Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-2019-4514823

M.Sc. Thesis

An Object Detecting Architecture using Spiking Neural Networks

Joppe Lauriks

Abstract

Spiking Neural Networks have opened new doors in the world of Neural Networks. This study implements and shows a viable architecture to detect and classify blob-like input data. An architecture consisting of three parts a region proposal network, weight calculations, and the classifier is discussed and implemented.

The region proposal network is build based on a blob detecting Laplacian of Gaussian function. The architecture is tested and verified on the Multi MNIST dataset that is generated based on the MNIST dataset that consists of handwritten digits. Results show that, on average, the region proposal network can locate the blobs in the input with an accuracy of within a single pixel distance from the ground truth. Two different ways of decoding the rate data coming from the region proposal network where discussed the Peak based decoder could propose regions even if these regions are situated closely together. A Center of Mass decoder is slightly more accurate than the Peak based decoder but at a higher computational cost and performance degradation when the regions are close together.

The region proposal network at worst only accounts for 3.19% of inaccuracy. The implementation shows that the architecture is a viable way of detecting and classifying multiple objects within the input. The data shows that the region proposal network itself is a feasible way of detecting blob-like objects within its input.

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

An Object Detecting Architecture using Spiking Neural Networks

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Computer Engineering

by

Joppe Lauriks
born in Amsterdam, The Netherlands

This work was performed in:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology
Department of
Microelectronics & Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"An Object Detecting Architecture using Spiking Neural Networks"** by **Joppe Lauriks** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 22-11-2019

Chairman: ______________________
dr.ir. T.G.R.M van Leuken

Advisor: ______________________
dr.ir. S.S. Kumar

Committee Members: ______________________
dr.ir. J.S.S.M Wong

____
dr. Amir Zjajo

Abstract

Spiking Neural Networks have opened new doors in the world of Neural Networks. This study implements and shows a viable architecture to detect and classify blob-like input data. An architecture consisting of three parts a region proposal network, weight calculations, and the classifier is discussed and implemented.

The region proposal network is build based on a blob detecting Laplacian of Gaussian function. The architecture is tested and verified on the Multi MNIST dataset that is generated based on the MNIST dataset that consists of handwritten digits. Results show that, on average, the region proposal network can locate the blobs in the input with an accuracy of within a single pixel distance from the ground truth. Two different ways of decoding the rate data coming from the region proposal network where discussed the Peak based decoder could propose regions even if these regions are situated closely together. A Center of Mass decoder is slightly more accurate than the Peak based decoder but at a higher computational cost and performance degradation when the regions are close together.

The region proposal network at worst only accounts for 3.19% of inaccuracy. The implementation shows that the architecture is a viable way of detecting and classifying multiple objects within the input. The data shows that the region proposal network itself is a feasible way of detecting blob-like objects within its input.

Acknowledgments

Let me start by thanking my advisor, dr.ir. T.G.R.M van Leuken with providing me with the chance to do this project within the Circuits and Systems group. The knowledge gained both from this project as well as other projects running within this group allowed me to grow and expand my knowledge in ways I could not have foreseen when I started. It has given me the luxury of working with a technology that already shows promising results and undoubtedly will see widespread use in the near future.

As my daily supervisor dr.ir. S.S. Kumar has given me the support I needed to be able to continue at times I felt stuck. His wise advice has helped through the times in which I saw no way forward. Without his and dr. A. Zjajo their support and critical feedback, I would not have been able to write this report.

Johan Mes and David Aledo Ortega, I would like to thank you both for the good times and wise advice you both gave me. Johan, in particular, with the support on the neural simulator. David, for the early advice he gave me concerning datasets. It goes without saying that the caffeinated beverages you guys both supplied me with fueled my study and writing of this thesis.

My colleague master students Bart, Shashanka, Ninad, Davide, and Rahul, deserve my sincere gratitude as well. We all started this journey together, helped each other and learned from each other.

I started this journey here at the Delft University of Technology together with Bas van Otterloo and Bastiaan Feenstra. The three of us have been together well before the start of this adventure, and you guys helped me both trough it as friends but also with advice as well as guys I could talk to for technical advice but also questions on a more personal level.

My mother, Christel, as well as my father Niek for support me and helping me pursue my ambitions. It goes without saying that my sister Kiki and my brother Pepijn have helped me with these ambitions as well.

And last but certainly not least, my girlfriend, Alma. Without whom I would not have been able to even coming close to finishing this thesis. Her never-ending emotional support has given me the energy to reach the point where I am now. Thank you!

Joppe Lauriks
Delft, The Netherlands
22-11-2019

Contents

List of Figures

List of Tables

Introduction 1

The introduction of Spiking Neural Networks asks for a new way of thinking about problems. Regular Artificial Neural Networks do not incorporate the notion of time within their core design, Spiking Neural Networks do. This provides a new way of processing data that has not been seen before. It opens a world full of possibilities that can be explored. Spiking Neural Networks already show promising results. Steffen et al. [15] wrote a survey about the possibilities in combining neuromorphic stereo vision and Spiking Neural Networks. They note the following:

> ”Neuromorphic systems have enormous potential, yet they are rarely used in a non-academic context. [. . .] event-based solutions are already far superior to conventional algorithms in terms of latency and energy efficiency” [15, p. 150]

Within the world of Spiking Neural Network research, classifiers and learning algorithms have been well studied. One of the next steps after image classification networks is object recognition. Combining both a localization and classification problem within a single network poses some exciting challenges. The extra dimension of time that Spiking Neural Networks possess are used within the proposed architecture, providing an excellent view of one of the many benefits of Spiking Neural Networks.

1.1 Motivation

This study started as research on processing radar data with Spiking Neural Networks. It was quickly discovered that not much research into processing data with Spiking Neural Networks was done. Even research on radar data processing with regular Artificial Neural Networks was very sparse. Open datasets that were useful to start with radar research were not suitable for the type of radar data processing using Spiking Neural Networks. Within the Circuit and Systems research group efforts have started to create such datasets. Nevertheless, other interesting problems could already be solved without having a radar dataset.

Radar data visualized in Range-Doppler format shows multiple targets within a single image as a blob. Good ways of tracking those blobs and classifying these radar blobs using Spiking Neural Networks were still missing. This study tries to solve the first problem be looking at the problem with a fresh view about the possibilities of Spiking Neural Networks.

The evaluation of such a region proposal network raises the need for a good test set. While radar datasets where hard to come by, the MNIST dataset consisting of handwritten digits is already used a lot in Spiking Neural Network research. From the MNIST dataset, a new dataset was created. This dataset, which also includes localization data, opened the doors to blob detection using Spiking Neural Networks.

The motivation for this study thus started from the need for processing radar data. However, the need for a good and straightforward dataset also presented itself. This study tries to solve both. A dataset that can be used within an object detecting Spiking Neural Network as well as a architecture and implementation that can locate and process this generated dataset.

1.2 Thesis Goals

The main goals of this study are to:

- Create a Spiking Neural Network architecture that can be used to locate and classify blob-like data within a single input sample.
- Implemented the architecture using Spiking Neural Networks to identify and classify blob-like data.
- Find or create a dataset that can be used to analyze networks that work with blob-like data.

1.3 Contributions

The main contributions of this thesis are as followed:

- A dataset generator that can be used to generate datasets based on the MNIST dataset. The generator can be used to generate a dataset according to precise specifications like minimum separation, the scale of the digits, and rotation.
- Three reference datasets that are used within this study to analyze the performance of the implemented region proposal network.
- A Spiking Neural Network architecture for locating and classifying multiple objects.
- A software framework that can be used to quickly test the proposed architecture with different region proposal networks or classifiers.
- A region proposal network that can locate blob-like input data to within sub-pixel precision of the ground truth.
- A full system implementation that combines an off the shelf classifier with the proposed region proposal network to locate and classify separate digits within the generated Multi MNIST dataset.
- An analysis of the region proposal network performance and the benefits compared to existing solutions.
- A full system analysis that is showing that the proposed architecture is a viable solution to identify and classify multiple blob-like objects within a single input.

1.4 Thesis Outline

The thesis is divided into four main parts. The first part gives the reader background knowledge about Neural Networks. The difference between typical Artificial Neural Networks and Spiking Neural Networks. The neurons and synapse these networks are based on. How data is pre-processed to be used in these neuron networks. But also how multiple object detection works right now in Artificial Neural Networks. After that, the system use case is discussed in chapter 3. Here the input data to the system will also be explained. Subsequently, the system architecture and implementation are described in chapter 4. How the network architecture is designed and the different parts of the architecture are all disclosed here. The implementation is clarifying how a region proposal network could be implemented and what kind of implementation is used for the full system implementation. The different parts of the region proposal network, as well as the full system, are being analyzed and reported in chapter 5. And finally the conclusion and future work are discussed in chapter 6 and chapter 7

Background 2

In order to better understand the proposed architecture and implementation, background information is needed. This chapter introduces the notion of Spiking Neural Networks and neuromorphic electronic systems in general. It tries to answer the following questions; which simulators are available to run these networks? How is input data encoded so it can be processed within a neuromorphic system? Furthermore, how are multiple object detection problems solved within Convolution Neural Networks?

2.1 Artificial Neural Networks

An Artificial Neural Network (ANN) is a network used to perform complex computing tasks. The computational model introduced by Warren McCulloch and Walter Pitts in 1943 [16] is inspired by the networks that are inside our brains. These networks consist of a set of nodes, which are sometimes called artificial neurons. These nodes have a specific value and transmit this value to connected nodes. Connected nodes compute their value by some non-linear function of the sum of its inputs. Weights increase or decrease the strength of a connection between the nodes, these weights are usually adapted during learning and are where the intelligence of an Artificial Neural Network

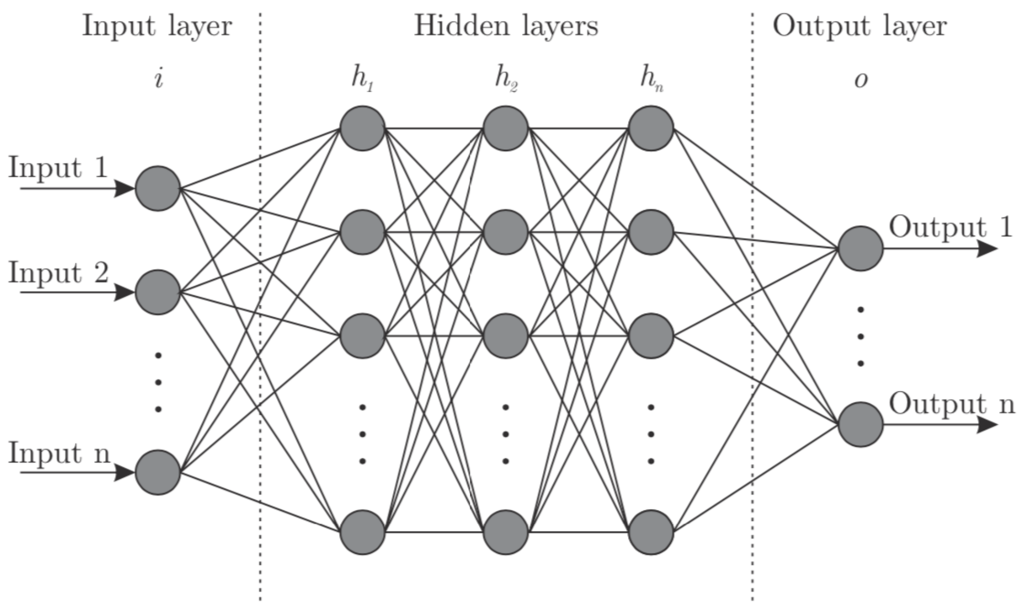


Figure 2.1: Figure showing an Artificial Neural Network architecture. [1]

originates. The function the nodes perform can be different per layer. Artificial Neural Networks are used for complex tasks such as computer vision.

2.2 Spiking Neural Networks

Spiking Neural Networks (SNN) are a special type of neural network. With an Artificial Neural Network, there is no notion of time. The input nodes of an ANN have a certain value, and based on that value, the output value is computed. Spiking Neural Networks try to mimic brain-like behaviour. Our brain is continuously processing events, which are also called spikes. Spiking Neural Networks consist of neurons and synapses that connect these neurons. The great benefit of Spiking Neural Networks is the theoretical lower power consumption and the ability to solve complex problems very power efficiently. How these Spiking Neural Networks are created and used to solve tasks such as computer vision problems is described in the next sections.

2.3 Neuromorphic Electronic Systems

In 1990 Carver Mead proposed the concept of neuromorphic electronic systems [17]. Mead states that for many problems, biological solutions exceed digital methods in terms of their effectiveness by orders of magnitude. The concept uses the analog domain to perform computation. The idea comes from the effectiveness of the brain. The brain can perform an estimate of 10^{16} operations per second once the fundamental limits of Moore's law have been reached Douglas et al. estimate that a digital system would dissipate 10 MW [18] to do the same. Comparing this to the few watts a brain dissipates shows the potential of neuromorphic computation. The TOP500 Supercomputers [19] currently show the IBM Summit as the number 1 computer in the world. The peak performance of this computer is about 200 PetaFlop/s. In other words, it can perform about 20 times the amount of operations per second the human brain performs, but the IBM Summit uses 10 MW to do this. This comparison is not truly fair because our brain does not perform floating-point operations. However, it does show the massive potential of doing compute in a more biological method.

2.3.1 Neurons

The neuron consists of thee parts, the soma, dendrites, and axon (see Figure 2.2). The neuron acts as the basic functional block of the human brain. Neurons can be divided roughly into three groups. The motor neurons which sense signals from the brain and the spinal cord and control the muscles but also our gland (responsible for hormones, for example). Sensory neurons are the neurons that receive external stimuli from our sensory organs. Such as the ears (sound), our eyes (vision) but also our skin (touch). The stimuli from these external sensory organs are converted to action potentials, also called spikes. Through the last group of neurons, the interneuron, these action potentials are connected into a neural circuit. The interneuron can be divided into two groups: the relay interneurons and the local interneurons. Relay interneurons have long axons and can connect a specific region of the brain with other regions. The

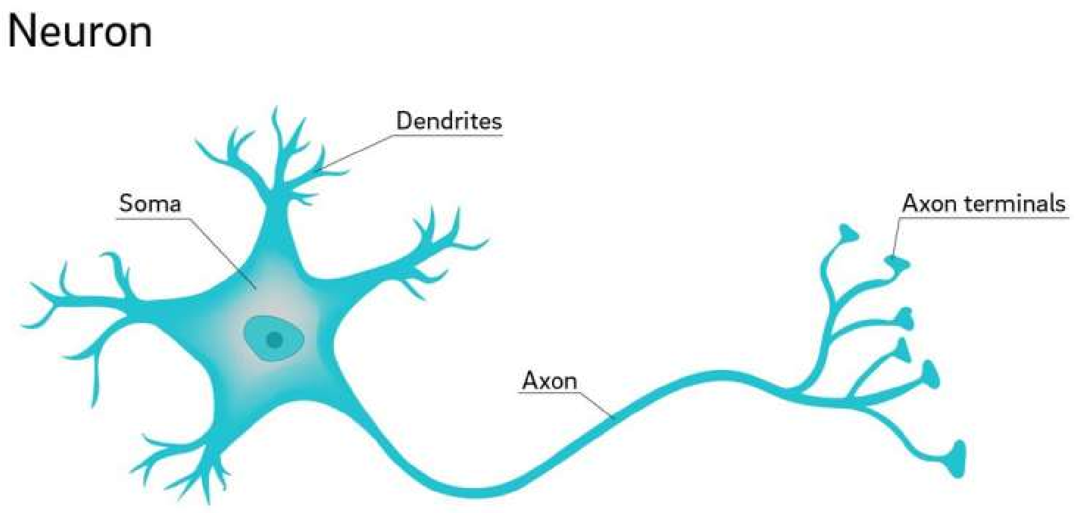


Figure 2.2: Figure showing a neuron with the three components, the soma, dendrites, and axon. [2]

local interneurons form local neural circuits with nearby neurons. These local neural circuits analyze pieces of information. Local neural circuits and the interaction between interneurons are a key process that allows the brain to carry out complex functions. It gives the brain the ability to learn and make decisions [20]. The human brain has an estimate of eighty-six billion neurons and a quadrillion synaptic connections in a child and 100 to 500 trillion connections for an adult [21][22].

A neuron responds to input stimuli, and when there are enough stimuli or the stimuli are large enough, it produces an output spike. These spikes go through the axon to all axon terminals. The spike is an all or nothing event. All other neurons that have a synaptic connection with the neuron that just spiked will receive this event through their dendrites. More on these synaptic connections will be discussed in subsection 2.3.3. After the neuron fires, the membrane voltage of the neuron resets itself, and the neuron will enter a refractory period in which the neuron will not produce new spikes and does not respond to input stimuli. When a neuron receives not enough input stimuli, the membrane voltage of the neuron slowly goes back to its resting state. Figure 2.3 shows the progress of responding to input stimuli the reset and the refractory period.

2.3.2 Neuron Model

To use the potential compute power of neurons, a model of a neuron is needed. Hodgkin and Huxley created a comprehensive model [23]. The model was created by inserting

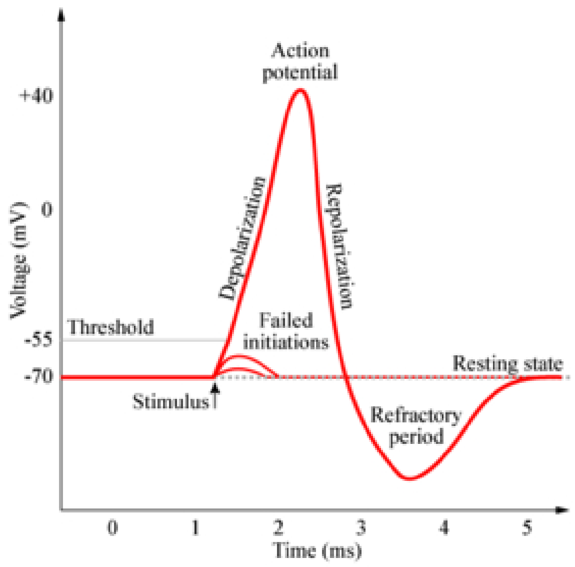


Figure 2.3: Figure showing the action potential in a neuron. The spiking phase, when it reaches the threshold, the reset, and the refractory period. [3]

micro-electrodes into the axons of a squid the resting and action potentials were measured [24] and converted into a mathematical model. The model describes the action potentials in neurons and how these are transmitted through the axons. The model consists of a set of nonlinear differential equations that describe the electrical characteristics of neurons. For the interested reader, the equations can be found in [23]. While this model represents the neuron behaviour to high precision, it is also very complex and hard to implement in silicon [25].

Izhikevich introduced a model that uses the biologically accurate model of Hodgkin and Huxley and combines it with the simplicity of the integrate-and-fire model (discussed below) [26]. This model is still able to reproduce the spiking patterns that can be found in the human brain but is computationally more efficient than the Hodgkin and Huxley model. The model is still able to represent the different spike trains that can be found in the human brain. The limiting factor of this model is the shape of the spikes. The spike shape is always the same. The Izhikevich model can therefore be used to study spike patterns and interactions between neurons but not to create and biologically accurate model.

The integrate-and-fire model is a very simplistic model [27]. It models the neuron as a simple summation process (integration). Input stimuli are summed, and when the sum over time reaches a certain threshold, an output spike is generated. This model is one of the earliest models introduced in 1907 by Louis Lapicque, and it is still widely used today. The model is described by Equation 2.1. The equation is the derivative of the equation for capacitance $C = {}^{q}/_{V}$.

$$I(t) = C_m \frac{dV_m(t)}{dt} \tag{2.1}$$

But the equation shown in 2.1 has a problem. The neuron here is modeled in a

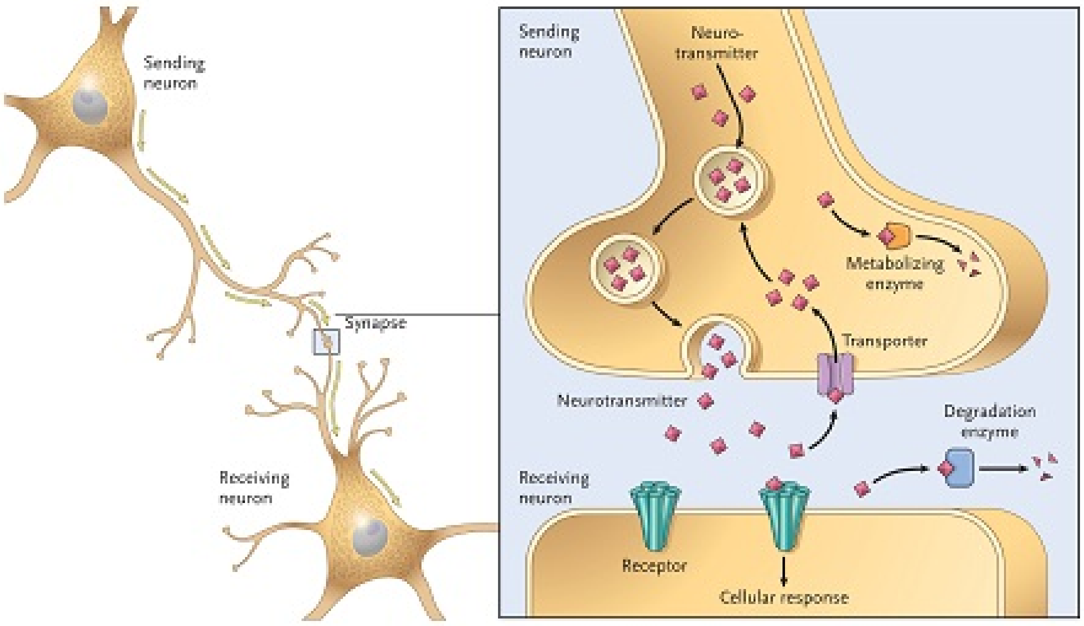


Figure 2.4: Figure showing two neurons with a synaptic connection. [4]

way where it has infinite memory. Once a stimulus has been received, the membrane is charged to a certain point and will never return to its resting state. To solve this problem, the leaky-integrate-and-fire model was introduced. The model added a discharge resistor. The membrane potential would leak through this resistor back to its resting state. This neuron can be more easily implemented in hardware.

$$I(t) - \frac{V_m(t)}{R_m} = C_m \frac{dV_m(t)}{dt} \tag{2.2}$$

The leaky-integrate-and-fire model shown in Equation 2.2 is the model that is being used in the implementation. The added complexity of the Izhikevich model is not necessary. Also, the implementation does not thrive on being biologically accurate, so the Hodgkin and Huxley model is not used. Also, the behaviour seen in the leaky-integrate-and-fire can be modeled using the Izhikevich and Hodgkin and Huxley model. So the implementation with some conversion should work with those models as well.

