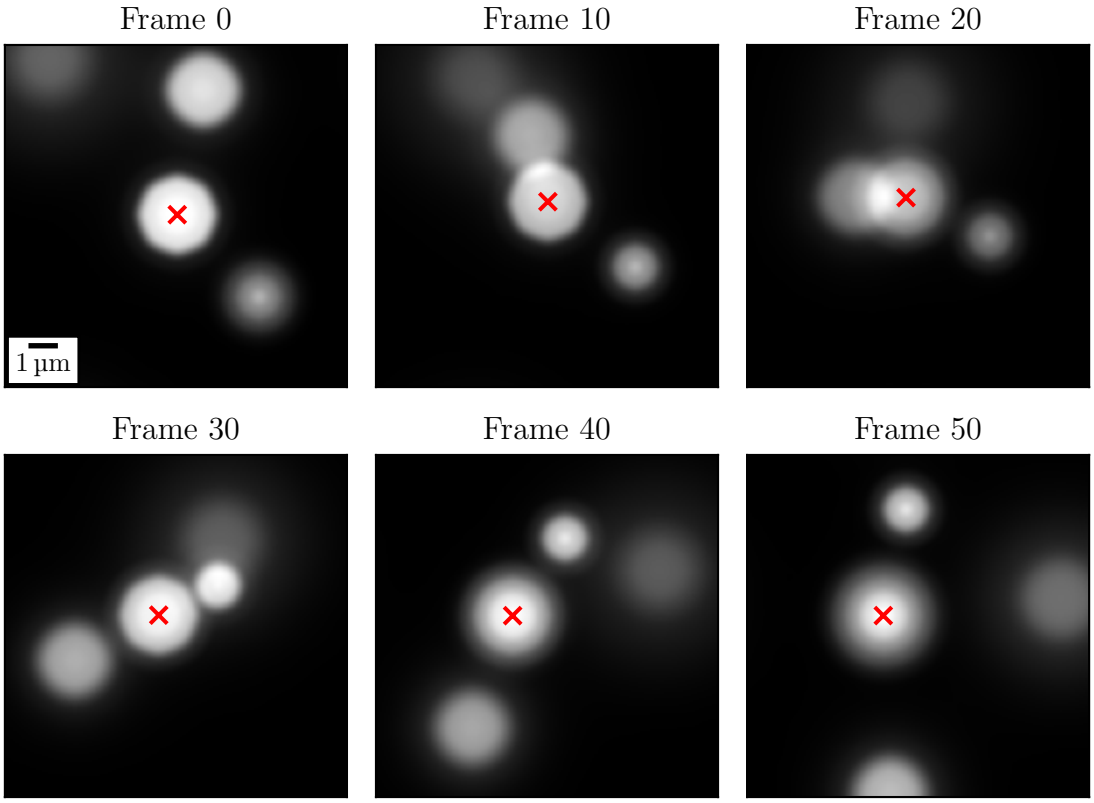

Recurrent Neural Networks for Lagrangian Tracking of Bacteria

A Deep Learning Approach for Tracking of Microorganisms over Long Periods of Time

Master's Thesis in Complex Adaptive Systems

MATHILDA GUSTAFSSON

DEPARTMENT OF PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Recurrent Neural Networks for Lagrangian Tracking of Bacteria

A Deep Learning Approach for Tracking of Microorganisms over Long Periods of Time

MATHILDA GUSTAFSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2025

Recurrent Neural Networks for Lagrangian Tracking of Bacteria
A Deep Learning Approach for Tracking of Microorganisms over Long Periods of Time
MATHILDA GUSTAFSSON

Supervisor: Daniel Midtvedt, Department of Physics, University of Gothenburg
Examiner: Giovanni Volpe, Department of Physics, University of Gothenburg

Master's Thesis 2025
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg

Cover: Example of frames in a sequence of simulated images from a fluorescence microscopy using DeepTrack2, with a centre bacterium (marked with a red cross) moving around in the centre of the frame and additional bacteria crossing the frame in different directions.

Typeset in LaTeX
Printed by Chalmers digitaltryck
Gothenburg, Sweden 2025

Recurrent Neural Networks for Lagrangian Tracking of Bacteria
A Deep Learning Approach for Tracking of Microorganisms over Long Periods of Time
MATHILDA GUSTAFSSON
Department of Physics
Chalmers University of Technology

Abstract

Research in microbiology is crucial for development of antibiotics, vaccines and other medicines that cure diseases and prevent spread of viral infections. One method of studying microorganisms is Lagrangian tracking, where the movement of single microorganisms, such as bacteria, is tracked over long periods of time, which is important when studying for example chemotaxis. Lagrangian tracking has previously been implemented using deep learning, showing promising results. However, the model, a convolutional neural network (CNN), struggled to handle overlapping bacteria, which resulted in failure of entire experiments when the model switched which bacterium was currently being tracked. This thesis aimed to create a model for Lagrangian tracking that could accurately track over long periods of time as well as handle overlapping bacteria. The method included simulation of fluorescence microscopic data as well as design, training and evaluation of recurrent neural networks (RNNs) using the simulated data. The results showed that the RNNs gave lower error distributions and were able to handle overlapping bacteria better compared to the CNNs implemented for benchmarking. An analysis of the importance of features of the bacteria for tracking indicated that the tracking was harder when surrounding bacteria were close to the focal plane or had higher intensity compared to the bacterium that was currently tracked. Although testing the RNNs in an experimental setup remains, the results suggest that replacing a CNN with an RNN can improve the accuracy of the Lagrangian tracking and to greater extent avoid losing the bacterium during an overlap. In turn, improving the accuracy of Lagrangian tracking contributes to the possibility of tracking single microorganisms over long periods of time and gain more knowledge about for example chemotaxis.

Keywords: recurrent neural networks, long-short-term-memory, Lagrangian tracking, microorganisms, bacteria.

Acknowledgements

First of all, I would like to express my deepest appreciation to my supervisor, Daniel Midtvedt. His valuable insights and our discussions were key for the success of this project. Secondly, I would like to thank my examiner, Giovanni Volpe, for all the ideas that helped me further improve this project. Thank you to all the members of the Soft Matter Lab for welcoming me into your group with such open arms. Being part of this group truly brightened this spring and gave me new friends. I am also grateful for the time spent with Linnea and Nienke — you really made the time spent at our little office meaningful. Thank you also to Gottfrid, who cheered me on and helped me improve the figures in this thesis. Last but not least, I would like to thank all my family and friends who supported me throughout the journey of writing this thesis. I am deeply grateful to you all.

Mathilda Gustafsson, Gothenburg, June 2025

Contents

1 Introduction **1**
1.1 Aim, Research Questions and Limitations 3

2 Artificial Neural Networks **5**
2.1 Recurrent Neural Networks 7
2.2 Convolutional Neural Networks 10

3 Methods **13**
3.1 Simulation of Data 13
3.2 Model Architecture 17
3.3 Training Procedure 20
3.4 Evaluation Procedure 21

4 Results **23**
4.1 Error Analysis 23
4.2 Analysis of Overlapping Bacteria 28
4.3 Important Features for Prediction 32

5 Discussion **35**

6 Conclusions and Outlook **39**

Bibliography **41**

A Usage of Artificial Intelligence **I**

B Training Processes **III**

C Additional Results **V**

1 Introduction

There is no doubt that research in microbiology and the technologies developed to study the subject have revolutionized our society over the past centuries. Inventing of the first microscopes in the 16th century [1], the first observations of bacteria in the 17th century [2] and the discovery of antibiotics in the early 20th century [3] are only a few examples that led to our current understanding of cells, diseases and medicines. Today we use the knowledge of microorganisms and the developed microscopy techniques to our benefit in a variety of areas such as medicine, agriculture and bioenergy [4]. One notable example in the area of medicine is the development of the vaccines against SARS-CoV-2 during the COVID-19 pandemic, a development which saved millions of lives [5]. Despite the progress within microbiology over the past centuries there are still gaps of knowledge to fill and challenges to address, with the antimicrobial resistance crisis to mention one [3]. Hence the study of microorganisms, their behaviour and characteristics are vital to continue the development of medicines and to tackle problems related to for example antibiotics.

One method of studying microorganisms is to track their movement. This is important when studying for example chemotaxis [6], which is how cells, such as bacteria, move in response to a chemical substance [7]. The history of tracking microorganisms started with manual tracking in the early 1930's [8]. Since then the field has developed into using digital microscopes and advanced computer algorithms and during the last decade deep learning has entered the scene of microscopic analysis and tracking [8].

Note that in several tracking methods both the objective of the microscope and the container with the microorganisms are kept fixed [9]. This means that microorganisms can swim in and out of the microscopic frame. A limitation arising from this is the impossibility to track specific microorganisms over long periods of time, which is important when studying for example chemotaxis [6]. Moreover, methods that keep the objective and the container with the microorganisms fixed might introduce bias in the data collected [9]. The bias towards microorganisms with lower velocities arises due to the fact that these microorganisms tend to stay in the frame for longer periods of time.

An approach for tracking that avoids these limitations is Lagrangian tracking, first described by Berg in 1971 [10]. The method is specialized in tracking single microorganisms over long periods of time. This is achieved by moving the container of the microorganisms in order to recentre (along the x- and y-axes) and refocus (along the z-axis) the microorganism currently being tracked. The shifts of the container are computed in real-time based on the current frame imaged by the microscope [10]. Using modern microscopic tools and data processing, Darnige et al. [9] created a Lagrangian tracking method based on the result by Berg using a standard image processing method. The Lagrangian tracking, here using fluorescence microscopy, captured the trajectory as well as the actual image sequence of a microorganism, allowing for long term analysis of its behaviour.

A project combining Lagrangian tracking and deep learning is currently carried out at the Soft Matter Lab at University of Gothenburg in collaboration with the group of Prof. E. Clements at Sorbonne University (the group creating the Lagrangian tracking method in [9]). The goals of the project are to improve the accuracy of the method, be able to track in complex environments and allow for tracking with other types of microscopes but fluorescence microscopes. The reason for the latter goal is that the emitted light in fluorescence microscopy can damage the microorganisms [11], which can disturb long term observations. A deep learning model can be trained with data from bright-field microscopy, where problems with damaged microorganisms affecting long term observations can be avoided. Note that the task of the deep learning model is to determine the shifts of the container. These shifts correspond to the position of the microorganism relative to the centre of the current frame and the focal plane. The model should therefore take the current frame as input and output the position of the microorganism currently tracked.

To this end, a convolutional neural network (CNN) has been trained and shows promising results, although there are situations the model cannot handle. The model struggles with overlapping microorganisms, that is, when other microorganisms come close to the microorganism currently being tracked. The model can then switch which microorganism is being tracked, which means that the entire experiment fails and that tracking has to restart. Therefore, there is an evident need for a model that can accurately track over long periods of time as well as handle overlapping microorganisms.

1.1 Aim, Research Questions and Limitations

This thesis aims to develop a deep learning model for Lagrangian tracking that can accurately track bacteria over long periods of time, as well as be able to handle situations with overlapping bacteria. Specifically, the project will aim to answer the following questions:

- How accurately can a new deep learning model for Lagrangian tracking determine the positions of a bacterium in sequences of microscopic images?
- How well can a new deep learning model for Lagrangian tracking handle overlapping bacteria?
- How do features of the bacteria such as size, position relative to the focal plane and the intensity (when using fluorescence microscopy) affect the tracking?

The thesis has two major limitations. First, this thesis focuses only on tracking in x- and y-directions and leaves the tracking along the z-axis unexplored. Secondly, the new model is not tested in an experimental setup during the time of this project. If the model performs well on simulated data it could be implemented in the experimental setup at a later time.

Note that throughout the report, the term *bacteria* is used interchangeably with *microorganism*. Thus, the methods presented can generally be applied to other microorganisms as well. Note also that all usage of AI in this thesis is stated in Appendix A.

2 Artificial Neural Networks

The study of artificial neural networks started as an attempt to model how information flows in the human brain [12]. The human brain consists of about 10^{10} neurons (more than the total population of humans on Earth), which both receive and send information to other neurons. The first mathematical model of a neuron was created in 1943 by McCulloch and Pitts [12]. The so-called McCulloch-Pitts neuron takes several binary values as input, computes a weighted sum of these and outputs a binary value based on whether the weighted sum exceeds some threshold or not. The neurons used in artificial neural networks today are reminiscent of the McCulloch-Pitts neurons, except the outputs are not necessarily binary.

The computation performed in a neuron is illustrated in Figure 2.1 (a). Here the neuron takes as input an array of continuous values, $\boldsymbol{x}$, of length N and computes an output value V according to

$$V = g\left(\sum_{j=1}^{N} w_j\, x_j - \theta\right), \tag{2.1}$$

where w_j, $j = 1, ..., N$, are the weights connected to each input value x_j and θ is the threshold. The expression in the parentheses is called the local field. As seen by the formula, the local field is a linear function. In order for a neuron to map inputs to an output in a non-linear fashion, we need to map the local field with a non-linear function. This function g is called the activation function. Some common activation functions are the hyperbolic tangent, the sigmoid, the ReLU [12], and the Leaky ReLU activation function [13], all shown in Figure 2.1 (b). The choice of activation function is based on the desired range of the output value V.

So much for a single neuron. The power of artificial neural networks arises when several neurons are coupled together. In 1958 Rosenblatt suggested to organize McCulloch-Pitts neurons in layers [12], as shown in Figure 2.2. Rosenblatt called these networks perceptrons, also known as feed forward networks or linear layers. Here each neuron takes inputs from the left and passes an output to the right. Passing an input $\boldsymbol{x}$ to the network, the information will therefore flow through the network from left to right and finally create an output. Depending on the task at hand the output can be for example a number for a regression problem or a vector representing probabilities of different classes for a classification problem.

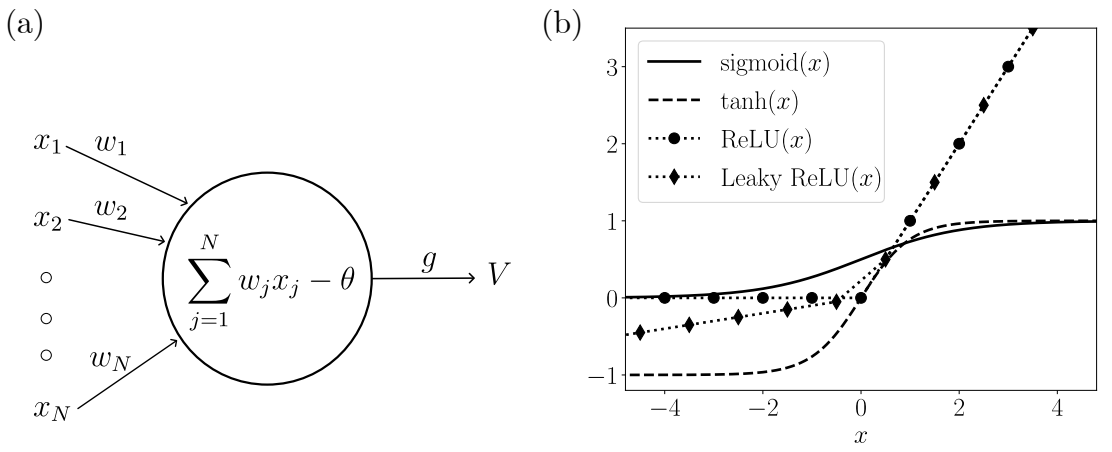

Figure 2.1: (a) The computation carried out in a single artificial neuron. The neuron takes as input an array $\boldsymbol{x}$ and computes a weighted sum of the elements in the input array. A bias θ is subtracted from the sum. Finally an activation function g maps the weighted sum to an output value V. (b) Examples of activation functions. The sigmoid function spans $(0, 1)$, the hyperbolic tanget spans $(-1, 1)$, while the ReLU and Leaky ReLU are unbounded and span $[0, \infty)$ and $(-\infty, \infty)$ respectively.

