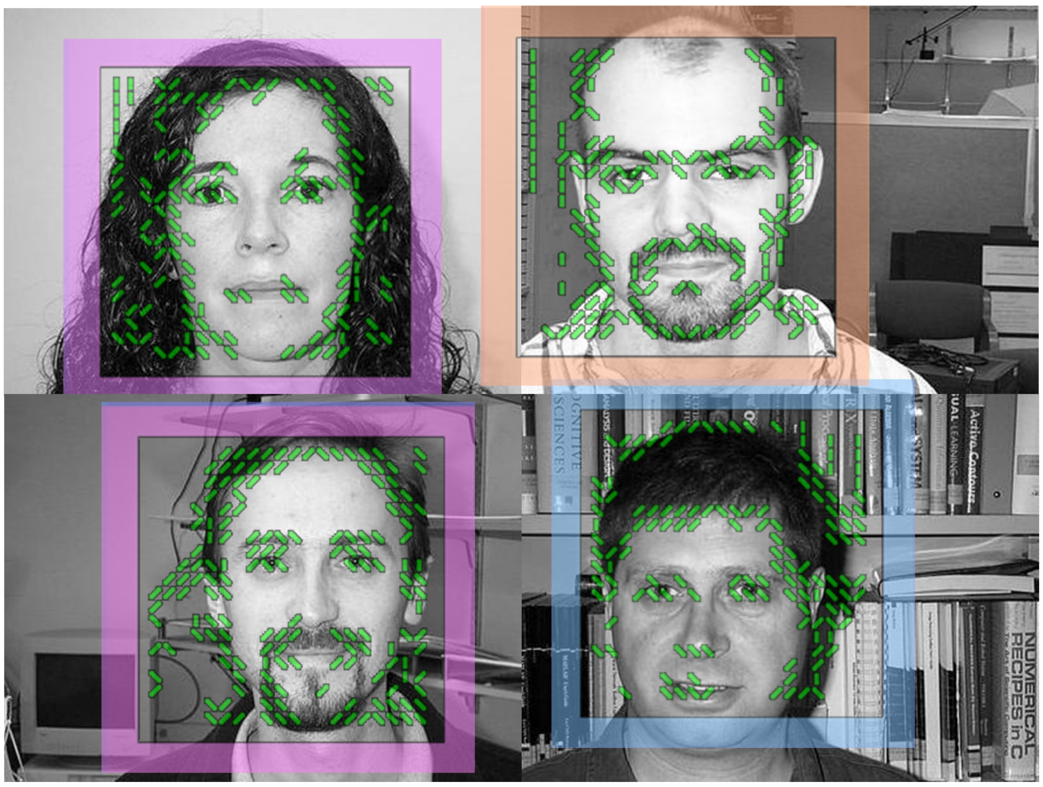


Fig. 5.12: Overlaid Features from Conv 2 onto Images from original Caltech Dataset, displaying how the learned features represent different faces

To test just how well the features are allowing the individuals within the dataset to be recognised, an experiment matching each sequence with its top 1, top 3 and top 10 matches was carried out. This permits an insight into whether or not the HULK-SMASH process was able to differentiate between the people within the dataset, purely on re-occurrence of similar salient features, or in other words distinctive facial features in a featural-temporal manner. The test processes each buffered input, within each sequence of the testing set, and compares it with the full test set, recording the top, top 3 and top 10 most similar spatial-temporal patterns to its own. The results show that over 95% of the test inputs for an individual match the highest with themselves. While 100% match with themselves if extended into the top 3 and top 10, displayed in table 5.1. This result is illustrated in Figure 5.13, where the top 10 similarity matches for a selection of inputs are shown in a coloured column-wise manner, with the left of the column showing the input sequence and the right showing the top 10 matches in descending order.

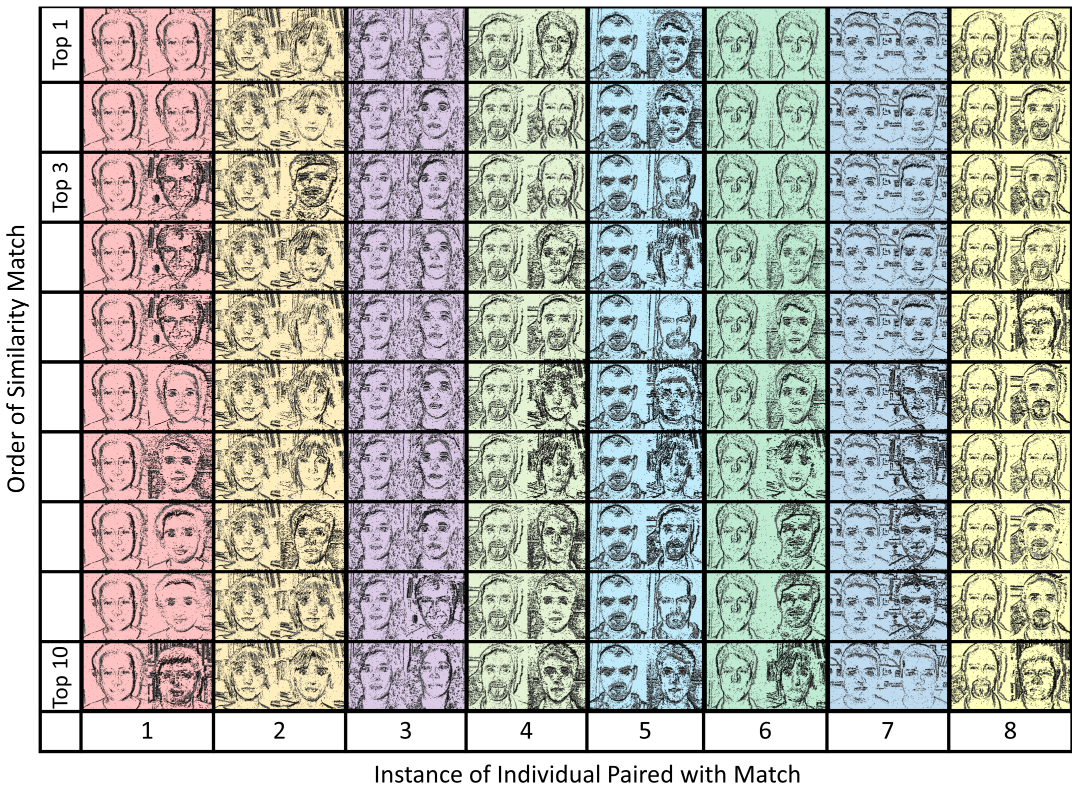

Top 1
Top 3
Top 10
Order of Similarity Match
1
2
3
4
5
6
7
8
Instance of Individual Paired with Match

Fig. 5.13: Columns of top 10 matches, left on each column is the test input repeated 10 times with each of the 10 matches to the right, ranked in descending order. Each colour represents a new column for easier visual separation

Figure 5.13 helps to build interpretability to the similarity matching process, allowing an understanding as to how and why the accuracy of the classification and object detection is as such. The sparse feature sets of a winner takes all approach to STDP allowing easier visualisation. This in turn helped to construct a method of matching these spatial-temporal features which are a result of the neuromorphic input to the spiking network both maintaining the important temporal causality of salient features.

5.5 Conclusion

In this chapter a new method for spiking object detection and instance segmentation, HULK-SMASH was presented. The system utilised STDP learned features that previously gave a semantic segmentation output. The classification layer was subsequently processed by unravelling each individual latent spiking neuron in the layer back to the pixel space, seen as HULK. This allows each instance of the classification layer to be treated as a class instance. It is then through comparing the class instances that objects can be identified through the SMASH score looking at the similarity and locality of instances. Permitting object detection and instance segmentation to be carried out on each buffered input. Then through similarity matching of the objects in sequences, sequence to sequence object detection and segmentation was achieved. This allowed object occlusion and reappearance to be realised, with the ability to deal with subtle changes to the object pre and post occlusion. Visualisation of experimental results gave a better understanding and intuition as to where the network was working well and why some failures occur. The HULK-SMASH processes were shown to be robust to multiple network instances with single and multi-class experiments with successful results while aiding in neural network interpretability.

Chapter 6

Spiking LiDAR

6.1 Introduction

Typically most imaging methods can be divided into two categories. In the first, the scene is flood-illuminated with light, that is, all regions of the scene are illuminated simultaneously. The light reflected by the scene is then imaged onto many detector pixels via a lens, which is what has been shown with the NVS in Chapters 4 and 5. In the second, only a known sub-region of the scene is illuminated, and the light reflected from that sub-region is collected onto a single pixel. By dividing up the scene into many of these sub-regions, and measuring light from only one region at a time, one can scan over the scene. By combining the time-of-flight (ToF) information from the sub-regions, the entire scene can be reconstructed, prompting the name temporal imaging, more commonly called Light Detection and Ranging (LiDAR). These processes extend also to three dimensional (3D) imaging, where the distance from the sensor can be inferred from stereoscopic imaging, holographic, or ToF methods [225–228].

This chapter presents a novel solution to this second temporal imaging problem, making use of the asynchronous sparse processing methodology of the SNN. This research looks to extended upon a single point solution to this inverse retrieval problem, that was presented in Turpin et al [229], in which a standard fully connected ANN with approximately 10 million parameters, was shown to deliver good image reconstruction results. However, this particular problem has many traits in which a neuromorphic approach with SNNs could be beneficial. In fact, the ToF sensor is akin to that of a spiking sensor, in that it records single instances of photons returning (after a pulse has been transmitted), assigning a time-stamp to each return. This is then similar to a NVS described in Section 3.5.2, with a high temporal resolution, albeit without the spatial and polarity information.

Typically this ToF sensor would capture thousands if not tens of thousands of photons to get a good distribution of the light reflecting in the scene, then process this histogram of time returns in order to retrieve a 3D depth map. An example of this is shown in Figure 6.1, where the histogram information of these time returns (a), is used to recreated a 3D Depth Map of the scene through the use of only temporal data from a single point, with ground truth 3D Maps shown in (b) as an example of the desired output.

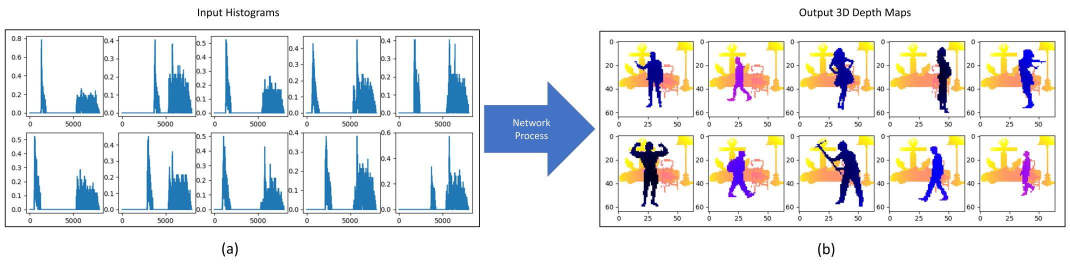


Fig. 6.1: Illustration of the input (a) ToF histograms collected from the scene, that once processed output 3D depth maps (b).

The inverse retrieval problem presented has two main areas that this chapter looks to improve, *accuracy* and *latency*. First is the re-imaging of the network from an ANN to a variational fully convolutional encoder-decoder network, which is an accurate, high quality image creator [230, 231], much like the examples seen in Section 2.5. This can later be adapted to a SNN, with methods similar to that shown in section 3.4.3. This network is then similar to the fully convolutional encoder-decoder SpikeSEG network presented in Chapter 4. However, in this case, the additional variational term is referring to the translation from the 1D temporal depth domain to the 3D depth map domain, where the transverse spatial features are inferred by the network. The second improvement comes in the processing overhead and latency. The processing overhead is seen as the number of calculations that are required to be carried out for the network to return an output. The spiking nature of the proposed network Spike-SPI (Spiking - Single Point Imager) allows a reduction in information being propagated through the network, thanks to the spiking neurons thresholding ability to reduce non-salient data flow. In terms of latency, each processing stage can only be complete when the photon counter has reached the desired captured value; this results in a dead time in waiting for the sensor to return enough information before processing, and a similar constraint in waiting for the processing stage to finish before the next batch can run. The asynchronous characteristic of the spiking neuron could help to reduce this by allowing continual processing of direct or buffered inputs. It is through utilising the feature extraction capabilities of CNN, with the high dimensional compressed latent space representations of an Encoder-Decoder network structure and the asynchronous processing capabilities of the SNN that both an increase in accuracy, while decreasing the latency and computational overhead can be achieved.

The remainder of this chapter is organised as follows. Section 2 the methodology, cover the simulation and proposed network details. Section 3 details the results of the simulations comparing the proposed Spike-SPI network with the ANN model of Turpin et.al [229]. Section 4 details the broader impact of this work, while Section 5 contains the conclusion.

