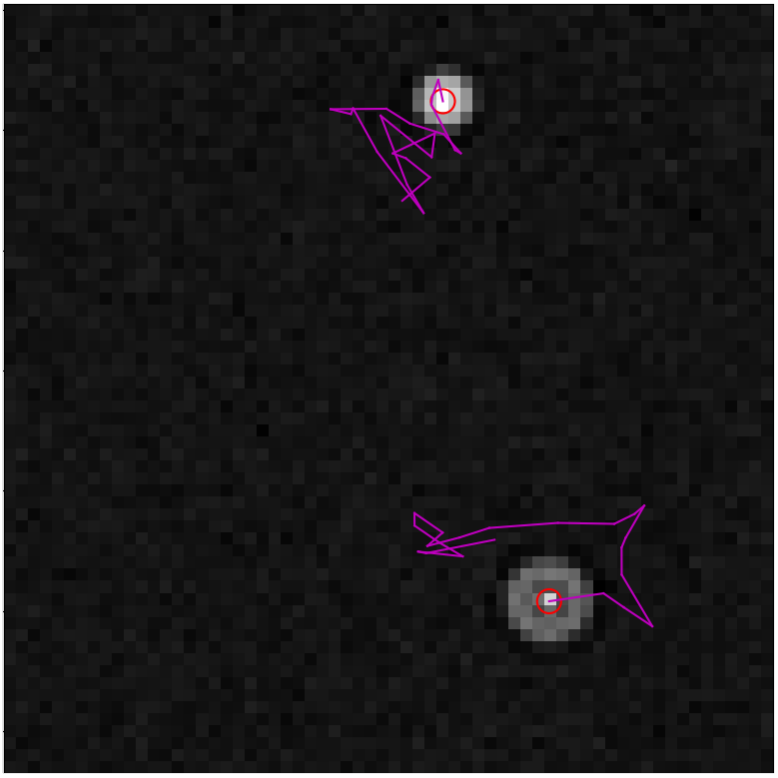


End-to-End Object tracking on simulated microscopy video

Using graph neural networks and variational autoencoders.

Master's Thesis in Complex Adaptive Systems

Gideon Jägenstedt

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

End-to-End Object tracking on simulated microscopy video

Using graph neural networks and variational autoencoders.

Gideon Jägenstedt

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2023

End-to-End Object tracking
on simulated microscopy video
Using graph neural networks and variational autoencoders.
Gideon Jägenstedt

Supervisor: Jesus Pineda, Department of Physics, University of Gothenburg
Examiner: Giovanni Volpe, Department of Physics, University of Gothenburg

Master's Thesis 2023:06
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Tracking of two simulated particles over 20 time-steps using the MVAE-based model presented in this work.

Typeset in LaTeX
Printed by Chalmers Digitaltryck
Gothenburg, Sweden 2023

End-to-End Object tracking
on simulated microscopy video
Using graph neural networks and variational autoencoders.
Gideon Jägenstedt
Department of Physics
Chalmers University of Technology

Abstract

Object tracking using neural networks has become a pivotal technology in digital microscopy, enabling automated analysis and interpretation of dynamic visual data. Typically, this tracking is performed in two steps, often utilizing two separate neural network models. First, images are segmented to detect individual objects within each time frame and to extract their centroids and other features using one neural network model. Next, a selected set of features from these objects is used as input to a second neural network, which creates temporal trajectories by linking the objects across a sequence of frames.

This work proposes a novel method that combines the object detection and linking steps, theoretically leading to better temporal links. Instead of relying on a fixed set of properties for each object, the combined model has access to the entire image and to temporal information, enabling it to autonomously select the most relevant features for optimal tracking. Two different architectures for a combined model were tested: a supervised model based on a graph neural network (GNN) and an unsupervised model based on a variational autoencoder (VAE).

The supervised GNN-based model did not succeed in predicting the position of the centroids, but it showed promise in linking the centroids between frames. Therefore, the VAE-based model was developed that uses the same approach for linking. The VAE-based model showed promising results with a mean absolute error of under 0.002 on its detection placement, a detection miss-rate of 2.69 %, and an F1-score of 81.2 % when tracking simulated data.

Keywords: object tracking, microscopy, geometric deep learning, variational autoencoders, adaptive particle representation.

Acknowledgements

Firstly, thanks to my Examiner Giovanni Volpe and my supervisor Jesus Pineda for being of huge help during the project.

Further, I would like to thank Joel Jonsson and Krzysztof Gonciarz at the Max Planck Institute of Molecular Cell Biology and Genetics for their insights into adaptive particle representation. I would also like to thank Mirja Granfors for a great collaboration on the multihead graph attention network as well as for being a great opponent.

Lastly, I would like to thank my girlfriend, Sara Larsson, for supporting me and helping me structure my thesis.

Gideon Jägenstedt, Gothenburg, June 2023

Contents

List of Figures

List of Tables

1
Introduction

One of the most common methods for object tracking in microscopy images using neural networks involves a two-step process [1][2]. Initially, all objects in each frame are detected by a segmentation model [3]. This gives a segmentation map from which the positions of individual objects can be extracted along with other relevant properties, such as the morphology and brightness of the object. The next step involves using the extracted properties to link the objects across frames. For this, one commonly employed technique is to compare the positions of the extracted objects in each frame and match them with the closest object in the next frame. This can be achieved through either algorithmic methods [4][5] or the utilization of a second neural network [6][7].

While these approaches have been successful in tracking microscopic objects, the linking step heavily relies on receiving sufficient relevant data to establish connections between the objects through time. This can become particularly challenging in, for example, scenarios where objects undergo significant movement or morphological changes. In such cases, relying solely on the position and morphology of objects as input may not yield accurate linking predictions. To get more accurate results, it becomes necessary to extract additional features and add them to the input of the linking model. However, determining the optimal set of object features to include for achieving the best results is a difficult task, and extracting these can be laborious.

To tackle this problem, this paper proposes employing a neural network model capable of extracting both the detections and links directly from the latent space. By doing so, the model autonomously determines the relevant characteristics to use, thus eliminating the uncertainty associated with manually selecting the object's features. Moreover, the maintenance will decrease, and computational efficiency increase since only one model will be used. In this work, two different types of neural networks have been proposed and investigated to solve this task: graph neural networks and variational autoencoders. Minor optimization analyses for the two models are also presented, along with test results from tracking particle objects in simulated microscopy videos using the two neural network models.

2
Theory

To provide the necessary background information, this chapter covers the fundamentals of neural networks and their mechanisms. It includes a brief explanation of convolutional neural networks, autoencoders, attention layers, and graph neural networks and concludes with a description of adaptive particle representation.

2.1 Neural Networks

Neural networks are a class of machine learning algorithms that are, as the name suggests, modeled after the structure and function of the neurons inside a human brain [8]. A neural network consists of a large number of connected nodes, called neurons, organized into layers. The outputs of the neurons in one layer serve as inputs to the neurons in the next layer. This forms a network structure consisting of an input layer, any number of hidden layers, and an output layer. The number of layers and the number of neurons in each layer depends on the specific problem being solved. Deep neural networks, which have multiple hidden layers, are capable of learning hierarchical representations of data and can handle more complex tasks.

The network is trained in two repeated steps. In the first step, commonly referred to as the forward pass, data flows from the input layer to the output layer. The computation occurs one layer at a time with each neuron computing a weighted sum of its inputs and a bias term. An activation function is then applied to the sum that will be the input to the next layer. The activation function introduces non-linearity, allowing the network to model complex relationships between inputs and outputs. Equation. 2.1 shows the computation of the layer output y, using the inputs x, the neuron weights w, the bias w_0, and the activation function σ.

$$y = \sigma\left(w_0 + \sum_{j=1}^{m} x_j w_j\right) \tag{2.1}$$

The second part of a neural network's training is called backpropagation, and it is commonly done using optimization algorithms such as gradient descent. During backpropagation, the output of the network gained from the forward pass is compared to the desired output. The error between these is propagated backward through the network. The gradients of the weights and biases with respect to the

error are computed using the chain rule of calculus and used to update the weights.

These two processes are repeated for multiple iterations, or epochs, with new inputs and targets until the network learns to produce accurate outputs for the given inputs. This way, the network will slowly get better at predicting the targeted output over many training iterations.

2.2 Convolutional neural networks

Convolutional neural networks (CNNs) are specialized neural networks designed to analyze structured grid-like data, such as images. Utilizing a fully connected network for image data becomes increasingly challenging due to the rapid growth in the number of neurons and connections as the image size expands. This results in large computational requirements during training as well as a higher risk of overfitting the model. By incorporating the underlying assumption that the inputs are images into the network architecture, CNNs can reduce the number of parameters needed, addressing these limitations.

Images are built up by 2D matrices where features can be extracted by convolution, which is utilized in the CNN by convolutional layers. Each layer uses a set of filters or kernels that, with a given stride, slide over the image and compute the output using element-wise multiplications and summations. The resulting matrices highlight the presence of different patterns at various spatial locations in the input and are called feature maps.

CNNs are often built with multiple convolutional layers to capture hierarchical representations, enabling the network to learn increasingly complex and abstract features. Low-level features like edges and textures are, therefore, often learned in the earlier layers, and higher-level concepts like shapes and objects in the deeper layers. Pooling layers are a special kernel that is used to downsample the feature maps and reduce their spatial dimensions. This downsampling helps reduce the computational complexity of the network and makes it more robust to small spatial variations. After several convolutional and pooling layers, the network typically ends with one or more fully connected layers. The fully connected layers integrate the high-level features extracted by the convolutional layers and map them to the desired output, such as class labels.

CNNs have revolutionized the field of computer vision and have achieved impressive results in tasks like image classification [9], object detection [10], image segmentation [11], and many more [12]. Their ability to automatically learn spatial hierarchies and extract meaningful features from raw input data makes them highly effective in analyzing visual information. Furthermore, CNNs have also been applied to other domains such as natural language processing, where they can learn representations from sequential data like text or speech [13].

2.3 Autoencoder

An autoencoder is a type of neural network architecture designed for unsupervised learning and dimensionality reduction [14]. It consists of an encoder and a decoder that together learns a compressed representation of the input data and reconstructs the original input from this compressed representation. The encoder applies a series of transformations on the input sample to map it to a lower-dimensional latent space representation. The transformation often involves several network layers with non-linear activation functions to extract the most salient features of the input data. The output of the encoder is then the latent code, which captures the essential information of the input in a lower-dimensional space. The decoder then takes the latent code as input and aims to reconstruct the original input data to its original form. It applies a series of inverse network layers that mirror the layers in the encoder.

The objective of training an autoencoder is to minimize the difference between the original input and the reconstructed output. Therefore, a loss function is used to measure the dissimilarity between the input and its reconstruction, such as the mean squared error. The weights and biases of both the encoder and decoder networks are then adjusted during training to minimize this reconstruction error. By reducing the dimensions of the latent space, the autoencoders learn to capture only the most important features of the input data and discard less relevant information. This dimensionality reduction property makes autoencoders useful for tasks such as data compression, denoising, and anomaly detection. The latent space could also be used to extract other desired output information.

2.3.1 Variation autoencoder

A variational autoencoder (VAE) [15] is a type of neural network model that combines the concepts of both autoencoders and generative models. Like a regular autoencoder, a VAE is composed of an encoder and a decoder. However, the key difference lies in the way the VAE learns to encode and decode data.

As mentioned, a regular autoencoder takes an input and maps it to a latent representation. This latent representation is then passed to the decoder which attempts to reconstruct the original input from this latent representation. A VAE, on the other hand, aims to learn a probabilistic model of the input data. It assumes that the input data follows a particular distribution, typically a standard normal distribution. The encoder in a VAE still maps the input to a latent space, but instead of producing a deterministic encoding, it generates a mean vector and a variance vector. These vectors represent the parameters of the normal distribution that the latent representation is assumed to follow, making the encoding probabilistic. These vectors are then sampled from to create latent features that can be used to reconstruct the input, as shown in Figure 2.1.

The probabilistic aspect of the VAE gives it the advantage of having a continuous

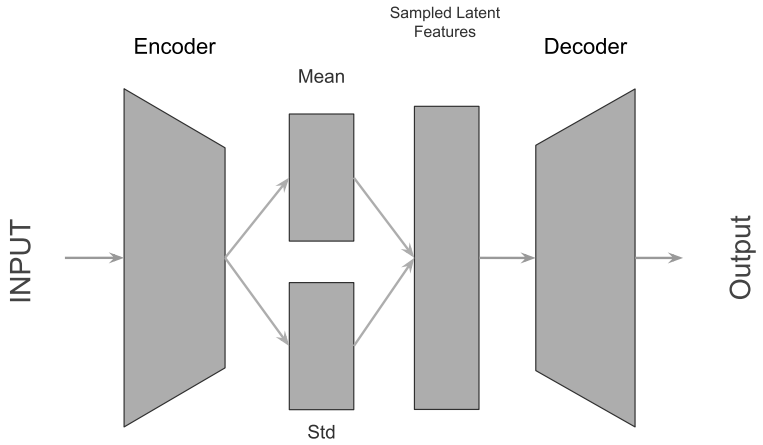


Figure 2.1: Variation autoencoder using an encoder-decoder structure. The input gets encoded into a latent distribution, which is then sampled from and passed through the decoder in order to reconstruct the input

latent space making it suitable for tasks such as feature exploration [16][17] and generative modeling [18][19][20].

2.4 Attention

Attention is a mechanism that allows neural networks to focus on different parts of the input data while performing computations. It is inspired by human cognitive attention, prioritizing certain elements of the input based on their relevance to the task at hand. In the context of neural networks, attention helps models weigh the importance of different parts of the input when making predictions or generating sequences. At its core, attention involves computing attention weights that indicate how much attention each element in the input sequence should receive when generating an output or making predictions. The attention weights are computed based on the interaction between the element being attended to (referred to as the ”query”) and other elements in the input sequence (referred to as the ”keys” and ”values”).

The attention process starts with calculating the similarity between the queries and keys. This similarity can be computed using various methods, such as the dot product or cosine similarity. These similarity scores are then turned into attention weights through a softmax operation. These weights reflect how much attention each key should receive when considering the current query. The attention weights are then used to combine the values associated with each key, resulting in a weighted sum. This sum represents the ”attended” representation of the input data for the

specific query. This representation is used in subsequent steps of the model and can be used in, for example, generating an output in machine translation [21] or text generation tasks [22].

2.4.1 Multihead attention

Multihead attention is an extension of the basic attention mechanism first presented along with the transformer neural network architecture [23]. It works by splitting the latent features into multiple ”heads” and calculating the attention on each head instead of using all latent features in one head. Each one of these heads has separate weights for its queries, keys, and values, allowing each head to focus on different aspects of the input data and enabling the model to capture various relationships and patterns simultaneously. The outputs from all of the heads are then concatenated and often transformed using a linear transformation giving us the whole attention that can be used in different ways depending on the task at hand.