2.3.3 Synapse

The neuron itself is described, but to do something meaningful with neurons, a network is needed. The connections between neurons that enable the neurons to form a network are called synapses. The synapse passes electrical signals between neurons (see Figure 2.4). When there is a spike event in the sending neuron, it is sent through the axon terminals to the synaptic cleft. Here neurotransmitters are being released from the presynaptic side. These neurotransmitters are received by receptors on the receiving neuron side, also called to postsynaptic side. The dendrite on the receiving neuron receives these neurotransmitters and is able to generate postsynaptic events. These are stimuli for the receiving neuron.

While the biological model of the synapses is complex and not fully understood yet, it is clear that in order to create neuron networks a connection between neurons is necessary. In the implementation, a current-based synaptic model is used. When a presynaptic event happens, the model adds a preset amount of current on the postsynaptic side. The amount of current that is added is variable and is called a weight. By changing the weights of the synaptic connections between neurons, the ability to create networks with complex behaviour exists.

2.4 Spike Encoding

Within Spiking Neural Networks, information can be encoded in numerous ways. The two types of encodings used in the implementation are discussed below, spike rate encoding and temporal coding.

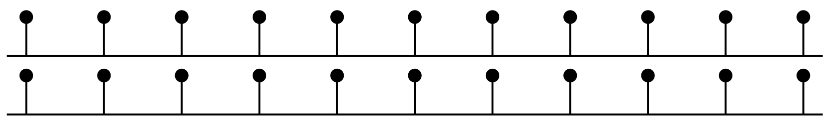
2.4.1 Spike rate encoding

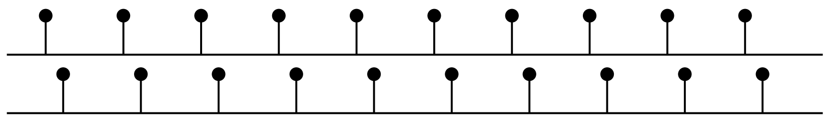
Adrian and Zotterman first introduced rate encoding in 1926 [28]. They showed that the spikes coming from sensory neurons changed rates based on weights that were hung from muscles.

Spike rate encoding is used in this study for encoding pixel-based inputs. These pixel-based inputs are converted to frequencies at which neurons will fire. The input neurons fire at the rate that corresponds to the converted pixel values. One way of doing this is by precisely timing the spike events. The periods between these precisely timed spikes are always the same. There is also the possibility to have a slight phase shift between the spike trains by applying a random offset. A more natural way of spike rate encoding is by using a Poisson distribution to generate spike events. This way, the periods between the spikes are not exact, but on average, the spike rate still corresponds to the intended rate. Both the Poisson based rate encoding and the precisely timed encoding will be used in this research. These different types of rate encodings are used because they illustrate how sensor data could potentially enter a neuromorphic system. The system should be designed to handle these types of spike trains. Figure 2.5 shows a visual representation of the generated spike trains using the discussed methods.

2.4.2 Temporal coding

Temporal coding looks at spike timing. The time at which spike events occur conveys the information. The Spiking Neural Network used for classifications in the implementation uses Temporal coding for its output. The first spike that occurs after the input is shown to the network describes the classification output. This process is also called time to first spike.


(a) Spike Rate Encoding


(b) Spike Rate Encoding with a random offset

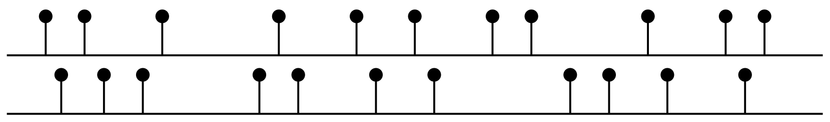

(c) Spike Rate Encoding using a Poisson distribution

Figure 2.5: Figure showing different ways of generating spike trains that represent a certain spike rate. The spike train with and without a random offset is shown, as well as spike trains generated using a Poisson distribution.

2.5 Spiking Neural Network Simulation

No hardware that implements the leaky integrate and fire model is currently available for testing these types of networks. In order to still experiment with Spiking Neural Networks, various simulators are available. Two simulators are used for the performed experiments.

Brian2 [29] is a python-based simulator. Its the successor to Brian1, which is a highly popular Spiking Neural Network simulator. Within Brian2, neuron and synaptic models are defined as a system of differential equations. Hodgkin and Huxley, Izhikevich, and leaky-integrate-and-fire can all be implemented within Brian2. One is also free in defining the synaptic model and how it connects while simulating. The synaptic connections can also dynamically be adapted while simulating. A feature that is greatly used within this implementation. The overall system is implemented in Brian2.

Within the Circuits and Systems group of the Delft University of Technology, a Spiking Neural Network simulator is also being developed. This simulator is highly performant and runs faster than Brian2. The simulator is hardware-focused and simulates a leaky integrate and fire model based on its resistor and capacitor values. The synaptic connections are modeled as current-based synaptic connections. This simulator was used for preliminary testing of the region locator and the classifier.

2.6 Detecting Multiple Objects

High performing machine learning techniques have been used to solve the object detection problem. Two of these techniques have been used as inspiration for the proposed Spiking Neural Network architecture. The first one is R-CNN, and the second one You

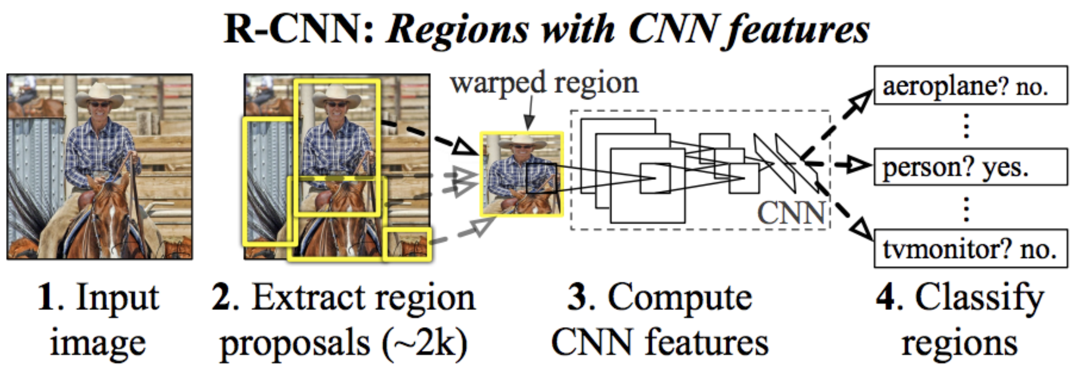


Figure 2.6: Figure showing a system overview of R-CNN. The system takes an input image (1). Extract region proposals from that image (2). Computes features for each proposal using a CNN (3). And classifies each region with a class-specific linear SVM (4). [5]

Only Look Once or YOLO for short. They will be briefly discussed below.

2.6.1 Regions with CNN features

Regions with CNN features or R-CNN for short [5] is a method that uses Convolutional Neural Networks to do region proposals. Girshick et al. came up with this solution because they saw object detection performance plateauing. Before, R-CNN object detection was done using complex systems that did feature recognition. Systems like SIFT [30] and HOG [31] which generate orientation histograms. Girschick et al. recognized the rise in CNN usage for image classification but knew that in order to do object detection, the localization problem also had to be solved. Their method generates about 2000 category-independent region proposals for the VOC 2007 dataset. The region proposal is done using a selective search, but they argue that the method itself is agnostic to a particular region proposal method. After the region proposal features are extracted from the regions using Caffe [32]. These features are then classified using a Support-vector machine (SVM) [33].

2.6.2 You Only Look Once

You Only Look Once, or YOLO for short, handles the object detection problem in a single network YOLO[6]. Instead of running a couple of smaller networks, YOLO resizes the input image to a fixed 488 by 488 pixels running a single Convolutional Neural Network on that resized image and output its predictions. The predictions consist of an x and y component, which represents the center of the box. But also the width and height of this bounding box. The class that belongs to that particular bounding box is included as well. Furthermore, a confidence level that tells how confident the network is that that prediction is correct is added. Compared to R-CNN, YOLO already generates fewer regions. YOLO generates 98 regions compared to the 2000 regions the selective search of R-CNN generates.

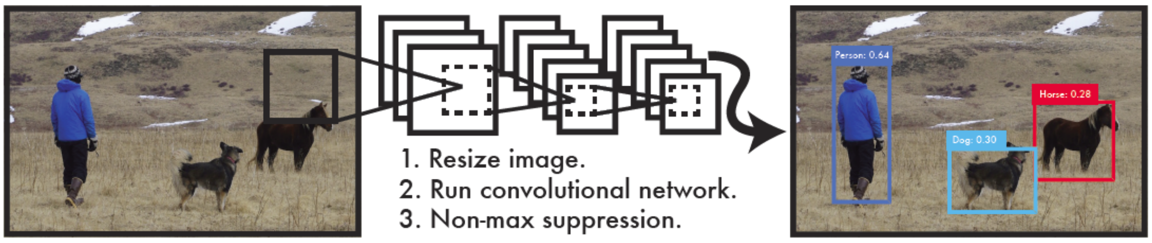


Figure 2.7: Figure showing a system overview of the You Only Look Once detection system. Three steps are given. First, the resizing of the image. Second, running the convolutional network. And third, the non-max suppression to filter out classifiers that were low on the confidence level. [6]

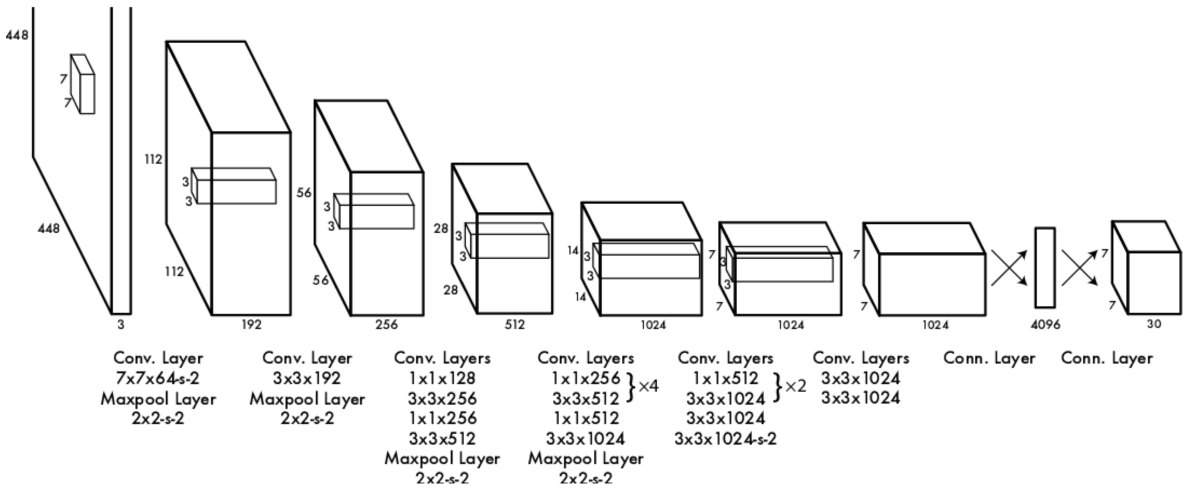


Figure 2.8: Figure showing the convolutional architecture of the You Only Look Once detector. Twenty-four convolutional layers are used, and two fully connected layers. [6]

System Use Case 3

Before the actual system architecture and implementation are discussed, let us briefly look at the system use case. Radar data is discussed because this is where the architecture has direct possible use cases and is where the reference region proposal system is designed for. A possible dataset to test this system ImageNet is discussed after. Next, the MNIST dataset is explained and why it is similar to radar data. The last section is about Multi MNIST. Multi MNIST is the dataset generator that was used for this proof of concept implementation. The connection with MNIST and the Multi MNIST generator are discussed. The contents of a Multi MNIST dataset and how the dataset is encoded is explained as well.

This study mainly focuses on blob-like data. However, the proposed architecture has more use cases. If the input data has a larger resolution than the classifier, the proposed architecture can be used to focus the classifier on a specific part of the input data. These use cases, outside of blob-like data, will not be discussed further in this study.

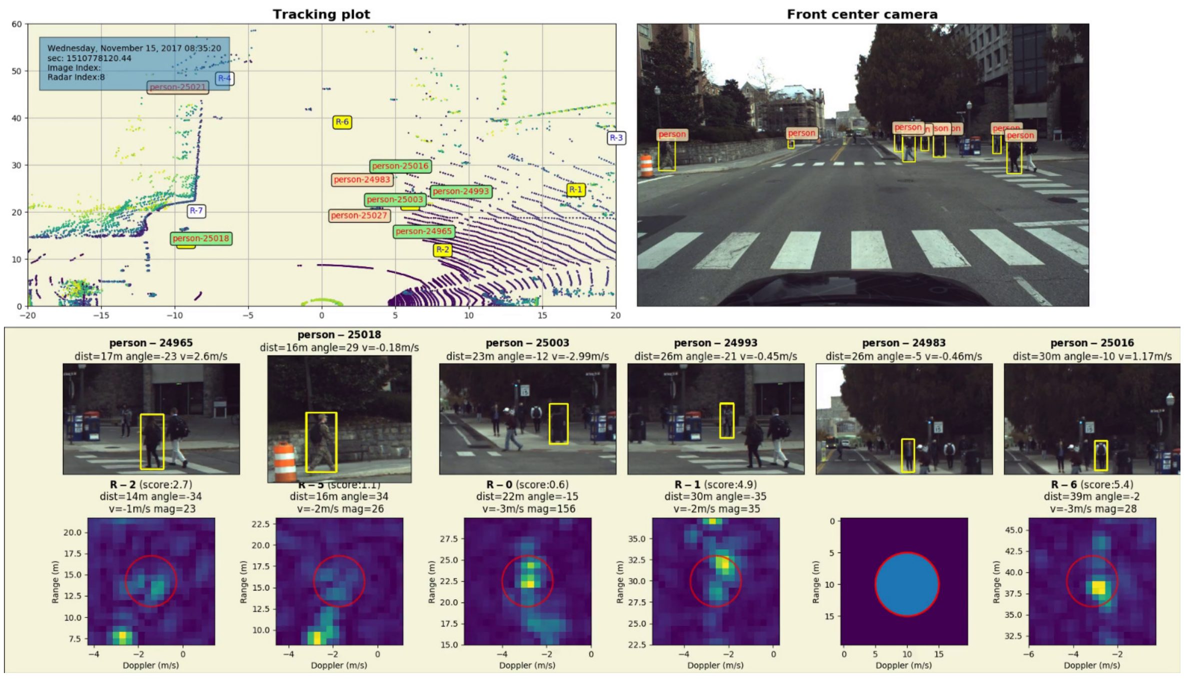


Figure 3.1: Example vehicle data with Range-Doppler data on the bottom.

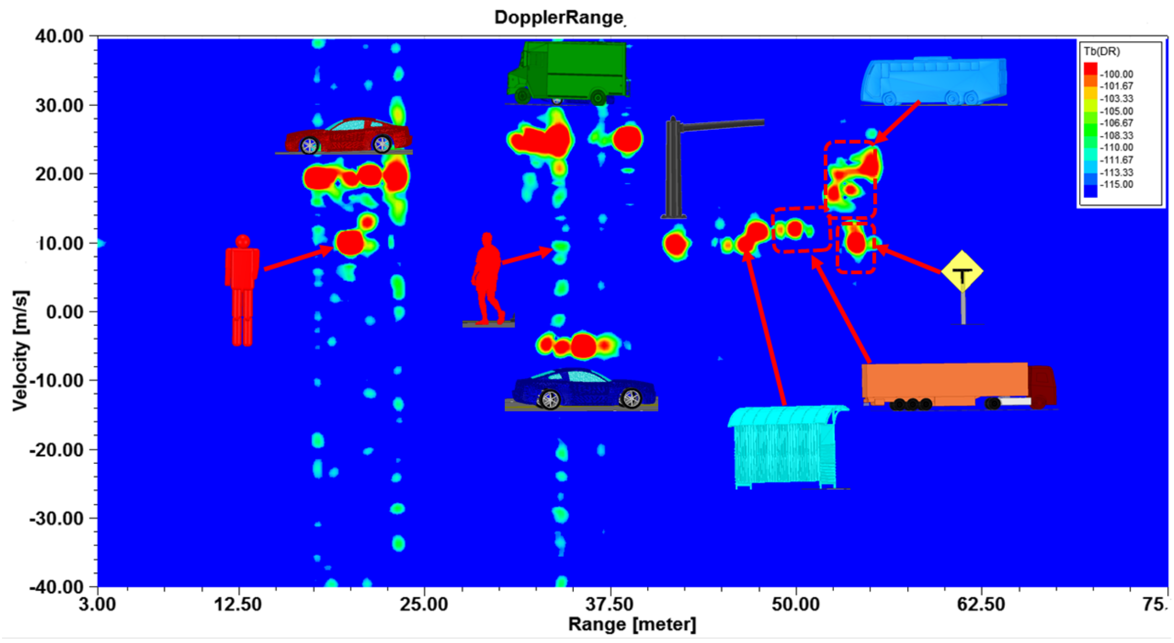


Figure 3.2: Example Doppler-Range data. [7]

3.1 Radar Data

Figure 3.1 shows a snapshot of data from a sensor-packed vehicle. On the bottom row, Range-Doppler data is shown. What one can see here is that persons are detected, and their specific radar signature is shown. All these blobs are within a single Range-Doppler data frame. In order to be able to classify these, they first need to be separated or in other words, located.

While radar data is where the idea for the architecture came from, there are some problems when using radar data. Suitable sized datasets are hard to find and are mostly kept private. There are also no existing Spiking Neural Network classifiers for these kind of datasets. Even research on deep Convolutional Neural Networks using these datasets is very sparse.

Figure 3.2 shows a figure of radar data. What can be seen here is that the radar data consists of multiple blobs. These blobs represent certain objects. In order to test the region proposal network, a dataset consisting of blobs located at different positions within a data sample is required.

3.2 Available Datasets

Due to the unavailability of radar datasets, alternative datasets that have similarities or show the same kind of problems as radar like data have been investigated. Three possibilities are discussed below.

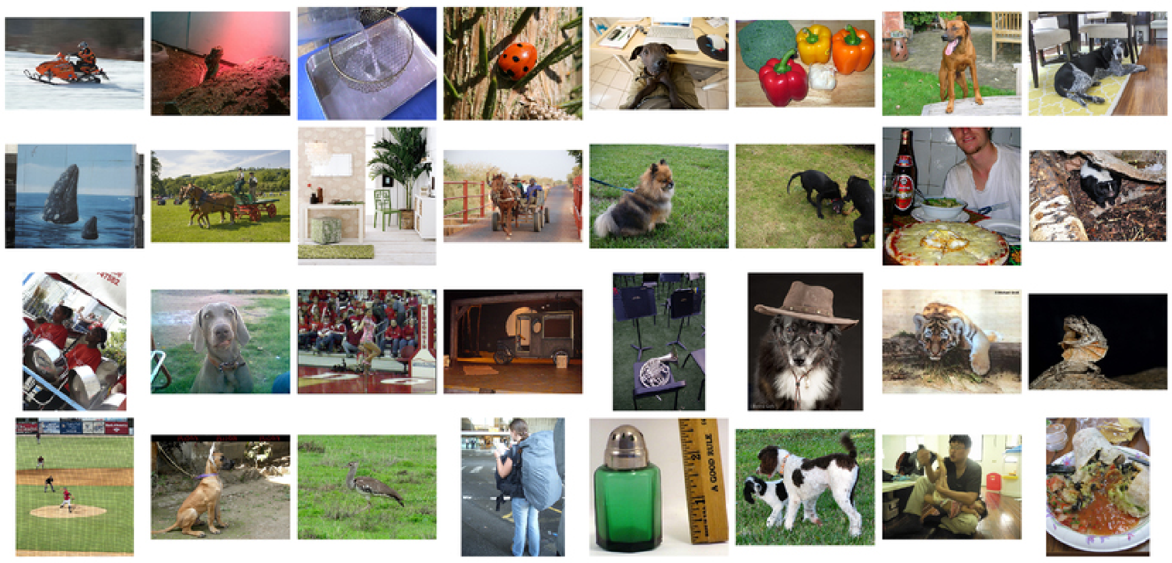


Figure 3.3: Random subset of images from the ImageNet dataset. [8]

3.2.1 ImageNet

Some very complex classification problems can be solved by neural networks. He et al. [34] already shows a deep Convolutional Neural Network with 1000 layers. With networks like that, they are able to solve complex classification problems like the ImageNet [8] dataset. The ImageNet dataset has over 1.2 million training images, 50 thousand images used for validation, and 100 thousand images that can be used for testing. These images are spread over a thousand classes (see Figure 3.3). This ImageNet dataset contains the labels as ground truth, but also the bounding boxes and thus it could be used as a localization dataset.

Some research has been done in trying to create a Spiking Neural Network classifier for ImageNet. Sengupta et al. [35] did this by training an Artificial Neural Network and converting it to a Spiking Neural Network. One of the main reasons to do it this way is because training Spiking Neural Networks is still non-trivial.

While datasets like these are available, they do not adequately represent blob-like data the radar produces. A different dataset is required to test the system implementation.

3.2.2 MNIST

MNIST[36] is a database of handwritten digits going from zero to nine. It contains over 70,000 samples. The samples are divided into 60,000 samples as a training set and 10,000 samples used for testing. The digits are centered at the center of mass and are size-normalized to fit within a 20 by 20 pixel box. The final MNIST dataset is 28 by 28 pixels in size and uses only a single monochrome 8-bit color channel. Figure 3.4 shows a small subset of the digits inside the MNIST dataset. The MNIST dataset is used to train classifications networks. There is no need for localization because a sample consists of only a single object with a defined location.