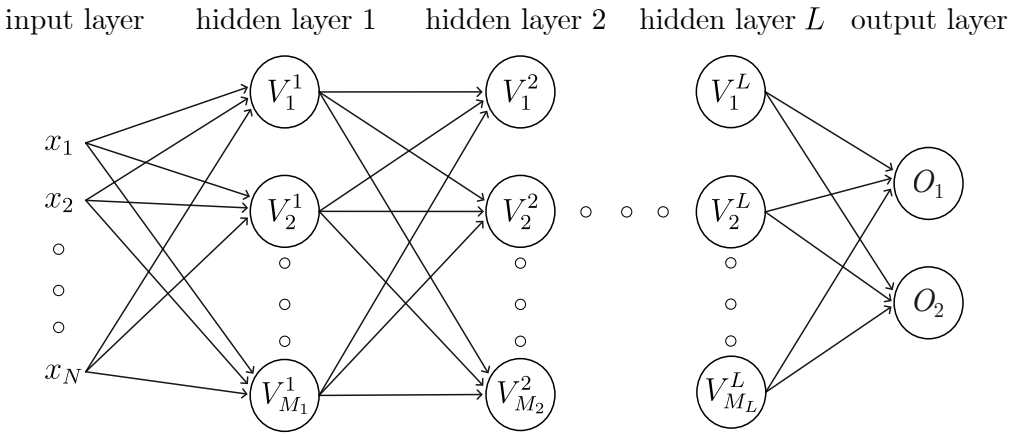

Figure 2.2: The architecture of a feed forward network. The input layer consisting of N input values is followed by L hidden layers with M_1, M_2,.., M_L neurons in each layer. The output layer consists in this example of two neurons. Note that the information always flows from left to right. Moreover, note that the weights, thresholds and activation functions are not shown.

In order to train a neural network to learn a specific task, data is required. For a feed forward network the data consists of pairs of input and desired output values, called targets [12]. During training, inputs are passed through the network and outputs are computed. The parameters of the network (the weights and the thresholds for each neuron) are then updated such that the outputs get closer to the targets. One common method to update the parameters in a feed forward network is error back-propagation [12]. In this algorithm, the weights and thresholds are updated using gradient descent on an energy function H. The energy function can for example be expressed as

$$H = \frac{1}{2} \sum_{\mu} \sum_{i} \left(t_i^{\mu} - O_i^{\mu} \right)^2 , \tag{2.2}$$

where the sum over μ is the sum over all pairs of outputs and targets, and the sum over i is the sum over the output nodes of the network. The energy function is a non-negative function that has its minimum when the outputs O_i^{μ} equal the targets t_i^{μ}. Performing gradient descent on H with respect to the network parameters can therefore make the network produce outputs similar to the targets. Update rules for weights w_{ji} connecting neurons i, $i = 1, \ldots, N$ in layer $l-1$ to neuron j in layer l and for the threshold of neuron j in layer l are given by

$$w_{ji}^{new} = w_{ji}^{old} - \eta \frac{\partial H}{\partial w_{ji}} \qquad \text{and} \qquad \theta_j^{new} = \theta_j^{old} - \eta \frac{\partial H}{\partial \theta_j} \tag{2.3}$$

where η is the learning rate adjusting the size of the updates. Derivation of the gradients of H together with other examples of energy functions are found in [12].

2.1 Recurrent Neural Networks

The connections between neurons in a neural network do not necessarily need to go from left to right. Consider a feed forward network with additional connections from right to left or between neurons in the same layer. A recurrent neural network (RNN) is then obtained [12]. With backward connections the aspect of time becomes relevant. The outputs of the network now depends not only on the current input but also on previous inputs, because of the memory of previous inputs stored with the backward connections. The network can therefore be trained not only on input and output pairs, but on sequences of input and output pairs. This property makes RNNs suitable for tasks such as text translation and speech recognition [12].

Early examples of RNNs include Jordan [14] and Elman [15] networks. In 1986 Jordan suggested to use connections from the output neurons to so-called state units that are connected back to the hidden neurons, see Figure 2.3 (a). Another approach by Elman in 1990 was to add connections from the hidden units to state units and then back to the hidden units, see Figure 2.3 (b). In both cases the state units serve as a memory of previous input values.

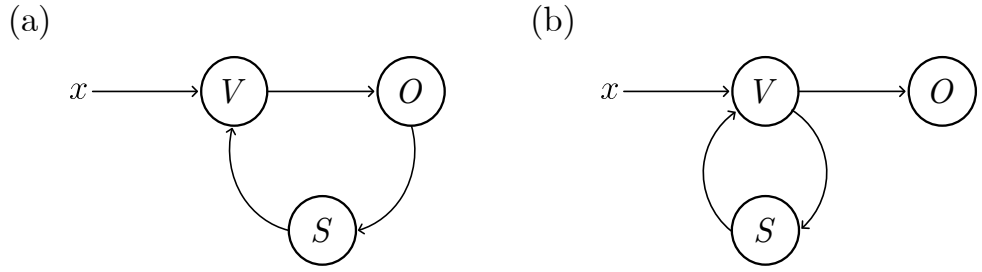

Figure 2.3: Examples of RNN architectures, here with only one input value x, one hidden neuron V, and one output neuron O. (a) shows a Jordan network where the recurrent connection goes from the output neuron O to the state neuron S and back to the hidden neuron V. (b) shows an Elman network where the recurrent connection goes instead from the hidden neuron to the state neuron and back to the hidden neuron.

RNNs such as Jordan and Elman networks can be trained using recurrent back-propagation through time [12]. In this method, the network is unfolded in time, meaning that backward connections are removed and that the original network is repeated representing different discrete time steps, see [12]. The unfolded network can be trained in a similar fashion as a feed forward network. However, one problem arising from backpropagation through time is the vanishing gradient problem. This problem occurs during training when gradients used in the backpropagation algorithm become so small that inputs at early time steps do not contribute to updates of the parameters, affecting the ability of the network negatively [12].

One attempt to solve the vanishing gradient problem is by using Long Short-Term Memory (LSTM) computational units, first invented by Hochreiter and Schmidhuber in 1997 [16]. The core principle of LSTMs is to use computational units with gates, controlling the flow of information through the network. The gates control what information to store in the unit's internal state, which serves as a memory for the unit. This design allows for memory of early inputs to be stored in the unit and contribute to the output over thousands of time steps forward [16]. Therefore, this design helps to have a constant error flow, that is neither vanishing nor exploding gradient problems [16].

The different gates in an LSTM unit have different purposes. The initial version of an LSTM had two types of gates: an input gate to protect memory from irrelevant input, and an output gate to protect other units from irrelevant memory stored [16]. In later versions of LSTMs, a forget gate was introduced in order to remove unnecessary information from the internal state [17]. The computational steps in an LSTM is shown in Figure 2.4. Note that in the figure the input gate is separated into two gates: the input gate and the cell gate. However this is only a matter of terminology [18].

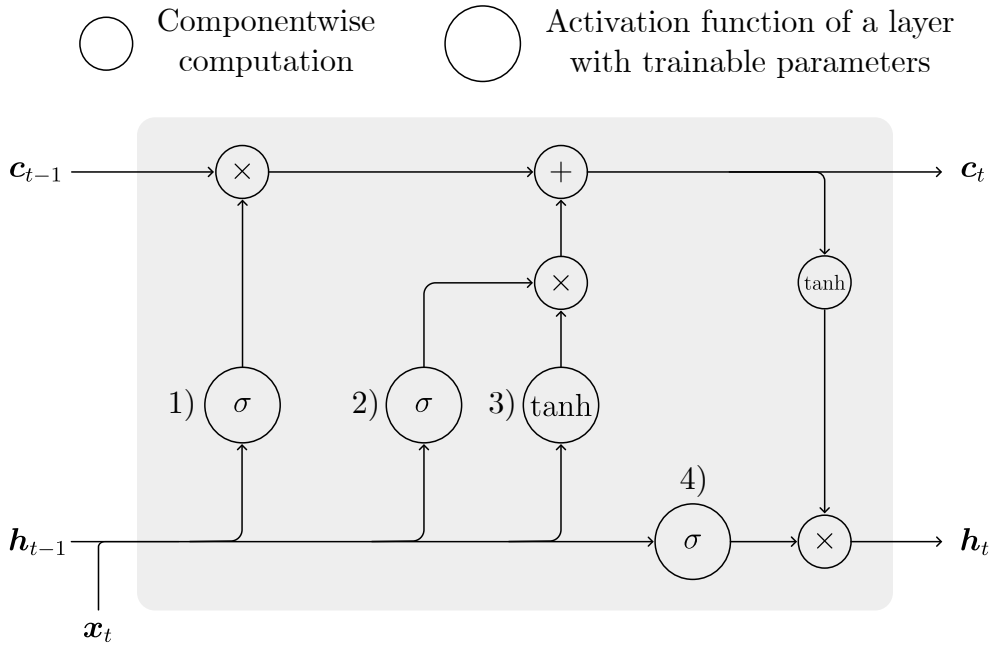

Figure 2.4: The computations carried out in an LSTM unit (the grey area) with the different gates numbered from 1) to 4). In the forget gate at 1), a linear neural network layer with sigmoid activation function σ takes the input vector $\boldsymbol{x}_t$ and the hidden state $\boldsymbol{h}_{t-1}$ from the previous time step as input. The output of the forget gate is multiplied componentwise with the cell state $\boldsymbol{c}_{t-1}$, which decides what part of the cell state to keep. At the input and cell gates at 2) and 3) a new contribution to the cell state is created by generating a suggestion in 3) and using 2) to decide what parts of the suggestion to add to the cell state. Finally at the output gate at 4) a linear layer with sigmoid activation decides what information from $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$ should contribute to the output of the cell. The output of the layer is multiplied with $\tanh(\boldsymbol{c}_t)$ which create the output of the LSTM unit: $\boldsymbol{h}_t$. Note that the output of the LSTM at the current time step will serve as the hidden state values for the unit at the next time step.

2.2 Convolutional Neural Networks

Convolutional neural networks were invented in the 1980s and are designed for image analysis tasks such as object recognition and image classification [12]. CNNs are suitable for these tasks due to their capability of extracting local features in an image, such as edges and corners. An additional advantage of CNNs is that they have less trainable parameters compared to a feed forward network with the same number of neurons, which makes CNNs faster to train [12].

The key element in CNNs is the convolutional layers, which make use of kernels [12]. Think of a kernel as a filter that takes a few pixels of an image as input and outputs a value representing a filtered version of these pixels. In a convolutional layer, a kernel is moved across an image with a certain stride $\boldsymbol{s} = [s_1, s_2]$, determining by how many pixels the kernel moves in x- and y-directions in each step. In each step the kernel reads an area of a fixed size, say $P \times Q$ pixels. The output of the reading, V_{ij}, is stored in a so-called feature map at position (i, j), see Figure 2.5. Note that the input image can consist of multiple channels R, for example representing the RGB colours. In this case the kernel reads all channels at each step. The computation of V_{ij} is given by

$$V_{ij} = g\left(\sum_{p=1}^{P}\sum_{q=1}^{Q}\sum_{r=1}^{R} w_{pq}\, x_{p+s_1(i-1),q+s_2(j-1),r} - \theta\right), \tag{2.4}$$

where g is the activation function, p and q are the indices of the kernel, r is the index of the channel, $\boldsymbol{x}$ is the input image, and w_{pq} and θ are the weights and the threshold of the kernel [12]. Equation 2.4 shows that the same weights and threshold are used for computation of all values V_{ij} in the feature map corresponding to the kernel, since w_{pq} and θ are independent of i and j. This architecture explains the statement about less trainable parameters in a CNN compared to a feed forward network [12]. Note that a feature map is simply a two dimensional array of neurons that all share the same weights and threshold. These feature maps are often referred to as the convolutional layers.

In addition to the convolutional layers, CNNs can contain pooling layers. The pooling layers also make use of kernels, but in contrast to convolutional layers, max-pooling layers have no trainable parameters. Instead they perform some predetermined compression of the input image or feature map [12]. A common pooling layer is the max pooling layer, where the feature map is compressed by taking the maximum value of the values read by the kernel in each step [12]. Another type of pooling is L_2 pooling, where the root mean square of the values is computed.

A CNN is created by combining convolutional layers with for example pooling layers and linear layers. An example of such a network is shown in Figure 2.6. Normally a CNN starts with convolutional and pooling layers in some order. Note that several kernels can read the same input image or feature map, resulting in one feature map corresponding to each kernel. The feature maps resulting from the last convolutional or pooling layer are then flattened into a one dimensional array, which can

be considered a latent space representation of the input image. The latent space representation is finally passed through linear layers. The format of the output will depend on the task of the network, for example regression or classification [12].

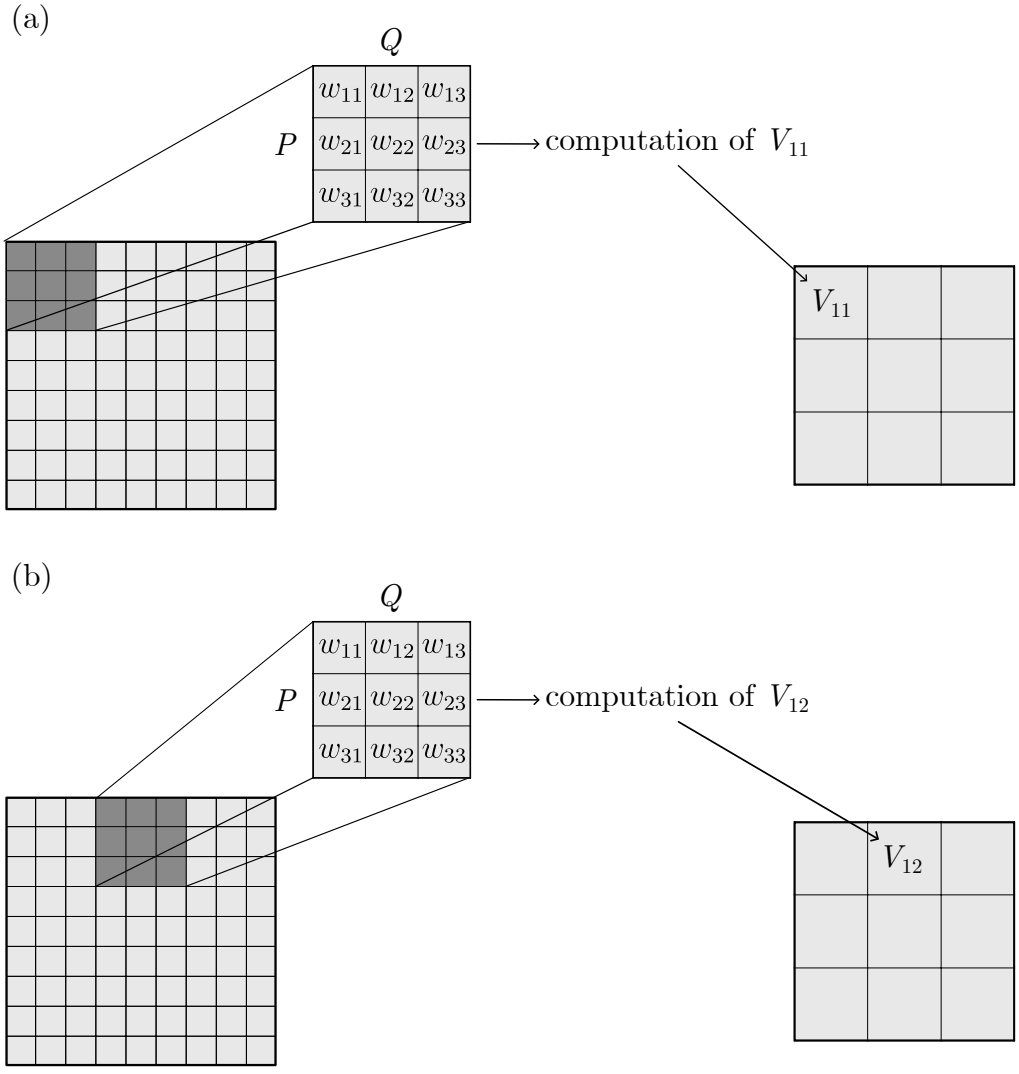

Figure 2.5: Examples of computations in a convolutional layer. (a) The upper left nine pixels are read by a kernel of size $P \times Q$. Connected to the kernel are PQ number of weights, which are used in the computation of V_{11} according to Equation 2.4. The output V_{11} is stored in the feature map at position $(1, 1)$. (b) Assuming stride $\boldsymbol{s} = [3, 3]$ the kernel next reads the upper middle nine pixels. Note that the same kernel with the same parameters is used. The output V_{12} is computed and stored in the feature map at position $(1, 2)$. The next reading would be the upper right pixels, followed by the left middle pixels, and so forth.