6.2 Related Work

In recent work, Turpin et al. [229] have demonstrated the feasibility of such a temporal imaging approach. They flood-illuminated a scene with a pulsed laser source and focused the back-reflected light onto a single detector pixel. To achieve this, a single point, single photon avalanche diode (SP-SPAD) [232] and Time Correlated Single Photon Counting (TCSPC) [233] were used to measure the ToF of the photons between emission and back-reflection from the scene. Instead of measuring the spatial structure of the light, as in the aforementioned methods, the images were reconstructed from the ToF alone [234], with the temporal data being interpreted by an ANN trained from both ToF histograms and ground-truth 3D images. The difficulty arises from losing the spatial structure of the scene, resulting in the inverse image retrieval problem becoming heavily ill-posed. As the whole scene is illuminated simultaneously, and all photons are collected onto only 1 pixel, a photon measured at time t may originate from any point on the surface of a sphere. The radius of the sphere of possible reflection point is given by $r = ct$, where r is the radius of the sphere and c is the speed of light. A cross-section of this sphere of possible points is illustrated in Figure 6.2, where the black circle represents the sphere and the sensor field of view is represented within the blue target plane hemisphere. The figure also depicts the sensor and illuminator within the grey box with a light source cone and photon returns illustrated.

In other words, just from the arrival time of a photon, it is analytically impossible to determine which point the light came from. However, that is not to say that the retrieved temporal information is fully uncorrelated to the spatial structure of the scene. Objects (such as people, chairs, cars, etc.) reflect photons with recognisable temporal traces [234], yet these traces are dependent on the

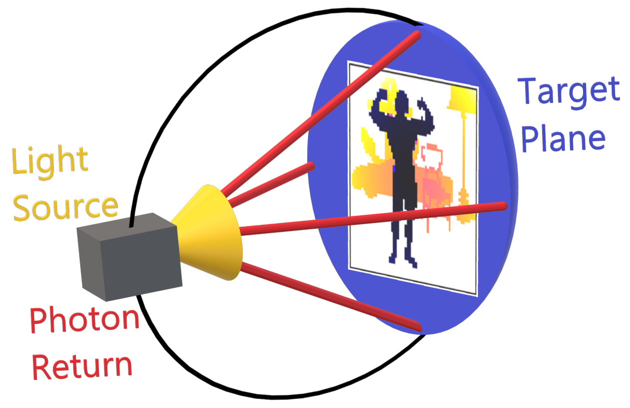


Fig. 6.2: Visualisation of the inverse retrieval problem, where multiple points within the scene can return the same time difference to the sensor due to the spherical geometry of the problem

object's orientation, reflectivity, distance from the camera, size, vicinity to other reflective objects and so on. To further complicate the issue, multiple objects can have the same temporal trace. As a result, it is practically infeasible to try to reconstruct objects by implementing a dictionary mapping temporal traces to potential sources. However, Turpin et al. [229] have shown that a machine learning algorithm can identify structures in the temporal signal which correspond to spatial structures in the scene, allowing them to recover a 3D scene from purely temporal data.

Outwith the aforementioned research LiDAR data has seen both CNNs and SNN used for a multitude of tasks, albeit not for turning temporal histograms into 3D depth maps. CNNs have been successfully utilised to perform depth estimation with a sparse 3D point cloud of LiDAR information, combined with conventional images [235, 236]. They have also been utilised with 3D point cloud and 3D LiDAR data for detection and localisation [237, 238]. Recently two review papers [239, 240] highlight the rise in popularity of LiDAR data and improvements made over two seminal works PointNet [237] and VoxelNet [238]. However, the use of CNNs on the histogram of LiDAR data is less commonly used, as all the previous methods utilise the spatial-depth information from the 3D point clouds. Two such networks that use the histogram data are LiDARNet [241] and FWNetAE [242]. LiDARNet was capable of identifying significant peak locations and clusters directly while delivering a better signal reconstruction performance at an order of magnitude faster than statistical methods. FWNetAE was able to encode meaningful feature into the latent space vector which allowed the decoder to reconstruct spatial geometry and its waveform. All of these CNN methods utilise the fully convolution network structure as described in section 2.5 SNNs and even their convolutional variants have been used to perform fast classification of objects with LiDAR data [243, 244]. Both of these methods utilise the supervised method of time coding error backpropagation [106], as seen in section 3.4.2, to enable object detection at an earlier stage in the LiDAR processing pipeline. The networks utilise the temporal returns from the LiDAR array as spiking impulses and rather than passing to a digital timing chip that would help to form a histogram the information can be processed directly from the sensor. The classification after processing in the SNN is carried out through a time-to-first-spike method as seen in section 3.3.2.

6.3 Methodology

The temporal data to 3D imaging problem consists of three main elements:

1. a pulsed light source
2. a single-point time-resolving sensor
3. an image retrieval algorithm

The focus of this research being a novel image retrieval algorithm Spike-SPI. However, the first two steps are necessary to collect the required data for processing, with the temporal data being collected as follows. The scene is flood-illuminated with the pulsed source and the resulting back-scattered photons are collected by the sensor. A SP-SPAD detector operated together with TCSPC electronics, forms a temporal histogram [Figure 6.1(a)] from the photons arrival times. Objects placed at different positions within the scene and objects with different shapes provide different distributions of arrival times at the sensor [234]

The original neural network [229] was trained on synthetic data, which had been designed to simulate the imaging set-up described above. With the experimental data capture in Turpin et al [229] being replicated in this experiment. The simulations contain humanoid silhouettes in various poses in front of a background consisting of some objects, as illustrated in Figure 6.3. The silhouettes and background objects form a 20m^3 3D environment, as seen by a simulated 3D

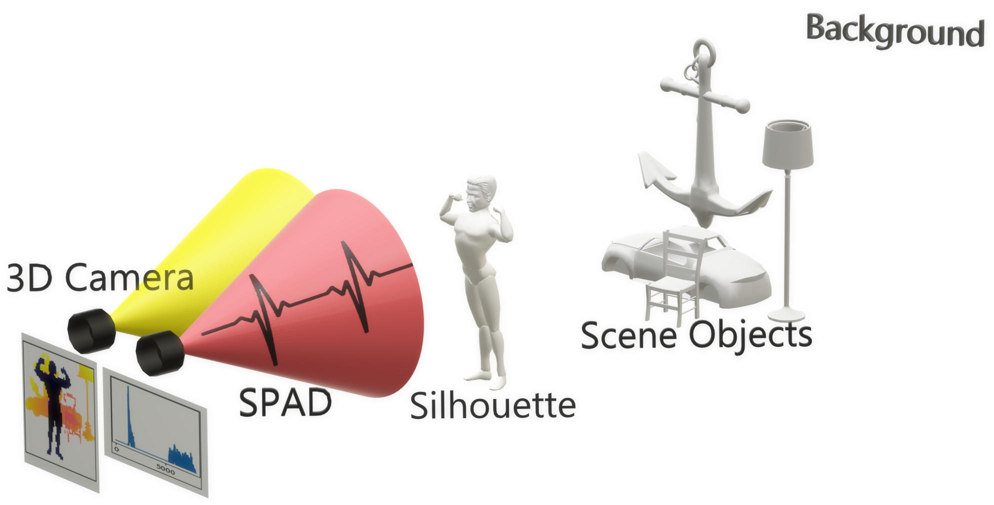


Fig. 6.3: Mock-up of the synthetic scene used, with the 3D camera and SP-SPAD capture the scene in depth image and histogram formats. The scene is made up of the human silhouette and the scene objects of a chair, car, lamp and anchor.

camera, where the distance from the virtual camera is encoded in the colour of the scene. The photon arrival time probability density function (PDF) for the virtual scene can be found. This then allows the signal observed by the virtual SP-SPAD to be estimated by convolving the photon arrival time PDF with a Gaussian instrument response function (IRF). Finally, assuming that subsequent photon measurements are independent and identically distributed, we simulate p photons detected by the virtual SP-SPAD with TCPSC by sampling the convolved photon arrival time PDF p times, assigning each photon into one of 8000 time bins, with the bin size being 2.3 ps. This bin width is convolved with the two IRFs of 20 and 100 ps, resulting in values more consistent with practical values of time counting electronics, 20 ps being the best case and 100 ps being the more typical real-world case. Photon counts of 1000 and 9500 were also selected to represent a typical amount of photon returns, as with previous experimental findings indicating that about 1000 photons per temporal histogram are needed to retrieve a meaningful image [229]. This value was then set as the minimal amount, with 9500 photons being a more realistic value observed in actual testing scenarios [234]. Giving 4 testing parameters with 2 photon values and 2 IRFs to test a range of scenarios from ideal (9500 photons with a 20 ps IRF) to challenging (1000 photons with a 100 ps IRF)

The novel variational Encoder-Decoder Spike-SPI network is shown in Figure

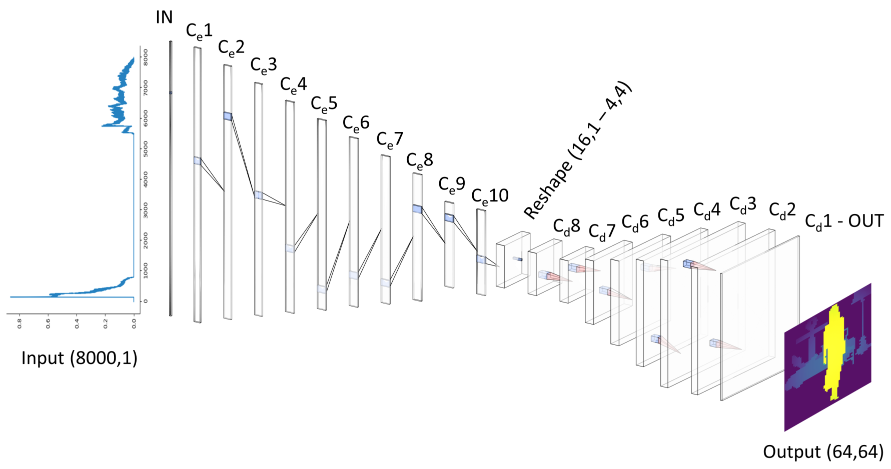


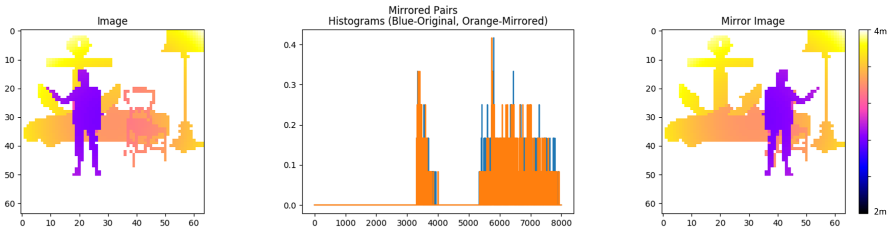
Fig. 6.4: Spike-SPI Network structure with 1D convolutions reshaped into 2D convolutions. The input of 8000 long vector, reshaped at the transformation stage from 16 long vector to 4x4 matrix, before being output as a 64,64 matrix with a colour graded intensity map.

6.4. It comprises 18 layers with 10 layers for the encoder and 8 layers for the decoder, this being the required number of layers to get the original histogram 1D vector into a size that could be mapped to a matrix for the 2D decoding (i.e. [16,1] to [4,4]). The input is comprised of the 8000 bin histogram of the captured ToF data and the output of a 64 by 64 matrix image. These are processed through the network's convolutional layers as detailed in Table 6.1, where kernel value sizes of 7 by 7 for encoding and 5 by 5 for decoding were empirically set with initial testing.

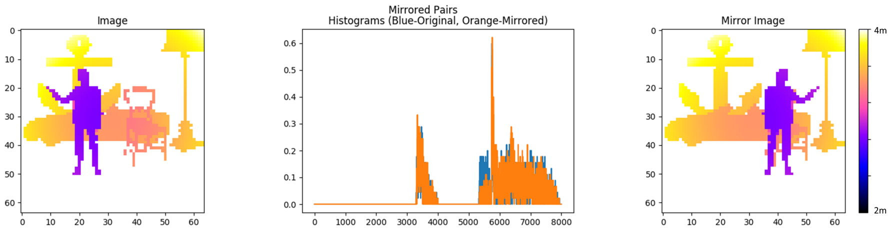
Table. 6.1: Details of the Spike-SPI network

Network	**Spike-SPI**							**ANN**		
Layer	C_{e1}	C_{e2-3}	C_{e4-10}	C_{e8-5}	C_{e4-3}	C_{e2}	C_{e1}	FC_1	FC_2	FC_3
Kernel Size	7	7	7	5	5	5	5	-	-	-
Number of Features	64	128	256	256	128	64	1	1024	512	256

The original ANN network [229] used as a comparison has 3 fully connected layers with 1024, 512, and 256 nodes respectively. Both networks were given the same training data which was split into 4 different experiments, with the two different photon counts and two different IRFs. This also helps to determine the robustness of the network to differing input conditions. The networks are trained on the 11600 samples using 29 different silhouettes, with all training and validation testing instances ensuring images and their mirrored version are kept together. These mirrored pairs are kept due to the histogram of silhouettes mirrored on the central point of the x or y axis are identical (due to the temporal information they return is the same); however, this does not mean the full histogram is identical as different sections of the scene are occluded in each case as illustrated in Figure 6.5, with the blue section between 5000-6000 being clearly different. Figure 6.5 also helps to illustrate the difference in the 1000 photon data (a) and the 9500 photon data (b) in terms of histogram fidelity, meaning less difference is perceptible between the scenes. For final testing, the networks are shown histograms of a



(a)



(b)

Fig. 6.5: Histograms for both (a) 1000 photons and (b) 9500 Photons, showing the similar silhouette histogram but differing scene histograms.

previously unseen silhouette in a range of depths and potions in which it has to reconstruct. In total there are 20 x positions, and 10 z positions, for each silhouette all with the exact same scene of objects.