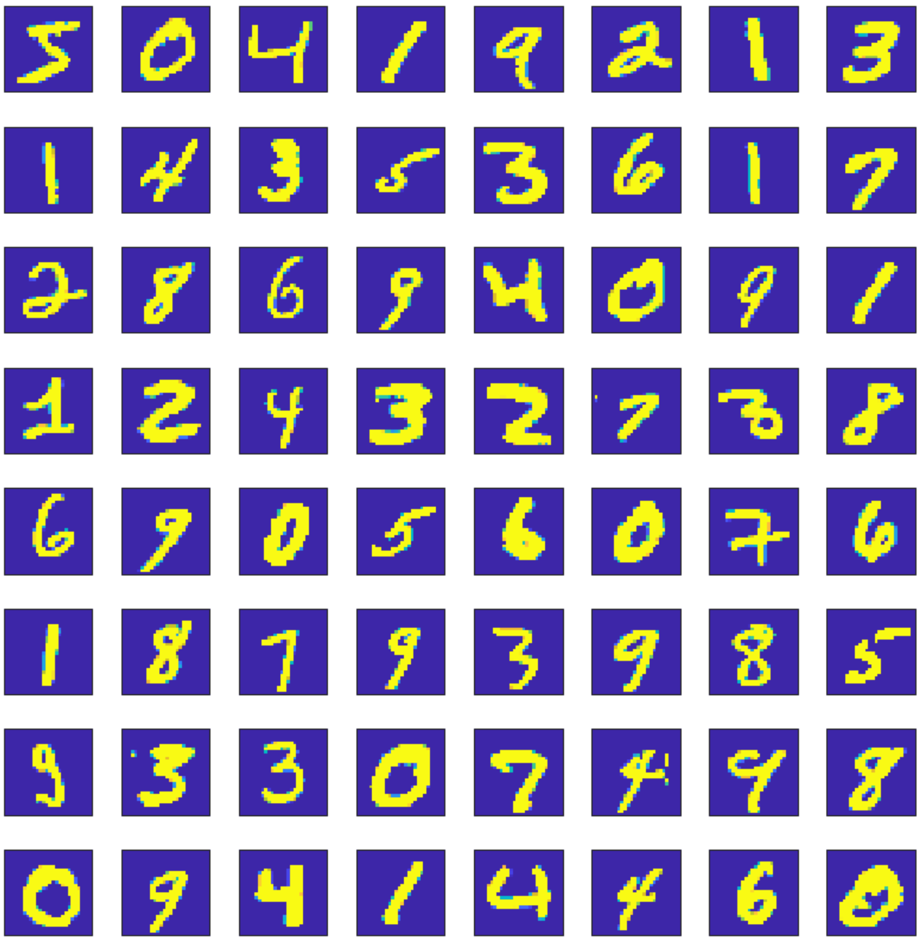


Figure 3.4: Visualization of a random subset of the MNIST dataset.

Using the MNIST dataset reasonable results can be achieved when converting networks from Artificial Neural Networks to Spiking Neural Networks. Hunsberger et al. [37] show performance on the MNIST dataset of 98.37% by introducing spiking behaviour with leaky integrate-and-fire neurons.

Also, learning methods like backpropagation [38] have been implemented in Spiking Neural Networks to train on the MNIST dataset. Lee et al. [39] show an accuracy of 99.59% when using spike-based backpropagation.

But even unsupervised learning methods that our brain could use, like spike-timing-dependent plasticity (STDP) [40] are being used to train networks to classify the MNIST dataset; Diehl et al. [41] show a 95% accuracy using this method.

Datasets like MNIST require fewer layers and less complex networks to get reasonable results. Lind published a reference implementation of a single-layer Artificial Neural Network that has a performance of 85% [42].

MNIST is a very suitable dataset to test out new networks and their classification performance. However, the MNIST dataset only consist of single digits per image sample. There is no bounding box information, which would also be useless because all the digits are sized normalized to fit within the 20 by 20 pixel box.

3.2.3 Multi MNIST

To combine the simplicity of MNIST but still be able to solve object detection problems with simple classifiers, the Multi MNIST generator was created. Multi MNIST is a dataset that consists of multiple digits from the MNIST dataset. A Multi MNIST sample consists of a larger input image. The Multi MNIST samples used in this proof of concept are 84 by 84 pixels, three times larger than MNIST. However, the power of this dataset is in the generation of it. The output size, which digits to use, how many digits per sample are all adaptable. A dataset with specific requirements can thus be very easily created.

3.2.3.1 Generation

The generation of a Multi MNIST dataset happens with a Matlab toolbox. The procedure is best explained by looking at the pseudo code in algorithm 3.1.

Per output image, the algorithm generates a bunch random numbers. First, how many digits there will be in this output image. It is configurable what the minimum and maximum amount of digits in a Multi MNIST image are. Next, the algorithms randomly picks which digits are being used in this output image. Then for every digit within this output image, the original MNIST dataset is being sampled randomly for a digit with the specified label. This MNIST digit is given a random translation, and the algorithm checks if the digit collides with other digits that are already in the output image. This is done until the MNIST digit fits or the maximum number of retries has been reached. If the maximum number of retries has been reached, the algorithm exits. This normally only happens if very large values for $Min_{separation}$ are being used, or one tries to fit a large number of digits in a small output image. The coordinates of the MNIST digit within the output, the original label, and the index within the MNIST dataset are all being saved as ground truth. This process is repeated until the output has the required total number of images. The toolbox also enables the user to generate a dataset that rotates or scale the original MNIST digits within certain bounds. This feature is not being used in the verification of this implementation.

3.2.3.2 Used Parameters and Results

A Multi MNIST dataset was generated to be used for this proof of concept implementation. The dataset has a size of 84 by 84 pixel size. All digits zero to nine were used. Within each sample of the generated dataset, one to three digits are shown. The dataset that was finally used to test the performance consisted of 10,000 Multi MNIST image samples. Multiple datasets with a different minimum separation or even overlap

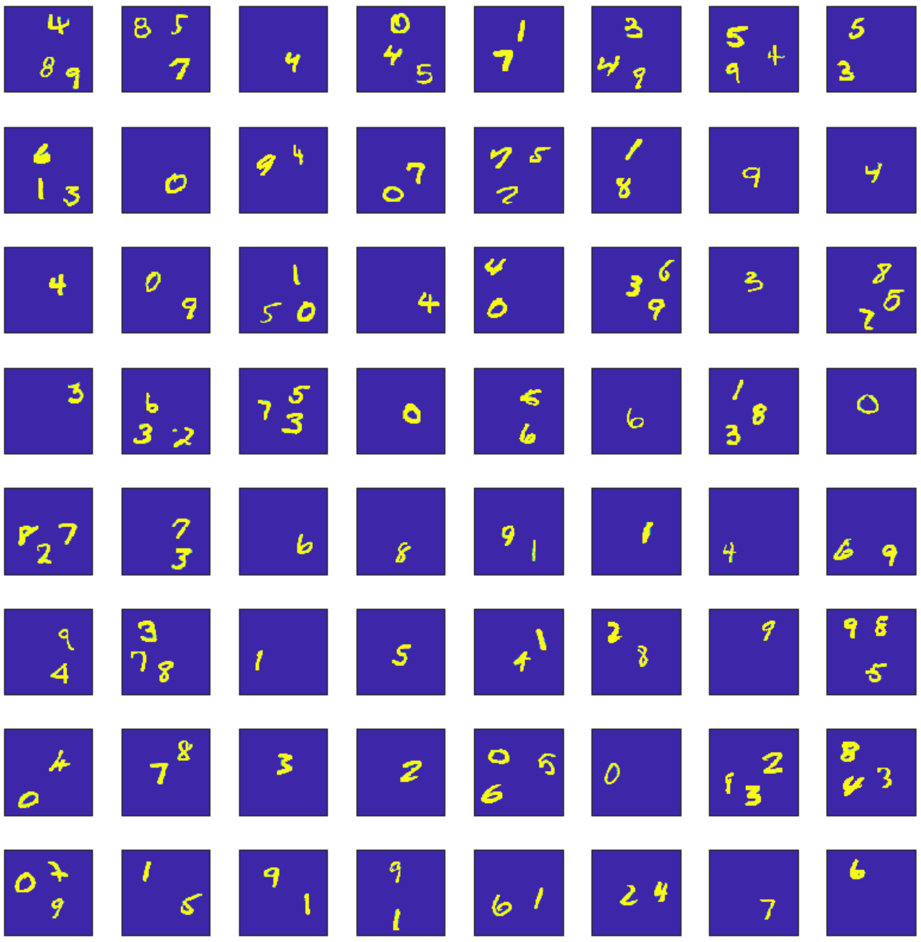


Figure 3.5: Visualization of a random subset of the Multi MNIST dataset.

were generated (see chapter 5). For tuning the region proposal network, a test set was generated that only used digits from the training set of MNIST. For all the performance metrics, the dataset generator used the test set images of MNIST.

A small subset of this generated dataset is shown in Figure 3.5. The generated datasets and the used parameters are shown in Table 3.1. The names of these datasets will be used in chapter 5 as future reference. The minimum separation guarantees a separation between black pixels of two different MNIST samples. It is still possible that another black pixel is shown within the 28 by 28 boundaries of the classifier. How this affects classification performance will be discussed in section 5.2

Algorithm 3.1: Multi MNIST dataset generation

```
1  $Output_{images} \leftarrow Zeros(N_{cols}, N_{rows}, N_{samples})$;
2  $Output_{labels} \leftarrow Cell(N_{samples})$;
3  $Output_{coordinates} \leftarrow Cell(N_{samples})$;
4  $Output_{index} \leftarrow Cell(N_{samples})$;
5  for $i \leftarrow 0$ to $N_{samples}$ do
6      $N_{digits} \leftarrow$ Random$(1, Max_{digits})$;
7      $Used_{digits} \leftarrow$ Sample$([0-9], N_{digits})$;
8      for $j \leftarrow 0$ to $N_{digits}$ do
9          $MNIST \leftarrow$ Sample$(MNIST)$ where label is $Used_{digits}$;
10         repeat
11             Random translate of $MNIST_{Sample}$;
12             Check if random translation has $Min_{separation}$ with digits in $Output_{images}[i]$;
13             if Max number of tries reached then
14                 Throw error;
15                 Exit;
16         until $Separation > Min_{separation}$;
17         $Output_{images}[i] \leftarrow Output_{images}[i] + MNIST_{Sample}$;
18         $Output_{labels}[i][j] \leftarrow MNIST_{label})$;
19         $Output_{coordinates}[i][j] \leftarrow CoordinatesofRandomTranslation)$;
20         $Output_{index}[i][j] \leftarrow MNIST_{index})$;
21 $Save(Output_{images}, Output_{labels}, Output_{coordinates}, Output_{index})$;
```

3.2.3.3 Encoding

In section 2.4 possible encodings within Spiking Neural Networks are discussed. For the proof of concept implementation, the Multi MNIST dataset is encoded using spike rates. Spike rate encoding has the benefit of continuous spike trains. It does not matter where one starts looking at the spike train; there is no defined start or end. The benefit of such an encoding is that the region proposal network can look at the spike train and determine which regions to propose. There is some time between this region proposal, setting the synaptic weights, and the classifier receiving its first spike. By not using an encoding that is time-dependent, the classifier can start its classification anywhere within the spike train.

For the generation of these spikes trains, three different encoding schemes here used.

- Spikes encoded using a fixed rate, so for the same spike frequency, every spike happens at the same time.
- Spikes encoded using a fixed rate but with a random starting offset. Spikes with the same frequency do not possibly happen at the same time anymore, but the time between two spikes is fixed.
- Spikes generated according to a Poisson distribution. Here the spikes with the same frequency do not happen at the same time anymore, and the time between spikes is random.

While the three schemes use a slightly different approach in generating the spike trains, the used frequencies to generate the spike trains are all the same. The multi MNIST data is all encoded into a 0 to 100Hz spike train. These are arbitrary numbers used for simulation and are taken because the used MNIST classifier already used these input frequencies. An actual hardware implementation would dictate the possible input frequency range. Moreover, the system is not designed to a specific frequency range with some slight parameter tuning the system could be adapted to also work with other frequencies ranges.

The difference between the performance of these encodings are disccused in section 5.2 for the region selection and in section 5.1 for the MNIST classifier. How the different rate encodings affect the system as a whole is shown in chapter 5.

dataset name	Minimum Seperation	Overlap allowed
full_test_set_overlap	0	Yes
full_test_set_3	3	No
full_test_set_6	6	No

Table 3.1: Table showing the three generated datasets and the used parameters. These three different datasets are encoded using three different schemes resulting in 9 total tests.

4 System Architecture and Implementation

In this chapter, the architectural design is discussed. An overview of the design is given, the reasoning behind this architecture, and the analysis of the design. The first section discusses a theoretical architecture. A proof of concept implementation is discussed in the sections thereafter.

The proposed design consist of three main parts:

- Region Proposal
- Weight Calculation
- Classifier

The function of these parts will be discussed in the next sections. The implementations of these parts will be explained in the sections thereafter.

4.1 Architecture

Figure 4.1 shows an overview of the proposed architecture. One of the key elements in this proposal is that the possible high bandwidth sensor data does not have to leave the neuromorphic system. This severely limits the amount of data a conventional

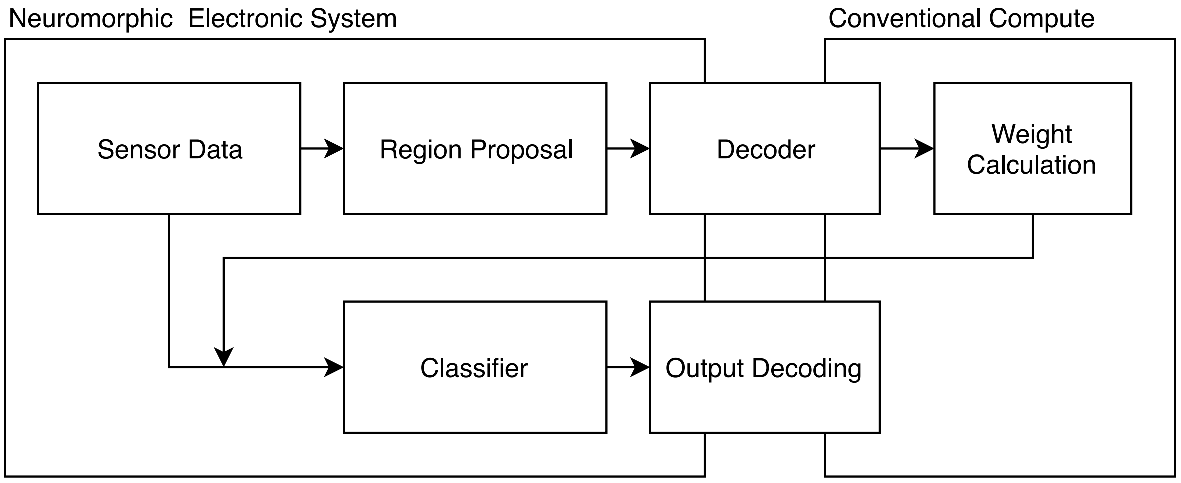


Figure 4.1: Proposed architectural overview. The architecture consists of three main parts; the Region Proposal network, weight calculations, which adapted the synaptic connections from the input data to the classifier and the classifier itself.

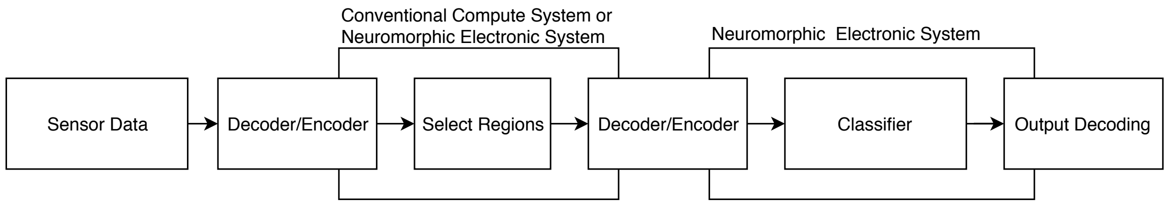

Conventional Compute System or
Neuromorphic Electronic System
Neuromorphic Electronic System
Sensor Data
Decoder/Encoder
Select Regions
Decoder/Encoder
Classifier
Output Decoding

Figure 4.2: Architectural overview of network without rerouting of the sensor data input to the classifier and re-encoding at every step.

compute system needs to process and possibly reducing the power consumption on the conventional compute side as well.

Sensor data, as discussed in chapter 3, transform the asynchronous events in the world around us in something electrical systems can process. Spiking Neural Networks are an ideal platform to process these asynchronous events. Moving the neuromorphic system close to this sensor possibly removes the need for digitizing the sensor output. The neuromorphic system would then only report relevant data and not stream all the raw sensor data to other systems.

Comparing the proposed architecture to an architecture where the region proposal is not done separately from the classification, one can already see the possible benefit of this implementation. Compared to an implementation like in Figure 4.2 where the data is first encoded for a region proposal network. This network could be a Spiking Neural Network or even a Convolutional Neural Network. The regions themselves need to be re-encoded to be at the appropriate size of the classifier. Also, one of the key benefits of using a Spiking Neural Network is the possible lower power usage (see section 2.2). Using Convolutional Neural Networks and the possible high bandwidth encoding decoding stages would greatly diminish these benefits.

The proposed architecture has benefits today as well, even though the premises of neuromorphic sensors that directly output spike trains is something to look forward to. Conventional digital sensors only have to be spike encoded once, removing the need for the decoding encoding stages in hardware. The possible high bandwidth data can stay within the neuromorphic system, keeping the conventional compute relatively simple.

The reference implementation in the next chapters shows an implementation to solve a specific problem. But the architecture could be implemented for any kind of object detection problem. The exact implementation of the classifier and possibly the region proposal changes, but the architecture still stands.

4.1.1 Region Proposal

The need for region proposal arises when the classifier is not able to classify the input as a whole, but needs to focus on certain regions. This is the case if objects in the input need to be localized as well as classified. The system now needs to answer two questions; where and what. Classifiers are usually trained on classifying single objects (see subsection 4.1.3). The task of the region proposal is to answer the first question, where.

As discussed in section 2.6 region proposal techniques like R-CNN [5] and YOLO

[6] show significant improvement in object detection tasks. Both techniques differ in their approach; R-CNN reuses classifiers, and YOLO threats it as a regression problem. Nevertheless, the intention of the system are the same, proposing regions that are later classified.

Within Spiking Neural Networks, complex region proposal mechanisms like R-CNN and YOLO could be implemented. But Spiking Neural Networks can also be used to do region proposal based on blob-like structures as discussed in section 4.2.

4.1.2 Decoding and Weight Calculations

The region proposal network encodes possible regions with a spike train. This region could be encoded in, for example, a time to first spike encoding or rate-based (see section 2.3). The information of the region proposal network needs to be passed to a conventional computing system to calculate the weights for the input of the classifier. For this weight calculation, the position (and possibly other parameters) need to be passed to the conventional computing system. The region proposal decoder translates the spike train to the digital input needed to do the weight calculations.

If a neuron spikes at the input of a synapse, this synapse transports a unit of current to its output neuron (see section 2.3 for more information). By manipulating the synaptic weights between the sensor output neurons and the classifier input neurons, certain transformations can be done.

The weight calculations are done based on the output of the region proposal decoder. In the reference region selection implementation discussed in section 4.2, the output of the region proposal decoder gives only the coordinates of a possible region. However, by manipulating the weights from the sensor output to the classifier, more transformations can be done. In order to do these kinds of translations, the region proposal network could output more information than positional information alone. Information like rotation and scale could also be included and used to get more use out of the weight calculation. If these transformations are necessary is up to the classifier. The classifier could be trained and build to handle scale and rotational variations. If a classifier is scale and rotational invariant, these transformations are not necessary. Otherwise, the input to the classifier should be scale and rotational normalized as well.

4.1.3 Classifier

The classifier is the part of the system that answers the second question of object detection; what is the object we are looking at? It attaches a label to the input given to the classifier. The classifier is attached to the input by connecting synapses. These synapses do not apply a weight function but transform the sensor output to the classifier input.

The classifier needs to be able to tolerate some error in positioning. A classifier that is translation invariant will score dramatically better than one that is not. The region proposal will always have some positional error. This is the same with rotation and scale. The region proposal can be used to correct for rotation and scale to get a normalized input to the classifier. However, the classifier still needs to tolerate an error. How significant the error is that the classifier needs to tolerate depends on the precision of the region proposal.

As with the region proposal, the classifier outputs its classification label with a spike train. In a full neuromorphic system, it might be possible to directly connect the next system to this spike train and let it interpret the classification. It would need positional data from the region proposal as well. If this data is processed in a conventional compute system, the data needs to be decoded. This process is the same as in the region proposal decoder. Albeit, the way of encoding in the classifier could be different from the region proposal encoding. If this is the case, a different implementation of the decoder is required.

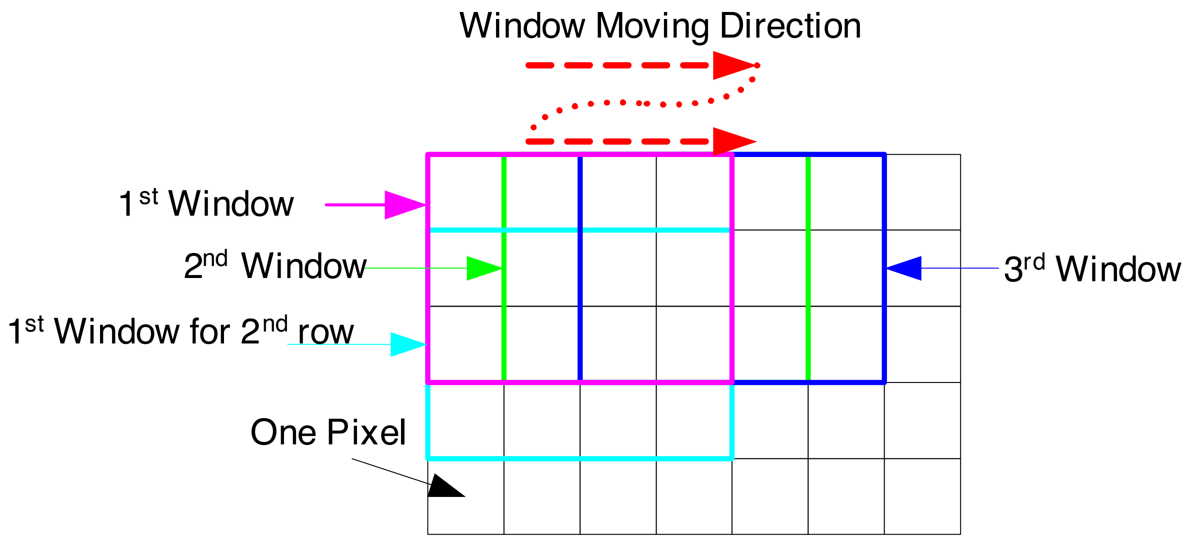


Figure 4.3: Figure showing a sliding window function [9]. A stride of one would mean that the sliding window would go from the 1^{st} window immediately to the 3^{rd} window.

4.2 Blob Region Selection

In order to correctly point the classifier at a selection of the input that needs to be classified, the region itself needs to be known. A couple of methods have been implemented for this implementation. First, the ground truth, this is not an actual region locator but uses the data from the dataset to set a performance metric to compare to. Next is the sliding window discussed in subsection 4.2.2. After that, two different blob locating techniques are shown. First, the Peak based and after that, the Center of Mass-based region selector.

4.2.1 Ground Truth Regions

Ground truth regions are coming from the dataset itself. As discussed in chapter 3 within the input data, the center coordinates of the to be classified regions are stored. The data from the dataset is used to compare the other datasets with, and validate the implementation. The ground truth should have a perfect score as a region selector.

The ground truth regions are also used in the overall performance metric. To determine if there are other variables outside of the selected region that influence the classification performance. For example, if the digits are close together, overlap within the regions could influence not only the region selection but also the classifier. It could see edges of other digits that influence the results.

4.2.2 Sliding Window

With a sliding window, there is no intelligent region selection going on. A window the size of the classifier, in this case, 28 by 28 pixels, slides over the whole input hence the name sliding window. Every time the windows slides one pixel to the side. It is also possible that the window generation has a stride; thus skipping a certain amount of columns and rows every time (see Figure 4.3). This drastically reduces the number of windows laid over the input, but the classifier needs to be more positional invariant (see section 4.3).

Because sliding windows without a stride will try every possible position within the input, there is always a case where it gets it exactly right. However, in order to do so for an 84 by 84 pixel input, it tries 3136 regions. The region proposal mechanisms introduced below tries to reduce the number of regions to the absolute bare minimum. It is thus also reducing the computational cost by not having to run 3136 regions trough the classifier. This also causes a time reduction by not having to wait for this classifier. Moreover, the number of spikes generated within the classifier is significantly reduced, which in turn reduces power consumption.

Sliding windows add another problem; the system needs to be able to determine which window had the correct classification. Usually, this is done by pooling the regions with the highest classification score and omitting the ones with scores below a certain threshold. Because these techniques introduce a new error to the system outside of the classification and positional errors, it is not used as a performance metric to gauge full system performance. However, the number of regions generated by a sliding window compared to the implemented approach is being discussed.

4.2.3 Peak Based Region Proposal

Peak based region proposal uses a Spiking Neural Network to transform the Multi MNIST input into blobs. The peak spiking intensity of these blobs correlates with the center coordinates of the original MNIST digit. This process uses two Spiking Neural Network layers, a blurring layer using a Gaussian function to calculate the weights and a blob detector using a weight distribution based on a Laplacian of Gaussian function.

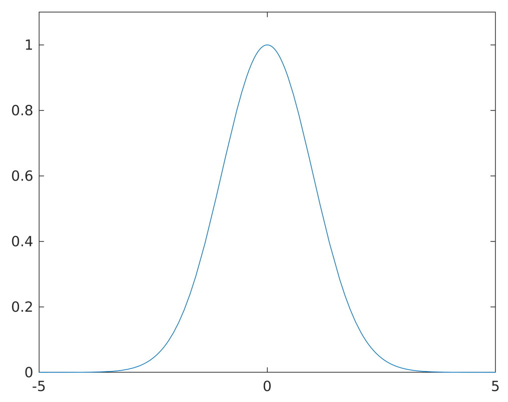
The Gaussian and its Laplacian are often used as a blob detector in image processing algorithms [10]. This is usually done by a convolution between a Laplacian of Gaussian kernel and the input. Here the Gaussian and Laplacian of Gaussian are used to calculate the weights between the input and subsequent layers. Mimicking the effect of a convolution within a Spiking Neural Network.

4.2.3.1 Gaussian layer

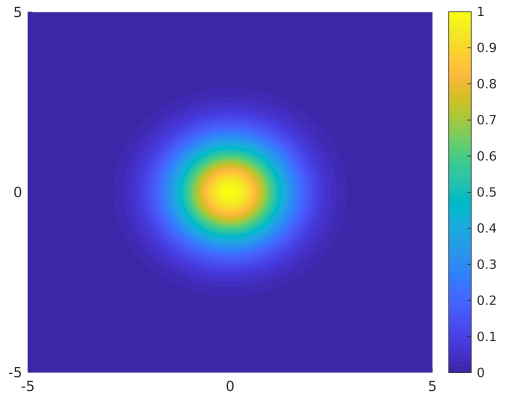
The 2D representation of the Gaussian curve used in the implementation is defined in Equation 4.1. The Gaussian is used in the network to mitigate noise and smooth the image. The Gaussian layer is directly connected to the input with an all-to-all topology. For every neuron in the Gaussian layer, the weight matrix to all the input neurons is calculated with Equation 4.1 where x and y are defined as the positional difference between the neurons.

$$z(x, y) = \exp\left(-\left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2}\right)\right) \tag{4.1}$$

The neurons in the Gaussian layer all represent a neuron in the input layer, the input layer and Gaussian layer have the same size. The index of the input neuron and



(a) 2D Gaussian



(b) 3D Gaussian

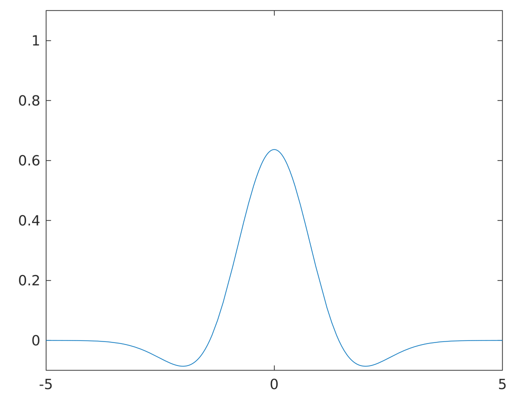
Figure 4.4: Figure showing both the 2D as well as the 3D Gaussian. It is plotted here for a σ value of 1. The values shown would be used as weights within the system. Notice that the Gaussian does not drop below zero.

a neuron in the Gaussian layer represent a x and y coordinate. The x and y coordinate can be calculated using Equation 4.2 and 4.3.

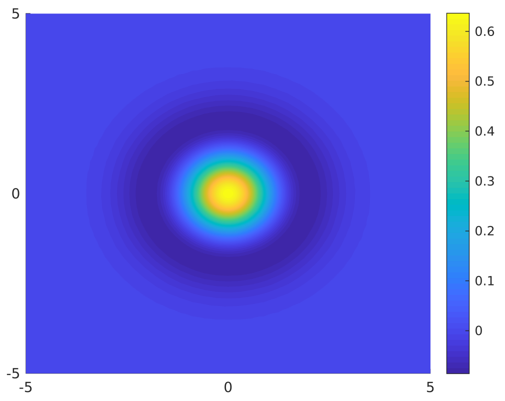
$$x = index \mod size_x \tag{4.2}$$

$$y = \frac{index}{size_x} \tag{4.3}$$

Using the index of the input neuron and the index of the Gaussian layer neuron will yield two sets of coordinates. The difference between these coordinates is used as input for the Gaussian equation. When the index of both the input and Gaussian layer are the same, the difference of these coordinates will be zero and the center of the Gaussian curve is used. The 2D Gaussian curve defined in Equation 4.1 will return one when both x and y are zero. This is done intentionally not to lose spike intensity and make the time needed until a region can be reported shorter. If the value would be less than one, there are more spikes necessary in the input layer before the Gaussian layer will spike. The implication of doing this is that the Gaussian will not filter single-pixel noise anymore. The smoothing property of the Gaussian still has an effect. The MNIST input is not that noisy, so not having noise filtering will most likely not be a problem. If noise filtering is necessary, the Gaussian described by Equation 4.4 can be used. This will lead to better noise filtering, but it does need more spikes for the membrane to reach the threshold voltage.

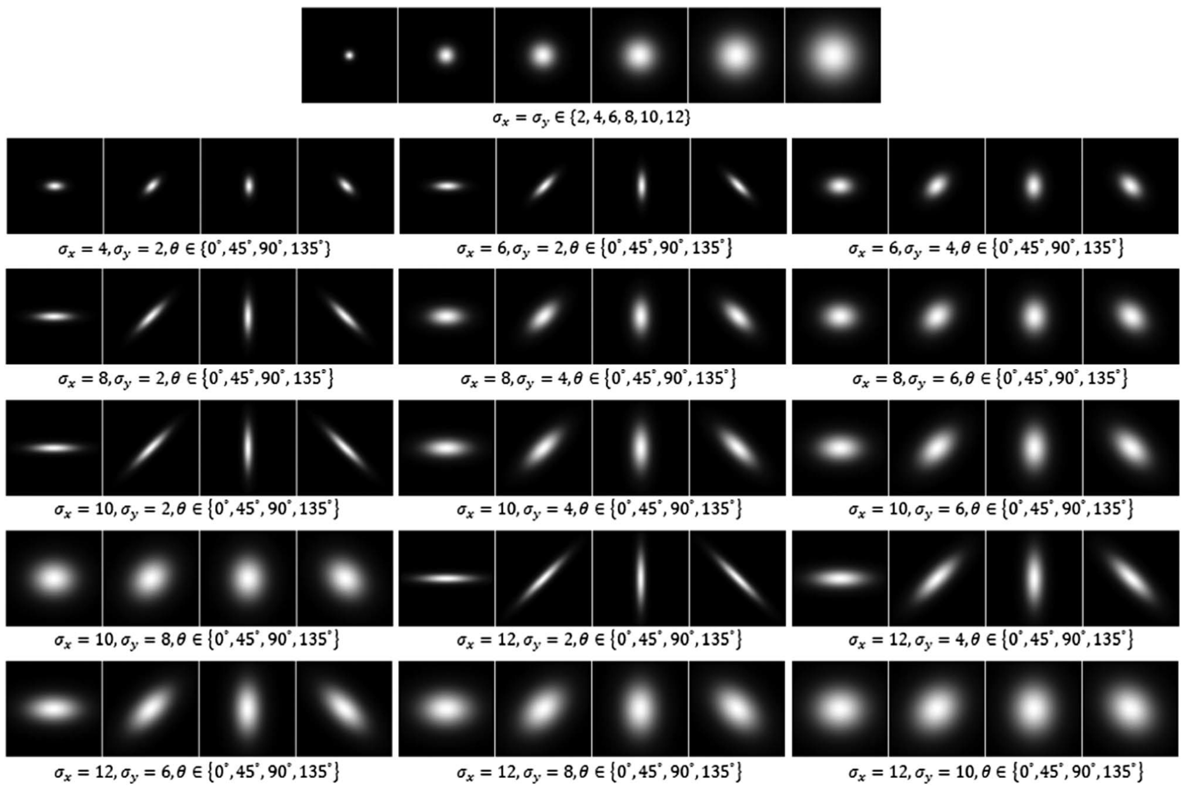


(a) 2D Laplacian of Gaussian



(b) 3D Laplacian of Gaussian

Figure 4.5: Figure showing both the 2D as well as the 3D Laplacian of Gaussian. It is plotted here for a σ value of 1. The values shown would be used as weights within the system. Notice that the Laplacian of Gaussian does drop below zero, indicating a negative weight.



$\sigma_x = \sigma_y \in \{2, 4, 6, 8, 10, 12\}$
$\sigma_x = 4, \sigma_y = 2, \theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$
$\sigma_x = 6, \sigma_y = 2, \theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$
$\sigma_x = 6, \sigma_y = 4, \theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$
$\sigma_x = 8, \sigma_y = 2, \theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$
$\sigma_x = 8, \sigma_y = 4, \theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$
$\sigma_x = 8, \sigma_y = 6, \theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$
$\sigma_x = 10, \sigma_y = 2, \theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$
$\sigma_x = 10, \sigma_y = 4, \theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$
$\sigma_x = 10, \sigma_y = 6, \theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$
$\sigma_x = 10, \sigma_y = 8, \theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$
$\sigma_x = 12, \sigma_y = 2, \theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$
$\sigma_x = 12, \sigma_y = 4, \theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$
$\sigma_x = 12, \sigma_y = 6, \theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$
$\sigma_x = 12, \sigma_y = 8, \theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$
$\sigma_x = 12, \sigma_y = 10, \theta \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$

Figure 4.6: Figure showing Laplacian of Gaussian kernels with different rotations and sizes. [10]

4.2.3.2 Laplacian of Gaussian

Given the following Gaussian function

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{4.4}$$

Applying the Laplacian operator to the Gaussian function yields the Laplacian of Gaussian.

$$\Delta G(x,y) = -\frac{x^2+y^2-2\sigma^2}{\pi\sigma^4} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \tag{4.5}$$

The σ values within this Laplacian of Gaussian function be separated into a σ_x and σ_y component. Using this non-spherical blobs can be detected better. Kong et al. [10] show even further manipulations of the Laplacian of Gaussian to detected rotation as well. These manipulated Laplacian of Gaussian kernels can be seen in Figure 4.6.

$$\Delta G(x,y) = -\frac{(x^2-\sigma_x^2)+(y^2-\sigma_y^2)}{\pi.\sigma_x^2.\sigma_y^2} \exp\left(-\frac{x^2+y^2}{\sigma_x^2+\sigma_y^2}\right) \tag{4.6}$$

The power of the Laplacian of Gaussian and why it is frequently used in blob detection comes from its response in a convolution. The peak intensity of the response will be the highest when the scale of the Laplacian of Gaussian matches the scale of the blob. This aspect of the Laplacian of Gaussian can be used to determine the scale of the blob but also its position. The middle of the object, where the Laplacian of Gaussian has the highest peak response correspondents to the center of the blob. By using the Laplacian of Gaussian for the weight calculations of the Spike Neural Network, a similar effect is achieved.

The Laplacian of Gaussian σ determines the radius of the kernel. If σ is equal to what is shown in Equation 4.7, the outcome of the Laplacian of Gaussian will be zero. Looking at Figure 4.4, the Laplacian of Gaussian is zero at a certain radius. By manipulating σ, the zero-crossing of the Laplacian of Gaussian and this its radius can be precisely controlled, which is also shown in Figure 4.6.

$$\sigma = \frac{\sqrt{x^2+y^2}}{\sqrt{2}} \iff \Delta G(x,y) = 0 \tag{4.7}$$

If the radius of the to be detected objects is known, it can be set by calculating σ with Equation 4.8.

$$\sigma = \frac{r}{\sqrt{2}} \tag{4.8}$$

The Gaussian layer before this Laplacian of Gaussian layer does alter the radius of the input data and thus affects the needed radius. This needs to be taken into account when setting the σ value for the Laplacian of Gaussian.

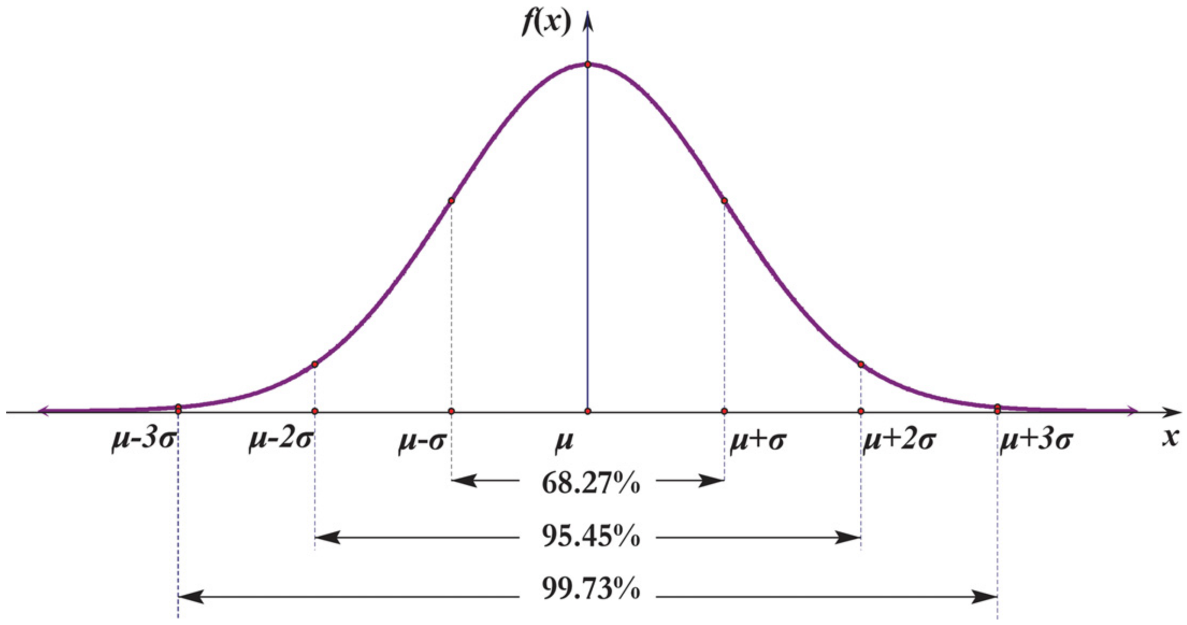


Figure 4.7: Figure showing the three-sigma rule. [11]

4.2.3.3 Parameter Values

Both the Gaussian and the Laplacian of Gaussian layer have a σ value. The σ_{gaus} for the Gaussian layer and the σ_{log} for the Laplacian of Gaussian layer needs to be determined. For the reference implementation, two approaches for calculating these values are used. The first one based on what is known about Gaussian functions, values for σ where calculated. The second approach is based on using a solver for a set of rules.

4.2.3.4 Calculating σ value

The three-sigma rule [43] states that 99.7% of the values of any Gaussian lay three standard deviations (3σ) away from its mean. In other words the energy of a Gaussian is concentrated for 99.7% at 3σ from the center (as seen in Figure 4.7). This is why Equation 4.9 is usually used to determine the σ value where r is the radius of the to be detected region. This can be used to both determine σ values for the Laplacian of Gaussian as the first Gaussian layer.

$$\sigma = \frac{r-1}{3} \tag{4.9}$$

Determining σ_{gaus} has no defined best option; it is highly application dependent. Because it is not possible to calculate an optimal value for the σ_{gaus}, some intuition and testing comes into play. For the testing, the peak based region proposal two possible σ_{gaus} values were used, and the region proposal network was tested without Gaussian layer altogether. The three possible σ_{gaus} come from a minimum, and maximum amount of blurring that could be applied. The Gaussian layer adds spikes to the network, but it also increases the size of the MNIST digit by spreading it over the input. If σ_{gaus} is too large, the MNIST digits will touch more easily, making it harder or impossible to differentiate between the regions.

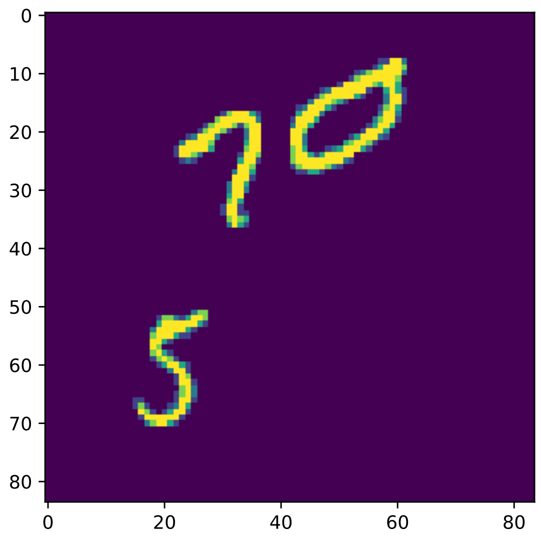
By first applying the Gaussian layer with the σ_{gaus} value described above and analyzing the output with the MATLAB image processing toolbox, the average radii for the MNIST training set could be determined. Using Equation 4.9 the values for the Laplacian of Gaussian layer σ_{log} can be calculated.

4.2.3.5 Solving

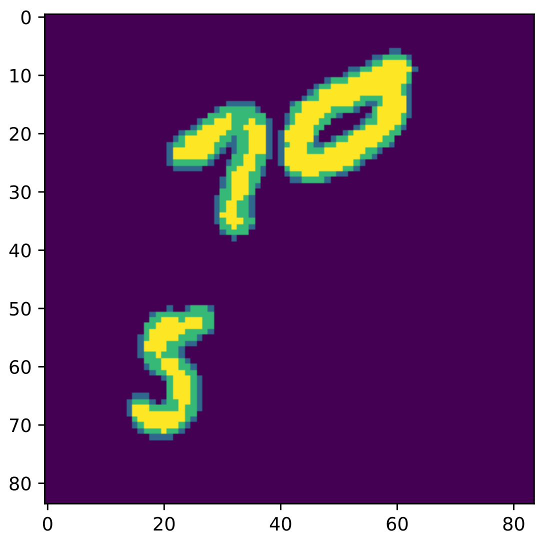
Instead of calculating possible σ values, the desired output itself could be expressed mathematically. There are a few parameters that need to be optimized. The peak response of the Laplacian of Gaussian needs to be as high as possible. Furthermore, the σ_{gaus} as low as possible to make sure it stays within 28×28 pixels. Using the MATLAB optimization toolbox, optimal σ values could be determined. The downside of the MATLAB optimization toolbox that it runs the simulation a couple of thousand times in order to determine these values. Running the region proposal Spiking Neural Network on all 60.000 images of the MNIST training set would take months. In order to overcome this, determining the optimal values was done in MATLAB using convolutions, a slight error by converting back to a Spiking Neural Network is expected. After preliminary testing, the peak response of the solved σ values was higher than the calculated ones. In the final design and performance overview, the solved values where used. The final values for σ are shown in 4.10 and 4.11.

$$\sigma_{blur} = 1.09 \tag{4.10}$$

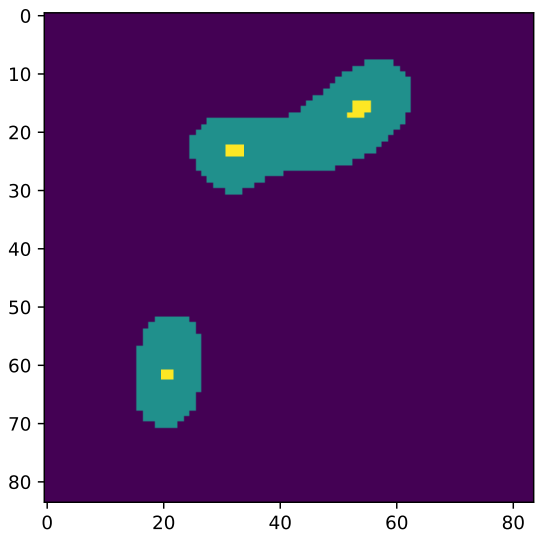
$$\sigma_{log} = 7.75 \tag{4.11}$$



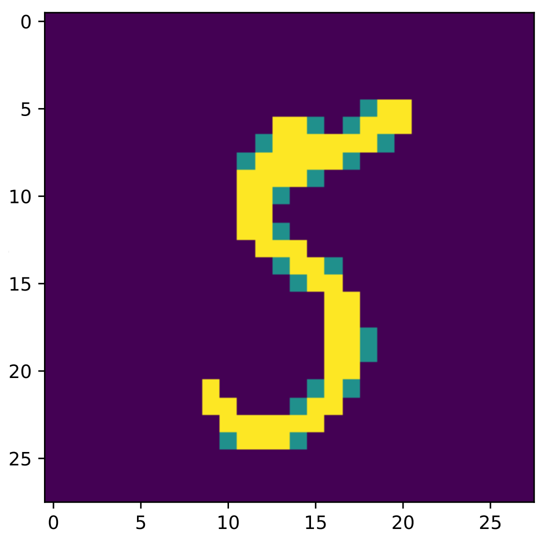
(a) Input



(b) Gaussian Layer

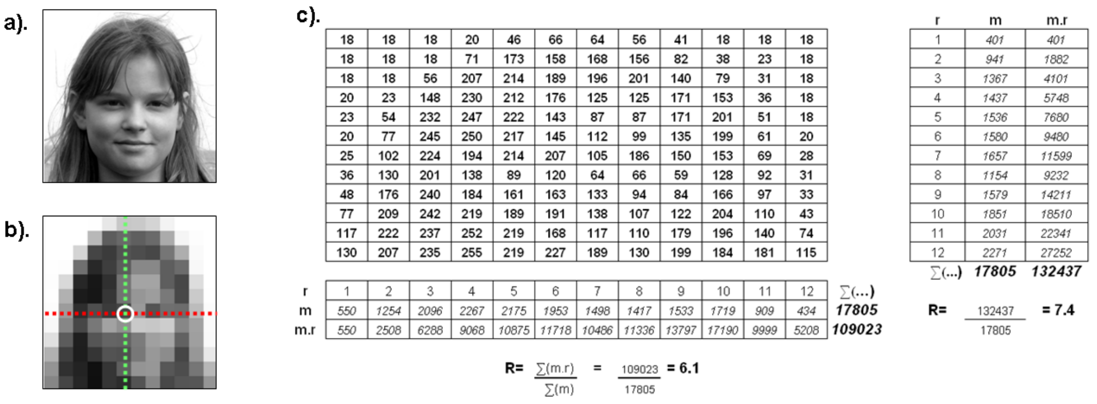


(c) Laplacian of Gaussian Layer



(d) Input of Classifier

Figure 4.8: Figure showing spike rates across the different layers of the system. First, the input image is shown, which is a sample from the Multi MNIST dataset. After that, the Gaussian and Laplacian of Gaussian layers are applied. After the region decoder, the following image is sent to the classifier. A problem can already be spotted, the regions of the 7 and 0 are merged. This is discussed in chapter 5.