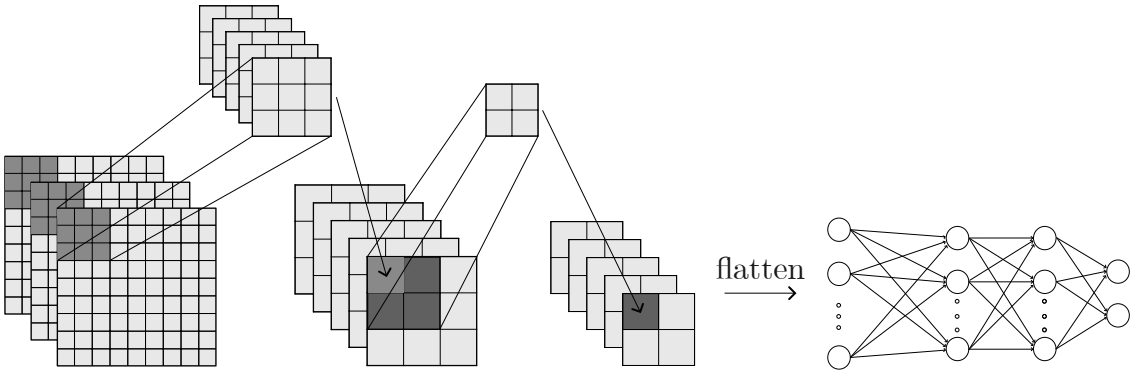

Figure 2.6: An example of a CNN. First, the input image consisting of three channels is read by five different kernels of size 3×3. Indicated in the figure is the fifth kernel reading the upper left corner of all channels of the input image. Assuming these are convolutional layers, the output is computed using Equation 2.4 and stored in the fifth feature map at position $(1, 1)$. Moving all kernels across the image will fill all positions in all feature maps. Assuming the next layers are max pooling layers, a kernel of size 2×2 reads the feature maps and compresses them into smaller sizes. Since the pooling layers do not contain any trainable parameters, only one kernel is displayed, resembling that the same operation is carried out on all feature maps. Finally, the feature maps are flattened into a one dimensional array that are fed into a feed forward network. The format of the output of the feed forward network depends of the type of task.

3 Methods

The goal of this thesis was to develop a model for Lagrangian tracking that could accurately track bacteria and handle overlapping bacteria. Specifically, we wished to keep the bacterium we wanted to track in the centre of the frame. The task of the model was therefore to predict the position of the bacterium relative to the centre of the frame, since this corresponded to the shifts of the container required to keep the bacterium in the centre. The methods required to create a model that fulfilled these criteria included 1) simulation of data for training and evaluation, 2) construction of a model and 3) training and evaluation of the model. Note that all code for the project is found at the GitHub page for the project [19].

3.1 Simulation of Data

The data used for training and evaluation was simulated using DeepTrack2, a framework for simulation of microscopic images [20]. Another option would have been to use an experimental setup to obtain the microscopic images, although this would have been incredibly time inefficient. Simulation of data allowed for fast generation of images and an extractions of positions of the bacteria in the images. Moreover, it was possible to simulate sequence of images where the trajectories of the bacteria were the same, but where for example size and intensity of the bacteria varied. This made it possible to compare a model's performance on different types of data.

The data was simulated as sequences of microscopic images of size 128×128 pixels, representing videos of recorded bacteria moving around in a container. The optical device in DeepTrack2 used in the simulations was a fluorescence microscope with numerical aperture 0.8, magnification 10, resolution 1 µm and wavelength 680 nm. While data from other types of microscopes that do not affect bacteria's behaviour over long measurements, would have been more suitable in practice, fluorescence microscope was used because it was considered to generate relatively simple images. In order to detect a difference between z-values above and below the focal plane (where the microscope has its focus), a spherical aberration with coefficient 0.5 was added to the optical device in DeepTrack2. Figure 3.1 shows different images of a bacterium located below, at and above the focal plane.

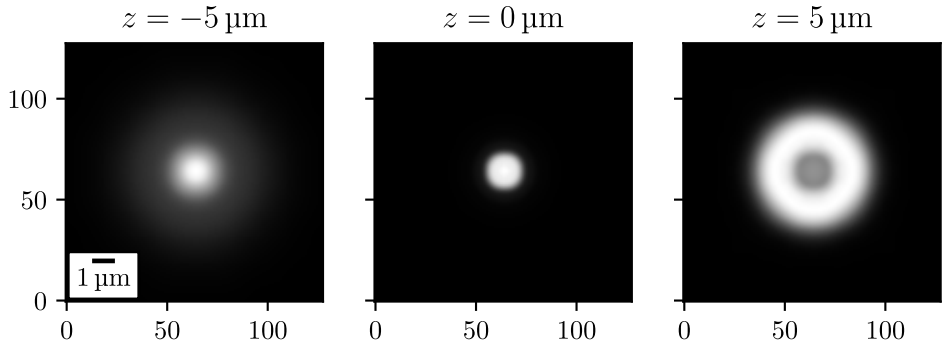

Figure 3.1: Simulated images of a bacterium located at different distances from the focal plane, namely at $-5\,\mu m$, $0\,\mu m$, and $5\,\mu m$.

One aspect to consider when simulating the data was that in an experimental setup the observed images are dependent on the output of the model. In other words, the shifts of the container, determining what the next image looks like, is based on the output of the model. The approach in this thesis was to train a model in a supervised manner, which required data disconnected from the model. Hence, the data needed to be simulated without the use of a model, but in such a way that the data mimicked the sequences observed in an experimental setup.

Sequences were simulated with the bacterium we wanted to track, from now on called the centre bacterium, kept near the centre of the image. In the simulation, this was achieved by moving the centre bacterium randomly around the centre of the image (x- and y-directions) and around the focal plane of the microscope (z-direction). If the bacterium moved to a position located more than 25 % of the image width away from the centre in x- or y-direction, the bacterium was pushed back towards the centre. Likewise, if the bacterium was located more than $5\,\mu m$ away from the focal plane, the bacterium was pushed back towards it. The value $5\,\mu m$ was chosen since it corresponds to the maximum size that a bacterium is assumed to move between two frames.

The simulated sequences had additional bacteria moving around in or passing the image. These additional bacteria had different sizes and intensities, and were located at different distances from the focal plane. In contrast to the centre bacterium, the additional bacteria did not move around randomly. Instead they were given velocities in the x-, y- and z-directions. These velocities could change in each step by adding small random values. The changing velocities helped mimic the experimental setup, because it gave the illusion that both the centre and additional bacteria were moving, despite the centre bacterium being kept in near centre of the frame. An example of a simulated sequence with both the centre and additional bacteria is shown in Figure 3.2.

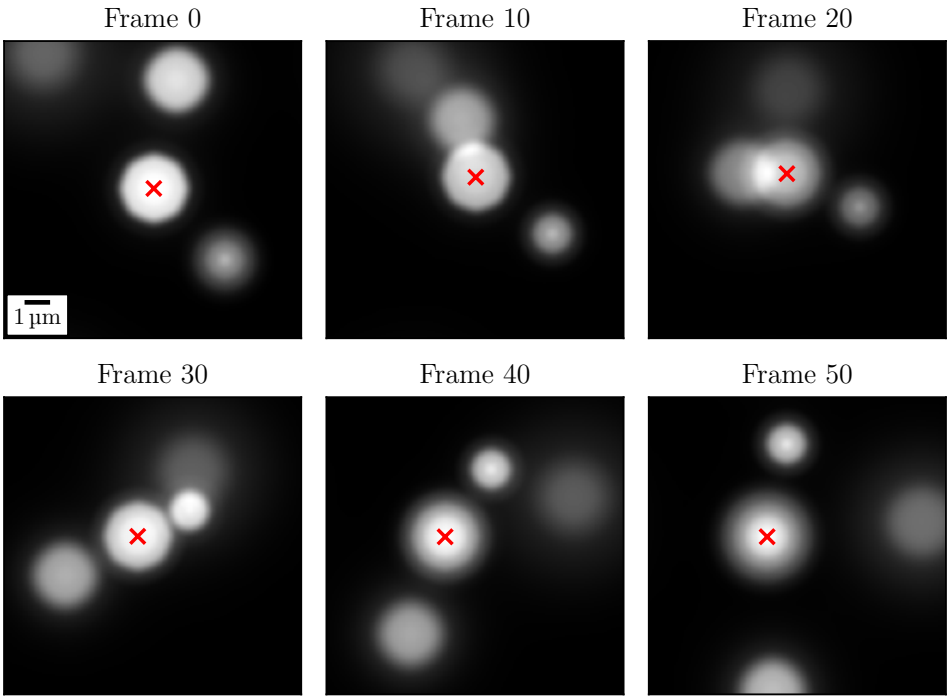

Figure 3.2: Example of frames from a simulated sequence with the centre bacterium (marked with a red cross) moving around in the centre of the frame and additional bacteria crossing the frame in different directions.

Datasets were simulated for both training, validation and testing. All datasets consisted of sequences of images with the centre bacterium and a number of additional bacteria. For training, validation and testing, two different datasets were created: one where all bacteria were identical (same radius, same intensity in the fluorescence microscopy, all located at $z = 0$) and one where the features of the bacteria varied (different radii, different intensities and varying z-values). The first dataset was mainly used to compare performances of the models in a setting as simple as possible, while the latter dataset was more realistic to the experimental setup. See Table 3.1 for the specifications of the datasets.

In order to specifically test how the models handle overlapping bacteria, datasets were created for this purpose. Here the centre bacterium was kept around the centre while an additional bacterium moved across the frame from one edge to the oppposite one. This guaranteed that the centre and additional bacteria came relatively close to each other, allowing for testing of the models' abilities to handle overlapping bacteria. The specifications of these datasets are shown in Table 3.1.

Furthermore, a number of datasets were created in order to test what features of the bacteria affected the models' performances. For this task one parameter was varied at a time, creating datasets corresponding to different values of the varying parameter. The varied parameters were the positions along the z-axis, the radii and the intensities of the bacteria. Table 3.2 specifies the values used for the varying parameters. The datasets were simulated with one centre bacterium and one additional bacterium crossing the frame, as in the datasets created to analyse overlaps.

Note that the movement of all bacteria were the same in all datasets, so the feature investigated was the only varying parameter. The idea was that evaluating a model trained on data with all the parameters varying would give a hint on what features made prediction of the position harder and which did not.

Table 3.1: The datasets simulated for training, validation and testing both errors and how models handle overlaps. The first column i identifies each dataset. The variables in the table are named such that S_{data} is the dataset size, S_{seq} is the length of each sequence, n_{add} is the number of additional bacteria, z_{c} and z_{add} are the z-values for the centre and the additional bacteria, r_{c} and r_{add} are the radii of the centre and additional bacteria, and I_{c} and I_{add} are the intensities for the centre and additional bacteria respectively. Note that for the z-values, the intervals specify the allowed values during a sequence, while for the radii and the intensity the intervals specify what radii and intensity the bacteria could be initialized with in the beginning of each sequence. Note also that only the relative intensities between the bacteria were relevant, not the values themselves.

i	Used for	S_{data}	S_{seq}	n_{add}	z_{c}, z_{add} [µm]	r_{c}, r_{add} [µm]	I_{c}	I_{add}
1	Train	300	300	$[1, 3]$	0	1	1	1
2	Validation	100	300	$[1, 3]$	0	1	1	1
3	Test errors	100	300	$[1, 2]$	0	1	1	1
4	Test overlaps	500	65	1	0	1	1	1
5	Train	500	300	$[1, 2]$	$[-5, 5]$	$[0.5, 1.5]$	1	$[0.2, 1.8]$
6	Validation	100	300	$[1, 2]$	$[-5, 5]$	$[0.5, 1.5]$	1	$[0.2, 1.8]$
7	Test errors	100	300	$[1, 2]$	$[-5, 5]$	$[0.5, 1.5]$	1	$[0.2, 1.8]$
8	Test overlaps	500	65	1	$[-5, 5]$	$[0.5, 1.5]$	1	$[0.2, 1.8]$

Table 3.2: The datasets simulated for analysing important features of the bacteria for tracking. Unless stated as the varied feature or specified in the last column, the values were set to $z_{\text{c}} = z_{\text{add}} = 0\,\text{µm}$ for the positions in the z-direction, $r_{\text{c}} = r_{\text{add}} = 1\,\text{µm}$ for the radii of the centre and additional bacterium, and $I_{\text{add}} = I_{\text{c}} = 1$ for the intensity of the bacteria (only the relative intensities matter). For each value of the varied feature a separate dataset was created.

Feature varied	Values of varied feature	Other specifications
z_{add}	$\{-5, 4, ..., 4, 5\}$	$z_{\text{c}} = -2\,\text{µm}$
z_{add}	$\{-5, 4, ..., 4, 5\}$	$z_{\text{c}} = 0\,\text{µm}$
z_{add}	$\{-5, 4, ..., 4, 5\}$	$z_{\text{c}} = 2\,\text{µm}$
r_{add}	$\{0.5,\ 0.6,\ ...,\ 1.4,\ 1.5\}$	$r_{\text{c}} = 0.6\,\text{µm}$
r_{add}	$\{0.5,\ 0.6,\ ...,\ 1.4,\ 1.5\}$	$r_{\text{c}} = 1\,\text{µm}$
r_{add}	$\{0.5,\ 0.6,\ ...,\ 1.4,\ 1.5\}$	$r_{\text{c}} = 1.4\,\text{µm}$
I_{add}	$\{0.2,\ 0.36,\ ...,\ 1.64,\ 1.8\}$	$I_{\text{c}} = 1$

3.2 Model Architecture

To construct a model, one should first understand why a CNN was not a suitable model for the problem at hand. The reason why the previous CNN struggled with overlapping bacteria was because it only used the current frame to make predictions of the position of the centre bacterium. Therefore, if two bacteria were equally close to the centre of the frame, the CNN did not have a clue which of the bacteria it should track. Thus, simply training a CNN with data that includes overlapping bacteria was not a sufficient solution.

The solution for the problem with overlapping bacteria was to take the history of the movement of the bacteria into account. The hypothesis was that information about previous frames could help the model to keep track of the movement of the centre and surrounding bacteria, hence making the prediction easier even when bacteria overlap. In other words, the idea was that the model should be able to learn how to keep track of the centre bacterium despite surrounding bacteria.

In practice, information about previous frames was used by implementing an RNN. The RNN consisted of a convolutional part, mapping the input image to a latent space, a recurrent part handling the memory of previous frames, and finally linear layers for regression. Figure 3.3 shows the outline for this model.

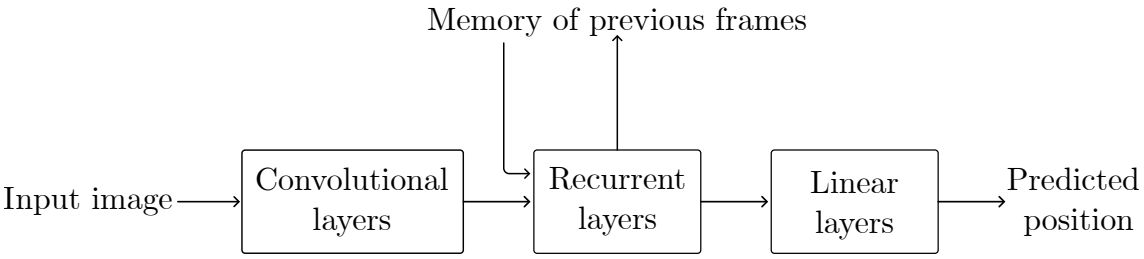

Figure 3.3: Outline of the RNN architecture. The input image is sent to the convolutional layers, the recurrent layers and finally the linear layers (a feed forward network) which output the predicted position. Note that the recurrent layers store and use a memory of the previous frames in the sequence. The boxes represent layers with trainable parameters.

For the recurrent part of the RNN, two potential types of networks were Elman networks and LSTM. The architecture of the Elman networks is more simple, although LSTMs are better at handling vanishing gradient problems and can learn faster [16]. Therefore LSTMs were used for the recurrent part. The number of LSTMs used was determined by training models with different number of LSTMs and comparing the computational time and the performance of the models. This analysis showed that three LSTM units were a suitable choice. With LSTM units as the recurrent part of the RNN the general architecture of the RNN is shown in Figure 3.4.

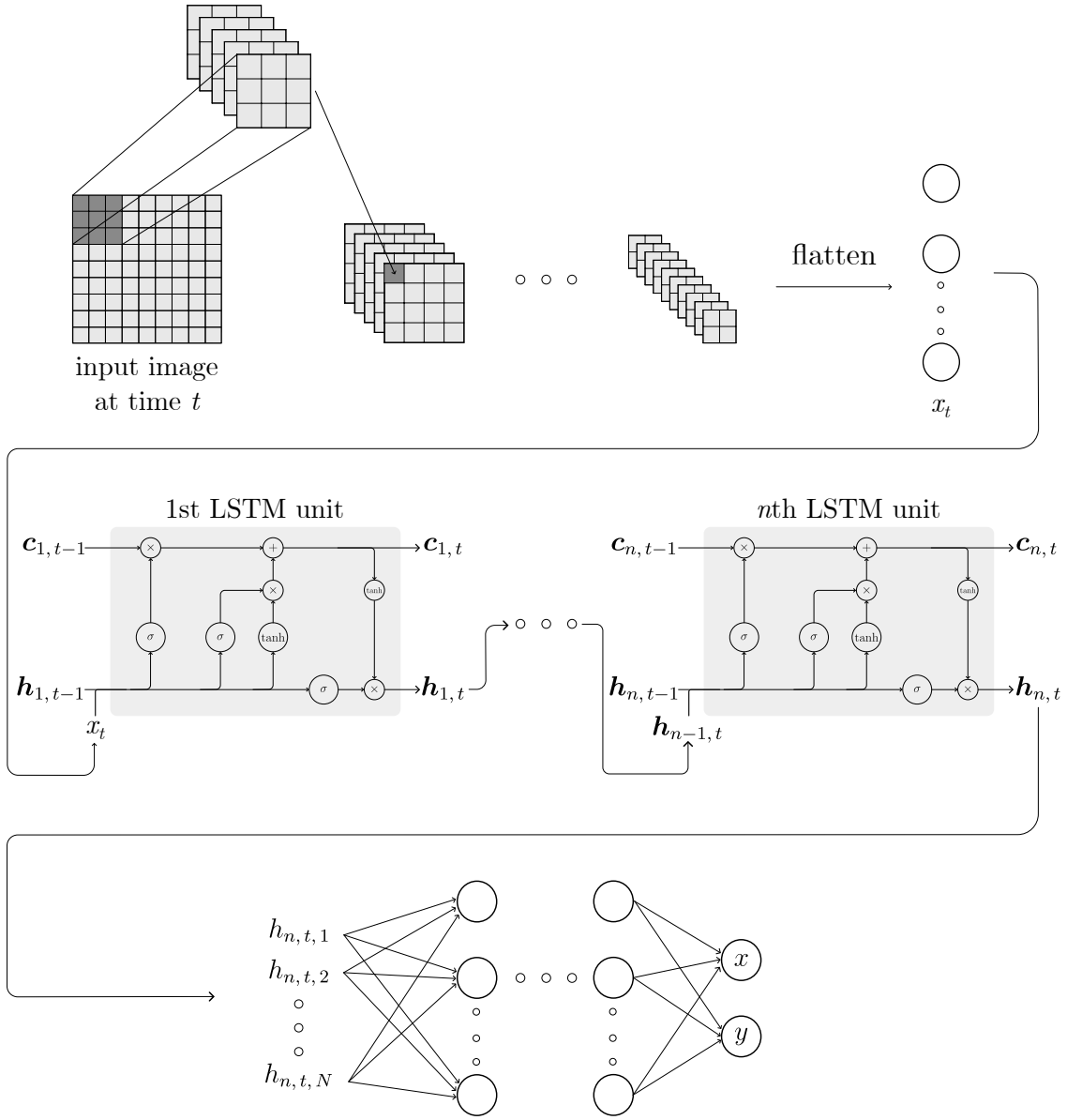

Figure 3.4: Architecture of the RNN, consisting of convolutional and pooling layers (top), recurrent layers (middle) and linear layers (bottom). During a pass, the input image is first processed by convolutional and max pooling layers. Note that the horizontal dots represent layers that are not shown. The output of the last layer is flattened, resulting in a latent space representation as a one dimensional array. The latent space representation $\boldsymbol{x}_t$ is fed to the first LSTM unit. Using memory from previous frames, an output of the first LSTM unit is computed. The output $\boldsymbol{h}_{1,t}$ is fed as input to the second LSTM unit. Note that the horizontal dots represent LSTM units not shown. The output of the second to last LSTM unit is fed to the last, the nth, LSTM unit. Finally, the output $\boldsymbol{h}_{n,t}$ of size N of the last LSTM unit is fed to a feed forward network, which outputs the predicted position in x and y. Note that the size of the input image, the number of feature maps, the size of feature maps and the number of linear layers are necessarily not accurate to that of the real model.

In addition to the RNN, a CNN was created in order to benchmark the RNN against a model without the ability to store memory of previous frames. The CNN had the same architecture as the RNN, except that the recurrent part in Figure 3.3 was replaced with standard linear layers, without any memory of previous frames. Comparing the RNN and the CNN could show the impact of the recurrent part in the RNN compared to the linear part in the CNN. The specific architectures of the RNN and the CNN are shown in Table 3.3.

Table 3.3: Architecture of the RNN (given in table) and the CNN (given in table except that the LSTM layers were replaced with linear layers with output size 32). Note that the format of the output size is (number of channels × width × height) for the convolutional and pooling layers. Note also that during training, batch normalization was used after each convolutional layer, but before the activation function was applied.

Type of layer	Output size	Kernel size	Stride	Padding	Activation
Convolutional	$(16, 129, 129)$	2	1	1	Leaky ReLU
Convolutional	$(16, 128, 128)$	2	1	0	Leaky ReLU
Max pooling	$(16, 64, 64)$	2	2	N/A	N/A
Convolutional	$(32, 65, 65)$	2	1	1	Leaky ReLU
Convolutional	$(32, 64, 64)$	2	1	0	Leaky ReLU
Max pooling	$(32, 32, 32)$	2	2	N/A	N/A
Convolutional	$(64, 33, 33)$	2	1	1	Leaky ReLU
Convolutional	$(64, 32, 32)$	2	1	0	Leaky ReLU
Max pooling	$(64, 16, 16)$	2	2	N/A	N/A
Convolutional	$(128, 17, 17)$	2	1	1	Leaky ReLU
Convolutional	$(128, 16, 16)$	2	1	0	Leaky ReLU
Max pooling	$(128, 5, 5)$	3	3	N/A	N/A
Linear	256	N/A	N/A	N/A	Leaky ReLU
LSTM	32	N/A	N/A	N/A	See Figure 2.4
LSTM	32	N/A	N/A	N/A	See Figure 2.4
LSTM	32	N/A	N/A	N/A	See Figure 2.4
Linear	16	N/A	N/A	N/A	ReLU
Linear	2	N/A	N/A	N/A	N/A

3.3 Training Procedure

The processes of training the RNN and the CNN were similar. Both models were trained on dataset 1 and 5 in Table 3.1. The models were trained between 24 and 30 epochs based on the loss on the validation data (dataset 2 and 6 in Table 3.1) during the training process, see Appendix B. One epoch consisted of iterating through each sequence in the dataset. For each frame in the sequence, a prediction of the position of the centre bacterium was computed using the model. Based on the prediction and the true value of the position a loss was computed. Summing up the losses for all frames in one sequence, the parameters of the model were updated between each sequence. Since the RNN used memory of previous frames to make its prediction, the first ten images in each sequence were passed through the network to store memory in the model and were not used to make predictions. Furthermore, note that the length of the sequences were 300 or 500 in the datasets used for training, which means that the batch size was effectively also 300 or 500. The learning rate was set to 10^{-3} and the optimizer used was the Adam optimizer in PyTorch.

Initially, the loss function used to train the models was the L1 loss, also known as mean absolute error. This loss was based on the distance between the predicted and true position, hence it would force the model to learn to make predictions close to the position of the centre bacterium. Note that during training the loss used was the sum of the L1 loss computed for each frame in the sequence, which effectively was the sum of absolute errors instead of the mean.

Although the L1 loss forced the model to make predictions close to the centre bacterium, the problem the model should solve was tracking despite overlapping bacteria. When an overlap happened, an additional bacterium was close to the centre bacterium. Since the bacteria were close to each other, incorrectly predicting the position as the one of the additional bacterium would not give a large loss using L1 loss. However, starting to track the wrong bacterium would have huge implications in the experimental setup, since the experiment needs to be restarted if the model loses the bacterium that is currently tracked. Hence, the loss function should optimally also reflect how close the prediction was to the centre bacterium compared to the additional ones.

A second loss function, the classification loss, was added. It took into account the distances from the predicted position to all bacteria in the frame, including the centre one. After a prediction and computation of the distances, a softmin function was applied. The softmin function transformed the distances such that the smallest distances got the highest values and so that all elements summed to 1. The softmin values could be viewed as a probability vector, stating which bacterium was more likely to be the centre one based on the prediction. By comparing the softmin value with targets consisting of zeros everywhere except at the index corresponding to the actual centre bacterium, where the value was set to 1, the models could be trained using cross-entropy loss. This was similar to a classification task, hence the name classification loss. The final loss was a weighted sum of the L1 and classification loss, such that they contributed approximately equally in the training process.

The combinations of datasets, types of model and loss functions used for training is shown in Table 3.4. The setup for training allowed for evaluation of the models' performances on the different datasets and analysis of the impact of the classification loss. Additionally the number of epochs used for training for each model is stated in Table 3.4.

Table 3.4: Models trained on different datasets with different loss functions for a specific number of epochs.

Dataset	Model	Loss function	Epochs trained
1 in Table 3.1	CNN	L1	27
1 in Table 3.1	CNN	L1 + classification loss	24
1 in Table 3.1	RNN	L1	30
1 in Table 3.1	RNN	L1 + classification loss	30
5 in Table 3.1	CNN	L1	26
5 in Table 3.1	CNN	L1 + classification loss	30
5 in Table 3.1	RNN	L1	30
5 in Table 3.1	RNN	L1 + classification loss	30

3.4 Evaluation Procedure

The models were evaluated on the test datasets where all bacteria were identical or the test dataset more realistic to the experimental setup, depending on which dataset the model was trained on. The evaluation on data with identical bacteria showed if and how much better the RNN could track a specific bacterium in a setting with as few varying parameters that could affect the results as possible. The evaluation on data where bacteria had varying radii, intensities and were allowed to move in the z-direction should give a result more realistic to the experimental setup.

The models' performances were evaluated by conducting an error analysis, using dataset 3 or 7 in Table 3.1. Errors were computed as the Euclidean distance between the predicted and the true position. The size of the errors as well as where in the frame the errors were largest were analysed. In order to quantify the difference of the means of the error distributions between models, Welch's t-test was used. Welch's t-test is a statistical test used to investigate the difference in means between samples from two normal distributions where the variances are not assumed to be equal [21]. In our case, we had a null hypothesis that the means were equal. We wanted to investigate if this null hypothesis could be rejected given some significance level, which is the probability that a true null hypothesis is rejected [22]. Using the t-statistic from Welch t-test, the p-value was computed. The p-value is the minimum significance value for which the null hypothesis can be rejected [22]. If the p-value was lower than the significance level we used, 0.05, we said that the two distributions had significantly different means. When the data did not seem

normally distributed by visual inspection, for example when errors only taking positive values were inspected, the t-test was made on ranked data instead. Then the means of the ranked values were tested instead, as suggested in [21].

The error metrics told how good the models were at predicting the true position in general, but they did not explicitly tell how well the models handled overlapping bacteria. This was because a prediction closer to an additional bacterium would not result in a substantial increase in the error if the bacteria were close to each other. Moreover, the datasets used were not guaranteeing overlapping bacteria. Therefore, an additional metric that explicitly tested how well the models handled overlapping bacteria was created. Using the datasets 4 and 8 in Table 3.1, a binary classification was carried out for each frame, determining if the prediction was closer to the centre bacterium or the additional one. The fraction of frames where the prediction was closer to the additional bacterium compared to the centre one was then computed. This metric better reflected how well the models could handle overlapping bacteria.

Additionally, a deeper analysis of how the models handle overlapping bacteria was conducted. This was necessary since the binary classification did not indicate whether the predictions were exactly at the centre bacterium or slightly pushed towards the additional one. In this analysis, the softmin value corresponding to the centre bacterium was computed based on the distances from the prediction to all bacteria. The softmin values could be interpreted as the probabilities that the centre bacterium was actually that, the centre bacterium, given the prediction. Note that the fraction of softmin values below 0.5 corresponded to the metric computed in the binary classification above. Again, Welch's t-test was used to compute the significance of the difference between the means of the ranked data of the softmin distributions.

Lastly, an analysis the importance of features of the bacteria for prediction was carried out. One can imagine that tracking is more difficult when the additional bacterium is similar to the centre one and they become more indistinguishable. In order to analyse if this was the case, the RNN trained using both L1 and classification loss was evaluated using datasets in Table 3.2, where the positions along the z-axis, the radii and the intensities of bacteria were varied. The metric used to evaluate the performance was the fraction of frames where the prediction was closer to the additional bacterium compared to the centre one. The importance of the feature could then be analysed by studying the values of this metric for each dataset.

4 Results

The results include evaluation of the RNN's and the CNN's performances. The analyses carried out are an error analysis and an analysis of how the models handle overlapping bacteria. Additionally, the importance of different features of bacteria for tracking is analysed using the RNN trained with L1 and classification loss.

4.1 Error Analysis

In order to analyse the accuracy of the tracking, an error analysis is carried out. The error for each prediction is computed as the Euclidean distance between the predicted and true position. Note that during training and evaluation the images are normalized so that values in x and y range from 0 to 1. This means that an error of for example 0.2 corresponds to 20 % of the image size.

The error analysis for models trained and evaluated on data with identical bacteria is shown in Figure 4.1. Figure 4.1 (a) shows that the RNNs have lower error distributions compared to the CNNs. Moreover, the classification loss raises the error distribution for the CNN, while for the RNN it lowers the distribution. Note that the means of all distributions are significantly different (p-values from Welch's t-test on ranked data $\ll 0.05$). In order to analyse outliers, which are hidden in (a) for clearer visualisation, the probability density functions are shown in Figure 4.1 (b). It shows that the RNNs give narrower error distributions compared to the CNNs. The classification loss does not affect the amount of outliers for the CNNs, while for the RNN the amount is increased. Furthermore, both RNNs and CNNs outperform a model always predicting in the centre, see Appendix C.

The error analysis for models trained and evaluated on data with varying bacteria is shown in Figure 4.2. Figure 4.2 (a) shows that the RNNs have lower error distributions compared to the CNNs also on this data. The means of all error distributions are significantly different according to Welch's t-test on ranked data. Specifically, the classification loss significantly decreases the means of the errors for both the CNN and the RNN (p-values $\ll 0.05$). The probability density functions for the errors, shown in Figure 4.2 (b), show that the RNNs have narrower distributions compared to the CNNs. The density function for the RNN with L1 and classification loss is narrower compared to all other models. Again, both RNNs and CNNs outperform a model that always predicts in the centre, see Appendix C.

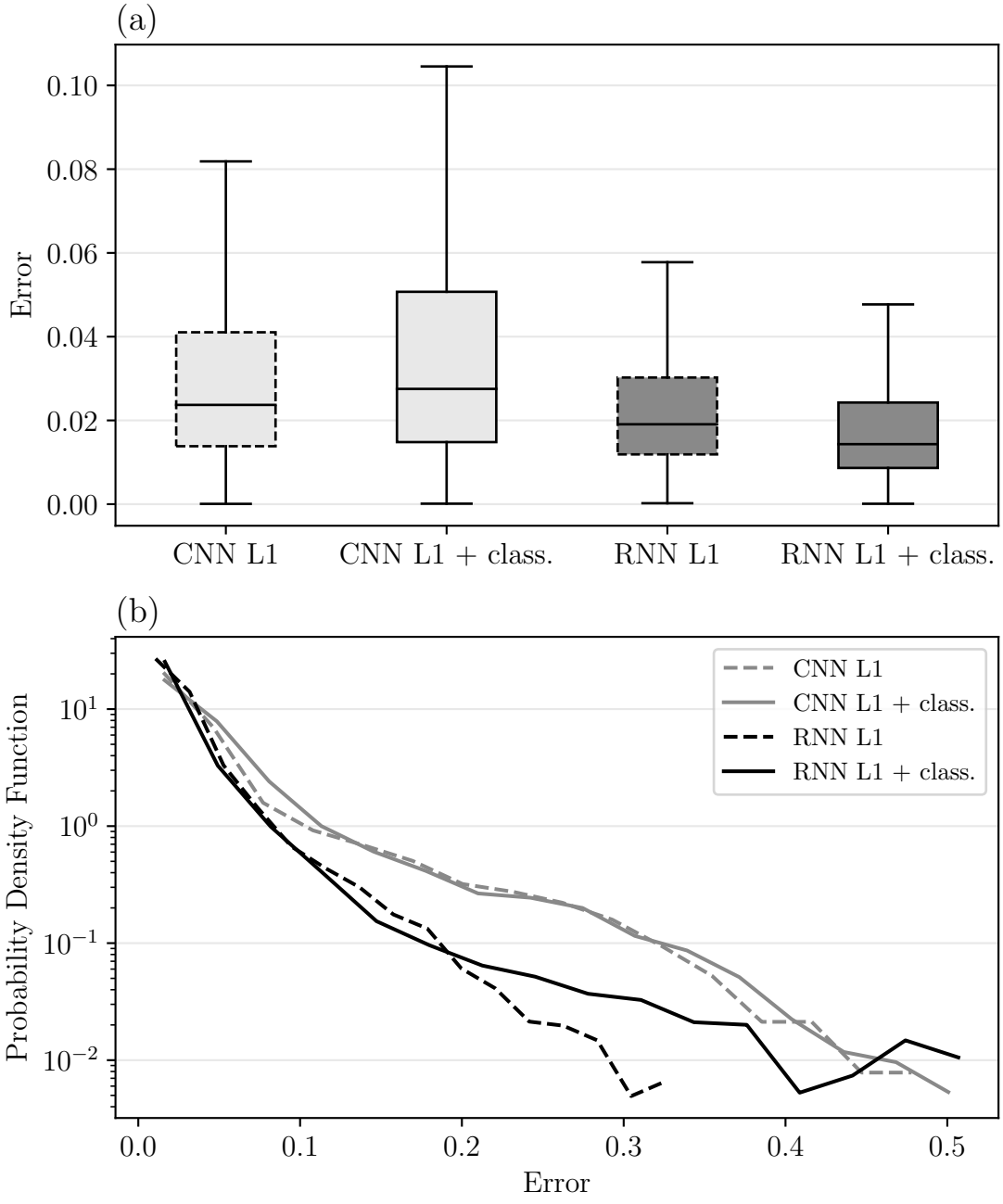

Figure 4.1: Errors for the RNNs and CNNs evaluated on data with identical bacteria. The errors are computed as the Euclidean distance between the predicted and true position. The box plot in (a) shows the error distributions, without any outliers for clearer visualisation. The probability density functions for the errors are shown in (b), with the y-axis in log scale. Note that the lines end at different positions because there are no larger errors for the models at these points.

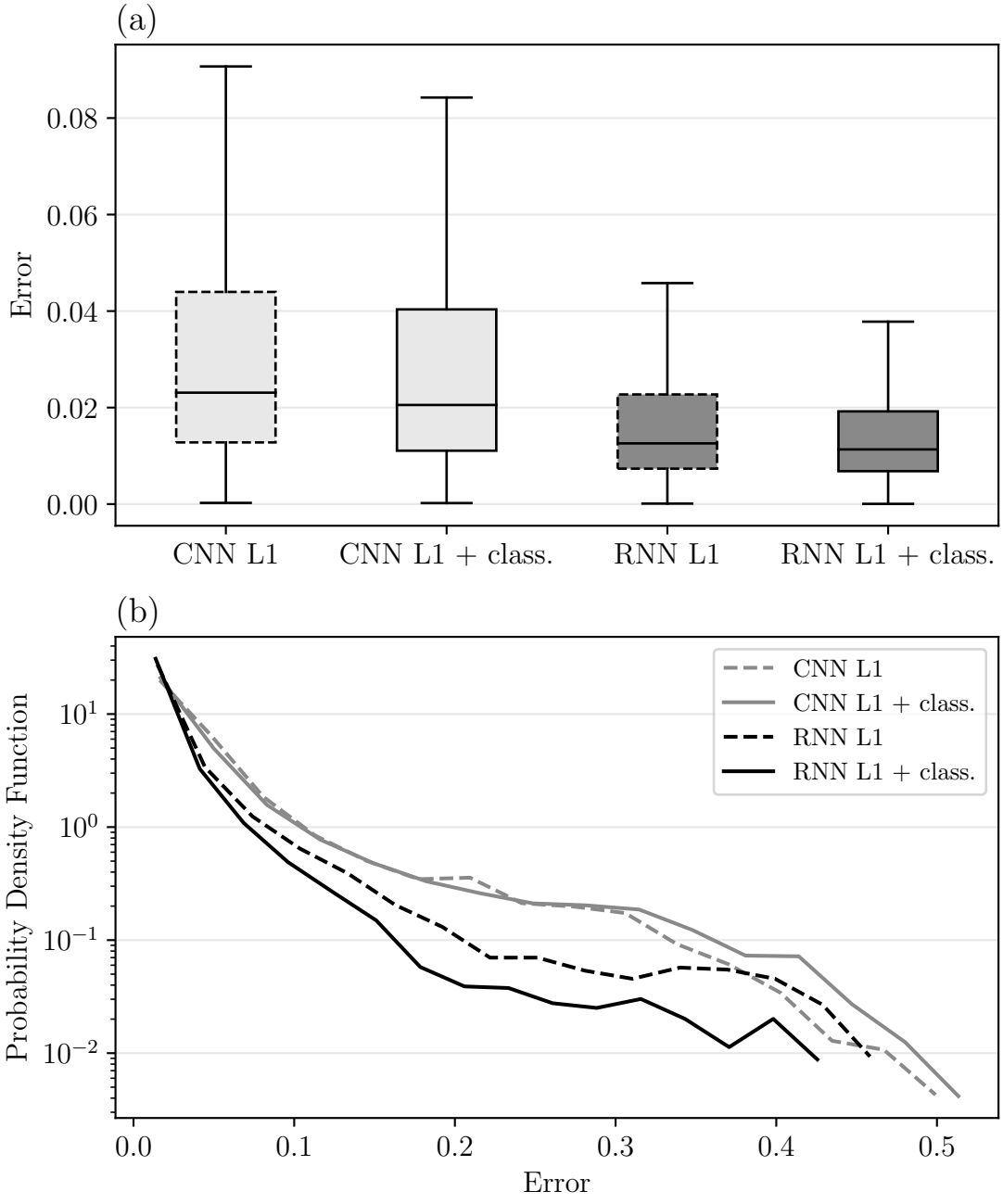

Figure 4.2: Errors for the RNNs and CNNs evaluated on data with varying bacteria. The errors are computed as the Euclidean distance between the predicted and true position. The box plot in (a) shows the error distributions, without any outliers for clearer visualisation. The probability density functions for the errors are shown in (b), with the y-axis in log scale. Note that the lines end at different positions because there are no larger errors for the models at these points.

In order to understand where in the frame the errors occur, an analysis of where in the image the errors are large is carried out. For models evaluated on the dataset with identical bacteria, Figure 4.3 shows the absolute errors in x and y for different values of the true position in x and y. The figure indicates that for all models the errors are largest when the true position in x or y is farther away from the centre, at 0.5. Comparing the RNNs and the CNNs it can be observed that farther away from the centre the RNNs have lower errors (lower means and lower standard deviations) compared to the CNNs. Regarding the classification loss there is no clear increase or decrease of the errors for the CNNs while for the RNNs error in x is larger for larger x positions when the classification loss is added.

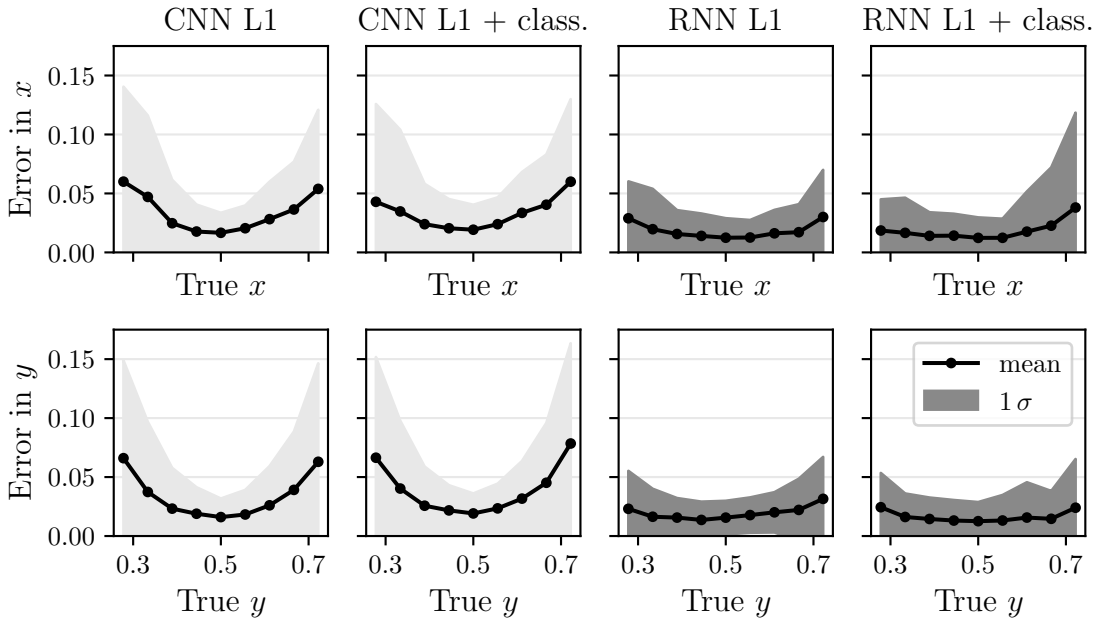

Figure 4.3: Absolute errors in x and y versus the true positions in x and y, for models trained and evaluated on data with identical bacteria. The binned statistics shows the mean and standard deviation for each bin, using a total of 10 bins. The range of true positions is based on the range of true positions in the test dataset 3 in Table 3.1.

For the models evaluated on data with varying bacteria, Figure 4.4 shows the absolute errors in x and y for different values of the true position in x and y. The figure shows, similar to Figure 4.3, that the RNNs have smaller errors farther away from the centre of the image compared to the CNNs. Adding the classification loss gives lower errors farther away from the centre for the RNNs, while no difference is observed for the CNNs.

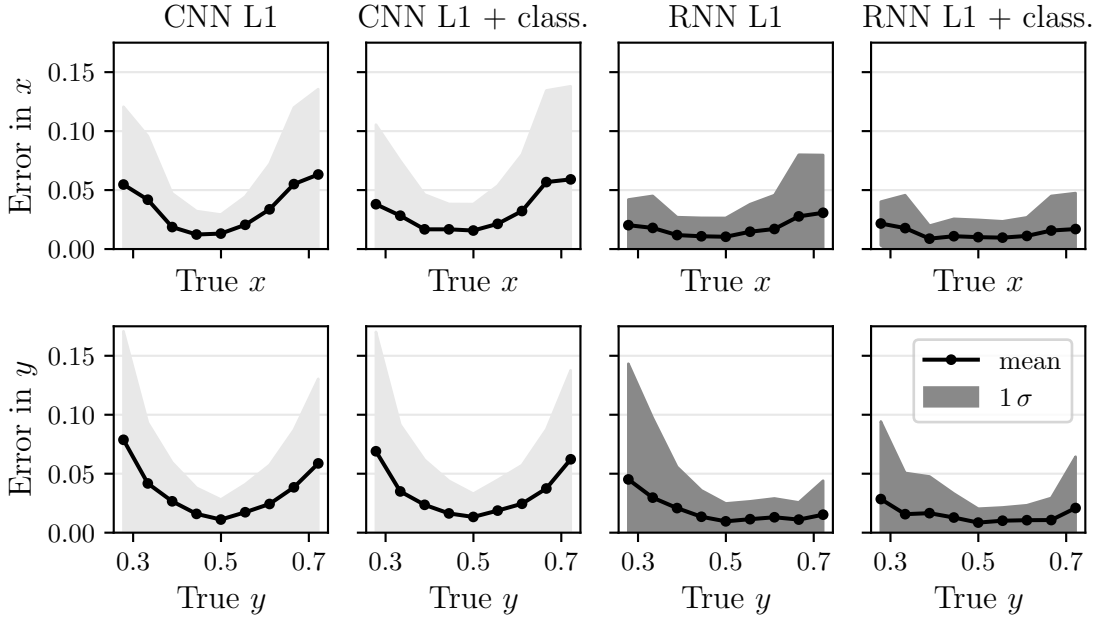

Figure 4.4: Absolute errors in x and y versus the true positions in x and y, for models trained and evaluated on data with varying bacteria. The binned statistics shows the mean and standard deviation for each bin, using a total of 10 bins. The range of true positions is based on the range of true positions in the test dataset 7 in Table 3.1.

4.2 Analysis of Overlapping Bacteria

A crucial goal for the new developed model is the ability to handle overlapping bacteria. Although the error analysis gives a general assessment of the models' performances it does not specifically test the models' abilities to handle overlapping bacteria. Therefore, an analysis of how the models handle overlapping bacteria is conducted. An example of a sequence used for the analysis of overlapping bacteria is shown in Figure 4.5. Here an additional bacterium moves across the frame from one edge to the opposite one. In this specific sequence the predictions of the RNNs stay close to the centre bacterium, while the predictions of the CNNs move towards the additional bacterium. The overlap analysis below includes a binary classification, determining whether the predictions are closer to the centre or additional bacterium, as well as a deeper analysis of the distributions of softmin values, corresponding to the probabilities that the centre bacterium is just that, the centre bacterium.

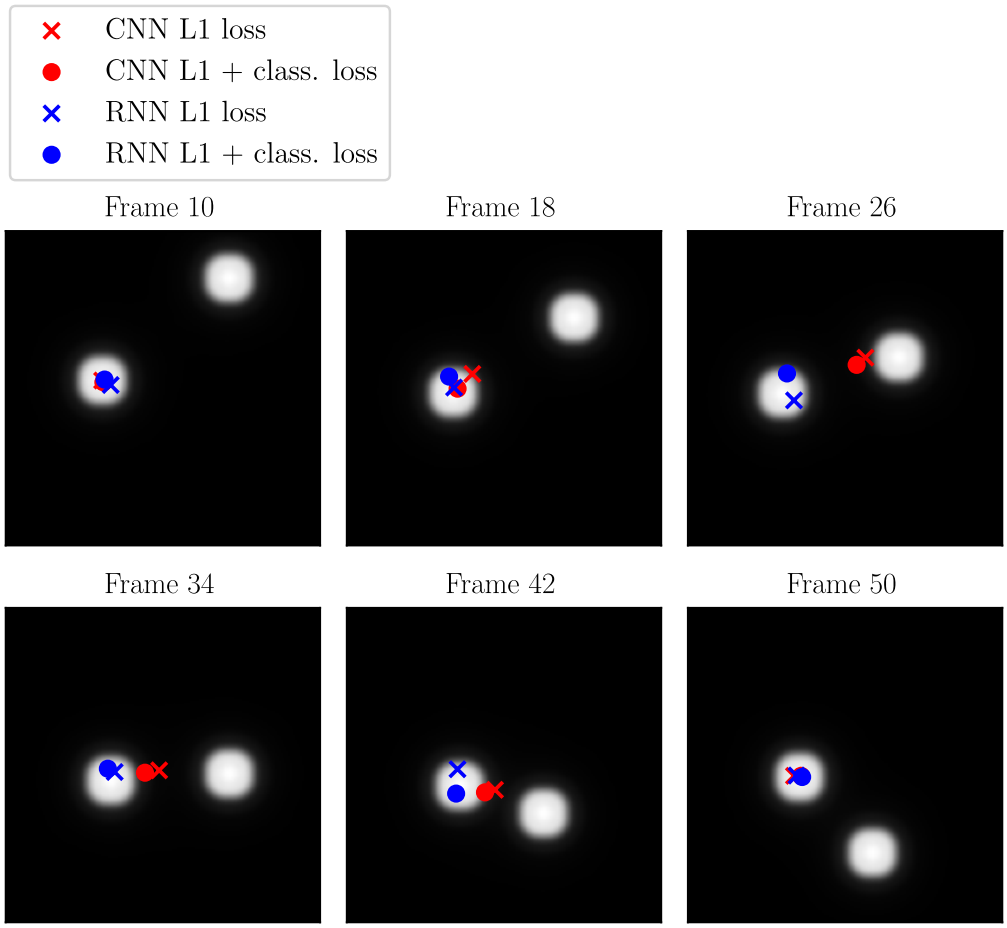

Figure 4.5: Example of frames and predictions from a sequence used for testing how models handle overlapping bacteria. In this example the centre bacterium is located to the left in frame 10, while the additional bacterium starts at the top of the image in frame 10. During the sequence, the additional bacterium moves from the top to the bottom of the image. The markers show the predictions of the position of the centre bacterium by the different models.

The result of the overlap analysis for models trained on identical bacteria is shown in Figure 4.6. The binary classification in Figure 4.6 (a) shows that the fraction of frames where the prediction is closer to the additional bacterium is 0.03 for the RNNs. For the CNNs on the other hand, the fraction of frames where the prediction is closer to the additional bacterium is nearly 0.25, which is larger compared to the model that always predicts in the centre of the image (0.23). The distribution of softmin values for each model is shown in Figure 4.6 (b). The figure shows that the distributions for the RNNs have larger mean values compared to the CNNs and the model that always predicts in the centre of the image. In practice, this means that the predictions of the RNNs are in general closer to the true position compared to the other models. Although the "centre"-model and the CNNs have similar fraction of frames where the softmin value is below 0.5 (the same value as in the binary classification), the CNNs have higher means compared to the "centre"-model. The classification loss increases the mean for the RNNs (p-value from Welch's t-test $\ll 0.05$ on ranked data), while no significant difference in the means for the two CNNs is observed ($p = 0.33$ in Welch's t-test on ranked data).

The result of the overlap analysis for models trained on varying bacteria is shown in Figure 4.7. The binary classification in Figure 4.7 (a) shows that the fraction of frames where the prediction is closer to the additional bacterium is lower for the RNNs (around 0.04) compared to the other models (around 0.24). On this data, the model always predicting in the centre of the image performs similar to the CNNs. The distribution of softmin values for each model, shown in Figure 4.7 (b), shows that the distributions look very similar to those in Figure 4.6 (b). The distributions of the RNNs have higher means compared to the CNNs and the "centre"-model. There is no significant difference of the means between the RNNs ($p = 0.71$) or between the CNNs ($p = 0.34$), according to Welch's t-test on ranked data.

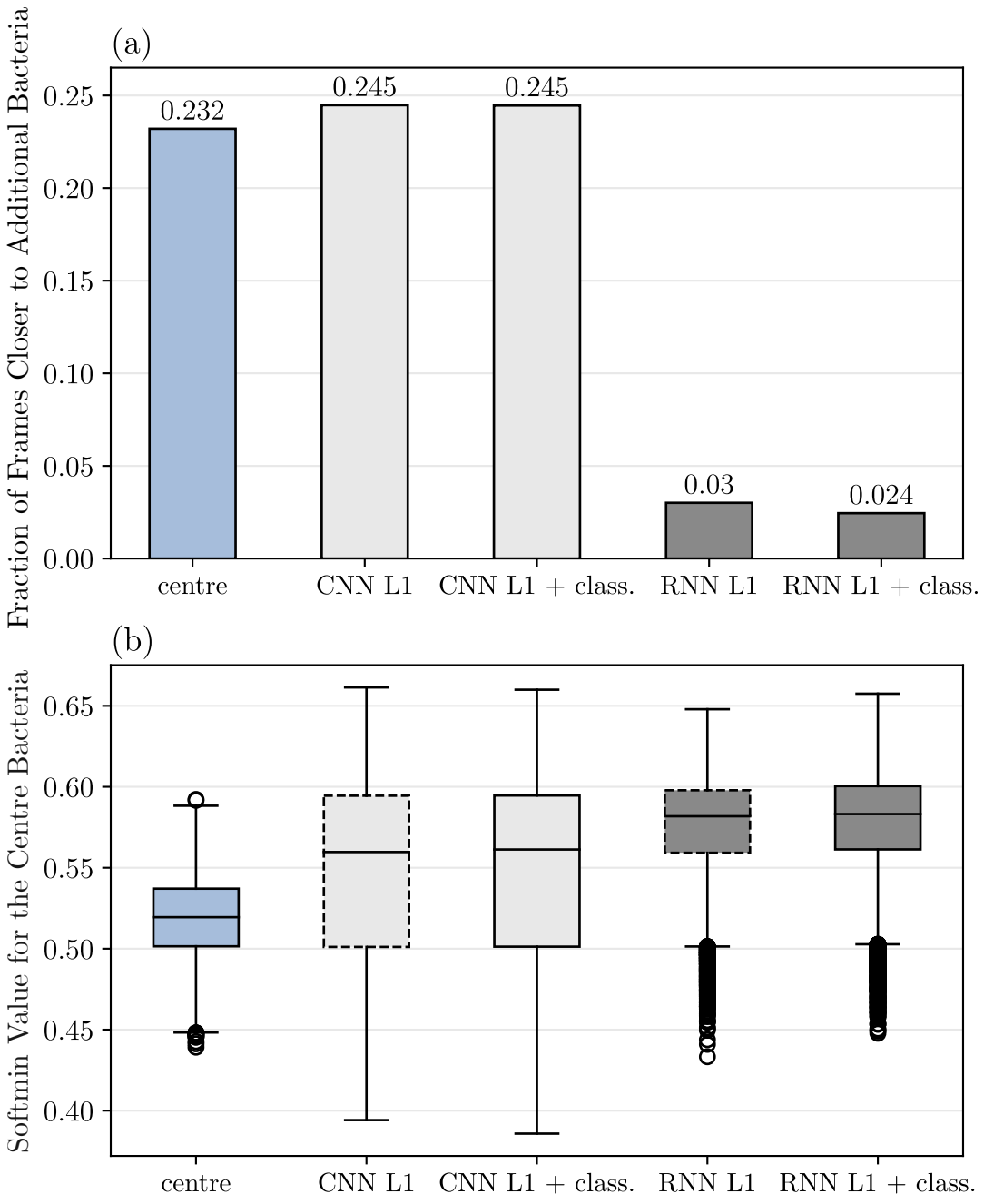

Figure 4.6: Result of overlap analysis for models trained and evaluated on data with identical bacteria. In (a) the fraction of frames where the prediction is closer to the additional bacterium compared to the centre bacterium is shown. In (b) the distributions of softmin values corresponding to the centre bacterium is shown. The softmin values are computed based on the distances from the prediction to the centre respectively additional bacterium. Note that the model "centre" is a model for reference, where the prediction is always in the centre of the image.

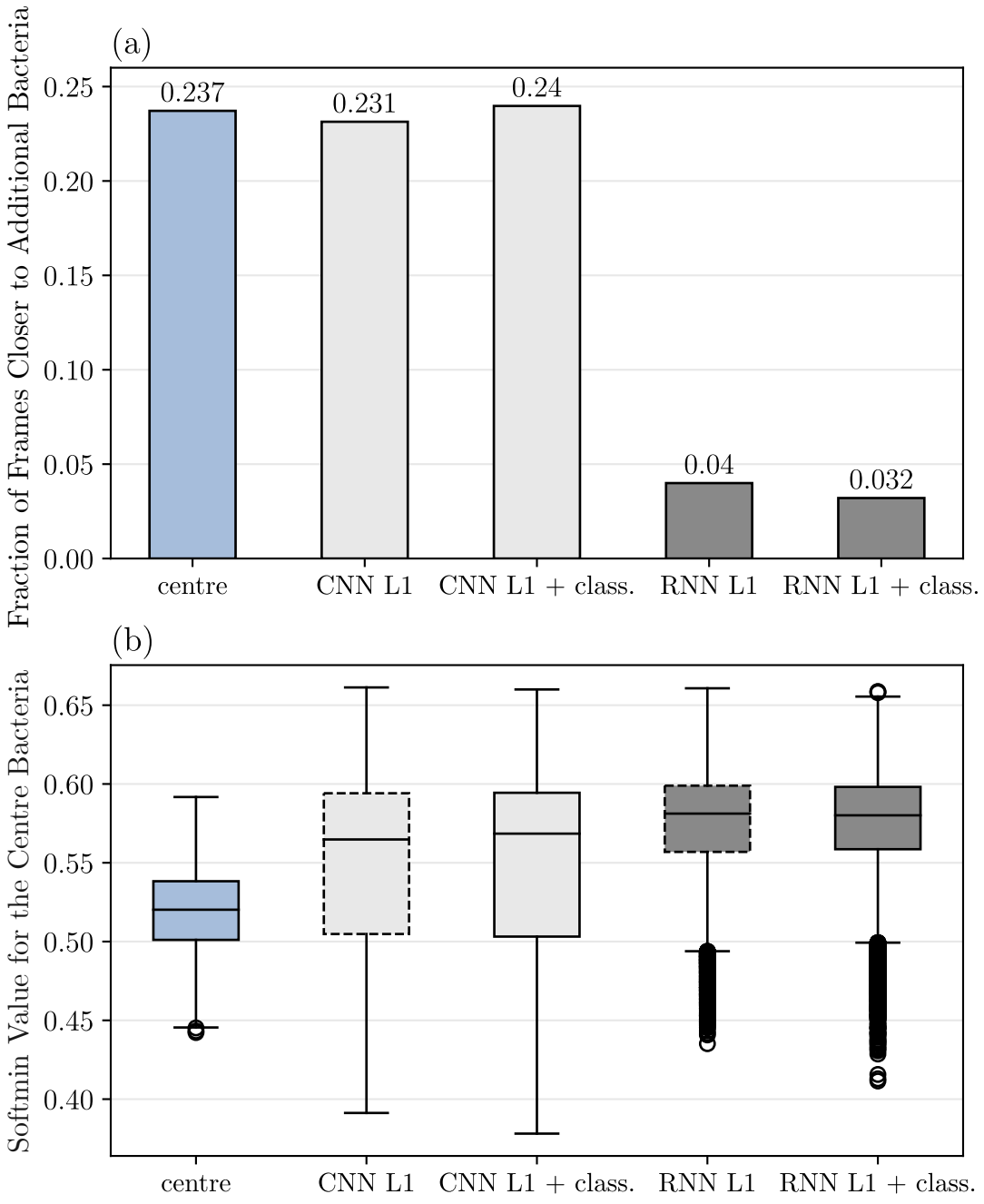

Figure 4.7: Result of overlap analysis for models trained and evaluated on data with varying bacteria. In (a) the fraction of frames where the prediction is closer to the additional bacterium compared to the centre bacterium is shown. In (b) the distributions of softmin values corresponding to the centre bacterium is shown. The softmin values are computed based on the distances from the prediction to the centre respectively additional bacterium. Note that the model "centre" is a model for reference, where the prediction is always in the centre of the image.

4.3 Important Features for Prediction

To understand what features of the bacteria influence the tracking, an analysis of the important features for tracking is conducted. The features analysed are the positions along the z-axis, the radii and the intensity of the bacteria. For each feature, the centre bacterium is given a fixed value according to Table 3.2 while the value of the feature for the additional bacterium is varied. For reference, a model that always predicts in the centre of the image gives a fraction of frames where the prediction is closer to the additional bacterium of 0.24. Since the positions and the movements of the bacteria are the same in all datasets, this value remains the same.

The results of the analysis are shown in Figure 4.8. In Figure 4.8 (a), where the position in z is varied, the fraction of frames where the prediction is closer to the additional bacterium is higher when the additional bacterium is closer to the focal plane at $z = 0$. This is true when the centre bacterium is below, at, and above the focal plane, but the fraction of frames is lower when the centre bacterium is also at the focal plane. Furthermore, the fraction of frames is slightly higher when the additional bacterium is above the focal plane compared to when it is below. Figure 4.8 (b), where the radius is varied, shows that when the radius of the additional bacterium deviates from 1 µm, the fraction of frames increases. This is true for all three values of the radius of the centre bacterium. By increasing the radius of the additional bacterium further, the fraction of frames decreases. By instead decreasing the radius of the additional bacterium further, the fraction of frames increases before it decreases again. Figure 4.8 (c), where the intensity is varied, shows that the fraction of frames increases as the intensity of the additional bacterium increases. Similarly, the fraction of frames decreases as the intensity of the additional bacterium decreases. Note that since the intensities of the bacteria are relative values, it is sufficient to analyse only the case when $I_c = 1$.

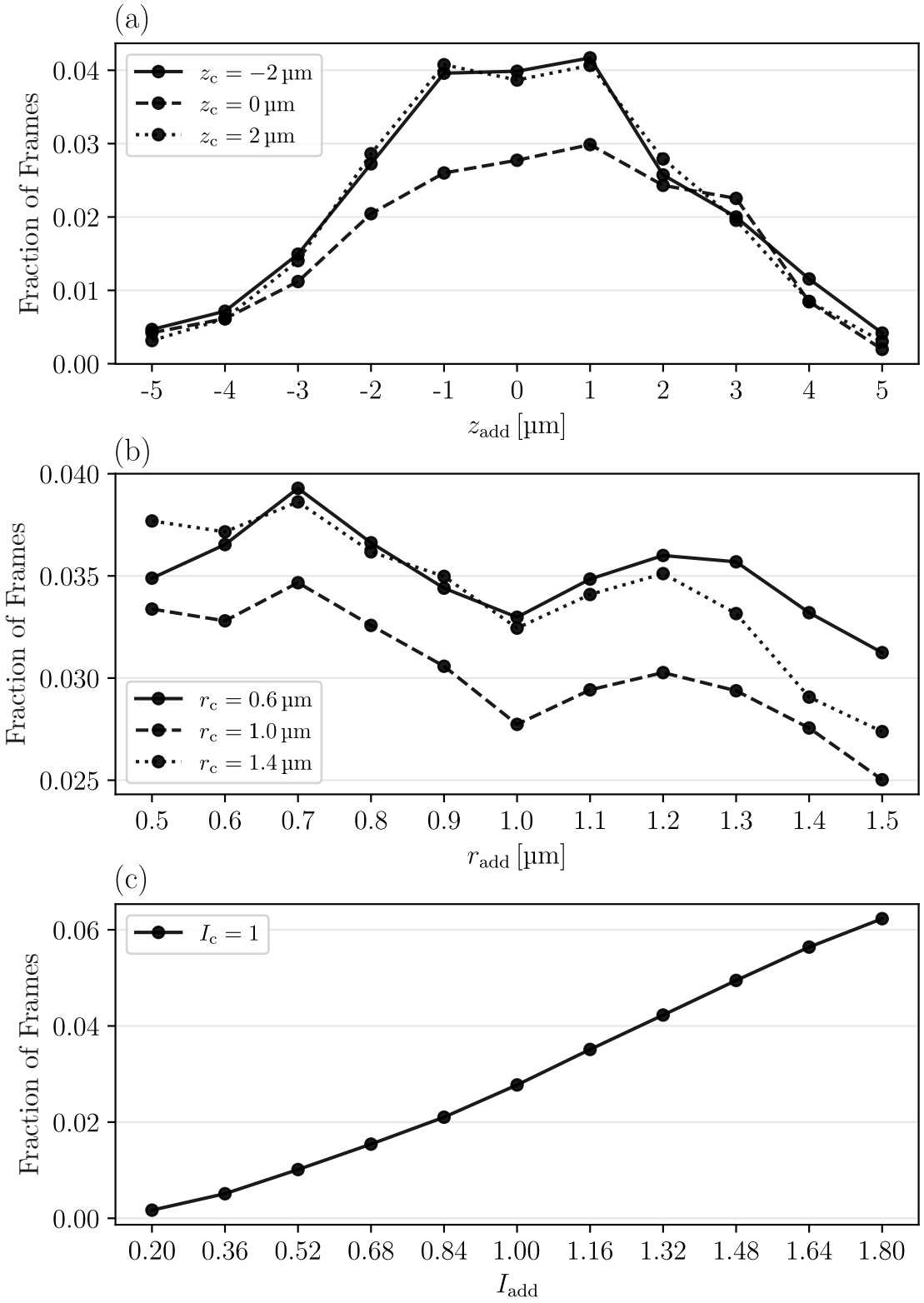

Figure 4.8: Analysis of importance of features of the bacteria for tracking. The features varied are positions along the z-axis in (a), the radii of the bacteria in (b), and the intensities of the bacteria in (c). The default values for all parameters are given in Table 3.2. Note that the y-axis indicates the fractions of frames where the prediction is closer to the additional bacterium compared to the centre bacterium.

5 Discussion

In this thesis, RNNs were trained and evaluated with the aim to improve tracking of bacteria using Lagrangian tracking. In addition, CNNs similar to a previous model were trained in order to benchmark the performance of the RNNs. The main result of the thesis was that the RNNs outperformed the CNNs in the error analysis as well as in the analysis of overlapping bacteria.

The RNNs gave lower error distributions compared to the CNNs when training and evaluating using data with identical bacteria as well as data with varying bacteria. Moreover, the RNNs gave fewer outliers and were able to track better when the centre bacterium was farther away from the centre, compared to the CNNs. In order to further improve tracking farther away from the centre, the training data would need to be adjusted, since the centre bacterium was kept around the centre of the image and hence the models learned these situations best.

The RNNs were better at handling overlapping bacteria compared to the CNNs. The predictions of the RNNs were closer to the additional bacterium in around 3 % of the frames compared to around 24 % for the CNNs, for both data with identical and data with varying bacteria. The initial hypothesis was that the RNNs would perform better on the dataset with varying bacteria, since the bacteria would be easier to distinguish when they were allowed to have varying size and intensity. The reason that this was not the case might be that the RNNs were not trained for enough epochs. Studying the training logs in Appendix B, it can be seen that the learning curves have not fully flattened out. It would be interesting to investigate if it is possible to improve the RNNs further on the dataset with varying bacteria or if the dataset itself was the limitation.

The analysis of feature importance, using the RNN trained with L1 and classification loss, showed that different features of the bacteria influenced the tracking differently. First, the position along the z-axis of the additional bacterium significantly influenced the tracking. Regardless of the z-position of the centre bacterium, an additional bacterium close to the focal plane made the tracking more difficult. This was reasonable since an additional bacterium near the focal plane could easily be mistaken for the centre bacterium, which was kept around the focal plane during training. Secondly, larger intensities of the additional bacterium made the tracking more difficult. This indicates that the RNN did not learn to track lower intensity bacteria so well when additional high-intensity bacteria passed the frame.

The results of the importance of the radius, on the other hand, were neither expected nor easily explainable. The initial expectation was that if the radius of the additional bacterium was smaller or larger than the centre bacterium, the tracking would be easier. However, no such expected pattern was observed. Instead the curves for the fraction of frames had the same notable shape for all values of r_c, see Figure 4.8 (b). One possible reason for the decrease of fraction of frames at $r_{add} = 1\,\mu m$ for all curves is that this point was in the middle of the range of radii allowed in the data. The model then learned to handle additional bacteria close to radius $1\,\mu m$ better compared to bacteria deviating from this radius. This however does not explain the decrease of fraction of frames for large radii of the additional bacteria. Another explanation is that the shape of the curve was an effect of the limited size of training data. The number of sequences in the training data was set to 500, which could be too small to include a large proportion of the possible combinations of radii of the centre and additional bacterium. With a smaller representation of possible combinations of radii it could happen that the model learned to handle additional bacteria closer to the median of the allowed range, at $1\,\mu m$. It would indeed be interesting to further investigate the effect of the radii of the centre and the additional bacterium in order to understand how they influence the accuracy of the tracking.

The effect of the classification loss during training is not completely clear. In most cases there was a significant increase of performance of the RNNs when training with the classification loss. However, on data with identical bacteria the RNN had more outliers in the error analysis and in the overlap analysis bacteria there was no significant improvement. Note that the choice of suitable test metrics to compare the distributions was not obvious since the data was non-normal, and that the significance of the differences could be affected by this. Regarding the effect of the classification loss for the CNNs, the performance was sometimes increases and other times decreased. The varying effects could be an effect of the CNN itself, as there is no obvious reason why the classification loss would benefit a model not taking memory of previous frames into account.

Although the RNNs showed better performances compared to the CNNs, the performances of the RNNs were not perfect. One example includes the fact that predictions of the RNNs were not always in the exact centre of the bacterium, see Figure 4.5. For the CNNs it is reasonable that the predictions deviated from the centre bacterium, since they only used the current frame to make predictions. During training the CNNs learned to predict in the middle of two overlapping bacteria, since this on average gave the smallest loss. For the RNNs however, the predictions should be able to stay in the exact centre of the centre bacterium. According to the training logs in Appendix B, it is possible to train all RNNs further, since no tendencies of overfitting are observed. It would be interesting to train the RNNs further to see if this affects the ability to predict in the centre of the bacterium. Moreover, training the models further could also make it possible to better understand the effect of the classification loss.

If the project would be conducted again, a few changes would be made to the method. The first change would be to train the RNNs for more epochs, in order to more accurately draw conclusions about the performance of the RNNs and the effect of the classification loss. Secondly, the models would be trained and evaluated on larger datasets, particularly for the dataset with varying bacteria. The reason would be to cover more combinations of features of the bacteria in order to get more reliable results. Finally, the number of additional bacteria would be the same for all datasets for training. In the dataset with identical bacteria up to three additional bacteria were used, while in the dataset with varying bacteria up to two additional bacteria were used. Although all datasets for evaluation had up to two additional bacteria, the different number of maximum additional bacteria in the training datasets could potentially have affected the possibility to compare the models' performances on data with identical respectively varying bacteria.

6 Conclusions and Outlook

This thesis aimed to develop a model for Lagrangian tracking of bacteria with the capabilities of accurate tracking and handling of overlapping bacteria. The results showed that the developed models, RNNs, gave lower error distributions compared to the CNNs similar to a previous model used for the task. Moreover, the results showed that the RNNs were better at handling overlapping bacteria compared to the CNNs. In summary, these results suggest that an RNN is more suitable as a model for Lagrangian tracking, compared to a CNN. The tracking using the RNNs can still be improved, for example by predicting closer to the centre of the bacterium and better track low-intensity bacteria. Furthermore, it still remains to test the RNNs in an experimental setup, or alternatively in a simulation similar to the experimental setup. Nevertheless, using an RNN can potentially decrease errors and to greater extent avoid losing the correct bacterium when an overlap happens, which is crucial for successful experiments and collection of data.

There are several additional performance analyses and alternative training methods that could contribute to the understanding of the RNNs' capabilities. Further analyses of the performance mainly include testing the RNNs in a setting that better reflects the actual experimental setup. In this case, one would test for how long the RNNs can track the bacterium if the output determines how the container with bacteria is moved and hence what the next frame looks like. An alternative training method of the RNN is to train using reinforcement learning. Here the frames in the sequence would be dependent on the output of the model, which would be penalised based on how long it is able to keep the correct bacterium in the centre of the frame. Furthermore, future studies should ideally also include refocusing of the bacterium, since this is necessary for tracking. Additionally, it would be interesting to test the limits of the capability of the RNNs, for example to investigate if the tracking of low-intensity bacteria can be improved.

Bibliography

[1] B. J. Ford and R. R. Shannon, *History of optical microscopes*, https://www.britannica.com/technology/microscope/History-of-optical-microscopes, Accessed: 2025-01-31, 2024.

[2] *Antonie van leeuwenhoek*, https://www.britannica.com/biography/Antonie-van-Leeuwenhoek, Accessed: 2025-01-31, 2024.

[3] M. I. Hutchings, A. W. Truman, and B. Wilkinson, "Antibiotics: Past, present and future," *Current Opinion in Microbiology*, vol. 52, pp. 72–80, 2019. DOI: 10.1016/j.mib.2019.10.008.

[4] V. C. Kalia, *Microbial Applications Vol.2*. Springer eBooks, 2017.

[5] *Press release*, https://www.nobelprize.org/prizes/medicine/2023/press-release/, Accessed: 2025-05-13.

[6] H. C. Berg and D. A. Brown, "Chemotaxis in escherichia coli analysed by three-dimensional tracking," *Nature*, vol. 239, pp. 500–504, 1972. DOI: 10.1038/239500a0.

[7] R. Kretschmer, *Molecules, Cells, and Parasites in Immunology*. Academic Press, 1980, pp. 91–102. DOI: https://doi.org/10.1016/B978-0-12-436840-8.50014-3.

[8] B. Midtvedt, S. Helgadottir, A. Argun, J. Pineda, D. Midtvedt, and G. Volpe, "Quantitative digital microscopy with deep learning," *arXiv*, 2020. DOI: 10.48550/arXiv.2010.08260.

[9] T. Darnige, N. Figueroa-Morales, P. Bohec, A. Lindner, and E. Clément, "Lagrangian 3d tracking of fluorescent microscopic objects in motion," *Review of Scientific Instruments*, vol. 88, 2017. DOI: 10.1063/1.4982820.

[10] H. C. Berg, "How to track bacteria," *Review of Scientific Instruments*, vol. 42, pp. 868–871, 1971. DOI: https://doi.org/10.1063/1.1685246.

[11] C. Boudreau, T.-L. (Wee, Y.-R. (Duh, M. P. Couto, K. H. Ardakani, and C. M. Brown, "Excitation light dose engineering to reduce photo-bleaching and photo-toxicit," *Scientific Reports*, vol. 6, 2016, Article ID 30892. DOI: 10.1038/srep30892.

[12] B. Mehlig, *Machine learning with with neural networks - An Introdution for Scientists and Engineers*. Cambridge University Press, 2022.

[13] *Leakyrelu*, https://docs.pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html, Accessed: 2025-05-27.

[14] M. I. Jordan, "Serial order: A parallel distributed processing approach," 1986.

[15] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, pp. 179–211, 1990. DOI: 10.1207/s15516709cog1402_1.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

[17] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," in *Artificial Neural Networks 1999*, 1999, pp. 850–855. DOI: 10.1049/cp:19991218.

[18] *Lstm*, https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html, Accessed: 2025-03-31.

[19] M. Gustafsson, *Masters_thesis*, https://github.com/mathilda1999/Masters_Thesis, 2025.

[20] B. Midtvedt, J. Pineda, H. Klein Moberg, H. Bachimanchi, C. Manzo, and G. Volpe, *Deeptrack2*, https://github.com/DeepTrackAI/DeepTrack2, 2024.

[21] G. D. Ruxton, "The unequal variance t-test is an underused alternative to student's t-test and the mann–whitney u test," *Behavioral Ecology*, vol. 17, pp. 688–690, 2006. DOI: 10.1093/beheco/ark016.

[22] P. Olofsson and M. Andersson, *Probability, Statistics, and Stochastic Processes*. Wiley, 2012, pp. 294–390. DOI: https://doi.org/10.1002/9781118231296.ch6.

A Usage of Artificial Intelligence

Artificial intelligence (AI) has been used in this thesis for different purposes. Since the aim of this thesis was to create and train deep learning models, machine learning has been used for this purpose, as explained in the Methods section. AI in the context of generative AI (chat GPT) has been used with the purpose of identifying typos, grammatically incorrect sentences and sentences that are for example too long, too complicated or with uncommon wording. Note that all text has been written by the author of this thesis and that AI has not been used to generate any text from scratch.

B Training Processes

The training process is shown in Figure B.1 for models trained on data with identical bacteria, and in Figure B.2 for models trained on data with varying bacteria.

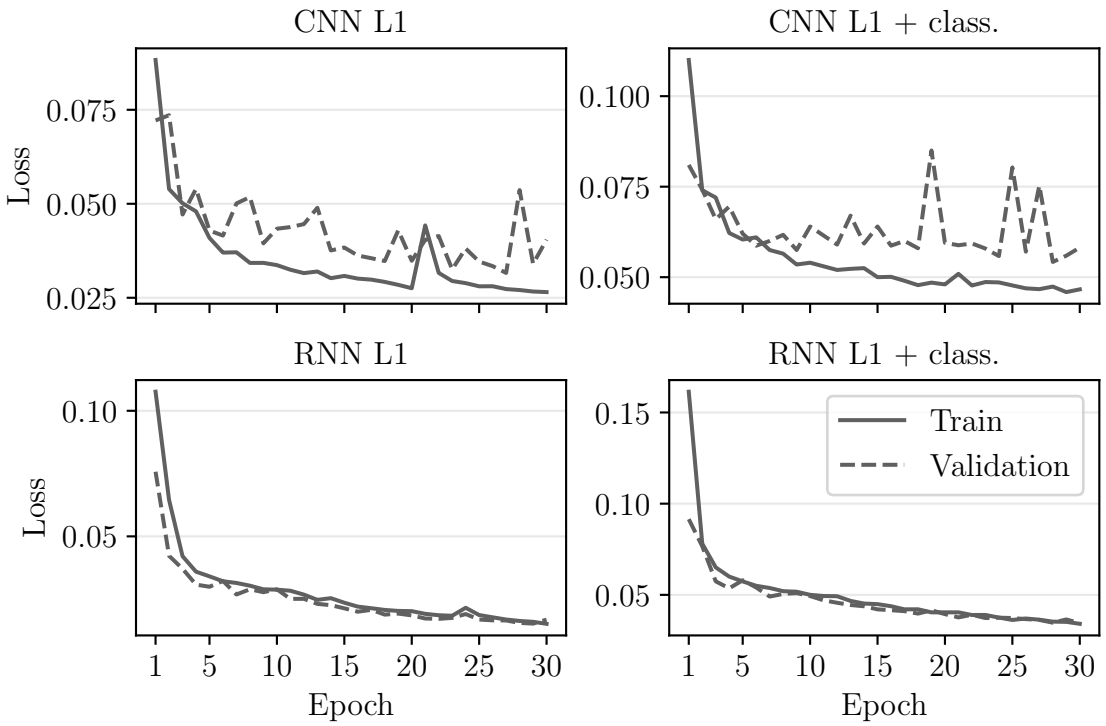

Figure B.1: Training processes for 30 epochs for models trained on data with identical bacteria.

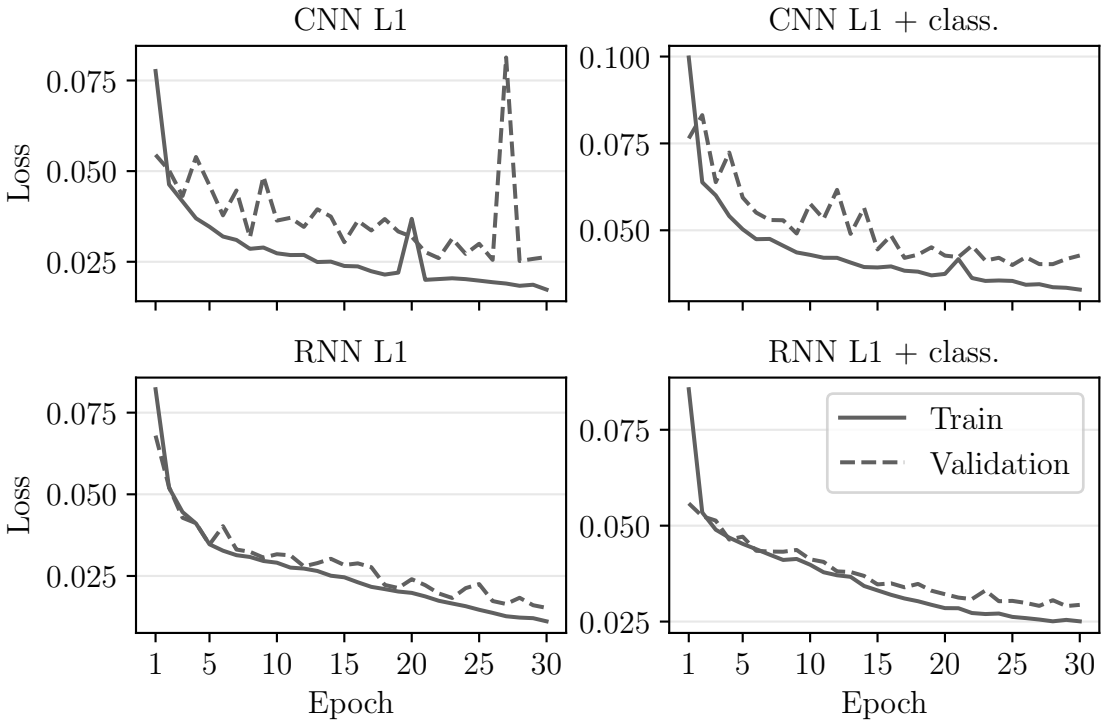

Figure B.2: Training processes for 30 epochs for models trained on data with varying bacteria.

C Additional Results

The results of the error analysis are shown in Figure C.1 for identical bacteria and in Figure C.2 for varying bacteria. The figures show that all RNNs and CNNs outperform the model that always predicts in the centre of the image.

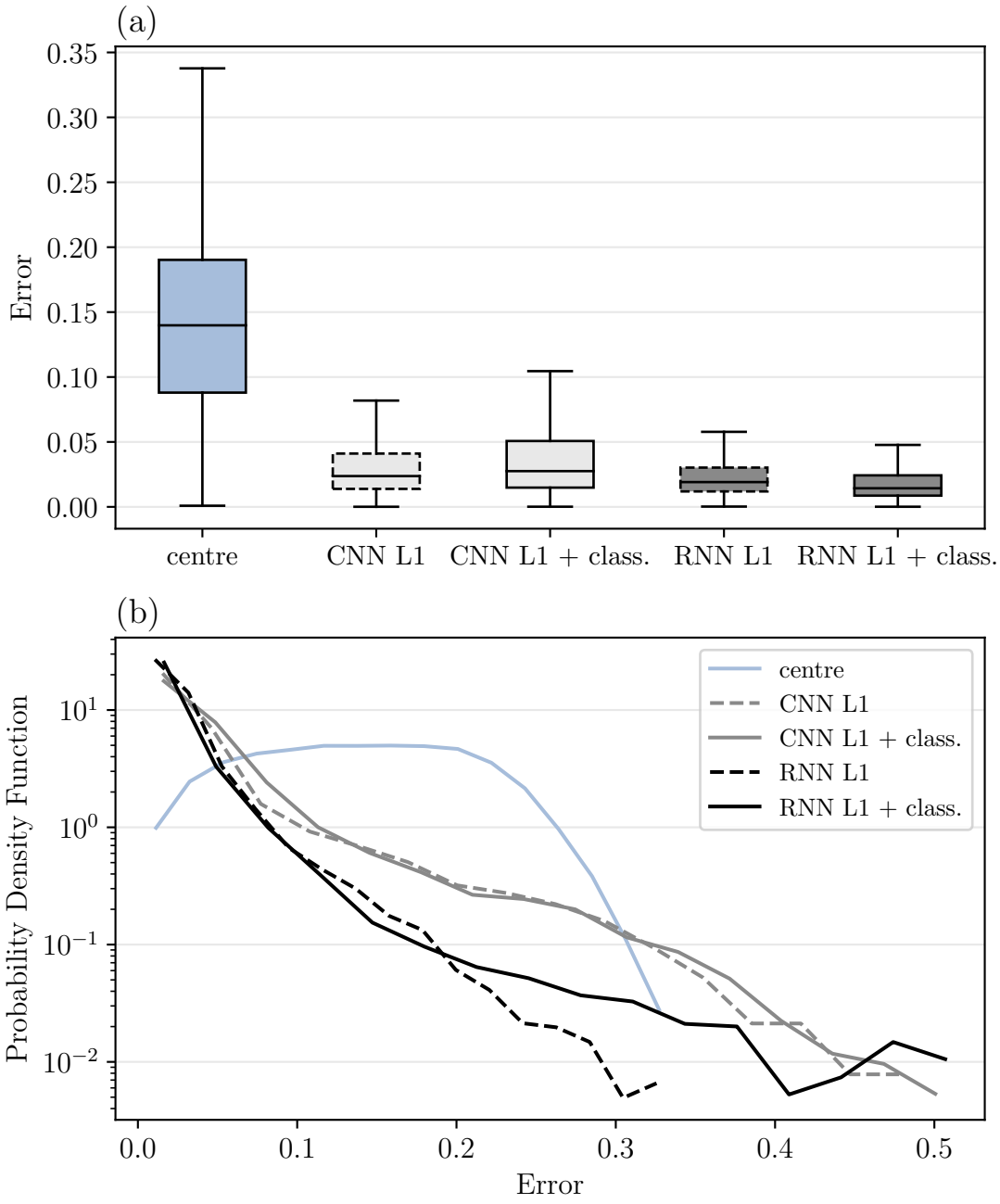

Figure C.1: Errors for the different models, including the model that always predicts in the centre of the image, evaluated on data with identical bacteria. The box plot in (a) shows the error distributions, without any outliers for clearer visualisation. The probability density functions for the errors are shown in (b), with the y-axis in log scale. Note that the lines end at different positions because there are no larger errors for the models at these points.

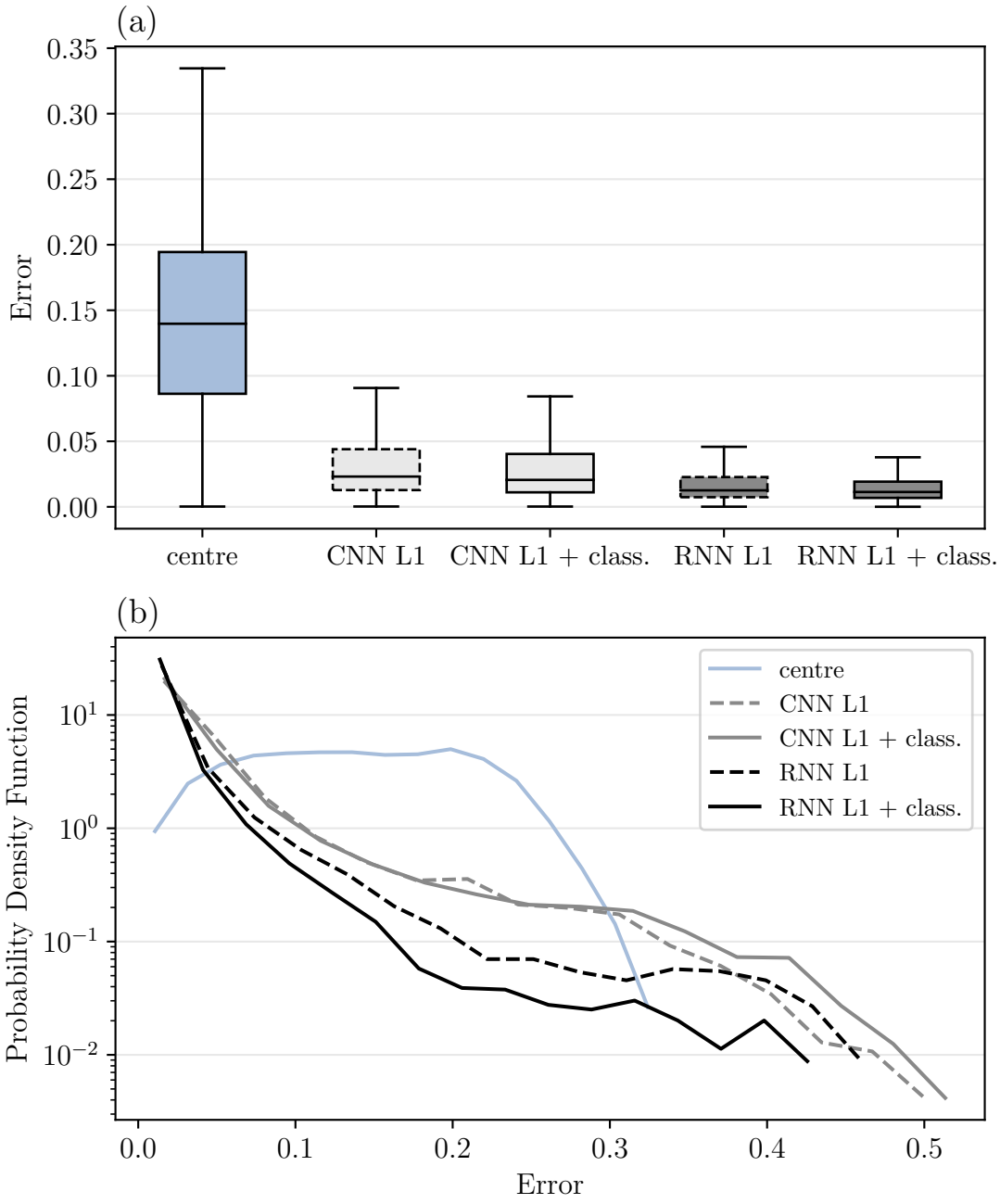

Figure C.2: Errors for the different models, including the model that always predicts in the centre of the image, evaluated on data with varying bacteria. The box plot in (a) shows the error distributions, without any outliers for clearer visualisation. The probability density functions for the errors are shown in (b), with the y-axis in log scale. Note that the lines end at different positions because there are no larger errors for the models at these points.

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY