Robust Object Classification for Autonomous Ship Navigation through Spiking Neural Networks and Meta Learning

ShippingLab project

Joaquín Royo Miquel (s192929)

Master's of Science in Engineering

2020

DTU Electrical Engineering
Department of Electrical Engineering

Robust Object Classification for Autonomous Ship Navigation through Spiking Neural Networks and Meta Learning, ShippingLab project

Report written by:
Joaquín Royo Miquel (s192929)

Advisor(s):
Roberto Galeazzi, Associate Professor @ DTU Electrical Engineering
Silvia Tolu, Associate Professor @ DTU Electrical Engineering
Frederik E. T. Schöller, PhD student @ DTU Electrical Engineering

DTU Electrical Engineering
Technical University of Denmark
2800 Kgs. Lyngby
Denmark

elektro@elektro.dtu.dk

Project period:	31 August 2020- 28 February 2021
ECTS:	35
Education:	M.Sc.Eng.
Field:	Electrical Engineering
Class:	Public
Edition:	1. edition
Remarks:	This report is submitted as partial fulfillment of the requirements for graduation in the above education at the Technical University of Denmark.
Copyrights:	©Joaquín Royo Miquel, 2020

Acronyms

AP	Average Precision
ANN	Analog (conventional) Neural Networks
CNN	Convolutional Neural Networks
DCC	DTU Computing Centre
DVS	Driving Event Camera
FS	Few-Shot
HPC	High Power Computing
IoT	Internet of Things
IoU	Intersection over Union
IF	Integrate and Fire
LIF	Leaky-Integrate and Fire
mAP	mean Average Precision
ML	Meta-Learning
ODC	Object Detection and Classification
SL	ShippingLab
SNN	Spiking Neural Networks

Abstract

Robust detection and classification of objects at sea is a major step towards safe autonomous navigation and collision avoidance for marine vessels. Objects of interest (OOI) include large commercial vessels, small leisure boats, kayaks, other objects without a clear radar signature, as well as stationary objects such as buoys, land, bridge pillars and similar. Object classification using electro-optical sensors can address this problem and has been shown to be a potential alternative to a human lookout.

This project emanates from the research performed in [75] and investigates alternative, avant-garde approaches to the implementation of the RetinaNet [47], deep-learning object detector, by means of neuromorphic computation and few-shot meta-learning. This is done with the aim of reducing the energy footprint and data consumed by the algorithm. Although these technologies are in early stages of development, the project manages to achieve state-of-the-art results when compared to other contemporary research.

The TFA [94] few-shot learning method was applied, leading to a reduction of 90% in the training time and 95% in the data consumption while achieving a 63.3% of the target performance on RetinaNet. In parallel, an algorithm was developed to convert the RetinaNet model into a spiking neural network following the work performed in [72] and [36]. This model maintained the characteristics of the original, showing a loss in performance of just 1.6% for a simulated time-window of $1000ms$.

Code: `https://github.com/joaromi/MSc_Thesis_DTU`

Contents

Acronyms **i**

Abstract **iii**

Contents **v**

1 Introduction **1**
1.1 Background and motivation 1
1.2 Objectives 2
1.3 Resources 3
1.4 Scope of the project 3
1.5 Thesis outline 6

2 Literature review **7**
2.1 ShippingLab project 7
2.2 Artificial Neural Networks 8
2.3 Object detection and classification 13
2.4 Meta-Learning 16
2.5 Summary 19

3 Spiking-RetinaNet **21**
3.1 Spiking Neural Networks overview 21
3.2 Spike-train encoding 23
3.3 First approach: Training 24
3.4 Final approach: Conversion 27
3.5 Whetstone 51
3.6 Summary 57

4 TFA-RetinaNet **59**
4.1 Meta-Learning overview 59
4.2 Applications to object detection and classification 60
4.3 TFA Method 63
4.4 Implementation 64
4.5 Results 67
4.6 Summary 68

5 Discussion **69**
5.1 Overall results analysis . . . 69
5.2 Fulfilment of the objectives . . . 72
5.3 Future work . . . 74
5.4 Summary . . . 74

6 Conclusion **77**

Appendices **79**

A Spiking-RetinaNet predictions **81**
A.1 Predictions on the ShippingLab data . . . 81
A.2 Predictions on MS COCO . . . 82

B Correlations **83**

C ShippingLab validation set size issue **87**

Bibliography **89**

CHAPTER 1

Introduction

1.1 Background and motivation

During the last decades, there has been an evergrowing trend towards the development of autonomous machinery that can substitute human workforce in tasks of increasing complexity. Following these events and with the rise of artificial intelligence, the future of sea transportation is likely to follow the same path and, thus, efforts are being made to propel the needed technical developments. To allow for safe unmanned navigation, obstacle detection and classification plays a crucial role and can potentially be addressed using electro-optical sensors and machine-learning techniques.

This project stems from the ShippingLab's autonomous navigation research initiative, as a follow-up to the work performed in [75]. In this previous paper, research was carried out on existing convolutional neural network architectures for computer vision and the performances of three state-of-the-art networks were assessed for object detection at sea. The benchmarked models were Faster R-CNN [25], YOLOv3 [67] and RetinaNet [47].

After the positive outcomes delivered on [75], the current document investigates alternative, somewhat unfamiliar approaches to the implementation of the RetinaNet deep learning model, exploiting more biologically accurate neural network architectures and meta-learning. These will be exploited towards the *Green AI* paradigm, aiming to reduce the resource usage of deep-learning. The advantages of mimicking the biological brain, unmatched in power efficiency, and few-shot learning to reduce the data consumption, advertise themselves as good research paths towards this objective. However, as shown by previous applications, these technologies are still in early phases of development and their relevance in computer vision follows more modest neural network structures and classification tasks (e.g. the popular MNIST handwritten digit database).

The object detection and classification problem is at a much higher level in what refers to the required network complexity, the data and computational cost, and the precision demanded. It is regarded as significantly more challenging as it involves both recognizing multiple overlapped objects and calculating precise coordinates for bounding boxes [36].

An up-scaling and adaptation of the aforementioned technologies will be fulfilled and the obtained results will be benchmarked against the original RetinaNet to finally evaluate their potential interest towards the ShippingLab objectives.

1.2 Objectives

The objectives stated in this section align with the roadmap of the ShippingLab project and were agreed between the author and supervisors of the MSc Thesis. These milestones are decisive to evaluate the project's outcome and their completion will be assessed in the Discussion (section 5).

1.2.1 Main objectives

The following objectives are expected to be met and documented in the thesis by the end of the project:

1. Perform a literature study of specific topics relevant to the thesis work. Emphasis should be given to:
    a) Deep learning and spiking neural networks (SNNs): advantages and limitations w.r.t. other types of learning systems/architectures.
    b) Meta-learning methods for vision-based detection and classification in the presence of small data sets.
    c) Visual-based detection and classification for autonomous ship navigation
2. Investigate possible approaches to spiking architectures and neuromorphic computing with the focus on deep-learning models for computer vision.
3. Develop and implement RetinaNet for detection and classification of ships and buoys exploiting an SNN architecture, and compare performances w.r.t. an deep (analog) neural network architecture.
4. Investigate meta learning methods for object detection and classification (on spiking neural networks) and identify one or more approaches that could be relevant for the autonomous ship navigation.
5. Develop and implement at least one meta learning method for object detection and classification.
6. Test and validate the implemented meta learning solution on relevant data sets and compare findings with the SNN-RetinaNet approach.
7. Draw conclusions about the potential of using SNN and meta learning for robust object detection and classification in autonomous ship navigation.

1.2.2 Side objectives

The objectives stated below are accessory and may be fulfilled if the available resources permit it:

1. Further development of existing spiking neural network frameworks targeting the object detection and classification problem.
2. Investigate potential improvements to the studied technologies towards the objectives of the ShippingLab project.
3. Implementation and testing of the developed solutions in dedicated hardware.

1.3 Resources

The following resources were identified at the beginning of the MSc project and will be available to carry out the project objectives:

- Labeled databases of images (MS COCO [48] and ShippingLab datasets).
- Unlabeled databases of images.
- Deep learning model for comparative study (RetinaNet model from [89] and [75]).
- NVIDIA Jetson TX1 or TX2 for implementation and testing of developed solution.
- The DTU Computing Center (DCC) [19] for the training of the deep-learning models.

1.4 Scope of the project

The workload of the project corresponds to that of a 35 ECTS MSc Thesis with a duration of six months. The tasks are grouped into two main work packages which are aligned with the objectives stated above.

The research revolves around the development and testing of novel techniques to address the issue of object detection and classification using AI. This problem can be defined as the identification of instances of semantic objects of a certain class (e.g: people, animals, vehicles of any sort, etc.) in digital images or video, and pertains to the areas of image processing and computer vision.

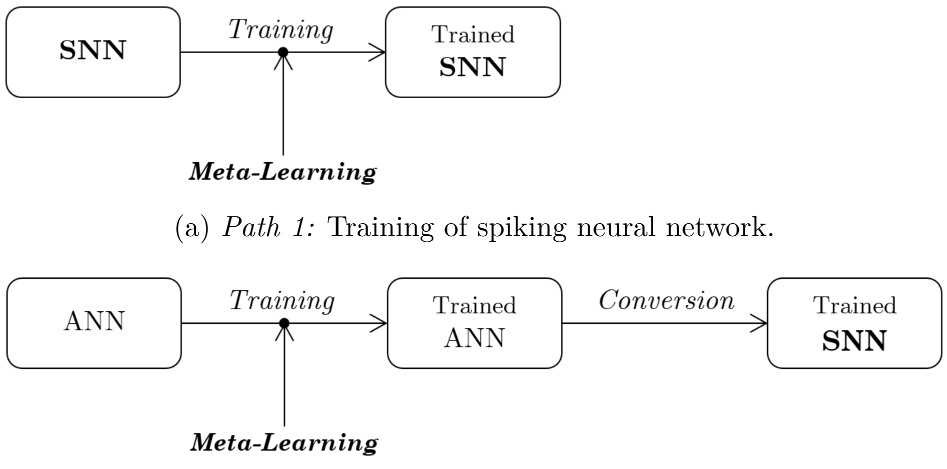
The first work package looks into the field of spiking neural networks and neuromorphic computation, relating to objectives 1a, 2 and 3. It has ended up constituting around 80% of the workload due to the necessity of developing a suitable framework for the realization of spiking object detection models. The developed Python library allows for conversion of convolutional object detection networks into spiking models and their simulation and testing.

The second work package is related to the analysis and application of meta-learning techniques on the RetinaNet network's training, covering objectives 1b, 4, 5 and 6.

The time available for this part has been very limited and accomplishment of the base objectives was prioritized over further upgrades. A few-shot learning method has been applied to perform a fast training of the RetinaNet model and the results have been compared against the fully trained RetinaNet.

It was initially contemplated to perform an implementation of the algorithms in dedicated hardware (Nvidia Jetson TX2). However, research on the available frameworks for SNNs revealed that none of them posessed the level of complexity required for this project. Resources were then reallocated to further develop these frameworks and the implementation on dedicated hardware was traded as a side objective.

Results obtained in both work packages could ideally fit together as described in figure 1.1, leading to improvements in the AI model training and its implementation in a spiking architecture.



(a) *Path 1:* Training of spiking neural network.

(b) *Path 2:* Training of deep (analog) neural network and porting to a spiking architecture.

Figure 1.1: Scheme to illustrate the studied approaches. Path 2 describes the approach that the final implementation produced in the project could potentially achieve.

1.4.1 Contributions

The research performed pertains to the fields of neuromorphic software architectures and few-shot learning.

Regarding the meta-learning work package, the project relies on previous work in few-shot object detection and classification, adapting an existing algorithm (TFA [94]) to the characteristics of the ShippingLab data and the RetinaNet model. The contributions made are essentially the implementation, analysis and benchmarking of the TFA-RetinaNet against relevant previous studies and against the fully-trained RetinaNet model.

In the spiking neural networks section, however, after a broad scanning for possible paths to implement the RetinaNet algorithm in a neuromorphic architecture, a method is developed to transform from the trained RetinaNet model to a spiking neural network. Although following a similar idea as shown in [36], the method used in that paper was not fully explained and the code was not accessible. The starting point was instead the research performed in [72], which had to be remodelled and upgraded to suit the object detection and classification (ODC) problem. Therefore, this section contributes with a framework for the ANN to SNN conversion and simulation of ODC convolutional neural networks and with the analysis of the results obtained using the RetinaNet model as input.

It is worth highlighting the novelty of the work realized in this section as, to the best of our knowledge, there is only one previous research paper that reports a similar level of success on the implementation of an ODC spiking neural network (*Spiking-YOLO* [36]).

1.4.2 Limitations

The main constraint for the project is the highly demanding nature of the environmental awareness issue towards ship autonomy. Safety codes dictate that the provided detections must be accurate and robust in any weather conditions. The recall (fraction of detected obstacles over the whole number of them) is an especially critical metric, as the consequence of failing to detect an object in a navigation setting could be adverse. The evaluation of the results must therefore be coherent with the laid expectations.

It is known beforehand that these newly explored technologies are in their early stages and will most likely not fulfill these hard requirements in their current state. The project aims instead to lay the ground for future investigations that may improve them and take advantage of the benefits they offer.

This project is noticeably ambitious in scope too, attempting to dig into multiple lines of research and very novel technologies. Time limitations have been the main dictators for the reach of the results accomplished in each work package. As it can be perceived in its description, this project has very strong internal dependencies, which made a successful result only achievable if all the steps in the project chain were performed correctly. Resources and schedules have been restructured from the way they were initially conceived to ensure a maximal fulfillment of the agreed objectives.

Besides, during the project's kick-off meeting, it was agreed that the RetinaNet[47] ODC algorithm would be one to be used throughout the project, to follow previous work done by the department in [75].

Finally, concerning the application of spiking neural networks to the ODC task, the biggest constraints have been the absence of a neuromorphic platform in the project resources and, as has been mentioned above, the shortage of time to perform this hardware implementation if available. The translation of a neural architecture this big to that kind of device would definitely have been a challenge too. This has impeded a study on the main advantages of neuromorphic deep-learning models and so, the first work-module's results will be limited to determining whether a spiking implementation of RetinaNet is a viable milestone.

1.5 Thesis outline

The document is tailored to be self-contained, independently address all the project objectives and have a user-friendly structure. It is divided into 6 different chapters:

Chapter 1 - Introduction. Provides context to the reader and briefly introduces the main concepts which will be relevant throughout the Thesis. Sets the expectations for the content of the following chapters.

Chapter 2 - Literature review. Familiarizes the reader with the state-of-the-art and properly references previous work in the studied fields. Provides some background knowledge that may be necessary to understand the following chapters.

Chapter 3 - Spiking-RetinaNet. Explains in detail the characteristics of spiking neural networks and neuromorphic computing and explores possible paths to perform the implementation of RetinaNet on this sort of architecture. A toolbox to convert it to a spiking model is produced and the results are analyzed on the ShippingLab data. It corresponds to work package 1.

Chapter 4 - TFA-RetinaNet. Reviews possible approaches to the enhancement of the ODC problem based on existing meta-learning studies. The TFA few-shot learning method is then chosen, setting the path for further analysis, implementation and testing on the ShippingLab data. It corresponds to work package 2.

Chapter 5 - Discussion. The final results of the Thesis are analyzed and benchmarked against previous work. The section elaborates on the findings of the project and balances the advantages and limitations of each of the studied paths. It also evaluates the accomplishment of the project objectives and suggests possible courses for future work in these fields. It is closely related to objective 7.

Chapter 6 - Conclusion. Provides a summary of the whole project and highlights its positive impact.

CHAPTER 2

Literature review

In this chapter, some context will be provided to the current project by performing a revision of the state-of-the-art within the fields of study. It serves the double purpose of properly referencing the minds behind preceding research and providing the reader with the needed, prior knowledge to master the work in following chapters.

2.1 ShippingLab project

Denmark is among the world's leading maritime nations and, overall, maritime companies account for approximately one-fourth of the country's total exports [3].

The Blue Denmark is one of Denmark's industrial positions of strength, consisting of shipowners and shipping companies and a wide number of businesses whose activities emanate from international and Danish shipping. These could be, for example, shipbrokers, ports, logistics companies, shipyards and service companies that supply equipment, components and service to ships.

About 60,000 persons are directly employed and about 42,000 persons are indirectly employed in Blue Denmark. In total, this corresponds to 3.8% of Danish employment.

It is the Danish Government's stated ambition to maintain and expand the Blue Denmark's position as a growth engine in the Danish economy. ShippingLab implements this ambition, having stakeholders across the maritime industry joining efforts on pre-competitive initiatives [78].

2.1.1 Project objectives

The project goal in the long term is to enhance the competitive edge of Danish maritime suppliers to secure Blue Denmark's leading position in the global market. This will be accomplished by investing in in ground-breaking research, development and innovation for the creation of quantifiable technological value in the short term. Three core areas are the target of current investigation: digital ship operations, decarbonization and autonomy, pertaining this Thesis to the third work package in the list.

The overall objective of ShippingLab Autonomy is to be able to demonstrate autonomy up to unmanned operations of a Danish vessel while integrating with existing technology. An open, modular system architecture and interface standard are to be defined aiming for an easier implementation in the Danish infrastructures.

The approach is twofold. The first case is an add-on autonomy functionality for existing vessels while the second case focuses on one-man and unmanned operations of new-built electric ferries.

2.1.2 Data characterization

The maritime dataset used in this project comes from previous research [75].It is conceived as a pool of images from different sea scenarios for the training and testing of AI models towards the autonomous navigation of ships. Objects of interest (OOI) include large commercial vessels, small leisure boats, kayaks, other objects without a clear radar signature, as well as stationary objects such as buoys, land, bridge pillars and similar. The images were acquired from ferries in near coastal service and objects at a wide range of distances are annotated in two main classes: *boats* and *buoys*. The data is divided into two separate training and validation subsets.

The chosen classes serve the basic purpose of locating and distinguishing between stationary obstacles (*buoys*) and moving vehicles (*boats*) within the income data, as key information for the ship to define its future course.

The dataset is limited in scope, as the images pertain to, at most, 3 different ferry routes and the climatic conditions are not adverse. Nevertheless, it suffices to make a first assessment on the performances of different vision algorithms and compare between them. It will be referred throughout the project as the ShippingLab dataset.

2.2 Artificial Neural Networks

Artificial neural networks are a category of computing systems within the family of machine-learning algorithms and AI. These systems are inspired to some degree by the biological neural networks that constitute animal brains.

2.2.1 Biological inspiration

When it comes to non-mathematical tasks, the animal brain possesses many advantages over traditional computing, especially in what regards to pattern recognition and improvisation. The brain is also remarkably robust, as its activity is decentralized in a massive mesh of connected units. These units, called neurons, have very limited data processing capabilities by themselves. However, their functional association in interconnected networks greatly increases their potential. Communications (synapses) between neurons take place through weighted electrochemical pulses which can be both positive (excitatory) and negative (inhibitory) [39].

Perhaps, the most fascinating aspect about the brain is that it is able to learn by establishing relationships between actions and stimuli. This key feature defines most animal forms of life, whose survival is in many cases related to their learning capabilities and awareness of their surroundings. According to the leading theory among neuroscientists, learning is performed through what they call *synaptic plasticity*, which is the modulation of the sensitivity of brain neurons due to repeated and persistent stimulation. This is referred to as *cell assembly theory*, *Hebb's postulate*, or *Hebbian learning* [27].

Artificial neural networks were designed with the objective of providing human technology with this kind of perception, becoming one of the pillars of artificial intelligence. Their main resemblance with the animal nervous system is their micro-structure, based on a collection of connected units called artificial neurons, and their ability to learn from error and stimuli.

2.2.2 Analog Neural Networks

The first research paper about the design of artificial neural networks was published in 1943 by McCulloch and Pitts [53]. A neuron was modeled as switch which, depending on the total weighted inputs from other neurons, produced a binary output. The first learning demonstration, though, was performed in 1958 with Rosenblatt's *Perceptron* [68], which could be taught linear problems. These networks have quickly developed into very complex models, which are trained on massive amounts of data for applications as diverse as weather forecasting, stock-market tracking, historical reconstruction and control of unmanned vehicles.

Due to the hardware constraints, neurons in these artificial networks are very simplified, consisting in a non-linear activation function that produces a numeric output based on a linear combination of all the inputs. For this reason and following the nomenclature established in [72], these networks will be referred throughout the project as analog neural networks (ANNs).

In the field of computer vision traditional hand-crafted algorithms have quickly been substituted by deep learning techniques, due to their higher accuracy on benchmark datasets. Convolutional neural networks nowadays constitute the unrivaled top-performers between image analysis methods. A more elaborate explanation of this topic shall be found in section 2.3.

2.2.3 Spiking Neural Networks

Artificial spiking neural networks (SNNs) are deep learning models which, when compared with ANNs, show a more biologically realistic behavior, incorporating a time dimension. Their main difference is that SNNs inner communications are driven by sparse and asynchronous binary signals, product of the neuronal dynamics. This is similar to how biological neurons behave, transmitting spiking signals which are generated by electrochemical reactions. In the context of artificial networks, these are replicated within a software simulation or using electronics in the so-called neuromorphic processing units. Some examples of these hardware platforms are the University of Manchester's *SpiNNaker* [24], IBM's *TrueNorth* [12] and Stanford's *Neurogrid* [5].

Spiking neural networks have mainly been approached from a neuroscientific perspective, being propelled by the need of a better understanding of the mammalian brain's remarkable information processing skills. With ongoing efforts toward these bio-inspired models, there are expectations that they will come closer to natural intelligence and reach higher computational efficiency than more abstract models. In particular it is shown that networks of spiking neurons can be, with regard to the number of neurons needed, computationally more powerful than ANN models. In one of the most influential papers on the topic [51], published in 1997, a concrete biologically relevant function is exhibited which can be computed by a single spiking neuron, but which requires hundreds of hidden units on a sigmoidal neural network. On the other hand, it is known that any function that can be computed by a small sigmoidal neural net can also be computed by a small network of spiking neurons.

With the success of deep-learning, the most common perspective is to view SNNs exclusively as a more efficient replacement of conventional ANNs. This is reflected in the way SNNs are benchmarked by their accuracy on ANN tailored datasets like MNIST or CIFAR, which lack the precise timing and low latency information that could benefit a neuromorphic approach. Such comparisons are certainly important because they show that SNNs can be powerful classifiers in the classical machine learning setup. However, they create a bias that hampers the proper development of the technology.

Many practical applications of SNNs can be found in the areas of robotics and control [16, 8], decision making and action selection, trajectory planning and tracking, rehabilitation, environment exploration, etc [50]. These while showing a big potential for the SNNs, are rather simplified approaches in the most part, set as probes for the technology's early development. Brain-machine interfaces (BMI) are also currently arising as a particularly interesting application field for neuromorphic computing [9] due to the low energy consumption, low heat dissipation, robustness, and ability to decode in real time, showcased by spiking models. The fact that SNNs can process biological spikes without further transformation adds to the appeal of such systems. The recent development of these spiking networks also appear as a promising opportunity for the *Green Artificial Intelligence* paradigm, which aims to reduce the resource usage while training and executing AI models [92].

An increasing trend is being observed in the interest in event-based computer vision showcased by research communities. Advantages of using event-based sensors have been demonstrated for several applications such as tracking, stereo vision, optical flow estimation, scene reconstruction and SLAM [50]. However, only few of these approaches use spiking models for event-based post-processing or run on neuromorphic hardware. As a pioneer in this field, it is necessary to mention the gesture recognition system in [2], developed to highlight the benefits of combining a dynamic vision sensor with IBM's

True North processing chip.

It is believed that there is great potential for fully event-based sensing and processing systems, where deep SNNs on neuromorphic hardware platforms seem like an obvious choice [63]. However, although initial demonstrations performed on simpler classification tasks are encouraging [84], more research is needed to develop spiking models that can benefit from event-based input to outperform traditional deep-learning methods. To stimulate research in this direction, the neuromorphic engineering community has been working on generating new benchmarks that do not carry the legacy of conventional machine learning [57, 7]. Such datasets have only recently became available but already had a favourable effect on providing visibility to the progress in SNN computing.

Finally, the object detection and classification problem, which motivates the current research, is a rather unexplored area due to the sheer size of the networks and datasets required, making it a bad test-bench for new SNN prototypes. Seoul National University's *Spiking-YOLO* neural network [36] is the pioneer in this front line, achieving state-of-the-art performances by conversion of a pre-trained ANN *Tiny-YOLO* model while consuming 280 times less energy on a neuromorphic chip.

Available Python libraries which have been tested in this project for SNN software simulation include *Norse* [61], *SNN Toolbox* [81], *BindsNET* [26] and *Brian2* [83].

2.2.4 Conversion from ANN to SNN

Despite their excellent potential, SNNs have been limited to relatively simple tasks, shallow structures, and small datasets, bue primarily the lack of scalability of their learning algorithms. This is caused by the complex dynamics and non-differentiable activations of spiking neurons. ANN-to-SNN conversion algorithms have been widely studied in recent years as a parallel approach. These methods rely on the idea of translating pre-trained parameters from an ANN, creating an equivalent SNN with minimal sacrifice in performance.

Early studies on this topic initiated in 2013 with the research performed in [62]. In this paper, convolutional units were translated into LIF-spiking neurons with refractory periods, aiming for compatibility with event-based sensors. A close link between the dynamics of a spiking neuron and a ReLU activation function was suggested in [10], reporting good performance error rates after implementation. This method was improved in [17], achieving nearly loss-less conversion on the MNIST dataset by using a parameter normalization step before the transformation. Other approaches include noise injection for more robust training and predictions, the use of binary weights and restricted connectivity targeting the *TrueNorth* hardware [20], and a conversion method that adapts the firing threshold of spiking units to minimize the number of spikes required [99].

Directly serving as foundation for this project, [72] builds upon previous works and implements support for many operators that are crucial for improved ANN error scores, such as pooling layers, softmax activations and batch-normalization. The underlying encoding scheme for this conversion approach is rate-based: The neurons generate a sequence of discrete spikes, which, when averaged over the integration time, it approximates the analog firing rate corresponding to the original ANN. This encoding becomes more accurate as the simulation duration is increased and more spikes are generated, entailing a higher resolution at the price of scaling up the computational cost.

A future research by the same authors, documented in [70], implements an inverse time-to-first-spike encoding, where each neuron fires a single spike and the information is encoded in its temporal location. This approach sacrifices some of the robustness and accuracy of the rate encoding in favour of a lower energy cost of information transfer.

Finally, [36] implements in neuromorphic hardware the first object detection model using a deep rate-SNN, achieving comparable results to the original ANN in non-trivial datasets (2% performance reduction). This is achieved by the use of a new, fine-grained normalization method that increases the information transmission, and by the design of a new spiking version of leaky-ReLU being able to transmit negative values using an imbalanced threshold.

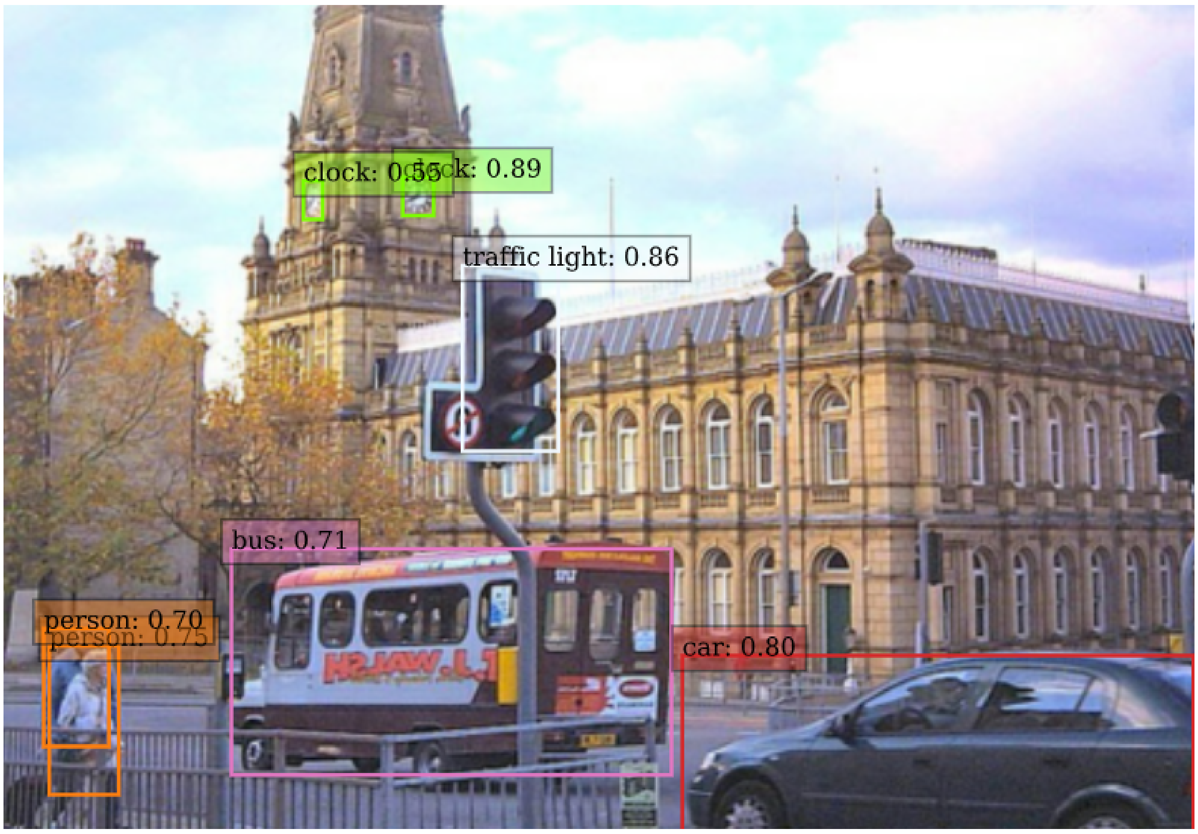


Figure 2.1: Object detections performed by the (Keras) RetinaNet model used in this project. Image belongs to the MS COCO dataset [48].