18	18	18	20	46	66	64	56	41	18	18	18
18	18	18	71	173	158	168	156	82	38	23	18
18	18	56	207	214	189	196	201	140	79	31	18
20	23	148	230	212	176	125	125	171	153	36	18
23	54	232	247	222	143	87	87	171	201	51	18
20	77	245	250	217	145	112	99	135	199	61	20
25	102	224	194	214	207	105	186	150	153	69	28
36	130	201	138	89	120	64	66	59	128	92	31
48	176	240	184	161	163	133	94	84	166	97	33
77	209	242	219	189	191	138	107	122	204	110	43
117	222	237	252	219	168	117	110	179	196	140	74
130	207	235	255	219	227	189	130	199	184	181	115

r	1	2	3	4	5	6	7	8	9	10	11	12	∑(...)
m	*550*	*1254*	*2096*	*2267*	*2175*	*1953*	*1498*	*1417*	*1533*	*1719*	*909*	*434*	***17805***
m.r	*550*	*2508*	*6288*	*9068*	*10875*	*11718*	*10486*	*11336*	*13797*	*17190*	*9999*	*5208*	***109023***

$$R = \frac{\sum(m.r)}{\sum(m)} = \frac{109023}{17805} = 6.1$$

r	m	m.r
1	*401*	*401*
2	*941*	*1882*
3	*1367*	*4101*
4	*1437*	*5748*
5	*1536*	*7680*
6	*1580*	*9480*
7	*1657*	*11599*
8	*1154*	*9232*
9	*1579*	*14211*
10	*1851*	*18510*
11	*2031*	*22341*
12	*2271*	*27252*
∑(...)	***17805***	***132437***

$$R = \frac{132437}{17805} = 7.4$$

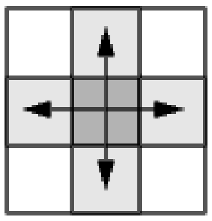
Figure 4.9: Figure showing an example of the center of mass calculation. [12]

4.2.4 Center of Mass Based Region Proposal

The Center of Mass-based [12] region proposal uses a Center of Mass calculation to determine a center point. MNIST itself is also centered on its Center of Mass (see subsection 3.2.2). This method uses the same network as the Peak Based Region proposal method. However, instead of looking at the peak response, the Center of Mass is calculated. This is done to determine if the peak response is a good indicator of the central position of the blob. In order to get multiple regions from an image, regions need to be extracted from the input images. This is done by connected-component labeling explained in the section hereafter.

4.2.4.1 Center of Mass Calculations

Figure 4.9 shows an example of a center of mass calculation. Figure 4.9.a is the original grayscale image. In the reference implementation, this would be an image showing the spike rates, as shown in Figure 4.8. As an example a pixellated version of this image is shown in Figure 4.9.b. The values of the pixels are shown in the table at Figure 4.9.c. The weight of each pixel is defined as the value of that pixel where 255 is completely black, and 0 is white. Black implies a heavyweight and white no weight at all. The table on the bottom has three rows r, m and $m.r$. r_i defines the horizontal distance from the edge. In the next row, m_i defines the sum of all the pixels (the summed weight) in that column of the original image. The last row of the table $m_i.r_i$ is the product of both the summed weight and the horizontal distance from the edge. The division of the sum of these two rows defines where the center of mass of the horizontal axis lays. In the case of this example, this is at a horizontal distance of 6.1 pixels from the edge. The table on the far right shows the same but now done for the rows of the image. The center of mass of the rows is calculated as being 7.4. That puts the center of mass of the whole image at 6.1 pixels from the left side and 7.4 pixels from the top. The center of mass is displayed as overlay in Figure 4.9.b.


(a) Von Neumann neighborhood (4-connected)

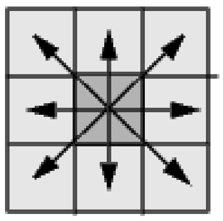

(b) Moore neighborhood (8-connected)

Figure 4.10: A figure showing an example of pixel connectivity. [13]

The center of mass calculation can be defined as Equation 4.12. This needs to be done for every axis. For a picture, this is done for two axes the horizontal and vertical axes.

$$R = \frac{\sum_{i=1}^{n} m_i . r_i}{\sum_{i=1}^{n} m_i} \tag{4.12}$$

4.2.4.2 Connected component labeling

The Center of Mass calculation for Figure 4.8 would just yield a single region. If the input image would only consist of a single region that needs to be located, the Center of Mass calculation would be enough. The input data used here consist of one to three regions. In order to extract these regions, a process called connected component labeling is used. With the labeling method, every group of pixels that are connected with one another gets a unique label. Almost all these algorithms work on binary images. The input image is easily converted to a binary image by setting a threshold. In the reference implementation, every non-zero pixel in the input image is set to one. Checking for connected components can be done in the von Neumann neighborhood [44] or 4 connected pixels see Figure 4.10a, but also in the Moore neighborhood [45] or 8 connected pixels see Figure 4.10b. For the reference implementation, the Moore neighborhood is used. This is done because the original MNIST digits have a relatively low pixel density. There are plenty of cases in which a pixel is connected by a cornering pixel and still belongs to the same MNIST digit. The only downside to using 8 connected is that when MNIST digits are being placed close to each other in the Multi MNIST dataset, they will be seen as a single region more quickly. However, once the digits start to come this close to each other, they will most likely be connected in a 4 connected setup as well.

There are a multitude of algorithms to perform connected component labeling. One method is based on graph traversal as implemented in [46] and [47]. The algorithm scans the image left to right top to bottom. When the scan finds the first unlabeled non-background (a non-zero value) pixel, it stops and gives this pixel a unique label.

The eight connected neighboring pixels are then scanned. If the surrounding pixels have no non-zero values, the pixel is marked as an unique object. If there is just a single surrounding pixel that has a non-zero value, this pixel gets labeled, and the process is repeated from the coordinates of this pixel. If there are multiple pixels with a non-zero value, the coordinates of these pixels are pushed onto the stack. The process described earlier for the single-pixel is executed for all pixels on the stack. Finally, when the whole connected object has a label, the process starts again from where it found the last unconnected pixel, scanning for a new pixel with a non-zero value that has no label. This process is repeated until it reaches the last pixel, and all objects received a label in the image.

Another algorithm called the Hoshen-Kopelman algorithm [48] (also known as the two-pass algorithm) uses a different approach. Instead of labeling each component separately, the Hoshen-Kopelman makes two passes over the binary image. The first pass iterates through each element of the data by doing raster scanning. When a non-zero value is found, it retrieves the values of the neighboring pixels. If there are no non-zero neighbours, this pixel gets a unique label, and the algorithm continues. If there are neighboring pixels with a non-zero value, the current pixel gets assigned the smallest label. The equivalence between the neighboring labels is stored. This is done for all pixels. On the second pass, all equivalent labels get relabeled with the lowest value label.

The reference implementation uses an optimized version of the two-pass algorithm [49]. In the reference implementation, these calculations are done on a CPU. However, there are also resource-efficient FPGA implementation of connected component labeling algorithms. Klaiber et al. [50] show a connected component labeling implementation on a Kintex 7 FPGA [51]. For a 256×256 image, this implementation uses 493 LUTs, 296 Registers, and 108k of BRAM. The worst-case throughput is reported as 148.47 megapixels per second. In other words, the implementation is capable of processing the 256×256 input 2256 times per second.

When the input images are separated by connected component labeling, the Center of Mass per label (component) in the input image is calculated and returned as a region.

4.3 MNIST Classifier

To test the full system, a MNIST Classifier is necessary. The reference implementation focuses on proposing the correct regions. The MNIST classifier was chosen for its simplicity. The classifier is a simple single layer (no hidden layers) classifier [42]. The classifier was initially implemented in an Artificial Neural Network and ported to a Spiking Neural Network. It was chosen because of two reasons. First, the classifier had a base classification score that was reasonably high of 80.2%. Furthermore, the classifier does not implement any max-pooling. This means that the classifier is not very positional invariant. Any shift in the input will result in a loss of performance. Generally, some sort of max-pooling will significantly help the classification performance. However, helping the classifier with positional invariance is not the goal. The region proposal should help position the input to the classifier. In a real-world application using max-pooling is recommended. Slight errors from the region proposal network will still be acceptable when using max-pooling [52].

The classifier was first tested in the simulator developed by the CAS group. After that it was ported to Brian2, where it was also tested with a Poisson input group, and the reported results are from the Brian2 simulations. Classification performance and an analysis of this classifier can be found in chapter 5.

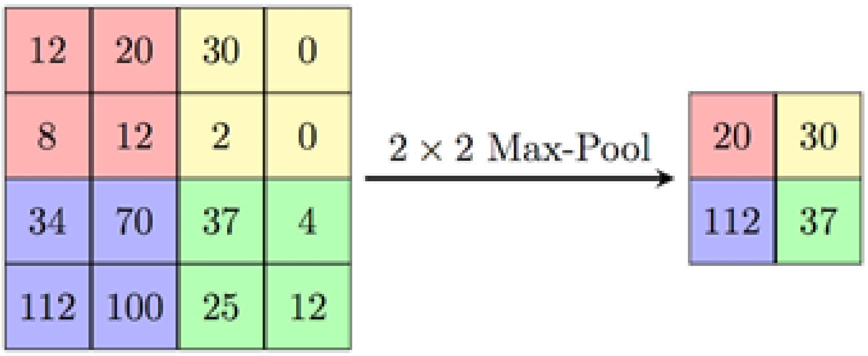


Figure 4.11: Figure showing 2 by 2 Max-pooling. Pixel blocks of 2 by 2 pixels are reduced to a single pixel. The value of that pixel is chosen by taking the maximum pixel value of the 2 by 2 region. This reduces the feature size but makes it more resilient towards positional errors. [14]

5 Results

The results are divided into three parts. First, the MNIST classifier, here the performance of the MNIST classifier is discussed without a region proposal network and on the standard MNIST dataset. Also, the positional invariance is investigated in order to determine the needed precision from the region proposal network. Thereafter, the two region proposal systems are investigated. The results show the positional precision and the number of missed regions. And last, the full system implementation. The system implementation shows the error as a complete system and is broken down into the error coming from different components.

5.1 MNIST Classifier Performance

In order to set a baseline performance, two types of experiments were performed. First, a baseline experiment in which the test set of MNIST is spike encoded and fed through the Classifier network. The spike encoding is again done in three different ways. With the Poisson input group, rate-based and rate-based with a random start offset. This is done to determine how the classifier performs when fed different styles of spike trains. After that, the MNIST test set is shifted, and the network is tested again. The loss in performance after the MNIST test set is shifted shows how positional invariant this MNIST classifier is. It also shows what kind of precision is needed from the region proposal network.

5.1.1 Normal Performance

Test Name	Total Correct	Total Incorrect	Classification Score
MNIST Rate Encoded	7949	2051	79.5%
MNIST Rate Encoded Random 10ms offset	8021	1979	80.2%
MNIST Poisson Rate Encoded	7843	2157	78.4%

Table 5.1: Performance of the MNIST classifier using three different types of spike rate encodings.

From Table 5.1 it can be seen that the MNIST classifier scores about the same with different styles of rate encodings. The rate encoded MNIST dataset is deterministic and will always produce the same results because the spike trains are always the same. The rate encoding with the random offset and the Poisson rate encoded spike trains have slight variations per run. The average classification performance stays the same without high fluctuations. However, the exact digits it classifies incorrectly are different in each run. This needs to be taken into account when using these types of encodings

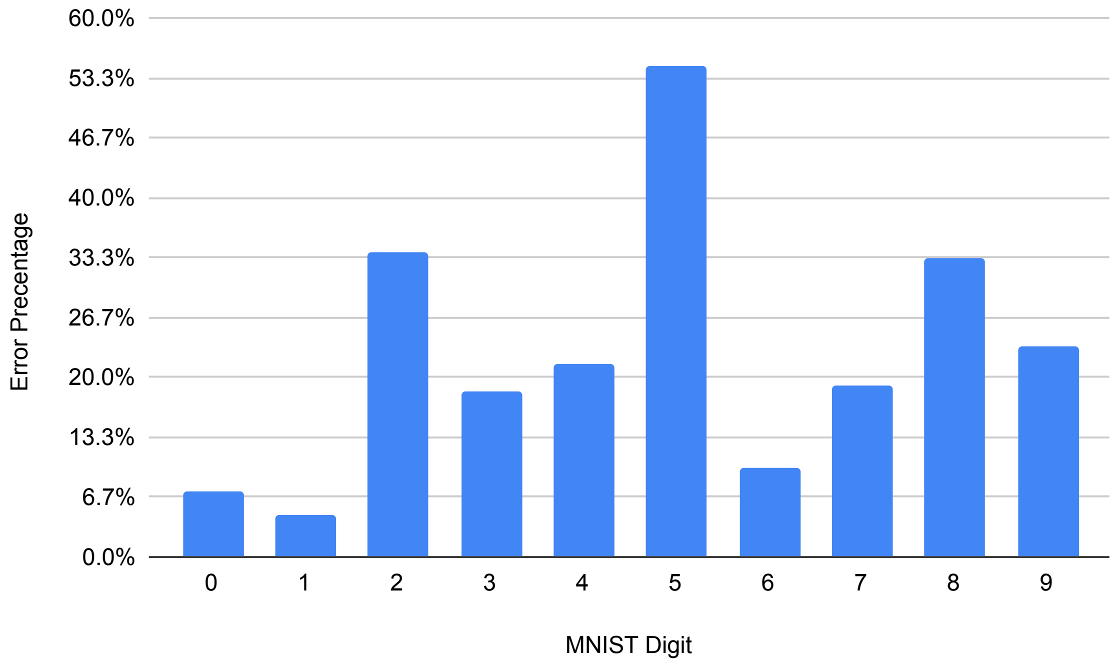


Figure 5.1: Chart showing the average classification error per digit over the 3 different types of encodings. The chart clearly shows that there are a few digits that perform better. For example, digit 5 has a higher error rate than any other digit.

within the full system. These results can later be used to compare against the full system where the positional error will also be taken into account.

	Correct Precentage		
Digit	MNIST Rate	MNIST Rate Random 10ms Offset	MNIST Poisson Rate Encoded
0	94.9%	93.5%	92.2%
1	96.7%	96.3%	95.2%
2	67.5%	68.3%	66.2%
3	84.3%	84.0%	81.7%
4	82.5%	82.1%	79.6%
5	47.0%	47.3%	47.1%
6	90.6%	91.6%	90.6%
7	82.4%	82.8%	81.6%
8	66.6%	71.0%	67.2%
9	77.8%	80.3%	78.0%

Table 5.2: Performance of the MNIST classifier per digit, using three different types of spike rate encoding.

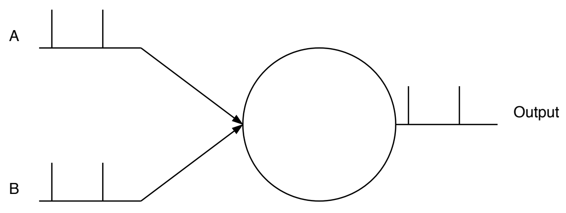
Table 5.2 and Figure 5.1 clearly show that the errors are not distributed equally over the classes within the MNIST dataset. This classifier has more trouble with the digit five than any other digit within the MNIST dataset. Up to 54.8% of the digits with label five are classified incorrectly. The three different types of encodings show very little difference.

Table 5.3 shows the average response time of the MNIST classifier. This can later be used to compare it to the region proposal. The exact numbers are not that meaningful. The simulation was done using an arbitrary 0 to 100Hz input frequency, as discussed in chapter 3. The speed at which the classifier responds is depended on the neuron model used in hardware and the speed at which the input is presented to the system. It does give a good indication of how long the classifier takes compared to region proposal.

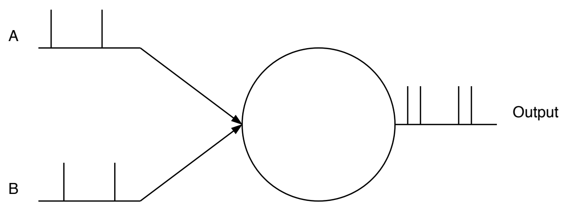
	Average Response Time	
Test Name	Correct Classification	Incorrect Classification
MNIST Rate Encoded	16ms	24ms
MNIST Rate Encoded Random 10ms Offset	23ms	30ms
MNIST Poisson Rate Encoded	25ms	31ms

Table 5.3: Table showing the average time to first spike for the MNIST classifier.

An interesting thing to note is the slightly higher classification performance when using a rate encoding with a random 0 to 10 ms offset. The reason can be found by looking at Figure 5.2. In this figure, the input frequency of A and B are both the same. However, in the second figure, the inputs are slightly shifted. This results in more output spikes. For this reason, fewer spikes are lost within the neurons. This is also more an issue within simulation where spikes can reach a neuron at exactly the same time, but per delta step, only one output spike can be generated. There is also no further improvement with the classification score, when going beyond a 10ms offset. The most common frequency found within the classifier is 100Hz, where the period between spikes is 10ms. Giving a 100Hz signal an 11ms offset would be the same as giving it a 1ms offset.



(a) Spike rate encoded



(b) Spike rate encoded with random offset

Figure 5.2: Figure showing the difference in spike output of a neuron when using spike rate encoding with and without a random offset for the same input frequency.

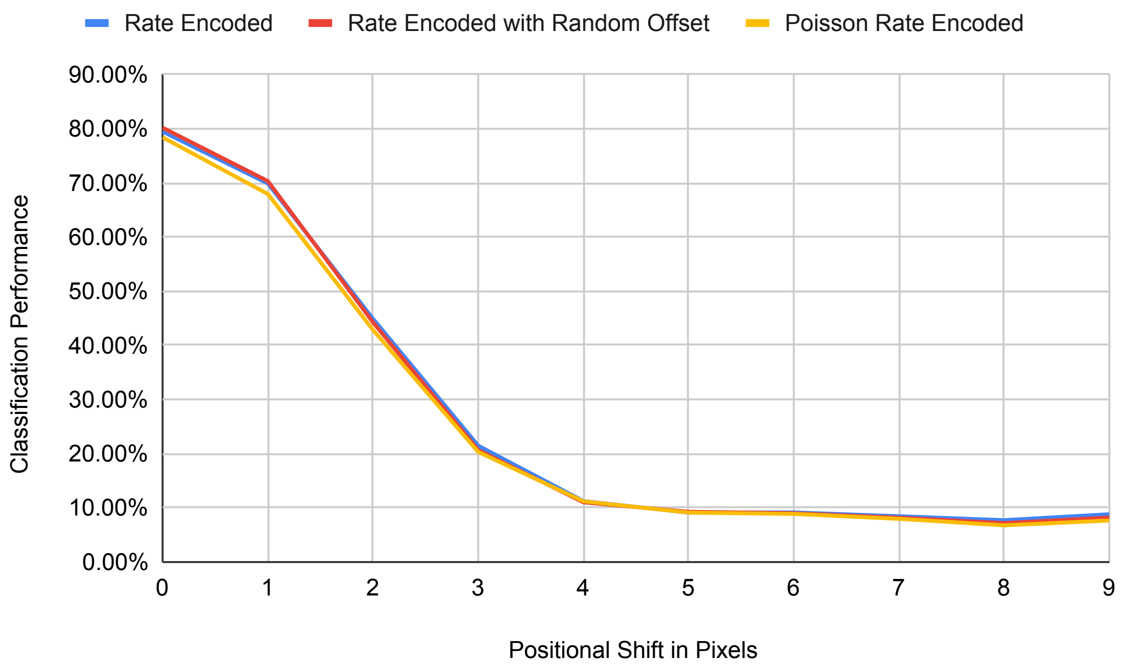


Figure 5.3: Chart showing the classification scores of the MNIST classifier when a shift to the input data is applied. A sharp drop in classification performance can be noted. Also no notable differences in performance when using a different type of rate encoding can be seen.

5.1.2 Positional Invariance

The MNIST classifier is tested again with the original test set of the MNIST dataset. However, this time, a shift to the input images is applied. This shift would correspond to what happens if the region proposal network is not 100% accurate with its location estimation. As discussed in section 4.3, this MNIST classifier does not use any max-pooling or other techniques to perform better when there are positional inaccuracies. It is recommended to implement these techniques in a real-life system, as region proposal will never be 100% accurate. However, for testing the system, this is actually preferred. The error can, later on, be more easily divided into a classification or positional error.