Utilising the feature extraction capabilities of Convolutional Neural Networks (CNN), the high dimensional compressed latent space representations of an Encoder-Decoder network structure and the asynchronous processing capabilities of a SNN, we develop a novel Spiking CNN (SCNN) structure that converts the 1D depth histograms into a Depth Map. Through the use of 1D convolutions, the network is able to encode subtle differences in local spatial regions within the depth histograms into a high dimensional latent space. This allows the subtle differences in the histogram due to the silhouette placement in the scene and the area in which it occludes to be captured. This latent space is then decoded by a 2D convolutional decoder, exploiting the strong spatially local correlation present in natural images. All this while the asynchronous processing nature of the spiking neurons allows for faster throughput and less computational overhead, reinforcing the benefits of the operational sparsity in the single point sensor. The final step to create the SNN described is done through training as a traditional CNN then converting to a SCNN through the use of Nengo DL [147], as described in Section 3.4.3.

Qualitative aspects from the reconstruction can be gauged visually by comparing the outputs of the two systems, namely the ANN and Spike-SPI. This allows for comparisons on not only the structural shapes of the object resolved but also the associated depth at which they reside, encoded within the colour data of the depth map. Along with qualitative measures, a series of quantitative measures are also proposed. The IoU (Jaccard similarity) [216], similar to that seen already in Sections 4.4 and 5.2.2 with bounding box comparison and featural temporal matching. The IoU allows the comparison of the foreground objects of the ground truth $\mathbf{Y}^*$, to be compared with the reconstructed foreground objects in $\mathbf{Y}$. This is done through first masking the foreground objects from the background. The background in these experiments are uniform and at maximal distance from the sensor. In order to threshold this and mask the foreground elements, the background is set at everything greater than 99% of the max distance. This means the constant scene objects and the silhouette mask are compared against the ground truth version to see how many pixels are correctly identified as belonging to an object. Meaning it is essentially looking for background pixels that have been incorrectly given a depth punishing a blurred edge smoothing or averaging approach. The IoU is calculated with the following in a similar manner to that of Equation 5.2

$$IoU(\mathbf{Y}_{mask}, \mathbf{Y}^*_{mask}) = \frac{\mathbf{Y}_{mask} \wedge \mathbf{Y}^*_{mask}}{\mathbf{Y}_{mask} \vee \mathbf{Y}^*_{mask}} \tag{6.1}$$

with $\mathbf{Y}_{mask}$ and $\mathbf{Y}^*_{mask}$ representing the masked reconstruction and ground truth images respectively. The Signal to Noise ratio is used to determine the amount of noise that exists in the reconstruction compared to the ground truth depth image.

$$\text{SNR} = 10\ \log_{10}\left(\frac{\|\mathbf{Y}^*\|_2^2}{\|\mathbf{Y}^* - \mathbf{Y}\|_2^2}\right) \tag{6.2}$$

where $\mathbf{Y}$ is the predicted depth map and $\mathbf{Y}^*$ is the ground truth. $\|\mathbf{Y}\|_2$ is the ℓ_2 norm given by $\sqrt{\mathbf{Y}^T\mathbf{Y}}$. The restoration quality was evaluated using the reconstruction signal-to-noise ratio (R-SNR) [245, 246] which is a metric that has been used for depth image reconstruction on sparse single photon data, where typically many of the pixels on the sensor are not active. This also acts as a depth invariant SNR measure.

$$\text{RSNR} = 10\,\log_{10}\left(\frac{\|\mathbf{Y}^*_{mask}\|_2^2}{\|\mathbf{Y}^*_{mask} - \mathbf{Y}_{mask}\|_2^2}\right) \tag{6.3}$$

The remaining metrics are typical quantitative measures in depth estimation [62]. The absolute relative error is calculated with

$$\text{Abs Rel} = \frac{1}{|T|}\sum_{y\in T}\frac{|y_i - y_i^*|}{y^*} \tag{6.4}$$

where each n^{th} pixel is indexed by i to form y_i, y_i^*, giving a per pixel metric. T is the total number of pixels per image. The squared relative error is calculated with

$$\text{Sq Rel} = \frac{1}{|T|}\sum_{y\in T}\frac{\|y_i - y_i^*\|^2}{y^*} \tag{6.5}$$

The root mean squared error is calculated with

$$\text{RMSE} = \sqrt{\frac{1}{|T|}\sum_{y\in T}\|y_i - y_i^*\|^2} \tag{6.6}$$

The scale-invariant log root mean squared error, si-logRSME [62] is calculated with

$$\text{si-logRMSE} = \frac{1}{2|T|}\sum_i\left(\log\, y_i - \log\, y_i^* + \frac{1}{|T|}\sum_i(\log\, y_i^* - \log\, y_i)\right)^2 \tag{6.7}$$

For any prediction y, $\frac{1}{n}\sum_i(\log\, y_i^* - \log\, y_i)$ is the scale that best aligns it to the ground truth. All scalar multiples of y have the same error, hence the scale invariance.

Lastly, the accuracy score is set through the number of pixels that remain within a threshold such that

$$\%\text{ of } y_i \mid \max\left(\frac{y_i}{y_i^*}, \frac{y_i^*}{y_i}\right) = \delta < thr \tag{6.8}$$

where δ is compared to three set thresholds 1.25, 1.25^2, 1.25^3 [62]. With each

value allowing a greater depth error to count in the accuracy.
The chosen metrics include some of the typical full image comparison measures, however, it is apparent that due to the averaging nature across some of those metrics if the network learns to blur across the areas of interest, on average it will receive a good score. This results in overall depth estimation being compared and not the spatial reconstruction. This is why there are a number of relative metrics and an IoU scoring system to test the two models outputs. As the testing should indicate which model best recovers the shape and depth of the silhouette and scene objects.

6.4 Results

This section details the results from the 4 experimental setups with photon counts being 9500 and 1000, while the IRF is set to 100ps or 20ps. The results of both networks are first tested on the validation test data and then on the unseen test data, to compare their depth map reconstruction abilities. Results from the unseen testing scenario are given in Table 6.2 comparing the two models, Spike-SPI and the ANN from Turpin et al. [229], against all the metrics detailed in the methodology.

Table. 6.2: Results from the experimental testing split into 4 sections with 2 photon count values of 1000 and 9500 and two IRF for each of those at 100ps and 20ps. **Best (Bold)**, *best in test (Italic)*, higher is better (↑), lower is better (↓)

	Photon Count	**IRF** *ps*	**IoU** ↑	**SNR** *dB* ↑	**R-SNR** *dB* ↑	**absRel** ↓	**sqRel** ↓	**RMSE** ↓	**si-log RMSE** ↓	$\delta < 1.25$ ↑	$\delta < 1.25^2$ ↑	$\delta < 1.25^3$ ↑
ANN	1000	100	0.650	*14.844*	2.880	22.074	4.444	*0.189*	0.474	0.853	0.886	*0.908*
Spike-SPI	1000	100	***0.783***	14.284	***6.502***	*3.597*	*1.323*	0.201	*0.456*	*0.871*	***0.890***	0.906
ANN	1000	20	0.650	*14.708*	2.890	26.354	5.142	*0.192*	0.476	0.853	0.886	*0.908*
Spike-SPI	1000	20	*0.760*	14.155	*5.842*	*6.571*	*1.502*	0.202	*0.456*	*0.868*	*0.889*	0.906
ANN	9500	100	0.637	***15.076***	2.614	20.558	3.846	***0.187***	0.468	0.856	0.889	*0.909*
Spike-SPI	9500	100	*0.780*	14.391	*6.360*	***3.066***	***1.163***	0.198	*0.456*	***0.871***	*0.890*	0.905
ANN	9500	20	0.631	*15.070*	2.500	15.124	2.559	*0.188*	0.470	0.856	0.888	***0.910***
Spike-SPI	9500	20	*0.778*	14.424	*6.358*	*3.461*	*1.438*	0.198	***0.456***	*0.870*	*0.888*	0.904

The experimental results are broken into two sections, with qualitative results for the validation testing and both qualitative and quantitative results for the unseen testing data, whereas highlighted in Table 6.2 Spike-SPI scores better on all metrics other than the SNR and RMSE. However, within the qualitative results, it is clear as to why this is the case when looking at the precision of both reconstructions. For all the other metrics tested, Spike-SPI was able to achieve better reconstruction results especially in metrics that focus on the spatial reconstruction such as the IoU, where an increase of over 0.12 on average represents a considerable increase in the accuracy. The accuracy metrics are shown in terms of $\delta < 1.25^{1,2,3}$ as seen in Equation (6.8) are a great indicator of why some of the full image averaging metrics score the ANN higher than Spike-SPI. As the threshold δ increases this is allowing a greater inaccuracy to count as correct on a per-pixel basis, showing that with the tightest threshold of 1.25 (25% relative difference between reconstruction and ground truth) that Spike-SPI scores better across all tests. However, when increasing the threshold to 1.25^2, the accuracy only slightly favours Spike-SPI and, with the largest threshold 1.25^3, the ANN scores higher. Further insight into these values based on the accuracy metrics would appear to show that around 87% of the pixels in the image have a very accurate depth estimate, while the remaining are considerably off. It is these misclassifications of pixels that brings the average score of Spike-SPI down on the full image metrics. Where a high precision, without 100% accuracy leads to instances where the majority of the pixels may be correct but those that are incorrect have a large error value. This can be seen as the manifestation of Spike-SPIs convolution layers for decoding, guessing which silhouette is in the image from the compressed latent space representation. This silhouette guess, then turns into a precise reconstruction which inevitably has errors.

The characteristics of the different qualitative metrics described previously are seen in greater clarity within Figures 6.6 (validation data) and 6.7 (unseen test data). Both figures illustrate that if the silhouette has a different outline from the ground truth, or is attempting to guess an unseen outline, the areas in which the reconstruction are incorrect happens to be a high magnitude depth error. This is due to the pixels normally belonging to the background which is set to max depth being misclassified. However, this resulting large depth error might actually be only 1 or 2 pixels off in terms of spatial error. Ultimately this results in metrics that favour depth accuracy over spatial accuracy, giving a better score to the ANN model over Spike-SPI. As a result, close attention has to be paid to what the actual values of the quantitative results in Table 6.2 mean. The results of this spatial or depth focus are shown within the results illustrated in Figures 6.6 and 6.7, where the ANN reconstructions with the blurred edges of the estimated silhouettes favouring depth, while the convolution process of Spike-SPI reduces this blurring significantly favouring spatial errors.