2.5 Graph neural networks

A graph neural network (GNN) is a type of neural network architecture designed for handling data organized as graphs with nodes containing features and edges between the nodes representing relationships [24]. Most GNNs operate by iterative updating node features based on their own features and the features of neighboring nodes. This process allows nodes to gather and propagate information from connected nodes, capturing the influence of neighboring entities. Because graphs are a natural way of representing non-Euclidean data, graph neural networks (GNNs) have garnered significant attention in various domains and have been used in everything from modeling social networks [25][26] to analyzing biological systems [27][28].

2.5.1 Message passing and graph convolutions

Message passing layers are a core building block of GNNs. In this layer, information is passed between connected nodes in the graph to update their representations. A message passing layer can be divided into two main steps: message aggregation and message updating. In the message aggregation step, the layer aggregates information from neighboring nodes for each individual node. This is done by collecting ”messages” derived from the features of neighboring nodes. The messages are then aggregated by using some sort of aggregation function like summation or averaging. Following the aggregation, all of the collected messages are combined with the node’s current features and in doing so the information from the neighbors gets integrated into the node’s own features. The resulting updated node now has access to more contextual information enabling network architectures using these layers to capture relationships, dependencies, and patterns within the graph-structured data without losing the graph-based structure of the underlying data [29].

A graph convolutional layer is a type of message passing layer specifically designed to perform convolution-like operations on graph-structured data by aggregating in-

formation from neighboring nodes, typically through weighted sums or similar mechanisms, where the weights are learnable parameters associated with edges [30]. The graph convolutional layer aims to generalize traditional convolutions used on regular grids to irregular graph structures.

In neural network architectures using message passing or graph convolutions, it is common to stack multiple such layers consecutively. The motivation behind this is that for each iteration of the layer, the nodes get information from nodes further away effectively increasing the receptive field of each node, allowing nodes to capture a broader context and more complex dependencies. If too many layers are stacked within the network architecture, every node will have information from every other node in the graph, leading to an oversmooth graph [31], giving decreased performance as the nodes become indistinguishable from one another.

2.5.2 Graph attention layer

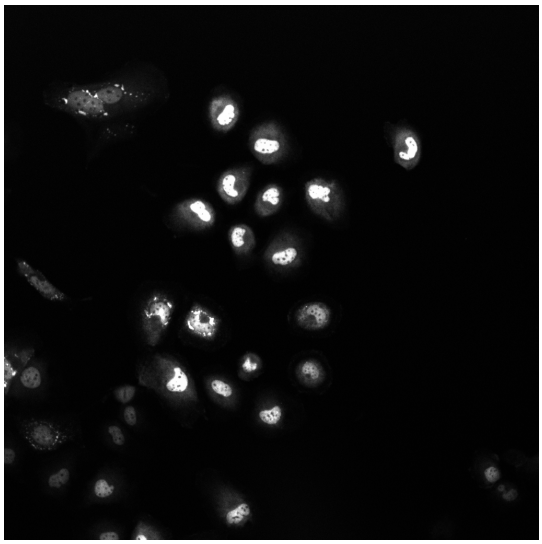
A graph attention layer is a specialized component in Graph Neural Networks designed to enhance information propagation within graph-structured data by assigning varying levels of attention to neighboring nodes during message passing [32]. It builds upon the concept of attention mechanisms, as described in Section 2.4, applied to graphs.

The key idea of a graph attention layer is to allow nodes in the graph to focus on different parts of their neighborhood when aggregating information. This is achieved through attention scores, which indicate the relevance or importance of each neighbor to the central node. By assigning attention scores, the graph attention layer enables nodes to prioritize more informative neighbors, thus capturing meaningful relationships and dependencies within the graph while simultaneously avoiding the problem of oversmoothing in deep network architectures. Like regular attention layers, graph attention layers can utilize multiple heads in order to learn multiple distinct sets of attention parameters, allowing the model to capture different types of relationships and patterns.

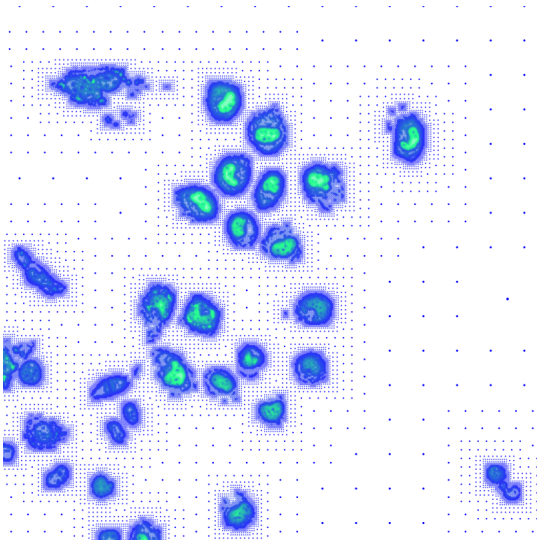
2.6 Adaptive particle representation

Adaptive particle representation (APR) is a method for representing images in a compressed and efficient way inspired by the adaptation and local gain control within the human visual system [33]. It was created as an alternative to a regular pixel-grid-based way of representing images that uses less memory and is more computationally efficient. Unlike a fixed pixel grid, APR employs a dynamic approach. It uses particles to represent image details by densely populating areas with more information and placing fewer particles in less variable regions. This approach ensures a representation that is both data-efficient and faithful to the original content.

A crucial aspect of APR is its ability to handle varying intensity levels in images. This is done through the ”Local Intensity Scale”, which allows APR to adapt to different brightness gradients and ensure a balanced representation. Particle positioning in APR is methodical, based on the image's content. The intensity gradient, measuring intensity change across the image, guides this process. By analyzing the gradient, the APR algorithm optimizes particle placement, capturing the core image features effectively. An example of an image and its APR-representation can be found in Figure 2.2. The original image is human hepatocarcinoma-derived cells expressing the fusion protein YFP-TIA-1 from the cell tracking challenge [6].



(a) Image of human hepatocarcinoma-derived cells expressing the fusion protein YFP-TIA-1.



(b) The APR-transformed image of 2.2a.

Figure 2.2: Example of a microscopy image of cells being turned into its APR-representation.

3
Methods

In this chapter, two neural network models for single-model end-to-end object tracking will be presented. The first network is a fully supervised neural network model based on a graph neural network (Section 3.1), and the second is an unsupervised neural network model based on a variational autoencoder (Section 3.2).

3.1 GNN model

The concept of employing a graph neural network (GNN) approach was inspired by the recent achievements in utilizing GNNs for linking trajectories, as highlighted in the work by Pineda et al. [34]. Furthermore, GNNs have demonstrated their effectiveness in object detection within point cloud-based imaging [35][36].

For clarity and comprehensibility, the network architecture has been divided into three sections. These are Section 3.1.1 Preprocessing and Encoding, Section 3.1.2 Graph Neural Network, and Section 3.1.3 Output. A full model of the architecture is available in Figure 3.1. The loss function used during training is then further explained in Section 3.1.4, and the hyperparameters used during testing are listed in Section 3.1.5.

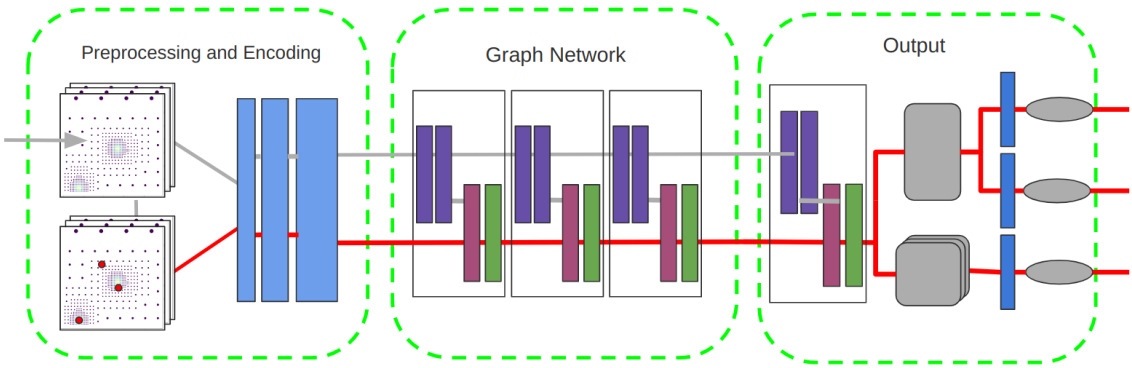
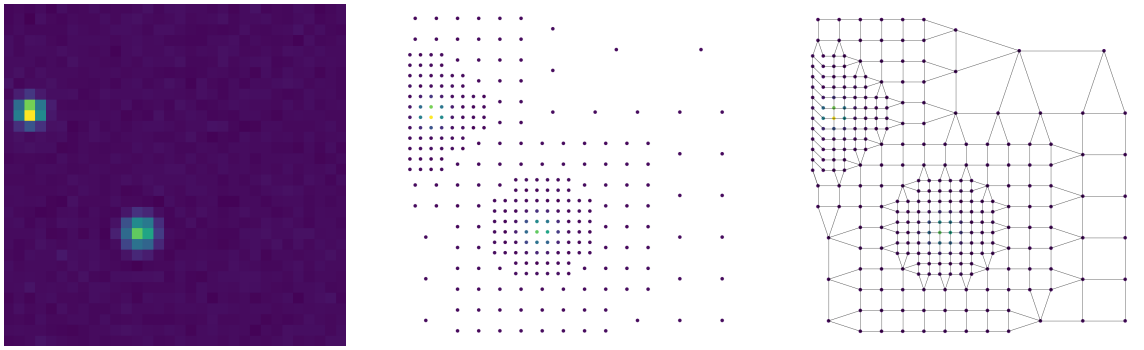


Figure 3.1: Overview over the GNN-based End-to-end object tracking model divided into three parts, more detailed figures of each of the three parts can be found in Figure 3.3, 3.4 and 3.6.

3.1.1 Preprocessing and Encoding

The preprocessing and encoding part of the network serves as the feature extractor by transforming and encoding an input sequence of images into a node feature space, similar to the manual selection step of properties in the current two-step method for object tracking in microscopy images. This feature space prepares the data and extracts only the most important features that are later inputted into the GNN.

Each image in the sequence undergoes an APR-transform, that transforms the image representation from a pixel grid format to a sparse particle representation with variations in the resolution (further explained in Section 2.6). Edges are then established between all particles and their face-neighbors, this being the nodes directly adjacent to a particular node. The new connections effectively convert the APR-image into a graph structure, capturing the relationships between the nodes. Figure 3.2 provides a visualization of this transformation.



(a) Simulated bright-field image of two Mie spheres.

(b) The APR-transformed image of 3.2a.

(c) Graph represented APR-image where edges have been added to 3.2b.

Figure 3.2: Example of an image transformation to APR- and graph-representation.

The nodes within an APR graph, in this paper referred to as image nodes, are characterized by five features: the x-coordinate, y-coordinate, z-coordinate, brightness, and resolution level. However, considering that the dimension of the input images is two-dimensional, the z-coordinate is always zero and is therefore omitted. As a result, the nodes in the graph retain only four features: the x-coordinate, y-coordinate, brightness, and resolution level. An additional feature in the edge is then incorporated when the edges are created, which encodes the distance between the two connected nodes. This distance feature provides information about the spatial relationship between the nodes.

Special nodes called *supernodes* are then placed into the transformed graph. The number of supernodes is estimated based on the maximum expected number of objects present in the images. The placement of the supernodes is random, and the connected edges link them to all nodes in the image graph. These supernodes serve

as the predicted output nodes, representing the objects of interest or the desired targets of the model. By connecting the supernodes to all other nodes in the graph, the model can learn the relationships between these supernodes and the rest of the graph, capture their association, and learn to ”move” these supernodes toward the target objects in the image.

The supernodes, together with the nodes from the APR graph, are then used as the input to the encoding neural network. It begins with three dense layers, each followed by a GELU activation function [37]. These dense layers are responsible for encoding the node features, transforming them from a feature space with four features into a final higher-dimensional feature space with 32 features, doubling the number of features at each dense layer. Encoding the features into a higher-dimensional space allows the network to capture and extract relevant information from the nodes, enabling it to learn more complex patterns and representations. Important to note is that both the nodes and the supernodes undergo the same encoding process, making sure that they are encoded in a consistent manner. This ensures that the network treats the nodes and supernodes equally.

Similarly, the edge features are encoded using another set of dense layers. This encoding process transforms the edge features, enabling the network to learn meaningful representations and extract relevant information from the relationships between nodes. A detailed visual representation of the preprocessing and encoding part of the model can be seen in Figure 3.3.

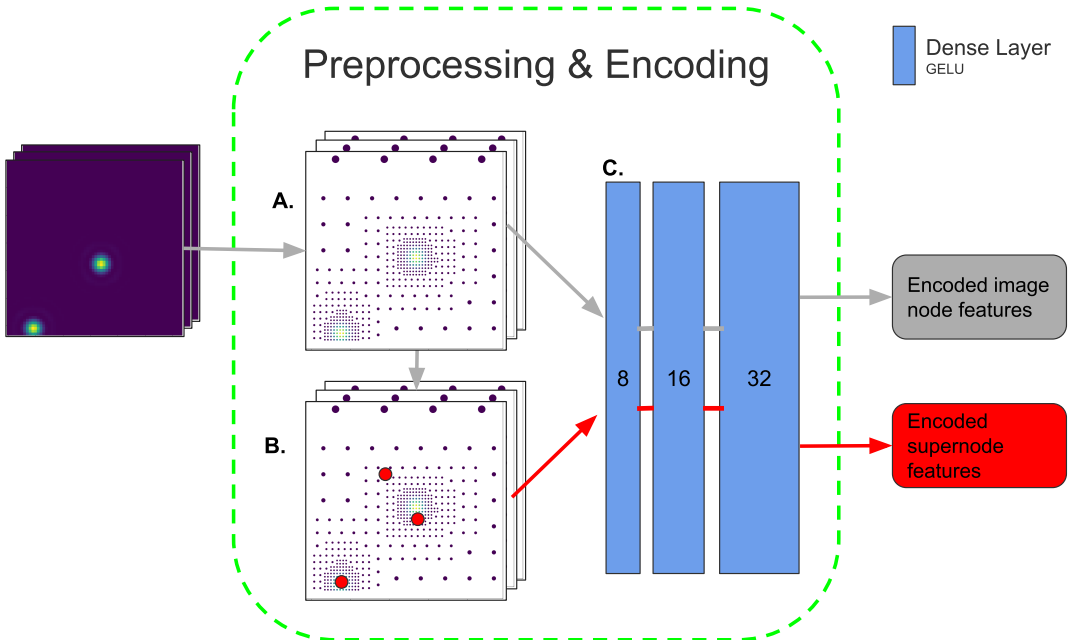


Figure 3.3: Overview over the preprocessing and encoding part of the GNN-based End-to-end object tracking model containing: (**A**) A sequence of images after APR-transformation, (**B**) supernodes being placed into the image-graph and (**C**) A series of dense layers encoding the features of the image nodes and the supernodes.