The classification scores for the applied shift values can be seen in Figure 5.3. The classification score already has a significant hit of 10% when only a single-pixel shift is applied. This shows that this classifier is not positional invariant and that a slight positional error will produce a loss in accuracy.

Figure 5.4 shows the data plotted per digit for only the rate encoded input. Two things are interesting to see here. First, the digit one has a lower performance degradation per pixel shift than the other digits. This is because it is a simple digit consisting of no corners, but it represents just a simple bar. The features that the classifier tries to recognize are therefor quite blurred and easy to recognize even if there is a slight shift applied.

The other interesting fact is that the digit six goes up in classification performance

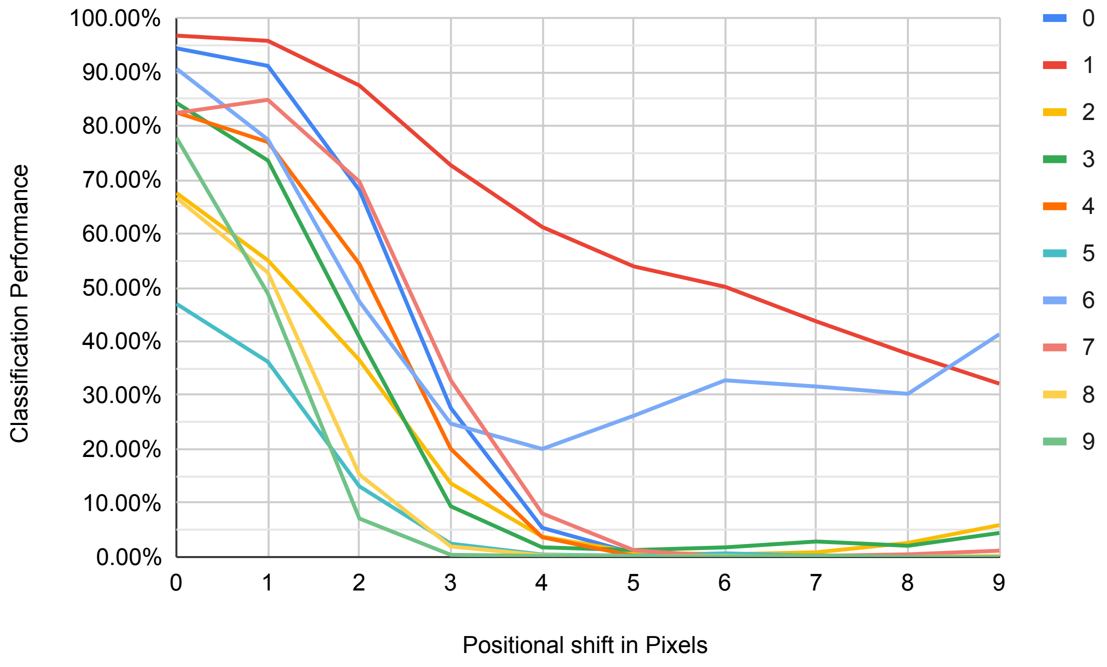


Figure 5.4: Chart showing the classification scores of the MNIST classifier per digit for the normal rate encoding, when a shift is applied to the input data. As can be seen here, the digit one has a lower performance degradation.

slightly from 4-pixel shifts onward. After some investigation, this is only seen in a horizontal shift. Due to the round bottom part of the six coming back over the original feature detector, as seen in Figure 5.5. This overlap makes that the output neurons are still able to send out a classification spike. The six also does not hit any part of the other features of the digits that would negatively impact the classification score. The only noticeable difference is that the time to first spike is lower for this shifted six. Also, compared to its original classification score of 90%, the difference is still notable.

For the interested reader, the raw data and the per digit classification data of the three different encoding schemes can be found in Appendix A.

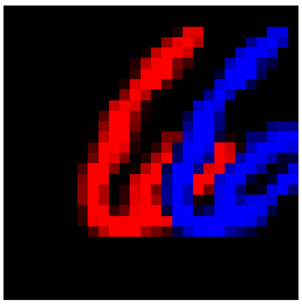


Figure 5.5: Figure showing digit six at its original position in red. And a shifted digit six in blue.

5.2 Region Selection

Two region proposal mechanisms have been implemented. A peak based region proposal network and a Center of Mass-based one. While the Center of Mass-based region proposal is more computational intensive, it is mostly used to verify that the peak based region proposal network is a valid choice. For the final network, both systems will be used and compared as well with a classifier attached. Both networks have been tested on the three generated datasets. Also, the networks have been tested with three different encoding schemes. The input is rate encoded, rate encoded with a random start offset, so spikes do not happen at the same time, and encoded by using a Poisson input group. This is done to test if the networks will still perform in a more natural environment where input stimuli happen at random and not at precisely timed intervals. For the interested reader all data is available in Appendix B.

5.2.1 Peak Based Region Proposal

Both the region proposal networks are rate based. In order to determine how long the rate decoder needs to run to achieve reasonable performance, the error and number of missed regions for a certain runtime are investigated. Table 5.4 shows the positional error compared to the runtime. The table starts at 20ms because this is when the network starts to produce output for all encoding schemes. This is due to the dominant frequency in the input. This frequency is 100Hz, and thus the second spike of the rate encoded input arrives at 10ms at the input. There is also no huge difference in performance between the different types of encoding schemes.

	Positional Error in Pixels		
Runtime	**Rate Encoding**	**Rate Encoding with Random Offset**	**Poisson Rate Encoding**
20	0.79	0.89	0.83
30	0.79	0.93	0.76
40	0.91	1.05	0.94
50	0.90	1.02	0.86
60	0.86	0.99	0.87
70	0.95	1.08	0.91
80	0.90	1.02	0.86
90	0.92	1.04	0.92
100	0.94	1.06	0.90

Table 5.4: Table showing a comparison of the positional error for the full_test_set_6 dataset with the three different encoding schemes and with different runtimes using the Peak Based Region Proposal network.

As can be seen from Table 5.4, the error hovers around the 0.90 pixels off from the ground truth. Running longer does not have a tremendous effect on improving the error. What is interesting to note is the number of missed regions Figure 5.6. After only 30ms of runtime, there is a sharp drop off detected in the number of missed regions. After 30ms of runtime, the number of regions that the Peak Based region proposal network missed has dropped to nearly zero. What is to be noted here is the significant difference between the number of missed regions at 20ms. With the Poisson

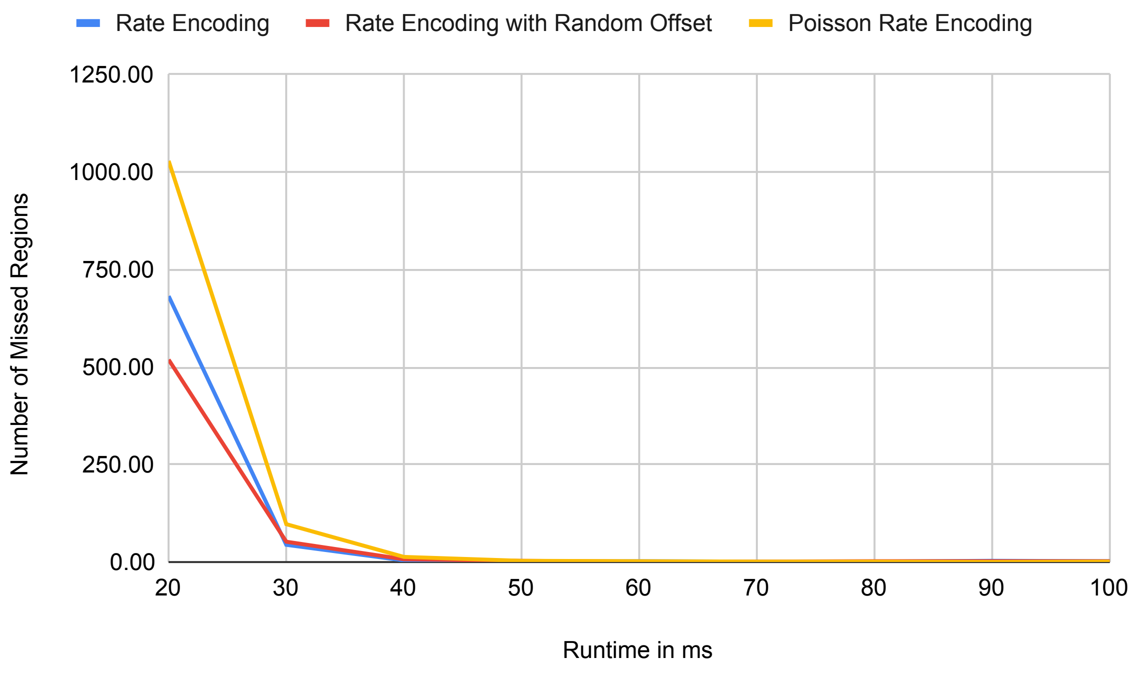


Figure 5.6: Chart showing the number of missed regions plotted against the runtime in ms for the full_test_set_6 dataset for the Peak Based Region Proposal Network.

Rate encoded input, the number of missed regions is the largest. This is mainly due to insufficient spikes reaching the decoder. The difference between the different input encoding schemes reduces significantly after 30ms.

Figure 5.7 shows that the network has the most trouble with the digit one. The same is also seen in the data of the Center of Mass-based region proposal network. This is actually due to the Laplacian of Gaussian. Because the one is the digit that misses a real round blob-like structure, which the other digits have, this causes the blob detecting Laplacian of Gaussian missing it sometimes. The 41 regions that are missed here only account for 0.20% of the total number of digits and 2.04% of the digits one. The impact is thus reasonably small, and the problem becomes less with a longer runtime.

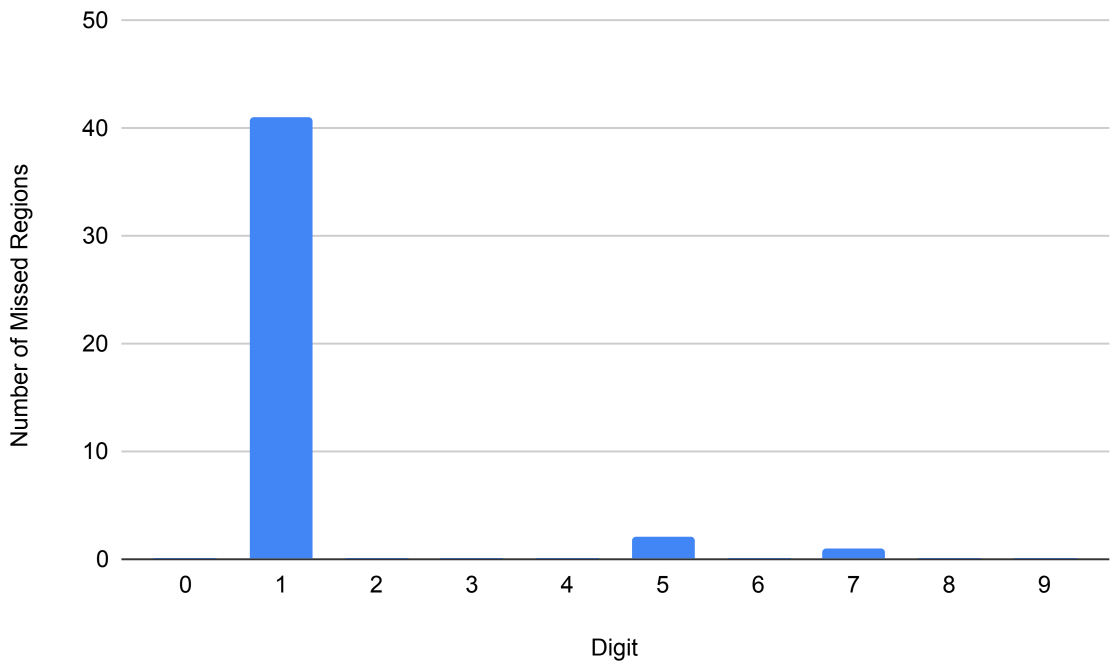


Figure 5.7: Chart showing the number of missed regions per digit with the full_test_set_6 dataset for the Peak Based Region Proposal Network.

5.2.2 Center of Mass Based Region Proposal

Table 5.5 again shows the positional error against the runtime. The error here hovers around 0.66 pixels from the ground truth, which is better than the Peak Based Region proposal network. However, what is to be noted here can be seen in Figure 5.8. There is an initial drop in the number of missed regions, but after that, the number of missed regions keeps rising. Note that this figure shows the number of missed regions for the dataset with the largest minimum in the separation of 6 pixels. The reason this happens is because of the connected component labeling that is discussed in section 4.2. When two regions start to touch each other, the connected component algorithm sees it as a single region, this is also the reason that a minimal increase in positional error can be noted from Table 5.5. The problem can be seen in Figure 5.9. The combination of the Gaussian and Laplacian of Gaussian layer makes the original MNIST digits slightly larger even to the point where they touch. The peak based region proposal network has no problems with them because two separate peaks are generated. But the connected component algorithm sees this as a single region.

	Positional Error in Pixels		
Runtime	**Rate Encoding**	**Rate Encoding with Random Offset**	**Poisson Rate Encoding**
20	0.66	0.63	0.71
30	0.63	0.60	0.63
40	0.64	0.61	0.63
50	0.65	0.64	0.64
60	0.68	0.65	0.65
70	0.68	0.67	0.65
80	0.69	0.67	0.66
90	0.70	0.68	0.66
100	0.70	0.68	0.67

Table 5.5: Table showing a comparison of the positional error for the full_test_set_6 dataset with the three different encoding schemes and with different runtimes using the Center of Mass Based Proposal network.

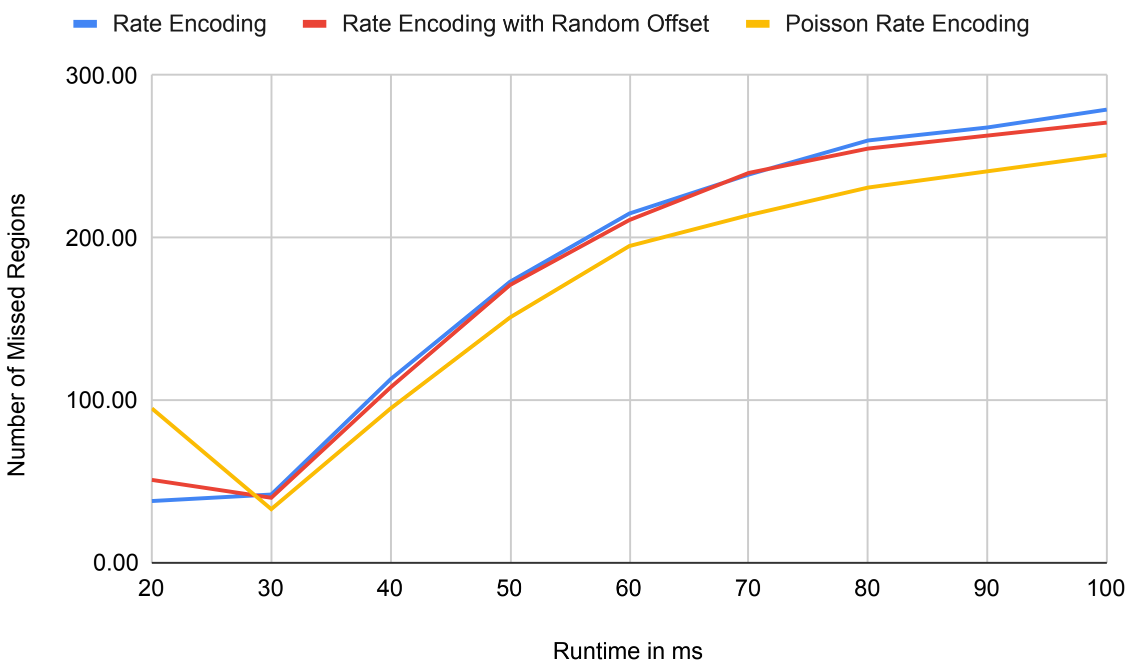
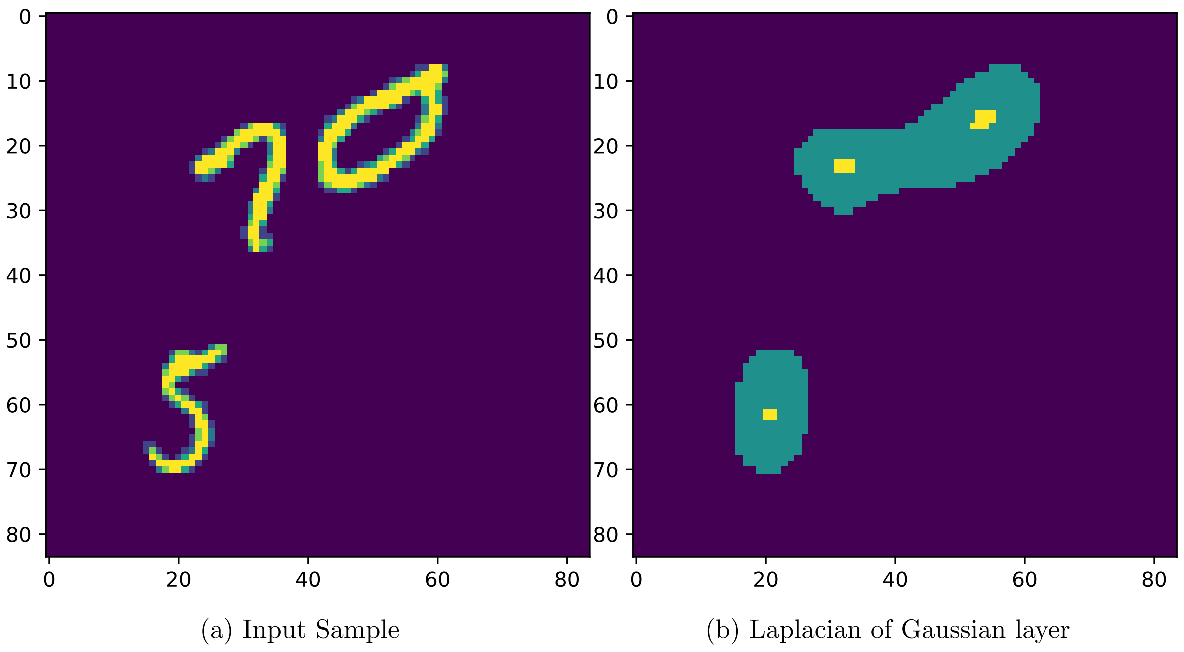


Figure 5.8: Chart showing the number of missed regions plotted against the runtime in ms for the full_test_set_6 dataset for the Center of Mass Based Region Proposal Network.



(a) Input Sample

(b) Laplacian of Gaussian layer

Figure 5.9: Figure showing two regions touching after the Laplacian of Gaussian layer. The connected component labeling sees it as a single region. This results in the Center of Mass based decoder outputting a single region while the Peak based decoder is able to output two separate regions.

5.2.3 Comparison

As can be concluded from the previous two sections, the Center of Mass-based regions are slightly more accurate. With a 0.2 pixel difference between the two in favour of the Center of Mass decoder. However, as shown in Figure 5.10, there is a large difference between the number of regions the Center of Mass decoder misses but the Peak Based network catches. With the overlapping set, there are more samples in the dataset that already touch or are very close together in the samples, increasing the problem of the connected component labeling. The Peak Based Region Proposal network can handle these situations better.

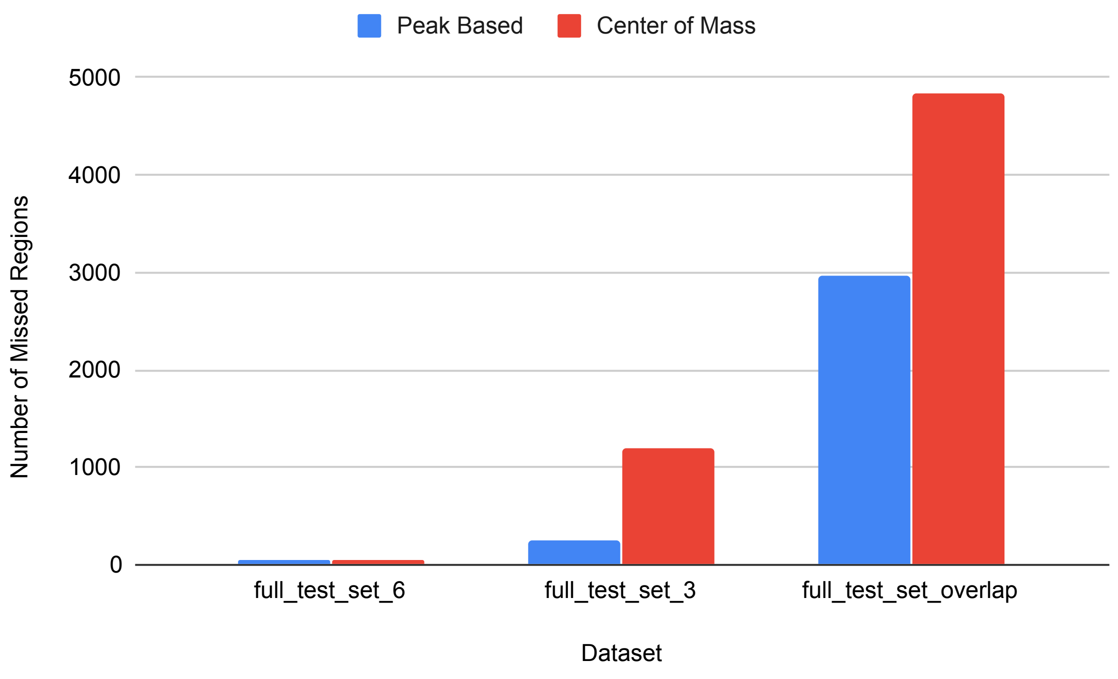


Figure 5.10: Chart showing the number of missed regions compared between the Peak Based and Center of Mass based region proposal, with the three different datasets after 30ms of runtime which is optimal for the Center Of Mass decoder.

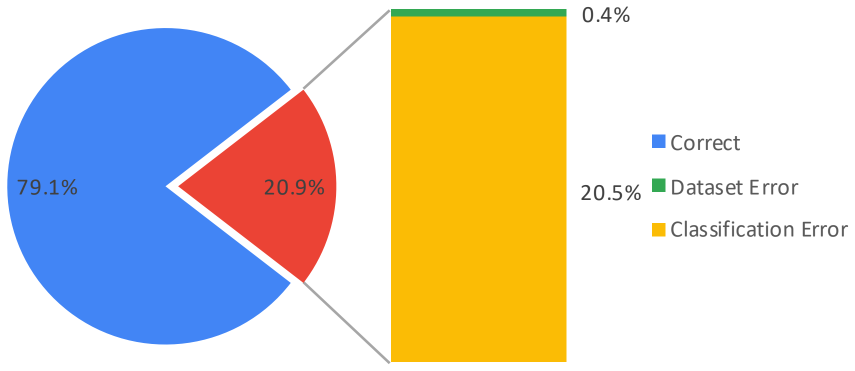


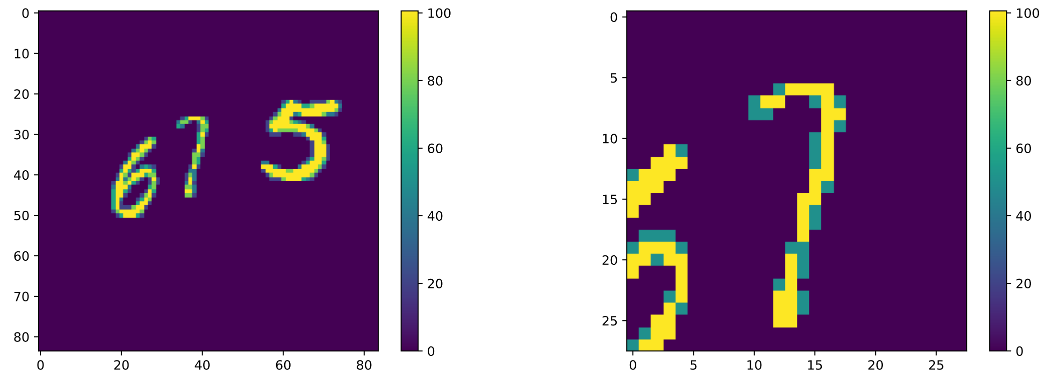
Figure 5.11: Chart showing baseline performance of the full system. The error is divided into an error coming from the classifier and one that comes from the dataset.

5.3 Full System Performance

In this section, the full system performance will be discussed. First, the system performance when using the regions that are stored within the dataset is analyzed. To see if there is a performance degradation here coming from the dataset and to gauge the performance of the region locators. After that, the Peak based and Center of Mass-based region proposal networks will be analyzed. All data from the tests are available in Appendix D. The data in Appendix D also shows that there is almost no performance difference between the three different encoding schemes. In both the MNIST results and the region locator results, the same behaviour is seen. To eliminate the random variable from the equation, the full system performance analysis will be done using the rate encoding. From the three encoding schemes, this is the only one without any random spike generation. The overlapping dataset will only be used in the true region to show that this only adds a greater error coming from the dataset. The classifier is not capable of classifying overlapping digits.

5.3.1 True Region Performance

Using the ground truth as region information in this network, also called the true regions, the system performance can be analyzed without the uncertainty of the region proposal network. This way, a theoretical optimal performance metric can be set. The original MNIST classifier performance, without the region proposal network, had a performance of 79.5% for the rate encoded input. As can be seen from Table 5.6, the performance dropped to 79.1% by going from MNIST to Multi MNIST. Figure 5.11 shows how this error is divided. 20.5% is coming from the classifier incorrectly classifying the digits. This is done by comparing the indexes of the original MNIST digits



(a) System Input

(b) Region sent to classifier

Figure 5.12: Figure showing the input to the classifier where another region is overlapping. These overlapping region causes misclassifications within the full system. The values shown in the picture are the spike rate in hertz.

and the Multi MNIST dataset and determining which digits the classifier is not able to classify. The other 0.4% must be coming from somewhere else. The positional accuracy is validated by comparing the spike trains between the normal MNIST classifier and the network that included the region selection. The classifier outputs a wrong label because, within the dataset, there are regions that are very close to each other and are included in the region that is being sent to the classifier. This error will be called the dataset error. It is an error that is not coming from positional inaccuracy or misclassification but from the generated dataset itself. It is likely that with a more complex multi-layer classifier, this would not be a problem at all.

Table 5.6 also shows the lower performance when using the dataset where overlap is allowed. The classifier is not able to differentiate two separate digits even if the exact location is known. Both the classifier, as well as the region proposal network, cannot handle overlapping digits like discussed in the sections above. The performance is only slightly lower, but because of the overlapping regions, it is impossible to tell if the error is coming from the region proposal network or the classifier. The dataset with overlapping regions will be omitted from the full system performance analysis because of this reason.

Dataset	**Classification Performance**
full_test_set_6	79.09%
full_test_set_3	79.01%
full_test_set_overlap	75.64%

Table 5.6: Table showing classification performance when using the true regions included within the dataset.

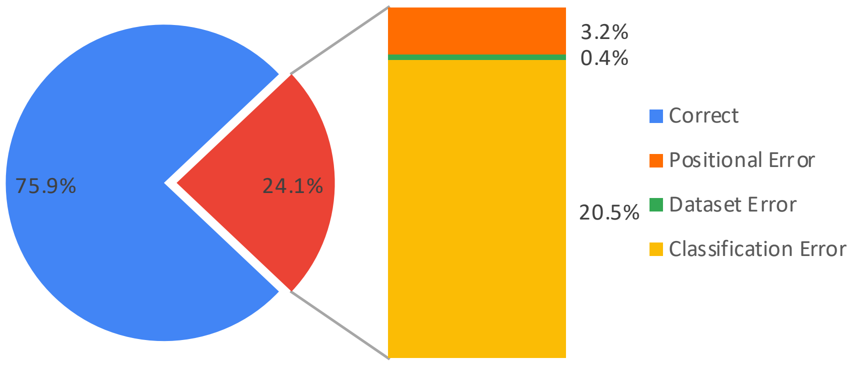


Figure 5.13: Chart showing the full system performance when using the Peak based regions.

5.3.2 Peak and Center of Mass Based Performance

With the region proposal network, a positional inaccuracy is added to the total error as well. Of all classified regions, only 3.2% is inaccurately classified because of missing regions or positional inaccuracies with the Peak based region proposal network. Figure 5.13 shows how to error is divided for this network. The classifier is still the largest contributor to the total error. The classification performance of this network is now 75.9%

Dataset	Classification Performance
full_test_set_6	75.68%
full_test_set_3	73.02%

Table 5.7: Table showing classification performance when using Peak based region proposals.

The performance of the Center of Mass-based region locator is comparable to the Peak based one. Table 5.8 shows the total classification score for the Center of Mass-based region network. There is a slight decrease in the percentage of error coming from positional inaccuracies, as is being shown by Figure 5.14. The differences in performances between these two networks will be discussed in the next section (see subsection 5.3.3).

Dataset	Classification Performance
full_test_set_6	77.34%
full_test_set_3	73.35%

Table 5.8: Table show classification performance when using Center of Mass based region proposals.

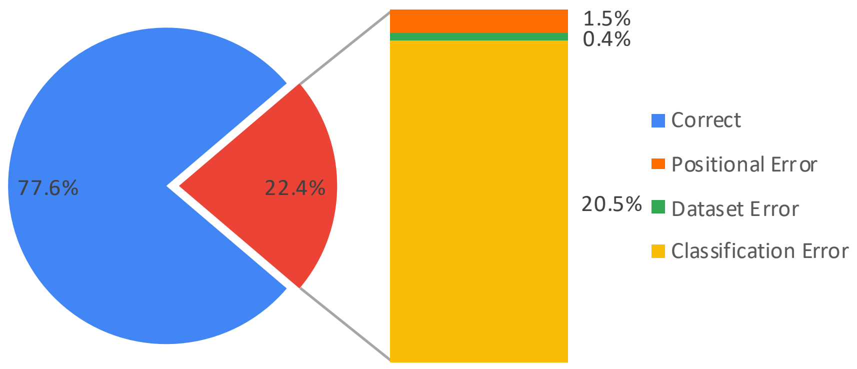


Figure 5.14: Chart showing the full system performance when using the Center of Mass based regions.

5.3.3 Comparison

Section 5.2 already showed that the Center of Mass-based proposals were slightly more positional accurate. This is also shown in the performance. The Center of Mass-based region proposal network slightly outperforms the Peak based one. In total, only 1.5% is misclassified due to a positional error using the Center of Mass method compared to 3.2% when using the Peak Based method. While the error is small, the difference is notable. Figure 5.15 shows the percentage of the total error coming from positional inaccuracies. Here one can see that the Center of Mass-based locator outperforms the Peak based one on both datasets. Even on the dataset of a minimum enforced separation of 3 pixels. In section 5.2, the loss of regions when using a closer separation was discussed for the Center of Mass method. The regions the Center of Mass based method loses because of the closely placed MNIST digits are compensated in the rest of the dataset by a better positional accuracy. Although the difference between the two networks for the 3 pixels separated Multi MNIST dataset is minimal.

What both methods have in common is the low region proposal count. Both methods, on average, propose the minimal number of regions that are within the dataset. A sliding window with no stride for this input would generate 3136 regions, which would be a significant performance hit on the system. Comparing to R-CNN or YOLO is not truely fair because both systems use a different input size and different type of dataset. However, it is known that R-CNN generates around 2000 regions and YOLO about 98. It is unknown how these systems would perform with a more blob-like dataset. Nevertheless, both R-CNN and YOLO implement some form of region pooling where overlapping regions are merged into one, and the label with the highest prediction score is picked. With both the Peak Based as the Center of Mass-based networks, this is not necessary. The Center of Mass-based network never proposed an overlapping region,

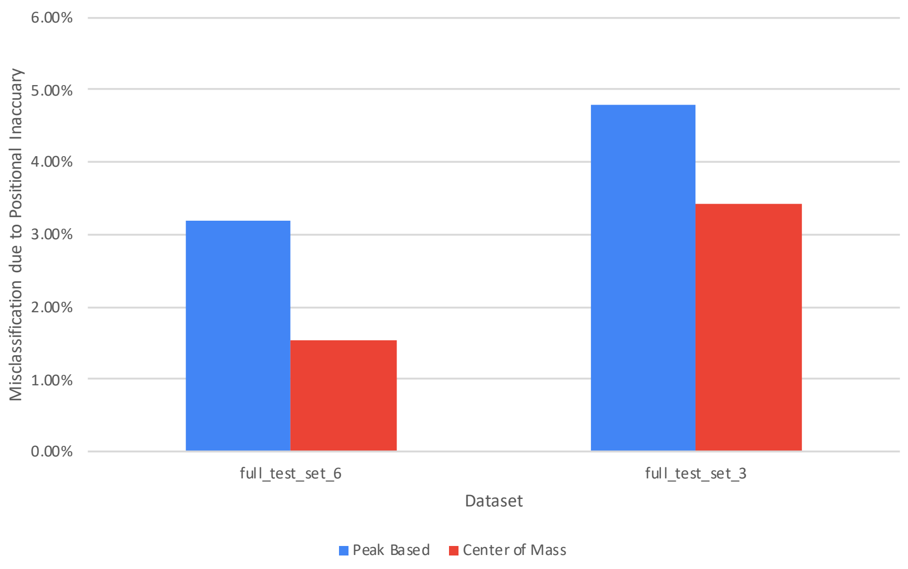


Figure 5.15: Chart showing a comparison between the percentage of the error coming from positional inaccuracies.

and the Peak based did this in 0.04% of the cases, which is neglectable.

The data shows that a network using a Laplacian of Gaussian based blob detector is a viable way of doing multi-object detection and classification of blob-like structures. The decision between a Peak or a Center of Mass-based decoder should be based on the minimum separation of the regions within the incoming data as well as the possible computer power available. The Center of Mass-based network has to make more loops over the rate decoded data. This is due to the connected component labeling, which already needs to do two passes over the data. For every region, the Center of Mass has to be calculated while the Peak based network is able to detect all the peaks with only two passes of the rate data regardless of the number of regions.

Conclusion 6

In this study, a promising architecture is proposed to detect and classify multiple regions within a particular input. The architecture is inspired upon state of the art architectures that are currently being used with Convolutional Neural Networks but uses the adaptability of the synaptic connections to steer data towards the classifier. The architecture consists of three main parts the region proposal, weight calculations, and the classifier.

In order to experiment with blob-like data that has similarities with radar data, this study proposed and implemented a dataset generator. This generator is able to take a dataset of handwritten digits called MNIST and generate datasets consisting of multiple samples of these digits within a single sample of the output. One is able to control the minimum separation between digits, the minimum and maximum amount of digits in the output, but also scale and rotation. This generator enables further research with blob-like multiple object data where one could also compensate for scale and rotation.

The study proposes and implements a region proposal network that can locate blob-like regions within its input. The region proposal network consisting of Gaussian and a Laplacian of Gaussian layer within a Spiking Neural Network which can accurately predict the location of blobs with sub-pixel precision on all tested datasets. The implemented region proposal network also does not create an abundance of regions like other region proposal networks do, and thus no region pooling is required.

A full implementation of the architecture is created and analyzed. The proposed region proposal network is implemented together with an off the shelf MNIST classifier to show the capabilities of the architecture and the region proposal network. The data shows that the architecture is a viable solution to solve a multi-object detection and classification problem. When using the Peak based regions, only 3.19% of the regions are misclassified due to the region proposal network.

Two types of rate decoders are being discussed in the implementation to show performance differences. A Peak-based decoder shows promising results for the least amount of computing power of the two. While the positional accuracy is still high, the Center of Mass-based shows a slightly better performance. Due to the classifier not being positional invariant, only a slight shift of the input results in performance degradation. This is reflected in the full system results as well. At the same time, the Center of Mass-based decoder cannot cope with regions that are close together.

7 Future work

While the study proves that the architecture is a viable idea and shows promising results, Spiking Neural Networks truly thrive in terms of power consumption with a hardware implementation. The architecture uses weights that dynamically change during runtime. While it is known that the hardware implementation requires that the weights are changeable, otherwise there it would be a fixed network, the consequence of doing this while running are not fully understood.

A hardware implementation where the input has a special synaptic connection with the first layer of neurons could also be investigated. These special synaptic connections would only consist of a gated connection between the input and the first layer of neurons. A region proposal network could then turn these synaptic connections on or off. This could potentially be faster than rewriting the synaptic weights.

In order to do more research towards blob-like region proposal and classification, a database that reflects a real-life problem should be created. The author believes that high quality and complete radar datasets could profoundly aid future research into possible use cases of the region locator.

As discussed, the Laplacian of Gaussian has more capabilities than just detecting round blobs. By using multiple parallel layers of the Laplacian of Gaussian multiple scales as well as rotation could be detected. By using these characteristics, a region proposal network could be created that could aid the classifier with scale as well as rotation invariance.

Bibliography

[1] F. Bre, J. M. Gimenez, and V. D. Fachinotti, “Prediction of wind pressure coefficients on building surfaces using artificial neural networks,” *Energy and Buildings*, vol. 158, no. November, pp. 1429–1441, 2018.

[2] D. Baillot, “Why are neuron axons long and spindly? Study shows they’re optimizing signaling efficiency,” *Medical X Press*, no. July, pp. 11–13, 2018. [Online]. Available: https://medicalxpress.com/news/2018-07-neuron-axons-spindly-theyre-optimizing.html%0A

[3] J. M. Bower, D. Beeman, M. Nelson, and J. Rinzel, “The Hodgkin-Huxley Model,” *The Book of GENESIS*, pp. 29–51, 1995.

[4] University of Maryland School of Medicine, “Synapses and Circuits.” [Online]. Available: https://lifesciences.umaryland.edu/neuroscience/Research-Focus-Groups/Synapses--Circuits/

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 6 2014, pp. 580–587. [Online]. Available: http://ieeexplore.ieee.org/document/6909475/

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2016-Decem. IEEE, 6 2016, pp. 779–788. [Online]. Available: http://ieeexplore.ieee.org/document/7780460/

[7] U. Chipengo, “Full Physics Simulation Study of Guardrail Radar-Returns for 77 GHz Automotive Radar Systems,” *IEEE Access*, vol. 6, no. September, pp. 70 053–70 060, 2018.

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 9 2014. [Online]. Available: http://arxiv.org/abs/1409.0575

[9] Y. Haiqian and M. Leeser, “Optimizing data intensive window-based image processing on reconfigurable hardware boards,” *IEEE Workshop on Signal Processing Systems, SiPS: Design and Implementation*, vol. 2005, pp. 491–496, 2005.

[10] H. Kong, H. C. Akakin, and S. E. Sarma, “A generalized laplacian of gaussian filter for blob detection and its applications,” *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 1719–1733, 2013.

[11] B. Wang, W. Shi, and Z. Miao, “Confidence analysis of standard deviational ellipse and its extension into higher dimensional Euclidean space,” *PLoS ONE*, vol. 10, no. 3, 2015.

[12] I. C. McManus, K. Stöver, and D. Kim, “Arnheim’s Gestalt Theory of Visual Balance: Examining the Compositional Structure of Art Photographs and Abstract Images,” *i-Perception*, vol. 2, no. 6, pp. 615–647, 8 2011. [Online]. Available: http://journals.sagepub.com/doi/10.1068/i0445aap

[13] Mathworks, “Pixel Connectivity,” 2003. [Online]. Available: https://www.mathworks.com/help/releases/R13sp2/toolbox/images/morph12.html

[14] Synopsys, “MAX-Pooling embARC Machine Learning Inference Library 1.00 documentation,” 2019. [Online]. Available: https://embarc.org/embarc_mli/doc/build/html/MLI_kernels/pooling_max.html

[15] L. Steffen, D. Reichard, J. Weinland, J. Kaiser, A. Roennau, and R. Dillmann, “Neuromorphic stereo vision: A survey of bio-inspired sensors and algorithms,” *Frontiers in Neurorobotics*, vol. 13, no. May, pp. 1–20, 2019.

[16] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 12 1943. [Online]. Available: http://link.springer.com/10.1007/BF02478259

[17] C. Mead, “Neuromorphic electronic systems,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990. [Online]. Available: http://ieeexplore.ieee.org/document/58356/

[18] R. Douglas, M. Mahowald, and C. Mead, “Neuromorphic Analogue VLSI,” *Annual Review of Neuroscience*, vol. 18, no. 1, pp. 255–281, 3 1995. [Online]. Available: http://neuro.annualreviews.org/cgi/doi/10.1146/annurev.neuro.18.1.255http://www.ncbi.nlm.nih.gov/pubmed/7605063http://www.annualreviews.org/doi/10.1146/annurev.ne.18.030195.001351

[19] TOP500, “June 2019 — TOP500 Supercomputer Sites,” 2019. [Online]. Available: https://www.top500.org/lists/2019/06/

[20] J. Wu, *Introduction to neural dynamics and signal transmission delay.* Walter de Gruyter, 2011.

[21] S. Herculano-Houzel, “The human brain in numbers: A linearly scaled-up primate brain,” *Frontiers in Human Neuroscience*, vol. 3, no. NOV, pp. 1–11, 2009.

[22] D. A. M. D. Drachman, “Do we have brain to spare? [Editorial],” *Neurology June 28, 2005;64(12):2004-2005*, vol. 64, no. 12, pp. 2004–2005, 2005.

[23] A. L. Hodgkin and A. F. Huxley, “A quantitative description of membrane current and its application to conduction and excitation in nerve,” *The Journal of Physiology*, vol. 117, no. 4, pp. 500–544, 8 1952. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1952.sp004764

[24] ——, “Resting and action potentials in single nerve fibres,” *The Journal of Physiology*, vol. 104, no. 2, pp. 176–195, 10 1945. [Online]. Available: http://doi.wiley.com/10.1113/jphysiol.1945.sp004114

[25] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. van Schaik, R. Etienne-Cummings, T. Delbruck, S. C. Liu, P. Dudek, P. Häfliger, S. Renaud, J. Schemmel, G. Cauwenberghs, J. Arthur, K. Hynna, F. Folowosele, S. Saighi, T. Serrano-Gotarredona, J. Wijekoon, Y. Wang, and K. Boahen, "Neuromorphic silicon neuron circuits," *Frontiers in Neuroscience*, vol. 5, no. MAY, pp. 1–23, 2011.

[26] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[27] L. F. Abbott, "Lapicque's introduction of the integrate-and-fire model neuron (1907)," *Brain Research Bulletin*, vol. 50, no. 5-6, pp. 303–304, 1999.

[28] E. D. Adrian and Y. Zotterman, "The impulses produced by sensory nerveendings: Part II. The response of a Single EndOrgan," *The Journal of Physiology*, vol. 61, no. 2, pp. 151–171, 1926.

[29] M. Stimberg, D. F. Goodman, V. Benichoux, and R. Brette, "Brian 2 - the second coming: spiking neural network simulation in Python with code generation," *BMC Neuroscience*, vol. 14, no. S1, p. 2202, 2013.

[30] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[31] N. Dalal, B. Triggs, N. Dalal, B. Triggs, O. Gradients, D. International, S. Diego, N. Dalal, and B. Triggs, "Histograms of Oriented Gradients for Human Detection To cite this version : HAL Id : inria-00548512 Histograms of Oriented Gradients for Human Detection," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 886–893, 2010.

[32] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," *Proceedings of the ACM International Conference on Multimedia - MM '14*, pp. 675–678, 6 2014. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2647868.2654889http://arxiv.org/abs/1408.5093

[33] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 9 1995. [Online]. Available: http://link.springer.com/10.1007/BF00994018

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2016-Decem. IEEE, 6 2016, pp. 770–778. [Online]. Available: http://ieeexplore.ieee.org/document/7780459/

[35] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going Deeper in Spiking Neural Networks: VGG and Residual Architectures," *Frontiers in Neuroscience*, vol. 13, no. 1998, pp. 1–16, 2019.

[36] LeCun Yann, Cortes Corinna, and Burges Christopher, "THE MNIST DATABASE of handwritten digits," *The Courant Institute of Mathematical Sciences*, 1998.

[37] E. Hunsberger and C. Eliasmith, “Spiking Deep Networks with LIF Neurons,” pp. 1–9, 10 2015. [Online]. Available: http://arxiv.org/abs/1510.08829

[38] D. Rumelhart, G. Hinton, and R. Williams, “Learning Internal Representations by Error Propagation,” in *Readings in Cognitive Science.* Elsevier, 1988, pp. 399–421. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9781483214467500352

[39] C. Lee, S. S. Sarwar, and K. Roy, “Enabling Spike-based Backpropagation in State-of-the-art Deep Neural Network Architectures,” pp. 1–25, 3 2019. [Online]. Available: http://arxiv.org/abs/1903.06379

[40] G.-q. Bi and M.-m. Poo, “Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type,” *The Journal of Neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 12 1998. [Online]. Available: http://www.jneurosci.org/lookup/doi/10.1523/JNEUROSCI.18-24-10464.1998

[41] P. U. Diehl and M. Cook, “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” *Frontiers in Computational Neuroscience*, vol. 9, no. August, pp. 1–9, 2015. [Online]. Available: http://journal.frontiersin.org/Article/10.3389/fncom.2015.00099/abstract

[42] M. Lind, “Simple 1-Layer Neural Network for MNIST Handwriting Recognition,” 2015. [Online]. Available: https://mmlind.github.io/Simple_1-Layer_Neural_Network_for_MNIST_Handwriting_Recognition/

[43] F. Pukelsheim, “The Three Sigma Rule,” *The American Statistician*, vol. 48, no. 2, pp. 88–91, 5 1994. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/00031305.1994.10476030

[44] E. W. Weisstein, “von Neumann Neighborhood.” [Online]. Available: http://mathworld.wolfram.com/vonNeumannNeighborhood.html

[45] ——, “Moore Neighborhood.” [Online]. Available: http://mathworld.wolfram.com/MooreNeighborhood.html

[46] A. AbuBaker, R. Qahwaji, S. Ipson, and M. Saleh, “One Scan Connected Component Labeling Technique,” in *2007 IEEE International Conference on Signal Processing and Communications*, no. November. IEEE, 2007, pp. 1283–1286. [Online]. Available: http://ieeexplore.ieee.org/document/4728561/

[47] L. Vincent, L. Vincent, and P. Soille, “Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 583–598, 6 1991. [Online]. Available: http://ieeexplore.ieee.org/document/87344/

[48] J. Hoshen and R. Kopelman, “Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm,” *Physical*

Review B, vol. 14, no. 8, pp. 3438–3445, 10 1976. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.14.3438

[49] K. Wu, E. Otoo, and A. Shoshani, "Optimizing connected component labeling algorithms," in *Medical Imaging 2005: Image Processing*, J. M. Fitzpatrick and J. M. Reinhardt, Eds., vol. 5747, 4 2005, p. 1965. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.596105

[50] M. J. Klaiber, D. G. Bailey, Y. O. Baroud, and S. Simon, "A Resource-Efficient Hardware Architecture for Connected Component Analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 7, pp. 1334–1349, 7 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7137657/

[51] Xilinx, "7 Series FPGAs Data Sheet: Overview (DS180)," vol. 180, pp. 1–18, 2010. [Online]. Available: www.xilinx.com

[52] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3642–3649, 2012.

MNIST Classifier Positional Invariance Data

A

A.1 Rate Encoded

	Percentage Correct										
Shift	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
0	94.39%	96.74%	67.54%	84.26%	82.48%	46.97%	90.61%	82.39%	66.63%	77.80%	79.49%
1	91.12%	95.77%	55.04%	73.56%	76.99%	36.21%	77.45%	84.82%	52.77%	48.86%	69.91%
2	68.06%	87.49%	36.53%	40.79%	54.38%	13.12%	47.39%	69.65%	15.30%	7.14%	44.91%
3	27.65%	72.69%	13.66%	9.41%	20.06%	2.47%	24.74%	32.78%	1.95%	0.40%	21.48%
4	5.41%	61.23%	3.78%	1.78%	3.67%	0.45%	20.04%	8.07%	0.31%	0.20%	11.25%
5	0.71%	53.92%	0.87%	1.29%	0.20%	0.22%	26.20%	1.26%	0.10%	0.10%	9.11%
6	0.00%	50.13%	0.48%	1.78%	0.00%	0.67%	32.78%	0.00%	0.21%	0.10%	9.15%
7	0.00%	43.70%	0.87%	2.87%	0.00%	0.22%	31.63%	0.19%	0.10%	0.30%	8.45%
8	0.00%	37.71%	2.62%	2.08%	0.00%	0.11%	30.27%	0.49%	0.00%	0.00%	7.72%
9	0.00%	32.16%	5.91%	4.46%	0.00%	0.11%	41.34%	1.17%	0.21%	0.00%	8.82%

Table A.1: Table showing positional invariance of the MNIST classifier using Rate encoding.

A.2 Rate Encoding with Random Offset

	Percentage Correct										
Shift	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
0	93.47%	96.30%	68.31%	83.96%	82.08%	47.31%	91.65%	82.78%	71.05%	80.28%	80.21%
1	89.59%	95.24%	54.75%	72.57%	74.44%	37.11%	79.23%	85.99%	54.31%	53.91%	70.35%
2	64.69%	86.61%	33.62%	38.32%	51.73%	12.78%	49.90%	71.40%	15.71%	8.72%	44.26%
3	23.27%	69.25%	12.79%	8.02%	19.14%	1.46%	26.93%	34.14%	2.77%	0.40%	20.68%
4	3.57%	58.94%	3.29%	1.29%	2.55%	0.11%	24.01%	8.75%	0.41%	0.10%	11.02%
5	0.41%	52.07%	0.87%	0.50%	0.00%	0.22%	31.63%	1.07%	0.10%	0.10%	9.27%
6	0.00%	47.14%	0.48%	0.99%	0.00%	0.56%	35.59%	0.19%	0.21%	0.10%	9.01%
7	0.00%	41.59%	0.87%	0.89%	0.00%	0.22%	33.30%	0.19%	0.10%	0.30%	8.17%
8	0.00%	34.19%	1.74%	0.59%	0.00%	0.00%	31.21%	0.68%	0.00%	0.10%	7.19%
9	0.00%	28.99%	4.75%	1.68%	0.00%	0.11%	43.11%	0.97%	0.21%	0.00%	8.21%

Table A.2: Table showing positional invariance of the MNIST classifier using Rate encoding with a random 0 to 10 ms offset.

A.3 Poisson Rate Encoded

	Percentage Correct										
Shift	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
0	92.24%	95.15%	66.18%	81.68%	79.63%	47.09%	90.61%	81.61%	67.25%	78.00%	78.43%
1	86.63%	94.71%	52.62%	70.59%	71.89%	35.31%	75.78%	85.02%	50.31%	50.15%	67.97%
2	61.73%	84.93%	33.14%	36.63%	47.66%	12.78%	48.43%	68.58%	17.76%	7.93%	42.85%
3	22.96%	68.46%	13.08%	7.92%	18.94%	2.02%	26.72%	30.93%	2.98%	0.89%	20.33%
4	4.59%	57.97%	3.78%	1.49%	3.56%	0.45%	23.28%	8.75%	1.03%	0.20%	11.21%
5	0.92%	51.89%	0.78%	0.50%	0.61%	0.22%	28.81%	1.07%	0.51%	0.10%	9.12%
6	0.00%	47.75%	0.48%	0.99%	0.00%	0.22%	33.51%	0.29%	0.41%	0.10%	8.88%
7	0.00%	41.32%	0.78%	1.09%	0.00%	0.11%	31.21%	0.39%	0.21%	0.30%	7.97%
8	0.00%	32.51%	1.84%	0.99%	0.00%	0.00%	27.66%	1.07%	0.10%	0.20%	6.77%
9	0.00%	26.34%	4.26%	2.28%	0.00%	0.22%	39.98%	0.97%	0.41%	0.10%	7.66%

Table A.3: Table showing positional invariance of the MNIST classifier using Poisson rate encoding.

B Region Proposal Positional Data

B.1 Peak Based Region Proposal

B.1.1 Rate Encoded

B.1.1.1 Dataset: full_test_set_3

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	0	20073	20073	59.8374	495.2148	nan
30	0	966	20073	270	831	0.89
40	3	243	20073	482	1195	0.93
50	16	138	20073	659	1570	1.05
60	13	98	20073	893	1936	1.02
70	14	124	20073	1082	2290	0.99
80	16	109	20073	1259	2646	1.08
90	15	122	20073	1516	3014	1.02
100	18	116	20073	1705	3393	1.04

Table B.1: Peak Based region proposal network with the full_test_set_3 dataset rate encoded. Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.1.1.2 Dataset: full_test_set_6

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	0	682	20023	283	829	0.79
30	0	44	20023	504	1193	0.79
40	19	4	20023	683	1568	0.91
50	0	1	20023	925	1933	0.90
60	3	2	20023	1116	2287	0.86
70	8	1	20023	1301	2641	0.95
80	5	1	20023	1562	3009	0.90
90	9	3	20023	1758	3388	0.92
100	11	2	20023	2025	3771	0.94

Table B.2: Peak Based region proposal network with the full_test_set_6 dataset rate encoded. Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.1.1.3 Dataset: full_test_set_overlap

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	1	4122	20029	261	796	1.55
30	9	2971	20029	461	1146	1.56
40	26	2733	20029	624	1505	1.59
50	15	2881	20029	846	1856	1.55
60	10	2881	20029	1021	2195	1.48
70	16	2854	20029	1191	2537	1.50
80	15	2930	20029	1428	2890	1.44
90	18	2900	20029	1608	3254	1.47
100	15	2962	20029	1849	3622	1.43

Table B.3: Peak Based region proposal network with the full_test_set_overlap dataset rate encoded. Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.1.2 Rate Encoding with Random Offset

B.1.2.1 Dataset: full_test_set_3

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	1	836	20073	267	817	0.82
30	1	390	20073	440	1177	0.93
40	14	151	20073	635	1533	1.00
50	14	112	20073	836	1890	0.94
60	13	138	20073	995	2236	0.97
70	12	117	20073	1229	2606	0.98
80	7	133	20073	1412	2968	0.95
90	18	123	20073	1619	3337	1.00
100	12	134	20073	1825	3704	0.98

Table B.4: Peak Based region proposal network with the full_test_set_3 dataset rate encoding with a random 0-10ms offset. Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.1.2.2 Dataset: full_test_set_6

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	0	519	20023	282	816	0.71
30	1	52	20023	458	1177	0.75
40	19	7	20023	659	1532	0.85
50	7	3	20023	865	1889	0.80
60	7	1	20023	1028	2234	0.83
70	5	1	20023	1269	2603	0.86
80	8	2	20023	1457	2965	0.83
90	11	2	20023	1670	3334	0.88
100	3	2	20023	1882	3701	0.87

Table B.5: Peak Based region proposal network with the full_test_set_6 dataset rate encoding with a random 0-10ms offset. Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.1.2.3 Dataset: full_test_set_overlap

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	1	4119	20029	261	788	1.51
30	6	3366	20029	423	1136	1.65
40	23	2972	20029	608	1477	1.59
50	19	3135	20029	796	1819	1.54
60	14	2958	20029	942	2152	1.51
70	14	3007	20029	1165	2508	1.48
80	12	3022	20029	1334	2857	1.47
90	17	3049	20029	1530	3211	1.47
100	8	3036	20029	1722	3564	1.45

Table B.6: Peak Based region proposal network with the full_test_set_overlap dataset rate encoding with a random 0-10ms offset. Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.1.3 Poisson Rate Encoded

B.1.3.1 Dataset: full_test_set_3

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	0	1341	20073	231	746	0.91
30	1	426	20073	390	1082	0.92
40	8	207	20073	562	1399	1.07
50	13	102	20073	751	1733	0.99
60	10	120	20073	907	2067	0.99
70	10	111	20073	1124	2404	1.02
80	13	124	20073	1294	2746	0.98
90	16	122	20073	1488	3088	1.03
100	9	131	20073	1671	3422	1.01

Table B.7: Peak Based region proposal network with the full_test_set_3 dataset poisson rate encoded. Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.1.3.2 Dataset: full_test_set_6

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	1	1029	20023	245	745	0.83
30	1	97	20023	405	1082	0.76
40	9	13	20023	584	1398	0.94
50	15	3	20023	778	1732	0.86
60	6	2	20023	938	2065	0.87
70	4	1	20023	1162	2402	0.91
80	4	1	20023	1336	2744	0.86
90	3	0	20023	1536	3085	0.92
100	9	1	20023	1724	3418	0.90

Table B.8: Peak Based region proposal network with the full_test_set_6 dataset poisson rate encoded. Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.1.3.3 Dataset: full_test_set_overlap

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	1	4438	20029	229	722	1.56
30	3	3754	20029	377	1048	1.68
40	18	2991	20029	544	1355	1.69
50	17	3182	20029	719	1675	1.59
60	14	3089	20029	865	1996	1.57
70	6	3065	20029	1070	2321	1.54
80	9	3116	20029	1228	2651	1.52
90	14	3042	20029	1412	2981	1.51
100	9	3095	20029	1583	3302	1.49

Table B.9: Peak Based region proposal network with the full_test_set_overlap dataset poisson rate encoded. Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.2 Center of Mass Based Region Proposal

B.2.1 Rate Encoded

B.2.1.1 Dataset: full_test_set_3

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	0	444	20073	482	1195	0.88
30	0	1193	20073	1082	2290	1.14
40	0	1684	20073	1966	3776	1.33
50	0	1906	20073	3023	5589	1.44
60	0	2034	20073	4264	7763	1.50
70	0	2105	20073	5742	10313	1.54
80	0	2165	20073	7425	13203	1.57
90	0	2191	20073	9283	16459	1.58
100	0	2207	20073	11391	20114	1.58

Table B.10: Center of Mass region proposal network with the full_test_set_3 dataset rate encoded. Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.2.1.2 Dataset: full_test_set_6

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	0	38	20023	504	1193	0.66
30	0	42	20023	1116	2287	0.63
40	0	113	20023	2025	3771	0.64
50	0	173	20023	3110	5582	0.65
60	0	215	20023	4384	7753	0.68
70	0	239	20023	5900	10300	0.68
80	0	260	20023	7628	13187	0.69
90	0	268	20023	9535	16439	0.70
100	0	279	20023	11699	20090	0.70

Table B.11: Center of Mass region proposal network with the full_test_set_6 dataset rate encoded. Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.2.1.3 Dataset: full_test_set_overlap

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	0	4163	20029	461	1146	1.68
30	0	4828	20029	1021	2195	1.93
40	0	5136	20029	1849	3622	2.06
50	0	5255	20029	2839	5361	2.12
60	0	5324	20029	4001	7446	2.16
70	0	5367	20029	5385	9892	2.18
80	0	5395	20029	6960	12666	2.20
90	0	5407	20029	8703	15791	2.21
100	0	5419	20029	10675	19297	2.21

Table B.12: Center of Mass region proposal network with the full_test_set_overlap dataset rate encoded. Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.2.2 Rate Encoding with Random Offset

B.2.2.1 Dataset: full_test_set_3

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	0	472	20073	440	1177	0.85
30	0	1149	20073	995	2236	1.10
40	0	1642	20073	1825	3704	1.31
50	0	1899	20073	2827	5518	1.44
60	0	2005	20073	4021	7686	1.49
70	0	2098	20073	5425	10229	1.54
80	0	2142	20073	7025	13131	1.56
90	0	2165	20073	8829	16401	1.58
100	0	2187	20073	10833	20035	1.59

Table B.13: Center of Mass region proposal network with the full_test_set_3 dataset rate encoding with a random 0-10ms offset Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.2.2.2 Dataset: full_test_set_6

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	0	51	20023	458	1176	0.63
30	0	40	20023	1028	2235	0.60
40	0	108	20023	1882	3701	0.61
50	0	171	20023	2912	5514	0.64
60	0	211	20023	4140	7680	0.65
70	0	240	20023	5582	10220	0.67
80	0	255	20023	7227	13120	0.67
90	0	263	20023	9081	16387	0.68
100	0	271	20023	11142	20018	0.68

Table B.14: Center of Mass region proposal network with the full_test_set_6 dataset rate encoding with a random 0-10ms offset Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.2.2.3 Dataset: full_test_set_overlap

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	0	4150	20029	424	1136	1.66
30	0	4783	20029	942	2152	1.91
40	0	5100	20029	1722	3565	2.06
50	0	5244	20029	2661	5310	2.14
60	0	5318	20029	3781	7395	2.18
70	0	5360	20029	5095	9840	2.20
80	0	5384	20029	6594	12631	2.22
90	0	5398	20029	8284	15776	2.23
100	0	5407	20029	10162	19271	2.23

Table B.15: Center of Mass region proposal network with the full_test_set_overlap dataset rate encoding with a random 0-10ms offset Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.2.3 Poisson Rate Encoded

B.2.3.1 Dataset: full_test_set_3

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	1	437	20073	1082	390	0.88
30	0	1039	20073	2067	907	1.08
40	0	1550	20073	3422	1671	1.29
50	0	1812	20073	5107	2609	1.40
60	0	1953	20073	7140	3734	1.47
70	0	2024	20073	9510	5047	1.50
80	0	2079	20073	12206	6542	1.53
90	0	2111	20073	15234	8219	1.54
100	0	2144	20073	18603	10087	1.56

Table B.16: Center of Mass region proposal network with the full_test_set_3 dataset poisson rate encoded. Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.2.3.2 Dataset: full_test_set_6

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	1	95	20023	1082	405	0.71
30	0	33	20023	2065	938	0.63
40	0	95	20023	3418	1724	0.63
50	0	151	20023	5103	2688	0.64
60	0	195	20023	7134	3846	0.65
70	0	214	20023	9502	5196	0.65
80	0	231	20023	12197	6734	0.66
90	0	241	20023	15223	8459	0.66
100	0	251	20023	18589	10380	0.67

Table B.17: Center of Mass region proposal network with the full_test_set_6 dataset poisson rate encoded. Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

B.2.3.3 Dataset: full_test_set_overlap

				Average Number of Spikes		
Runtime	**Extra Regions**	**Missed Regions**	**Total Regions**	**LoG Layer**	**Gaussian Layer**	**Average Error in Pixels**
20	0	4059	20029	1048	377	1.66
30	0	4707	20029	1996	865	1.91
40	0	5032	20029	3302	1583	2.04
50	0	5194	20029	4925	2463	2.12
60	0	5287	20029	6885	3521	2.17
70	0	5330	20029	9168	4754	2.19
80	0	5357	20029	11766	6158	2.20
90	0	5374	20029	14684	7734	2.22
100	0	5385	20029	17931	9489	2.22

Table B.18: Center of Mass region proposal network with the full_test_set_overlap dataset poisson rate encoded. Table showing the number extra regions, missed regions and total regions in the dataset. As well as the number of spikes in a certain layer and the positional error expressed in the amount of pixels of from the truth.

Region Proposal Missed Digit Data C

C.1 Peak Based Missed Regions

		Number of Missed Regions									
Dataset	**Encoding**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
full_test_set_3	Rate	11	120	14	15	7	14	11	31	2	18
full_test_set_3	Rate Offset	9	159	24	22	17	31	21	52	11	37
full_test_set_3	Poisson	10	194	24	26	16	25	28	52	16	35
full_test_set_6	Rate	0	41	0	0	0	2	0	1	0	0
full_test_set_6	Rate Offset	0	49	0	0	0	2	0	1	0	0
full_test_set_6	Poisson	0	88	0	1	0	4	0	3	0	1

Table C.1: Table showing the number of missed regions for the Peak Based Regions.

C.2 Center of Mass Based Missed Regions

		Number of Missed Regions									
Dataset	**Encoding**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
full_test_set_3	Rate	26	147	36	31	21	37	30	48	17	53
full_test_set_3	Rate Offset	19	167	29	34	22	37	34	57	17	52
full_test_set_3	Poisson	12	196	27	29	16	26	29	52	14	39
full_test_set_6	Rate	0	41	0	0	0	2	0	1	0	0
full_test_set_6	Rate Offset	0	46	0	0	0	2	0	0	0	0
full_test_set_6	Poisson	0	88	0	1	0	4	0	3	0	1

Table C.2: Table showing the number of missed regions for the Center of Mass based regions.

D Full System Accuracy Data

D.1 True Regions

		Accuracy										
Dataset	**Encoding**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
full_test_set_3	Rate	94.10%	96.95%	66.95%	84.18%	84.02%	47.25%	90.13%	81.18%	67.74%	77.88%	79.01%
full_test_set_3	Rate Offset	93.38%	96.65%	67.91%	83.98%	82.54%	47.45%	91.47%	81.93%	71.95%	80.15%	79.71%
full_test_set_3	Poisson	92.15%	95.75%	66.38%	80.88%	80.32%	46.65%	89.64%	80.67%	68.36%	77.48%	77.80%
full_test_set_6	Rate	94.47%	96.77%	66.85%	85.08%	83.56%	47.60%	90.65%	82.89%	66.22%	76.82%	79.09%
full_test_set_6	Rate Offset	93.42%	96.57%	67.71%	84.82%	83.66%	47.85%	91.99%	82.94%	71.78%	79.31%	80.02%
full_test_set_6	Poisson	92.78%	95.68%	65.44%	81.90%	80.50%	46.64%	90.55%	82.29%	67.31%	77.17%	78.04%

Table D.1: Table showing the classification accuracy for the full system. The accuracy shown here is using the ground truth for the region generation.

D.2 Peak Based Regions

		Accuracy										
Dataset	**Encoding**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
full_test_set_3	Rate	88.81%	90.39%	60.87%	73.53%	82.78%	42.41%	87.48%	79.11%	60.41%	64.59%	73.02%
full_test_set_3	Rate Offset	86.55%	88.19%	60.97%	71.45%	79.67%	42.71%	85.79%	79.27%	60.97%	65.53%	72.10%
full_test_set_3	Poisson	83.11%	86.09%	58.52%	69.93%	78.24%	39.56%	83.53%	77.10%	58.72%	62.77%	69.75%
full_test_set_6	Rate	92.53%	94.49%	63.43%	76.76%	84.80%	45.54%	89.00%	83.14%	61.21%	65.65%	75.68%
full_test_set_6	Rate Offset	90.28%	93.40%	63.63%	76.34%	82.63%	44.24%	89.15%	84.47%	62.24%	67.25%	75.39%
full_test_set_6	Poisson	87.05%	91.06%	60.65%	72.96%	80.21%	43.34%	86.77%	81.75%	61.36%	66.75%	73.23%

Table D.2: Table showing the classification accuracy for the full system. The accuracy shown here is using the Peak based regions.

D.3 Center of Mass Based Regions

		Accuracy										
Dataset	**Encoding**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
full_test_set_3	Rate	87.99%	89.54%	61.69%	76.17%	80.91%	44.26%	86.03%	78.66%	62.87%	65.63%	73.35%
full_test_set_3	Rate Offset	86.04%	88.14%	62.02%	75.20%	79.97%	43.31%	86.08%	78.66%	64.87%	67.75%	73.18%
full_test_set_3	Poisson	83.62%	86.04%	59.24%	71.35%	78.74%	40.01%	83.91%	77.00%	61.69%	64.89%	70.63%
full_test_set_6	Rate	92.92%	94.54%	64.99%	81.70%	84.01%	47.29%	90.35%	82.74%	64.80%	69.89%	77.34%
full_test_set_6	Rate Offset	91.73%	93.59%	65.39%	80.55%	83.61%	45.74%	89.85%	83.33%	68.39%	72.83%	77.53%
full_test_set_6	Poisson	87.74%	91.06%	62.22%	74.88%	79.96%	43.69%	87.46%	81.45%	64.26%	69.24%	74.23%

Table D.3: Table showing the classification accuracy for the full system. The accuracy shown here is using the Center Of Mass based regions.