Looking closer at the qualitative results in Figure 6.6, which depicts the ground truth images shown above the ANN (left) and Spike-SPI (right), it can be seen

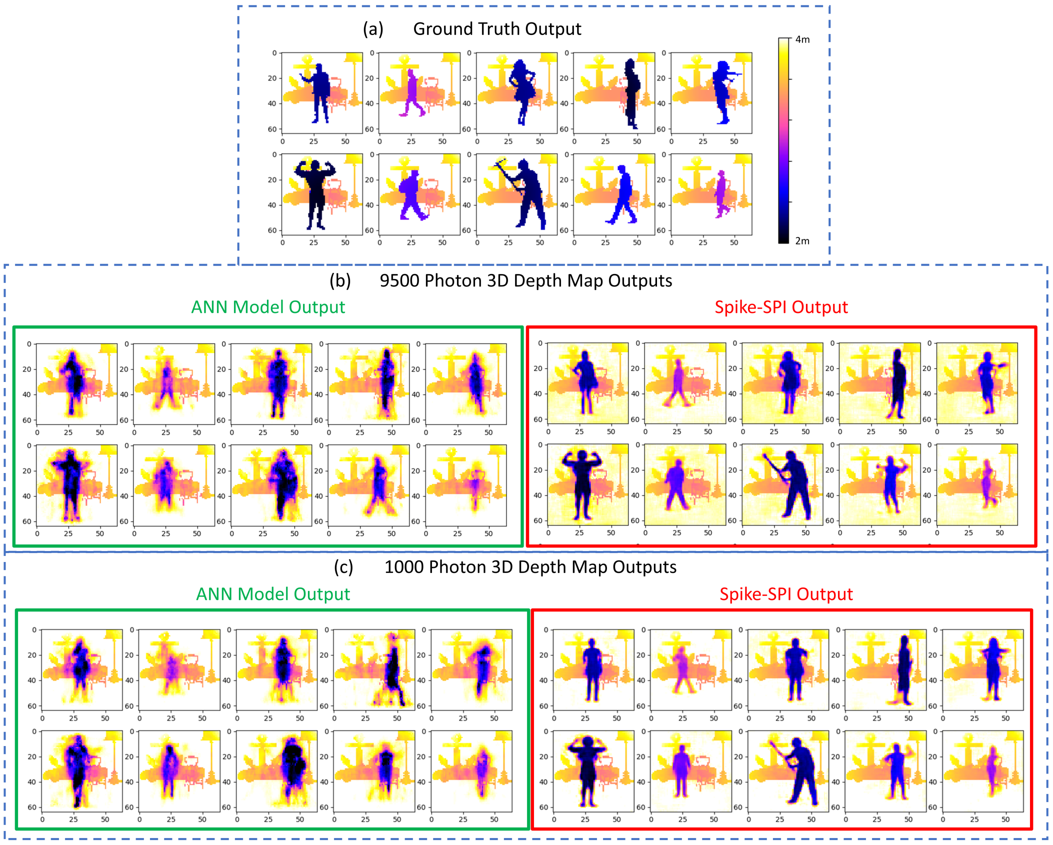


Fig. 6.6: Outputs of the two networks, ANN and Spike-SPI for (b) 9500 Photon, 20 IRF and (c) 1000 Photon, 100 IRF validation data. With (a) the ground truth data displaying 10 examples of silhouettes in the scene at the top. ANN in green on the left and Spike-SPI in red on the right.

that Spike-SPI has a higher spatial acuity, highlighted within Figure 6.6 as the silhouette holding the pic axe, where both the arms and the handle of the axe have been well resolved. This acuity is also still captured within the 1000 photo Figure 6.6 (a) data as well as the 9500 photo data Figure 6.6 (b). It is noticeable the deterioration in the spatial outline of the silhouettes within the ANN data when comparing the 1000 and 9500 photon results of Figure 6.6 (a) and (b) respectively. This consistency across a lower photon count is also reflected within the quantitative results in Table 6.2, where the results of Spike-SPI have less of a spread than the ANN across the 4 experiments.

Figure 6.7 illustrates the results of the unseen silhouette testing, while also illustrating the results of the masking process for the IoU measurement. The masking process not only highlights the spatial acuity of the Spike-SPI model compared to the ANN but serves as a visual explanation of the accuracy results with the increasing thresholds. Within both photon counts shown in Figure 6.7

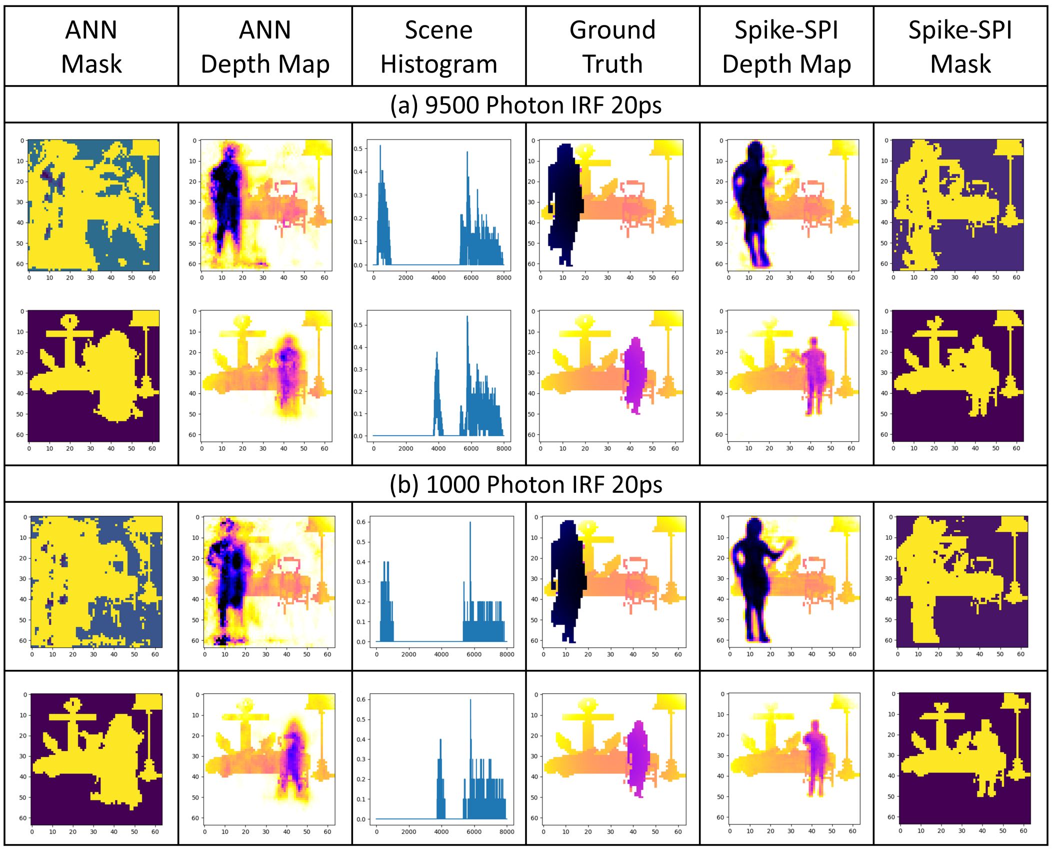


Fig. 6.7: Results on unseen silhouette data with reconstructions shown within depth map columns. Results of IoU masking also shown for this test data within the Mask columns. Testing results shown for both (a) the 9500 Photon, 20 IRF and (b) the 1000 Photon, 100 IRF data.

(a) and (b), Spike-SPIs results consistently mask a smaller percentage of the scene as foreground. This helps to visualise this acuity that penalises the method on some metrics, as it is apparent that it has misclassified some of the pixels when compared to the ground truth. This resulting high error value across a small number of pixels is in contrast to the ANN model, which has a lower spatial acuity as seen in the mask, but it also has a lower error value across a larger number of pixels. Since within this task our objective is to spatially resolve from depth measurements, Spike-SPI can be seen to quantitatively and qualitatively outperform the ANN.

6.4.1 Spiking Benefits

From the results discussed so far, the main benefits of the Spike-SPI network stem from the CNN approach of the SCNN and not the spiking elements. To see the results of the spiking neurons the histograms must be feed into a simulation with an asynchronous processing pipeline. By doing so the benefits of the asynchronous nature of SNNs can be made clearly visible, not only can the network process smaller chunks of data savingin sensing latency, but the sparse processing nature allows for a reduction in overall computation. From this simulation, the benefits of a spiking approach can be illustrated by displaying the neural activities and system out, as seen in Figure 6.8, where the results of the spiking impact on the network are seen with the spike rates of a cross-section of neurons shown within the middle section called Neural Activities, for the given input histogram with corresponding ground truth image. Figure 6.8 also shows the output of the spiking network at given time steps, as the information is processed asynchronously. Figure 6.8 (a), showing that not only is Spike-SPI better at image reconstruction, due to the CNN Encoder-Decoder structure, but it can achieve this with less processing power. By highlighting this reduction in processing power with the displaying of the spiking activity in the networks over 60 time steps within the simulation. Figure 6.8 (a) shows an average neuron firing rate of 1Hz with a maximum neuron rate of 150Hz, while the overall activity rate is only 11%. This method also has the ability to scale the neurons firing rate, meaning more processing power being drawn, but as a result, the simulation requires fewer steps to produce an image as shown in Figure 6.8 (b), where now the image is mostly formed by the 18th time step, and by the 24th it is resolved but with the background needing to settle. Figure 6.8 (b) also shows that some of the neurons are now firing twice during the activity, with similar higher activity in the mean, max and activity rate overall. Even with this increase only a quarter of the overall neurons available are active. This ability to process with fewer neurons is in fact a negative characteristic of the CNN structure, that the SCNN can exploit. Often within a CNN many of the neurons are not propagating useful information forward, in that they are only reporting a very small similarity of the kernel within a location of the image. As highlighted in Figure 6.8, the majority of neurons in our model at any given time are inactive. However, within the typical CNN approach, there is no neuron threshold to stop the forward propagation of this information. This reduction in information propagation can lead to a reduction in reconstruction accuracy if the hyper-parameters of the spiking neurons are not correctly set. Although, throughout the testing in this chapter no conceivable difference was recorded between the CNN and SCNN models in both qualitative and quantitative terms, and as such the CNN results are not compared.

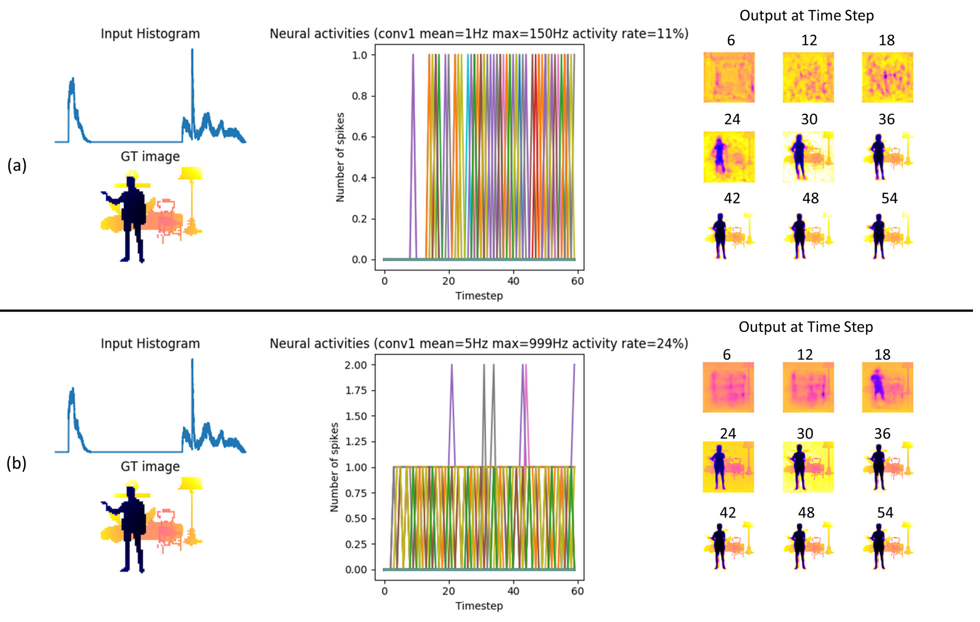


Fig. 6.8: (a) illustrates a lower firing rate while (b) illustrates a higher firing rate with faster processing but more neural activity.

6.4.2 Feature Visualisation

This section is used to form insight and understanding into what the SCNN network, Spike-SPI is processing. That is, what information does the network use to help understand the histogram and convert it into a depth map. Similar to sections 4.4.3 and 5.4 in the previous chapters, this visualisation can help to build understanding for the user through greater interpretability for the network. However, with a converted SCNN approach the network was trained with back-propagation like a traditional CNN. As a result, the weights no longer resembling commonly occurring features, but instead resemble the features deemed best to get the least error back-propagation in terms of the loss function, which in this case was the Huber loss (a combination of mean square error and mean absolute error), in other words, the difference between the output image and ground truth. For this type of learning, CNN trained converted SCNN, understanding of the network can be formed through weight and feature map visualisation. Figure 6.9 shows the weights and features maps associated with Spike-SPI from the Photon Count 9500, IRF 20, where weights and feature maps from 4 layers are illustrated. The first of these is the Conv 1 Encoding layer of the encoding, which looks at the histogram and tries to find useful spatial features that can help to describe the histogram within a windowed area covering 7 time bins. Figure 6.9 displays the corresponding weights from this process and the resulting feature map of each of the weights. The feature maps at this point look very similar as the kernel size of 7 compared to 8000 is relatively small. In the Conv 9 layer of the network, the relative size of the kernel to the start histogram is considerably larger with each kernel now covering around 500 time bins. The resulting feature maps now depict a latent representation of the original histogram, with feature maps highlighting if the feature belonged to an object that was shallow of deep within the scene. It can also be seen within Figure 6.9 that some of the feature maps of Conv 9 have a horizontal line meaning no useful information from this weight is found. Considering 3 of the 24 maps shown contain this out of the 256 available it is apparent how the spiking neuron can reduce the neuron activity of the network. The second half of Figure 6.9 illustrates the 2D convolution weights and feature maps of the decoding process. These 2D weights decode both the spatial and depth information from the latent space, with the feature maps showing the output of the network for the corresponding weight. Conv 4 has a much more subtle output compared to Conv 1 Decoding, in which the resulting feature maps highlight the weight interest area. That being either the silhouette, scene objects or background. Similar to the encoding process the decoder also has some neurons that are essentially inactive depicted with all back feature maps. The image associated with these activations is the same as the top image in Figure 6.7, which is why the Conv 1 decoding feature maps appear to show the silhouette of a woman on the left of the scene. Overall it is apparent that these features are not quite as useful as the ones seen previous in sections 4.4.3 and 5.4. This is due to the visualisation for Spike-SPI showing the statistically salient features in terms of backpropagation, while the previous STDP research illustrated the most common salient features. However, the feature map representation for Spike-SPI does still allow some understanding as to what features are important within the network helping with interpretability.

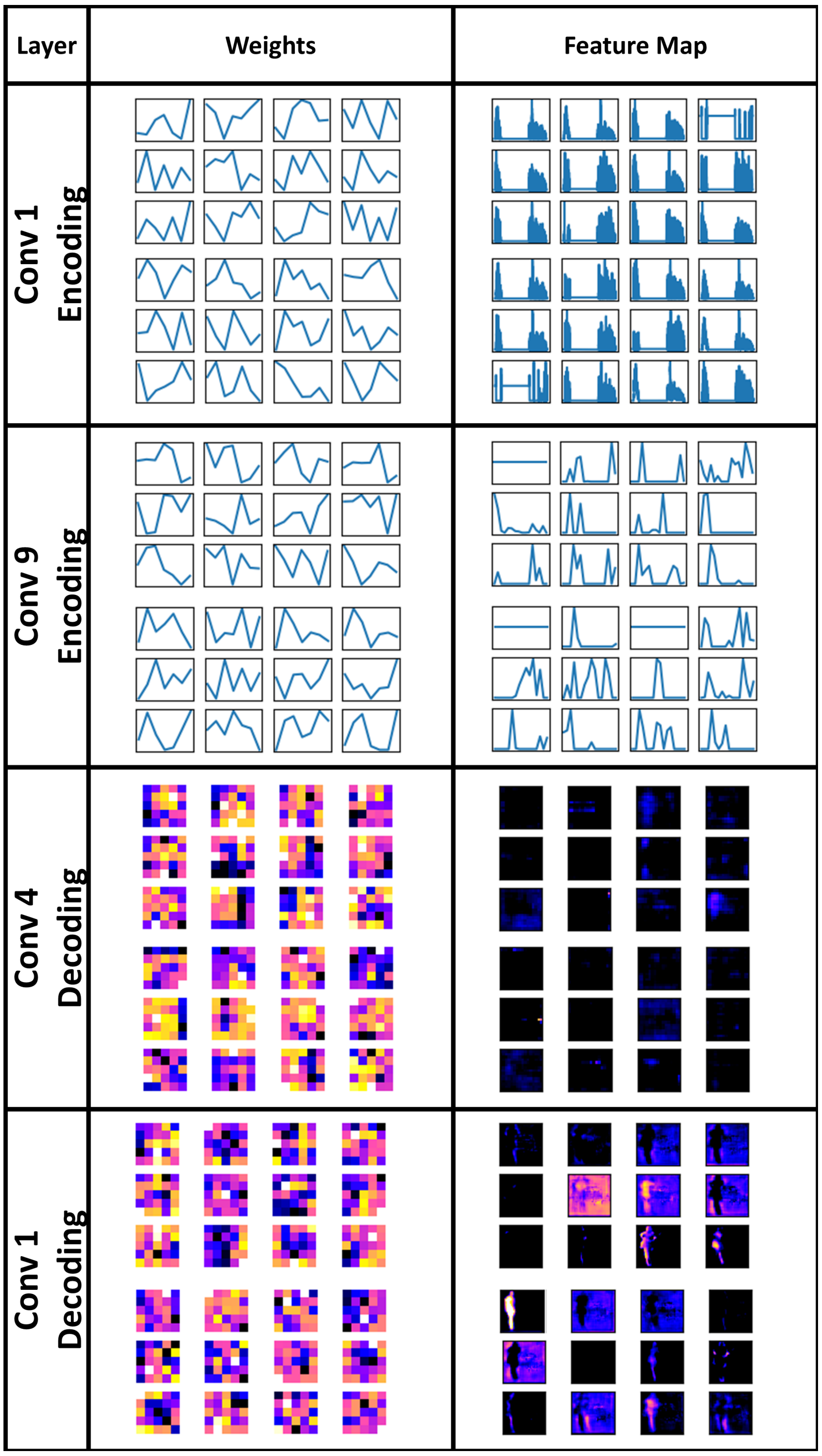


Fig. 6.9: Showing selected weight and feature map results from the 9500 Photon, 20ps IRF data. From top to bottom depicts the encoding early and late stage layers weights and feature maps, then a late and early stage decoding layers weights and feature maps. Feature maps are the results of the activation of the weights to a given input, with the equivalent weight and feature map being collocated within respective columns.

6.5 Discussion

The experiments were carried out in scenes where objects were moving in front of a static scene. This makes the approach well suited for applications where the device needs to be placed at a fixed position during operation, i.e. with a fixed scene. There are multiple situations where operating in a fixed environment is useful. Examples are surveillance and security in public spaces, etc. These are examples where the scene and background (e.g. walls of the room, buildings) do not change at all and they are also very widespread scenarios. Currently, cities have spaces that are constantly monitored with CCTV cameras that also potentially record information from which it is possible to extract information that breaches data protection policies. Our approach is therefore useful for cases where one requires human activity in a fixed area and in a data-compliant way. The approach shown here would be also valid in a slowly changing environment, where training could in principle be continuously updated. Indeed, background objects within the scene will appear static if they change at a slower rate (and/or are at a larger distance) with respect to the dynamic elements of the scene or slower than the acquisition rate of the sensor. An interesting route for future research is, to also investigate methods that account for dynamic scenes, especially considering the new ability of asynchronous processing at lower photo return counts, which would allow for a much shorter determination of what is relatively “static” and “dynamic”.

6.6 Conclusion

Throughout this chapter, a novel approach to depth imaging is shown to be able to outperform the previous state of the art. Spike-SPI not only delivers better spatial reconstruction and depth estimation within the depth maps but is able to do so in an asynchronous spiking manner. This allows not only a theoretical reduction in processing time but an actual reduction in the amount of processing power required to produce an image, thanks to less neuron activity and the ability to produce a depth map with fewer photons captured. These characteristics are illustrated with Spike-SPI being able to resolve the scene with fewer photons to a higher fidelity, while only using a fraction of the computations. Through utilising multiple aspects of a variety of machine and deep learning approaches, Spike-SPI is able to exploits the useful characteristic of these approaches while offsetting the drawbacks. This research highlights the benefits of a pragmatic approach to problem solving, utilising benefits of many systems to deliver state of the art results.

Chapter 7

Conclusions and Future Work

Throughout this thesis, a range of novel event-driven processing techniques and SNNs have been proposed to address a variety of image processing problems. With novel solutions to semantic and instance segmentation using unsupervised biologically inspired techniques, along with a novel neuromorphic solution to the temporal single point imaging problem, using a CNN to SCNN conversion. Offering neuromorphic benefits such as asynchronous, low latency and low computational processing, which help to exploit the asynchronous advantage of the sensor used.

Chapter 4 presented a novel spiking encoder/decoder architecture for use with a Retinamorphic sensor and Neuromorphic processing. The proposed network SpikeSEG was utilised within the PUA framework, which allows the sparse asynchronous benefits of the sensor to be passed through the SpikeSEG network and into an action-based spiking controller. The SpikeSEG network was able to successfully semantically segment up to 10 individual classes within the N-Caltech101 dataset, even under adverse noisy conditions. The SpikeSEG network was able to deliver accurate results thanks to the addition of the novel adaptive thresholding and neuron pruning algorithms. Increasing the quality of learned feature representations achievable with the network, which was then able to help with network visualisation and interpretability. Overall the chapter was able to provide an alternative method to the Perception-Action cycle with the added element of semantic contextual understanding. A few possible extensions of this work are to implement a more sophisticated spiking controller which could allow the output of the network to be dealt with in a more proficient manner over the current proportional controller. Another advancement that could be made to the system is to introduce a reinforcement element to the training, maintaining the STDP learning mechanism however with the additional guidance of a loss function to help tackle the inter- / intra-class variational limits of STDP.

Chapter 5 details the design of a novel featural-temporal matching algorithm called HULK-SMASH, a method for unsupervised spiking object detection and instance segmentation. The unsupervised method makes use of the STDP learned features shown within Chapter 4 unravelling each individual latent spiking neuron in the layer back to the pixel space. This allows both in sequence class instance recognition and sequence to sequence recognition of that same class instance even within a high noise environment and with full occlusion recovery. The HULK-SMASH algorithm was shown to correctly cluster the same people within the NCaltech101 Face category without any supervision, through saliency (early spiking) of commonly occurring spatial features. Additional features that could be added to the HULK-SMASH algorithm are the use of a SNN instead of the Jaccard similarity and a wider investigation into the featural-temporal patterns that exist in other SNNs. Currently, the similarity metric compares the ratio of matched to unmatched featural-temporal information over a given sequence, however further work could investigate the potential of smaller patches of the featural-temporal space being used to identify patterns through the use of an SNN. Furthermore, these featural-temporal patterns used for matching could be identified in other network structure other than SpikeSEG, perhaps even networks without pseudo classification layers, testing the importance of feature timing occurrence for identification.

Chapter 6 presented a novel spiking temporal imaging network Spike-SPI, which was shown to have state of the art performance, delivering better spatial reconstruction and depth estimation, while requiring fewer data to be captured per histogram and allowing a faster and less computationally expensive processing system. Spike-SPI exploits all the features of CNNs, SNNs and encoder-decoder networks with a pragmatic engineering approach exploiting the feature extraction capabilities of CNNs, asynchronous processing capabilities of a SNN and the high dimensional compressed latent space representations associated with the encoder-decoder networks. This bespoke solution helps to realise the processing ability of a single point ToF sensor for imaging purposes, with the lower photon required and asynchronous processing potentially allowing easier implementation. Future iterations of this work could look at removing the need to collect the histograms at all, instead, using the SNN directly to capture the temporal information, allowing the minimal amount of processing to produce an image. Another avenue for further research could see the removal of the conversion process, instead of making use of one of the spiking supervised methods for training, coupled with an investigation into better comparison metrics for training, as it was shown that some metrics interpret spatial errors as depth errors.

Lastly, a number of the algorithms and network presented through Chapters 4, 5 and 6 could be adapted and tested on neuromorphic hardware to explore and demonstrate the low latency and computational advantages.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. 1

[2] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, “Deep learning for computer vision: A brief review,” *Computational intelligence and neuroscience*, vol. 2018, 2018. 1

[3] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.,” *Psychological review*, vol. 65, no. 6, p. 386, 1958. 1, 2, 9, 29, 45

[4] K. Fukushima, “Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.,” *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980. 1, 11, 12, 15, 29, 68

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, pp. 1097–1105, 2012. 1, 7, 16, 18, 21

[6] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016. 1, 18, 21, 25

[7] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Advances in neural information processing systems*, pp. 91–99, 2015. 1, 18

[8] J. Dai, Y. Li, K. He, and J. Sun, “R-fcn: Object detection via region-based fully convolutional networks,” in *Advances in neural information processing systems*, pp. 379–387, 2016. 1, 18

[9] L. Wang, W. Ouyang, X. Wang, and H. Lu, “Visual tracking with fully convolutional networks,” in *Proceedings of the IEEE international conference on computer vision*, pp. 3119–3127, 2015. 1

[10] H. Nam and B. Han, “Learning multi-domain convolutional neural networks for visual tracking,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4293–4302, 2016. 1

[11] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, IEEE, jun 2015. 1, 18, 20, 21, 22, 23, 24, 64, 70

[12] V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 2481–2495, dec 2017. 1, 8, 18, 23, 24, 25, 70

[13] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015. 1, 18, 24, 26

[14] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017. 1, 18

[15] G. Marcus, "Deep learning: A critical appraisal," *arXiv preprint arXiv:1801.00631*, 2018. 2, 3

[16] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, dec 1943. 2, 7, 8, 29

[17] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex.," *The Journal of physiology*, vol. 160, pp. 106–54, jan 1962. 2, 11, 40

[18] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75–88, 2019. 2

[19] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," *arXiv preprint arXiv:2007.05558*, 2020. 2

[20] M. Mahowald, "The silicon retina," in *An Analog VLSI System for Stereoscopic Vision*, pp. 4–65, Springer, 1994. 2

[21] C. Mead, *Analog VLSI and neural systems.* Addison-Wesley Longman Publishing Co., Inc., 1989. 2, 29

[22] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990. 2, 51

[23] "Hebb, D. O. Organization of behavior. New York: Wiley, 1949, pp. 335, $4.00," *Journal of Clinical Psychology*, vol. 6, pp. 307–307, jul 1949. 2, 8, 43

[24] G.-q. Bi and M.-m. Poo, "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type," *Journal of neuroscience*, vol. 18, no. 24, pp. 10464–10472, 1998. 2, 43, 44, 54, 59, 68

[25] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997. 2, 41

[26] D. Zambrano, R. Nusselder, H. S. Scholte, and S. M. Bohté, "Sparse computation in adaptive spiking neural networks," *Frontiers in Neuroscience*, vol. 12, p. 987, 2019. 2, 66

[27] V. Demin and D. Nekhaev, "Recurrent spiking neural network learning based on a competitive maximization of neuronal activity," *Frontiers in Neuroinformatics*, vol. 12, p. 79, 2018. 2

[28] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: opportunities and challenges," *Frontiers in neuroscience*, vol. 12, p. 774, 2018. 2

[29] P. Panda, S. A. Aketi, and K. Roy, "Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization," *Frontiers in Neuroscience*, vol. 14, 2020. 2

[30] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, A. J. Leutenegger, S. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. 2

[31] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2004-Janua, pp. 178–178, IEEE, 2004. 7, 61, 62, 90

[32] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015. 7

[33] S. Linnainmaa, "The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors," *Master's Thesis (in Finnish), Univ. Helsinki*, pp. 6–7, 1970. 7

[34] P. Werbos, "Beyond regression:" new tools for prediction and analysis in the behavioral sciences," *Ph. D. dissertation, Harvard University*, 1974. 7, 10

[35] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," 2006. 7

[36] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th annual international conference on machine learning*, pp. 873–880, 2009. 7, 17

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015. 7

[38] M. Minsky and S. A. Papert, *Perceptrons: An introduction to computational geometry*. MIT press, 2017 (1969). 10

[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986. 10

[40] D. B. Parker, "Learning logic technical report tr-47," *Center of Computational Research in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, MA*, 1985. 10

[41] Y. LeCun, "A learning scheme for asymmetric threshold networks," *Proceedings of Cognitiva*, vol. 85, no. 537, pp. 599–604, 1985. 10

[42] K. Hornik, M. Stinchcombe, H. White, *et al.*, "Multilayer feedforward networks are universal approximators.," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989. 10

[43] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat," *Journal of neurophysiology*, vol. 28, no. 2, pp. 229–289, 1965. 11

[44] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989. 11, 13

[45] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. 11, 15

[46] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. 13, 15

[47] M. A. Nielsen, *Neural networks and deep learning*, vol. 2018. Determination press San Francisco, CA, 2015. 14

[48] Y.-T. Zhou and R. Chellappa, "Computation of optical flow using a neural network.," in *ICNN*, pp. 71–78, 1988. 15

[49] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014. 15, 48

[50] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006. 16, 70

[51] A.-r. Mohamed, G. E. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *IEEE transactions on audio, speech, and language processing*, vol. 20, no. 1, pp. 14–22, 2011. 16

[52] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," in *2009 IEEE 12th international conference on computer vision*, pp. 2146–2153, IEEE, 2009. 17

[53] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010. 17

[54] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011. 17, 18

[55] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, p. 3, 2013. 17

[56] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012. 17

[57] E. Shelhamer, J. Long, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 640–651, apr 2017. 18, 21, 22

[58] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár, "Panoptic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 9404–9413, 2019. 21

[59] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. 21, 70, 90

[60] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, IEEE, jun 2015. 21

[61] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *2010 IEEE Computer Society Conference on computer vision and pattern recognition*, pp. 2528–2535, IEEE, 2010. 22, 23

[62] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, (Cambridge, MA, USA), p. 2366–2374, MIT Press, 2014. 23, 130

[63] H. Noh, S. Hong, and B. Han, "Learning Deconvolution Network for Semantic Segmentation," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1520–1528, IEEE, dec 2015. 23, 24

[64] M. D. Zeiler, G. W. Taylor, and R. Fergus, "Adaptive deconvolutional networks for mid and high level feature learning," in *2011 International Conference on Computer Vision*, pp. 2018–2025, IEEE, 2011. 23

[65] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, pp. 818–833, Springer, 2014. 23, 70

[66] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *European conference on computer vision*, pp. 391–405, Springer, 2014. 23

[67] M. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. LeCun, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *2007 IEEE conference on computer vision and pattern recognition*, pp. 1–8, IEEE, 2007. 23, 70

[68] V. Badrinarayanan, A. Handa, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling," *arXiv preprint arXiv:1505.07293*, 2015. 24

[69] M. Drozdzal, E. Vorontsov, G. Chartrand, S. Kadoury, and C. Pal, "The importance of skip connections in biomedical image segmentation," in *Deep Learning and Data Labeling for Medical Applications*, pp. 179–187, Springer, 2016. 25

[70] T. Hey, "Richard feynman and computation," *Contemporary Physics*, vol. 40, no. 4, pp. 257–265, 1999. 29

[71] S. Choudhary, S. Sloan, S. Fok, A. Neckar, E. Trautmann, P. Gao, T. Stewart, C. Eliasmith, and K. Boahen, "Silicon neurons that compute," in *International conference on artificial neural networks*, pp. 121–128, Springer, 2012. 29

[72] E. M. Izhikevich, "Which model to use for cortical spiking neurons?," *IEEE transactions on neural networks*, vol. 15, no. 5, pp. 1063–1070, 2004. 29, 31, 37

[73] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, C. Tang, and D. Rasmussen, "A large-scale model of the functioning brain.," *Science (New York, N.Y.)*, vol. 338, pp. 1202–5, nov 2012. 29, 47, 60

[74] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network.," *Frontiers in neuroscience*, vol. 7, p. 178, 2013. 30

[75] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 2015-Septe, Institute of Electrical and Electronics Engineers Inc., sep 2015. 30, 47, 48, 49

[76] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in neural information processing systems*, pp. 1117–1125, 2015. 30

[77] E. Hunsberger and C. Eliasmith, "Spiking deep networks with lif neurons," *arXiv preprint arXiv:1510.08829*, 2015. 30

[78] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in Neuroscience*, vol. 11, p. 682, 2017. 30, 47

[79] D. P. Moeys, F. Corradi, E. Kerr, P. Vance, G. Das, D. Neil, D. Kerr, and T. Delbrück, "Steering a predator robot using a mixed frame/event-driven convolutional neural network," in *2016 Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, pp. 1–8, IEEE, 2016. 30

[80] I.-A. Lungu, F. Corradi, and T. Delbrück, "Live demonstration: Convolutional neural network driven by dynamic vision sensor playing roshambo," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–1, IEEE, 2017. 30

[81] L. Wang, Y.-S. Ho, K.-J. Yoon, *et al.*, "Event-based high dynamic range image and very high frame rate video generation using conditional generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10081–10090, 2019. 30

[82] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, "High speed and high dynamic range video with an event camera," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019. 30

[83] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "Ev-flownet: Self-supervised optical flow estimation for event-based cameras," *arXiv preprint arXiv:1802.06898*, 2018. 30

[84] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "Unsupervised event-based learning of optical flow, depth, and egomotion," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 989–997, 2019. 30

[85] H. Paugam-Moisy and S. M. Bohte, "Computing with Spiking Neuron Networks," in *Handbook of Natural Computing* (G. Rozenberg, T. Back, and J. Kok, eds.), pp. 335–376, Springer-Verlag, Sept. 2012. 30

[86] L. Lapicque, "Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization," *Journal de Physiologie et de Pathologie Generalej*, vol. 9, pp. 620–635, 1907. 31, 32, 33

[87] A. L. HODGKIN and A. F. HUXLEY, "A quantitative description of membrane current and its application to conduction and excitation in nerve.," *The Journal of physiology*, vol. 117, pp. 500–44, aug 1952. 31, 35, 36

[88] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, vol. 99, pp. 56–67, mar 2018. 32, 41, 60, 61, 64, 68, 75

[89] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking neural networks hardware implementations and challenges: A survey," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15, no. 2, pp. 1–35, 2019. 34

[90] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal dynamics: From single neurons to networks and models of cognition.* Cambridge University Press, 2014. 35, 36, 39, 40

[91] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014. 35, 51

[92] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014. 35, 50

[93] S. A. Aamir, Y. Stradmann, P. Müller, C. Pehle, A. Hartel, A. Grübl, J. Schemmel, and K. Meier, "An accelerated lif neuronal network array for a large-scale mixed-signal neuromorphic architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 12, pp. 4299–4312, 2018. 35

[94] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, pp. 82–99, jan 2018. 35, 51, 65

[95] K. S. Cole and H. J. Curtis, "Electric impedance of the squid giant axon during activity," *The Journal of general physiology*, vol. 22, no. 5, pp. 649–670, 1939. 35

[96] D. H. Perkel and T. H. Bullock, "Neural coding.," *Neurosciences Research Program Bulletin*, 1968. 37

[97] R. V. Rullen and S. J. Thorpe, "Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex," *Neural computation*, vol. 13, no. 6, pp. 1255–1283, 2001. 37

[98] A. Kumar, S. Rotter, and A. Aertsen, "Spiking activity propagation in neuronal networks: reconciling different perspectives on neural coding," *Nature reviews Neuroscience*, vol. 11, no. 9, pp. 615–627, 2010. 37

[99] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," *nature*, vol. 381, no. 6582, pp. 520–522, 1996. 38, 41

[100] A. L. Fairhall, G. D. Lewen, W. Bialek, and R. R. d. R. van Steveninck, "Efficiency and ambiguity in an adaptive neural code," *Nature*, vol. 412, no. 6849, pp. 787–792, 2001. 38

[101] W. Gerstner, "Population dynamics of spiking neurons: fast transients, asynchronous states, and locking," *Neural computation*, vol. 12, no. 1, pp. 43–89, 2000. 40

[102] N. Brunel, F. S. Chance, N. Fourcaud, and L. Abbott, "Effects of synaptic noise and filtering on the frequency response of spiking neurons," *Physical Review Letters*, vol. 86, no. 10, p. 2186, 2001. 40

[103] Q. Zhou, Y. Shi, Z. Xu, R. Qu, and G. Xu, "Classifying melanoma skin lesions using convolutional spiking neural networks with unsupervised stdp learning rule," *IEEE Access*, 2020. 41

[104] J. J. Hopfield, "Pattern recognition computation using action potential timing for stimulus representation," *Nature*, vol. 376, no. 6535, pp. 33–36, 1995. 41

[105] R. S. Johansson and I. Birznieks, "First spikes in ensembles of human tactile afferents code complex spatial fingertip events," *Nature Neuroscience*, vol. 7, no. 2, pp. 170–177, 2004. 41

[106] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 7, pp. 3227–3235, 2017. 41, 46, 125

[107] S. R. Kheradpisheh and T. Masquelier, "S4nn: temporal backpropagation for spiking neural networks with one spike per neuron," *International Journal of Neural Systems*, vol. 30, no. 6, p. 2050027, 2020. 41

[108] S. M. Bohte, "The evidence for neural information processing with precise spike-times: A survey," *Natural Computing*, vol. 3, no. 2, pp. 195–206, 2004. 42

[109] A. Borst and F. E. Theunissen, "Information theory and neural coding," *Nature Neuroscience*, vol. 2, no. 11, pp. 947–957, 1999. 42

[110] W. Maass, R. Legenstein, and H. Markram, "A new approach towards vision suggested by biologically realistic neural microcircuit models," in *International Workshop on Biologically Motivated Computer Vision*, pp. 282–293, Springer, 2002. 42

[111] F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting," *Neural computation*, vol. 22, no. 2, pp. 467–510, 2010. 42

[112] A. Morrison, M. Diesmann, and W. Gerstner, "Phenomenological models of synaptic plasticity based on spike timing," *Biological cybernetics*, vol. 98, no. 6, pp. 459–478, 2008. 43

[113] G. G. Turrigiano and S. B. Nelson, "Homeostatic plasticity in the developing nervous system," *Nature reviews Neuroscience*, vol. 5, no. 2, pp. 97–107, 2004. 43

[114] A. Artola, S. Bröcher, and W. Singer, "Different voltage-dependent thresholds for inducing long-term depression and long-term potentiation in slices of rat visual cortex," *Nature*, vol. 347, pp. 69–72, 1990. 43

[115] E. Vasilaki and M. Giugliano, "Emergence of connectivity motifs in networks of model neurons with short-and long-term plastic synapses," *PloS one*, vol. 9, no. 1, p. e84626, 2014. 43

[116] C. Clopath, L. Büsing, E. Vasilaki, and W. Gerstner, "Connectivity reflects coding: a model of voltage-based stdp with homeostasis," *Nature Neuroscience*, vol. 13, no. 3, p. 344, 2010. 43

[117] H. Markram, J. Lübke, M. Frotscher, and B. Sakmann, "Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps," *Science*, vol. 275, no. 5297, pp. 213–215, 1997. 43

[118] P. J. Sjostrom, E. A. Rancz, A. Roth, and M. Hausser, "Dendritic excitability and synaptic plasticity," *Physiological reviews*, vol. 88, no. 2, pp. 769–840, 2008. 43

[119] G.-q. Bi and M.-m. Poo, "Synaptic modification by correlated activity: Hebb's postulate revisited," *Annual review of Neuroscience*, vol. 24, no. 1, pp. 139–166, 2001. 43

[120] W. Gerstner, R. Kempter, J. L. Van Hemmen, and H. Wagner, "A neuronal learning rule for sub-millisecond temporal coding," *Nature*, vol. 383, no. 6595, pp. 76–78, 1996. 45, 54

[121] R. Kempter, W. Gerstner, and J. L. Van Hemmen, "Hebbian learning and spiking neurons," *Physical Review E*, vol. 59, no. 4, p. 4498, 1999. 45

[122] L. I. Zhang, H. W. Tao, C. E. Holt, W. A. Harris, and M.-m. Poo, "A critical window for cooperation and competition among developing retinotectal synapses," *Nature*, vol. 395, no. 6697, pp. 37–44, 1998. 45

[123] S. Song, K. D. Miller, and L. F. Abbott, "Competitive hebbian learning through spike-timing-dependent synaptic plasticity," *Nature Neuroscience*, vol. 3, no. 9, pp. 919–926, 2000. 45

[124] E. I. Knudsen, "Supervised learning in the brain," *Journal of Neuroscience*, vol. 14, no. 7, pp. 3985–3997, 1994. 45

[125] J. I. Glaser, A. S. Benjamin, R. Farhoodi, and K. P. Kording, "The roles of supervised machine learning in systems neuroscience," *Progress in Neurobiology*, vol. 175, pp. 126–137, 2019. 45

[126] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1-4, pp. 17–37, 2002. 45, 46

[127] F. Ponulak, "Resume-new supervised learning method for spiking neural networks technical report," *Institute of Control and Information Engineering, Poznan University of Technology*, 2005. 45

[128] R. V. Florian, "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity," *Neural computation*, vol. 19, no. 6, pp. 1468–1502, 2007. 45

[129] R. V. Florian, "The chronotron: A neuron that learns to fire temporally precise spike patterns," *PloS one*, vol. 7, no. 8, p. e40233, 2012. 45

[130] A. Mohemmed, S. Schliebs, S. Matsuda, and N. Kasabov, "Span: Spike pattern association neuron for learning spatio-temporal spike patterns," *International journal of neural systems*, vol. 22, no. 04, p. 1250012, 2012. 45, 46

[131] J. C. Tapson, G. K. Cohen, S. Afshar, K. M. Stiefel, Y. Buskila, T. J. Hamilton, and A. van Schaik, "Synthesis of neural networks for spatio-temporal spike pattern recognition and processing," *Frontiers in Neuroscience*, vol. 7, p. 153, 2013. 45

[132] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in Neuroscience*, vol. 10, p. 508, 2016. 46, 68

[133] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in Neuroscience*, vol. 12, p. 331, 2018. 46

[134] D. Huh and T. J. Sejnowski, "Gradient descent for spiking neural networks," in *Advances in Neural Information Processing Systems*, pp. 1433–1443, 2018. 46

[135] F. Zenke and S. Ganguli, "SuperSpike: Supervised learning in multilayer spiking neural networks," *Neural Computation*, vol. 30, pp. 1514–1541, jun 2018. 46

[136] S. B. Shrestha and G. Orchard, "Slayer: Spike layer error reassignment in time," in *Advances in Neural Information Processing Systems*, pp. 1412–1421, 2018. 46, 54

[137] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. Gómez-Rodríguez, L. Camuñas-Mesa, R. Berner, M. Rivas-Pérez, T. Delbruck, *et al.*, "Caviar: A 45k neuron, 5m synapse, 12g connects/s aer hardware sensory–processing–learning–actuating system for high-speed visual object recognition and tracking," *IEEE transactions on neural networks*, vol. 20, no. 9, pp. 1417–1438, 2009. 47

[138] J. A. Pérez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen, and B. Linares-Barranco, "Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward convnets," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 11, pp. 2706–2719, 2013. 47

[139] C. Eliasmith and C. Anderson, "Neural engineering: Computation," *Representation, and Dynamics in Neurobiological Systems*, 2004. 47, 51

[140] Y. Hu, H. Tang, Y. Wang, and G. Pan, "Spiking deep residual network," *arXiv preprint arXiv:1805.01352*, 2018. 47

[141] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: very and residual architectures," *Frontiers in neuroscience*, vol. 13, 2019. 47

[142] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith, "Benchmarking keyword spotting efficiency on neuromorphic hardware," in *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, pp. 1–8, 2019. 47

[143] A. J. Yu, M. A. Giese, and T. A. Poggio, "Biophysiologically plausible implementations of the maximum operation," *Neural computation*, vol. 14, no. 12, pp. 2857–2881, 2002. 47

[144] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015. 47, 48

[145] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman, "Hfirst: a temporal approach to object recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 10, pp. 2028–2040, 2015. 48

[146] D. Neil, M. Pfeiffer, and S.-C. Liu, "Learning to be efficient: Algorithms for training low-latency, low-compute deep spiking neural networks," in *Proceedings of the 31st annual ACM symposium on applied computing*, pp. 293–298, 2016. 48, 49

[147] D. Rasmussen, "NengoDL: Combining deep learning and neuromorphic modelling methods," *arXiv*, vol. 1805.11144, pp. 1–22, 2018. 48, 54, 129

[148] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org. 48, 54

[149] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017. 48

[150] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014. 50

[151] H. Markram, K. Meier, T. Lippert, S. Grillner, R. Frackowiak, S. Dehaene, A. Knoll, H. Sompolinsky, K. Verstreken, J. DeFelipe, *et al.*, "Introducing the human brain project," *Procedia Computer Science*, vol. 7, pp. 39–42, 2011. 50

[152] C. Mayr, S. Hoeppner, and S. Furber, "Spinnaker 2: A 10 million core processor system for brain simulation and machine learning," *arXiv preprint arXiv:1911.02385*, 2019. 50

[153] J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 1947–1950, IEEE, 2010. 50

[154] A. Neckar, S. Fok, B. V. Benjamin, T. C. Stewart, N. N. Oza, A. R. Voelker, C. Eliasmith, R. Manohar, and K. Boahen, "Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 144–164, 2018. 51

[155] E. P. Frady, G. Orchard, D. Florey, N. Imam, R. Liu, J. Mishra, J. Tse, A. Wild, F. T. Sommer, and M. Davies, "Neuromorphic nearest-neighbor search using intel's pohoiki springs," *arXiv preprint arXiv:2004.12691*, 2020. 51

[156] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps)," *IEEE transactions on biomedical circuits and systems*, vol. 12, no. 1, pp. 106–122, 2017. 51

[157] C. S. Thakur, J. L. Molin, G. Cauwenberghs, G. Indiveri, K. Kumar, N. Qiao, J. Schemmel, R. Wang, E. Chicca, J. Olson Hasler, *et al.*, "Large-scale neuromorphic spiking array processors: A quest to mimic the brain," *Frontiers in Neuroscience*, vol. 12, p. 891, 2018. 51

[158] Q. Liu, O. Richter, C. Nielsen, S. Sheik, G. Indiveri, and N. Qiao, "Live demonstration: Face recognition on an ultra-low power event-driven convolutional neural network asic," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0, 2019. 51

[159] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 120 dB 15micro s Latency Asynchronous Temporal Contrast Vision Sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008. 51, 59, 61

[160] C. Posch, D. Matolin, and R. Wohlgenannt, "A QVGA 143 dB Dynamic Range Frame-Free PWM Image Sensor With Lossless Pixel-Level Video Compression and Time-Domain CDS," *IEEE JOURNAL OF SOLID-STATE CIRCUITS*, vol. 46, no. 1, p. 259, 2011. 51

[161] C. Brandli, R. Berner, Minhao Yang, Shih-Chii Liu, and T. Delbruck, "A 240 x 180 130 dB 3 \mui s Latency Global Shutter Spatiotemporal Vision Sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, pp. 2333–2341, oct 2014. 51, 52, 61

[162] B. Son, Y. Suh, S. Kim, H. Jung, J.-S. Kim, C. Shin, K. Park, K. Lee, J. Park, J. Woo, Y. Roh, H. Lee, Y. Wang, I. Ovsiannikov, and H. Ryu, "4.1 A 640x480 dynamic vision sensor with a 9\(\mu\)m pixel and 300Meps address-event representation," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 66–67, IEEE, feb 2017. 51, 52

[163] V. Chan, S.-C. Liu, and A. van Schaik, "Aer ear: A matched silicon cochlea pair with address event representation interface," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 1, pp. 48–59, 2007.

[164] S.-C. Liu, A. van Schaik, B. A. Minch, and T. Delbruck, "Asynchronous binaural spatial audition sensor with 2 x 64 x 4 channel output," *IEEE transactions on biomedical circuits and systems*, vol. 8, no. 4, pp. 453–464, 2013. 52

[165] J. W. Gardner and P. N. Bartlett, "A brief history of electronic noses," *Sensors and Actuators B: Chemical*, vol. 18, no. 1-3, pp. 210–211, 1994. 52

[166] W. W. Lee, S. L. Kukreja, and N. V. Thakor, "Discrimination of dynamic tactile contact by temporally precise event sensing in spiking neuromorphic networks," *Frontiers in Neuroscience*, vol. 11, p. 5, 2017. 52

[167] K. A. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 5, pp. 416–434, 2000. 52

[168] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019. 54

[169] M. L. Hines and N. T. Carnevale, "The neuron simulation environment," *Neural computation*, vol. 9, no. 6, pp. 1179–1209, 1997. 54

[170] N. T. Carnevale and M. L. Hines, *The NEURON book*. Cambridge University Press, 2006. 54

[171] M.-O. Gewaltig and M. Diesmann, "Nest (neural simulation tool)," *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007. 54

[172] D. F. Goodman and R. Brette, "Brian: a simulator for spiking neural networks in python," *Frontiers in Neuroinformatics*, vol. 2, p. 5, 2008. 54

[173] J. Vitay, H. Ü. Dinkelbach, and F. H. Hamker, "Annarchy: a code generation approach to neural simulations on parallel hardware," *Frontiers in Neuroinformatics*, vol. 9, p. 19, 2015. 54

[174] H. Cornelis, A. L. Rodriguez, A. D. Coop, and J. M. Bower, "Python as a federation tool for genesis 3.0," *PLoS One*, vol. 7, no. 1, p. e29018, 2012. 54

[175] N. K. Kasabov, "Neucube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data," *Neural Networks*, vol. 52, pp. 62–76, 2014. 54

[176] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. Stewart, D. Rasmussen, X. Choo, A. Voelker, and C. Eliasmith, "Nengo: a Python tool for building large-scale functional brain models," *Frontiers in Neuroinformatics*, vol. 7, no. 48, pp. 1–13, 2014. 54

[177] T. C. Stewart, "A technical overview of the neural engineering framework," *University of Waterloo*, 2012. 54

[178] H. Hazan, D. J. Saunders, H. Khan, D. Patel, D. T. Sanghavi, H. T. Siegelmann, and R. Kozma, "Bindsnet: A machine learning-oriented spiking neural networks library in python," *Frontiers in Neuroinformatics*, vol. 12, p. 89, 2018. 54

[179] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, and T. Masquelier, "Spyketorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron," *Frontiers in Neuroscience*, vol. 13, 2019. 54

[180] N. Frémaux and W. Gerstner, "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules," *Frontiers in neural circuits*, vol. 9, p. 85, 2016. 54

[181] Z. Brzosko, S. Zannone, W. Schultz, C. Clopath, and O. Paulsen, "Sequential neuromodulation of hebbian plasticity offers mechanism for effective reward-based navigation," *Elife*, vol. 6, p. e27756, 2017. 54

[182] M. Mozafari, S. R. Kheradpisheh, T. Masquelier, A. Nowzari-Dalini, and M. Ganjtabesh, "First-spike-based visual categorization using reward-modulated STDP," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, pp. 6178–6190, dec 2018. 54, 94

[183] A. P. Davison, D. Brüderle, J. M. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, "Pynn: a common interface for neuronal network simulators," *Frontiers in Neuroinformatics*, vol. 2, p. 11, 2009. 55

[184] R. W. Sperry, "Neurology and the mind-brain problem," *American scientist*, vol. 40, no. 2, pp. 291–312, 1952. 55

[185] U. S. Bhalla and R. Iyengar, "Emergent properties of networks of biological signaling pathways," *Science*, vol. 283, no. 5400, pp. 381–387, 1999. 55

[186] S. Duran-Nebreda and G. W. Bassel, "Plant behaviour in response to the environment: information processing in the solid state," *Philosophical Transactions of the Royal Society B*, vol. 374, no. 1774, p. 20180370, 2019. 55

[187] T. Stewart, F.-X. Choo, and C. Eliasmith, "Spaun: A perception-cognition-action model using spiking neurons," in *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 34, 2012. 56

[188] J. Conradt, M. Cook, R. Berner, P. Lichtsteiner, R. Douglas, and T. Delbruck, "A pencil balancing robot using a pair of AER dynamic vision sensors," in *2009 IEEE International Symposium on Circuits and Systems*, pp. 781–784, IEEE, may 2009. 59

[189] T. Delbruck and M. Lang, "Robotic goalie with 3 ms reaction time at 4% CPU load using event-based dynamic vision sensor," *Frontiers in Neuroscience*, vol. 7, p. 223, nov 2013. 59

[190] X. Clady, S.-H. Ieng, and R. Benosman, "Asynchronous event-based corner detection and matching," *Neural Networks*, vol. 66, pp. 91–106, jun 2015. 59

[191] V. Vasco, A. Glover, and C. Bartolozzi, "Fast event-based Harris corner detection exploiting the advantages of event-driven cameras," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4144–4149, IEEE, oct 2016. 59

[192] E. Mueggler, C. Bartolozzi, and D. Scaramuzza, "Fast event-based corner detection," 2017. 59, 61

[193] X. Lagorce, C. Meyer, S.-H. Ieng, D. Filliat, and R. Benosman, "Asynchronous Event-Based Multikernel Algorithm for High-Speed Visual Features Tracking," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, pp. 1710–1720, aug 2015. 59, 61

[194] H. Li and L. Shi, "Robust Event-Based Object Tracking Combining Correlation Filter and CNN Representation," *Frontiers in Neurorobotics*, vol. 13, p. 82, oct 2019. 59

[195] O. Bichler, D. Querlioz, S. J. Thorpe, J.-P. Bourgoin, and C. Gamrat, "Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity," *Neural Networks*, vol. 32, pp. 339–348, aug 2012. 60

[196] E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco, "An Event-Driven Classifier for Spiking Neural Networks Fed with Synthetic or Dynamic Vision Sensor Data," *Frontiers in Neuroscience*, vol. 11, p. 350, jun 2017. 60

[197] L. Paulun, A. Wendt, and N. Kasabov, "A Retinotopic Spiking Neural Network System for Accurate Recognition of Moving Objects Using NeuCube and Dynamic Vision Sensors," *Frontiers in Computational Neuroscience*, vol. 12, p. 42, jun 2018. 60

[198] P. Falez, P. Tirilly, I. Marius Bilasco, P. Devienne, and P. Boulet, "Multi-layered Spiking Neural Network with Target Timestamp Threshold Adaptation and STDP," in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, jul 2019. 60, 66

[199] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47–63, mar 2019. 60

[200] G. Wiesmann, S. Schraml, M. Litzenberger, A. N. Belbachir, M. Hofstatter, and C. Bartolozzi, "Event-driven embodied system for feature extraction and object recognition in robotic applications," in *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 76–82, IEEE, jun 2012. 60

[201] S. Seifozzakerini, W.-Y. Yau, B. Zhao, and K. Mao, "Event-Based Hough Transform in a Spiking Neural Network for Multiple Line Detection and Tracking Using a Dynamic Vision Sensor," in *Procedings of the British Machine Vision Conference 2016*, pp. 94.1–94.12, British Machine Vision Association, 2016. 60

[202] Z. Jiang, Z. Bing, K. Huang, and A. Knoll, "Retina-Based Pipe-Like Object Tracking Implemented Through Spiking Neural Network on a Snake Robot," *Frontiers in Neurorobotics*, vol. 13, p. 29, may 2019. 60

[203] S. D. Levy, "Robustness Through Simplicity: A Minimalist Gateway to Neurorobotic Flight," *Frontiers in Neurorobotics*, vol. 14, p. 16, mar 2020. 60

[204] T. DeWolf, T. C. Stewart, J.-J. Slotine, and C. Eliasmith, "A spiking neural model of adaptive arm control.," *Proceedings. Biological sciences*, vol. 283, no. 1843, 2016. 60

[205] P. Kirkland, G. Di Caterina, J. Soraghan, and G. Matich, "Spikeseg: Spiking segmentation via stdp saliency mapping," in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2020. 60, 63, 66, 76

[206] D. Gehrig, H. Rebecq, G. Gallego, and D. Scaramuzza, "Asynchronous, photometric feature tracking using events and frames," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 750–765, 2018. 61

[207] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in neuroscience*, vol. 9, p. 437, 2015. 61, 62, 74, 75, 98, 105, 108

[208] B. Siddoway, H. Hou, and H. Xia, "Molecular mechanisms of homeostatic synaptic downscaling," *Neuropharmacology*, vol. 78, pp. 38–44, 2014. 66

[209] P. Kirkland, G. Di Caterina, J. Soraghan, Y. Andreopoulos, and G. Matich, "Uav detection: a stdp trained deep convolutional spiking neural network retina-neuromorphic approach," in *International Conference on Artificial Neural Networks*, pp. 724–736, Springer, 2019. 66

[210] H. H. Amin, W. Deabes, and K. Bouazza, "Clustering of user activities based on adaptive threshold spiking neural networks," in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 1–6, IEEE, 2017. 66

[211] M. Hopkins, G. Pineda-García, P. A. Bogdan, and S. B. Furber, "Spiking neural networks for computer vision," *Interface Focus*, vol. 8, no. 4, p. 20180007, 2018. 66

[212] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS computational biology*, vol. 3, no. 2, 2007. 68

[213] W. Maass, "On the computational power of winner-take-all," *Neural computation*, vol. 12, no. 11, pp. 2519–2535, 2000. 68

[214] P. Panda, J. M. Allred, S. Ramanathan, and K. Roy, "Asp: Learning to forget with adaptive synaptic plasticity in spiking neural networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 51–64, 2017. 68

[215] A. Renner, M. Evanusa, and Y. Sandamirskaya, "Event-Based Attention and Tracking on Neuromorphic Hardware," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1709–1716, IEEE, jun 2019. 73

[216] P. Jaccard, "Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines," *Bull Soc Vaudoise Sci Nat*, vol. 37, pp. 241–272, 1901. 74, 97, 103, 129

[217] C. L. Zitnick and P. Dollár, "Edge Boxes: Locating Object Proposals from Edges," pp. 391–405, Springer, Cham, 2014. 76

[218] F. Zamani and M. Jamzad, "A feature fusion based localized multiple kernel learning system for real world image classification," *EURASIP Journal on Image and Video Processing*, vol. 2017, p. 78, dec 2017. 79

[219] F. Barranco, C. Fermuller, and Y. Aloimonos, "Contour Motion Estimation for Asynchronous Event-Driven Cameras," *Proceedings of the IEEE*, vol. 102, pp. 1537–1556, oct 2014. 89

[220] C. Olah, A. Mordvintsev, and L. Schubert, "Feature visualization," *Distill*, 2017. https://distill.pub/2017/feature-visualization. 89

[221] R. Gopalakrishnan, Y. Chua, and L. R. Iyer, "Classifying neuromorphic data using a deep learning framework for image classification," in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 1520–1524, IEEE, 2018. 90

[222] H. Kabir, M. Abdar, S. M. J. Jalali, A. Khosravi, A. F. Atiya, S. Nahavandi, and D. Srinivasan, "Spinalnet: Deep neural network with gradual input," *arXiv preprint arXiv:2007.03347*, 2020. 90

[223] E. M. Izhikevich, “Solving the distal reward problem through linkage of stdp and dopamine signaling,” *Cerebral cortex*, vol. 17, no. 10, pp. 2443–2452, 2007. 94

[224] R. Legenstein, D. Pecevski, and W. Maass, “A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback,” *PLoS Computational Biology*, vol. 4, p. 1000180, oct 2008. 94

[225] S. T. Barnard and M. A. Fischler, “Computational stereo,” *ACM Comput. Surv.*, vol. 14, p. 553–572, Dec. 1982. 121

[226] Y. Frauel, T. J. Naughton, O. Matoba, E. Tajahuerce, and B. Javidi, “Three-dimensional imaging and processing using computational holographic imaging,” *Proceedings of the IEEE*, vol. 94, pp. 636–653, March 2006. 121

[227] B. Sun, M. P. Edgar, R. Bowman, L. E. Vittert, S. Welsh, A. Bowman, and M. Padgett, “3d computational imaging with single-pixel detectors,” *Science*, vol. 340, no. 6134, pp. 844–847, 2013. 121

[228] M.-J. Sun, M. P. Edgar, G. M. Gibson, B. Sun, N. Radwell, R. Lamb, and M. J. Padgett, “Single-pixel three-dimensional imaging with time-based depth resolution,” *Nature communications*, vol. 7, no. 1, pp. 1–6, 2016. 121

[229] A. Turpin, G. Musarra, V. Kapitany, F. Tonolini, A. Lyons, I. Starshynov, F. Villa, E. Conca, F. Fioranelli, R. Murray-Smith, and D. Faccio, “Spatial images from temporal data,” *Optica, Vol. 7, Issue 8, pp. 900-905*, vol. 7, pp. 900–905, dec 2020. 121, 123, 124, 125, 126, 127, 131

[230] R. Garg, B. G. Vijay Kumar, G. Carneiro, and I. Reid, “Unsupervised CNN for single view depth estimation: Geometry to the rescue,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9912 LNCS, pp. 740–756, Springer Verlag, 2016. 122

[231] C. Zheng, T. J. Cham, and J. Cai, “T2 Net: Synthetic-to-realistic translation for solving single-image depth estimation tasks,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11211 LNCS, pp. 798–814, Springer Verlag, sep 2018. 122

[232] S. Cova, M. Ghioni, A. Lacaita, C. Samori, and F. Zappa, “Avalanche photodiodes and quenching circuits for single-photon detection,” *Applied optics*, vol. 35, no. 12, pp. 1956–1976, 1996. 123

[233] D. O’Connor, *Time-correlated single photon counting*. Academic Press, 2012. 123

[234] P. Caramazza, A. Boccolini, D. Buschek, M. Hullin, C. F. Higham, R. Henderson, R. Murray-Smith, and D. Faccio, “Neural network identification of people hidden from view with a single-pixel, single-photon detector,” *Scientific reports*, vol. 8, no. 1, pp. 1–6, 2018. 123, 125, 126

[235] K. Park, S. Kim, and K. Sohn, “High-precision depth estimation with the 3d lidar and stereo fusion,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2156–2163, IEEE, 2018. 124

[236] Y. Xu, X. Zhu, J. Shi, G. Zhang, H. Bao, and H. Li, “Depth completion from sparse lidar data with depth-normal constraints,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2811–2820, 2019. 124

[237] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017. 124

[238] Y. Zhou and O. Tuzel, “Voxelnet: End-to-end learning for point cloud based 3d object detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4490–4499, 2018. 124

[239] Y. Li, L. Ma, Z. Zhong, F. Liu, M. A. Chapman, D. Cao, and J. Li, “Deep learning for lidar point clouds in autonomous driving: a review,” *IEEE Transactions on Neural Networks and Learning Systems*, 2020. 124

[240] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, “Deep learning for 3d point clouds: A survey,” *IEEE transactions on pattern analysis and machine intelligence*, 2020. 124

[241] A. Aßmann, B. Stewart, and A. M. Wallace, “Deep learning for lidar waveforms with multiple returns,” in *28th European Signal Processing Conference*, 2020. 124

[242] T. Shinohara, H. Xiu, and M. Matsuoka, “Fwnetae: Spatial representation learning for full waveform data using deep learning,” in *2019 IEEE International Symposium on Multimedia (ISM)*, pp. 259–2597, IEEE, 2019. 124

[243] S. Zhou and W. Wang, “Object detection based on lidar temporal pulses using spiking neural networks,” *arXiv preprint arXiv:1810.12436*, 2018. 125

[244] W. Wang, S. Zhou, J. Li, X. Li, J. Yuan, and Z. Jin, “Temporal pulses driven spiking neural network for fast object recognition in autonomous driving,” *arXiv preprint arXiv:2001.09220*, 2020. 125

[245] Y. Kang, L. Li, D. Liu, D. Li, T. Zhang, and W. Zhao, “Fast long-range photon counting depth imaging with sparse single-photon data,” *IEEE Photonics Journal*, vol. 10, jun 2018. 130

[246] A. Halimi, Y. Altmann, A. McCarthy, X. Ren, R. Tobin, G. S. Buller, and S. McLaughlin, “Restoration of intensity and depth images constructed using sparse single-photon data,” in *European Signal Processing Conference*, vol. 2016-November, pp. 86–90, European Signal Processing Conference, EUSIPCO, nov 2016. 130