2.3 Object detection and classification

The object detection and classification problem constitutes one of the most advanced issues related to computer vision and image processing. It deals with detecting and distinguishing instances of semantic objects of a certain class in digital images or video frames. Some well-researched domains in the ODC field include pedestrian detection and facial recognition, mostly for surveillance, target tracking, and security applications [87].

Object detection methods are benchmarked within the scientific community in relevant datasets which contain a very diverse pool of labelled classes. These environments are somewhat more challenging that many real applications due to such a varied object range and the sheer size of the image sets. The most relevant data collections are:

- *COCO* [48], a large-scale dataset of images with Common Objects in Context (COCO) for object detection, segmentation, and captioning. These objects are labelled in 80 different classes. It is sponsored by Microsoft, Facebook, CVDF and Mighty AI.
- *Pascal VOC* [21] (Visual Object Classes), comprised of 20 semantic classes. The preparation and running of this challenge is supported by the EU-funded PASCAL2 Network of Excellence on Pattern Analysis, Statistical Modelling and Computational Learning.
- *ImageNet* [14], backed by Stanford and Princeton universities, is an image dataset organized according to the WordNet hierarchy. Each meaningful concept in WordNet, possibly described by multiple words or word phrases, is called a ”synonym set” or ”synset”. There are more than 100,000 synsets in WordNet with an average 1000 images to illustrate each synset.

The performance of object detectors is measured using the mean Average Precision (mAP), a popular metric used to evaluate models doing document/information retrieval and object detection tasks. This metric is computed by averaging across all object classes the AP scores obtained in each of them. The AP (Average Precision) is calculated by integrating the precision[1] of the detections of each class as a function of their recall[2].

Traditional machine-learning approaches to this problem rely on a prior definition of features, which are used for a later classification using techniques such as a support vector machine (SVM). A feature is a specific area of the image which contains unique information that makes it easily distinguishable for computer-vision algorithms. Usually, features correspond to specific structures in the picture that may show a big difference in lighting and color between adjacent pixels, for example, corners and edges of objects

[1] $Precision = \frac{\text{True good detections}}{\text{All detections}}$; a.k.a. positive predicted value.

[2] $Recall = \frac{\text{True good detections}}{\text{All labeled objects in the image}}$; a.k.a. true positive rate or sensitivity.

[29]. Early iterations of these feature detectors, such as the Harris Corner Detector [15], measured the rate of change of pixel values when shifting a window through the image in different directions. More advanced techniques are available, such as SIFT and SURF, two robust and widely used algorithms which are under patent protection [35]. As a faster and license-free alternative to these, ORB was developed in 2011 by OpenCV engineers to reach comparable performances, building on the lighter, FAST corner detector and BRIEF descriptor extractor [69].

The above-mentioned methods have since then been outrun by deep-learning techniques, able to do end-to-end object detection without specifically defining features, and typically based on convolutional neural networks (CNN). Something interesting about these structures is that the connectivity pattern between their neurons resembles the animal visual cortex. Up to this date, the best performing model to accomplish the ODC task is the *YOLOv4-P7*, achieving a score of 60% on the COCO test-dev [59].

Regarding direct previous work to this document, in [75], they benchmarked three deep-learning techniques on the ShippingLab dataset. A summary of this benchmarking is displayed in table 2.1. The RetinaNet architecture, which is a key component of this project, is among these tested models and will be further discussed in the following section (2.3.1). This model will be used as bedrock for the techniques applied in the present research.

Table 2.1: Comparison of deep-learning algorithms on the ShippingLab (SL) dataset performed on [75]. Score corresponds to mAP@0.5IoU.

Method	Source	SL Score	COCO Score	Description
Faster R-CNN	Ross B. Girshick [25]	81%	46.7%	Classical two-stage detector which generally yields the best results, but is quite slow (2015).
YOLOv3	Redmon, J. & Farhadi, A. [67]	90%	46.6%	A Very fast one-stage detector, which does sacrifice some performance to achieve these fast inference times (2018).
RetinaNet	Lin *et al.* [47]	90%	52%	One-stage detector that uses a novel loss function to improve its performance, the *Focal Loss* (2018).

2.3.1 RetinaNet

RetinaNet [47] is a state-of-the-art one-stage object detector which achieves performances comparable to classical two-stage approaches (e.g. Faster R-CNN) while being computationally cheaper. The key to these results is the novel loss function it incorporates, named *Focal Loss*. It is a single, unified convolutional neural network consisting of a backbone and two task-specific subnetworks in charge of the bounding box regression and object classification (figure 2.2).

The backbone is an off-the-shelf convolutional neural network which is responsable for highlighting potential areas of interest within the input image by computing a convolutional feature map. Following the path taken in [75], the ResNet50 CNN will be used for this task. The feature map is aranged using a Feature Pyramid Network (FPN) [46] with the objective of improving detections over a wider range of scales. The *top* of the pyramid refers to the final layers of the CNN, containing spatially coarser but semantically stronger features. The *bottom* instead, is linked to shallower layers in the CNN, containing higher resolution but lower-semantic feature maps capable of more precise keyspot location. In essence, the FPN's function is to augment the backbone with a top-down pathway that up-samples features from above in the pyramid to generate higher resolution versions in lower stages. These are then enhanced via lateral connections with features from the bottom-up pathway creating rich semantics in all levels. The pyramid is structured in 5 stages of 256 channels each.

RetinaNet makes use of a predetermined set of sizes and aspect ratios that serve as translation-invariant bounding box candidates for the areas of interest. These are referred to as anchor boxes and each one is assigned to a one-hot vector of classification targets and a size 4 vector for the predicted box coordinates. An anchor is assigned to a ground-truth (labelled) object if their IoU is above a threshold h_o, or to the background class if below h_b ($h_b < h_o$). Anchors with an IoU between both thresholds are ignored during training. The parameters chosen were $h_o = 0.5$ and $h_b = 0.4$.

The classification head is in charge of estimating the probability of object presence for each of the C classes in each of the K archors. It consists of a sequential, fully

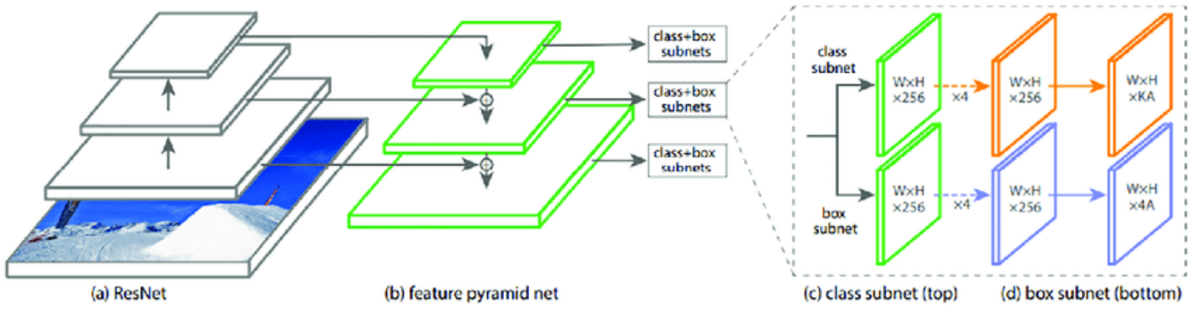


Figure 2.2: Architecture of RetinaNet as described in the source paper [47].

convolutional network whose inputs are the feature maps of each of the FPN levels. It is structured as four iterations of 3x3 convolutional layers with ReLU activations followed by a 3x3 convolutional layer with a [K,C] sized output. This output can be decoded using sigmoid activations as in the standard classification networks.

The box regression head shares the overall structure with the classification head except it uses different parameters and terminates in a [K,4] sized output. It serves as a class-agnostic estimator of the offset between each anchor box in the feature map to the nearest found object.

The Focal Loss, the novel loss function introduced in the RetinaNet classification task, has been designed with the purpose of better addressing the imbalance regarding the background class, much more abundant than the object classes. It also allows for better classification of distant objects in areas with high object population or large differences between object sizes.

$$FL\left(p_t\right) = -\alpha_t\left(1-p_t\right)^{\gamma}\log\left(p_t\right) \tag{2.1}$$

This loss, shown in equation 2.1, builds upon the traditional cross entropy formula adding a modulating factor: $\alpha_t\left(1-p_t\right)^{\gamma}$. Here, γ serves as a focusing parameter to down-weight the easy, over-represented examples, focusing on hard negatives, and α as a balancing factor.

In this project, RetinaNet achieves a mean Average Precision at 0.5 IoU of 50.45% in the MS COCO dataset and 97.16% in the ShippingLab data.

2.4 Meta-Learning

From a social psychology perspective, the term meta-learning is a branch of meta-cognition referred to the understanding of the interaction between the mechanisms of learning and the concrete contexts in which they are applicable. Hence, meta-learning is viewed as an adaptation of learning itself on a higher level than merely acquiring subject knowledge [91]. The term meta-learning was first conceived by Donald Maudsley to describe a process by which people become self-aware and ‘increasingly in control of habits of perception, inquiry, learning, and growth that they have internalized’ [52]. The prefix *meta-* in this context implies an abstract recursion: the word meta-learning can then be defined as ‘learning about learning’.

In the field of AI, this term addresses the study of meta-data from learning algorithms pursuing improvements in their efficiency and performance. In many cases, this refers to the application of other automatic learning algorithms to this data. However, meta-learning might also refer to a manual approach to the study of a model’s hyperparameters or learning context in the seek to improve or automate the algorithm tuning [43].

The term meta-learning is first addressed in literature in 1987 within two independent pieces of work, by J.Schmidhuber and G.Hinton. Schmidhuber's PhD Thesis [74] built the theoretical framework for a new family of methods using *self-referential learning*, which involve the training of neural networks that can receive as inputs their own learnable weights and predict updates for said weights. The document further proposed that the model itself could be induced using evolutionary algorithms. The paper by Hinton *et al.* [28], on the other hand, suggests to duplicate the weights in each neuron synapse. The first weight would be the standard *slow-weight*, which prompts a slow acquisition of knowledge, whereas the second weight, the *fast-weight*, produces a fast-paced training aiming to recover slow weights learnt in the past that have been since forgotten due to the optimizer updates (*deblur*).

After both of these papers gave rise to the concept of meta-learning, one can see a rapid branching-out and increase in the usage of the idea in multiple different areas. Some of the most relevant milestones include the meta-learning of biologically plausible learning rules in 1995 [4], the first proposals for meta-learning frameworks using gradient descent and backpropagation in 2001 [98], the first use of meta-learning in combination with reinforcement learning in 2003 [76], and the first attempt of zero-shot learning in 2008 [42].

In the modern deep-learning context, meta-learning can be classified into four main categories [56]:

1. **Optimizer meta-learning**, which search for the optimal hyperparameter configuration in order to maximize the performance of the base neural network.

2. **Few-shot meta-learning**, where a deep neural network is engineered so that it is able to generalize from a base training to new unseen datasets from the same task family, in an attempt to move away from traditional, energy and data-hungry

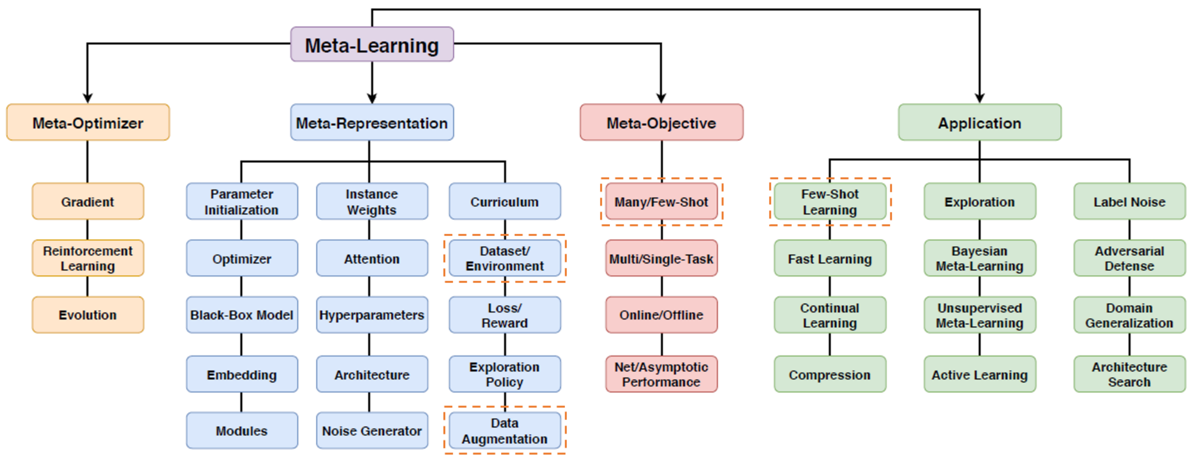


Figure 2.3: Meta-learning taxonomy as described in [30]. Fields relevant to the current paper are framed in orange.

methods. The basic concept is that individual training samples are minimalistic, varied, and provide enough abstraction so that the network can learn to identify new objects just like a child would learn, using few but significant samples.

3. **Metric meta-learning** is the use of an AI model to determine if a metric is effective for a set of problems and if the target networks are hitting it properly. The metric space is learnt by the use of few examples and can be used across diverse domains.

4. **Recurrent model meta-learning** basically consists in the application of meta-learning techniques to recurrent neural networks and long-short term memory networks.

Regarding the ambit of computer vision, which encapsulates this project, the main body of work has been generated in the field of few-shot learning, with a major focus on the few-shot classification problem [6]. This is defined as the issue of identifying to which of a set of categories (sub-populations) a new observation belongs using scarce training data.

Many effective solutions have been proposed for this problem in the past few years, obtaining 1-shot performances as good as 98% on the *Omniglot* benchmark [41] and 82% on the more challenging *miniImageNet* set [85]. A brief overview of these solutions will be provided now. The majority of these algorithms can be labelled as either a gradient-based meta-learner or as a metric learning algorithm.

In the gradient-based meta-model setting, a distintion is made between the meta-learner, which is the model that learns across episodes, and the base-learner, which learns from the data within each episode. The meta-learner's parameters are trained at the end of each episode from the classification loss. Some examples of this approach are *Meta-LSTM* (2016) [66], which uses a long-short-term-memory and a custom update rule to improve the learning on the few-shot set; and *maML* (2017) [23], which learns how to best initiate the base-model parameters to maximize learning in each training episode. This meta-model is conceived to be agnostic from the base model and could be applied to any machine-learning problem.

Metric-learning, on the other hand, involves learning to compare data samples by means of a distance function. In classification problems, they classify query instances depending on their similarity to support set instances. This setting has given rise to approaches such as matching networks [93], which are trained to compute a representation of images enabling them to be classified by comparison *in-situ*; and relation networks [86], which evolve over the previous ones by allowing the algorithm to also learn the distance function.

Finally, the object detection and classification (ODC) problem, which is our direct area of interest, is a very novel field of study, having barely a few years of relevance within the meta-learning paradigm. For instance, to the best of our knowledge, the first few-shot object detector was proposed in early 2019 by Kang *et al.* [33]. The current performance of 10-shot object detectors in the MS COCO dataset is capped at a mAP@0.5 of 12.5% [58], being this performance very poor for the objectives of the ShippingLab project. This field of study will be further characterized in section 4.

2.5 Summary

Object detection and classification methods are an essential technology for the fabrication of unmanned, fully autonomous vehicles as they enable machines to achieve advanced perception of the environment through the extraction of detailed information from image or video inputs. These objectives align with the ShippingLab Autonomy research effort, which aims to develop the first autonomous Danish ship.

The best performing object detectors are deep-learning methods with an artificial analog neural network structure (ANN). These structures loosely mimic the animal brain by emulating its mesh structure formed by thousands of simple, interconnected units. Among them we find RetinaNet [47], a one-stage ODC which will be used for the experiments in this project.

Spiking neural networks (SNNs), on the other hand, are considered the 3rd generation of artificial neural networks, as they draw even more inspiration from the animal brain trying to emulate the dynamics from its neurons and its learning rules. These algorithms are still in early phases of development but show a big potential in computational power and energy saving.

Finally, meta-learning strives to equip deep-learing models with a deeper understanding of their learning process, achieving improvements in their training time, the data consumption and the performances obtained.

CHAPTER 3
Spiking-RetinaNet

This chapter corresponds to the first work package of the MSc Thesis project. The aim of this module is to explore the possibility and benefits of implementing the RetinaNet object detection algorithm using spiking neural networks. This is not an easy task, as vision applications of spiking neural networks are currently limited to classification tasks on smaller and simpler network architectures.

The chapter will start with a brief overview of the characteristics of spiking neural networks and how information can be conveyed using binary spikes. Possible paths to perform the implementation of RetinaNet on this sort of architecture will be explored, leading to a final discussion and benchmarking between each of the followed studylines.

3.1 Spiking Neural Networks overview

As presented previously in section 2.2.3, artificial spiking neural networks (SNNs) are deep learning models that more closely mimic the constitution and dynamics of the nervous system. SNNs first emerged in computational neuroscience, as an attempt to model the behavior of biological neurons. This resulted in the *Leaky-Integrate-and-Fire* (LIF) model, which is the most prominent spiking neuron mathematical implementation.

The LIF-neurons activity is described as an integration of the received spike voltages, paired to a weak dissipation (leakage) to the environment. This leakage term produces a slow decay of the neuron voltage if not enough consecutive stimuli are received. When this voltage reaches a threshold, the neuron fires a spike on its own (figure 3.1). Following further inspiration from the brain, neuronal inputs and outputs are encoded in spike-trains. In the current project, though, the leakage term of the model has been neglected for simplicity and the neurons are shaped using the *Integrate-and-Fire* (IF) model. The mathematical explanation for it can be found in section 3.4.1.

Even though the first models for these kind of networks range to 1952 [1], spiking neural networks have not met a large interest within the deep-learning community until quite recently due to their lack of efficient training algorithms for supervised learning [65]. Commonly used learning algorithms require continuous valued, differentiable functions for the neuron's output, being spiking activations not suitable for them. Several workarounds have been performed by calculating the time-of-arrival of the spikes or the spiking-rate, nevertheless, this is inefficient and adds complexity to the problem.

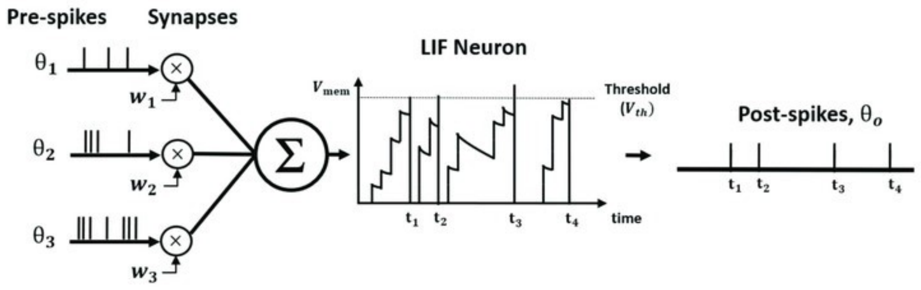


Figure 3.1: Graphical explanation of the *Leaky-Integrate-and-Fire* (LIF) neuron model. From left to right, input spike-trains (Θ) are modulated by the synaptic weights (w) to be integrated as the current influx in the membrane potential (V_{mem}). The leakage term causes an exponential decay of this potential over time. Whenever this voltage crosses a firing threshold (V_{th}), the neuron fires a spike and resets the membrane potential.

Spiking neurons are, on the other hand, widely used in neuromorphic computing. This consists in building neural networks directly in hardware, using electronic and optic systems. Physical device neurons are connected by electronic synapses in a distributed, energy efficient architecture. This approach has been used, for example, within the *Human Brain Project* [31], in an attempt to recreate a functional *Silicon Brain*. With ongoing efforts toward these bio-inspired models, there are expectations that they will come closer to natural intelligence and reach higher computational efficiency than more traditional approaches.

SNNs on neuromorphic hardware exhibit favorable attributes such as fast inference and low-power consumption due to event-driven information processing [63]. Their architecture also avoids energy voracious data shuffling between the memory and the processing units. Being a standard computer's estimated power usage around $100W$, neuromorphic architectures aim to reduce it to comparable levels to the human brain, which has an estimated average energy consumption of roughly $20W$ [18].

In this project, though, the spiking neural networks will be operated via software simulation. This sacrifices many of the mentioned advantages in favour of a more accessible approach and the ability to analyze the relationship between the SNNs the more widespread ANNs. This choice is also motivated by the possibility of using more widespread, well-known frameworks (PyTorch, Keras/TensorFlow) for the programming of the models, and by the hardware resources available, which do not bear a neuromorphic architecture.

Neuromorphic platforms currently face the challenge of scaling up and increasing their range of applicability. They will have to deal with more complex neural networks and device variability, while keeping the devices compact and low energy consuming [90]. Having said that, a previous research has already implemented an object detection spiking model (*Spiking-YOLO*) in neuromorphic hardware achieving state-of-the-art results [36]. This project strives to emulate this success following a different path. Due to the pre-existing complexity of the object detection and classification problem and the scarcity of previous research, the project adheres when possible to the most reliable and well-documented approaches to deep SNNs, to maximize its chances.

Aside from the potential energy saving, the interest of spiking neural networks lies on a two more reasons. Firstly, the robustness of rate-encoded spike-trains, which allow for partial loss of the communicated signals and weaker transmissions. And secondly, the possibility of using bio-inspired learning rules, such as Hebbian learning, while also exploiting the network dynamics thanks to their temporal encoding.

3.2 Spike-train encoding

The first step towards the implementation of a spiking neural network is to determine how the information is encoded in and transmitted across neurons. In the literature there are two methods that stand out: *temporal-mean-rate* encoding and *time-to-first-spike* encoding.

The most direct connection between analog and spiking neural networks is made by considering the activation of an analog neuron as the equivalent of the firing rate of a spiking neuron assuming a steady state [63]. This is addressed as *temporal-mean-rate* [72], rate encoding [38] or r-SNN. The information is encoded in the average number of spikes that occur during the simulation. This way of transmission is very robust, being inherently redundant because it integrates multiple binary events over a determined period of time.

In contrast, *time-to-first-spike* (ttfs) [70] or t-SNN encodings carry the relevant information within the delay between the spikes and the stimulus [40, 60]. This approach is more energy efficient as it produces sparse or single spike trains. However, precise detection of every spike can be crucial and its mathematical relationship with analog neural networks is less intuitive.

Figure 3.2 shows a simplified example of a signal's encoding in a $100ms$ time window using both r-SNN and t-SNN with the purpose of comparing both approaches. The rate coding approach can transmit signals in the [0, r_{max}] range, being 0 the absence of any event and r_{max} the emision of a spike at every single timestep (maximum rate). The ttfs interprets this interval differently, coding its maximum value as an immediate spike and the minimum as a whole time window delay before the spike (no spike).

The advantage that *time-to-first-spike* has over rate encoding is manifest in terms of energy consumption, as it uses just a single spike to transmit the information. Nevertheless, they both show an improvement over the analog signal's energy footprint.

It is worth noting that the spike-train encodings suffer from one major limitation: the signal's discrete sampling. For a fixed clock rate, the available resolution of the conveyed data is capped by the length of the time interval where it is encoded. High

[1]The encoding of the spike trains has been simplified for clarity.

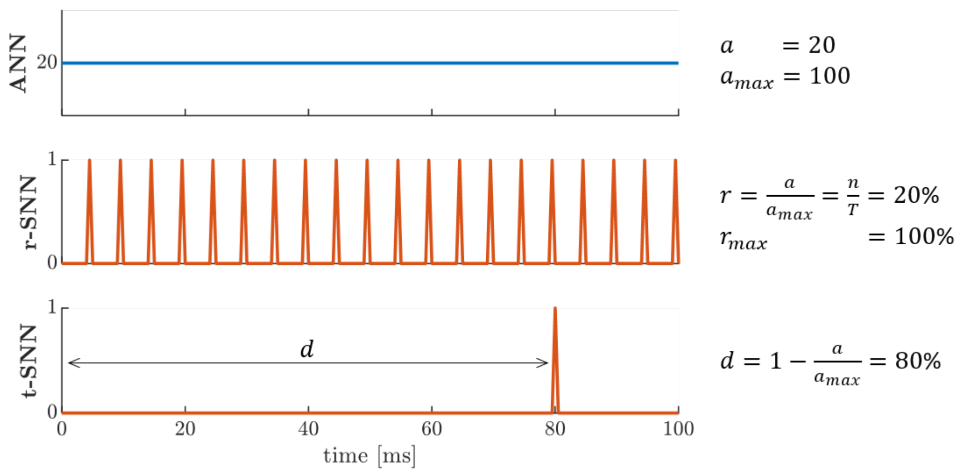


Figure 3.2: Explanatory[1]graph for the two studied spike-encoding approaches. Note how the constant signal transmitted in the ANN is likely to be more energy consuming than the event-based binary signals of the SNNs.

precision tasks like object detection and classification (ODC) are likely to need a large time window to achieve good performances.

The SNN implementation of RetinaNet will run on *temporal-mean-rate* encoding due to its more straightforward connection to ANN, mathematical simplicity and prominence among existing research. Furthermore, it also exhibits a higher robustness to errors, as a failure to detect some of the spikes does not induce a significant error in the average spike-rate.

3.3 First approach: Training

This section studies the learning capabilities of an artificial spiking neural network and explores the possibility of applying a training process to a SNN version of RetinaNet.

3.3.1 Learning methods for SNNs

In virtually all artificial neural networks, spiking or non-spiking, the learning process is realized through the adjustment of scalar-valued synaptic weights [88]. Nevertheless, SNNs are compatible with more bio-plausible Hebbian learning rules that are not possible to replicate in analog neural networks (see section 2.2.3). These are referred to under the umbrella term of *spike-timing-dependant plasticity* (STDP), their common key feature being that the strength (weight) of the connection between two consecutive neurons is adjusted according to their relative spike times within a certain interval. The adjustment is therefore performed locally, both to the synapse and to the time window.

The most common instance of biological STDP can be intuitively interpreted as follows: Given the event of a neuron in layer l firing, if a presynaptic neuron in layer $l-1$ had fired shortly before, then their connection would be strengthened due to the likelihood of a causal link. However, if the presynaptic neuron fired instead briefly after, then the causal relationship between both events would be spurious and the weight of their connection, weakened. This algorithm is more straightforwardly related to unsupervised learning, focus of the vast majority of SNN experimentation due to their distinctive potential. Nevertheless, STDP is also embedded in some SNN supervised learning techniques which could be suitable for this project.

In the context of spiking neural networks, supervised learning aims to minimize the error between desired and output spike trains, adjusting the model's weights via gradient descent on a cost function. Although this concept is similar to the main ANN training techniques, the main disparity lays on the non-differentiable nature of the spikes.

The simplest and most traditional approach to supervised learning is called surrogate gradient backpropagation [96, 55]. The method is a direct translation of ANN's successful gradient descent algorithm, applied to SNNs by using a differentiable approximation for the spiking activations. Following these in popularity come STDP-based backpropagation methods, which infer that the equivalent of the gradient descent rule is best expressed as a compound of two Hebbian processes in SNNs (figure 3.3). These techniques, referred first as *ReSuMe* in [64] and then further developed in [82], establish an explicit link between biological and artificial learning patterns. An additional mention should be made to temporal backpropagation [54, 49], which adapts the backpropagation method to delay-encoded SNNs.

Further research performed on these methods shows that all of them achieve good

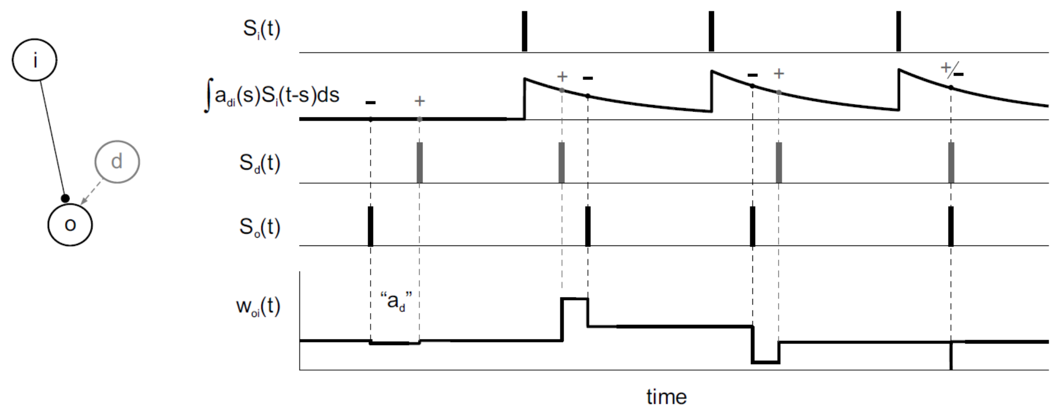


Figure 3.3: Graphical explanation of Hebbian (STDP) learning within the *ReSuMe* training algorithm [64]. The weight modifications Δw_{oi} are computed using the difference between the desired (S_d) and the obtained (S_o) output spike trains and their relative location from the presynaptic spikes (S_i) modulated by a decaying exponential. This is a theoretical spiking equivalent to the ANN backpropagation algorithm.

performances on the MNIST handwritten digit database but could be very inefficient when scaled up to more complex problems if not ran in a neuromorphic platform [88]. Nevertheless, the surrogate gradient backpropagation algorithm was selected for further testing as it is the widest-spread approach among popular deep-learning libraries.

3.3.2 Implementation and results

Before trying to apply the chosen learning method to the objectives of this project it is worth noting that all mentioned techniques have only been tested on classification tasks. It is important then, to assess the scalability of the surrogate gradient descent method for its use in the training of a bigger object detection neural network.

To evaluate the learning capabilities of a spiking neural network, a test training of a three layer convolutional model was performed on the MNIST handwritten digit database using the PyTorch Norse SNN library [61]. An ANN with the same structure was used for benchmarking.

Two key metrics have been chosen for the comparison: the training time and the accuracies obtained after three epochs, as this number was sufficient to achieve good-enough performance in the ANN model.

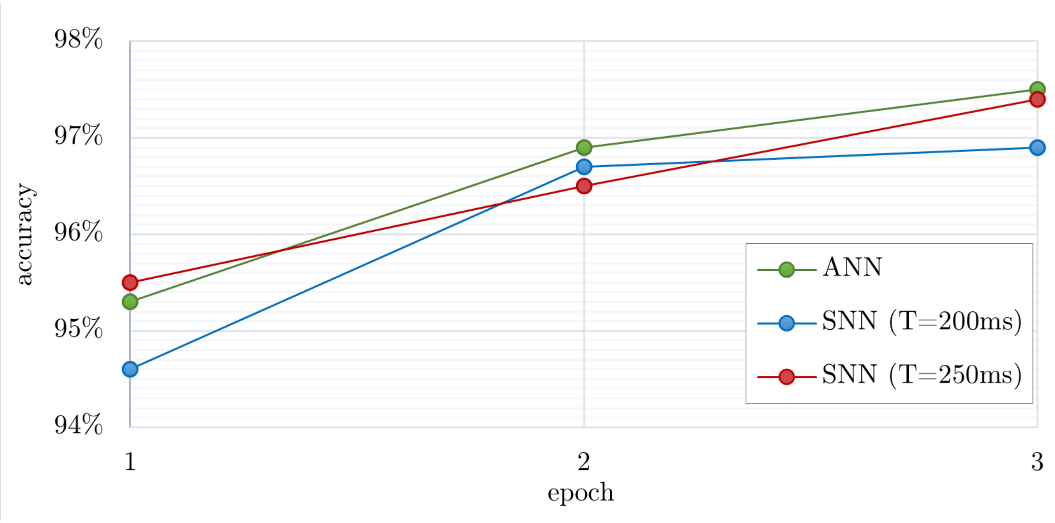


Figure 3.4: Evolution of the accuracies of the models trained on the MNIST dataset.

The training was performed using gradient backpropagation in the case of the ANN and surrogate gradient backpropagation for the SNN. As displayed in table 3.1, a time window of 250ms is needed for the spiking model to achieve similar performance to the original ANN.

A look at the training time, however, reveals the big downside this entails. The computational cost of the SNN training makes it no rival to the ANN. The source of this issue is likely to be a combination of the following:

Table 3.1: Benchmarking between ANN and SNN training on MNIST dataset.

	ANN	SNN $t = 200ms$	SNN $t = 250ms$
Training method	backprop.	surr. backprop.	surr. backprop.
Training time	7,3 s/epoch	200 s/epoch	478 s/epoch
Final accuracy	97.5%	96.9%	97.4%

- The PyTorch framework is not designed for the use of SNNs, their implementation is done by simulation. This is magnified in the training, as the spiking network is run through the desired time window for every sample in the dataset.
- The gradient descent method trains the network by backpropagation of the error through the derivatives of the activation functions. Due to the non differentiability of the spikes in the model, approximate gradients are needed. These, in combination with the more complex dynamics of the neurons, could have a negative impact on the training time.
- The training method used is a direct translation of an algorithm which was designed for ANN. However, this might not be the right approach due to the different nature of the spiking model. This is now an open line of research whose most interesting results shift towards Hebbian learning based algorithms.

Due to the larger scale of the RetinaNet network (100+ layers) and the already long time it takes to train its ANN version (∼6 days), the decision was made to discard this approach and steer the project towards the conversion of a trained model to a spiking neural network architecture.

3.4 Final approach: Conversion

The aim of this section is to implement and test a method that translates from a trained ANN to an equivalent SNN with minimal sacrifices in its performance. This approach should be much less time consuming than training the SNN and the converted model could potentially be deployed to specific harware for efficient execution.

The chapter draws its main inspiration from the research carried out in [72] and [70], where deep-learning classification models are automatically converted to spiking neural networks. Nonetheless, it moves one step ahead by applying it to the much more complex object detection problem in a comparable way as the work done in [36]. For more details on previous works regarding this topic see section 2.2.4.

3.4.1 Method

The method used for the ANN to r-SNN conversion is based on the principle that the firing rate of the spiking neurons over a certain time window should approximate the graded activations of the original analog neurons [72]. A one-to-one correspondence between analog and spiking neurons is assumed.

The ReLU activations for neuron i in layer l of the ANN (a_i^l) can be expressed as:

$$a_i^l := \max\left(0, \sum_{j=1}^{M^{l-1}} W_{ij}^l a_j^{l-1} + b_i^l\right) \tag{3.1}$$

where W_{ij}^l is the kernel from neuron j of layer *l-1* to neuron i of layer l and b_i^l is the bias on neuron i of layer l (weights of the analog neuron).

On the other hand, the dynamics of the neurons in the SNN are modelled using the *Integrate and Fire (IF)* membrane equations. Each SNN neuron has a membrane potential $V_i^l(t)$, which is driven by the input current $z_i^l(t)$ and the event of the generation of a spike $\Theta_i^l(t)$.

$$\begin{aligned} V_i^l(t) &= V_i^l(t-1) + \underbrace{z_i^l(t)}_{\text{input}} - \underbrace{V_{th}\Theta_i^l(t-1)}_{\text{reset}} \\ z_i^l(t) &= V_{th}\left(\sum_{j=1}^{M^{l-1}} W_{ij}^l \Theta_j^{l-1}(t) + b_i^l\right) \\ \Theta_i^l(t) &= \begin{cases} 1 & \text{if } V_i^l(t) \geq V_{th} \quad \longleftarrow \textit{spike} \\ 0 & \text{else.} \end{cases} \end{aligned} \tag{3.2}$$

The parameter V_{th} is the membrane voltage threshold that, if surpassed, triggers the event of a spike in the neuron. The membrane voltage then drops by that amount (*reset by subtraction*).

The input current $z_i^l(t)$ is obtained as a linear combination of all the inputs to the neuron, however unlike in the ANN case, this time the inputs are binary spikes $\Theta_j^{l-1}(t)$ received from the previous layer.

The *IF* neuron firing rate is assumed to range from the non-spiking state of the neuron (0%) to its spiking at every single time step (100%). For this reason, a requisite for a one-to-one correspondence of these rates to the original analog activations is that they all fall in the unit interval too. This can be accomplished through a normalization of the ANN parameters.

Assuming the former is fulfilled, the SNN can be built as an equivalent copy of the ANN's structure and weights.

3.4.2 Approximation errors

After the conversion, the SNN is affected by two different sources of approximation error: firing saturation and discrete sampling.

Firing saturation takes place when the original analog activation values fall outside the unit range. This results in $z_i^l > V_{th}$ and the best a spiking neuron can do in this case is fire with its maximal rate, but never reach the target frequency.

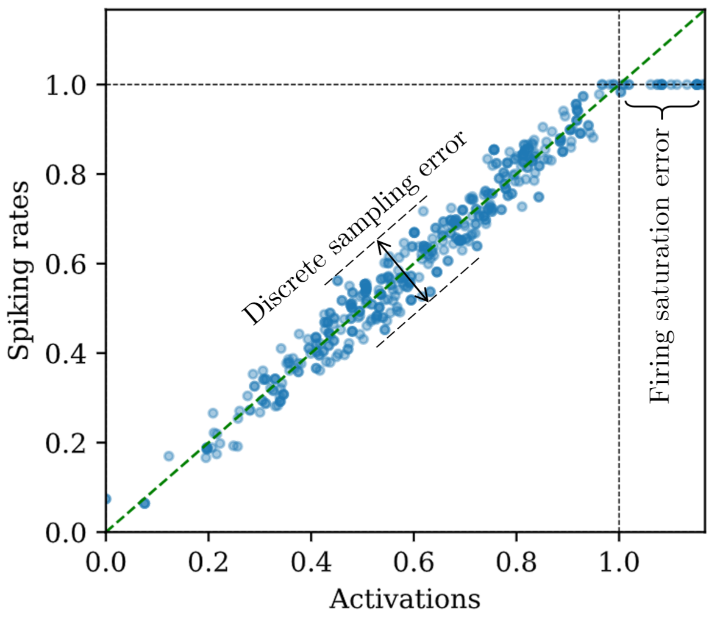


Figure 3.5: Approximation errors highlighted in a correlation plot between the original activations and the obtained spiking rates of a sample layer which has been converted from ANN to SNN

Regarding **discrete sampling**; with the *reset by substraction* mechanism the membrane potential of the first layer of a neuron can be expressed as: $V_i^1(t) = t \cdot z_i^1 - n_i^1(t) \cdot V_{th}$, considering z_i^1 as a constant input current. The number of spikes that have been fired is then:

$$n_i^1(t) = \frac{t \cdot z_i^1 - V_i^1(t)}{V_{th}} \tag{3.3}$$

And the spiking rate can be approximated as:

$$r_i^1(t) = \frac{n_i^1(t)}{t} = \frac{z_i^1}{V_{th}} - \frac{V_i^1(t)}{V_{th} \cdot t} = a_i^1 - \underbrace{\frac{V_i^1(t)}{V_{th} \cdot t}}_{\text{error}} \tag{3.4}$$

The firing rate of the neuron converges to the original analog activation with an approximation error due to discrete sampling. This was briefly referred to in section 3.2 in a more qualitative way. In deep SNNs this error is propagated through the network to deeper layers as explained in [73] yielding the following expression.

$$r_i^l(t) = \sum_{j=1}^{M^{l-1}} W_{ij}^l r_j^{l-1}(t) + b_i^l - \frac{V_i^l(t)}{V_{th} \cdot t} \tag{3.5}$$

This shows that the approximation errors of previous layers are recursively multiplied by the kernels of higher layers and accumulated with their own discrete sampling errors.

Thus, a neuron will receive an input spike train with a rate reduced according to the sampling error of previous layers, resulting in deteriorated firing rates in higher layers.

As seen in eqs. 3.4 and 3.5, these discrete sampling errors are inversely proportional to the simulation time, so the deeper the network, the longer the simulation time needed to achieve high correlations with the ANN activations.

These two error sources can be addressed in the normalization phase prior to conversion. However, there is a trade off to be made between them. Priorizing the complete removal of the neurons' saturation means fitting all activations in the $[0-1]$ interval, which may produce very low firing rates in a significant percentage of the neurons. This will increase the simulation time needed to reduce the discrete sampling error. Some workarounds to these issues are applied in section 3.4.4.2.

3.4.3 Tools

The implementation of the conversion method explained in this section has been based on the *Spiking Neural Network Conversion Toolbox* (`SNN_Toolbox`) available at [71]. This library is grounded on the research paper [72], and will need to be augmented to target object detection models.

This toolbox automates the conversion of pre-trained analog to spiking neural networks (ANN to SNN), and provides options for testing the SNNs in spiking neuron simulators or neuromorphic hardware. The algorithm operates within the Keras (Tensorflow) framework.

Most of the findings reported in this section were achieved using reverse engineering, as the documentation is scarce and concise. This is because the toolbox is designed as a black box with which you interact through a configuration file. The user can navigate and choose between the offered options by means of this file as explained in the available documentation [81].

3.4.3.1 Pipeline

The toolbox currently supports input networks generated with Keras, PyTorch, Lasagne, or Caffe. However, as the toolbox's backend is Keras-based, the Keras models offer the best compatibility; hence being the ones used in this project.

The `SNN_Toolbox`'s algorithm follows the scheme displayed in figure 3.6. The pipeline takes in an ANN in one of the aforementioned frameworks and produces a functional equivalent spiking neural network. It is divided in four very distinct stages which are detailed below.

Parsing. For the toolbox to be flexible to accept input models from several frameworks and structures, the toolbox has a first translation step that extracts the relevant

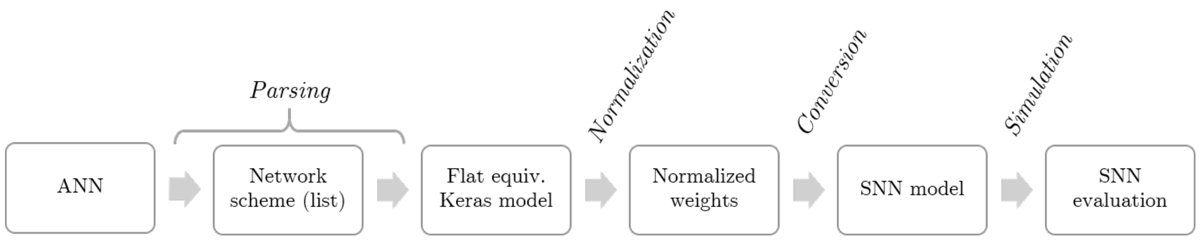


Figure 3.6: Pipeline scheme for the `SNN_Toolbox`

information from the input neural network and creates an equivalent Keras model. This allows for some abstraction and is used as base for the spiking model generation.

The parsing step is also in charge of handling different types of layers for posterior porting to the spiking model. For example:

- The parameters of the batch normalization layers are absorbed into the parameters of the previous convolutional layer.
- Activation layers are absorbed into the previous convolutional layer.
- Max pooling layers can optionally be replaced by average pooling layers that show better compatibility with the spiking dynamics.
- Add layers are replaced by a combination of a concatenate and convolution layers. This is done to allow for the subsequent normalization of branched networks.
- For the same reasons, a global average pooling layer is replaced by a combination of an average pooling layer and a flatten layer.

Normalization. As mentioned in section 3.4.1, the SNN will be built with a one-to-one transformation of the neurons from the parsed ANN to *Integrate-and-Fire* spiking neurons, seeking that the firing rate of these converges to the analog activations of the original ANN neurons.

To make this functional, the activations of the ANN must be kept within the 0—1 range, as the I-F spiking neurons can only spike with a rate between 0 and 100%.

With this objective, a normalization is performed to the parsed ANN before its conversion to SNN. The parameters of the network are modified to scale all activations to the desired range while preserving their logic. In the `SNN_Toolbox`, this is executed layer-wise, with respect to the largest activation of each layer. For that reason, it will be referred to as layer normalization. Further explanation can be found in section 3.4.4.2.

Conversion. After the weight normalization, the parsed model is ready for conversion. The model is mirrored using equivalent spiking counterparts for all its neurons.

The `SNN_Toolbox` is capable of generating spiking neural network outputs in the INI/Keras (default), pyNN, Brian2, MegaSim, Loihi and SpiNNaker frameworks. In the INI/Keras backend, which is the one chosen for this project, these spiking layers have all been built as children of the `keras.layer` class, including new attributes and methods to compute the dynamic behavior of the spiking neurons.

The toolbox can generate spiking neural networks which use temporal-mean-rate coding (*r-SNN*), time-to-first-spike coding (*t-SNN*) and temporal pattern coding.

Table 3.2 shows the layers which have a spiking-compatible version currently implemented in the toolbox. Note that not all of them rely on a spiking *I-F* dynamic model.

Table 3.2: Layers which have been implemented in the `SNN_Toolbox`

Spiking I-F layers	*Spiking-compatible layers*[2]
• `Dense`	• `Reshape`
• `Conv1D`	• `Flatten`
• `Conv2D`	• `Concatenate`
• `DepthwiseConv2D`	• `ZeroPadding2D`
• `Sparse`	
• `SparseConv2D`	
• `SparseDepthwiseConv2D`	
• `AveragePooling2D`	
• `MaxPooling2D`[3]	

Simulation. The produced spiking neural network can either be deployed on third party frameworks or simulated in the built-in INI simulator (Keras).

The simulator accepts inputs in the shape of either Poisson spike trains or constant input currents, to which our analog input can be transformed. Inputs from DVS event sequences are also allowed.

The network is then simulated for a given time window. At each timestep, the input is propagated through spikes modifying the inner states of the neurons. The spikes that take place in the final layer are accumulated throughout the duration of the experiment.

With an appropriate time window, the spiking rate (accumulated spikes over a certain duration) will converge to the analog activations of the original ANN.

[2]Spiking-compatible layers use the standard Keras implementation with small tweaks to make them work with the spiking simulator.

[3]`MaxPooling2D` layer is not yet fully compatible with the Tensorflow backend of Keras.

3.4.3.2 Toolbox limitations

In spite of its smart layout and great flexibility, the `SNN_Toolbox` was conceived as a first experimental approach to ANN-SNN conversion, targeting much simpler network architectures than the one being contemplated in this project.

The toolbox had to be redesigned and have some of its core components completely rebuilt to adapt it to the RetinaNet neural network.

The main limitations found were:

1. Due to its size and modularity, RetinaNet is built as three interconnected subnetworks, each with its own task (see section 2.3.1). The `SNN_Toolbox` lacks compatibility with this architecture as it expects a neural network which is framed in a single layer iterable (‘flat’ list of layers). This affects mainly the *Parsing* step in the pipeline.

2. RetinaNet is an object detection and classification neural network, whereas the *Conversion* and *Simulation* processes in the toolbox have been conceived for more simple classification problems.

3. The list of layers that the toolbox can work with is limited. RetinaNet has upsampling layers that fall out from this set.

4. The *Normalization* method used in the toolbox is not optimal for ODC problems and would yield to very low firing rates in many network channels.

5. Other issue with the *Normalization* is that it was designed for only-positive activations (e.g. ReLU). RetinaNet uses also linear activations and all information transmitted in the negative subspace is susceptible of being lost.

6. The toolbox does not allow for the use of custom loss functions or optimization algorithms.

3.4.4 Implementation

Due to time and scope constraints, the modifications performed to the toolbox have focused only on the Keras/Tensorflow framework. This did sacrifice the compatibility of the algorithm with other deep learning libraries but made possible to scale the method and apply it to a Keras RetinaNet implementation.

3.4.4.1 Parsing

As mentioned in 3.4.3.2 (lim. 1), the *Parsing* stage in the `SNN_Toolbox` needed to receive modifications to prepare it for input models which are structured in multiple levels or subnetworks.

Table 3.3: Python version and modules used for the implementation.

Python	3.8.6
NumPy	1.18.5
TensorFlow	2.3.1
Keras	2.4.3

The modus operandi at this stage is to iterate through each of the subnetworks, extracting its structure and relevant information, and then create a 'flat' scheme of the complete network that makes the data immediately accessible. This scheme is performed using a Python dictionary with the layer names as *keys* and the layer data as *values*. The dictionary is then fed to `.build_parsed_model` and used to generate an equivalent Keras model of the network that can be used downstream in the pipeline.

To assemble the dictionary, a new method was created within the `SNN_Toolbox`'s `ModelParser` class taking as foundation the original `.parse` from the toolbox. This new function, `.parse_subnet`, is explained briefly in table 3.4.

A very useful feature of the parsing method is that it allows for modifications of the layer data before appending it to the dictionary. This largely improves compatibility of the pipeline with a wider range of layer types. An example of this is the substitution of the Add layers for custom layers which are prepared for the following *Normalization* phase (3.4.4.2).

To further increase this versatility, the `.build_parsed_model` was revised to include customization for the model's loss function and optimizer, as this characteristic was a requirement for RetinaNet's parsing.

3.4.4.2 Normalization

The *Normalization* step receives the parsed model and modifies the weights of its layers so that every activation falls in the $0-1$ interval. This makes the network suitable for conversion to SNN with minimal information loss due to firing saturation (3.4.2).

The original layer normalization method used in the `SNN_Toolbox` had two issues: it led to low firing rates and was not designed for negative activation values. In [36], the first issue is addressed by performing the normalization channel-wise instead of layer-wise. This approach magnifies channels with small activations and greatly improves the firing rates of the network (see figure 3.9).

This channel normalization, however, keeps being incompatible with negative activations. In this section, a novel normalization method will be introduced which builds on top of the latter, fixing this issue. The method will be referred to as *shifted* channel normalization.

Table 3.4: Overview of the `.parse_subnet` method

`ModelParser.`**`parse_subnet()`**
Inputs:
• `self`. The class `ModelParser`, that contains the dictionary which is being written and the rest of the tools to generate the parsed model.
• `layers`. The layer iterable of the subnetwork that is being targetted.
• `idx`. Dictionary index of the last layer that was parsed. For naming and reference purposes.
• `prev_out_idx`. A list of the dictionary indices of the outbound layers from the previous subnetwork. These layers will be connected to the inbound layers from the current subnetwork.
• `in_layers`. A list of the inbound layers of the targetted subnetwork. Needs to be of the same length as `prev_out_idx` and their elements need to be correctly arranged so that they are correctly coupled for the connection.
• `out_layers`. A list of the outbound layers of the targetted subnetwork. It will be used to generate `out_idx`.
Outputs:
• `idx`. New dictionary index of the last layer that was parsed.
• `out_idx`. A list of the dictionary indices of the outbound layers from the current subnetwork. These layers will be connected to the inbound layers from the following subnetwork.
Implicit outputs:
• `self._layer_list`. Dictionary (filled in by this method) which sorts the relevant data and relationships for all the layers in the model. New layers are appended as new entries of the dictionary as the method iterates through the targetted subnetwork.

Shifted channel normalization uses an estimate of the distribution of all possible activation values for each channel in the network to perform a scaling and a *shift* which will fit them in the unit range.

This estimate consists of the 0.01 and 99.99th percentiles[4] of the activation values, computed channel-wise over a significant portion of the training data ($\sim 10\%$). These values will be referred to as ε and λ, as shown in figure 3.7.

The desired behavior of the network after normalization can be better explained as a multiple step recursion. The normalized activations of channel i in layer l ($\tilde{a}_i^l$) compare to the original activations (a_i^l) as $\tilde{a}_i^l = (a_i^l - \varepsilon_i^l)(\lambda_i^l - \varepsilon_i^l)^{-1}$. A downstream neuron in layer $l+1$ will get these normalized activations, decode them to their original magnitude, compute their own activation, and finally normalize this new activation.

This process, however, is performed in a single step and absorbed to the convolution weights of each channel as shown in equation 3.6: where $\tilde{w}_{i,j}^l$ is the normalized kernel

[4]These percentiles have been empirically found to give the best results.

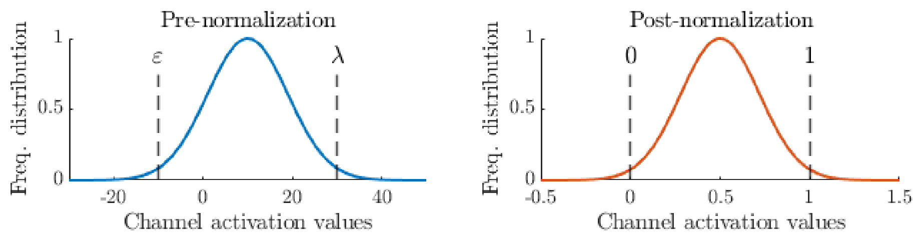


Figure 3.7: Explanatory graph for the shifted channel normalization.

from channel i in layer $l-1$ to channel j in layer l, and $\tilde{b}_j^l$ is the normalized bias for channel j in layer l.

$$\begin{array}{ll} \text{Input layers:} & \text{Hidden layers:} \\ \tilde{w}_{i,j}^l = w_{i,j}^l \dfrac{1}{\lambda_j^l - \varepsilon_j^l} & \tilde{w}_{i,j}^l = w_{i,j}^l \dfrac{\lambda_i^{l-1} - \varepsilon_i^{l-1}}{\lambda_j^l - \varepsilon_j^l} \\ \tilde{b}_j^l = \dfrac{b_j^l - \varepsilon_j^l}{\lambda_j^l - \varepsilon_j^l} & \tilde{b}_j^l = \dfrac{\left[b_j^l + \sum_i \left(w_{i,j}^l \varepsilon_i^{l-1}\right)\right] - \varepsilon_j^l}{\lambda_j^l - \varepsilon_j^l} \end{array} \tag{3.6}$$

It is necessary to address the case of layers with multiple inputs, as the implementation done in the `SNN_Toolbox` needs to be adapted to the shifted channel normalization method. A new layer type `NormAdd`, has been designed for this purpose.

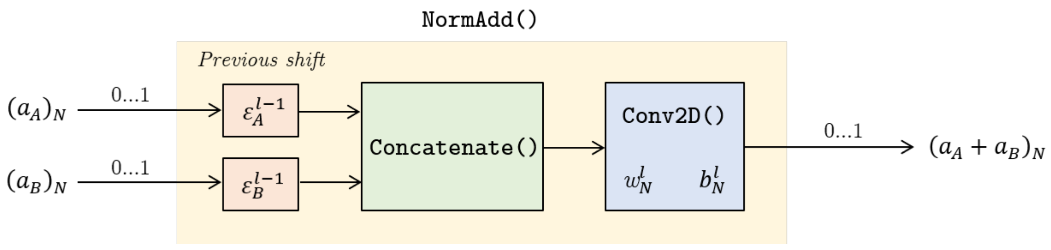


Figure 3.8: Structure of layer `NormAdd`

The new Keras `NormAdd` layer wraps in it the concatenation plus convolution approach originally used in the toolbox, but adds a previous step that allows for proper decoding of several inputs, applying a different shift to each of them. This makes possible to preserve the relative scales and locations of the input values prior to the summation so that the information is correctly transmitted. A scheme of the layer structure is displayed in figure 3.8.

The above-named method has been applied through the function `channel_norm_J`, as explained in table 3.5.

A few samples of the resulting normalized activations of RetinaNet are presented in figure 3.9. The channel normalization shows a better distribution of the data, something very advantageous against the discretization source of error. As the normalization was carried out using the 99.99th percentile, overflow can be spotted in some layers. However, the presence of these few outliers showed no impact in the final model's performance.

It is worth mentioning that the final predictions of the model are also normalized. The outputs of the normalized model and of the generated spiking model will need to be up-scaled to its original size before decoding the predictions or computing the loss.

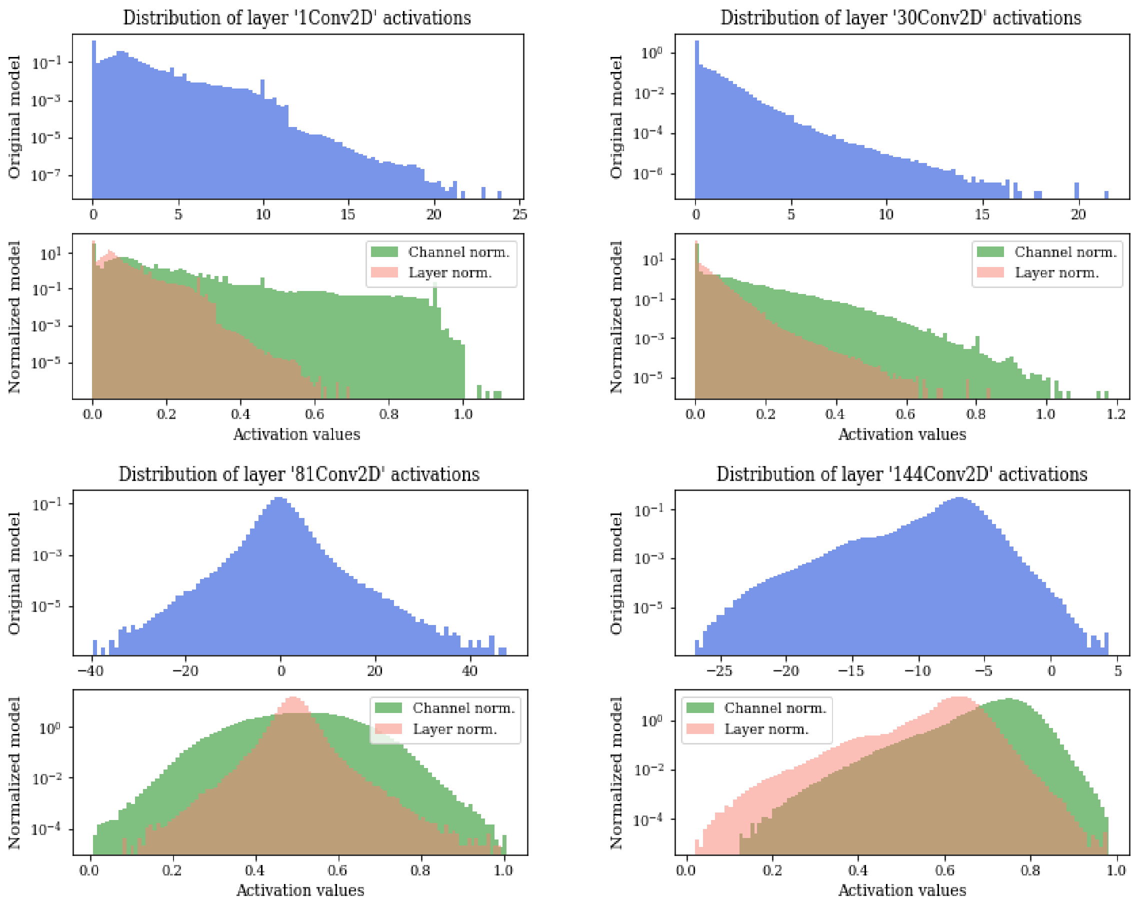


Figure 3.9: Activation values distributions before and after (shifted) normalization. Notice how channel normalization makes better use of the 0−1 interval to display the information and achieves larger activation values (and higher spiking rates after the conversion).

3.4.4.3 Conversion

From all the options available in the toolbox (3.4.3.1) the project will be limited to perform the conversion to a spiking neural network with temporal-mean-rate coding (*R-SNN*). The chosen platform for deployment is the keras INI simulator, which is embedded in the toolbox and allows for modifications.

Table 3.5: Overview of the `channel_norm_J` function

`channel_norm_J()`
Inputs:
• `model`. The parsed model (Keras).
• `config`. A configuration file used to input in the toolbox parameter values for the whole conversion process (e.g. norm. percentiles).
• `norm_set`. Image dataset to be used for the gathering of ε and λ.
Implicit outputs:
• `model`. Normalized parsed model (Keras).

Table 3.6: Overview of the `.build_v2` function

`SNN.`**`build_v2()`**
Inputs:
• `self`. The class `SNN`, that contains the new spiking model and the tools to generate it.
• `model`. The parsed model (Keras).
• `loss_fn`. The loss function to use for the compilation.
• `optimizer`. The desired optimizer.
Implicit outputs:
• `self.snn`. Spiking model (Keras) equivalent to the input model.

Although most of the layers in RetinaNet do have a spiking version implemented in the INI simulator (table 3.2), it was necessary to develop a spiking-compatible version of two additional layers: `UpSampling2D` (as mentioned in 3.4.3.2 (3)) and `NormAdd` (section 3.4.4.2).

The conversion step is performed through the class `SNN`, which stores the generated spiking model along with the necessary tools. The main protagonist is the `.build` method, which had to be modified to accept a custom loss function and optimizer. This new `.build_v2` function assembles and compiles a Keras model with the structure and parameters of the parsed model, but using the custom spiking layers from the toolbox. The biases in the model are adjusted to the time resolution of the simulation as $\hat{b}_i^l = b_i^l \cdot \Delta t$. An overview of the function can be found in table 3.6.

3.4.4.4 Simulation

The INI simulator runs on the preexisting Keras functionalities. The class `SNN` contains the method `.predict` that simulates the spiking model within a given time window.

The input (image) is fed to the model as a matrix of constant input currents. Then, it is propagated through the network altering the states of the neurons across each timestep of the simulation. The spikes generated in the final layer are accumulated and averaged over the duration to compute the spiking rates. These are returned as the encoded predictions of the model.

Listing 3.1: Code for `SNN.predict` method.

```python
def predict(self, x=None):
    """ x --> Input to the network. """
    input_b_l = x * self._dt #Input scaled to the chosen dt
    num_timesteps = self._get_timestep_at_spikecount(input_b_l)
    output_b_l_t = np.zeros(self.snn.layers[-1].output_shape)
    self._input_spikecount = 0

    for sim_step_int in  range(num_timesteps):
        #Computation of the spikes in each timestep
        sim_step = (sim_step_int + 1) * self._dt
        self.set_time(sim_step)
        out_spikes = self.snn.predict(input_b_l)
        #Accumulation of the generated spikes
        output_b_l_t += out_spikes[0] > 0

    return output_b_l_t / self._duration
    """ Return the spiking rate """
```

Other more elaborate methods have also been created to evaluate the performance of the spiking model in section 3.4.5 such as `.run_analysis`, which, in addition of the predictions of the model, it plots the evolution of the loss through the simulation time and the correlation of the obtained spike rates with the analog activations of the parsed model.

3.4.5 Results

The spiking neural network conversion took as basis a publicly available implementation of RetinaNet, which is pre-trained on the on the Microsoft Common Objects in Context (COCO) dataset [48]. This was used as test-bench for the adaptation and testing of the `SNN_Toolbox`, a previous step to the application of the algorithm to the objectives of the ShippingLab project. Training on the marine data was only launched when good results were obtained at this first stage.

It is important to clarify that the spiking neural networks' results have been obtained by emulation in the INI/Keras framework, where the computational cost of $1ms$ of virtual time is $300ms$ of real time. The objective of the `SNN_Toolbox` is to act as a translation and validation step towards the future porting of the obtained spiking model to the appropriate platform, where it would run in real time.

3.4.5.1 COCO implementation

Although the model used has its source in the official Keras RetinaNet code example [89], it has been altered and fine-tuned on COCO for 1 epoch to swap the ResNet50's

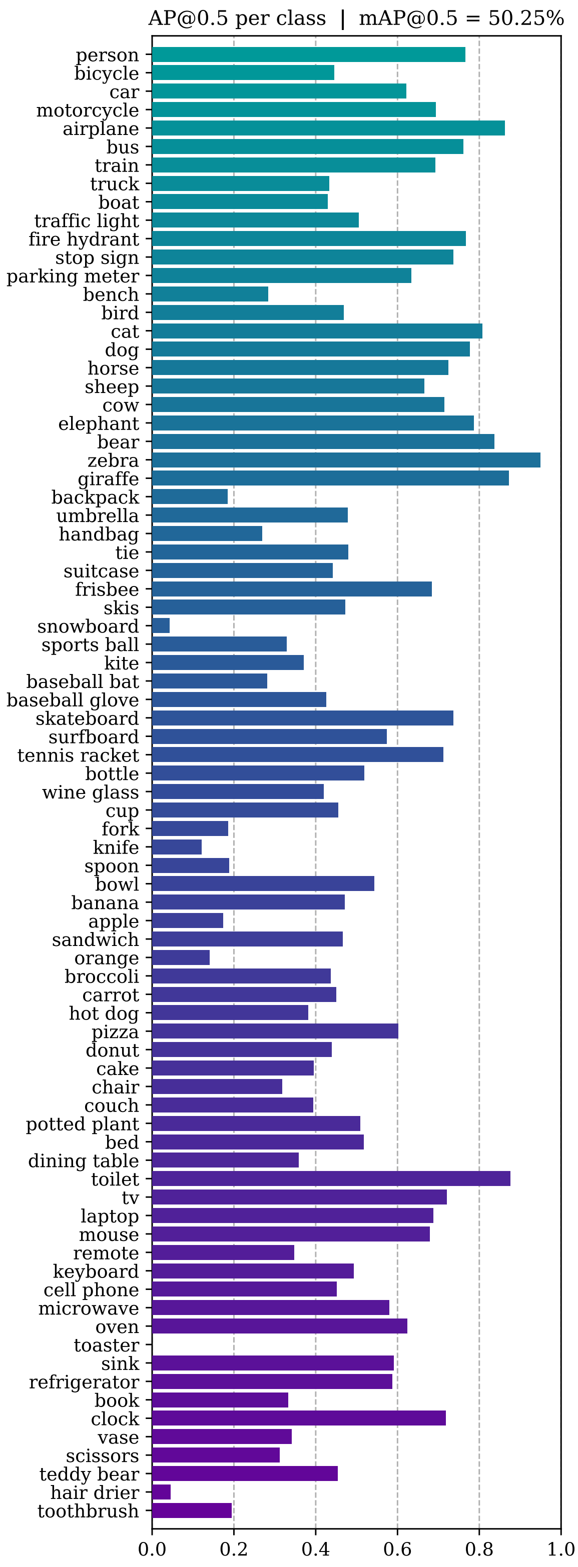


Figure 3.10: Mean Average Precision for the average-pooling, analog Keras RetinaNet on the MS. COCO dataset. This metric was briefly explained in section 2.3.

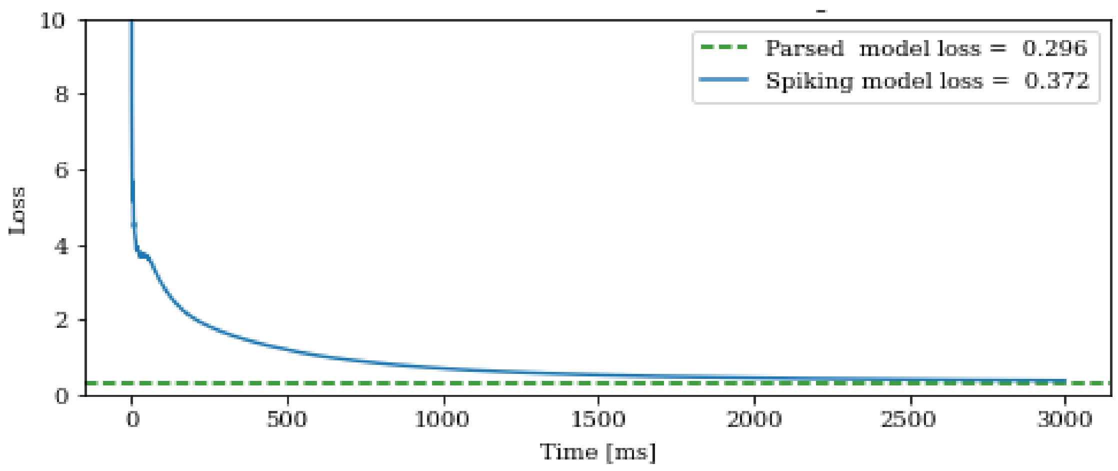


Figure 3.11: Evolution of the SNN RetinaNet loss with the size of the simulated time window. Acceptable predictions were obtained after a simulated time of $2500ms$. The curve displayed corresponds to sample 73 of the MS. COCO validation subset.

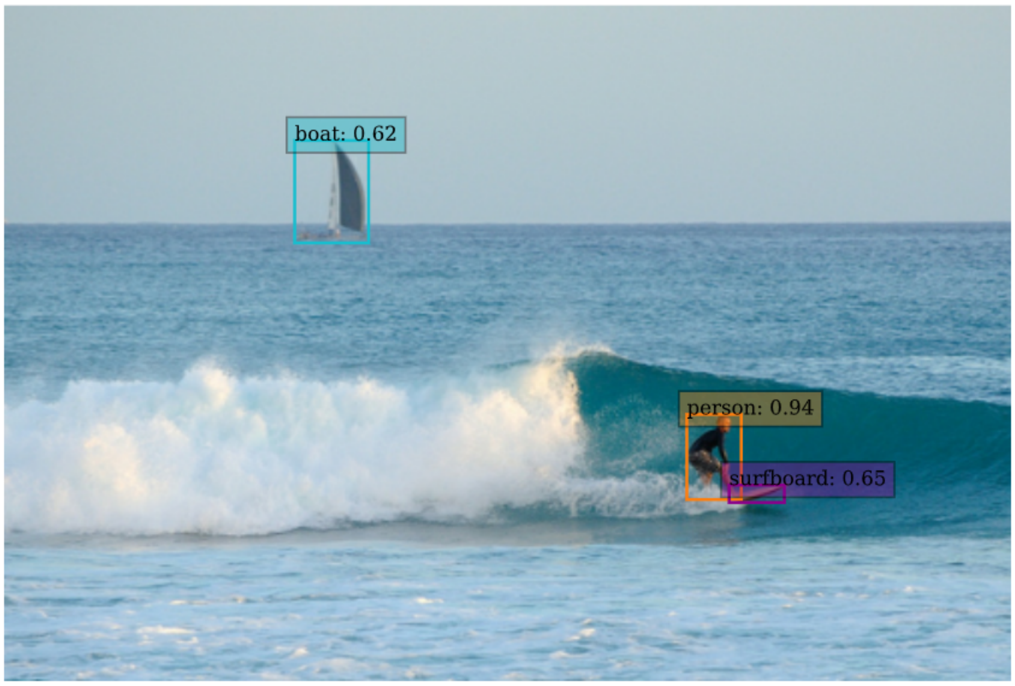
max-pooling layers for compatible average-pooling ones. It achieves a mAP@0.5[5] of 50.25% on the COCO data as shown in figure 3.10.

The performance of the obtained spiking model essentially converges to the one achieved by the analog network given enough running time of the spiking model. This will be illustrated with sample 73 of the COCO validation subset (see figure 3.12), but can be extrapolated to the whole data.

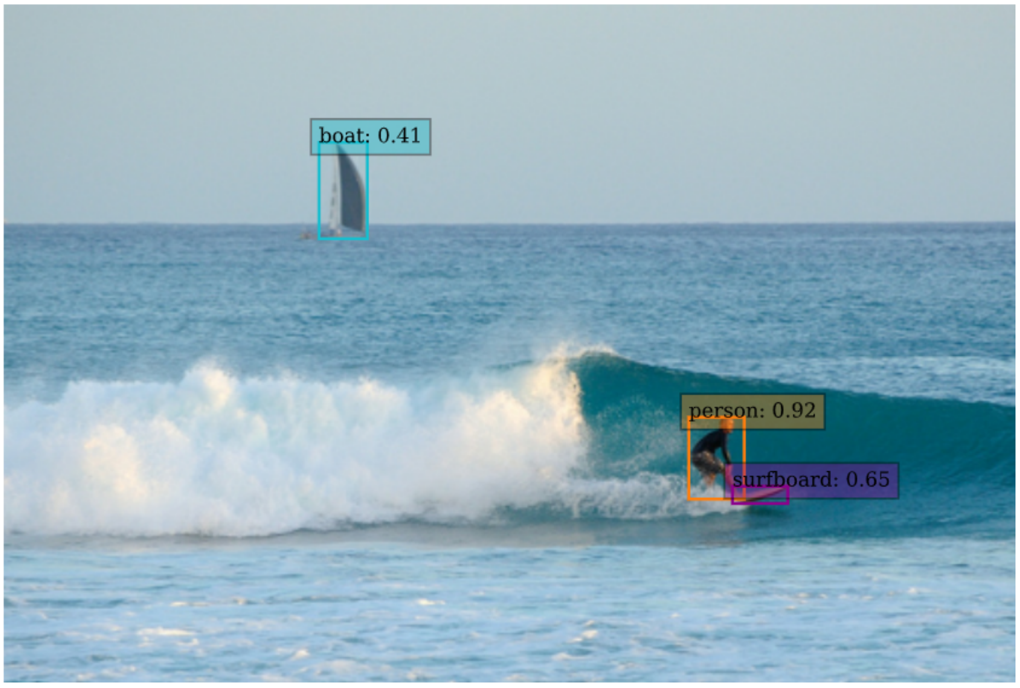
As shown in figure 3.11, the achieved loss falls very close to the analog model's (dashed green line) after $3000ms$ of simulation. This is the window size needed for the resolution of the spike trains to hit the requirements of the object detection and classification problem. The discrete sampling error gets then low enough that the predictions are minimally affected by it. The evolution of this error can be appreciated by checking the correlations of between the original analog activations and the obtained spiking rates in each layer of RetinaNet. Table 3.7 does this for the output layer, and graphically shows the elegant increase of the correlations with the integration time. This convergence would have been slower with the layer normalization approach due to lower spiking rates, needing more time to convey the information and lead the final layers to the desired steady state.

A further improvement of this convergence time is within reach, though, with a simple modification to the prediction function. Looking closely at the loss curve in figure 3.11, it is possible to notice an irregular behavior in the first $150ms$. The source has been traced down to be the transient state of the network, which is the time needed for the spiking rate of the last layers to stabilize. This obeys to the delay caused by the propagation of the information through such a deep spiking model.

[5]The mean average precision (mAP) is a popular metric used to measure the performance of models doing document/information retrieval and object detection tasks (see section 2.3).



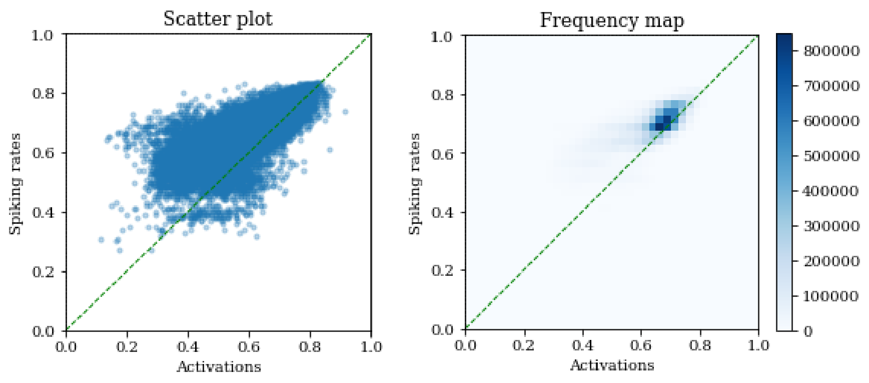
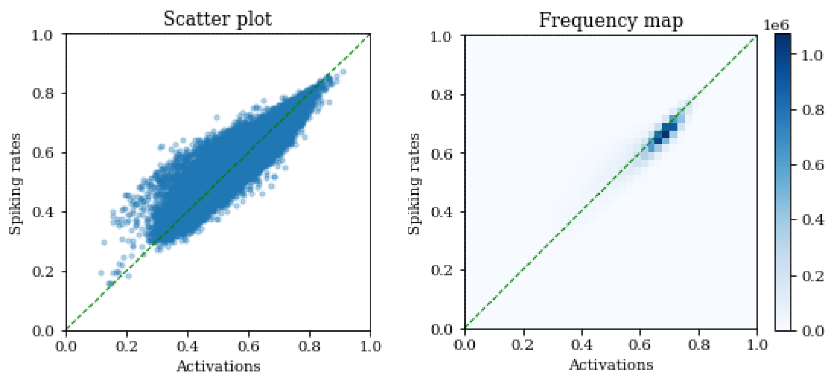
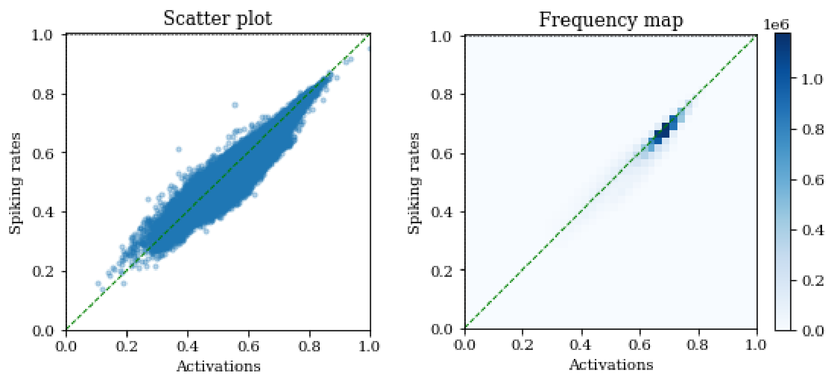
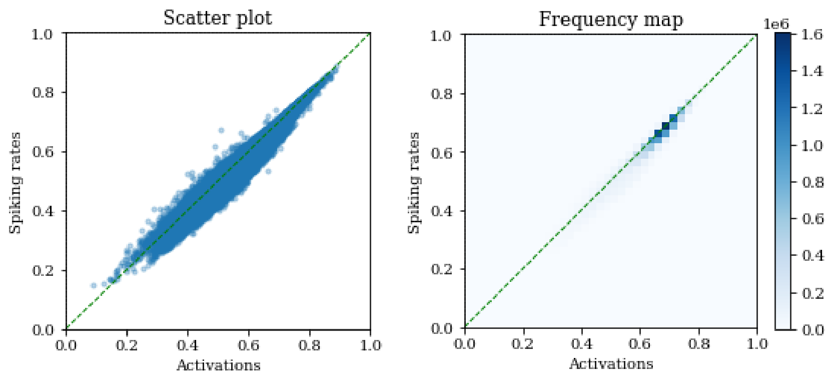
(a) Predictions of the original ANN RetinaNet.



(b) Predictions of the obtained SNN RetinaNet.

Figure 3.12: Predictions of the models over sample sample 73 of the MS. COCO validation subset. More can be found in appendix A.

Table 3.7: Correlations between analog activations and obtained spiking rates of the output layer after the conversion of RetinaNet. Notice the convergence over the model run-time due to the reaching of the network steady-state and the diminution of the discrete sampling error. Additional plots can be found in appendix B.

100 ms	
500 ms	
1500 ms	
3000 ms	

The spikes captured in the last layer during this transient state provide an unreliable prediction and, when taken into account they add a substantial error to the spike-rates calculation. While it will eventually be compensated by future spikes in the simulation, this will be time-consuming. By modifying the prediction method in 3.4.4.4 to ignore outputs in this time interval it is possible to functionally speed up the convergence of the spike-rates between 3 and 5 times. The transient time has been empirically found to be approximately of one timestep ($1ms$) per layer, and has been rounded up to $200ms$ as displayed in figure 3.13. The obtained predictions are shown in figure 3.12b.

Moving on to other issues, there is yet another type of error to be addressed. This one obeys to a phenomenom caused by the shifted channel normalization which affects to the borders of the activation matrices.

In convolutional layers with kernel size of 3x3 or superior, the shift performed to negatively activated channels is not properly absorbed to the layer weights and creates a distortion in the borders of the matrix. The error in this area can reach relative values as big as 80% when propagating to higher layers, something very worrying at first glance. This error is well depicted in figure 3.14, where activations of layer 58 barely show any signs of it (3.14a), but activations of layer 126 have a clear stripe framing the error matrix (3.14b).

The most susceptible objects to be miss-detected due to its influence are relatively big items located close to the borders of the image, as these features undergo the most convolutions and fall into the smallest level of the feature pyramid. However, after several tests on the data it was concluded that this error had virtually no major impact on the model's performance and no further actions were taken. This could be due to the majority of the relevant information not being located near the borders of the matrix

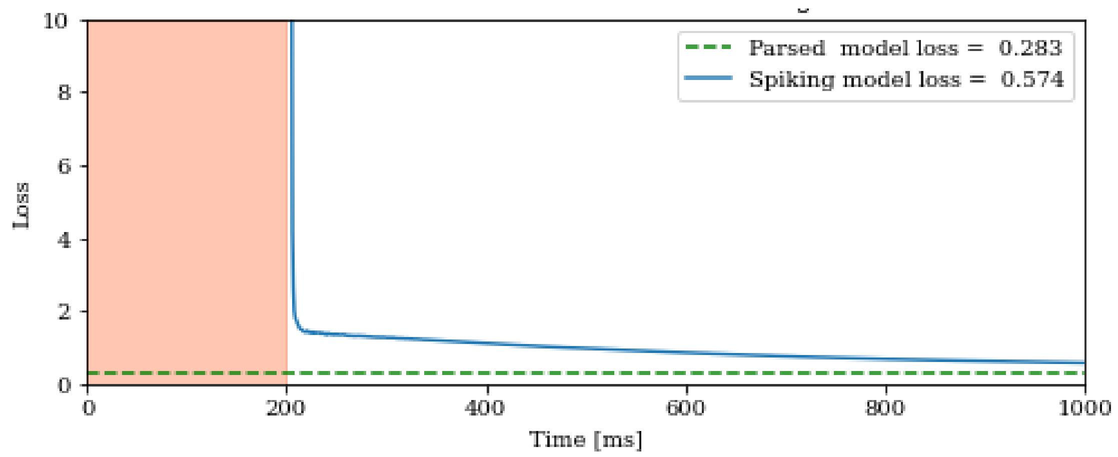
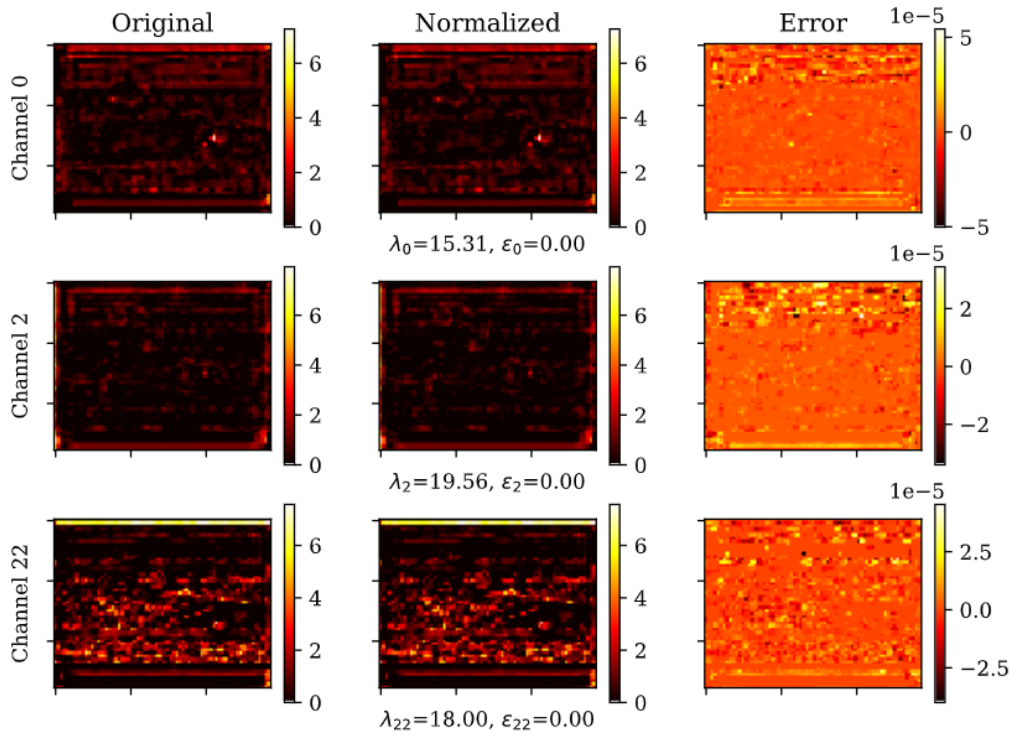
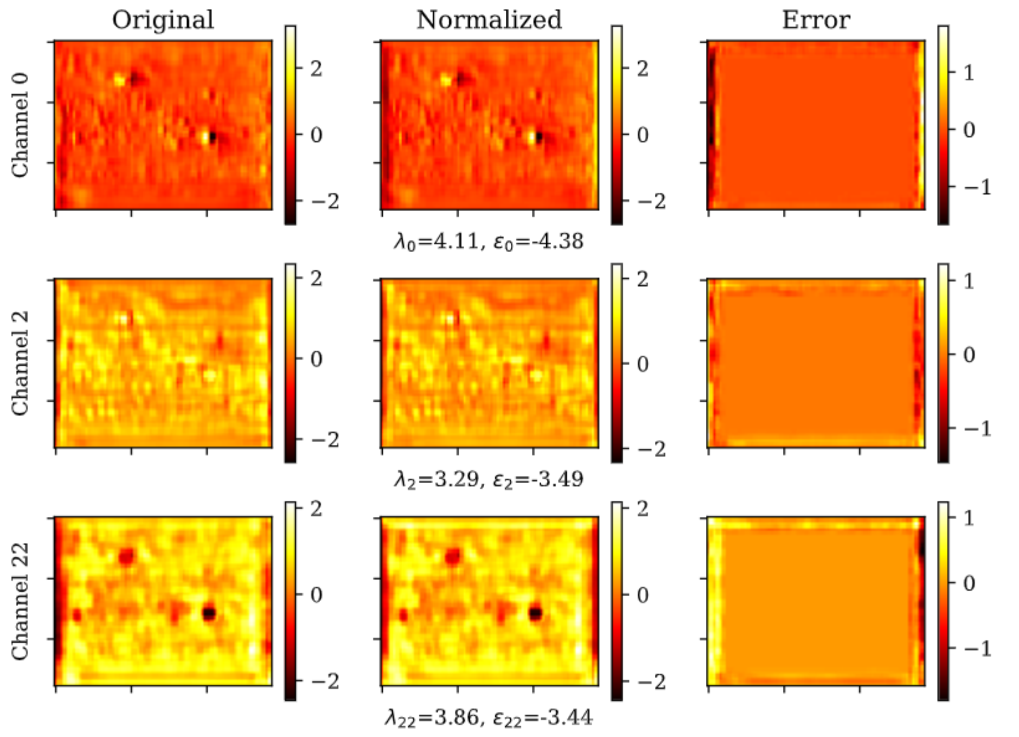


Figure 3.13: Evolution with time of the SNN RetinaNet loss after disregarding the transient state. After a big first drop in the loss, the resolution of the model increases slowly until reaching good results at $1000ms$. The curve displayed corresponds to sample 73 of the MS. COCO validation subset.



(a) Activations for layer `58Add`.



(b) Activations for layer `126Conv2D`.

Figure 3.14: Comparison between the analog activations of layers 58 and 126 of the parsed and normalized RetinaNet models. The input is sample 73 of the MS. COCO validation subset. The normalized activations have been decoded and resized to enable a visual comparison. The error is computed as the difference between the two.

and therefore not being affected.

Having said that, this phenomenon could be fully eliminated by falling back to the regular channel normalization method from [36] and only converting networks with no negative activation values (e.g. only ReLU or similar).

3.4.5.2 ShippingLab implementation

Having obtained good results on the COCO dataset, the RetinaNet model was retrained on the ShippingLab dataset and converted to SNN for further testing of the method. This dataset is conceived for an application of deep-learning models towards environmental awareness in the sea, having two possible object classes: *boat* or *buoy* (see section 2.1.2 for more information). The model adaptation to this new data entailed two major issues that had to be solved before exposing the model to it.

The first issue was related to the format in which data was stored in the datasets. While COCO was obtained in `tensorflow_datasets (tfds)` format [13], the ShippingLab data was streamed from less straightforward `tfRecord` files. The training pipeline had to be adapted by the inclusion of new functions to properly handle and decode the data.

Conversely, the second issue was much harder to discover. The anchor box sizes and ratios, which were introduced in [47] and worked beautifully in regular benchmarking datasets, were unable to properly fit the bounding boxes in the ShippingLab data. This is due to the much higher range of possible object sizes in the marine pictures compared to the COCO data, from obstacles far away in the horizon to ships crossing a few meters ahead. This was previously addressed in [75] by the use of K-means clustering to find the best anchor sizes and ratios that would accomodate all objects. The results obtained in that article were used as reference, and the values finally used are:

$$\begin{aligned} \text{scales} &= \left[2^{-2/3}, 2^{0}, 2^{2/3}\right] \\ \text{aspect ratios} &= [0.8, 1.4, 2.6] \\ \text{sizes} &= [13.8, 21.0, 34.3, 59.8, 131.7, 320.0] \\ \text{strides} &= [8, 16, 32, 64, 128] \end{aligned} \tag{3.7}$$

Transfer-learning was performed to speed up the training process; in other words, the learned parameters of COCO RetinaNet were used as starting point for the training of the new model. The results shown in this section correspond to a 24h training on the ShippingLab data, which achieved a mAP@0.5 of 97.16% on the ShippingLab validation subset (figure 3.15).

RetinaNet achieves a much more robust performance here than on the COCO data and, after the conversion, the spiking model shows a much faster convergence time. This was expected beforehand and can be explained by the small number of classes in the ShippingLab data (only 2), which make for an easier training target for the algorithm and can be correctly expressed in a smaller spike-train resolution.

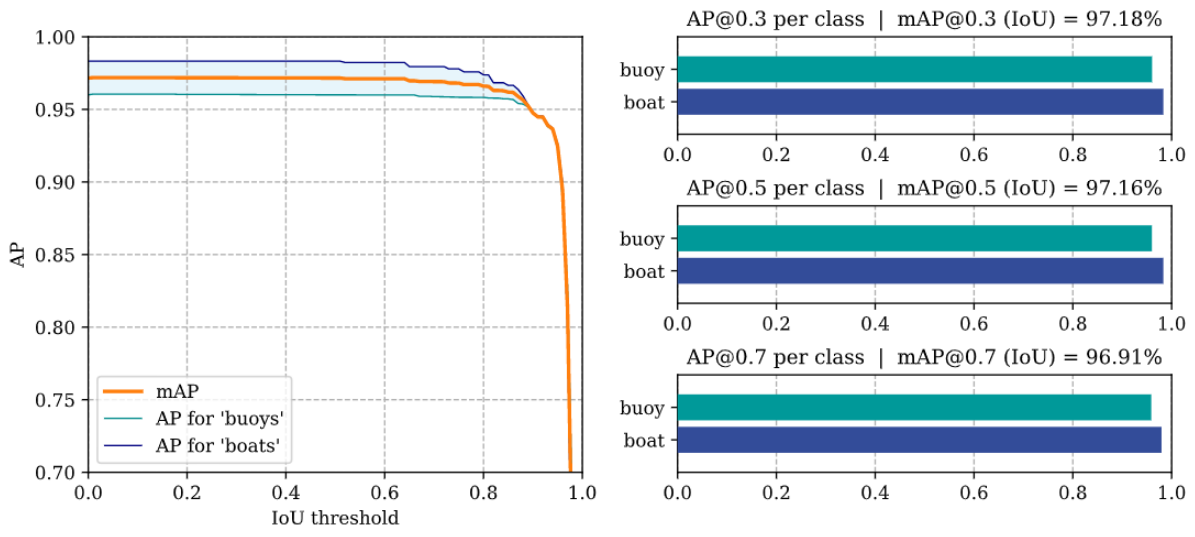


Figure 3.15: Mean Average Precision of the analog RetinaNet model on the ShippingLab validation set.

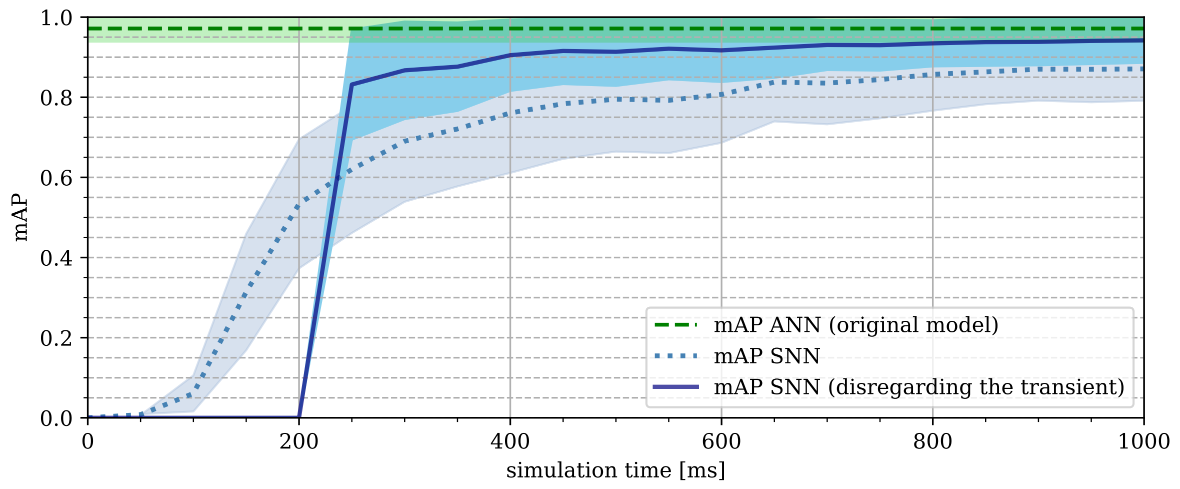


Figure 3.16: Mean Average Precision (@0.5 IoU) evolution with the simulation time of the spiking model (computed every 50ms). The original model score is displayed as a reference (green dashed line). The graph was computed using 20 data-samples from the ShippingLab validation set.

An adequate benchmarking between the converted (spiking) model and the original RetinaNet has been carried out through the use of the mean Average Precision (mAP) metric, a state-of-the-art criterion which targets specifically the object detection problem (see section 2.3). Figure 3.16 displays the mAP score for the spiking model using a threshold IoU of 0.5 between the predicted bounding boxes and the labels (ground truths). As expected, the accuracy of the spiking model predictions effectively increases with the integration time window reaching comparable results to the original ANN RetinaNet (marked in green for reference).

By ignoring the spiking output produced in the first $200ms$ (network's transient state), the results reach a mAP@0.5 of 95.43% after a network run-time of $1000ms$. This is roughly equivalent to a drop in performance of 1.6% from the original model. This decrease could be reduced at the price of rising the simulation time for applications where greater precision is required and detection speed is not a decisive factor.

It is important to clarify that this test was performed over 20 samples of the ShippingLab validation set for computational reasons. However, these samples were picked from all relevant scenes of the video sequence and the mAP obtained on them for the analog RetinaNet matches the score thrown by the full validation set. Obeying to this logic, the results in figure 3.16 are liable to provide a good insight into the real performance of the spiking model, nevertheless, analysis on a larger dataset should be conducted if implemented in neuromorphic hardware.

For a more qualitative scrutiny, figures 3.17 and 3.18 compare the predictions obtained by both analog and spiking RetinaNet over samples 190 and 220 of the validation subset. Notice the fast convergence of the loss function, only needing $100ms$ to reach a suitable result after the $200ms$ transient length.

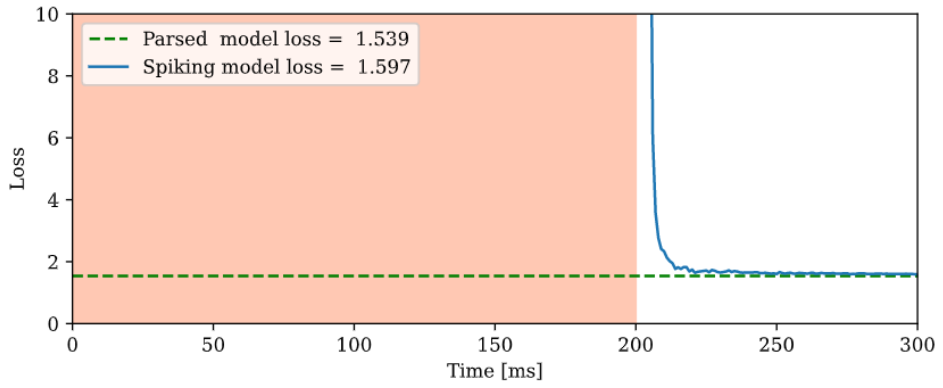
3.4.6 Limitations

Although promising results were shown in the previous section, several trade-offs were needed to achieve a working spiking-RetinaNet model through conversion of an off-the-shelf Keras RetinaNet implementation.

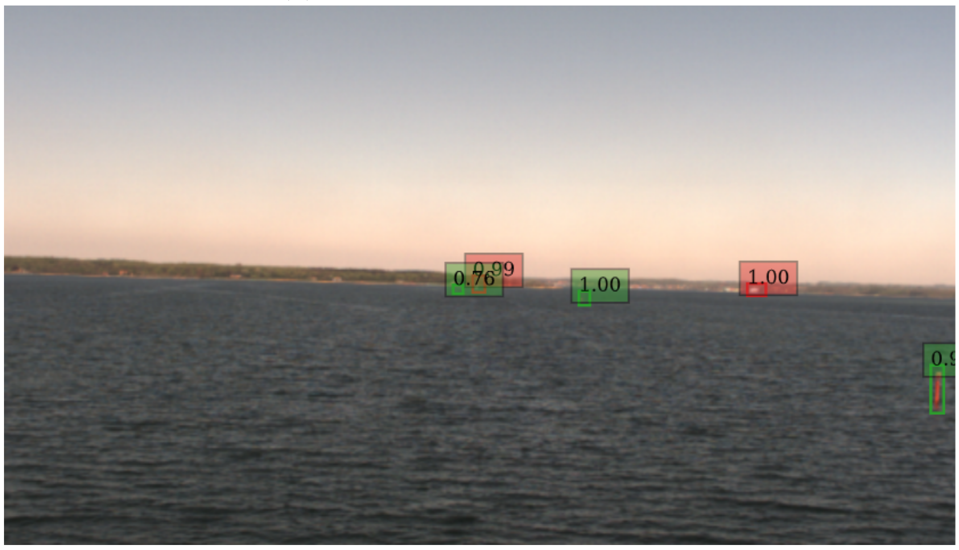
Regarding the shifted channel normalization and custom `NormAdd` layers (section 3.4.4.2), although they enable for a meaningful translation of deep-neural networks with negative activation values, they generate distortions in the borders of the spike-train matrices. The reach of this phenomenon is yet to be studied, along with the full compatibility of the `NormAdd` layers with a neuromorphic chip implementation.

A possible alternative is to stick to the regular channel normalization method proposed in [36] by training a strictly-positive RetinaNet version as base ANN. This network would also benefit for faster inference times due to the more constrained activation subspace, which would bear a better distribution when normalized to the unit interval. Nevertheless, it is yet to be seen if these limitations could negatively affect its reliability as an object detector.

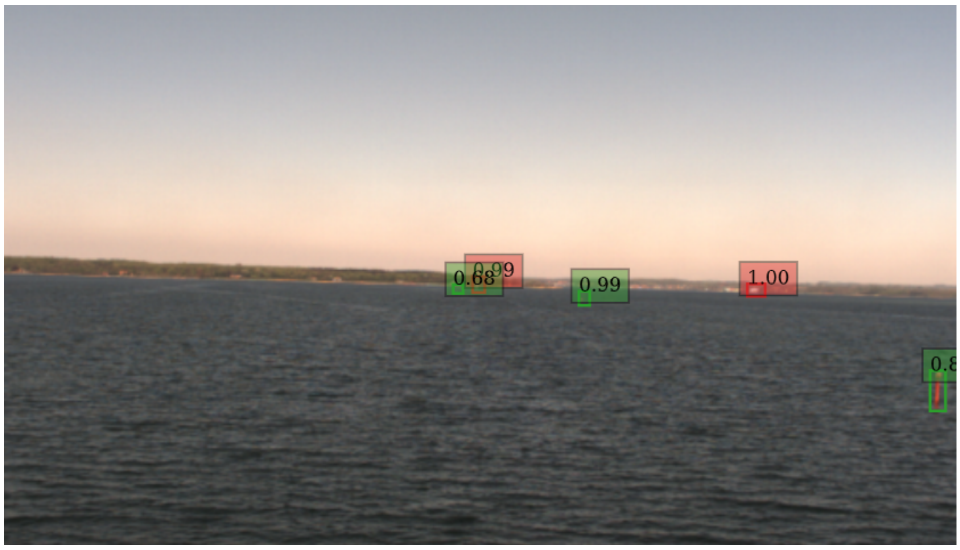
Regarding the chosen encoding scheme for the spike-trains, achieving state-of-the-art accuracy with rate-based networks comes at the cost of using high firing rates and long integration times to obtain reliable results. This reduces the energy consumption gap between traditional methods and SNNs. Using temporal encoding is an attractive alternative, but, for the moment these approaches could not be easily adapted to the object detection problem.



(a) Loss of the SNN prediction.

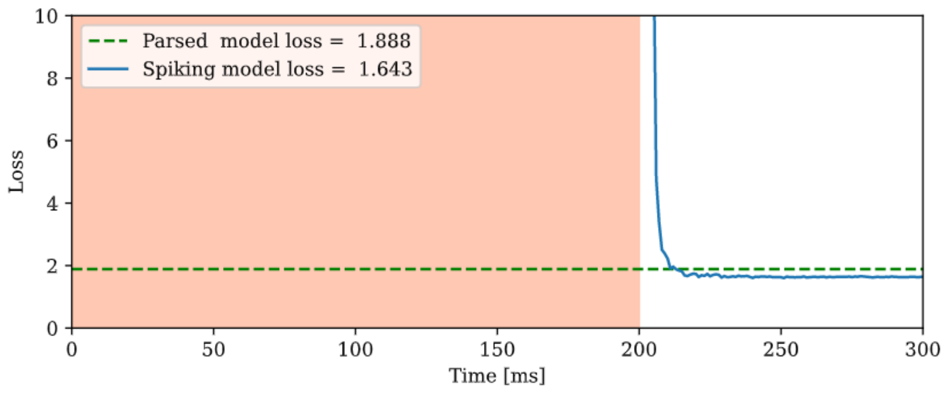


(b) Detections of the ANN RetinaNet.

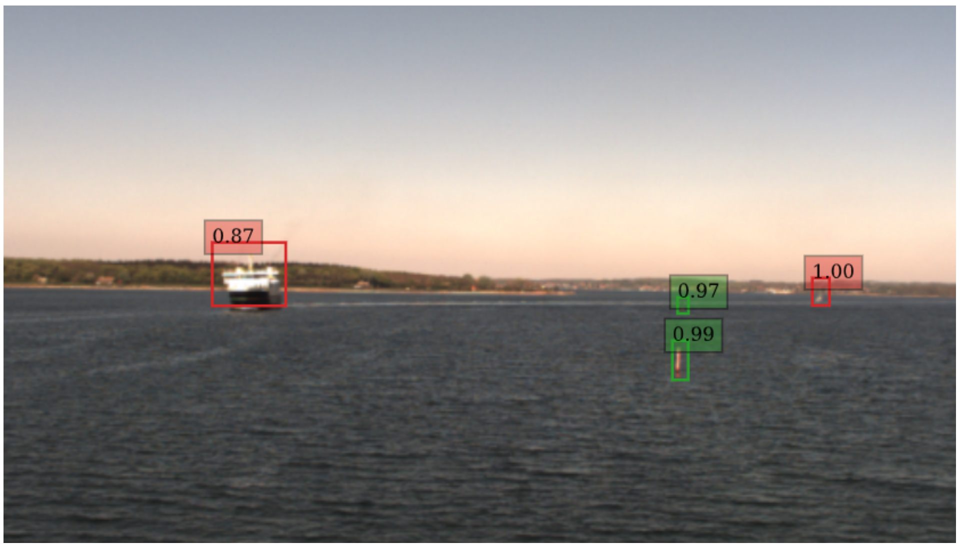


(c) Detections of the SNN RetinaNet.

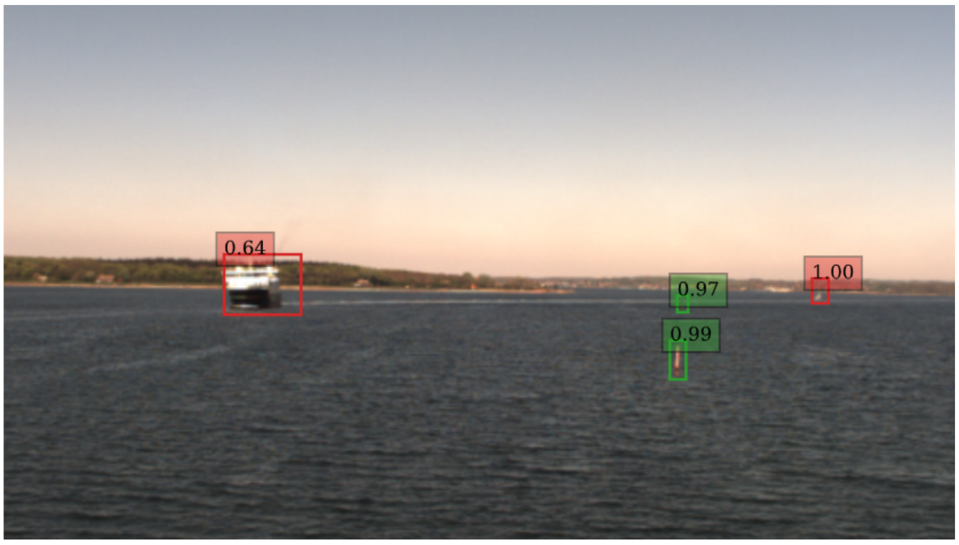
Figure 3.17: Predictions of the models over sample 190 of the ShippingLab validation subset. The 'boat' class is marked in red and the 'buoy' class in green. Good results are obtained ~ 5 times faster than on the COCO dataset.



(a) Loss of the SNN prediction.



(b) Detections of the ANN RetinaNet.



(c) Detections of the SNN RetinaNet.

Figure 3.18: Predictions of the models over sample 220 of the ShippingLab validation subset. The 'boat' class is marked in red and the 'buoy' class in green.

For the decoding of the predictions, the followed approach was to integrate the final layer's spikes over the simulation time and compute the spike rates. Although these do converge to the original ANN activations, they lead better results when ignoring the spikes generated during the initial transient state. This scheme contradicts other of the advantages of neuromorphic computing, which states that 'Deep SNNs can be queried for results already after the first output spike is produced, unlike ANNs where the result is available only after all layers have been completely processed [17]'. It might be a better approach to infer the predictions using the last layer's membrane potentials or using a sliding time window.

It is also worth mentioning that the analysis performed on the spiking-RetinaNet is constrained by the nature of the used datasets, which were tailored for ANN conventional methods. Performance of the spiking models during training and predictions would benefit of the use of event-based streamed data as input, as SNN show better results in online learning and precise-timing frameworks. These would require though, to redesign how the data is captured by the vessel sensors and was out of the project's scope.

Finally, the converted network has been tested in a simulation environment, within the same framework where the conversion was performed. This, although being logistically convenient, has sacrificed all the computational and efficiency advantages that the spiking model running in compatible hardware would display. This project serves then as a preliminary analysis that demonstrates the possibility of generating state-of-the-art ODC deep-learning models, just missing their implementation in a neuromorphic platform for a conclusive benchmarking.

3.5 Whetstone

In parallel to the conversion approach taken in section 3.4, a different method was found in [77] to specifically train deep artificial neural networks for binary neuron synapse using existing deep learning methods. This method does not produce spiking neural networks, but analog networks with binary, threshold-activation functions.

In contrast with the spiking neurons, the Whetstone neurons behave in a static manner, with no time dimension. The network responds instantaneously, making it compatible both with existing deep learning frameworks and many neuromorphic processors (using a single timestep window). The obtained model is portable and can easily be instantiated on neuromorphic and conventional platforms.

This subsection will explore the compatibility of this method with the ODC problem and evaluate its performance on the COCO dataset. The investigation was simultaneous to the activities in section 3.4.5.1 yet it never progressed to being adapted for the ShippingLab dataset due to the results not being promising.

3.5.1 Method

The method used is very intuitive, as it consists on an iterative modification of the back-propagation optimization algorithm. It incorporates the obtention of binary activations directly into the training process.

The method works with custom activation functions named as `Spiking_BReLU`. These have an initial stage that mirrors the behavior of a bounded-ReLU function but is designed to morph into a binary step function during the training. This process is referred to as sharpening.

This design choice obeys to the nature of the most common ANN training techniques, which rely on stochastic gradient descent methods. These require the activations of the neurons to be differentiable during the process. Having said that, it is possible to incorporate additional constraints in latter stages of the training without compromising stability, such as a slow mutation of the nodes towards binary communication.

In the Whetstone algorithm, outlined in figure 3.19, a state machine is used to control the sharpening process during the training. This will be triggered after a certain number of epochs or after a certain metric stops improving. The sharpening will be performed very slowly and even halted for some time if the performance of the model is severely affected. The controller will wait until the targeted parameters improve with some more training to relaunch the process. This simple feedback control has the objective of minimizing the impact that binarization may have in the model's performance.

The process is best performed through the incremental sharpening of each layer one-by-one, starting from the input and finishing at the output of the network. It is highlighted in the documentation that the early layers are especially crucial and the biggest source of performance degradation. Once they are properly sharpened, the loss introduced by the following layers is greatly minimized.

3.5.2 Implementation

The application of the Whetstone method has been performed using the resources available in the Github repository: [80]. Nevertheless, some adjustments were needed to make it compatible with newer versions of Python and able to handle deep learning models with multiple subnetworks.

The implementation consists of two main additions to the Keras API: the custom activation layer, `Spiking_BReLU`, and several sharpening schedules that are incorporated as callbacks for the training. These callbacks target the `Spiking_BReLU` layers in the network progressively morphing them into binary units. From these, we will focus on the `AdaptiveSharpener` schedule, which implements the method explained in figure 3.19, allowing for the finest control over the process.

The Whetstone algorithm, however, is designed for small neural networks and the original sharpening schedule operated in a layer per epoch rate. This is non viable for

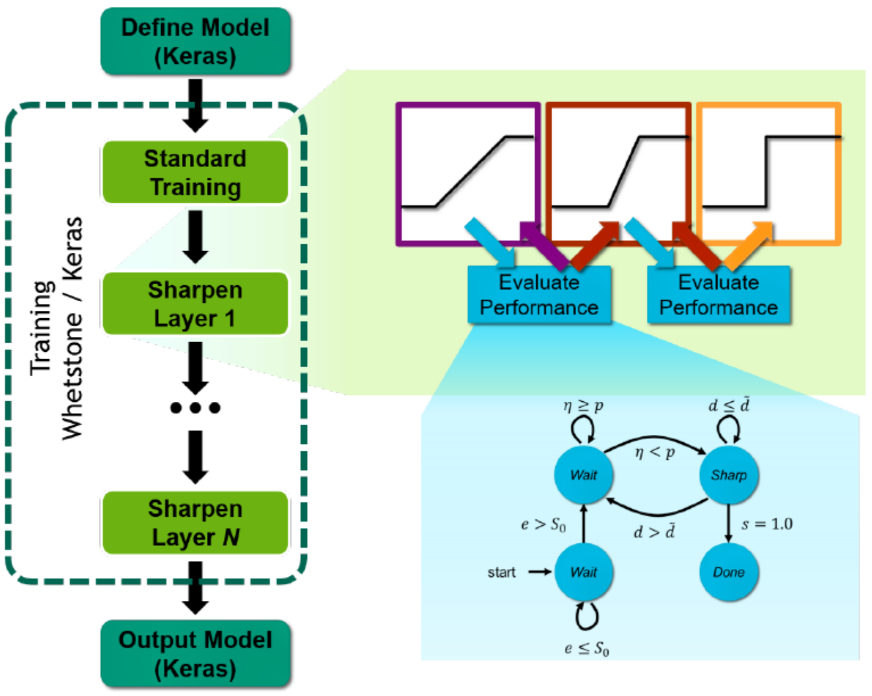


Figure 3.19: Scheme of the Whetstone sharpening method as featured in the source paper [77].

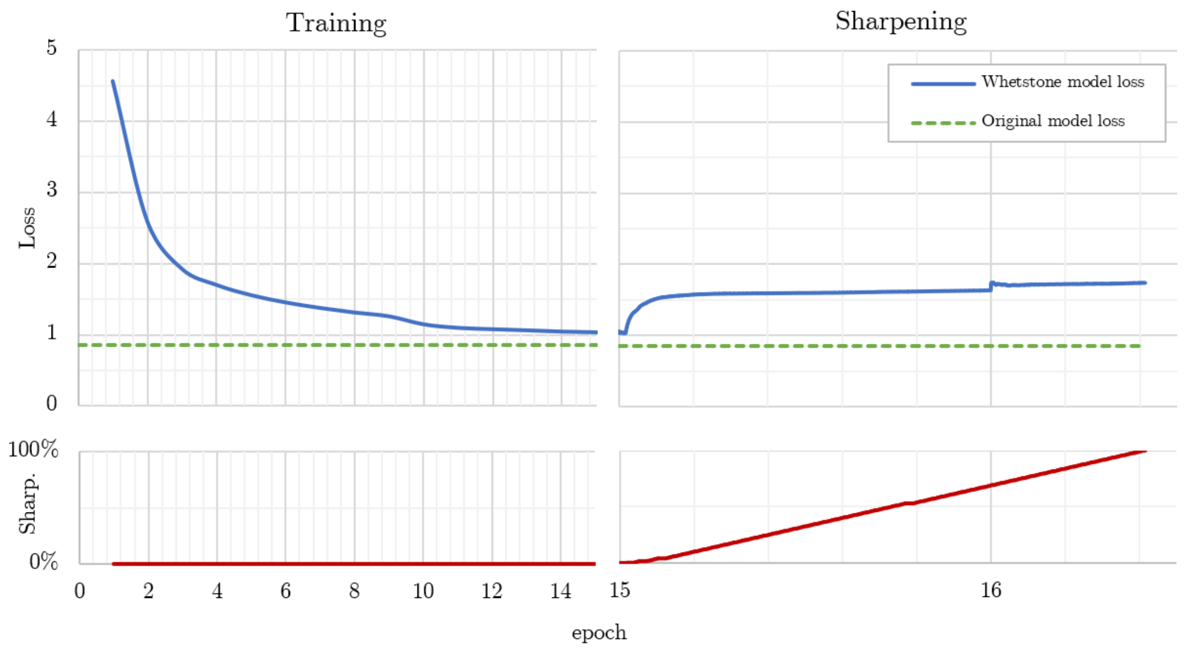


Figure 3.20: Loss evolution during the training and sharpening of Whetstone-RetinaNet.

RetinaNet, as an epoch of the COCO dataset takes around 11h in the DTU Computing Centre and the network has more than 80 activation layers. This has been modified in the new `AdaptiveSharpener_J`, which performs the sharpening batchwise. The main parameters used to guide the process execution are described in table 3.8.

Table 3.8: Overview of the `AdaptiveSharpener_J` callback.

`AdaptiveSharpener_J`	
Configurable parameters:	
• `min_init_epochs=10`	Number of training epochs before launching the Whetstone sharpening.
• `batch_interval=40`	Number of batches between assessments.
• `rate=0.06`	Rate at which the `Spiking_BReLU` layers are sharpened. The sharpness of the layers is controlled by a slider between 0 and 1 and this variable dictates the increments between assessments.
• `critical=0.9`	Sharpness value at which sharpening is performed in critical mode. This mode aims to slow down the sharpening when the derivatives starts diverge.
• `cz_rate=0.02`	Sharpening rate during the critical mode.
• `sig_decrease=0.05`	Decrease in the loss which is considered as an improvement in performance.
• `sig_increase=0.05`	Increase in the loss which is considered as a degradation of performance. If this happens in sharpening stage, the algorithm will immediately fall back to training stage and wait for improvement.
• `patience=20`	If not in sharpening stage and performance is not improving, number of consecutive assessments to wait for improvement before finally deciding to move on to sharpening stage.
• `subnet_idx=[0]`	Index of the targetted subnet (backbone) in the Keras model iterable.
• `start_sharp.=False`	Start training in sharpening stage directly.
Methods:	
• `.assess_sharpening`	Function that automatizes the sharpening control explained in 3.5.1 using the mentioned parameters.
• `.perform_sharpening`	Function that applies the sharpness changes to the targetted `Spiking_BReLU` layers.

3.5.3 Results

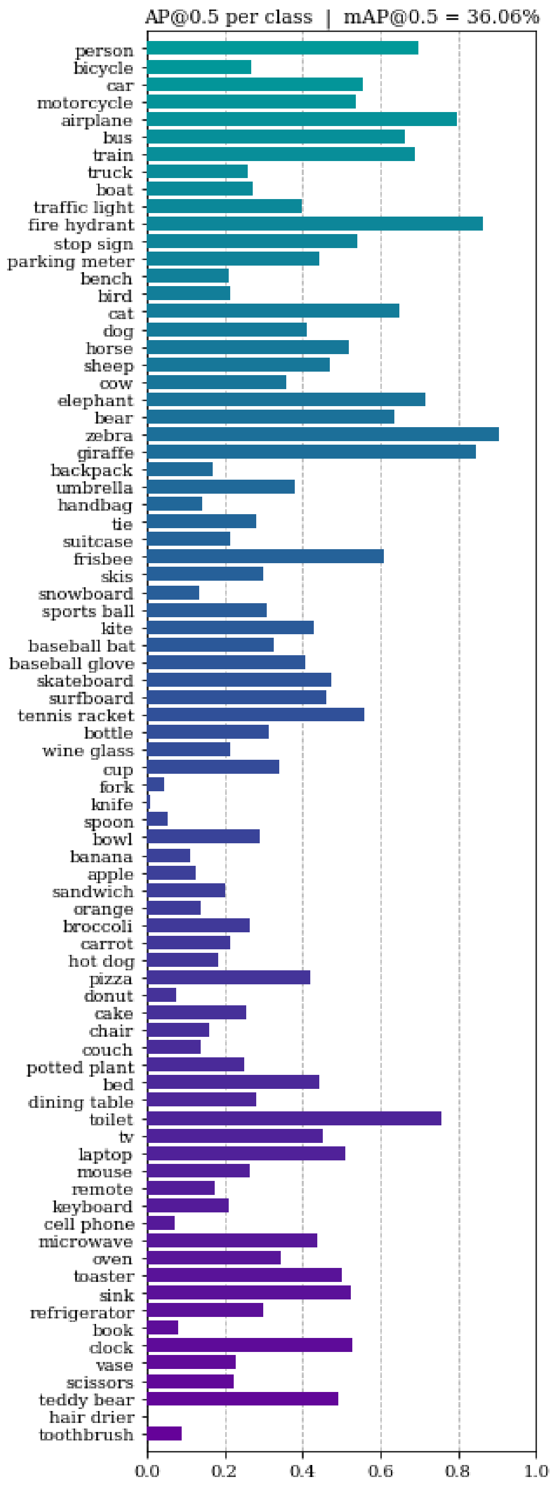
To test the method on RetinaNet it was necessary to build a version of the network that exclusively used the `Spiking_BReLU` layers. The sharpening was only performed on the backbone of the network, however, if good enough results were shown, the box regression and classification subnetworks would have been the next step.

The monitored variable during the training is the RetinaNet loss, whose evolution is displayed in figure 3.20. The process has been performed on the Microsoft COCO dataset with the objective of benchmarking with the RetinaNet implementation of section 3.4.5.1 (dashed green line in the graph).

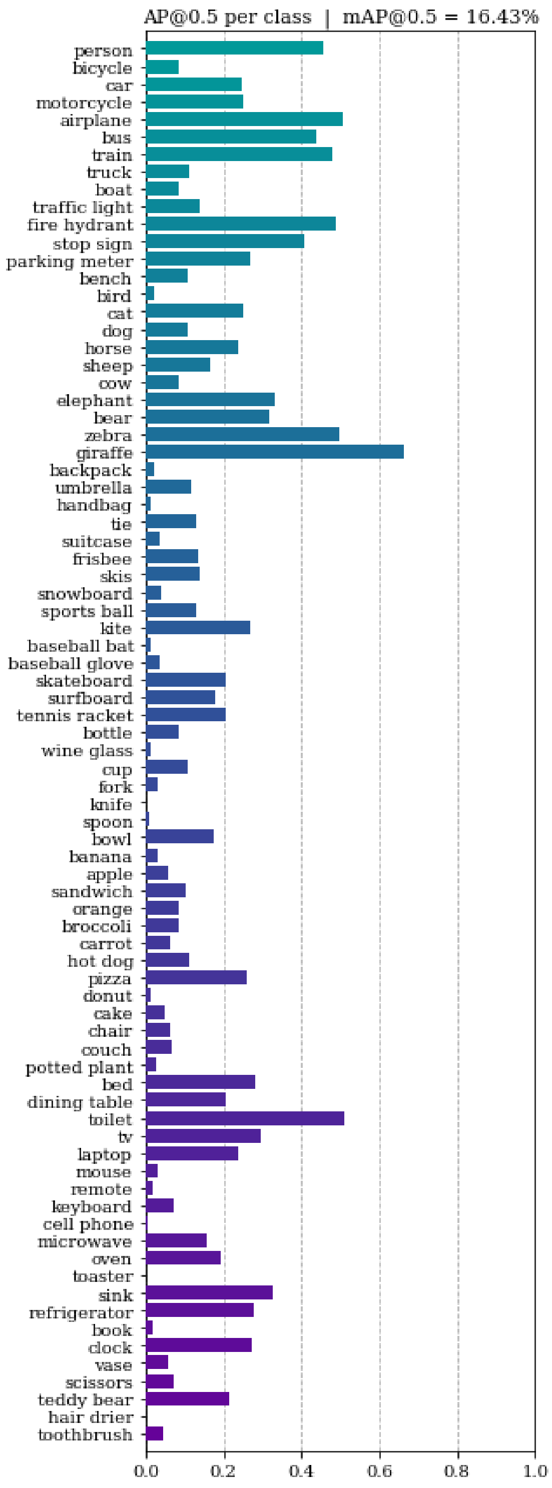
Due to the limit in range and resolution of the bounded-ReLU layers, the model did not reach the performance of the standard RetinaNet model after 165h of training (epoch 15). The sharpening worsens the performance even further, especially in the first steps of the process which concern the initial layers. This is displayed in detail in figure 3.21, which shows side by side the mAP of the model before and after the sharpening takes place.

As the worst performance drop occurs during the sharpening of the first layers, improvement might be possible by using a finer control algorithm that allows to slow the process further down in this first stages. The sharpening rate could be scheduled to start at a very low value and increase progressively for deeper, less critical layers.

Nevertheless, due to the Whetstone model's underwhelming performance even before the sharpening, it is concluded that this method is not suitable for the objectives of our project.



(a) Before sharpening.



(b) After sharpening.

Figure 3.21: Mean Average Precision for the Whetstone Keras RetinaNet on the MS. COCO dataset. Notice the limitations in the model performance due to the bounded-ReLU activations and the negative impact of the sharpening process.

3.6 Summary

An extensive analysis has been performed into the applicability of neuromorphic architectures into object detection and classification deep-learning models. This research has led to the conclusion that, although these approaches still show a low technology readiness level (TRL), it is possible to develop a spiking version of a complex CNN such as RetinaNet, achieving state-of-the-art results.

Three potential paths towards this objective were investigated and tested via software: Training of a deep ANN for the use of binary neuron activations (*Whetstone*), training of a proper SNN, and conversion of a previously trained ANN to an equivalent SNN via rate-encoding. Nevertheless, satisfactory results were only reached through the last of the mentioned approaches: ANN-to-SNN conversion. The conclusions obtained from this analysis have been summarized in table 5.1, located in the Discussion (section 5).

The ANN-to-SNN conversion method was then applied to the ShippingLab objectives, regarding obstacle detection and recognition in a marine environment, and reached a mAP@0.5 of 95.43% on the ShippingLab validation data using an integration time window of $1000ms$. This is roughly equivalent to a drop in performance of 1.6% from the original model.

These results are on pair with the work performed in [36], the only previous attempt at fabricating a deep spiking object detector, which was run using the *TrueNorth* neuromorphic chip. That experiment was also performed using an ANN-to-SNN conversion, reaching a performance drop of 2% in $8000ms$ on the more complex MS COCO dataset. A fair comparison, though, could not be performed, as the evaluations were computed using different datasets and the lack of a neuromorphic platform made it impossible for us to run a proper evaluation on the COCO dataset due to its huge computational cost within a simulation framework.

The results obtained are a remarkable feat on their own, due to the efforts of research and enhancement of existing SNN frameworks which were required, as these were designed for shallow neural network structures and classification tasks. Aside from its imperfections, this work demonstrates the potential and scalability of the conversion algorithm to deep ODC models and could be a source of inspiration for future interpretations of the method.

CHAPTER 4

TFA-RetinaNet

This chapter corresponds to the second work package of the MSc Thesis project. The aim of this module is to investigate possible enhancements to the ShippingLab object detection and classification problem using novel meta-learning techniques.

The outset for the chapter is a review and discussion around possible approaches to the enhancement of the ODC problem based on existing meta-learning studies. One of these methods will be chosen, setting the path for further analysis, implementation and testing on the ShippingLab data.

4.1 Meta-Learning overview

One of the fastest-growing areas of investigation in machine-learing is the field of meta-learning. Meta-learning, in this context, is most commonly understood as *learning to learn*, which refers to the process of improving a learning algorithm over multiple learning episodes in terms of generalization, performance or learning speed. This contrasts with conventional machine-learning, which focuses on the process of improving model prediction performances over multiple data instances.

The current most common approach to machine learning problems is to train the model from scratch for a specific task using a hand-crafted, fixed learning algorithm. This approach has been successful at the cost of huge computational resources and vast quantities of data in the cases where both of these are available.

Contemporary neural-network meta-learning constitutes an alternative paradigm, whose salient characteristic is an explicitly defined *meta-level objective*, and end-to-end optimization of the inner algorithm with respect to this objective. Often, meta-learning is conducted on learning episodes sampled from a a collection of related tasks, leading to a base learning algorithm that is tuned to improve its future learning performance on new tasks sampled from this task family. This 'learning-to-learn' entails benefits such as data and computational efficiency, and is better aligned with how this activity is present in biology [44].

A wide variety of perspectives on meta-learning can be found in the literature. As different sources may refer to the term somewhat discordantly it can be difficult to narrow down. Yet, taken broadly, the term meta-learning can include fields like automatic algorithm tailoring, transfer learning, multi-tasking, feature-selection and model-ensemble learning.

The use of meta-learning techniques obeys to the desire of improving the flexibility and performance of AI and broaden its applicability to a wider problem spectrum, aiming to eliminate the need of a costly retraining for each specific scenario. Successful applications can be spotted in areas like few-shot learning, unsupervised learning, self-directed reinforcement learning, hyperparameter optimization and neural architecture search [30].

For more information about previous research in meta-learning and related topics, please refer to section 2.4.

4.2 Applications to object detection and classification

The embracing of meta-learning approaches to target the object detection and classification issue is a very recent advent, which started just a couple years ago. A broad scanning will be performed in this section for possible improvements that could be achieved by these means and latest findings in each of the study lines. An evaluation of the interest and trade-offs will be performed for each approach and, in the end, one of them will be chosen for a software implementation and testing, following the ShippingLab objectives and criteria.

Current limitations to the applicability of object detection algorithms are related, for the most part, to their need of vast quantities of labelled data and computational power. This makes them inaccessible for the general public due to the cost of producing a big-enough labelled dataset and the investment needed to acquire and run powerful hardware. These reasons explain why, up to this date, all reported meta-learning applications to the ODC problem pursuit a reduction in the amount of data needed for successive realizations of an object detector in different scenarios. Addressing these limitations entails an increase in efficiency for these algorithms and applicability to a wider range of problems. As to the previous description, most meta-learning approaches to the ODC problem can be classified as variations on the few-shot learning paradigm (previously introduced in section 2.4). This concept, though, allows for multiple interpretations, some of which will be briefly explained now.

4.2.1 Incremental learning

The class-incremental object detection aims to address the phenomenon known as *catastrophic forgetting*, which commonly takes place when an already trained model is retrained in a dataset comprised only of instances of unseen classes. An abrupt degradation of performance is then provoked on the original set of classes, as traditional loss functions do not penalise rewriting of existing parameters. Incremental learning methods allow for a progressive increase in the applicability of the network and number of classes it is able to detect, and can either be achieved online or by fine-tuning on a new, smaller dataset.

Said issue is typically addressed using *distillation* methods, intended to minimize the discrepancies between responses to old classes from the original and the updated models. An example for this is the work in 2017 by Shmelkov *et al.* [79], where this is performed through a novel loss function which balances the interplay between old and new class predictions. These *distillation* methods, though, have as a downside that they enforce some intransigence in the training procedure, making it harder to learn novel classes. Joseph *et al.* [32] suggested a fix for this limitation in 2020, by using a meta-learning approach that learned to reshape model gradient directions towards an optimal sharing of information across incremental tasks (*optimal plasticity*).

Another trend in this area are the real-time incremental ODC methods, whose aim is to increase the label pool of the network during its regular operation. The 2019 paper from Li *et al.* [45] is an example of this approach, suggesting the use of automatic image annotation and cloud computing on a handheld device to incrementally train the model on datasets generated from taken pictures of target objects. This is performed by means of image-based web browsing technologies in a remote server.

It is also worth mentioning the research paper of Sharma *et al.* in 2012, which applied this philosophy instead to increase the performance of an offline trained detector on crowded and challenging samples. This is achieved through the use of unsupervised multiple instance learning (MIL) incremental solutions, which include a MIL loss function and a tracking-based unsupervised on-line sample collector for the incremental learning.

4.2.2 Few-shot learning

The few-shot object detection (FSOD) problem aims to learn to effectively localize and classify objects from a single reference or a set of a few image samples. The general approach to this concept relays on a model which has been trained in a general *base* dataset which grants it with enough abstraction to easily recognise new unseen objects.

The usual setting is comprised of the abovementioned dataset of abundant *base* classes and a set of *novel* classes that has K instances per category [33]. The tailoring of these two sets generally plays a major role in the performance of the trained model, which is measured as its average precision (AP) on both the novel and base classes and evaluated on a balanced test set of all the known classes.

The first few-shot object detector was allegedly documented in 2019 by Kang *et al.* [33]. Their proposal combined a standard single-stage detector with an auxiliary network responsible of reweighting the feature extractor outputs. The goal of this reweighting was to give more importance to features directly related to the specific detection task, and its training was performed in a multi-episodic fashion.

A number of the most recent FSOD methods have been studied following a ranked listing over a mAP benchmarking on MS COCO dataset, performed by the *Papers with Code* machine-learning research archive [58]. Among the catalogued methods it was judged necessary to highlight the methods listed in table 4.1.

Table 4.1: Ranking of few-shot object detection (FSOD) methods as in [58] as per Feb. 2021. Score corresponds to mAP@0.5 on MS COCO.

	Method	Source	Score	Description
1	*FSDet View*	Xiao, Y. & Marlet, R [97]	12.5	In addition to FSOD it inferes the orientation of the detected objects using 3D data. Exploits a joint feature-embedding module to improve feature sharing from base classes. Top performer on the ranking.
2	*Att-RPN*	Qi Fan *et al.* [22]	11.1	Performs novel object detection on-the-go with the help of a support image of the target class while suppressing false detections in the background. Builds a base dataset specifically designed for FSOD.
3	*TFA*	Xin Wang *et al.* [94]	10.0	Hypothesises that the feature detector components are class-agnostic and thus can be frozen during the few-shot fine tuning, which then focuses in the last layers of the model. Builds new benchmarks for FSOD.
4	*MPSR*	Jiaxi Wu *et al.* [95]	9.8	Addresses the problem of scale variations between datasets. Performs a previous resizing of the input data in a feature-pyramid fashion to target a much wider range of object scales during training and detection.

4.2.3 Chosen approach

Following the previous study on meta-learning applications to the ODC problem, an argumented selection has been made for the method that will be tested in the ShippingLab scenario.

The chosen path leans towards a pure few-shot learning approach, as there is not a real need for incrementing the class range in the ShippingLab problem as per today. However, testing of new neural network structures is performed on a regular basis, as well as re-training of models for different applications. Reducing the training time of each new learning episode would have a very big impact and thus, it is the path this project takes.

Between the listed state-of-the-art few-shot ODC techniques in table 4.1, the chosen method to further investigate is TFA [94], a fine-tuning procedure which ranked 3rd on the *Papers with Code* benchmarking, only behind *FSDetView* and *Att-RPN*, and stands out for its simplicity. This approach outperforms other, more complex algorithms and is more suitable for the characteristics of our scenario than the two higher-rated techniques. These, in addition to their higher intricacy, target objectives which diverge from this project's, such as the 3D viewpoint estimation of the objects.

Having said that, from previous studies it is evident that the state-of-the-art performances of all few-shot approaches are currently way below the requirements of the ShippingLab project. This will be addressed in the following sections and possible workarounds and trade-offs will be contemplated.

4.3 TFA Method

The *Two-stage fine tuning approach* (TFA) [94] is built using the Faster R-CNN, a two-stage object detector, and its key component is to separate the feature representation learning and the box regression and classification learning into two stages. This technique has its focus on the tailoring of training data while using transfer-learning as a kick-off.

The method relays on the hypothesis that the features generated on the feature extractor components of the network are class agnostic, only responsible of detecting key elements in the image. Therefore, features learned from the base classes are capable of transfering to the unseen classes without further parameter tuning. This is potentiated in the base training by the use of a very diversified dataset.

Stage I: Base training. In this stage the entire object detector, including both the feature extractor and the object predictor, is trained only on the base classes.

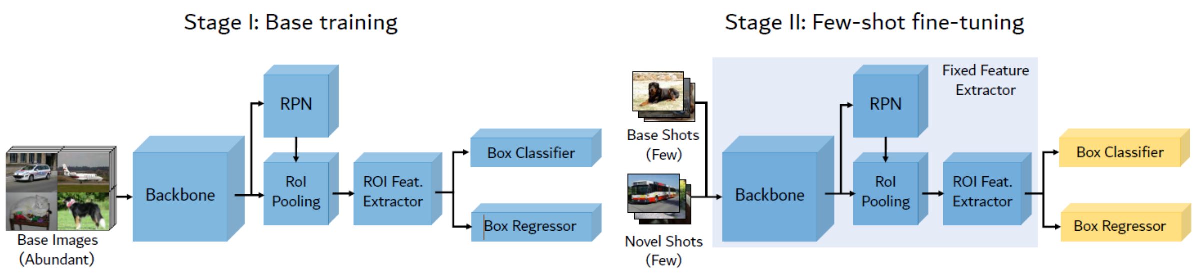


Figure 4.1: Illustration of the TFA method. In stage I, the full network is trained on the base classes. Then, in stage II, the feature extractor components are fixed and only the box regression and classification subnetworks are fine-tuned on a balanced subset consisting of both the base and novel classes [94].

Stage II: Few-Shot fine-tuning. A small balanced set of K-shots per label is created with both the base and unseen classes. The weights of the feature extractor are then frozen and random initial weights are assigned for the novel classes to the box predictor subnetworks. A smaller learning rate is used for this phase to prevent forgetting and overfitting, approximately 20 times smaller than in the first stage. It is also possible in this stage to use a classifier based on cosine similarity reducing the intra-class variance and improving the detection accuracy of novel classes with less decrease in base-class performance.

4.4 Implementation

The version of TFA applied in this project sticks to the already familiar RetinaNet architecture having as base classes the COCO dataset collection. As the starting model is already pre-trained (see 3.4.5.1), the only data and time needed will be the ones the few-shot training requires. This makes explicitly clear the potential savings this kind of approach, if effective, could deliver.

4.4.1 Dataset tailoring

The key component for the success of TFA is, in essence, the data selection used in both the training and the fine-tuning. The design of both datasets will now be explained together with the implications they may have towards the final results of the method.

Base dataset. As noted above, the data used for the base training of the network belongs to the Microsoft COCO dataset [48]. This collection delivers a range of 80 different classes in a total of 118287 training samples and was estimated as a good source of abstraction to the model's feature detection components. It is undermined, however, by the relative uniformity in the sizes of framed objects throughout the images. This is likely to impact the final performance of the model on the ShippingLab data, as this does contain items of very diverse sizes.

Few-shot dataset. In contrast to the usual few-shot settings, this application does not require the use of any of the base classes. The few-shot data will only include instances of the two new 'boat' and 'buoy' classes and the *catastrophical forgetting* [37] will not be an issue to be dealt with. The set has been designed with the objectives of the project in mind, building a collection that summarises all possible events of a real case scenario while achieving an even distribution of the data for optimal learning.

Current benchmarks for the few-shot ODC method use 10-shot learning to compute the scored predictions, reaching results that, despite being promising, are way below the standards required for this application. This will be addressed by the use of a bigger few-shot dataset that can potentially train a more reliable network while keeping a sufficiently low training time.

In the Spiking-RetinaNet work package, section 3.4.4.2, the normalization performed on the RetinaNet model used an estimate of the distribution of all possible activation values for each channel in the network. This estimate was computed over aprox. 10% of the data, sampled randomly, which turned to be a significant portion and led to good predictions of the normalized model. This comes to show that, just with this subset, all relevant neurons in the network were activated across their full range and all extra samples may be considered redundant.

The hipothesis behind the building of the few-shot set is then that using a random 10% of the data should lead to good performances. However, a smaller set of well-chosen images is also likely to pull it off. The resulting dataset has an amount of 300 pictures and its characteristics are displayed in figure 4.2. An even representation of both classes has been fully achieved, as well as the presence wide-ranging sized items in every relevant region of the images. This is roughly a 3.6% of the training dataset and is estimated to lead to a time and energy saving of 95% when compared to the full training process.

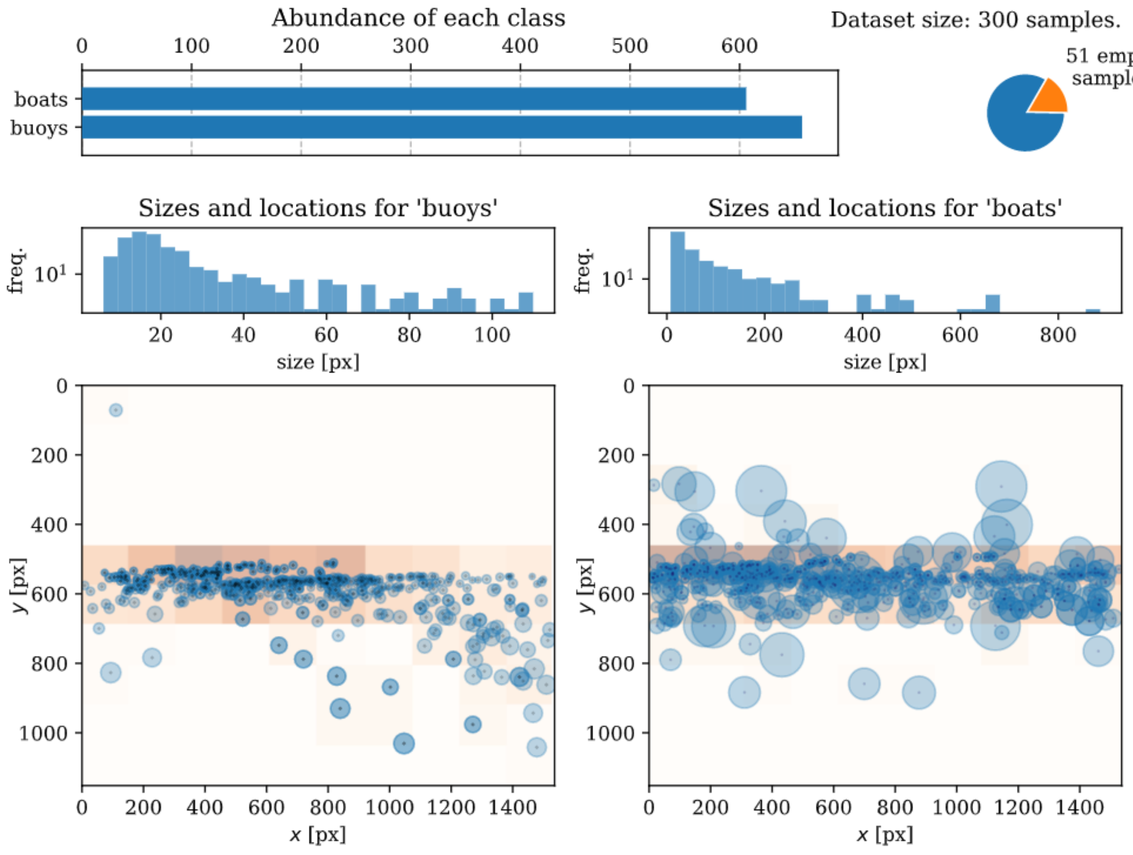


Figure 4.2: Monitored parameters for the building of the few-shot subset. The design took into account the significant presence and even distribution of items of all classes, sizes and locations. In the bottom graphs, the sizes of the blue circles are proportional to the area of the object in that spot, while the background orange color scale references the distribution of these in the image.

4.4.2 Model adaptation

The RetinaNet model used for the few-shot training is a child of the one used in the COCO base training. As displayed in figure 4.3, it has been generated by swapping the classification subnetwork by a new one with the right layer dimensions for the new number of classes and randomly initialized weights. The parameters of the feature pyramid network (FPN) and the box regression head have both been transferred from the base model, as they are both theoretically class agnostic. Additionally, FPN has been frozen, its weights will be preserved from the base training as explained in section 4.3.

As the model adaptation performed here is a more advanced version of what was done in section 3.4.5.2, the same issues regarding data encoding and bounding box shapes were equally addressed.

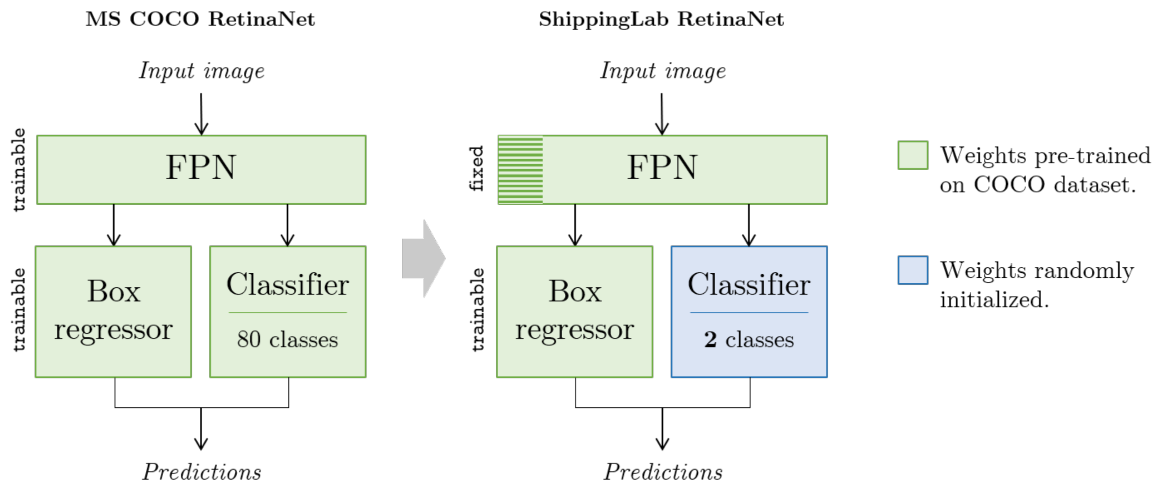


Figure 4.3: RetinaNet model adaptation for the training in the ShippingLab few-shot subset. Transfer-learning has been performed in the FPN and box regressor. The classifier, on the other hand, has been rebuilt from scratch for the *buoy*/*boat* labels.

4.4.3 Training details

The few-shot training was performed in the DTU Computing Centre on a total of 115 epochs taking an average of 101s per epoch. The learning rate was scheduled, using a piecewise constant decay function, to descend gradually from the minimum base-training's learning rate to a value 15 times smaller, an amount similar to what was used in [94]. This has been done in an attempt to speed up the training while minimizing overfitting in latter stages. The full training time was of 3.5h, ten times shorter than the full training performed in section 3.4.5.2, which also benefited from COCO transfer-learning.

4.5 Results

The trained model displays a mAP@0.5 of 61.5% as shown in figure 4.3. The performance shown is much higher than that shown by previous applications due, firstly, to the larger few-shot data pool and, secondly, to the fact that the ShippingLab data has only two different classes, being the scores displayed in table 4.1 computed on the non-trivial 80 classes of the MS COCO set. These results, though, are not sufficient to support the application of this method to a real case-scenario, as the accuracy of an object detector used for a vehicle needs to be as close to 100% as possible. After further investigations on the training data, two potential issues have been identified that could be deteriorating the score obtained on the ShippingLab validation subset.

The first issue is related to the anatomical difference between the base COCO dataset and the few-shot ShippingLab subset, referred to in sections 3.4.5.2 and 4.4.2. This is, that the anchor box size collections needed to fit the objects in each of both datasets are different. This big difference in object size is likely to be limiting the ability to apply the ground hypothesis of 4.3 and achieve good results with the COCO-trained feature extractor components, simply because the base dataset does not have enough diversity.

In a similar fashion, the second issue is found in the size correlations between the few-shot and the validation datasets. While the few-shot samples were hand-picked to obtain an even distribution across classes, locations and sizes of objects; the validation data is randomly picked, being >90% of the class instances comprised by small objects. This essentially means that the few-shot training objective is not aligned with what the validation set rewards the most, which is good performance on small objects. Scores would improve by using different few-shot samples, which better fit the nature of the validation set. This statement is backed-up by a small experiment, explained in appendix C.

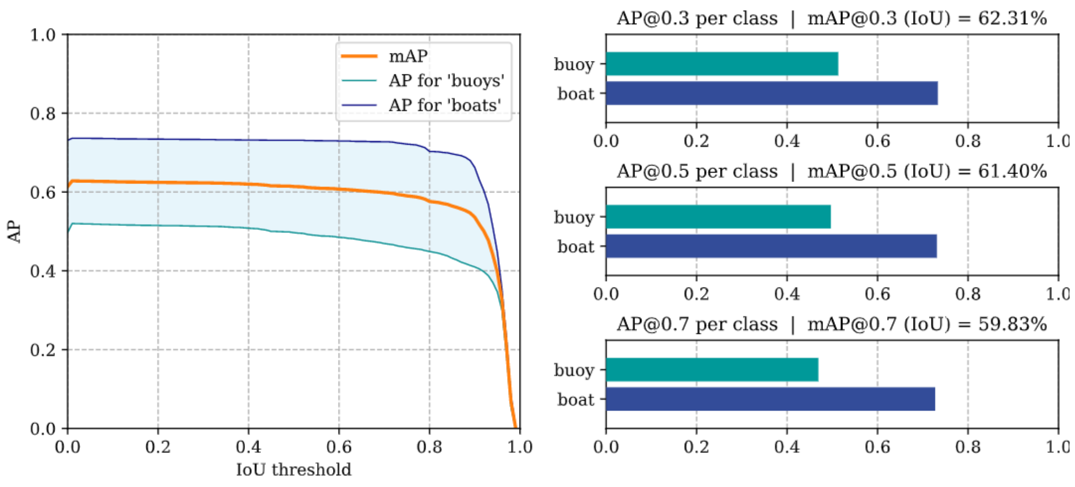


Figure 4.4: Mean Average Precision metric of the few-shot trained model on the validation set calculated for different IoU box-matching thresholds.

4.6 Summary

After a broad investigation on the applicability of meta-learning to object detection and classification models, a few-shot learning approach was chosen to address the huge data consumption and long training times of these kinds of algorithms.

The TFA method [94], while ranking 3rd in the 10-shot COCO benchmark, stood out from the rest for its simplicity and straightaway applicability to the project, aligning perfectly with the ShippingLab objectives. The method was adapted to the problem by cutting down the number of classes to just the two, *boat* and *buoy* labels, and by increasing the size of the few-shot dataset, aiming for better results than the ones reported in the original research paper.

While this certainly was achieved, the obtained scores remained too poor for its application to autonomous navigation, and an investigation was performed for possible causes. Some structural differences between the base, few-shot, and validation datasets were detected and it is believed that, if these were addressed, a significant improvement in the scores would be achieved.

CHAPTER 5

Discussion

In this chapter, the main work-threads followed throughout the project will be analyzed in retrospective. A reflection will be performed on the main technological findings and limitations, and overall results will be compared with the objectives set in the Introduction (section 1.2). Finally, possible study-lines for the future will be suggested.

5.1 Overall results analysis

As stated in the Introduction, the ambition motivating the project is the exploration of alternative, avant-garde approaches to the implementation of the RetinaNet deep learning model to the environmental awareness of vehicles in the sea. A suitable way to kick-off our work's final breakdown is to look back to the initial hypothesis made, which motivated the chosen paths. Figure 5.1 shows the initial concept, which intertwines both work packages into a coherent mechanism to augment a deep-learning neural network and permit its implementation in neuromorphic frameworks. Nevertheless, this has never fully come to life in the project and both work-lines have been kept independent.

Regarding the use of meta-learning to enhance the training of the deep-learning model, the use of a few-shot learning allowed for a reduction of 90% in the training time and 95% in the data consumption of the model without leading to overfitting. This, on the flip side, did not achieve a sufficient performance to support a real application to the ShippingLab problem, stalling at a 63.3% of the score achieved through a full-training (section 4.5). Having said that, the differences in the bounding box size ranges between the base training, few-shot training and validation datasets have been identified to be negatively biasing this result.

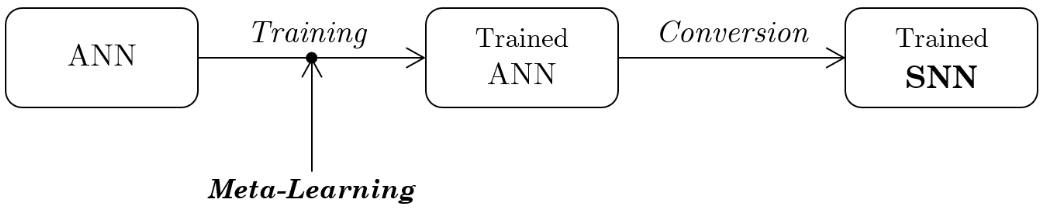


Figure 5.1: Scheme showing the interlocked applications of both work modules. Corresponds to *Path 2* in figure 1.1: Training of deep (analog) neural network for latter porting to a spiking architecture.

We find especially significant to comment on the over-represented small sizes in the validation set, which contrast with the more evenly distributed emphasis in the few-shot data. Although this would be normal in a real scenario, the scores given by this set reward detection of small, faraway targets over big, close obstacles, something that was purposefully addressed in the construction of the few-shot collection. We suggest that this dataset should be modified to properly evaluate the skills of object detectors targeting real sea maneuvers. A new metric could also be implemented with this in mind, weighting the scores obtained with the sizes or proximities of the ground-truth objects, however, this could be dangerous towards the detections of smaller bodies (e.g: kayaks or small buoys).

Another possible enhancement to the few-shot training would be the use of *active-bias* [11] or *importance sampling* [34] techniques during the training, to dynamically increase the emphasis given to examples where the model performs the worst during the learning phase. This would lead to a more meaningful training and perfectly harmonize with the few-shot approach.

On the other hand, the instantiation of the RetinaNet model in a spiking fashion led to more sophisticated results, mainly due to the much greater resource allocation to this study line. In spite of the absence of a neuromorphic hardware implementation and the subsequent handicap of not being able to properly test the Spiking-RetinaNet model, the results obtained in this work module could be considered state-of-the-art, as they may rival with those offered by Spiking-YOLO [36], the only previous attempt to creating a deep spiking object detector. This Spiking-YOLO network, when implemented in a neuromorphic chip, consumed 280 times less energy than its equivalent ANN running on a regular GPU.

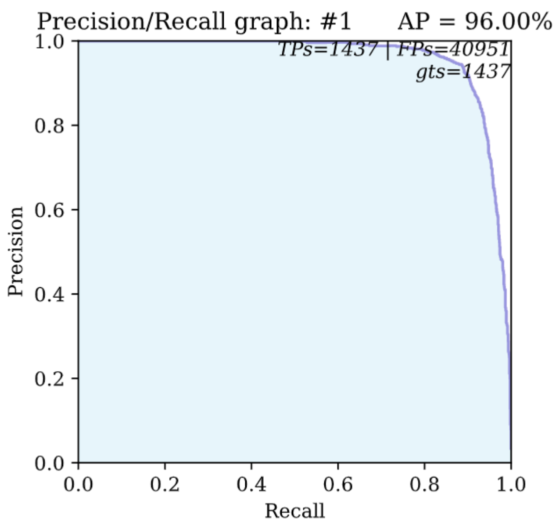
The Spiking-RetinaNet model was generated through ANN-to-SNN conversion, which was one of three main open fronts through the span of the MSc Thesis, being the other two the trainings of a binary activated ANN and of a real SNN. The findings and conclusions retrieved have been summarized in table 5.1 for a better intelligibility.

Placing this object detector into the marine context, in section 1.4.2 of the Introduction the recall metric was addressed as specifically critical, due to the need of detecting every single obstacles for the vessel to perform safe maneuvers in the sea. The original Keras RetinaNet scored a 100% on the *buoy* class and a 99.5% for the *boat* class over the full validation dataset (figure 5.2). On the 20-sample set for the SNN validation, which contained 65 buoys and 30 boats, the ANN scored 100% in both classes. Meanwhile, the $1000ms$ SNN achieved a perfect score in buoy detection, but failed in one boat instance (96.6%). This could be caused by the shift channel normalization 'matrix frame error', as explained in page 44. This phenomenon should be further investigated before the use of this method towards a real application.

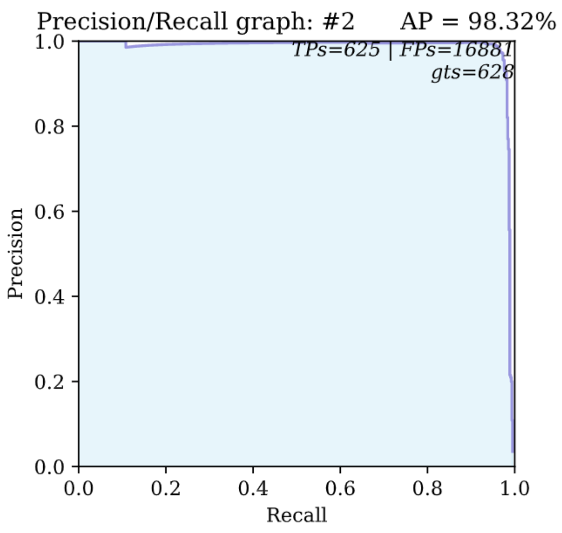
Finally, it is also necessary to comment on the neuromorphic hardware implementation of the Spiking-RetinaNet model. Although in [36] the Spiking-YOLO network was put to work in the *TrueNorth* platform, this model was based on Tiny-YOLO, a smaller network. It is yet to be seen how costly could this turn out to be for the larger and more complex Spiking-RetinaNet.

Table 5.1: Brief analysis of all the tested approaches towards a spiking-RetinaNet model.

	Advantages	Drawbacks
Whetstone	• Model conformed by static, binary-activated neurons. Extremely energy efficient.	• Bad results on the ODC problem due to the limitations of static, binary activations ($mAP@0.5 < 20\%$ in MS COCO).
	• Intuitive training method, built upon the well-known backpropagation algorithm.	• Lacks the potential benefits of event-based encoding.
	• Compatible with both conventional deep-learning frameworks and neuromorphic platforms.	• Limited applicability to big neural networks.
SNN Training	• Possibility to use bio-plausible learning rules and exploit SNN unique characteristics.	• SNN learning algorithms <u>not ready</u> for such a big neural network.
	• Possibility of reaching better performances than ANNs by tailoring the learning data for an event-based logic.	• Obtaining the needed precise-timing data is costly. Need of a change of paradigm.
	• Can be fully executed in a neuromorphic platform.	• Computationally expensive if not performed on neuromorphic hardware.
ANN-to-SNN conversion	• Able to use widely-known, reliable ANN learning methods and benefit from their wide research and documentation.	• Constrained by original ANN model. Unable to exploit bio-plausible learning rules and time information.
	• Able to reach state-of-the-art performances on non-trivial datasets. $mAP@0.5(1s) \sim 95\%$ in ShippingLab data. $mAP@0.5(1.5s) \sim 50\%$ MS COCO.	• Ceiling for performance set by the original ANN model.
	• Converted models can be deployed in a neuromorphic platform, where they Would benefit from low energy consumption, fast inference times and distributed computation.	• Slow, computationally inefficient execution within the INI/Keras simulation environment.
		• The use of a r-SNN entails the need of high spiking rates and relatively big time windows to reach accurate results.



(a) Scores for *buoys* (#1).
Recall = 100%



(b) Scores for *boats* (#2).
Recall = 99.5%

Figure 5.2: Precision/Recall graph of RetinaNet on the ShippingLab validation data.

5.2 Fulfilment of the objectives

The addressing through the project of the stated objectives in section 1.2 has been evaluated by means of table 5.2. The *State* field can be either ✓(fulfilled) or ✗ (missed).

All in all, the MSc thesis project managed to realize all its objectives except the hardware implementation of the algorithms, due to lack of time and resources. This was foreseen during the third month of project work and, during a meeting, all parts agreed to set this target appart in favour of dedicating more time to the design of an SNN framework for the ODC problem.

Table 5.2: Completion of the objectives set at the start of the MSc project.

#	Objective	State	Outcomes	Location
Main objectives				
1	Perform a literature study of specific topics relevant to the thesis work.	✓	Wide-ranging theory review on ODC, SNNs, meta-learning and related topics.	Sections 2, 3.2, 3.3.1, 4.1 and 4.2.

2	Investigate possible approaches to spiking architectures and neuromorphic computing with the focus on deep-learning models for computer vision.	✓	Three studied approaches. Summarized in table 5.1.	Sections 3.3, 3.4 and 3.5.
3	Develop and implement RetinaNet for detection and classification of ships and buoys exploiting an SNN architecture, and compare performances w.r.t. an ANN architecture.	✓	Spiking-RetinaNet model via ANN-to-SNN conversion [72].	Section 3.4.
4	Investigate meta learning methods for ODC and identify one or more approaches that could be relevant for the autonomous ship navigation.	✓	Study on few-shot ODC and choosing of the TFA method [94].	Section 4.2.
5	Develop and implement at least one meta learning method for ODC.	✓	TFA-RetinaNet	Sections 4.3 and 4.4.
6	Test and validate the implemented meta learning solution on relevant data sets and compare findings with the SNN-RetinaNet approach.	✓	Benchmark using dataset size, training time and mAP as metrics.	Sections 4.5 and 5.
7	Draw conclusions about the potential of using SNN and meta learning for robust ODC in autonomous ship navigation.	✓	As reasoned before. These solutions are certainly promising but still in early phases of development.	Sections 3.6 and 4.6, 5.
Side objectives				
1	Further development of existing spiking neural network frameworks targeting the ODC problem.	✓	Adaptation of the `SNN_Toolbox`[81] and other libraries to the ODC problem.	Section 3.4.4.
2	Investigate potential improvements to the studied technologies towards the objectives of the ShippingLab project.	✓	Study of limitations in the project's results and how to address them.	Sections 3.4.6, 4.5 and 5.
3	Implementation and testing of the developed solutions in dedicated hardware.	✗	–	–

5.3 Future work

This section is a natural follow-up to the brainstorming made addressing side-objective 2, in sections 3.4.6, 4.5 and 5. Several limitations were found in the results obtained by both Spiking-RetinaNet and TFA-RetinaNet and possible work paths to address them could be:

- Attempting the implementation of spiking-RetinaNet in a neuromorphic platform to benefit from its advantages.
- Use *time-to-first-spike* encoding to reach a lower energy footprint of the SNNs.
- Further inverstigate the issues regarding the shifted channel normalization method.
- Collect event-based data for the training of the spiking ODC models on neuromorphic hardware using bio-plausible learning rules.
- Perform a proper mAP benchmark of the spiking-RetinaNet on relevant datasets.
- Use of importance sampling to enhance the few-shot training.
- Automatize the generation of the few-shot dataset using AI techniques.

Additionally, to be able to process the *RetinaNet* architecture and perform its conversion to a spiking model, the parsing stage had to be performed in a semi-automatic fashion. Manual input is required to correcly perform the connections between subnetworks and, during this project, we were unable to find a suitable way to fully automate them. This would be a big improvement for the conversion toolbox.

Finally, for the Spiking-YOLO model in [36], the standard IF neuron model was substituted by a novel signed neuron with imbalanced threshold to address the issue of negative activations (*Leaky-ReLU w/IBT*). These neurons are fully implementable in neuromorphic platforms and using them could potentially eliminate the issues derived from the shift channel normalization algorithm.

5.4 Summary

To summarize, all main objectives of the MSc Thesis were properly fulfilled within the duration of the project along two different lines of study. These lines, when combined, could entail a big energy footprint reduction in ODC models and enable its application in neuromorphic frameworks.

The results obtained are state-of-the-art when compared to other contemporary research in [59] and [58], nevertheless, these are not good enough for use in a vehicle navigation algorithm. Due to safety reasons, the performance requirements in this kind of environment are extremely high. With this in mind, an analysis has been performed in the search for improvement pathways. Some of the most relevant would be the use of importance sampling [34] with the few-shot learning, and the hardware implementation of the SNN on a neuromorphic platform like *TrueNorth* [12].

CHAPTER 6
Conclusion

This paper constitutes an elaborate study on the applicability of neuromorphic computing and few-shot learning to object detection and classification deep-learning models. The particularity of these algorithms is their convolutional neural network (CNN) structure and their greater complexity, being integrated by more than 100 layers. This posed a big challenge throughout the project as both SNNs and meta-learning are in very embryonic stages and have very scarce research available towards object detection applications.

It was reckoned necessary for said reasons to take matters into our own hands and, over the course of the project, two lines of work were conducted towards *(a)* the reduction in the data and time consumption during training and *(b)* the generation of a SNN object detector. The results obtained by both study lines earn these methods a position in 2020's state-of-the-art, as they rival with those in other contemporary studies of the field.

The neuromorphic computing work module deserves a special mention, due to the needed efforts of research and enhancement of existing SNN frameworks, as these were designed for shallow neural network structures and classification tasks. Stemming from an existing library [81, 72], a novel framework has been developed for the conversion of object detection neural networks to spiking models, supporting subsequent simulation and possible deployment to neuromorphic hardware platforms.

The used technologies, however, do not reach the quality standards to be used for for environmental awareness of autonomous ships yet. It will be necessary for them to mature before new work is carried out towards this objective. Nevertheless, this work demonstrates the potential and scalability of these approaches. We hope it serves a source of inspiration for future investigations.

Given all the resource and time limitations and technological challenges provided by such an avant-garde field, the project manages to fulfill the proposed objectives and deliver tangible and justified results. We believe it is a step in the right direction towards more energy efficient artificial intelligence and the *Internet of Things (IoT)* paradigm, as both neuromorphic computation and meta-learning could open the door for low-power deep-learning methods.

Appendices

APPENDIX A

Spiking-RetinaNet predictions

A.1 Predictions on the ShippingLab data

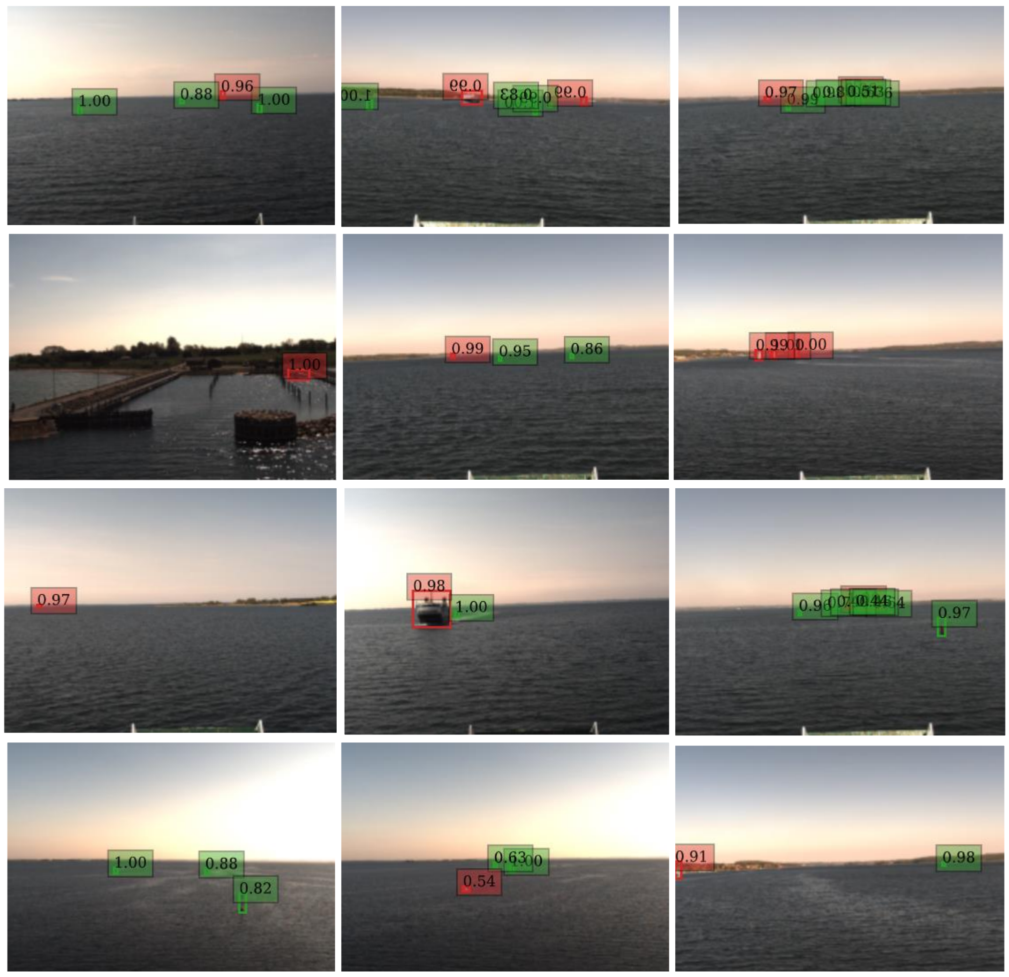


Figure A.1: Object detections performed by the spiking-RetinaNet model with a 1s integration time-window and disregarding of the transient. *Boats* are framed red and *buoys* green. Images belongs to the ShippingLab dataset.

A.2 Predictions on MS COCO

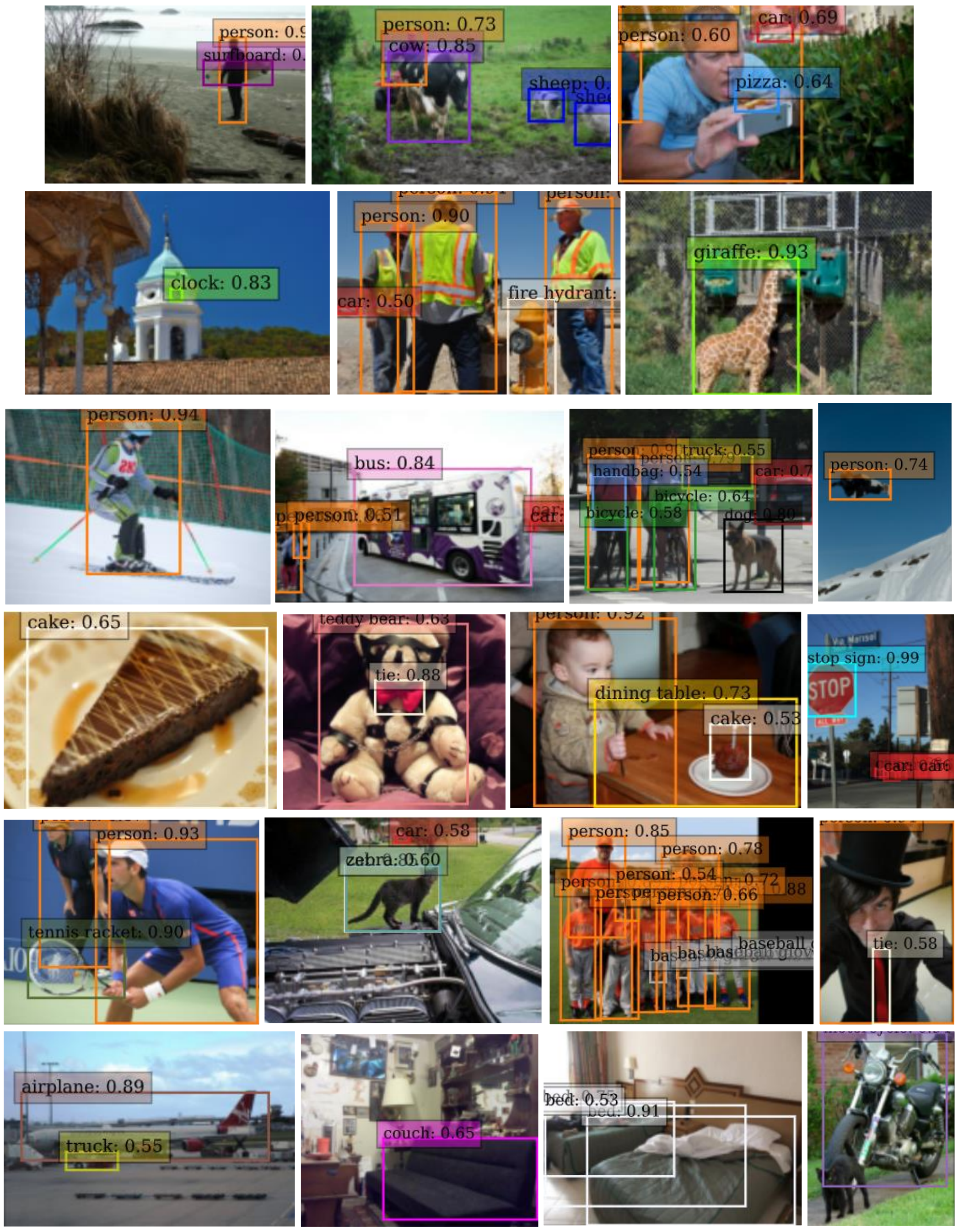


Figure A.2: Object detections performed by the spiking-RetinaNet model with a 2s integration time-window and disregarding of the transient. Images belongs to the MS COCO dataset [48].

APPENDIX B
Correlations

This section contains correlation plots similar to the ones displayed in table 3.7, but for layers 13 and 79 of the network.

Notice how the correlation is higher and the convergence faster in more shallow layers (figure B.1), while deeper layers take more time to reach acceptable results (figure B.2) due to the delay in the spike propagation. The error in these deeper layers is also higher, as it is transmitted by up-stream layers and accumulated.

These plots pertain to the simulation of sample 73 of the MS COCO validation set without ignoring the transient (that is why the simulation takes $3s$ to converge).

Table B.1: Correlations between analog activations and obtained spiking rates of layer `13Conv2D` after the conversion of RetinaNet.

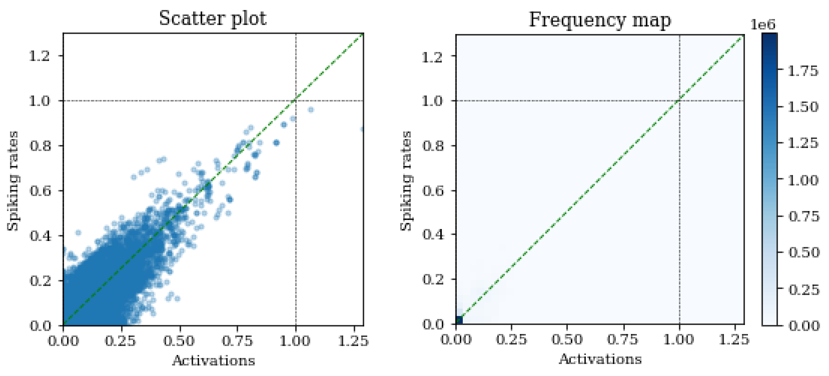
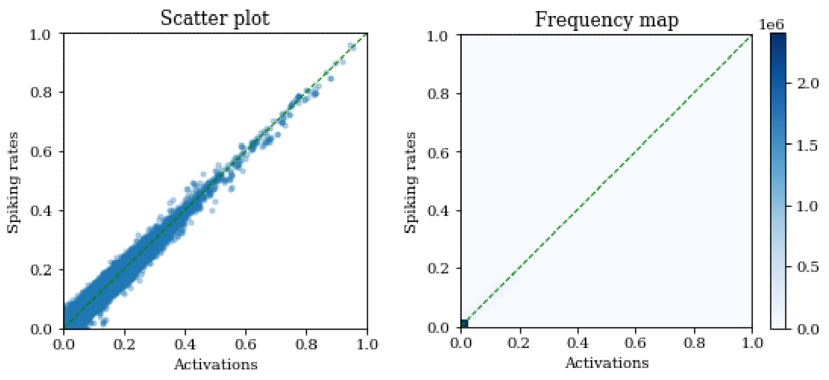
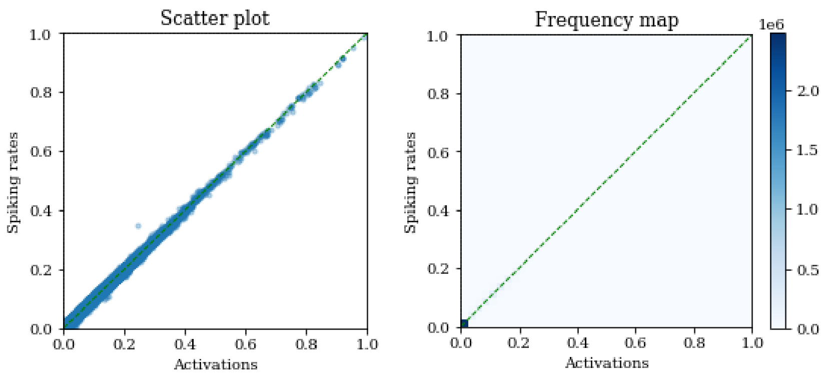
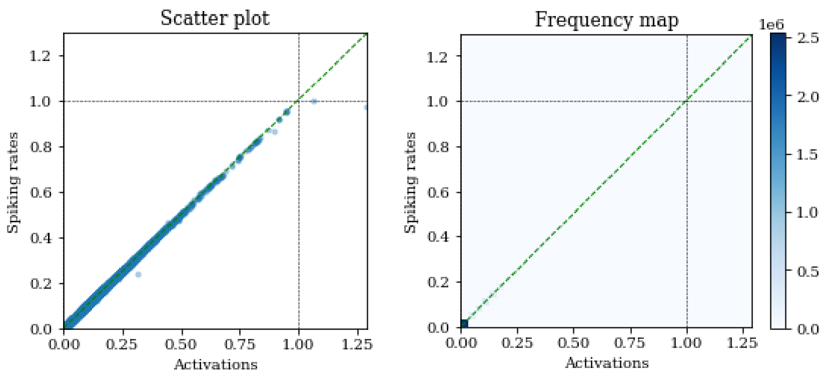
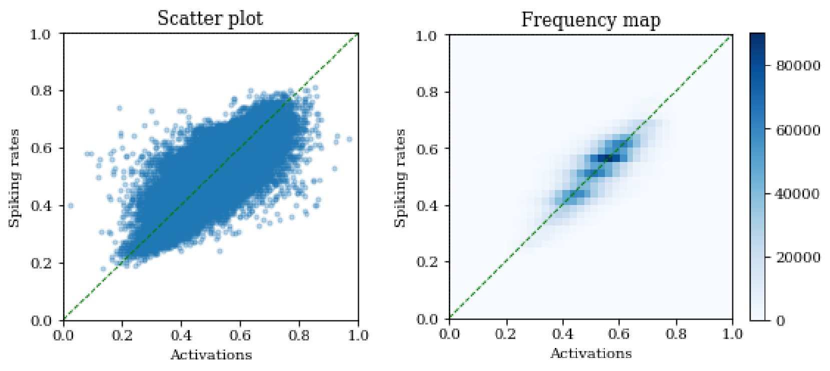
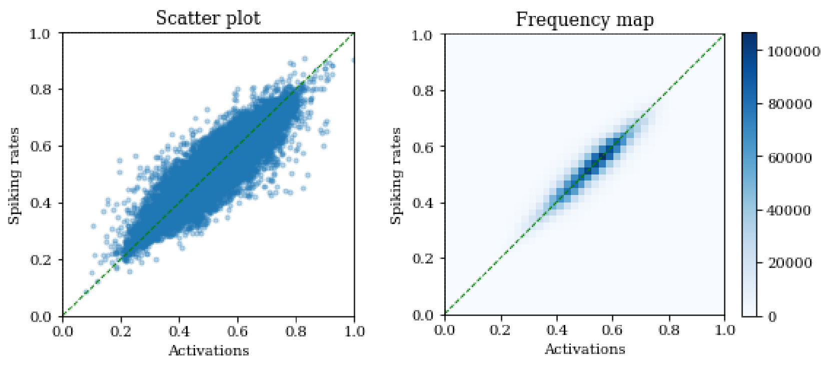
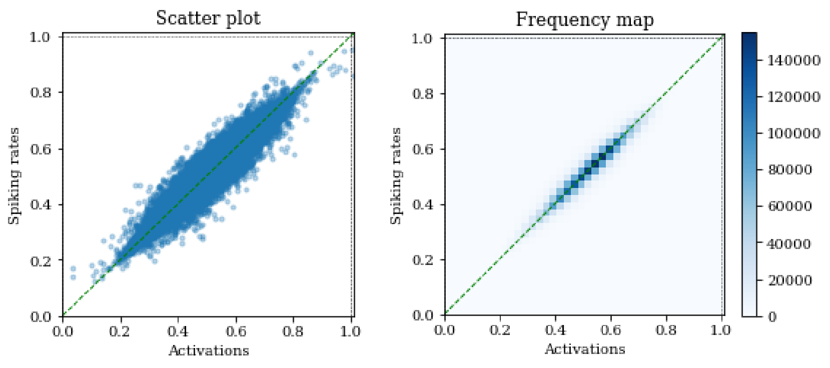
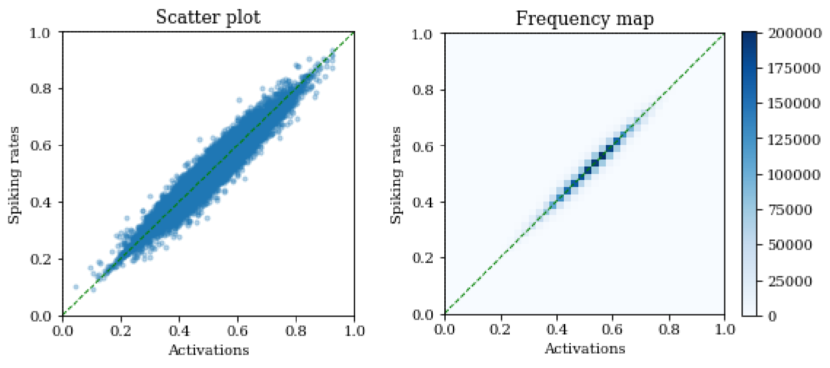
100 ms	
500 ms	
1500 ms	
3000 ms	

Table B.2: Correlations between analog activations and obtained spiking rates of layer `79Conv2D` after the conversion of RetinaNet.

100 ms	
500 ms	
1500 ms	
3000 ms	

APPENDIX C
ShippingLab validation set size issue

The study performed in this section aims to back the theory that the evaluations performed in the validation dataset are biased, rewarding more the detections of small objects. Figure C.2 shows a much higher representation of small objects in the dataset, which are focused around the horizon line. Additionally, figure C.1 compares the performance of the RetinaNet model when evaluated on the regular validation set and the few-shot set.

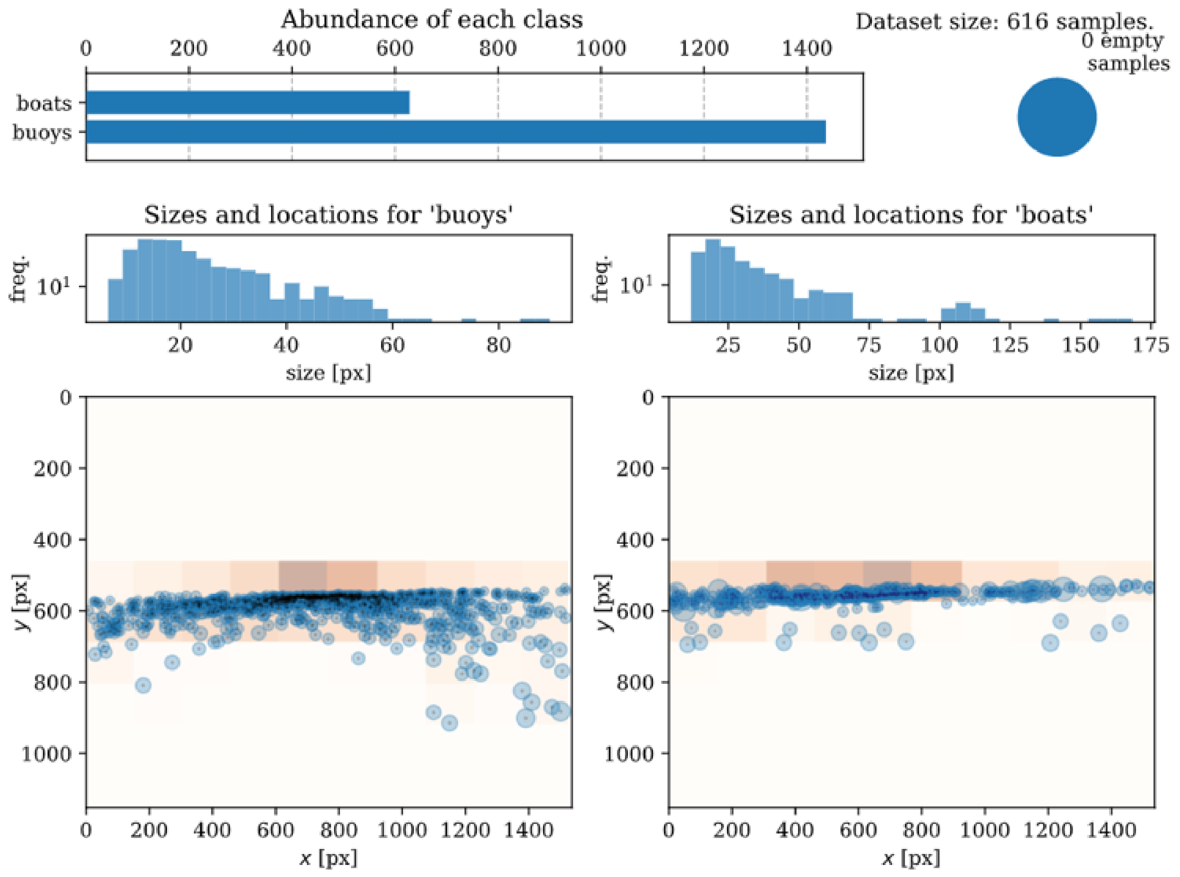


Figure C.1: Characteristics of the ShippingLab validation dataset. Notice the predominance of small objects in comparison to 4.2.

Notice how, despite the excellent mAP shown for the validation data, the score obtained for the few-shot set, which has a more even distribution of object sizes, is mediocre. This unveils that this model might not be doing well with bigger objects despite being very good at detecting smaller ones and that this problem goes undetected if evaluated only in the validation set.

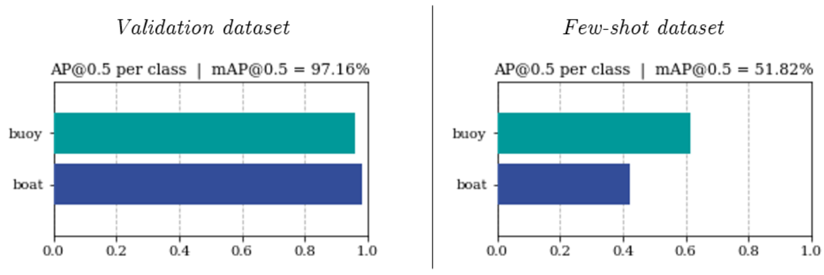


Figure C.2: Performance of the fully trained RetinaNet when evaluated on the validation (left) and few-shot (right) datasets. Notice the bad performance on the few-shot data despite the high score obtained in the validation set.

Bibliography

[1] LF Abbott and Thomas B Kepler. "Model neurons: from hodgkin-huxley to hopfield." In: *Statistical mechanics of neural networks*. Springer, 1990, pages 5–18.

[2] Arnon Amir et al. "A low power, fully event-based gesture recognition system." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pages 7243–7252.

[3] Danish Maritime Authority. *Blue Denmark*. URL: `https://www.dma.dk/Vaekst/MaritimErhvervspolitik/DetBlaaDanmark/Sider/default.aspx`.

[4] Samy Bengio, Yoshua Bengio, and Jocelyn Cloutier. "On the search for new learning rules for ANNs." In: *Neural Processing Letters* 2.4 (1995), pages 26–30.

[5] Ben Varkey Benjamin et al. "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations." In: *Proceedings of the IEEE* 102.5 (2014), pages 699–716.

[6] Etienne Bennequin. "Meta-learning algorithms for Few-Shot Computer Vision." In: *arXiv preprint arXiv:1909.13579* (2019).

[7] Jonathan Binas et al. "DDD17: End-to-end DAVIS driving dataset." In: *arXiv preprint arXiv:1711.01458* (2017).

[8] Zhenshan Bing et al. "A survey of robotics control based on learning-inspired spiking neural networks." In: *Frontiers in neurorobotics* 12 (2018), page 35.

[9] Fabio Boi et al. "A bidirectional brain-machine interface featuring a neuromorphic hardware decoder." In: *Frontiers in neuroscience* 10 (2016), page 563.

[10] Yongqiang Cao, Yang Chen, and Deepak Khosla. "Spiking deep convolutional neural networks for energy-efficient object recognition." In: *International Journal of Computer Vision* 113.1 (2015), pages 54–66.

[11] Haw-Shiuan Chang, Erik Learned-Miller, and Andrew McCallum. "Active bias: Training more accurate neural networks by emphasizing high variance samples." In: *arXiv preprint arXiv:1704.07433* (2017).

[12] Hsin-Pai Cheng et al. "Understanding the design of IBM neurosynaptic system and its tradeoffs: A user perspective." In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE. 2017, pages 139–144.

[13] *COCO/2017: TensorFlow Datasets*. URL: `https://www.tensorflow.org/datasets/catalog/coco#coco2017`.

[14] Jia Deng et al. “Imagenet: A large-scale hierarchical image database.” In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pages 248–255.

[15] Konstantinos G Derpanis. “The harris corner detector.” In: *York University* 2 (2004).

[16] Travis DeWolf et al. “A spiking neural model of adaptive arm control.” In: *Proceedings of the Royal Society B: Biological Sciences* 283.1843 (2016), page 20162134.

[17] Peter U Diehl et al. “Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing.” In: *2015 International joint conference on neural networks (IJCNN)*. ieee. 2015, pages 1–8.

[18] Daniel Drubach. *The brain explained*. Pearson, 2000.

[19] *DTU Computing Center website*. URL: `https://www.hpc.dtu.dk/`.

[20] Steven K Esser et al. “Convolutional networks for fast, energy-efficient neuromorphic computing.” In: *Proceedings of the national academy of sciences* 113.41 (2016), pages 11441–11446.

[21] Mark Everingham et al. *The PASCAL visual object classes homepage*. 2015.

[22] Qi Fan et al. *Few-Shot Object Detection with Attention-RPN and Multi-Relation Detector*. 2020. arXiv: `1908.01998 [cs.CV]`.

[23] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-agnostic meta-learning for fast adaptation of deep networks.” In: *International Conference on Machine Learning*. PMLR. 2017, pages 1126–1135.

[24] Steve B Furber et al. “The spinnaker project.” In: *Proceedings of the IEEE* 102.5 (2014), pages 652–665.

[25] Ross B. Girshick. “Fast R-CNN.” In: *CoRR* abs/1504.08083 (2015). arXiv: `1504.08083`. URL: `http://arxiv.org/abs/1504.08083`.

[26] Hananel Hazan et al. “BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python.” In: *Frontiers in Neuroinformatics* 12 (2018), page 89. ISSN: 1662-5196. DOI: `10.3389/fninf.2018.00089`. URL: `https://www.frontiersin.org/article/10.3389/fninf.2018.00089`.

[27] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[28] Geoffrey E Hinton and David C Plaut. “Using fast weights to deblur old memories.” In: *Proceedings of the ninth annual conference of the Cognitive Science Society*. 1987, pages 177–186.

[29] Berthold Horn, Berthold Klaus, and Paul Horn. *Robot vision*. MIT press, 1986.

[30] Timothy Hospedales et al. *Meta-Learning in Neural Networks: A Survey*. 2020. arXiv: `2004.05439 [cs.LG]`.

[31] *Human Brain Project: Silicon Brains*. URL: `https://www.humanbrainproject.eu/en/silicon-brains/`.

[32] KJ Joseph et al. "Incremental Object Detection via Meta-Learning." In: *arXiv preprint arXiv:2003.08798* (2020).

[33] Bingyi Kang et al. *Few-shot Object Detection via Feature Reweighting.* 2019. arXiv: 1812.01866 [cs.CV].

[34] Angelos Katharopoulos and François Fleuret. "Not All Samples Are Created Equal: Deep Learning with Importance Sampling." In: *CoRR* abs/1803.00942 (2018). arXiv: 1803.00942. URL: http://arxiv.org/abs/1803.00942.

[35] Nabeel Younus Khan, Brendan McCane, and Geoff Wyvill. "SIFT and SURF performance evaluation against various image deformations on benchmark dataset." In: *2011 International Conference on Digital Image Computing: Techniques and Applications.* IEEE. 2011, pages 501–506.

[36] Seijoon Kim et al. "Spiking-YOLO: Spiking Neural Network for Energy-Efficient Object Detection." In: (2019). arXiv: 1903.06530 [cs.CV].

[37] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks." In: *CoRR* abs/1612.00796 (2016). arXiv: 1612.00796. URL: http://arxiv.org/abs/1612.00796.

[38] Mikhail Kiselev. "Rate Coding vs. Temporal Coding – Is Optimum Between?" In: July 2016. DOI: 10.1109/IJCNN.2016.7727355.

[39] A. Krogh. "What are artificial neural networks?" English. In: *Nature biotechnology* 26.2 (2008). Cited By :156, pages 195–197. URL: www.scopus.com.

[40] Julius von Kügelgen. *On Artificial Spiking Neural Networks: Principles, Limitations and Potential.* June 2017.

[41] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. "The Omniglot challenge: a 3-year progress report." In: *Current Opinion in Behavioral Sciences* 29 (2019), pages 97–104.

[42] Hugo Larochelle, Dumitru Erhan, and Yoshua Bengio. "Zero-data learning of new tasks." In: *AAAI.* Volume 1. 2. 2008, page 3.

[43] Christiane Lemke, Marcin Budka, and Bogdan Gabrys. "Metalearning: a survey of trends and technologies." In: *Artificial Intelligence Review* DOI: 10.1007/s10462-013-9406-y (June 2013). DOI: 10.1007/s10462-013-9406-y.

[44] Christiane Lemke, Marcin Budka, and Bogdan Gabrys. "Metalearning: a survey of trends and technologies." In: *Artificial intelligence review* 44.1 (2015), pages 117–130.

[45] Dawei Li et al. "RILOD: near real-time incremental learning for object detection at the edge." In: *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing.* 2019, pages 113–126.

[46] Tsung-Yi Lin et al. "Feature pyramid networks for object detection." In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2017, pages 2117–2125.

[47] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection." In: *CoRR* abs/1708.02002 (2017). arXiv: 1708.02002. URL: http://arxiv.org/abs/1708.02002.

[48] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context.* 2015. arXiv: 1405.0312 [cs.CV].

[49] Tao Liu et al. "MT-Spike: A Multilayer Time-based Spiking Neuromorphic Architecture with Temporal Error Backpropagation." In: *CoRR* abs/1803.05117 (2018). arXiv: 1803.05117. URL: http://arxiv.org/abs/1803.05117.

[50] Jesus L Lobo et al. "Spiking neural networks and online learning: An overview and perspectives." In: *Neural Networks* 121 (2020), pages 88–100.

[51] Wolfgang Maass. "Networks of spiking neurons: The third generation of neural network models." In: *Neural Networks* 10.9 (1997), pages 1659–1671. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(97)00011-7. URL: https://www.sciencedirect.com/science/article/pii/S0893608097000117.

[52] D.B. Maudsley. *A Theory of Meta-learning and Principles of Facilitation : an Organismic Perspective.* Thesis (Ed.D.)–University of Toronto, 1979. URL: https://books.google.dk/books?id=nIiYmQEACAAJ.

[53] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5.4 (1943), pages 115–133.

[54] Hesham Mostafa. "Supervised learning based on temporal coding in spiking neural networks." In: *CoRR* abs/1606.08165 (2016). arXiv: 1606.08165. URL: http://arxiv.org/abs/1606.08165.

[55] Emre Ozgur Neftci, Hesham Mostafa, and Friedemann Zenke. "Surrogate Gradient Learning in Spiking Neural Networks." In: *CoRR* abs/1901.09948 (2019). arXiv: 1901.09948. URL: http://arxiv.org/abs/1901.09948.

[56] Daniel Nelson. *What is Meta-Learning?* August 2020. URL: https://www.unite.ai/what-is-meta-learning/.

[57] Garrick Orchard et al. "Converting static image datasets to spiking neuromorphic datasets using saccades." In: *Frontiers in neuroscience* 9 (2015), page 437.

[58] *Papers with Code - MS-COCO 10-shot Benchmark (Few-Shot Object Detection).* URL: https://paperswithcode.com/sota/few-shot-object-detection-on-ms-coco-10-shot.

[59] *Papers with Code - Object Detection on COCO test-dev benchmark.* URL: https://paperswithcode.com/sota/object-detection-on-coco.

[60] Seongsik Park et al. *T2FSNN: Deep Spiking Neural Networks with Time-to-first-spike Coding.* 2020. arXiv: 2003.11741 [cs.NE].

[61] Christian-Gernot Pehle and Jens Egholm Pedersen. *Norse - A deep learning library for spiking neural networks.* Version 0.0.5. Documentation: norse.github.io/norse/. January 2021. DOI: 10.5281/zenodo.4422025. URL: https://doi.org/10.5281/zenodo.4422025.

[62] José Antonio Pérez-Carrasco et al. "Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward ConvNets." In: *IEEE transactions on pattern analysis and machine intelligence* 35.11 (2013), pages 2706–2719.

[63] Michael Pfeiffer and Thomas Pfeil. "Deep Learning With Spiking Neurons: Opportunities and Challenges." In: *Frontiers in Neuroscience* 12 (2018), page 774. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00774. URL: https://www.frontiersin.org/article/10.3389/fnins.2018.00774.

[64] Filip Ponulak and Andrzej Kasiński. "ReSuMe learning method for spiking neural networks dedicated to neuroprostheses control." In: (January 2006).

[65] Damien Querlioz et al. "Bioinspired networks with nanoscale memristive devices that combine the unsupervised and supervised learning approaches." In: July 2012, pages 203–210. ISBN: 978-1-4503-1671-2. DOI: 10.1145/2765491.2765528.

[66] Sachin Ravi and Hugo Larochelle. "Optimization as a model for few-shot learning." In: (2016).

[67] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement." In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[68] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), page 386.

[69] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF." In: *2011 International conference on computer vision*. Ieee. 2011, pages 2564–2571.

[70] B. Rueckauer and S. Liu. "Conversion of analog to spiking neural networks using sparse temporal coding." In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2018, pages 1–5. DOI: 10.1109/ISCAS.2018.8351295.

[71] Bodo Rueckauer. *SNN_Toolbox Github repository*. https://github.com/NeuromorphicProcessorProject/snn_toolbox.

[72] Bodo Rueckauer et al. "Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification." In: *Frontiers in Neuroscience* 11 (2017), page 682. ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00682. URL: https://www.frontiersin.org/article/10.3389/fnins.2017.00682.

[73] Bodo Rueckauer et al. "Theory and Tools for the Conversion of Analog to Spiking Convolutional Neural Networks." In: (December 2016).

[74] Jürgen Schmidhuber. "Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook." PhD thesis. Technische Universität München, 1987.

[75] Frederik Emil Thorsson Schöller et al. "Assessing Deep-learning Methods for Object Detection at Sea from LWIR Images." English. In: *I F A C Workshop Series* 52.21 (2019). 12th Control Applications in Marine Systems, Robotics, and Vehicles, IFAC CAMS 2019 ; Conference date: 18-09-2019 Through 20-09-2019, pages 64–71. ISSN: 1474-6670. DOI: `10.1016/j.ifacol.2019.12.284`. URL: `http://cams-wroco.org/`.

[76] Nicolas Schweighofer and Kenji Doya. "Meta-learning in reinforcement learning." In: *Neural Networks* 16.1 (2003), pages 5–9.

[77] William Severa et al. "Training deep neural networks for binary communication with the Whetstone method." In: *Nature Machine Intelligence* 1.2 (January 2019), pages 86–94. ISSN: 2522-5839. DOI: `10.1038/s42256-018-0015-y`. URL: `http://dx.doi.org/10.1038/s42256-018-0015-y`.

[78] *ShippingLab - Driving Future Maritime Technology.* URL: `https://shippinglab.dk/`.

[79] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. "Incremental learning of object detectors without catastrophic forgetting." In: *Proceedings of the IEEE International Conference on Computer Vision.* 2017, pages 3400–3409.

[80] SNL-NERL. *SNL-NERL/Whetstone Github repository.* `https://github.com/SNL-NERL/Whetstone`.

[81] *Spiking neural network conversion toolbox - SNN toolbox 0.5.0 documentation.* URL: `https://snntoolbox.readthedocs.io/en/latest/`.

[82] Ioana Sporea and André Grüning. "Supervised Learning in Multilayer Spiking Neural Networks." In: *CoRR* abs/1202.2249 (2012). arXiv: `1202.2249`. URL: `http://arxiv.org/abs/1202.2249`.

[83] Marcel Stimberg, Romain Brette, and Dan FM Goodman. "Brian 2, an intuitive and efficient neural simulator." In: *Elife* 8 (2019), e47314.

[84] Evangelos Stromatias et al. "Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on SpiNNaker." In: *2015 International Joint Conference on Neural Networks (IJCNN).* IEEE. 2015, pages 1–8.

[85] Qianru Sun et al. "Meta-transfer learning for few-shot learning." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2019, pages 403–412.

[86] Flood Sung et al. "Learning to compare: Relation network for few-shot learning." In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2018, pages 1199–1208.

[87] Richard Szeliski. *Computer vision: algorithms and applications.* Springer Science & Business Media, 2010.

[88] Amirhossein Tavanaei et al. "Deep Learning in Spiking Neural Networks." In: *CoRR* abs/1804.08150 (2018). arXiv: `1804.08150`. URL: `http://arxiv.org/abs/1804.08150`.

[89] Keras Team. *Keras documentation: Object Detection with RetinaNet.* URL: `https://keras.io/examples/vision/retinanet/`.

[90] *Towards data science: Spiking neural networks.* February 2020. URL: `https://towardsdatascience.com/spiking-neural-networks-558dc4479903`.

[91] Ricardo Vilalta, Christophe Giraud-Carrier, and Pavel Brazdil. "Meta-Learning - Concepts and Techniques." In: July 2010, pages 717–731. DOI: `10.1007/978-0-387-09823-4_36`.

[92] Cédric Villani, Yann Bonnet, Bertrand Rondepierre, et al. *For a meaningful artificial intelligence: Towards a French and European strategy.* Conseil national du numérique, 2018.

[93] Oriol Vinyals et al. "Matching networks for one shot learning." In: *arXiv preprint arXiv:1606.04080* (2016).

[94] Xin Wang et al. *Frustratingly Simple Few-Shot Object Detection.* 2020. arXiv: `2003.06957 [cs.CV]`.

[95] Jiaxi Wu et al. *Multi-Scale Positive Sample Refinement for Few-Shot Object Detection.* 2020. arXiv: `2007.09384 [cs.CV]`.

[96] Yujie Wu et al. "Spatio-Temporal Backpropagation for Training High-performance Spiking Neural Networks." In: *CoRR* abs/1706.02609 (2017). arXiv: `1706.02609`. URL: `http://arxiv.org/abs/1706.02609`.

[97] Yang Xiao and Renaud Marlet. *Few-Shot Object Detection and Viewpoint Estimation for Objects in the Wild.* 2020. arXiv: `2007.12107 [cs.CV]`.

[98] A Steven Younger, Sepp Hochreiter, and Peter R Conwell. "Meta-learning with backpropagation." In: *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222).* Volume 3. IEEE. 2001.

[99] Davide Zambrano and Sander M Bohte. "Fast and efficient asynchronous neural computation with adapting spiking neural networks." In: *arXiv preprint arXiv:1609.02053* (2016).