3.1.2 Graph neural network

After the encoding step comes the Graph neural network part of the model. It is made up of blocks consisting of three different kinds of layers: two message-passing layers, one multi-head graph attention layer for propagating information in space, and one multi-head attention layer for creating edges and propagating information through time. This block is repeated three times in order to make the network deeper and allow the network to learn more complex relations. This part of the model is visualized in Figure 3.4.

The first part of the GNN block consists of two message-passing layers. These layers propagate information in space between the image nodes within a time frame. The number of message-passing layers directly determines the receptive field of each node, determining the extent of information it can gather from its surroundings. If a larger receptive field is desired to capture more comprehensive spatial dependencies, the number of message-passing layers can be increased. This can be the case if, for example, the objects within the images are large. By iteratively applying these message-passing layers, the network enables information to flow and propagate through space, facilitating the incorporation of local and contextual information from neighboring image nodes into each node's representation. This mechanism helps the model capture spatial relationships and context within each time frame of the input data.

The second part of the GNN block consisted of a multi-head graph attention layer specifically designed to update the supernodes with spatial information from the image nodes. This layer divides the features of each supernode and the connected image nodes into multiple heads. The attention within the head between each supernode and its edge-connected node is then computed using cosine similarity. This calculation helps to determine the relevance or importance of the image node feature in relation to the supernode feature. The cosine similarity is shown in Equation 3.1.

$$\alpha_{i,j} = softmax_j\left(\beta_i \frac{\mathbf{x_i} \cdot X_{Sj}}{\|\mathbf{x_i}\|\|X_{Sj}\|}\right) \tag{3.1}$$

where $\alpha_{i,j}$ is the attention between image node V_i and supernode V_{Sj}, $\mathbf{x_i}$ is the feature vector of the ith image node V_i, X_{Sj} is the feature vector of the jth supernode V_{Sj}, and $\beta_i = \frac{1}{\mathbf{d_{i,V_S}}}$, where $\mathbf{d_{i,V_S}}$ is the shortest distance between an image node v_i to any of its connected supernodes V_S. The term β_i acts as regularization and helps strengthen the relationship between the supernode and the closest image node, which results in *supernode pairs*. A similar method has previously been used for clustering nodes [38]. Using the resulting attention matrix as weights, the supernode features are updated with information from the image node features using Equation. 3.2.

$$X_{Sj} = \frac{1}{n_j}\sum_{i=0}^{n_j} \alpha_{i,j} * \mathbf{x_i} \tag{3.2}$$

Where X_{Sj} is the feature vector of the supernode, $\mathbf{x_i}$ is the feature vector of the im-

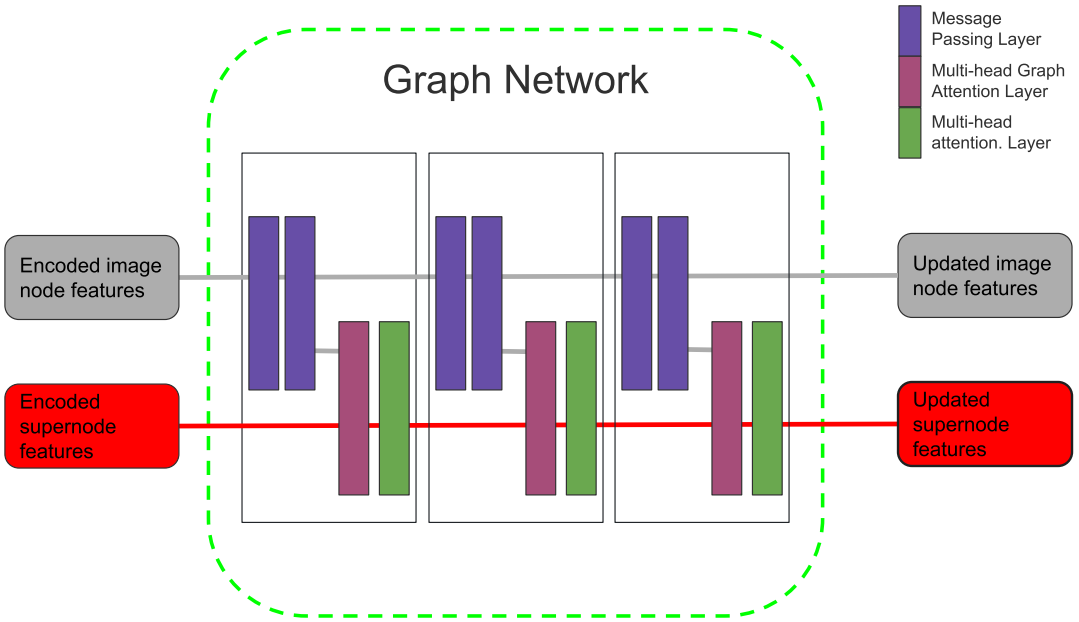


Figure 3.4: Overview over the graph neural network part of the GNN-based End-to-end object tracking model comprised of three blocks each containing two message passing layers for updating the image node features followed by a multi-head graph attention layer for propagation information from the image nodes to the supernodes and lastly a multi-head attention layer to propagate information between the supernodes in time.

age node, $\alpha_{i,j}$ is the attention calculated in Equation 3.1, n_j is the number of image nodes connected with edges to each supernode. The updated supernode features of each head are then concatenated to pass on the supernodes in their orignal shape. In order to add learnable weights to this layer, the image nodes and supernodes are passed through a dense layer before being separated into multiple heads and having their attention calculated, as well as one dense layer after the heads are concatenated.

The third and final part of the GNN-block consists of a masked multi-head attention layer designed to propagate information between supernodes at different time frames. It works very similarly to the previous multi-head graph attention layer in space described in Section 3.1.2. First the features of the supernodes in all time frames are divided up into multiple heads. Then a time mask is created by setting a large negative value to the supernode pairs where the difference in time between the frames of the two supernodes is zero or greater than a constant T, and 0 otherwise (Equation 3.3).

$$m_{i,j} = \begin{cases} -\inf & \text{if } |t_i - t_j| > T, \text{ or } |t_i - t_j| = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

Where t_i and t_j is the time step of supernode V_{Si} and V_{Sj}, T is a hyperparameter detailing how many time steps into the future and past each supernode will obtain

information from. The attention between the supernodes of different time frames is then calculated using the cosine similarity, the same way as in Equation 3.1 but with some minor changes, seen in Equation 3.4.

$$\alpha_{i,j} = softmax_j\left(\frac{\beta_i}{\tau}\left(\frac{X_{Si} \cdot X_{Sj}}{\|X_{Si}\|\|X_{Sj}\|} + m_{i,j}\right)\right) \tag{3.4}$$

Where X_S, β and α have the same definition as in Equation 3.1, m is the time mask, and τ is a temperature hyperparameter commonly used in neural networks to change the distribution of a softmax activation function. When $\tau \to \infty$, the distribution becomes more uniform, and when $\tau \to 0$, the distribution collapses to a point mass [39]. Note that the mask defined in Equation 3.3 restricts each supernode to a limited number of frames in the future and from the past, since $softmax(x) \to 0$ when $x \to -\infty$. The supernodes X_S are then updated using matrix multiplication between the attention and the old supernodes (Equation 3.5).

$$\mathbf{X_S} \leftarrow \frac{\boldsymbol{\alpha} \times \mathbf{X_S}}{2} \tag{3.5}$$

By updating the supernodes using attention between them through time, the supernodes should be able to recognize which supernode in the future and past frames corresponds to the same object as themselves, which would help with predicting the correct positions within their own frame. The features of each head are then again concatenated into the supernodes' original shape. Just like in the multi-head graph attention layer, learnable weights are added by passing the supernodes through a dense layer before and after all the head-wise operations.

3.1.3 Output

The final part of the model, as can be seen in Figure 3.6, uses a similar block as the GNN part of the model (Section 3.1.2). This block consists of two message-passing layers, followed by a multi-head graph attention layer and a multi-head attention layer. However, there are notable differences in the last multi-head attention layer.

In this final multi-head attention layer, the attention matrix is calculated without applying the softmax activation function, allowing the attention scores to retain their original values. Moreover, the updating of the supernodes is also skipped in this final multi-head attention layer. Instead, the un-updated supernodes and the attention matrix are further processed into the output.

The features of the un-updated supernodes are then duplicated into three parts, where one copy of the supernode features is passed through a series of three dense layers, which gradually reduces the number of features. The purpose of these dense layers is to transform the supernode features into a final representation consisting of only two features. These two features make one part of the output and represent the x-coordinate and y-coordinate of each node, resulting in the model being able

to predict the spatial position of each object within each frame.

The second copy of the supernode features is passed through a single dense layer with a sigmoid activation function. The resulting output, ranging from 0 to 1, represents the certainty of each supernode being a valid detection. If the certainty value falls below a specified threshold, the supernode can be removed from further consideration. This allows for filtering out less confident detections and improving the overall reliability of the model's predictions. This detection confidence is needed as the number of supernodes placed in the graph at the beginning is only an approximation of the actual number of objects in the sequence. All supernodes that do not correspond to an actual object, therefore, need to be removed.

The final part of the output involves the supernode attention matrix. All the attention heads from this matrix are concatenated into a single representation and passed through a dense layer with a sigmoid activation function. The resulting output represents the certainty of connections between supernodes in different time frames. By applying a threshold, this certainty score can be used to create an adjacency matrix that reveals the trajectory of each object.

The combination of detected objects and the temporal adjacency matrix can be represented using a graph structure. This representation assigns each detection as a node within the graph, and the edges between these nodes represent the trajectory of the objects over time. An illustrative example of this graph-based representation is depicted in Figure 3.5.

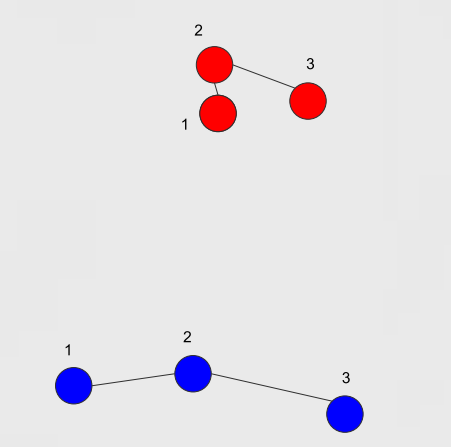


Figure 3.5: Example of two objects, shown in different colors, and their trajectories through 3 timesteps modeled as a graph.

3.1.4 Loss function

The loss function is inspired by the tracking accuracy measurement used in the cell tracking challenge [6][40] in combination with a loss function used in the GraphVAE

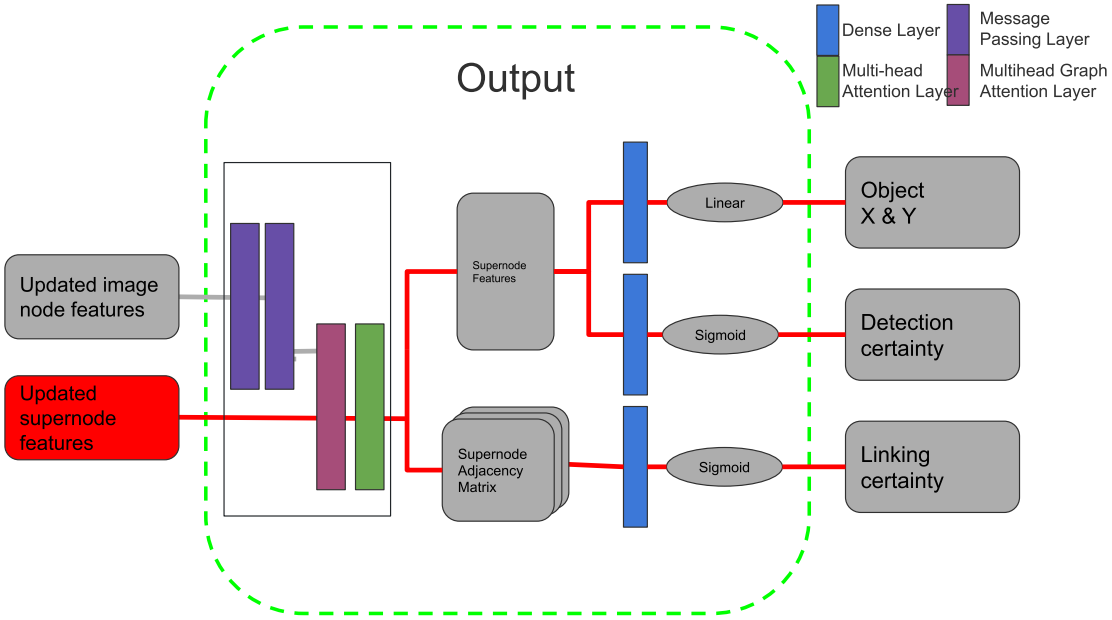


Figure 3.6: Overview over the output part of the GNN-based End-to-end object tracking model comprised of a final block of message passing and multi-head attention layers. After this block the model is split into three heads with each head being passed through a separate dense layer and an activation function. This results in three predicted outputs: positions, detection certainty, and clinging certainty.

paper [41]. It is comprised of three terms, where each serves one of the three outputs: the localization of the object, the certainty of a supernode being an object, and the certainty of edges between frames. One term is penalizing the model if the predicted nodes are not correctly placed, the second term is penalizing the model for not removing excess predicted nodes, and the last term is penalizing the model for incorrectly classifying the temporal edges between the predicted nodes.

Before calculating the terms of the loss function, the output prediction graph and the target graph need to be matched to each other. This matching step is necessary because graph structures lack inherent grid-based indexing, making it challenging to determine what predicted node corresponds to the same node in the target graph. While various graph-matching techniques exist, most of them are computationally expensive and, therefore, impractical for this context. Instead, this model adopts a simpler approach. It matches each node in the target graph to the closest node in the predicted output graph based on the Euclidean distance. It's important to note that each target node can only be matched to one predicted output node, while multiple output nodes can be matched to the same target node. Permitting predicted nodes to be connected to multiple target nodes in this way helps prevent the model from being heavily penalized for missing events where a matched object splits into two, such as a cell undergoing mitosis.

From the matched nodes, we get our first term of the loss function, the placement loss, $loss_p$. This is calculated by taking the mean square distance between all

matched predicted (Y_m) and target nodes (Y_t) as in Equaiton 3.6, which penalizes the model more the larger the distance between Y_m and Y_t.

$$loss_p = \frac{1}{n}\sum_{i=1}^{n}(Y_{t,i} - Y_{m,i})^2 \tag{3.6}$$

The second term of the loss function is the detection loss ($loss_d$). It is calculated by taking the binary cross entropy between the predicted detection certainty (C_p) and a ground truth vector (C_t). The ground truth contains ones for all the predicted nodes that are matched to a target node and zeroes for all unmatched predicted nodes. The model is therefore penalized for not removing nodes that do not correspond to the target objects.

$$loss_d = -\frac{1}{n}\sum_{i=1}^{n}(C_{pi} \cdot logC_{ti} + (1 - C_{pi}) \cdot log(1 - C_{ti})) \tag{3.7}$$

To compute the third loss, a binary mapping matrix $M \in \{0, 1\}$ is first created from the distance matrix where the rows and columns are the indexes of target nodes and predicted nodes, where each of the matched target node and predicted node pairs is given a value of one. This matrix is then used to map the target adjacency matrix A_t to the domain of the predicted graph, resulting in a mapped adjacency matrix A_m, enabling comparisons to be made to the predicted adjacency matrix A_p. This mapping is shown in Equation 3.8.

$$A_m = MA_tM^T \tag{3.8}$$

The linking loss is calculated by taking the binary cross entropy between A_m and A_p shown in Equation 3.9.

$$loss_l = -\frac{1}{n}\sum_{i=1}^{n}(A_{pi} \cdot logA_{mi} + (1 - A_{pi}) \cdot log(1 - A_{mi})) \tag{3.9}$$

From all of these, the total loss of the model can be summed together with weightings λ to balance the three terms.

$$loss = \lambda_p loss_p + \lambda_d loss_d + \lambda_l loss_l \tag{3.10}$$

3.1.5 Hyperparameters

Tale 3.1 lists all the model hyperparameters together with their standard value used for all tests in this paper.

Hyperparameter	Standard value
placement weight (λ_p)	1
detection weight (λ_d)	1
linking weight (λ_l)	1
attention heads	4
softmax temperature	4

Table 3.1: List of all the used hyperparameters for the GNN-model together with their standard value.

3.2 MVAE model

Due to unsatisfactory results obtained from the GNN-based model (Section 4.1), particularly in terms of node placement and the removal of excess nodes, a second model has been developed. This new model retains the concept of creating temporal trajectory links using multi-head attention but is significantly different in all other aspects. This new model is an unsupervised model based on a multi-entity variational autoencoder (MVAE) [42]. Since the detection component was not developed in this work, only a brief description will be presented in this paper. The focus will instead be directed toward the temporal trajectory linking and its influence on the detection of objects. First, the building blocks of MVAE will be briefly explained in Section 3.2.1, followed by the multihead attention in Section 3.2.2. Section 2.3.1 then explains the post-processing step to detect the right number of objects and lastly, Section 2.3.1 lists the hyperparameters used in testing the MVAE. The entire network architecture is visualized in Figure 3.7.

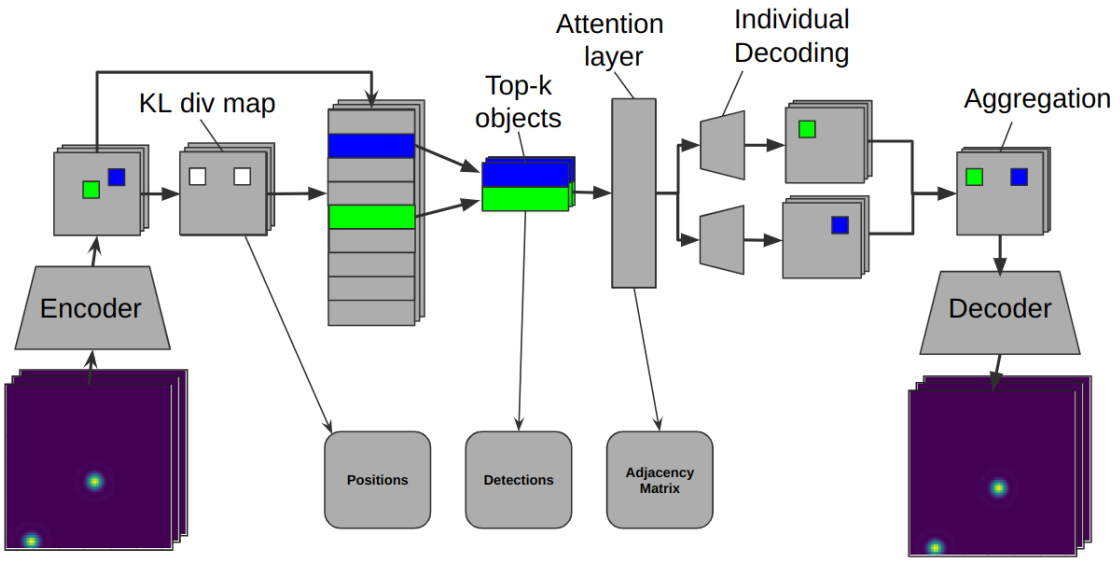


Figure 3.7: Visualization of the MVAE based tracking model described is Section 3.2.

3.2.1 Multi-entitiy variational autoencoder

The MVAE works similarly to a regular variation autoencoder as described in Section 2.3.1. The difference lies in the input where images are encoded into a grid of posterior parameters, where each spatial location represents a potential object within the image. The KL-divergence between the distribution of each potential object's latent variables and a standard normal distribution is calculated. From this, only the top-N potential objects with the highest KL-divergence are retained, which should correspond to the actual objects in the image. In this paper, N is an estimate of the maximum number of actual objects within an image sequence and will be referenced as the network parameter "number of entities".

The underlying idea behind this approach is that in order to accurately reconstruct the input images, the network needs to encode substantial amounts of information in locations corresponding to the objects, such as their shapes and colors. As a result, these regions exhibit higher KL-divergence values, while less information is required to encode background information, leading to smaller KL-divergence values.

Once the top-N objects are identified, their latent distributions are sampled by the encoder part of the network to obtain individual latent representations for each object. These latent representations are then decoded individually for each object using the first of two decoders used by the network. The first decoder helps with keeping the individuality of the objects in the reconstruction of the image.

The output is then combined after each object has been individually decoded. Since the positions of the objects are known based on their spatial locations in the feature map, it is possible to overlay the latent spaces of all objects onto a grid of the same size as the input image. By doing this, we can aggregate the latent features of all objects at each grid location effectively consolidating the information from multiple objects into a single combined latent feature map for all objects in the entire input image. This is achieved by taking the maximum value of the latent features for each grid location. This aggregated feature map is subsequently used as input to the second decoder, which generates the final reconstructed image.

The spatial location used to combine the latent features of the objects into one image serves a dual purpose. Not only does it enable the aggregation of latent features, but it also provides the actual positions of each object in the input image. So by associating each grid location in the feature map with a specific object's position, we can determine the precise location of every detection in the image.

3.2.2 Multihead attention through time

To establish temporal connections between the objects, a multi-head attention layer is introduced after sampling the latent variables from the regions with the highest KL-divergence (explained in the first part of Section 3.2.1). This multi-head attention works almost the same as the one used in the GNN model, described in Section 3.1.2.

First, the latent features are divided into multiple heads after which a time mask is generated following Equation 3.3. The attention between the latent features is then calculated, using the masked cosine similarity as in Equation 3.4 but omitting the regularization term β. Note again that the time mask defined in Equation 3.3 restricts each supernode to a limited number of frames in the future and from the past, since $softmax(x) \to 0$ when $x \to -\infty$.

The latent features undergo an update process by performing matrix multiplication between the attention weights and the un-updated latent features. This multiplication operation allows the attention mechanism to prioritize and combine information from the same object across different time steps, improving the model's ability to capture temporal dependencies. The updated latent features are then passed through a dense layer that introduces learnable parameters to enhance the attention mechanism. This design encourages each object to receive more relevant information from its own temporal counterparts, facilitating improved detection performance by leveraging temporal consistency.

The attention matrix obtained from the previous step provides valuable information about the temporal connections between objects. By examining the attention values, we can identify the strongest connections between an object and its counterparts in different time steps. Specifically, for each object, we select the object with the highest attention value in the subsequent time frame as its temporal connection. It is important to note that this approach limits each object to be connected to only one object in the next or previous frame, disregarding the possibility of multiple connections. This simplification helps establish clear and unambiguous temporal connections between objects for further analysis and processing.

3.2.3 Loss function

To train the model, a modified version of the β-VAE [43] is used. The loss function consists of two main components: the reconstruction loss which measures how well the autoencoder is able to reconstruct the input data from the latent representation. The reconstruction loss is calculated using mean squared error (MSE). The second component is the KL-divergence loss, which measures the difference between the learned latent distribution and the standard normal distribution. The KL divergence ensures that the latent variables follow a distribution that is close to the prior (the standard normal distribution), enabling better generalization. To balance the trade-off between the losses, a parameter β is added before the KL-divergence term, which encourages disentanglement of the learned latent representations. For the KL-divergence loss, only the KL-divergence of the top-N objects is included in order to stabilize the training of the model.

3.2.4 Combining the detected objects

Because the number of entities (N) is a parameter manually set when creating the network, the number will not correspond to the actual number of objects within the images. It is required that the number of entities is larger than or equal to the actual number of objects, for the network to be able to detect all the objects. It is, therefore, a high possibility that the network makes more detections than there are objects in each image. However, this issue can be mitigated by leveraging the fact that multiple top-N KL-divergent entities often correspond to the same object. As a result, these entities will be detected at similar positions and could be combined into a single object.

To address this, a post-processing step is employed to compare the positions of each detection. By considering a specific "combine radius," we can identify detections that are located within the specified radius of another detection, indicating that they likely correspond to the same underlying object. All identified detections corresponding to the same object are combined in order to remove all superfluous detections.

3.2.5 Hyperparameters

Tale 3.2 lists all the model hyperparameters and the used standard values in all tests during the project.

Hyperparameter	Standard value
Latent dimension	12
beta	0
combine radius	3
number of entities (N)	10
attention heads	1
softmax temperature	1

Table 3.2: List of all the hyperparameters used in the MVAE-model together with their standard value.

3.3 Evaluation

The evaluation method used in this project was the precision of edge classification to evaluate the trajectories, mean absolute error (MAE) for evaluating the accuracy of the placement of the detection centroids, and the average number of missed detections (MD) and false detections (FD).

The output of both the GNN and the MVAE, as well as the target, is modeled as a graph. Therefore, to evaluate the centroid placement and tracking accuracy, the

graphs must first be matched to the target graph, which is done in the same way as graph-matching in Section 3.1.4. First, the Euclidean distance between all target nodes and predicted nodes is calculated and only the closest predicted node is then matched.

The Euclidean distance between the matched target and prediction nodes is then used to calculate the mean absolute error between the matched target and predicted nodes, shown in Equation 3.11.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|Y_{t,i} - Y_{m,i}| \tag{3.11}$$

where i is the target node index, $Y_{t,i}$ the target node, $Y_{m,i}$ the matched predicted node and n the number of target nodes. MAE evaluates the quality of object localization.

In addition to matching each target node to one predicted node, the reverse is also done, matching each predicted node to one target node. Comparing these two ways of matching gives the number of false detections and missed detections. If the target and predicted node pair are the same both ways the detection is valid. But if they are not, either a false detection or a missed detection has occurred. A visualization of this is shown in Figure 3.8. The number of missed detections is then divided by the total number of target detections to get the missed detection rate and the false detections by the total number of predicted detections to get the false detection rate. Both of these are used as evaluation metrics for the ability to detect objects.

From the original matching of targets, a binary mapping matrix $M \in \{0, 1\}$ is created to map the target adjacency matrix A_t to the domain of the predicted graph, precisely as in the GNN loss function in Section 3.1.4. The mapping is then used to map the target adjacency matrix A_t to the domain of the predicted graph, just like in Equation 3.8, enabling comparisons to be made to the predicted adjacency matrix A_p.

By comparing A_m to A_p the true positive (TP), false positive (FP), true negative (TN), and false negative (FN) link predictions can be extracted. these are then used in the calculation of precision (Equation 3.12), recall (Equation 3.13), and F1-score (Equation 3.14) between a pair of time frames. How TP, FP, TN, and FN relate to the predicted and target trajectory linking is visualized in Figure 3.9.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.12}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.13}$$

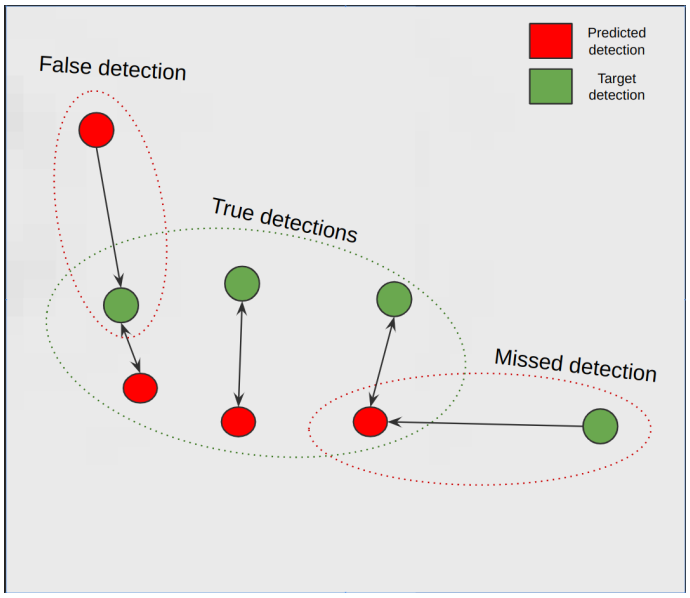


Figure 3.8: A visualization of the difference between true detections, false detections, and missed detections between the targets (green) and the predictions (red). The arrows between the target and predicted detections indicate a directional matching.

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.14}$$

The Precision, Recall, and F1-score are all used to evaluate the quality of the trajectory-linking classification.

3.4 Dataset

The dataset used in this project is simulated data using DeepTrack [44]. The simulated particles are Mie Spheres generated with a uniformly random diffusion constant between 10^{-13} and 10^{-12} for each image sequence, a refractive index of 1.55, a radius of 10^{-6}, and random x and y coordinate within the bounds of the image for each particle in a sequence. The code implementation for generating Mie Spheres can be found in Listing 3.1.

Listing 3.1: Simulating Mie spheres

```
mie_particle = dt.MieSphere(
    diffusion_constant=(1+np.random.rand()*9)*1e-13,
    refractive_index=1.55,
    radius=1e-6,
    z=lambda: np.random.uniform(-2, 2)*dt.units.um,
    position=lambda: np.random.uniform(5, image_size-5, 2)
)
```

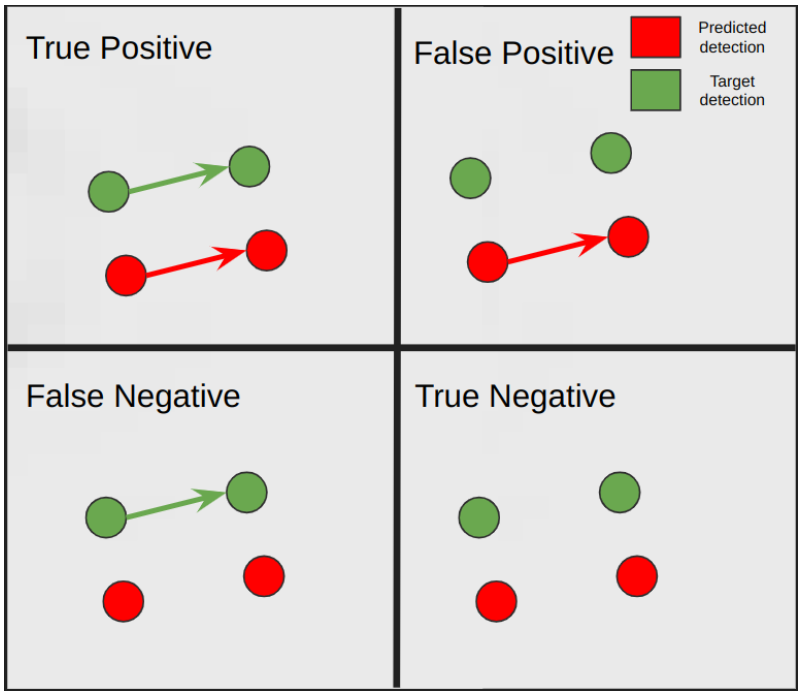


Figure 3.9: A visualization of what a true positive, false positive, true negative, and false negative represents in the trajectory linking with the target detections and links in green and the predicted detections and links in red. The circles represent objects and the arrows between show how the objects link in time.

These simulated particles are imaged using a simulated bright-field microscopy setup. The code for this can be found in Listing 3.2.

Listing 3.2: Simulating bright-field microscope

```
optics = dt.Brightfield(
    NA=0.4,
    magnification=5,
    resolution=3e-6,
    wavelength=0.6e-6,
    output_region=(0, 0, image_size, image_size),
)
```

To simulate the motion of particles between consecutive time steps, Brownian motion is used based on the diffusion constant associated with each image sequence. The particle positions are updated using this stochastic process, using the image size as a boundary constraint to prevent particles from moving outside of it. The code snippet in Listing 3.3 provides the function responsible for updating the positions of the particles in the simulation.

Listing 3.3: Function for updating a particles position

```
def update_position(position, diffusion_constant, dt=1/10):
```

```
2
3       new_position = np.clip(position + np.random.randn(2)
4               * np.sqrt(diffusion_constant * dt)
5               * 1e7, 0, image_size)
6
7       return new_position
```

Using this, an image pipeline is created where a sequence of images is created with a uniformly random amount of particles between 2 and 6 generated for each sequence. The particles are imaged using simulated Brightfield microscopy and then Gaussian noise is applied to each image. The code for this can be found in Listing 3.4

Listing 3.4: Pipeline for simulating image sequences

```
1  particle = dt.Sequential(mie_particle,
2           position=update_position)
3  particles = particle^(lambda: np.random.randint(2, 6))
4
5  image_pipeline = optics(particles) >> dt.Gaussian(sigma=0.1)
6  image_pipeline = dt.Sequence(image_pipeline,
7                sequence_length=sequence_length)
```

Most training and tests included 1000 generated sequences of three timesteps each. These sequences were randomly split into a training and validation set with an 80/20 split. A separate set was then created for the tests containing 100 sequences.

3.5 Hardware

All training and testing have been done on a ”Nvidia A100 Tensor Core” GPU with 40 GB video memory and an ”AMD EPYC 7302 16-Core Processor” CPU.

4
Results

In this chapter, results from both the GNN-based model and the VAE-based model are presented. Most results are based on the metrics described in Section 3.3.

4.1 GNN-model

This section presents the results of the GNN-based model described in Section 3.1. This mainly contains results from testing different hyperparameters and model architectures in order to determine the optimal configuration for the model.

4.1.1 Number of heads

Figure 4.1 presents the results obtained by varying the number of heads for the attention layer in the model. Here, a latent feature space of size 16 was utilized, with the number of heads ranging from 1 to 16. Consequently, the number of features per head ranged from 16 (for a single head) to 1 (for 16 heads). As shown in the figure, the model's performance improves as the number of heads increases up to a point. Specifically, the model performs better with up to four heads, showing enhanced detection capabilities and lower missed detection rates. However, increasing the number of heads beyond four results in a decline in performance. When using more than four heads, the model starts to struggle with object detection, leading to a significantly higher missed detection rate.

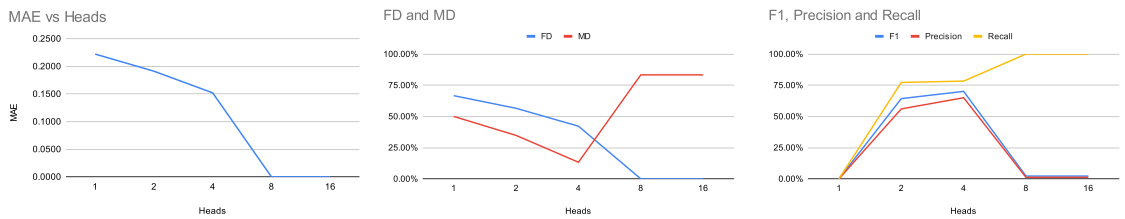


Figure 4.1: Mean average error (MAE), false detection rate (FD), missed detection rate (MD), and F1-score of a model trained with different numbers of heads in the multiheaded attention layers.

Notably, when the number of heads exceeds eight, the model's performance degrades severely to the point where it fails to detect any objects at all. This trend indicates

that while multi-head attention can be beneficial up to a certain number of heads, splitting the latent feature space into too many heads can be very bad for the model's ability to effectively detect objects.

4.1.2 Softmax temperature

Figure 4.2 presents the results obtained by varying the values for the softmax temperature in the model. The performance of the model remains relatively stable across different values of the softmax temperature, suggesting that this parameter does not greatly influence the detection and linking performance.

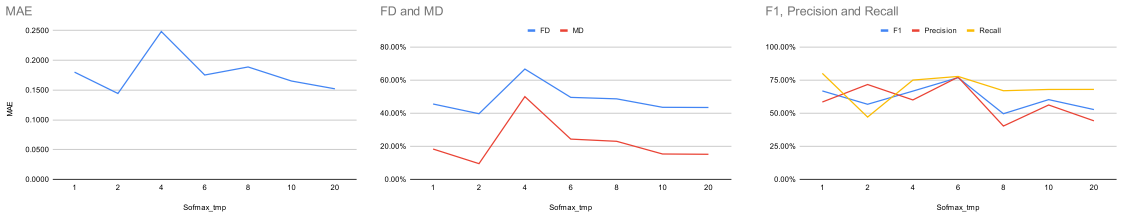


Figure 4.2: Mean average error (MAE), false detection rate (FD), missed detection rate (MD), and F1-score of a GNN model trained with different softmax temperatures.

4.1.3 Graph blocks

In order to see how the depth of the neural network affected the performance of the model, the model was tested with varying amounts of graph blocks, that being the blocks making up the GNN part of the model, see Section 3.1.2. The results are shown in Figure 4.3.

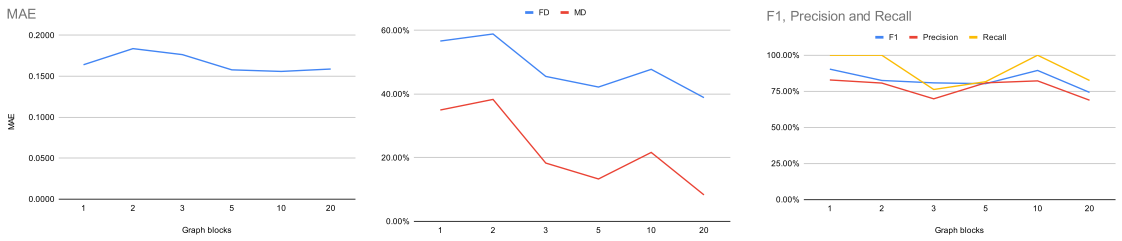


Figure 4.3: F1-score, missed detections, false detections, and mean absolute error between the detection and target for GNN-model with a varying number of graph-blocks in its architecture.

The figures show a correlation between the number of graph blocks in the model and its performance, with a higher number of graph blocks seemingly giving better performance in all metrics. This suggests that increasing the number of graph blocks enhances the model's capacity to learn and represent complex relationships within

the data, thereby improving its overall effectiveness in tasks such as object detection and tracking.

4.1.4 Inital placement of supernodes

To investigate whether the initial placement of the supernodes affects model training, two different initialization methods where tested. In the first method, all supernodes were initialized at the same position, ensuring uniformity in their starting points. In the second method, supernodes were placed randomly, introducing variability in their initial positions.

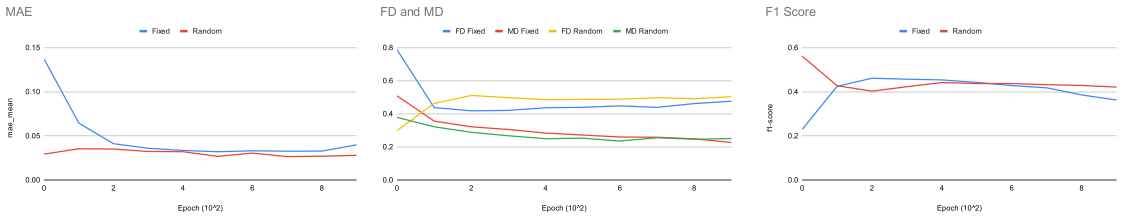


Figure 4.4: Mean average error (MAE), false detection rate (FD), missed detection rate (MD), and F1-score of a GNN model trained using two different initialization methods for the supernodes, one where all where initialized in the same position and one where all where initialized at random positions.

As can be seen in Figure 4.4 the initial position of the supernodes significantly impacts the early stages of model training. A random initialization outperforms the uniform initialization during the initial epochs, demonstrating faster progress and better performance early on. However, after a few hundred epochs, the differences between the two methods diminish, and both approaches converge to similar performance levels. This suggests that while random initialization provides an early advantage, the model's long-term performance is not heavily dependent on the initial placement of the supernodes.

4.1.5 Loss weights

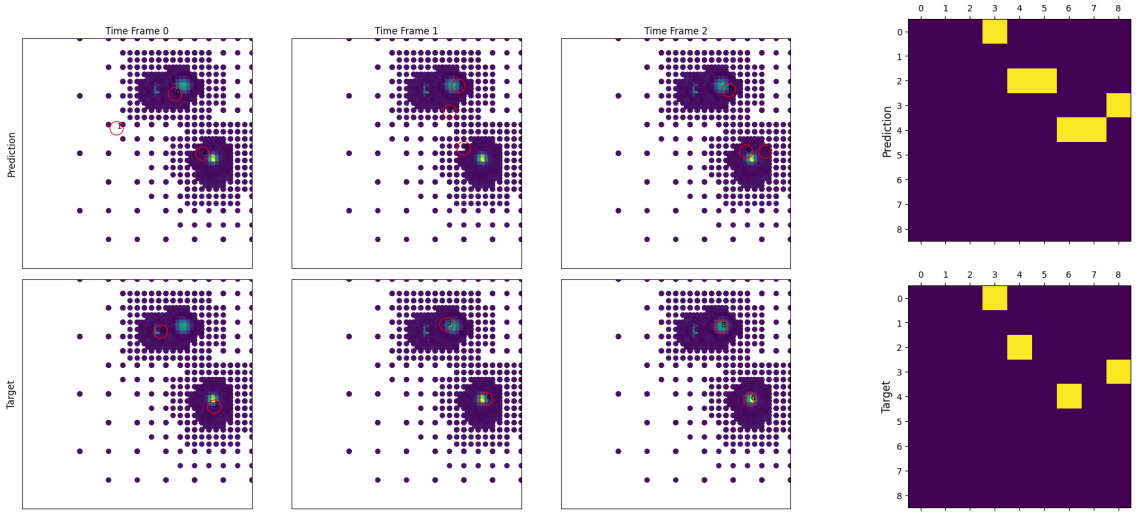
Table 4.1 displays the results obtained by experimenting with different weights λ on the three components of the loss function described in Section 3.1.4. Furthermore, figures showcasing the detections and adjacency matrices are presented further down for selected loss weight combinations that yielded particularly interesting results. These figures provide visual insights into the performance of the model under different weighting schemes.

Figure 4.5 provides a prediction example by a model trained with equal weights for all three loss components ($\lambda_p = \lambda_l = \lambda_d = 1$). In this base case, both the detection and linking performance are not particularly strong. The predicted detections are

Loss weights	MAE	False detections	Missed detections	F1-score
1-1-1	0.1206	40.00 %	10.00 %	66.81 %
1-1-10	0.1607	42.44 %	13.67 %	67.63 %
10-1-1	0.1172	39.56 %	9.33 %	54.91 %
1-10-1	0.1253	41.00 %	11.50 %	62.88 %
1-10-20	0.1786	15.33 %	43.56 %	79.64 %

Table 4.1: Mean average error (MAE), false detection rate, missed detection rate, and F1-score of a model trained with different loss weights. The loss weights column is labeled using three numbers where the first number corresponds to λ_{place}, the second corresponds to λ_{detect}, and the third one to λ_{link} (λ_{place}-λ_{detect}-λ_{link}).

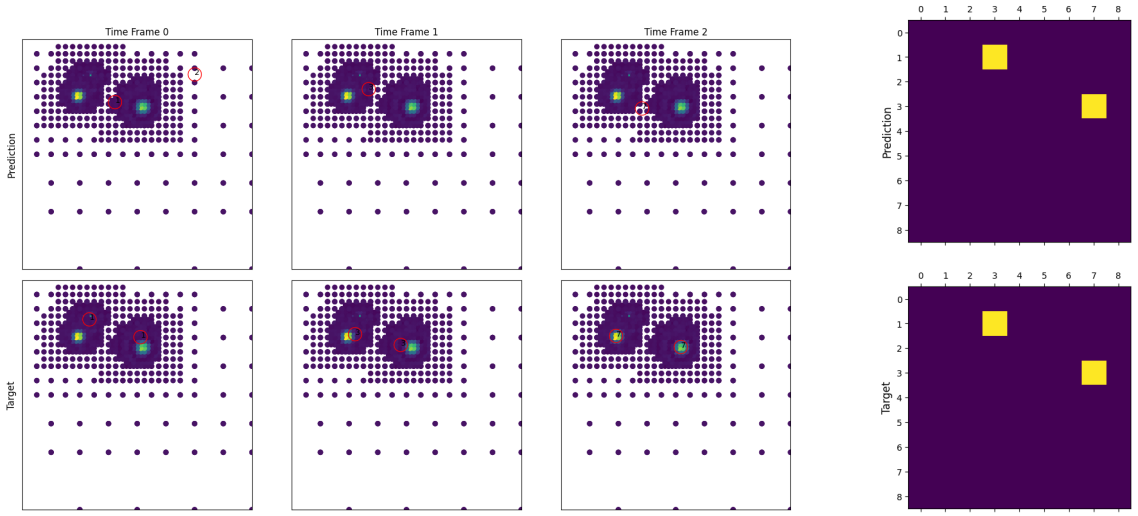
notably distant from the target detections, indicating a significant error in object localization. Additionally, in the adjacency matrix, while the model successfully links each object in the current frames to the corresponding objects in the next frame, it creates double links for objects 2 and 4, falsely suggesting a split on these objects.



(a) **(b)**

Figure 4.5: **(a)** Predicted detections and targets (red circles) in three timesteps, using a model trained with $\lambda_{place} = \lambda_{link} = \lambda_{detect} = 1$. **(b)** Predicted and target adjacency matrices. The yellow squares indicate a connection between the detections with the same labels as the row and column index.

Figure 4.6 illustrates a prediction example by a model trained with $\lambda_{place} = 1$, $\lambda_{detect} = 10$, $\lambda_{link} = 20$. In this case, the model fails by assigning only one detection to represent both objects instead of individual detections for each object. This can be observed in the figure, where the red detection circle is placed in the middle between the two target objects



(a) **(b)**

Figure 4.6: **(a)** Predicted and target detections (red circles) in three timesteps made using a model trained with $\lambda_{place} = 1$, $\lambda_{detect} = 10$, and $\lambda_{link} = 20$. **(b)** Predicted and target adjacency matrix to the detections seen in (a). The yellow squares indicate a connection between the detections with the same labels as the squares' row and column index.

4.2 MVAE-model

The results of the VAE-based model, as outlined in Section 3.2, are presented here. This section begins with an examination of the testing conducted to determine the optimal hyperparameters. Following this, there is a discussion of the findings obtained from varying the input data. Lastly, example results from testing the model with optimal parameters are provided and compared to detecting without utilizing any attention mechanisms.

4.2.1 Hyperparameters

In the process of finding suitable hyperparameters, each parameter was individually tested, and the resulting metrics were recorded. The following hyperparameters were examined: combine radius, number of entities, softmax temperature. Each test is performed on a test dataset never before seen by the model comprised of 100 sequences containing 3 frames each. The model was trained for 100 epochs for each test.

Figure 4.7 reveals that there is a trade-off between false detections and missed detections as the radius parameter is adjusted. A low radius results in many false detections, while a high radius leads to numerous missed detections. The optimal radius appears to be around three, balancing these two types of errors to achieve the best overall performance. Figure 4.8 indicates that the optimal number of entities

for this specific model configuration is ten. At this number, the model performs best across all metrics, suggesting that this configuration allows the model to effectively represent and process the data. Figure 4.9 demonstrates that the model performs better with a higher softmax temperature. This implies that increasing the temperature helps in achieving more confident and accurate predictions, leading to improved model performance.

Figure 4.7: F1-score, missed detections, false detections, and mean absolute error between the detection and target for different radii for which detections are combined.

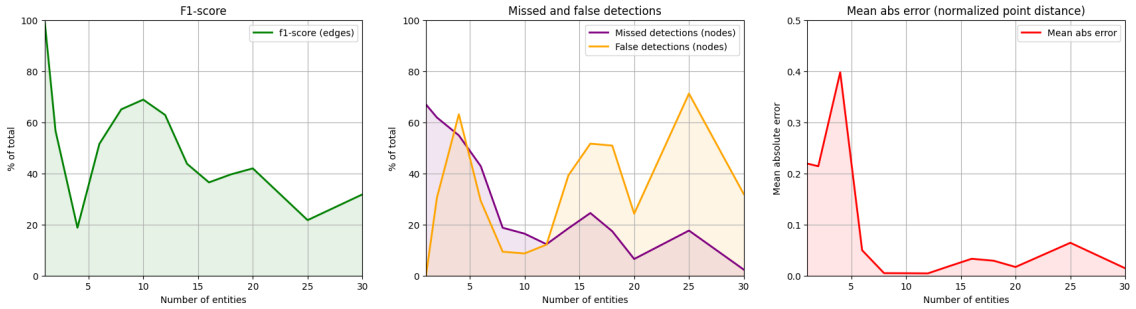


Figure 4.8: F1-score, missed detections, false detections, and mean absolute error between the detection and target for different values of the hyperparameter "number of entities".

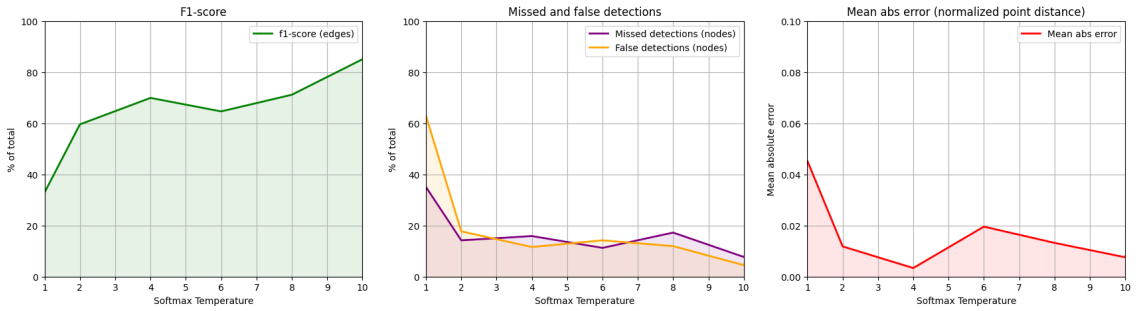


Figure 4.9: F1-score, missed detections, false detections, and mean absolute error between the detection and target for different values of the hyperparameter "softmax temperature".

4.2.2 Object difference

To assess the model's performance with different levels of uniformity between the objects, two experiments were conducted: one using fixed particle sizes with a radius of 1 μm and another using random particle sizes with radii ranging uniformly between 0.5 μm and 2 μm.
The experiments were conducted by repeating each test three times, and in each test, the models were trained for 100 epochs. The results of these experiments are summarized in Table 4.2. The findings clearly indicate that the model performs better in all evaluated aspects when applied to a dataset with greater variations in particle sizes.

	MAE	**F1-score**	**Missed detections**	**False detections**
Fixed (1)	0.019	49.78 %	15.36 %	29.29 %
Fixed (2)	0.017	48.76 %	12.10 %	29.00 %
Fixed (3)	0.012	44.84 %	20.68 %	29.10 %
Fixed (mean)	0.016	47.79 %	16.05 %	29.13 %
Random (1)	0.013	60.44 %	5.19 %	22.22 %
Random (2)	0.011	58.82 %	13.29 %	14.11 %
Random (3)	0.012	62.54 %	12.12 %	28.85 %
Random (mean)	0.012	60.93 %	10.20 %	21.83 %

Table 4.2: Comparison between how well the model predicts on particles with fixed radii versus on particles with uniformly random radii. The comparison is made using Mean absolute error (MAE), F1-score, percentage of missed detections, and percentage of false detections.

4.2.3 Sequence length

To evaluate the model's ability to handle longer video sequences without sacrificing inference performance, tests were conducted on sequences ranging from 3 frames to 20 frames. The results of these tests can be observed in Figure 4.10.

From the experiments, the average time required to train one epoch was also recorded and plotted in Figure 4.11. This figure provides insights into the computational efficiency and scalability of the model when processing longer video sequences. The analysis of the results presented in Figure 4.10 makes it evident that the inference performance of the model does not significantly suffer or improve with increasing sequence length. Additionally, The linear scaling of training time with sequence length, seen in Figure 4.11 implies that the model's training process is efficient and predictable.

4.2.4 Example of tracking using MVAE

Figure 4.12 shows two examples of objects in three subsequent time steps being detected and linked temporally using the MVAE model. The model has been trained

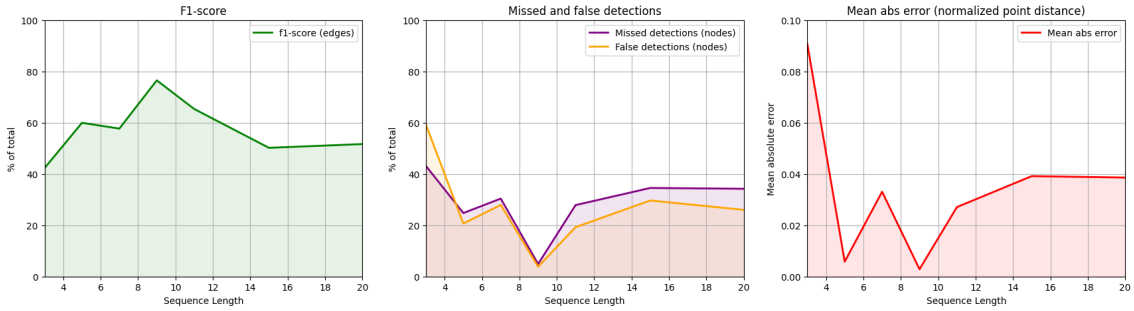


Figure 4.10: F1-score, missed detections, false detections, and mean absolute error between the detection and target for different video sequence lengths.

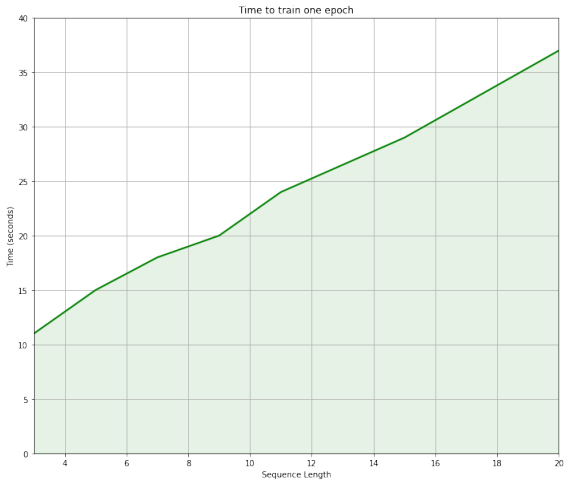


Figure 4.11: The time it takes to train the model for one epoch on different video sequence lengths.

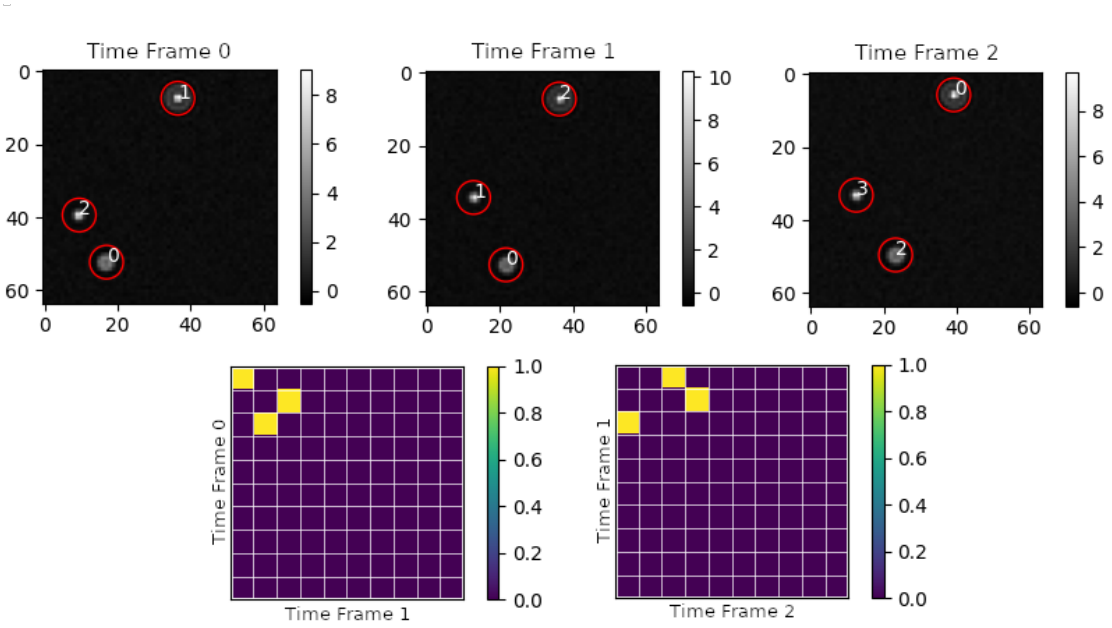
for 500 epochs using the optimal hyperparameters found in the results from the tests in Section 4.2.1, with ”softmax temperature”: 10, ”number of entities”: 10, ”combine radius”: 4, and ”number of heads”: 1. The resulting mean absolute error, false detection rate, missed detection rate, and F1-score can be found in Table 4.3.

MAE	F1-score	Missed detections	false detections
0.00217	81.2 %	2.69 %	7.84 %

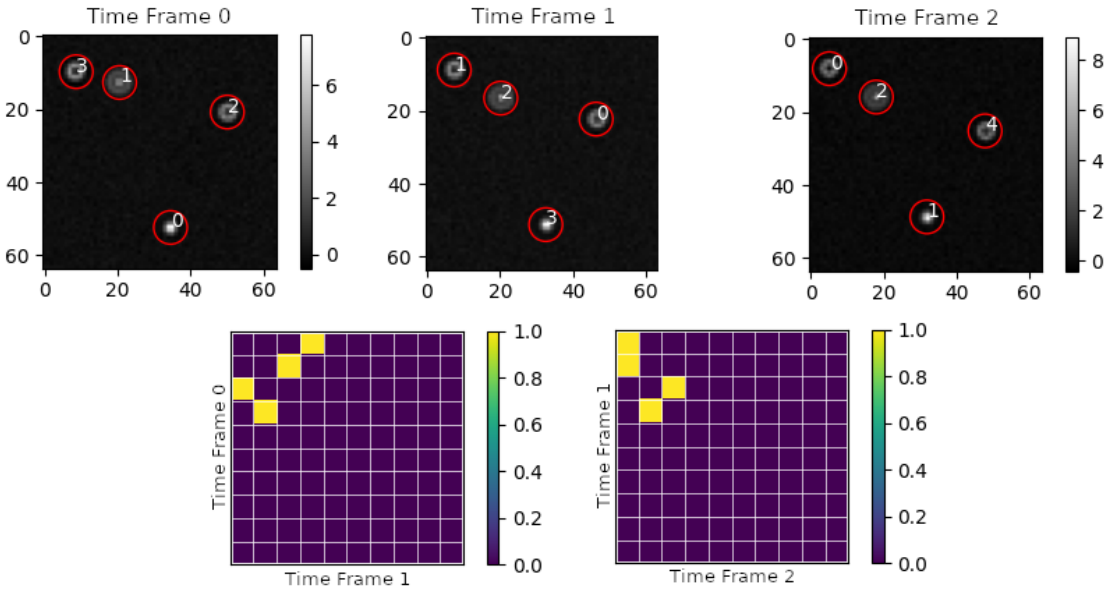
Table 4.3: The mean average error, missed detection rate, and false detection rate of the MVAE-model trained for 500 epochs with ”optimal” hyperparameters.

4.2.5 Detection without attention

In order to find out if the added attention helps the detection to improve a model without attention was trained using the same training and testing data set as well as the same hyperparameters used in the previous section (Section 4.2.4). The mean absolute error, missed detection rate, and false detection rate of both the model with attention as well as the model without attention can be seen in Table 4.4



(a) Successful object tracking with perfect detections and temporal trajectory links.



(b) Object tracking with perfect detections but imperfect temporal trajectory links between time frame 1 and time frame 2.

Figure 4.12: Two examples of object tracking over three frames using the fully trained MVAE model. The top row in each subfigure shows the original input image as well as the predicted object detection (red circles). The bottom row shows the temporal trajectory linking between objects in different time frames.

	MAE	Missed detections	false detections
With attention	0.00217	2.69 %	7.84 %
Without attention	0.00612	2.50 %	6.3 %

Table 4.4: The mean average error, missed detection rate, and false detection rate when comparing the MVAE with and without attention. Each number is an average over 3 test runs.

5
Discussion

5.1 GNN-model

In this section, the results obtained from testing the GNN-based model, that can be found in Section 4.1 will be discussed. The primary focus will be on analyzing the aspects that did not work as expected and understanding the reasons behind these limitations. In addition to this, what worked with the model will also be explored along with what can be learned from it.

5.1.1 Balancing the loss functions

The results from Section 4.1.5 show that balancing the loss of the model is a challenging task due to the use of three independent loss functions. Despite various attempts to balance the loss weights, the mean average error between the placement of the predicted detections and the target detections did not improve beyond a value of 0.1172. It's important to note that the full width and height of the image have been normalized to 1, meaning that an error of 0.1172 corresponds to approximately 12 % of one image side, which is quite high. This can clearly be visualized when comparing the red detection circles in the predicted and target image in Figure 4.5.

Furthermore, the results indicate that the model exhibits two distinct behaviors depending on the values of the loss weight hyperparameters. When λ_d is small, the model consistently predicts that every detection is a true detection. On the other hand, when λ_d and λ_l are relatively large compared to detection loss "outcompetes" the placement loss and the model, therefore, tends to predict only one object in the images, positioning it at the midpoint between all the target objects. An example of this can be seen in Figure 4.6. This is a local minimum that arises due to the fact that the loss function allows multiple targets to be assigned to the same prediction. It is possible that this local minimum could be eliminated by modifying the loss function to not allow for multiple assignments of targets to the same prediction.

5.1.2 Computational cost

In the GNN-neural network model, the calculation of the spatial attention between the APR-graph nodes and supernodes involves creating a matrix of size $M \times N$, where M is the number of supernodes and N is the number of APR-graph nodes. This can lead to computational and memory challenges as the number of supernodes

or APR-graph nodes grows larger. For the number of supernodes, this is not a big problem as it is a hyperparameter and approximation of the sum of objects in the input sequence and should, therefore, never grow uncontrollably. However, when dealing with larger image sizes, higher object occupancy, or longer image sequences, the number of APR-graph nodes can significantly increase. For instance, with a sequence length of 6 and an image size of 256×256, the number of APR-graph nodes may, in the worst case, approach 400 000. This results in the creation of a large attention matrix, which can pose computational and memory challenges.

There are ways to mitigate this problem. One approach is to connect only a subset of the APR-graph nodes to each supernode. By doing so, the attention calculation can be limited to the connected subset while still allowing supernodes to gather information from non-connected APR-graph nodes through the message-passing mechanism between APR-graph nodes. This strategy should reduce the overall computational burden with a slight reduction in prediction quality. Additionally, the APR-transform itself offers ways to reduce the number of APR-particles created. By adjusting the parameters of the transform, of which there are many, the number of APR-particles can be controlled. However, it's important to note that modifying these parameters would result in some loss of information during the transformation process.

5.1.3 Adaptive particle representation

This section will outline the reasons why using the APR to transform images into graphs was not suitable for the task investigated in this project.

Firstly, the anticipated result was that using the APR to transform images into graphs would be advantageous in terms of memory usage since compression is a key concept of APR. However, it was discovered that while using the straight APR would have provided memory benefits, our approach of transforming the APR into a graph by connecting particles with edges resulted in increased memory usage compared to using the raw images as every edge added increased memory usage. This posed a challenge, particularly considering the scalability issues that graph neural networks already face [45].

Secondly, and more importantly, the transformation of an image into a graph using this approach is not conceptually appropriate, as the primary advantage of employing graph neural networks lies in their ability to handle irregular data. Other studies have demonstrated the feasibility of object detection using graph neural networks, particularly on point clouds [35], [36], which are inherently irregular. In such cases, graph neural networks are preferred over attempts to fit irregular points into a regular grid. However, when working with images, we already have a regular grid structure, making grid-based approaches such as convolutional neural networks more suitable than attempting to force the image into a graph representation.

Therefore, to use APR for neural network tasks, a recommended approach is us-

ing it with convolutional neural networks. Previous successful work in this area, specifically the master's thesis by Jonsson [46], demonstrated the effectiveness of integrating APR with CNNs. Subsequent advancements have been made to enhance the usability of APR for such tasks, as highlighted in a more recent work by Jonsson [47]. These studies provide valuable insights into leveraging APR in conjunction with CNNs, offering a promising direction for applying this approach to neural network tasks that is not seen when combining APR with GNNs.

5.1.4 Linking trajectories

As discussed above both the placement and the detections were not working very well in this model. And trying different hyperparameters and depths of the network did not help improve this, as can be seen in Section 4.1.1, 4.1.2, and 4.1.3. However, the trajectory linking showed promise with the model consistently scoring an F1-score of over 80 %. This suggests that using attention between the latent features of objects in different time steps as a mechanism to link the trajectories of the objects through time has the potential to work but that it would require a model with a better and more robust method for object detection, which is the reason why the second model presented in this project, the MVAE-based model, was created.

5.2 MVAE-model

In this section, we will discuss the results obtained from testing the MVAE-based model, presented in Section 4.2. We will begin by exploring the optimization of hyperparameters, as they play a crucial role in the model's performance. We will then discuss other factors that can influence the model's overall performance. Lastly, we will provide a brief discussion on what worked well with the model and what requires further improvement.

5.2.1 Optimization of hyperparameters

Here, the results found in Section 4.2.1 when exploring hyperparameters related to the MVAE-based model will be discussed.

From the results in Figure 4.7, it can be determined that the best value for combine radius is somewhere around three and four, corresponding to a radius of approximately 5 or 6 percent of the image side length. Using a smaller radius leads to a lot of the detections placed over the same target object not being combined, leading to a bad linking F1-score as well as many false detections. Having a higher radius means that some detections that do not correspond to the same target object are combined leading to one of the targets not having an exact match and therefore increasing the amount of missed detections. Around a radius of three or four seems to be the sweet spot where both the false detection rate and missed detection rate are low, but this will, of course, differ between datasets.

The model's performance in relation to the number of entities used aligns with our expectations. Initially, the model exhibits poor performance with fewer than six entities, which is the maximum number of target objects in our simulation settings. This is understandable since it is impossible for the model to detect all the objects in cases where the number of target objects outnumbers the number of entities, resulting in a high missed detection rate and high MAE. As the number of entities increases to between six and ten, the performance improves significantly. The model is able to accurately place the majority of the detections on top of the actual target objects and combine them when multiple detections correspond to the same target.

When the number of entities exceeds ten, performance starts to deteriorate. This occurs because the top KL-divergences no longer exclusively include the target objects but also include the background. As a result, multiple false detections emerge, which are not eliminated as they do not correspond to any of the targets. This leads to a decline in both the linking F1-score and the false prediction rate. However, the MAE between the targets and predictions is less affected because the detections associated with the actual targets remain correctly placed.

Varying the softmax temperature in our experiments also yielded expected results, with noticeable impacts on the model's performance. Specifically, we observed that increasing the softmax temperature had a positive effect on inference performance, particularly in the area of temporal linking.

The improvement in temporal linking performance can be attributed to the increased decisiveness of the attention mechanism at higher softmax temperatures. By raising the temperature, the attention becomes more confident in determining which temporal links should be established. This results in each object in a given time frame receiving more informative input from themselves in other time frames while reducing the influence of other objects. Consequently, this enhanced training and inference process contributes to improved performance.

5.2.2 Uniqueness of objects

The model uses attention based on cosine similarity between the latent spaces of objects. As a result, it is reasonable to expect that the temporal linking performance of the model improves when there is a greater difference between objects. This expectation is supported by the results presented in Section 4.2.2.

Less expected is that the mean average error between the predicted and target placement of the detections also improved when there was a higher difference between the objects. This observation can be explained by considering the way the latent space of each object in a time frame receives information from other objects in different time steps based on the attention mechanism. When the difference between the objects is greater, it is expected that the attention between two unrelated objects in different time frames is lower. This results in less irrelevant and potentially damaging information being propagated between unrelated objects, leading to improved

accuracy in the predicted placement of detections.

These results suggest that the model has the potential to perform even better when applied to real-world data compared to the simulated data used in this study. This is particularly true when considering objects such as cells, where the inherent differences between objects can be substantial

5.2.3 Suquence length

Based on the findings presented in Section 4.2.3, it is evident that the sequence length does not have a significant impact on the inference performance of the model. This observation aligns with expectations, as each object within a time frame only receives information from the object one time step before and one time step after its own time frame. Therefore, adding more time steps to the sequence should not inherently improve or degrade the model's performance. The model's ability to capture temporal dependencies and make accurate predictions, therefore, remains consistent regardless of the sequence length as long as the temporal receptive field of each object remains limited to its neighboring time steps.

However, it is important to note that there are challenges associated with longer video sequences. Because the model processes whole image sequences at a time it becomes computationally expensive to handle long sequences as you need to keep all image frames in memory at the same time when predicting. This is an even bigger problem in training where a batch size larger forces the model to keep multiple of these video sequences in memory at the same time. While this was never a problem that came up in this work, it's important to keep in mind that the training and testing were done on an Nvidia A100 with 40 GB video memory and that the longest sequence that was trained and tested was 20 frames with a batch size of 8, leading to a total of 160 images of resolution 64x64 px had to be kept in memory.

If the computational and memory limitations become an issue, one possible solution is to split the long video sequences into multiple shorter overlapping sequences and process them individually. These shorter sequences can then be connected afterward to obtain the complete trajectory. But this approach comes at the cost of losing some of the end-to-end model's usability, as it might become more complex and less straightforward compared to a two-model-based architecture. The advantage of simplicity and ease of use, which is a key benefit of the end-to-end model, may be lost in this case.

A potential problem hypothesized when training on longer sequences was that longer sequences would increase the training time in a non-linear fashion. But the results shown in Figure 4.11 show that the time it takes to train the model one epoch does in fact scale linearly with time and training times should therefore not be a problem when training on longer sequences.

5.2.4 Detection with and without attention

When comparing the performance of the MVAE model with and without the added temporal attention layer, it was expected that the model with the attention layer would outperform the model without it in terms of the placement of detections. The assumption was that temporal attention would provide contextual information to improve the model's performance. However, the results presented in Section 4.2.5 indicate that there is no significant difference in terms of mean absolute error of detection placement, false detection rate, or missed detection rate. These findings suggest that the added attention layer does not have a significant impact, neither positive nor negative, on the model's ability to detect objects.

5.2.5 Resulting model

A significant advantage of utilizing a variation autoencoder model is its completely unsupervised nature, eliminating the need for labeled data during training. This characteristic provides several benefits over supervised models, as it eliminates the labor-intensive task of manually labeling data which can be subjective and prone to human error, potentially introducing inaccuracies into supervised models trained on such data.

In terms of detection placement, this model exhibited excellent performance, achieving as low mean absolute as 0.002, equivalent to just 0.2 % of an image side, as seen in Section 4.2.4. Additionally, the model demonstrated proficiency in accurately eliminating superfluous detections, resulting in a low missed detection rate. However, it did exhibit a slight drawback by generating a few excessive false detections, with a false detection rate of 7.84 %. If the false detections were isolated to a single time step and lacked connections to subsequent frames, they could be easily eliminated by discarding detections without temporal links. Unfortunately, the way the current model incorporates temporal linking by having every detection in each time frame always connect to exactly one detection in the subsequent frame makes this solution unfeasible. An alternative solution to removing false detections, therefore, needs to be made.

Another existing issue with the model pertains to the reliability of its temporal linking. Although the model achieved F1-scores exceeding 80 %, which is a promising result, it falls short of meeting the quality necessary for practical applications. To enhance the quality of these temporal links, one potential approach could involve introducing a separate loss function, similar to what is done in the GNN model. This modification is likely to improve the accuracy of temporal linking. However, it would come at the cost of requiring labeled training data, thereby negating the advantages associated with the model's unsupervised nature, as discussed earlier.

An observation from Figure 4.12b is that the model tends to struggle with temporal linkings between objects that have similar shapes and sizes but are positioned far apart, such as objects 0 and 4 in the third time frame. This suggests that the model

places more emphasis on object features like shape and brightness rather than their spatial positions when comparing objects across frames. To improve the temporal trajectory linking F1-score, it may be beneficial to encourage the model to pay more attention to the positional information of the objects by adding positional features to the object information that gets passed to the attention layer.

5.3 Future work

The MVAE-model exhibits promising potential, opening up numerous avenues for future research in the domain of end-to-end object tracking. The following list presents some of the most interesting directions for further exploration of this model:

- As discussed previously the temporal linking is not good enough to be trusted in real applications, therefore, the first thing to explore in future research should be focused on ways to improve the use of spatial position when calculating the temporal attention.

- Another approach to potentially increase the performance of the temporal trajectory linking that would be interesting to explore is to add a separate loss function similar to the one used in the GNN-model. This could have the potential to increase the temporal linking by quite a lot but would come at the cost of sacrificing the unsupervised nature of the model.

- Another improvement that could be interesting to investigate is allowing the model to create longer temporal links between objects instead of only allowing each object to connect to the object in neighboring frames. This would give the model the ability to easily remove false detections that are not connected to detections in other frames.

6
Conclusion

The idea of a single end-to-end object-tracking neural network model is promising. The supervised GNN was able to track the object in time with a consistent F1-score of over 80 % but did not succeed in localizing the objects with precision, while the unsupervised MVAE performed well in both.

The unsatisfactory performance of the GNN model can largely be attributed to its attempt to convert an image into a graph representation instead of harnessing the inherent advantages of image data. The transformation of the image into an APR-graph did not yield any discernible benefits. Additionally, the GNN model encountered challenges in effectively balancing multiple loss functions, adding to its complexity. Moreover, the computational cost of the model increased significantly as the attention matrix between the APR-graph and supernodes grew substantially with an increase in the number of nodes in the APR graph, which typically occurs when the image size increases.

The MVAE model, in contrast, yielded promising results in terms of detection and temporal linking. It achieved low mean absolute error in detecting object placements and demonstrated a minimal number of missed detections. There were slightly more false detections observed. The temporal linking performance was comparable to that of the GNN model, with an F1-score exceeding 80 %. While this is an encouraging result, further improvements are necessary to make the model viable for practical applications.

Even though neither of the models presented is able to perform at the same level as the current state-of-the-art models, the project shows that it is possible to track objects in time in a single end-to-end model. By spending more time refining the MVAE model further, by for example, making the attention put more focus on the spatial position of the objects, it is possible that the model will have the potential to outperform the current state-of-the-art models.

Bibliography

[1] Z. Liu *et al.*, "A survey on applications of deep learning in microscopy image analysis," en, *Computers in Biology and Medicine*, vol. 134, p. 104 523, Jul. 2021, ISSN: 00104825. DOI: 10.1016/j.compbiomed.2021.104523. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0010482521003176 (visited on 06/12/2024).

[2] H.-J. Cheng *et al.*, "A review for cell and particle tracking on microscopy images using algorithms and deep learning technologies," en, *Biomedical Journal*, vol. 45, no. 3, pp. 465–471, Jun. 2022, ISSN: 23194170. DOI: 10.1016/j.bj.2021.10.001. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2319417021001359 (visited on 06/12/2024).

[3] M. Weigert *et al.*, "Star-convex Polyhedra for 3D Object Detection and Segmentation in Microscopy," en, in *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Snowmass Village, CO, USA: IEEE, Mar. 2020, pp. 3655–3662, ISBN: 978-1-72816-553-0. DOI: 10.1109/WACV45572.2020.9093435. [Online]. Available: https://ieeexplore.ieee.org/document/9093435/ (visited on 06/12/2024).

[4] C. Versari *et al.*, "Long-term tracking of budding yeast cells in brightfield microscopy: CellStar and the Evaluation Platform," en, *Journal of The Royal Society Interface*, vol. 14, no. 127, p. 20 160 705, Feb. 2017, ISSN: 1742-5689, 1742-5662. DOI: 10.1098/rsif.2016.0705. [Online]. Available: https://royalsocietypublishing.org/doi/10.1098/rsif.2016.0705 (visited on 06/12/2024).

[5] L. L. Taixe *et al.*, "Automatic tracking of swimming microorganisms in 4D digital in-line holography data," en, in *2009 Workshop on Motion and Video Computing (WMVC)*, Snowbird, UT: IEEE, Dec. 2009, pp. 1–8. DOI: 10.1109/WMVC.2009.5399244. [Online]. Available: https://ieeexplore.ieee.org/document/5399244/ (visited on 06/12/2024).

[6] V. Ulman *et al.*, "An objective comparison of cell-tracking algorithms," *Nature Methods*, vol. 14, no. 12, pp. 1141–1152, Oct. 2017. DOI: 10.1038/nmeth.4473. [Online]. Available: https://doi.org/10.1038/nmeth.4473.

[7] C. Manzo and M. F. Garcia-Parajo, "A review of progress in single particle tracking: From methods to biophysical insights," *Reports on Progress in Physics*, vol. 78, no. 12, p. 124 601, Oct. 2015. DOI: 10.1088/0034-4885/78/12/124601. [Online]. Available: https://doi.org/10.1088/0034-4885/78/12/124601.

[8] B. Mehlig, *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*. Cambridge University Press, 2021. DOI: 10.1017/9781108860604.

[9] N. Sharma, V. Jain, and A. Mishra, "An Analysis Of Convolutional Neural Networks For Image Classification," en, *Procedia Computer Science*, vol. 132, pp. 377–384, 2018, ISSN: 18770509. DOI: 10.1016/j.procs.2018.05.198. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877050918309335 (visited on 06/12/2024).

[10] A. R. Pathak, M. Pandey, and S. Rautaray, "Application of Deep Learning for Object Detection," en, *Procedia Computer Science*, vol. 132, pp. 1706–1717, 2018, ISSN: 18770509. DOI: 10.1016/j.procs.2018.05.144. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877050918308767 (visited on 06/12/2024).

[11] W. Baccouch *et al.*, "A comparative study of CNN and U-Net performance for automatic segmentation of medical images: Application to cardiac MRI," en, *Procedia Computer Science*, vol. 219, pp. 1089–1096, 2023, ISSN: 18770509. DOI: 10.1016/j.procs.2023.01.388. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877050923003976 (visited on 06/12/2024).

[12] L. Alzubaidi *et al.*, "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions," en, *Journal of Big Data*, vol. 8, no. 1, p. 53, Mar. 2021, ISSN: 2196-1115. DOI: 10.1186/s40537-021-00444-8. [Online]. Available: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8 (visited on 06/12/2024).

[13] N. I. Widiastuti, "Convolution Neural Network for Text Mining and Natural Language Processing," en, *IOP Conference Series: Materials Science and Engineering*, vol. 662, no. 5, p. 052 010, Nov. 2019, ISSN: 1757-8981, 1757-899X. DOI: 10.1088/1757-899X/662/5/052010. [Online]. Available: https://iopscience.iop.org/article/10.1088/1757-899X/662/5/052010 (visited on 06/12/2024).

[14] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE Journal*, vol. 37, no. 2, pp. 233–243, 1991. DOI: https://doi.org/10.1002/aic.690370209.

[15] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, 2022. arXiv: 1312.6114 [stat.ML].

[16] J.-H. Jang *et al.*, "Unsupervised feature learning for electrocardiogram data using the convolutional variational autoencoder," en, *PLOS ONE*, vol. 16, no. 12, U. Qamar, Ed., e0260612, Dec. 2021, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0260612. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0260612 (visited on 06/12/2024).

[17] N. Miolane *et al.*, "Estimation of Orientation and Camera Parameters from Cryo-Electron Microscopy Images with Variational Autoencoders and Generative Adversarial Networks," en, in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Seattle, WA, USA: IEEE, Jun. 2020, pp. 4174–4183, ISBN: 978-1-72819-360-1. DOI: 10.1109/

CVPRW50498.2020.00493. [Online]. Available: https://ieeexplore.ieee.org/document/9150810/ (visited on 06/12/2024).

[18] A. Vahdat and J. Kautz, "NVAE: A Deep Hierarchical Variational Autoencoder," en,

[19] C. Nash and C. K. I. Williams, "The shape variational autoencoder: A deep generative model of part-segmented 3D objects," en, *Computer Graphics Forum*, vol. 36, no. 5, pp. 1–12, Aug. 2017, ISSN: 0167-7055, 1467-8659. DOI: 10.1111/cgf.13240. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1111/cgf.13240 (visited on 06/12/2024).

[20] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar Variational Autoencoder," en,

[21] M.-T. Luong, H. Pham, and C. D. Manning, *Effective Approaches to Attention-based Neural Machine Translation*, en, arXiv:1508.04025 [cs], Sep. 2015. [Online]. Available: http://arxiv.org/abs/1508.04025 (visited on 06/12/2024).

[22] W. Yu *et al.*, "A Survey of Knowledge-enhanced Text Generation," en, *ACM Computing Surveys*, vol. 54, no. 11s, pp. 1–38, Jan. 2022, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3512467. [Online]. Available: https://dl.acm.org/doi/10.1145/3512467 (visited on 06/12/2024).

[23] A. Vaswani *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL].

[24] F. Scarselli *et al.*, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009. DOI: 10.1109/TNN.2008.2005605.

[25] W. Fan *et al.*, "Graph Neural Networks for Social Recommendation," en, in *The World Wide Web Conference*, San Francisco CA USA: ACM, May 2019, pp. 417–426, ISBN: 978-1-4503-6674-8. DOI: 10.1145/3308558.3313488. [Online]. Available: https://dl.acm.org/doi/10.1145/3308558.3313488 (visited on 06/12/2024).

[26] Q. Cao *et al.*, "Popularity Prediction on Social Platforms with Coupled Graph Neural Networks," en, in *Proceedings of the 13th International Conference on Web Search and Data Mining*, Houston TX USA: ACM, Jan. 2020, pp. 70–78, ISBN: 978-1-4503-6822-3. DOI: 10.1145/3336191.3371834. [Online]. Available: https://dl.acm.org/doi/10.1145/3336191.3371834 (visited on 06/12/2024).

[27] X.-M. Zhang *et al.*, "Graph Neural Networks and Their Current Applications in Bioinformatics," en, *Frontiers in Genetics*, vol. 12, p. 690 049, Jul. 2021, ISSN: 1664-8021. DOI: 10.3389/fgene.2021.690049. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fgene.2021.690049/full (visited on 06/12/2024).

[28] W. Hu *et al.*, *Strategies for Pre-training Graph Neural Networks*, en, arXiv:1905.12265 [cs, stat], Feb. 2020. [Online]. Available: http://arxiv.org/abs/1905.12265 (visited on 06/12/2024).

[29] G. Kumichev, *The intuition behind graph convolutions and message passing*, Jan. 2022. [Online]. Available: https://towardsdatascience.com/the-intuition-behind-graph-convolutions-and-message-passing-6dcd0ebf0063.

[30] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, 2017. arXiv: 1609.02907 [cs.LG].

[31] Q. Li, Z. Han, and X.-m. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. DOI: 10.1609/aaai.v32i1.11604.

[32] P. Veličković *et al.*, *Graph attention networks*, 2017. DOI: 10.48550/ARXIV.1710.10903. [Online]. Available: https://arxiv.org/abs/1710.10903.

[33] B. L. Cheeseman *et al.*, "Adaptive particle representation of fluorescence microscopy images," *Nature Communications*, vol. 9, no. 1, Dec. 2018. DOI: 10.1038/s41467-018-07390-9. [Online]. Available: https://doi.org/10.1038/s41467-018-07390-9.

[34] J. Pineda *et al.*, "Geometric deep learning reveals the spatiotemporal features of microscopic motion," *Nature Machine Intelligence*, vol. 5, no. 1, pp. 71–82, Jan. 2023. DOI: 10.1038/s42256-022-00595-0. [Online]. Available: https://doi.org/10.1038/s42256-022-00595-0.

[35] W. Shi and R. Rajkumar, "Point-gnn: Graph neural network for 3d object detection in a point cloud," *CoRR*, vol. abs/2003.01251, 2020. arXiv: 2003.01251. [Online]. Available: https://arxiv.org/abs/2003.01251.

[36] D. Gehrig and D. Scaramuzza, *Pushing the limits of asynchronous graph-based object detection with event cameras*, 2022. arXiv: 2211.12324 [cs.CV].

[37] D. Hendrycks and K. Gimpel, *Gaussian error linear units (gelus)*, 2016. DOI: 10.48550/ARXIV.1606.08415. [Online]. Available: https://arxiv.org/abs/1606.08415.

[38] E. Noutahi *et al.*, *Towards interpretable sparse graph representation learning with laplacian pooling*, 2019. DOI: 10.48550/ARXIV.1905.11577. [Online]. Available: https://arxiv.org/abs/1905.11577.

[39] P.-H. Wang *et al.*, *Contextual temperature for language modeling*, 2020. DOI: 10.48550/ARXIV.2012.13575. [Online]. Available: https://arxiv.org/abs/2012.13575.

[40] P. Matula *et al.*, "Cell tracking accuracy measurement based on comparison of acyclic oriented graphs," *PLOS ONE*, vol. 10, no. 12, T. Abraham, Ed., e0144959, Dec. 2015. DOI: 10.1371/journal.pone.0144959. [Online]. Available: https://doi.org/10.1371/journal.pone.0144959.

[41] M. Simonovsky and N. Komodakis, *Graphvae: Towards generation of small graphs using variational autoencoders*, 2018. DOI: 10.48550/ARXIV.1802.03480. [Online]. Available: https://arxiv.org/abs/1802.03480.

[42] C. Nash *et al.*, "The multi-entity variational autoencoder," 2018.

[43] I. Higgins *et al.*, "Beta-vae: Learning basic visual concepts with a constrained variational framework," en, 2017.

[44] B. Midtvedt *et al.*, "Quantitative digital microscopy with deep learning," *Applied Physics Reviews*, vol. 8, no. 1, p. 011 310, Mar. 2021. DOI: 10.1063/5.0034891. [Online]. Available: https://doi.org/10.1063/5.0034891.

[45] H. Ma, Y. Rong, and J. Huang, "Graph neural networks: Scalability," in *Graph Neural Networks: Foundations, Frontiers, and Applications*, L. Wu *et al.*, Eds. Singapore: Springer Nature Singapore, 2022, pp. 99–119, ISBN: 978-981-16-

6054-2. DOI: 10.1007/978-981-16-6054-2_6. [Online]. Available: https://doi.org/10.1007/978-981-16-6054-2_6.

[46] J. Jonsson, "Aprnet: Convolutional neural networks for a content-adaptive particle representation of images," M.S. thesis, Chalmers University of Technology, 2019. [Online]. Available: https://odr.chalmers.se/items/d00d1869-0773-4b08-bc71-a44ef0712783.

[47] J. Jonsson *et al.*, "Parallel discrete convolutions on adaptive particle representations of images," 2021. DOI: 10.48550/ARXIV.2112.03592. [Online]. Available: https://arxiv.org/abs/2112.03592.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY