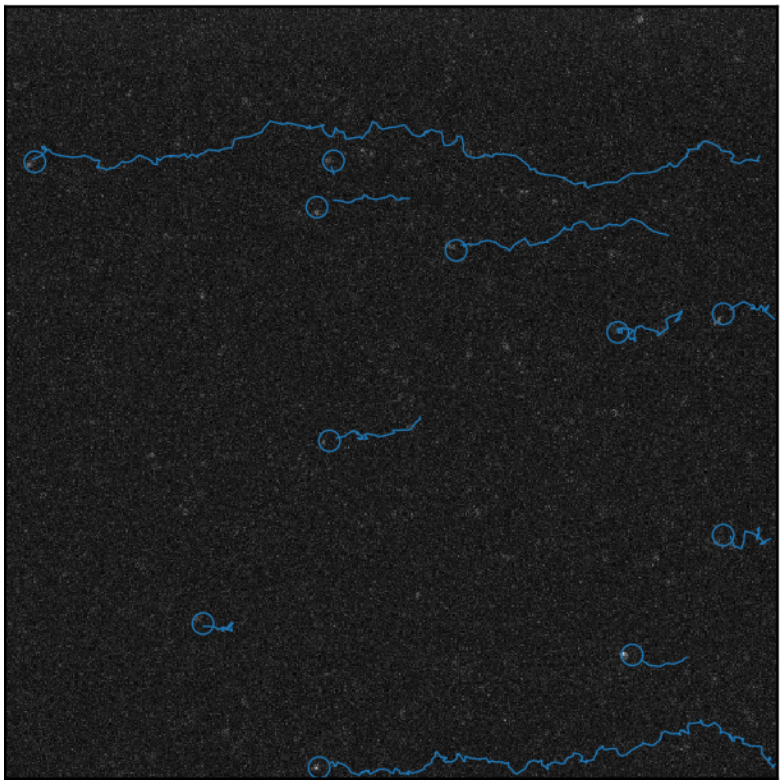


Deep learning for particle tracking

Bachelor's thesis at the department of Physics

Adrian Lundell
David Tonderski
Fredrik Meisingseth
Dennis Kristiansson

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2020

Deep learning for particle tracking

Supervisors: Daniel Midtvedt, Giovanni Volpe
Examiner: Lena Falk

Bachelor's Thesis TIFX04-20-17
Department of Physics
Chalmers University of Technology

Cover: Visualization of tracked particles (circled) in an image from microscopy.

Acknowledgements

First of all, thank you, Daniel, for all the help you have given us during the course of this project, for your patience and for your guidance.

We also want to thank you, Giovanni, for your great input at the start of the project as well as the rest of the group that developed DeepTrack for the base that our research has been built upon.

Finally, we want to thank you, Erik, for your help in our testing of U-track on real data.

Adrian, David, Fredrik and Dennis

Deep learning for particle tracking

Adrian Lundell David Tonderski
Fredrik Meisingseth Dennis Kristiansson
Supervisor: Daniel Midtvedt Supervisor: Giovanni Volpe

May 2020

Abstract

The use of machine learning for classification has in recent years spread into a wide range of disciplines, amongst them the detection of particles for particle tracking on microscopy data. We modified the Python package DeepTrack, which makes use of deep learning to detect particles, creating a package called U-Track. By using a new network architecture based on a U-Net, better performance and higher computational efficiency than DeepTrack was achieved on images with multiple particles. Furthermore, functionality to track detected particles over series of frames was developed. The application of U-Track on experimental data from two-dimensional flow nanometry produced tracks consistent with theory, as well as tracking larger quantities of particles over longer periods of time compared to a digital filter based benchmark algorithm.

Sammandrag

De senaste åren har användningen av maskininlärning för lösning av klassificeringsproblem spridits till en mängd discipliner, däribland partikeldetektion för partikelspårning på mikroskopidata. Vi modifierade Pythonpaketet DeepTrack som använder djupinlärning för att detektera partiklar och skapade paketet U-Track. Med en ny nätverksarkitektur baserad på ett U-Net uppnåddes bättre resultat och högre beräkningseffektivitet än Deeptrack på bilder med flera partiklar. Vidare lades funktionalitet för att spåra partiklar över en sekvens av bilder till. Användningen av U-Track på experimentell data från tvådimensionell flödesnanometri producerade resultat som överrenstämmer med teorin, samt med fler partiklar spårade över längre tidsperioder jämfört med en algorithm baserad på digital filtrering.

Contents

1 Introduction

The usage of machine learning algorithms in image analysis practices has been an increasing trend in the last decade as technological improvements has made the required data handling feasible. Advantages of neural networks include robustness against disturbances in data, generality for performance on different types of data and a possibility for automation, decreasing both time consuming work load and subjective decision making in scientific image analysis.

In a 2019 paper, the research group of Giovanni Volpe showcased how their machine learning software named *DeepTrack* captured these advantages and outperformed several manual algorithms in the field of microscopy particle tracking [1]. The result has applications in a variety of fields, including calibrating optical tweezers, measuring biomolecular forces and monitoring growth of crystals. However, the network used in DeepTrack only detects one particle for each image, handling images containing multiple particles by partitioning them into parts and analysing each part separately. For applications on large images or images of high particle density, this method results in long computation times.

An approach for particle tracking in two-dimensional microscopy adapted for tracking a larger number of particles in one image is therefore of interest. Hence, one of the purposes of this project is to change the network architecture to a *U-Net*. This type of network was first proposed by Ronneberger, Fischer, and Brox in 2015 [2] and has since become one of the most commonly used network architectures in image segmentation, especially within medical analysis [3].

1.1 Purpose

The purpose of this project is to develop a framework for extracting particle tracks from microscopy data. A first step is a rework of DeepTrack based on a U-Net neural network structure, which is better suited for images containing multiple particles. With this new approach, the aim is to perform at least as well as the preexisting software with regard to classification performance and computational efficiency, while providing improved performance on large images and images of high particle density.

Secondly, a pipeline for post-processing the output from the network is developed, turning image segmentation data on a sequence of frames into tracked particles upon which statistical analysis may be performed. Post-processing further allows for evaluation of particles from situational context between frames not considered by the network, and may thus help in filtering of falsely predicted particles. Contextual information is derived from expected physical behaviour of the input data and so this part was developed for use on particular experimental data supplied from a PhD thesis project [4]. While the network and post-processing methods are general, a particular use case helps exemplifying usage considerations and showcases the usability of the final package. This final package is from here on called *U-Track* [5].

1.2 Scope

Development is confined to the use of a U-Net architecture in tandem with a particle tracking algorithm based on linear sum assignment. These methods are specifically chosen for the problem of multi frame tracking of many particles and thereby evaluation is focused on showing improved results on this problem.

1.3 Report structure

This project can be thought of as consisting of three subprojects, to some extent carried out in parallel throughout all parts of the report:

1. Implementing a U-Net architecture adapted for particle tracking.
2. Improving the image generation routine used in network training.
3. Developing a post-processing pipeline to extract particle tracks from the network's output.

In chapter 2, the basic theory behind the neural network, particle tracking and some relevant metrics is given. Chapter 3 treats the developed methods and gives some general insight into the considerations behind them, as well as the methods used for evaluation. This is followed by a presentation of results in chapter 4 where each project part is evaluated or benchmarked. In chapter 5 and 6, the methods and results are discussed, with a finishing conclusion of the overall outcome of the project.

2 Theory

The theory chapter starts off with a subchapter on the architecture of the network, explaining the roles of its most important building blocks and how they interconnect, as well as discussing the meaning of the network's output. Following this is a subchapter presenting the metrics and terminology used to quantitatively measure the network's performance. In the last subchapter, the groundwork for post-processing of the network's output in connection to a real world use case is laid out, specifically introducing the concept of particle tracking in the context of two dimensional flow nanometry. This includes a description of the input data, theory about which information can be extracted, and methods for statistically analysing this information.

2.1 Network architecture

A neural network transforms input data into a set of features containing information about the data. The term network architecture refers to the arrangement of *layers* in the neural network. A layer is a collection of artificial neurons, which are modelled as mathematical operations within a neural network. In this project, three types of layers are mainly used (see figure 1):

- *Convolutional layers*, which in our case utilise convolution to detect local features in an input image. In practice, this means that a set of *kernels* (or matrices) is slid over the input in steps. For each step and for each kernel, the dot product between the kernel and the corresponding part of the image is inserted into the output image for that kernel, resulting in an output image for each kernel. This dot product can be understood as the weighted average of that part of the input. The number of output images (or *feature maps*) is called the number of *feature channels* of the layer. Before applying this layer, padding is often added to the input to preserve the image resolution.
- *Max pooling layers*, which split the input image into smaller rectangles, choosing the largest value in each such rectangle and inserting it into the output image. This reduces the resolution of the input image, allowing the network to detect large-scale features in the following convolutional layers. The size of these rectangles is called the *pool size*.

- *Upsampling layers*, which for each input pixel create a small rectangle of pixels and insert it into the output image. All the pixels in the output rectangle are equal to the input pixel. The size of these rectangles is called the *upsampling factor*. Upsampling layers are used to increase the resolution of the input image.

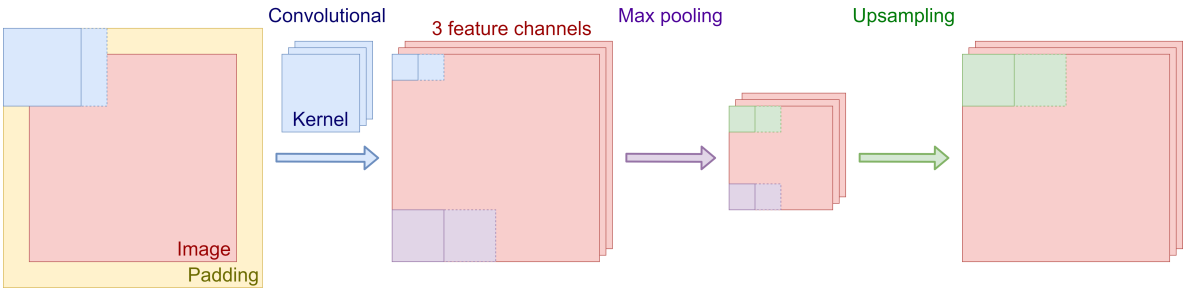


Figure 1: Visualisation of the three main types of layers that make up the neural network used in this project. First, a convolutional layer (blue) is applied to the input image with padding added. Then, a max pooling layer is applied (purple), followed by an upsampling layer (green). The upsampling factor is identical to the pool size, which means that the final resolution is the same as the input image resolution.

After each of these layers, an *activation function* is usually applied to the output before passing it on to the next layer. In this project, the ReLU activation function visualised in figure 2 is mainly used. It can be described as the mapping of all negative values of the data to 0, while leaving other values unchanged.

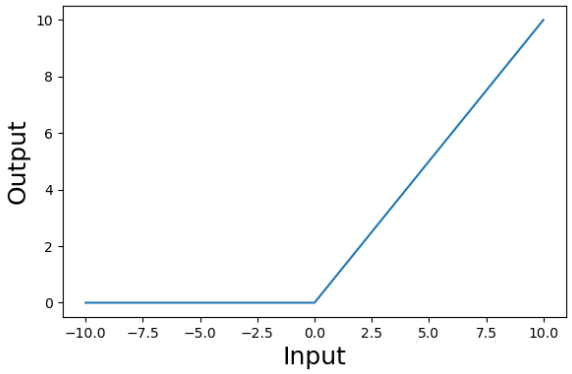


Figure 2: Visualisation of the ReLU activation function, which is used for most layers in both the DeepTrack and the U-Track networks.

The original DeepTrack architecture consists of a convolutional base followed by a *dense top*. This dense top connects all outputs from the convolutional base to each one of the network's output features. The network's features are the x and y position of the center of the particle, as well as its distance from the center of the image. This approach is suitable for images containing a single particle, but the generalization to multiple particle images is often inefficient, which is shown in this project.

A more common approach for this problem is image segmentation, which assigns information locally

to an image. In our case, this means that U-Track's features consist of images of the same size as the input data, outputting information on a pixel-to-pixel basis. This is accomplished by using an architecture based on a U-Net, which can be understood as a combination of two parts:

1. A contracting path, sometimes understood as the encoder because of it's ability to detect large-scale patterns [6]. It can be seen as a series of blocks consisting of convolutional layers followed by a max pooling layer. Each such block decreases the resolution of the feature maps while increasing the number of feature channels to retain information. This process is repeated until the minimum desired feature map resolution is reached.

2. An expanding path or decoder, which is applied after the contracting path. Here, the blocks consist of convolutional layers followed by an upsampling layer. Therefore, each block increases the resolution of the feature maps while decreasing the number of feature channels, inversely to the blocks in the contracting path. To retain the more detailed information from the contracting path, the output of each block is also concatenated with the corresponding feature map from the contracting path before being fed into the next block. After the final block, convolutional layers are applied to reach the desired output shape.

An important feature of the U-Net architecture is that it is scale invariant. This means that the network can be used on a variety of image sizes without retraining. Additionally, its prediction quality is not affected by the image size. An example of the original U-Net architecture is shown in figure 3.

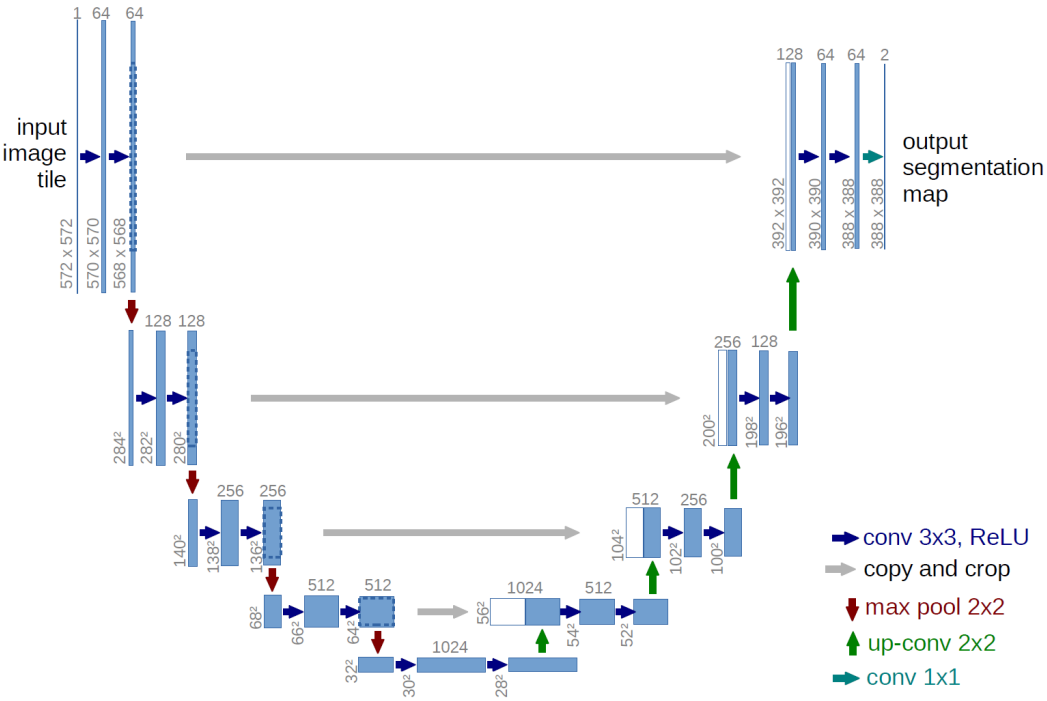


Figure 3: Visualisation of the original U-Net architecture, using an example with an input size of 572x572 pixels and 32x32 pixels at the lowest resolution. The blue boxes represent feature maps, while the arrows denote operations (or layers). Note that padding isn't used for the convolutional layers, thus resulting in a decreasing resolution. Source: adapted from [2].

2.2 Measures of network's detection performance

One purpose of our network is to classify each pixel in an image either as being part of a particle or not. This can also be done using a scanning box method, with the idea being to classify if such a scanning box contains a particle or not. The terminology being presented in this subsection is formulated for pixels, but can analogously be translated to be used regarding these scanning boxes. It is presented for the use case of particle detection, but it can be generalized to any binary classification problem.

To evaluate the binary map resulting from the predictions for each pixel, one needs an answer key, called a *label*. Each pixel of the label corresponding to the presence of a particle is called a *positive (P)*, while all others are *negatives (N)*. A correct prediction that a pixel is a positive is called a *true positive (TP)*, whilst an incorrect prediction that a pixel is a positive is called a *false positive (FP)*. We define *true negatives (TN)* and *false negatives (FN)* in the corresponding way.

The *true positive rate (TPR)*, also known as *recall*, is calculated as $\frac{TP}{TP+FN}$ and the *false positive rate (FPR)* is calculated as $\frac{FP}{FP+TN}$. The *precision* is calculated as $\frac{TP}{TP+FP}$. Precision describes the amount of positive-guesses that are correct, while recall describes how many of the positives are correctly classified. There are several other accepted measures, each describing a certain aspect of the classifier's performance. Summarising different measures into one value may also be done in different ways, with the one primarily used when evaluating U-Track being calculated from a so called *ROC curve*. A ROC (Receiver Operating Characteristics) curve is constructed by plotting TPR against FPR for different values of some parameter. The best possible prediction would yield a point on the ROC curve in the upper left corner (0,1), meaning TPR = 100% and FPR = 0%, whilst worse predictions would yield points further away from (0,1). Therefore, the *distance to the upper left corner (DULROC)*[1] may be used as a measure of the performance of the classifier[7], which is what we use in this project.

2.3 Two dimensional flow nanometry

Two dimensional flow nanometry (2DFN) is a subclass of the larger problem of tracking nanoparticles by illuminating them and recording their *fluorescence intensity* [4]. The tracking then involves detecting particles in each frame and linking corresponding particles over a series of frames forming *particle tracks* $(\mathbf{x}_n)_{n=0}^{N}$ for a particle tracked over N frames. These tracks are then used for determining the properties of the particles[2]. 2DFN is characterized by tying the particles to a surface through one or more DNA tethered to each particle, constraining the movement to two dimensions, and subjecting them to a laminar flow, causing them to move in a deterministic fashion. This setup simplifies tracking and greatly improves intensity determination compared to usual nano-tracking analysis in three dimensions. Even under these conditions recorded intensities may vary much within tracks, but a reasonable range is expected [4].

2.3.1 Determination of particle radius

Typically, scientific interest lies in statistically determining the radius of the particles. This radius is smaller than the wavelength of the reflected light, and thus the radius seen in the video differs from the actual particle radius. For this purpose, current state-of-the-art techniques make use of the diffusion of the particle [4].

Due to the laminar flow of the surrounding medium, particles will be influenced by flows with dif-

[1] There is no widely accepted abbreviation for this measure, so for the simplicity of this report, we chose this one.
[2] For convenience, particles are identified with their positions throughout this report.

ferent velocities. This is due to the fact that the flow velocity of the medium increases with the distance from the surface. Each particle experiences an average velocity $\mathbf{v}_{medium}$ approximately dependent on the radius r of the particle and two constants describing the laminar flow, u_0 and λ. The speed $v_{medium} = |\mathbf{v}_{medium}|$ dependence is described by

$$v_{medium} = u_0(r + \lambda) \tag{1}$$

The resulting particle velocity $\mathbf{v}_f$ is similarly in the direction of the laminar flow but of a different speed. Perpendicular to this motion, direction $\mathbf{e}_D$, it can be shown that the particle only undergoes diffusion, a behaviour showcased in figure 4. Particle speed v_f and the characterizing diffusion coefficient D are estimated for each tracked particle. The particle radius may then be calculated through the diffusion relations of Stokes and Einstein, assuming the friction of the linking DNA to be negligible [4]. Other needed constants are the Boltzmann's constant k_B, the temperature T, the viscosity η, and A, representing inhomogeneities in the flow, where the non-physical constants A, u_0 and λ are determined by calibration on known particles. The resulting equation is

$$\frac{k_B T v_f}{D} = A\eta r u_0(r + \lambda), \tag{2}$$

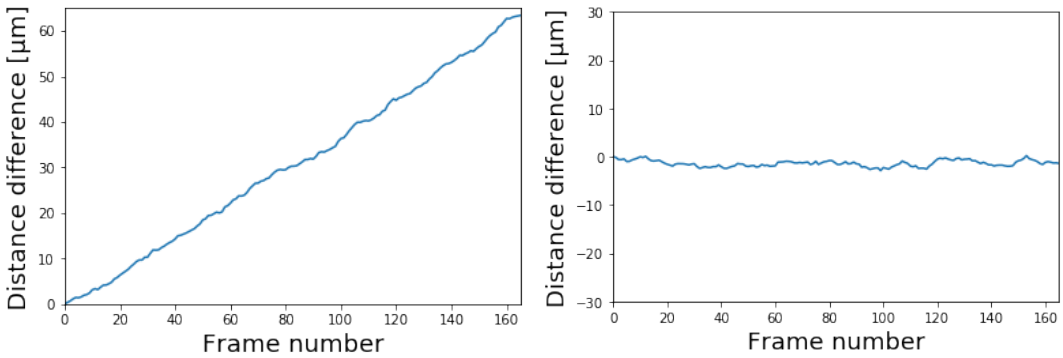
which can be rearranged letting $a = \frac{k_B T v_f}{D}$ and $b = A\eta u_0$ and picking the physically feasible root, yielding the particle radius

$$r = \frac{\sqrt{\lambda^2 + 4ab} - \lambda}{2b}. \tag{3}$$

If v_{medium} is approximated as equal for all particles, neglecting the influence of the radii, the particle radius r is given by

$$r \approx \frac{v_f}{D} \cdot const, \tag{4}$$

where $const = k_b T / A\eta v_{medium}$. The intensities of the particles are also of interest, since intensity is expected to be correlated with the particle radius, $I_{rel} \sim r^2$.



(a) Distance difference in the direction of $\mathbf{v}_f$

(b) Distance difference in the direction of $\mathbf{e}_D$

Figure 4: Distance difference from the first frame for a track of 165 frames. The particle moves 60 µm in the direction of $\mathbf{v}_f$ while staying around a displacement of zero in the direction of $\mathbf{e}_f$, showing the differentiating movement behaviours.

2.3.2 Covariance based diffusion estimators

The statistic estimations used throughout the rest of this section are based on a paper by Vestergaard, Blainey, and Flyvbjerg, which provides a method for determining the diffusion coefficient D for individual particle tracks through the covariance matrix of position displacements of a track [8]. Furthermore, it provides a method for quality assessment of the tracking through the variance of localisation errors σ^2 and a test of the assumption that the particles undergo diffusion, on which the estimates are grounded. As opposed to other methods, this method is proven to be unbiased and practically optimal for relatively short time series, making it suitable for this project [8].

Knowing the direction of flow for all particles allows for computation of the perpendicular unit vector $\mathbf{e}_D$ in the direction of diffusion. Diffusion displacements $(\Delta x_n)_{n=1}^N$ may then be computed as projections of position differences onto this vector.

$$\Delta x_n = (\mathbf{x}_n - \mathbf{x}_{n-1}) \cdot \mathbf{e}_D \tag{5}$$

and their coviariances

$$\overline{\Delta x_n \Delta x_{n+j}} = \frac{1}{(N-j)^2} \sum_{n=1}^{N-j} \Delta x_n \Delta x_{n+j}. \tag{6}$$

Through a theoretical model of the camera's capture of the particle motion, it is then possible to find closed-form expressions for the covariance matrix of the displacements as a function of D, σ^2, the time step between two consecutive frames Δt, and the constant R related to the lens shutter behaviour. Solving for σ^2 and D yields equations 7 and 8

$$\hat{\sigma}^2 = R\overline{(\Delta x_n)^2} + (2R-1)\overline{\Delta x_n \Delta x_{n+1}}, \tag{7}$$

$$\hat{D} = \frac{\overline{(\Delta x_n)^2} - 2\sigma^2}{2(1-2R)\Delta t}. \tag{8}$$

where an average σ^2 from all tracks is used in equation 8. Assuming approximately equal tracking errors, this is a more stable estimator for D than using individually computed $\hat{\sigma^2}$ for each track. This value is expected to be around $2\,\mu m^2/s$ for particles with one DNA link between the particle and surface, decreasing with more links. [4].

2.3.3 Testing of the diffusion hypothesis

Testing of whether a track is consistent with free diffusion is also based on the covariances of particle displacement, specifically on the assumption that displacements further than one time step apart should not covariate [8]. More precisely, covariances for large N should be normally distributed around zero, with a variance given by equation 10

$$\overline{\Delta x_n \Delta x_{n+j}} \approx 0, \text{ if } j > 1 \tag{9}$$

$$\mathrm{var}(\overline{\Delta x_n \Delta x_{n+j}}) = \frac{\alpha^2 + 4\alpha\beta + 6\beta^2}{N-j} - \frac{2\beta^2}{(N-j)^2}, \text{ if } j > 1. \tag{10}$$

A more robust test transforms displacements $(\Delta x_n)_{n=1}^N$ with a discrete sinus transform to instead examine the periodogram of values $\check{P}_k$, as this transformation can be proven to remove statistical dependencies within the type of particle tracks treated here. Statistical independent data ensures that

results from several shorter tracks ($N \approx 100$) may be averaged over for a certainty equal to that of one long track assuming equal values of D and σ^2.

$$\Delta\check{x}_k = \Delta t \sum_{n=1}^{N} \sin(\frac{\pi k t}{N+1})\Delta x_n \tag{11}$$

$$\check{P}_k = \frac{2(\Delta\check{x}_k)^2}{[(N+1)\Delta t]}. \tag{12}$$

These results may then be compared to the expected periodogram for free diffusion

$$\langle\check{P}_k\rangle = 2D(\Delta t)^2 + 2\left[\sigma^2\Delta t - 2DR(\Delta t)^2\right](1 - \cos\frac{\pi k}{N+1}) \tag{13}$$

by testing that $\frac{\check{P}_k}{\langle\check{P}_k\rangle}$ should follow a gamma distribution of shape parameter $\frac{1}{2}$ and scale parameter 2.

3 Method

The method chapter begins with a subchapter detailing the inner workings of the U-Net, including the attributes of its layers, the loss function and the training routine. Then follows a subchapter on post-processing, elaborating on the extraction of useful information from the output of the U-Net, as well as the details of how information about particles in separate images can be connected into tracks of said particles over a sequence of images. Then, there are three subchapters on the evaluation of different aspects of the package's functionality. The final subchapter is of the need for a new routine for image generation and the different approaches that are considered.

3.1 Network architecture

The network architecture used is a U-Net, as described in the section 2.1. The contracting path contains 4 blocks. Each such block contains 3 convolutional layers with padding, 3 by 3 kernels and ReLu activation functions. These are then followed by a max pooling layer. The initial number of feature channels used in the convolutional layers is 8, and the number is doubled after each max pooling layer. These max pooling layers have a pool size of 2 by 2, which corresponds to halving the image resolution in each block.

The expanding path similarly consists of 4 blocks. The convolutional layers are identical to the ones used in the contracting path. The upsampling layers have size 2 by 2, which corresponds to a doubling of the image resolution. The number of feature channels is halved after each upsampling layer. In the concatenations, half the feature map from the contracting path is dropped to prevent *overfitting*.

Overfitting is what happens when a model is too specialized on training data and does not generalize well to data it has never seen before. In this project, the model never trains on the same image twice, so it will not overfit to a certain image. However, the simulated data is often only an approximation for the experimental data. In this case, the network might overfit to the simulated data type. Then, if the experimental data is of a slightly different type, the network's performance might decrease. This effect should mainly be counteracted by simulating the data as realistically as possible, but it can also be reduced by the dropout layer.

After the four blocks in the expanding path, two more convolutional layers of the same type are

applied. Finally, a convolutional layer with padding, a 1 by 1 kernel, no activation function and 5 feature channels is applied. The output from this last layer is the output of the network. The five features of the output are:

1. A measure of the network's certainty that the pixel belongs to a particle. A sigmoid can be applied to this feature, mapping it to the interval $(0, 1)$, which allows it to be interpreted as a probability. Therefore, the first feature of the prediction after the sigmoid is in this report called the *probability prediction*.

The other features are only meaningful for pixels that are part of a particle. Therefore, the network is designed to only train on those features for pixels that are part of a particle.

2. The pixel's x-distance to the center of the particle.
3. The pixel's y-distance to the center of the particle.
4. The particle's radius.
5. The particle's relative intensity. Simulated particles are generated using a Bessel function, and the relative intensity is the constant with which this Bessel function is multiplied. For Bessel order 1, this is the highest point in the curve.

This approach is proposed with the aim of achieving higher accuracy in particle localization and characterisation.

3.2 Training and loss function

The term *training* denotes the process of minimizing the *loss function* for training data. The loss function is a measure of the discrepancy between label entries y_{ij} and predictions p_{ij}, where i is the feature index and j is the pixel index.

A common loss function used for image segmentation problems is binary cross-entropy, which measures the performance of classification problems with outputs between 0 and 1 [9]. In particle tracking, data sets are often highly imbalanced with many more background pixels than pixels belonging to particles. This can lead to the training getting stuck in local minima where the network detects no particles. To avoid this, true positive detections are valued over true negatives through the weighted binary cross-entropy loss function

$$L_1 = -\sum_{j=0}^{N} \beta \cdot y_{1j} \log\left(\tilde{p}_{1j}\right) + \left(1 - y_{1j}\right) \log\left(1 - \tilde{p}_{1j}\right), \tag{14}$$

where y_{1j} is the first feature of the label ($y_{1j} = 0$ or $y_{1j} = 1$), $\tilde{p}_{1j}$ is the first feature of the prediction after the sigmoid, which maps its values to the interval (0,1), β is the weighting parameter, and N is the number of pixels. During development, it was found that a value $\beta = 30$ worked well for most data sets. This loss function grows toward infinity when $\tilde{p}_{1j}$ tends to 1 as $y_{ij} = 0$ or the other way around, which makes it suitable for binary image segmentation.

For the other features, the absolute error is used:

$$L_i = \sum_{j=0}^{N} |p_{ij} - y_{ij}|, \ i \in \{2, 3, 4, 5\}. \tag{15}$$

Crucially, the loss functions for the last four features are only calculated for pixels that belong to particles according to the label, not the prediction.

The final loss function is then calculated as a weighted sum of the loss functions for each feature:

$$L = \sum_{i=1}^{5} w_i \cdot L_i \tag{16}$$

Weights are then chosen to both prioritize certain features and normalise the scales of the different parts of the loss function [3].

After the loss function is formulated, the exact training method must be chosen. In the training examples of the original DeepTrack package, the training is set-up in a way where it starts with many iterations on small *batches*, which are the sets of images used in an iteration of the network's training. For each iteration, a new batch is generated. As the training progresses, the batch size is gradually increased, while the number of iterations for each batch size is decreased. To simplify testing and analysis, U-Track is instead trained with a fixed batch size. It is generally accepted that larger batches lead to more stable training [10]. Therefore, the largest batch size that fits into the GPU's memory is chosen, which in our case is 64. It is then trained for about 8000 iterations, with a new batch being generated for each iteration. The training takes about five hours[4], compared to around three hours for DeepTrack.

The hardest task for the network is prediction of images with low SNRs. A simple solution would be to only train using images with low SNRs, but this could lead to overfitting and decreased performance on high SNR images. To train mainly on images with low SNRs, but also some images with high SNRs, the SNRs of the images generated for each training batch are drawn out of an exponential distribution.

3.3 Post-processing

Performing meaningful analysis on the network's output requires post-processing in several steps, from identifying particles to sorting them into particle tracks and lastly performing statistical analysis on the resulting data. This process is made clear with the diagram in figure 5. In addition to creating tractable research data, the post-processing step allows for sorting out false positives from the network's output.

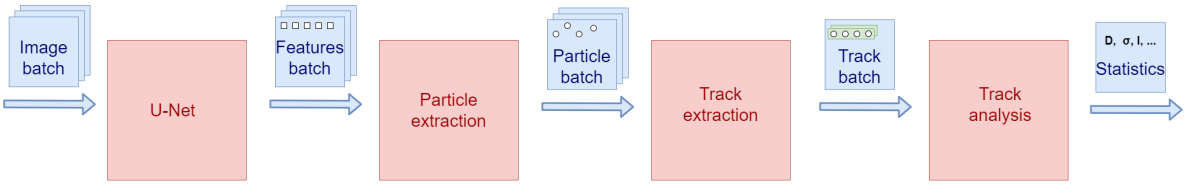


Figure 5: Data processing chain from input images to statistics, with blue squares representing forms of data and red squares representing processing steps. The post-processing begins with the extraction of particle detection information for each image. Particle detections are then linked into tracks and statistically analysed.

[3]For example, if the intensity can take values (0.1, 0.3) while the other features take values (1,3), $w_5 \approx 10$ might be chosen. However, if the quality of intensity prediction is not of interest to the user, w_5 might be lower.

[4]Computer specifications: Processor: Intel Core i7-8750H 2.20 GHz, RAM: 16GB DDR4 2400 MHz, GPU: NVIDIA GeForce GTX 1060 6 GB.

3.3.1 Extracting particle information from network output

The condensation from the pixel by pixel prediction down to information about a set of particles is done through the following process:

1. The first feature of the prediction is transformed into a probability prediction through application of a sigmoid.
2. The probability prediction is transformed into a binary map, with 1's representing pixels where a particle is predicted and 0's representing absence of a particle. This is done using a threshold called a *cutoff*, with all pixels with values larger than the cutoff being set to 1, and all others being set to 0.
3. A union-find algorithm groups clusters of ones in the binary map. Each such group is defined as a particle.
4. The particle center positions, radii and intensities are computed as averages over the remaining four features for pixels belonging to the particle.

Choosing the cutoff value is a step of vital influence for the rest of the processing, as it determines the balance between the inclusion of particles and allowing false positives to filter through. Optimization is utilized for finding the cutoff maximizing detection[5], but in practice this choice depends on the attributes of the input data and on the desired usage of the output. For example, one might accept more noise if the post processing is known to be robust against noise.

The approach of averaging values over clusters is a somewhat naive method, as the uncertainty of the particle attribute values then depend on the number of pixels per particle. It also does not handle cases of overlapping particles, treating such occurrences as one large particle. Still, overlapping particles were deemed to be of rare occurrence in the data used in this project, and should thus have little impact. It is however an important consideration for use cases with a higher particle density.

3.3.2 The particle tracking algorithm

Tracking particles over time is the next step in the post-processing pipeline. With little to no intrinsic information carried over between frames, which are predicted individually and under varying lighting conditions, the tracking algorithm is mostly based on the assumption that particle positions are sparse compared to particle movement between frames. This is a usually good assumption and is used previously in problems of similar type [11]. Additionally, if more information is known about the expected behaviour of the particles, the algorithm can be further modified to take this into consideration.

Connections between detected particles identified as the same physical particle detected in subsequent frames n and $n+1$ are formed inductively, with sets of particle detections $(\mathbf{x}_{n,j})_{j=1}^{J}$ already part of tracks and the new particle detections $(\mathbf{x}_{n+1,i})_{i=1}^{I}$ yet to be assigned to tracks. Note that the number of particles is different from frame to frame and not all particles should be linked, taking into account false positives as well as particles entering and exiting frames. A cost C_{ij} is then assigned for each pair of particles from both collections based on how well their connection follows the behaviour assumptions, resulting in a *cost matrix* C. The problem can now be formulated as the linear sum assignment of for each row i finding the corresponding column j such that the cost $\sum c_{ij}$ of pairs ij is minimised and is solvable through the Hungarian algorithm [12].

[5] See chapter 3.5.2 for more details.

The next step is handling cases of track ends and beginnings, which stem from the varying number of particle detections between frames as well as unfeasible connections. The Hungarian algorithm is able to solve non square matrices by not find pairings for all rows and columns. Beyond that, a stronger condition is applied, where pairings with cost $c_{ij} \geq c_{max}$ are discarded as well. Particles in $(\mathbf{x}_{n,j})_{j=1}^{J}$ without a good enough connection are then considered ends of tracks. Similarly, particles $(\mathbf{x}_{n+1,i})_{i=1}^{I}$ are beginnings. After this step, very short tracks are filtered as noise, as false positives typically only appear in a few frames. A choice has to be made of the minimal number of frames in a track for it to be regarded as a tracked particle.

3.3.3 Calculation of the cost matrix

The cost matrix (C) is a matrix with values in the range $[0, c_{max}]$ summed from terms C_k representing assumptions about particle behavior that can be evaluated by comparing particles $\mathbf{x}_{t,j}$ and $\mathbf{x}_{t+1,i}$. These terms should also be normalized to the same range as the cost matrix to work on the same comparable scale. In this project, the terms for the distance between particles C_D and the fluctuations in intensity C_I are used, but other factors can be imagined as well. This results in the equation

$$c_{ij} = \min\left(c_{D,ij} + c_{I,ij}, c_{max}\right), \tag{17}$$

where the minimum value is taken to ensure the maximum cost in C to be c_{max}.

As known from theory, particles are subjected to a laminar flow and are expected to move with a certain velocity $\mathbf{v}_{f,j}$. From that information the distance cost is made more precise by estimating this velocity, predicting the next position $\mathbf{x}_{t,j} + \Delta t \cdot \mathbf{v}_{f,j}$ and comparing it to the position $\mathbf{x}_{t+\Delta t,j}$ instead. A maximum accepted value M for this distance is determined by an initial tracking with M much greater than the actual maximum. The mean squared displacement (MSD) is then calculated for each track. MSD is given by

$$MSD = \frac{1}{\tilde{N} - 1} \sum_{t=1}^{\tilde{N}-1} (x_t - x_{t+1})^2, \tag{18}$$

where $\tilde{N}$ is the length of the track, which is the number of frames in the track. $\log(N)$ is plotted as a function of the MSD, as seen in figure 6, where N is the number of particles tracked with a certain MSD.

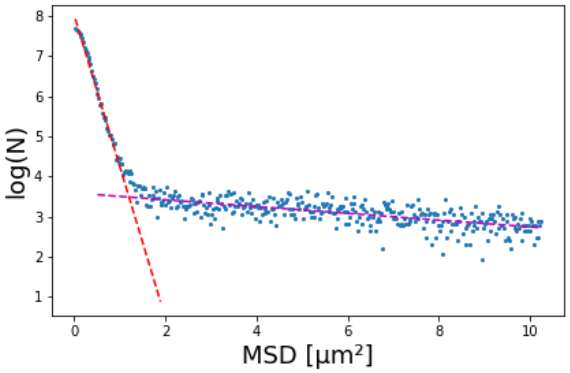


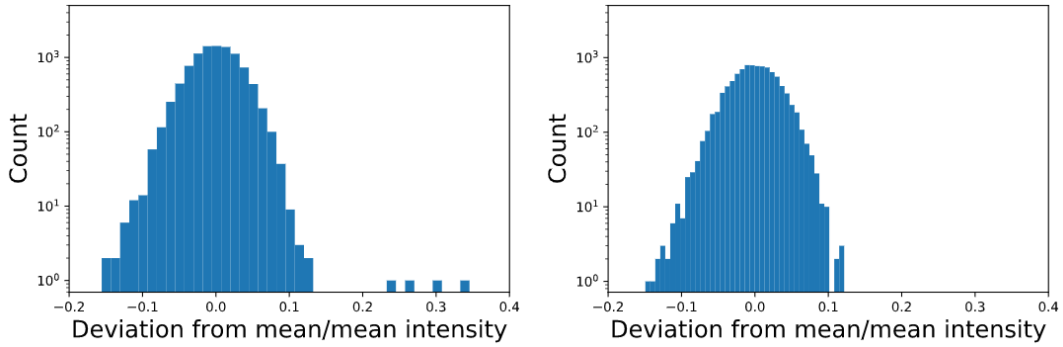
Figure 6: Plot of $\log(N)$ as a function of MSD, where N is the number of particles. Two curve fits are plotted for the tracks with low MSD (particles) and tracks with high MSD (noise). They intersect at $MSD = 1.2245\,\mu m^2$, which gives a max distance $M = 1.1066\,\mu m$.

By linear fitting of the values with lower MSD (consisting of tracked particles) and comparing it to a linear fit of the higher MSD values (consisting of noise), the cutoff distance M can be determined by calculating the MSD value of the intersection of these two curves and taking its square root [4]. For the video tracked in figure 6, this value is $1.1066\,\mu m$. Normalising the distance difference by M^2 results in the final distance cost

$$c_{D,ij} = \begin{cases} \frac{c_{max}}{M^2}(\mathbf{x}_i - \mathbf{r}_j - \mathbf{v}_{v,j})^2 & \text{if } (\mathbf{x}_i - \mathbf{r}_j - \mathbf{v}_{v,j})^2 < M^2 \\ c_{max} & \text{otherwise.} \end{cases} \tag{19}$$

The intensity term represents another way of using C, functioning as a hard filter. Because of the fluctuations in intensities they are unsuitable to use as a measure for likeness, but can instead be used as a sharp bound. As a rule of thumb, fluctuations over 15% of the mean intensity can instantly be filtered as tracking errors [4]. Referring to plot 7 confirms such occurrences in the tracks. Letting $I(x)$ represent the intensity of a particle, the intensity term is expressed as

$$c_{I,ij} = \begin{cases} 0 & \text{if } |\frac{I(x_{t,j}) - I(x_{t+1,i})}{\sum_{\tilde{t}=0}^{t} I(x_{\tilde{t},j})/N}| < 0.15 \\ c_{max} & \text{otherwise.} \end{cases} \tag{20}$$



(a) Without filter term C_I

(b) With filter term C_I

Figure 7: Histograms of particle intensity deviations from mean relative to mean before and after introduction of C_I in the cost matrix, logarithmised. The few terms to the right in (a) are considered falsely linked outliers which are removed in (b).

3.4 Evaluation of network performance

The evaluation of U-Track's performance is done on a pixel-by-pixel basis, since this is the most detailed scale of predictions and labels available. The following metrics were chosen[6]:

1. To evaluate the network's ability to detect particles (the first feature):
   - DULROC.
2. To evaluate U-Track's ability to predict particle attributes (other features)[7]:
   - Absolute error of position of the particle center.
   - Absolute error of radius prediction.
   - Absolute error of intensity prediction.

3.5 Benchmarking against DeepTrack

Evaluating U-Track on a pixel-by-pixel basis leads to complications when comparing performance against DeepTrack, since DeepTrack does not produce a prediction for each pixel. Therefore, a particle-by-particle approach is used when benchmarking.

The translation from pixel-by-pixel predictions to particle-by-particle is outlined in subsection 3.3.1. For the purpose of evaluating the ability to predict particle attributes, such a predicted particle is deemed a true positive if its center is within the radius of a true particle, otherwise it is deemed a false positive.

For the purpose of evaluating U-Track's ability to detect particles, a scanning box approach is used to calculate operating characteristics. Each image is broken down into overlapping smaller boxes roughly

[6] All measures are calculated as the mean of each metric for a batch of images and as a function of SNR.

[7] These metrics are only calculated for pixels belonging to particles, as calculating them for the background would be nonsensical.

the size of a large particle, placed a certain step size apart. Smaller step sizes result in better results, but longer computational times. Then, each of the scanning boxes yields operating characteristics based on the comparison between the number of true and the number of predicted particles within the box. Here, it is not demanded that the predicted particle centers fall within the radius of a true particle. It is rather assumed that a predicted particle is a true positive if there is a true particle within the same scanning box. It is also assumed that the grid of scanning boxes is fine enough to cancel the inaccuracies arising as a consequence of the first assumption. For single particle images though, the use of operating characteristics are assumed to be ineffective. The reasoning behind this is that since there is only one particle, the number of data points is too small for the operating characteristics to fairly describe the overall performance of the prediction. Therefore, precision is used for single particle images, with the hypothesis that precision should converge to 1 as SNR grows. Precision is calculated according to the previous paragraph.

To benchmark U-Track against DeepTrack, the following metrics were chosen[8]:

- Precision (for single particle images).
- DULROC (for images with multiple particles).
- Absolute error of position of particle centers.
- Calculation time for prediction.

3.5.1 Training

A new method for training the DeepTrack network was needed, as we found no ready way to train a network that could work with both multiple particle images and images containing no particles. To simulate this without changing the DeepTrack package, the particle position is chosen randomly from an interval bigger than the image size. To the best of our knowledge, this was also the approach of the original DeepTrack paper. The network is then trained according to the default training settings found in the DeepTrack repository. If the network has not converged by the end of the training, it is trained further on the biggest batch size. All in all, the training takes about three hours, which is slightly shorter than the time taken for U-Track's training.

3.5.2 Optimal parameters

Since the performance of both U-Track and DeepTrack depend on what parameters are fed to their post-processing methods, it is essential to optimize those parameters to get an accurate measure of the performance of the packages.

Optimization methods are implemented mainly using the *Nelder-Mead* optimization algorithm [13], with DULROC chosen as the function to be minimized[9]. The methods for calculating optimal parameters are quite slow, taking roughly 1 hour both for optimizing U-Track's cutoff using 400 images and for optimizing DeepTrack's parameters using 40 images on a mid-range laptop [10]. These computation times makes it unfeasible to use a larger amounts of images per optimization run. There arises a risk

[8] All measures are calculated as the mean of each metric for a batch of images and as a function of SNR.

[9] For optimizing the cutoff used in U-Track, *Nelder-Mead* is used to generated an initial guess. It is then followed by a check that the guess is within the allowed interval. If it is not, then the initial guess is set to a predetermined cutoff. In both cases, the initial guess is fed to a bounded optimization using the *L-BFGS-B* algorithm.

[10] Computer specifications: Processor: Intel Core i7-9750H 2.60 GHz, RAM: 8GB DDR4 2400 MHz, GPU: NVIDIA GeForce GTX 1050 3 GB.

from using a small batchsize and that is that the optimization simply might not find the true optimum.

The optimization methods are evaluated by plotting the value of the DULROC using parameter values around the ones suggested by the optimization. For U-Track, only the cutoff is optimized, so these graphs show a relatively detailed picture of the optimality of the calculated cutoff. For DeepTrack however, there are two parameters being optimized, and thus the analysis is more complicated. The method used here to investigate optimality is to vary one of the parameters at a time, with the other one being kept constant. This can be seen as the parameters only being varied in two distinct directions on the plane that their possible values span, which is not enough to assume optimality, but it can be seen a rough check. Using these "tests" of convergence, the optimization methods were deemed functional when using the number of images mentioned in the paragraph above. It should be kept in mind, though, that the values suggested by the methods are not proven to be optimal.

3.6 Evaluation of particle tracks on experimental data

The network and post-processing are also tested by tracking on experimental data. The evaluation of this is made difficult due to the absence of information about the ground truth. This chapter explains how this is handled through qualitative comparison with a previously used tracking algorithm and reasoning around the consistency with theoretical results.

3.6.1 Comparison with a digital filter based approach

An alternative tracking script not based on machine learning was generously provided by the authors of [11] for a rough comparison of tracking abilities. The script differs in most steps of the process:

- Input data is processed through digital filters to extract light particles on a black background, as exemplified in figure 8.

Figure 8: An example image processed through the benchmarking script, visualising the function of digital filtering.

- Particle centers are computed by a cutting gradient algorithm [14].
- Tracking is done with a similar algorithm to the one developed in subsection 3.3.2, although allowing links formed with several frames in between and discarding short tracks.

- Intensity is calculated as the sum of pixel values from the image which differs from the intensity value predicted by U-Track.

The script is also mostly suitable for data with high SNR and is not to be interpreted as the best possible alternative, especially as results might be improved through parameter tweaking. To fairly compare track lengths, they are compared both before and after an unlinking of tracks from the benchmark which contains gaps.

3.6.2 Bias of intensity and pixel size

One desirable property of the tracking is equal detection performance regardless of particle intensity and size. The absence of this may result in particle distributions being skewed toward particles of high intensity and large size. The method chosen for evaluating potential bias is studying how the track average of the particle intensity and size varies with the number of frames in the track. A mean increasing with track length would imply that particles of higher intensity and size are more easily tracked. This conclusion is based on the assumption that in reality, most particle tracks should be of about the same length. The real distribution of intensity and size is unknown, ruling out a direct comparison between detected particles and the expected distribution.

3.6.3 Consistency with the diffusion hypothesis

The diffusion hypothesis is used for evaluation in two different ways. Firstly, coming from a real use case, it has to be confirmed whether particles really are subject to diffusion. It is then important that U-Track is able to produce data which converges to the expected results from the periodogram theory developed in section 2.3.3. Comparison between the expected and measured periodogram is done quantitatively through Pearson's chi-squared test by binning the computed values of $\check{P}_k$, counting the quantities in each bin O_i and comparing to the expected number E_i by calculating

$$\chi^2 = \sum_{i=1}^{N} \frac{(O_i - E_i)^2}{E_i^2}. \tag{21}$$

The null hypothesis of likeness between the two distributions is then rejected with significance α if this value exceeds the α value of the χ^2 distribution with $N-3$ degrees of freedom, and otherwise nothing can be said for singular tests [8].

The next step is to use testing of diffusion as a metric of individual track quality, that is the closeness to a true particle track. The idea is that even though short single tracks do not provide enough basis for confirming diffusion, true tracks should at least not break expected behavior. Covariances with large deviations from zero then imply a track which does not behave as diffusion, indicating that the track is faulty. Similarly to the first method, the likeness of the distribution of covariances to a normal distribution around zero with variance given by theory is quantified through a statistical test, this time using Pearson's test of variance

$$\chi^2 = \sum_{i=1}^{n} \frac{x_i - E[x_i]}{\mathrm{var}(x_i)}. \tag{22}$$

3.7 Image generation

Training the network requires a large quantity of simulated images and for DeepTrack, this generation of images is often the single most calculation heavy part of training the network.

DeepTrack's image generation routine consists of looping over the parameters for each particle and calculating the value of an adaptation of the Bessel function for each element in the matrix representing the image to be generated. The Bessel function can be of different orders, in this report often called *Bessel orders*, but in this paper order 1 is primarily used, which simulates particles with their maximum intensity at the particle center and with an absolute intensity converging to 0 as function of the distance from particle center. Order 2 of the Bessel function could however also be useful, as it simulates a dark particle with an intense outer radius.

To speed up the image generation, three different approaches were considered:

- Near the center of the particle, the adaptation of the Bessel function of order 1 has a similar shape to Gauss function, likeness visualized in figure 9, and since the Gauss function has a less complicated formula, it might be cheaper to calculate. Therefore, since Bessel order 1 is the most relevant, it could be worth removing the ability to use other orders of the Bessel function to instead use the (potentially) cheaper to calculate Gauss function.

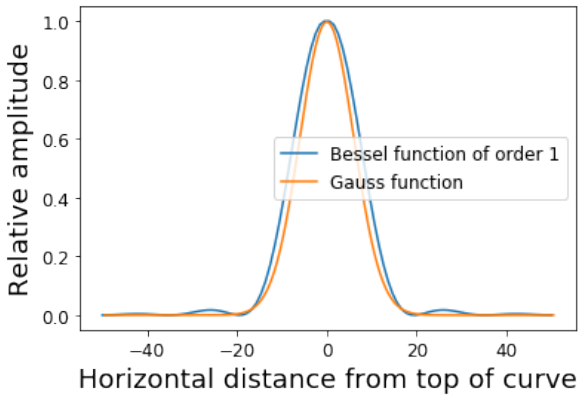


Figure 9: Comparison between a Gaussian function and an adaptation of Bessel function of order 1.

- Because the values of the adaptation of the Bessel function converge to zero relatively quickly as function of distance to particle center, the impact on image quality from only calculating the Bessel function for elements relatively close to the particle center in question might be negligible. Therefore, calculation time might be able to be cut by in this way limiting the number of calculations made.
- The values of the adaptation of the Bessel function are calculated for each element in the image matrix. Since these calculations are many but individually relatively cheap, the process could be sped up by using the GPU instead of the CPU, as GPUs generally are faster at performing large quantities of smaller calculations.

4 Results

The structure of the results chapter corresponds to the last 4 subchapters of the method chapter. The subchapters generally consist of a short summary of the results, followed by graphs visualizing them and short graph descriptions.

4.1 Evaluation of network performance

The evaluation of U-Track's performance starts with three measures, as shown in figures 11 and 12, that showcases the network's ability to detect particles. These three measures all tend towards satisfactory values. The evaluation goes on to show the network's ability to predict particle attributes. The network tends towards yielding predictions with small but significant mean errors. This is shown in figures 13 and 14.

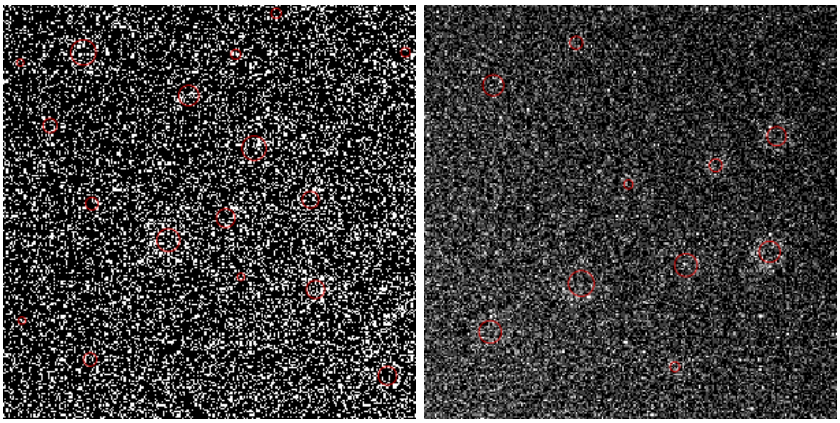
For each SNR, the network is evaluated on batches of 1000 images with the following parameters:

Parameter	**Value**
Image size	256 by 256 px
Particle number	10 - 20
Particle radii	2 - 8 px
Particle intensities	0.1 - 0.3
Bessel orders	1 or 2
Gradient intensities	0.1 - 0.3
Background intensities	0.0 - 0.4
Noise	Poisson

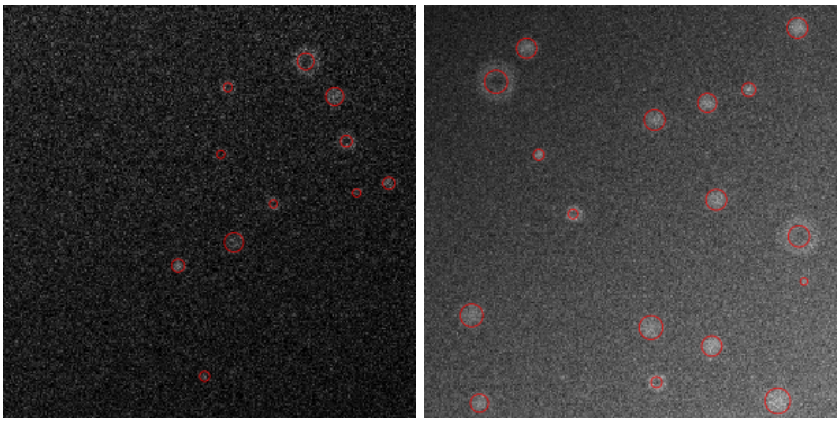
Table 1: The image parameters and their respective values used when evaluating U-Track's performance.

The weights of the loss function are mainly chosen through testing. To begin with, $w_1 = 30$ is chosen, as lower values often lead to the network not predicting any particles, while higher values sometimes cause too many predictions. The other weights are chosen so that their respective losses typically take values in the same interval, with $w_2 = 1, w_3 = 1, w_4 = 1, w_5 = 5$. Thus intuitively, $w_5 = 10$ might have been chosen, but at the time of the choice it was deemed that the intensity prediction was of lower priority than the predictions of positions and radii.

Example images at chosen SNRs are shown in figure 10 below. Thereafter, the results of the network evaluation outlined in section 3.4 are shown.



(a) SNR = 1 (b) SNR = 2



(c) SNR = 5 (d) SNR = 10

Figure 10: Examples of simulated images on which the network is trained, evaluated and benchmarked against DeepTrack at chosen SNRs (1, 2, 5, 10). The red circles denote particles in the image. The radius of the red circle is the same as the radius of the particle.

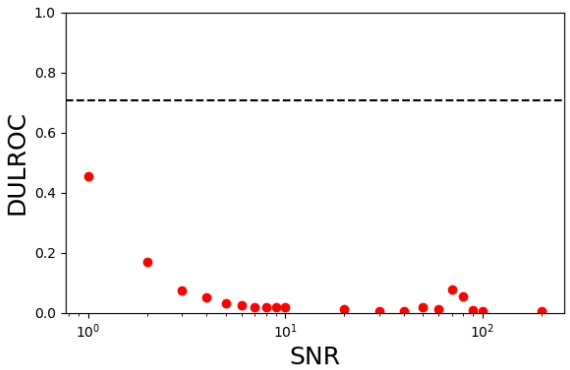
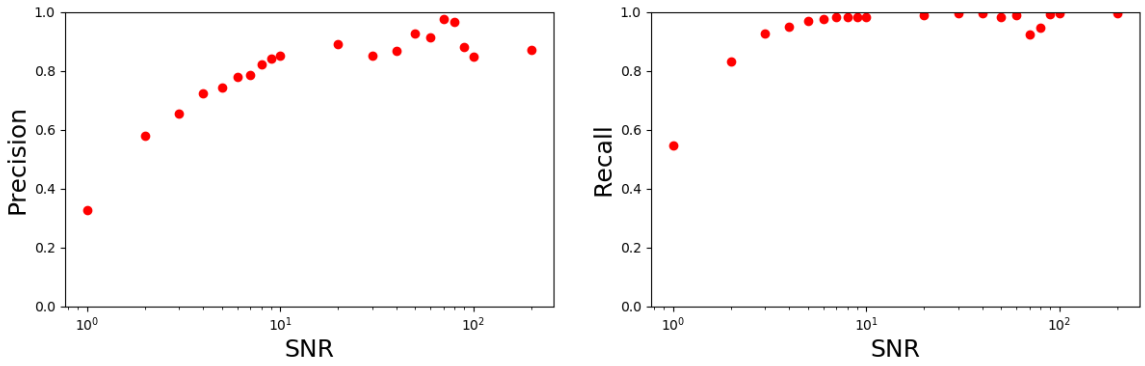


Figure 11: Pixel-by-pixel DULROC for U-Track as a function of SNR. The dotted line is the minimal DULROC expected from random predictions and 1 is the maximal DULROC expected from random predictions[11].



(a) (b)

Figure 12: (a). Precision of U-Track as function of SNR. (b). Recall of U-Track as function of SNR. Both the precision and recall increase heavily until they reach a maximum at around SNR = 10, after which they stay roughly constant.

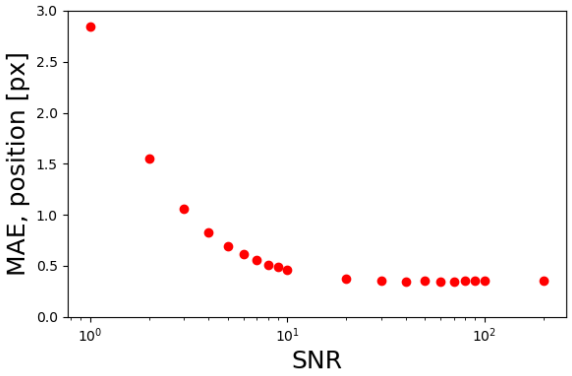
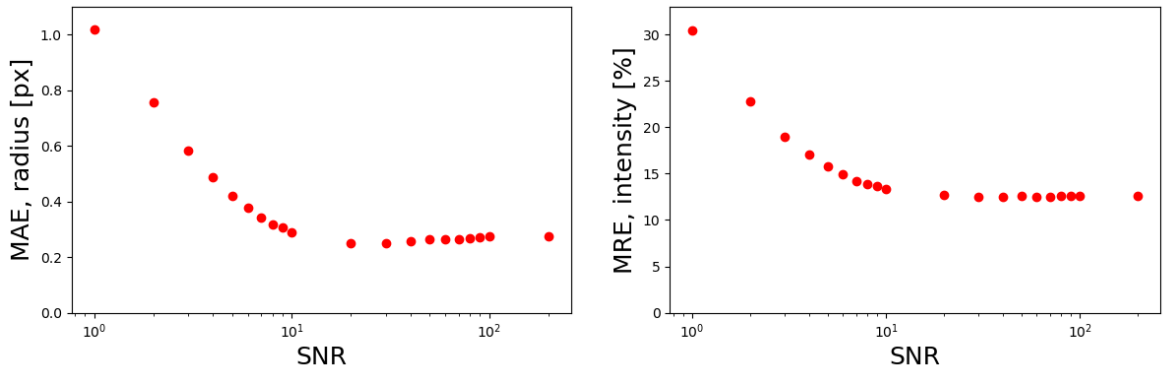


Figure 13: MAE of particle center predictions for U-Track as a function of SNR. This is calculated on a pixel-to-pixel basis for pixels that belong to a particle in the label. A similar pattern to the one seen in figure 12 is found.

[11] Random predictions are expected to fall on the diagonal $(0, 0) \rightarrow (1, 1)$ in the ROC curve.



(a) (b)

Figure 14: (a). MAE of radii for U-Track as function of SNR. (b). MAE of intensities for U-Track as function of SNR. A downward trend is seen until around SNR = 10. Interestingly, a slightly upward trend seems to exist for higher SNRs in the prediction of the radii.

Figures 11 and 12 show that, as expected, U-Track's performance is poor for SNR = 1 (although still significantly better than a random guess). It then improves greatly up until around SNR = 10, after which it stays roughly constant. The fluctuations of performance for larger SNRs are thought to be the consequence of suboptimal parameters, as elaborated on in section 5.1.

The same trend can be seen in graphs 13 and 14, with performance improving rapidly until SNR = 10 and then staying roughly constant. Interestingly, there are little fluctuations but U-Track's performance seems to decrease slightly for SNRs over 12, which is especially visible for the prediction of radii. This could be a sign of overfitting, as the network is trained on images with a mean SNR of 10.

4.2 Benchmarking against DeepTrack

The benchmarking against deeptrack consists of two main parts; one using only images containing a single particle and one using images containing multiple particles. For single particle images, U-Track consistently achieves a lower MAE for predicting the position of particles, as shown in figure 15. The results for precision and calculation time, shown in figure 16, are more varied, as the difference between the packages' performance grows from being small for small SNRs to being considerably in the favor of U-Track for large SNRs.

For multiparticle images, the results regarding detection and prediction of position are similar to the result for single particle images. This is seen in figures 17 and 18. Similarly as for single particle images, U-Track is much faster for large images, whereas DeepTrack is faster for small images, as seen in figure 19.

4.2.1 Single particle

Just as in section 4.1, for each SNR, the networks were tested on batches of 1000 images. The image parameters used are shown in table 2. The results of the benchmarking are shown in figures 15 and 16.

Parameter	Value
Image size	51 by 51 px
Particle number	1
Particle radii	2-8 px
Particle intensities	0.1-0.3
Bessel orders	1 or 2
Gradient intensities	0.1-0.3
Background intensities	0.0 - 0.4
Noise	Poisson

Table 2: The image parameters and their respective values used when benchmarking U-Track against DeepTrack on single particle images.

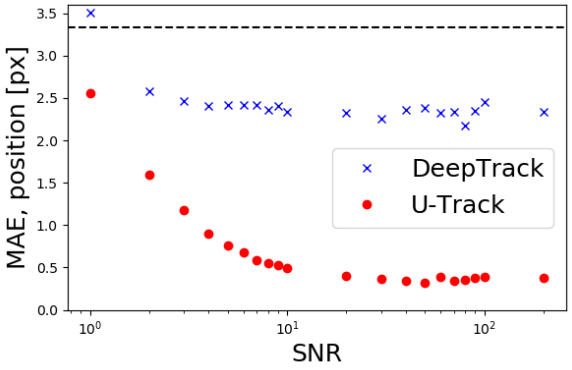
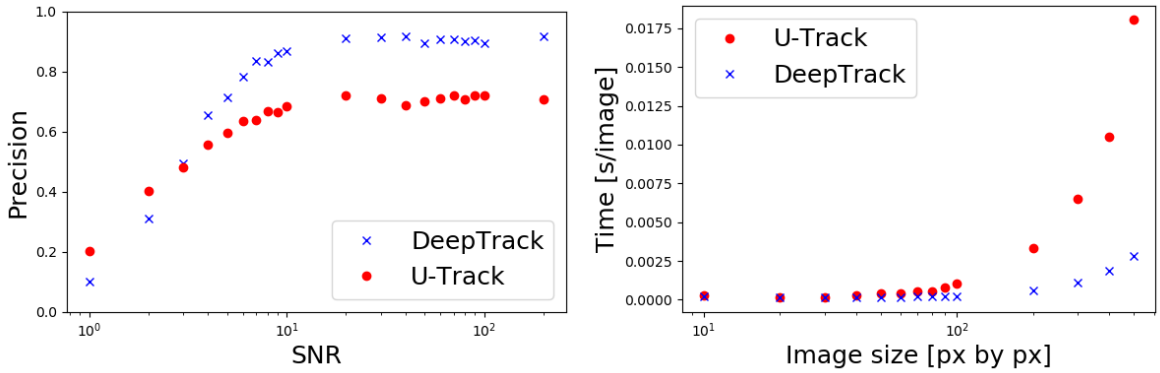


Figure 15: MAE of particle center predictions for U-Track and DeepTrack as a function of SNR for single particle images. The dotted line is the expected MAE for random predictions, as only predictions within a particle are counted.



(a) Precision as function of SNR.

(b) Calculation time per image.

Figure 16: Precision as function of SNR and calculation time as function of image size for U-Track and Deep-Track. The precision converges to almost 1 for DeepTrack, but not for U-Track. U-Track is faster for small images, but not for large ones.

As seen in figure 15, U-Track outperforms DeepTrack for all SNRs and converges to a level of roughly one fifth of the one DeepTrack converges to.

For the precision measured in figure 16, U-Track performs better than DeepTrack for very low SNR, but then converges to a level roughly 20% worse than that of DeepTrack. This is a symptom of U-Track being more prone to predict too high a number of particles, which DeepTrack cannot do. Figure 16 also shows that DeepTrack is slightly faster for smaller images and proceeds to being much faster for larger images.

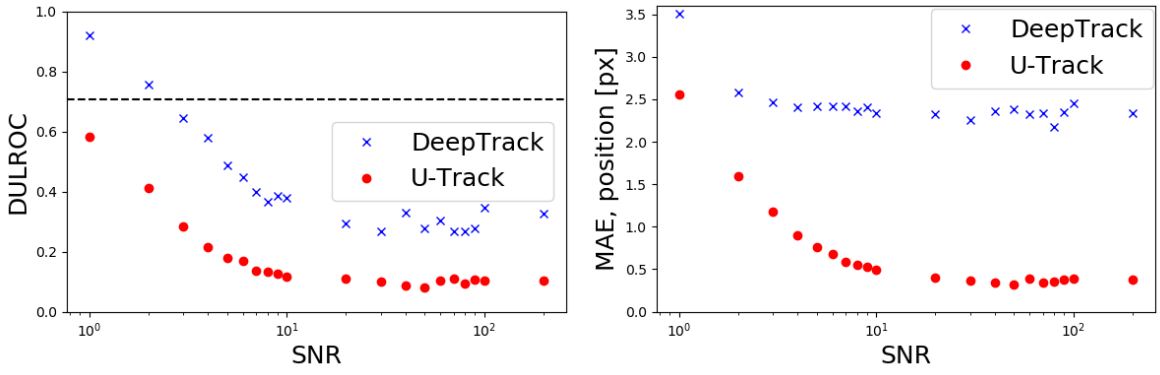
4.2.2 Multiple particles

Unlike for single particle images, the batch sizes used for multiparticle images are only 100. This decrease is made to lower overall computation time. The image parameters used were:

Parameter	Value
Image size	256 by 256 px
Particle number	10-20
Particle radii	2-8 px
Particle intensities	0.1-0.3
Bessel orders	1 or 2
Gradient intensities	0.1-0.3
Background intensities	0.0 - 0.4
Noise	Poisson

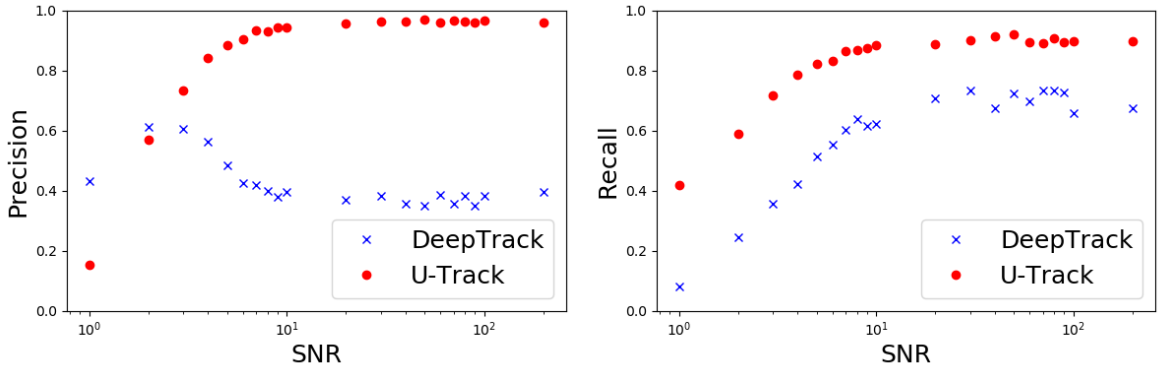
Table 3: The image parameters and their respective values used when benchmarking U-Track against DeepTrack on multiple particle images.

The results of the benchmarking are shown in the following figures:



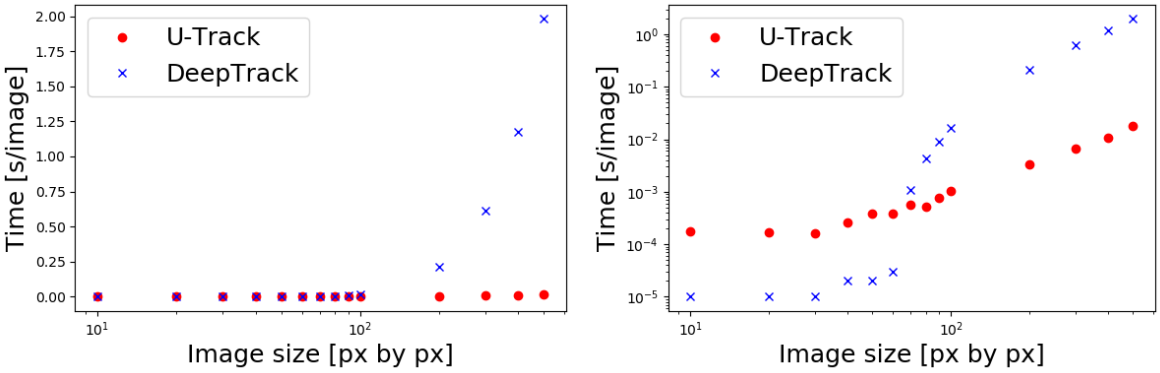
(a) (b)

Figure 17: (a). DULROC of U-Track and DeepTrack as functions of SNR for multiple particle images. The black dotted line is the minimal DULROC expected from random predictions, see figure 11. (b). Particle center position MAE for U-Track and Deeptrack as function of SNR for multiparticle images.



(a) (b)

Figure 18: (a). Precision for U-Track and DeepTrack as functions of SNR for multiple particle images. (b). Recall for U-Track and DeepTrack as functions of SNR for multiple particle images.



(a) (b)

Figure 19: (a). Calculation time per image for U-Track and DeepTrack as functions of image size for multiple particle-images. (b) is the logged results from (a). This is done to obtain a more nuanced picture of the packages' efficiency compared to each other rather than in general.

In figure 17 it is shown that for both mean DULROC and particle center position MAE, U-Track perform better than DeepTrack for practically all SNR. In particular, the DULROC of U-Track's predictions coverges to a level of rougly a sixth of DeepTrack's and the positional error of U-Track's predictions converges to a level of roughly a fourth of that obtained by DeepTrack.

As a more nuanced complement to the DULROC, the graphs of precision and recall in figure 18 shows that U-Track generally outperform DeepTrack in not only the ratio of predictions that are correct but also in the number of particles that are detected.

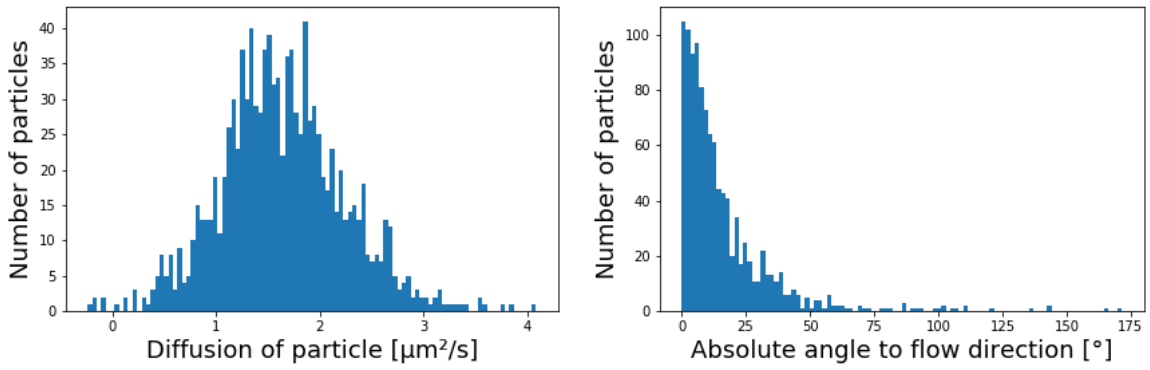
In figure 19, the calculation times for DeepTrack exceeds U-Track's for images larger than around 80 by 80 px. The difference for small images is small per image, but in the event of running predictions on a vast number of small images, it might be non-negligible.

4.3 Tracking on experimental data

The analysis of the tracking on experimental data is split into three parts. In the first section, the results from calculations of scientific interest are examined and compared to expected results. Following this is the distribution of track lengths together with an overview of eventual biases towards tracking particles of different intensities, and in the last section the quality of tracks are measured through their agreement with the diffusion hypothesis. When relevant, results are compared to a digital filtering based algorithm as a reference.

4.3.1 Tracking results

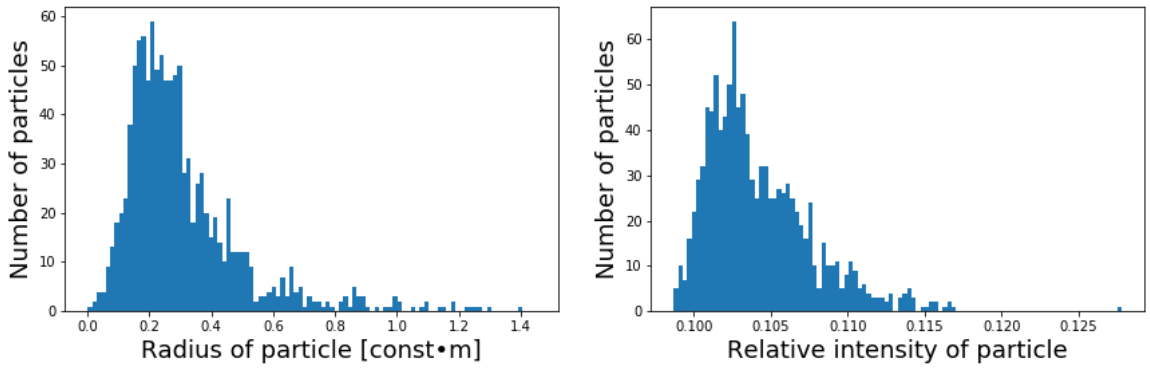
For the results presented in this section a total of 10 000 frames were tracked, and only tracks containing 20 or more frames were included for analysis. The diffusion of the tracked particles is estimated with equation 8 and plotted in figure 20a. The median diffusion and the mean diffusion are 1.61 µm^2/s and 1.64 µm^2/s respectively. Particle radius is calculated approximately with equation 4 and is plotted in figure 21a. The relative intensity distribution is shown in figure 21b. The tracked particles are expected to follow a constant flow direction. This direction is calculated from the tracked particles, and each particle velocity is compared to this flow, as shown in figure 20b. The median particle direction differ 10.2° from the laminar flow direction.



(a) (b)

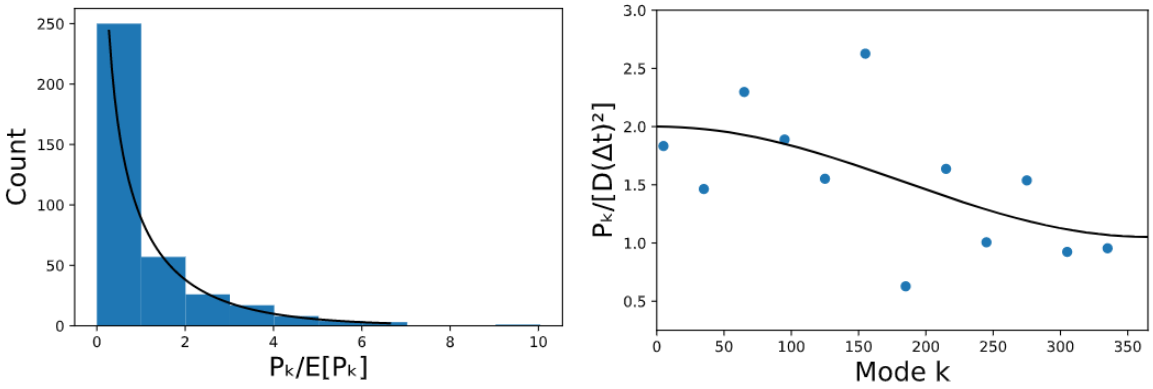
Figure 20: (a) shows a histogram visualizing estimated diffusion for tracked particles, with length of minimum 20 frames. The median value is 1.61 µm^2/s. In figure (b), the direction of particle velocities are compared to the total flow direction off the tracked video. The angular difference is plotted, with median value 10.2°.

Statistical tests of the diffusion hypothesis can be seen in figure 22, performed on the longest continuous track available. Some similarity to the theoretical values is seen, but ultimately not enough to say anything for certain, with a p-value of 0.5 from Pearson's goodness of fit test. The number of data points were also much fewer than the recommended amount of 1000 from the original article as no other tracks with the same diffusion coefficient and σ were found to create an average from.



(a) (b)

Figure 21: (a) shows estimated radius of tracked particles. Radius is calculated approximately with equation 4, and is given up to a constant. In figure (b), relative intensity of tracked particles is plotted.



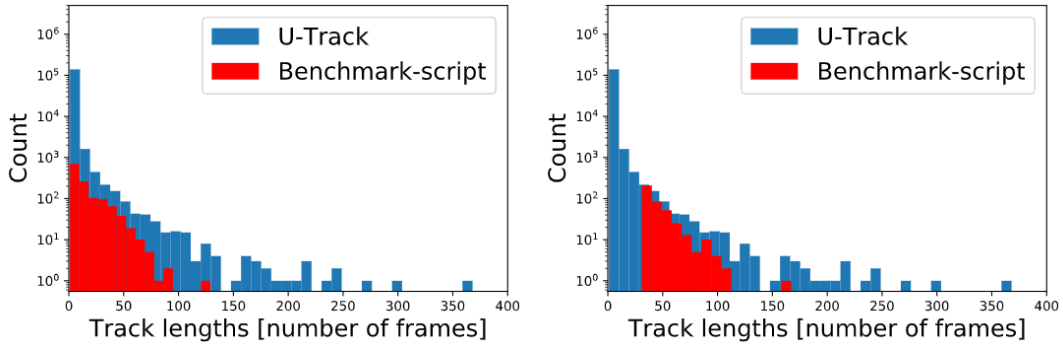
(a) Histogram of $\check{P}_k$ relative to expected P_k

(b) Sampled periodogram of $\check{P}_k$

Figure 22: Two visualisations of transformed displacements for a track of length 356 compared to expected results for diffusion (in black). The shape of the data makes it difficult to draw definite conclusions on whether the track describes diffusion.

4.3.2 Track length

In absolute numbers, U-Track finds 1029 tracks above length 20, compared to the 405 found by the benchmark script. When tracks with gaps from the benchmark are split the number of such tracks fall to 306. The distributions of track lengths can be seen in figure 23, from which it is clear that U-Track also manages to find longer tracks.

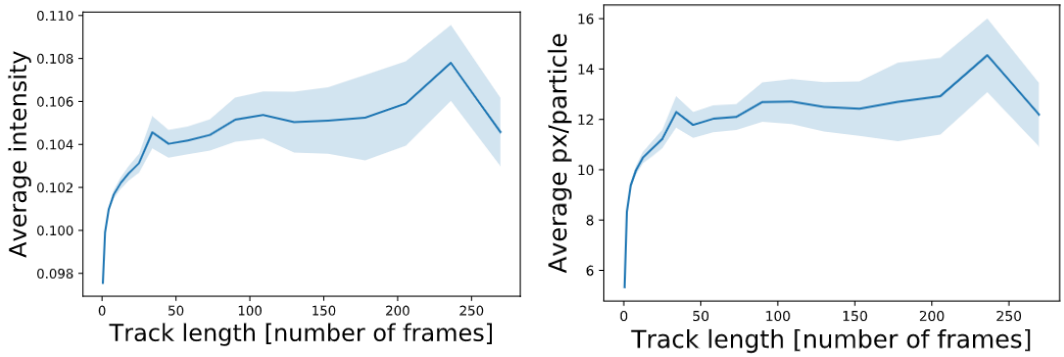


(a) Tracks without gaps

(b) Tracks with gaps

Figure 23: Track length distributions where U-Track is compared to the benchmark script. Note that U-Track find more tracks for all lengths.

From figure 24, there are several interesting observations to be made. Most striking is the clear resemblance between the graphs of average intensity and average number of pixels per particle. Calculating the correlation between the two values for individual tracks yields a coefficient of 0.67, generally considered a strong correlation, which confirms this likeness. Both graphs also follow a clear pattern of two different behaviours, a sharply increasing mean up to a breaking point around twenty frames in, beyond which only a slight positive slope may be distinguished. This slope does not indicate any significant bias. It should finally be pointed out that tracks below twenty frames have significantly many more outliers also of high intensity and pixel size, which is not shown by the graph.



(a)

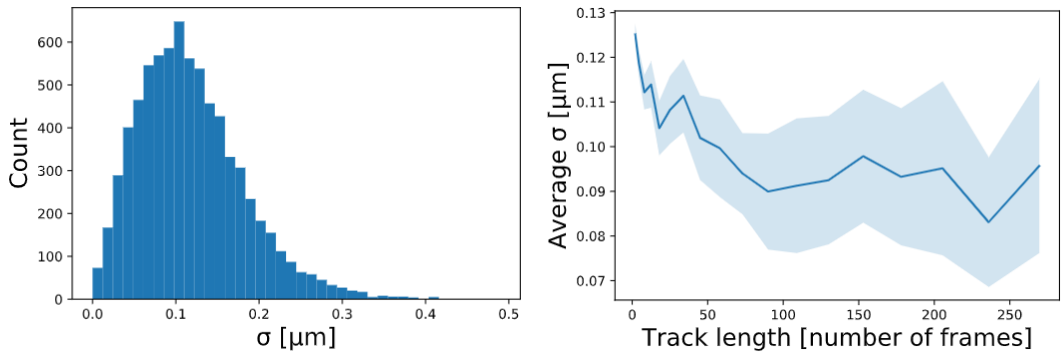
(b)

Figure 24: The filled area indicates a 95% confidence interval, where the darker line indicates the mean value. In (a) the average intensity is shown for different track lengths, where a short track tends to have smaller intensities. In (b), the y-axis instead shows pixel size of the first feature. The plots are close to identical, which is surprising.

4.3.3 Position accuracy

The estimated position error σ is mostly distributed around 0.1 µm, decreasing noticeably between track lengths 0 to 100 frames, referring to figure 25. This result is comparable to slightly better than the benchmark script which does not have as many long tracks to analyse.

Pearson's test of variance with a significance level of $\alpha = 0.99$ keeps the null hypothesis for 98.9% of all tracks, which is very close to the expected result of 99% would the values be drawn from an actual normal distribution. For the benchmark script, the corresponding value was 99.4% and as a comparison with a non diffusing direction, a test of diffusion in direction $\mathbf{v}_f$ rejects the null hypothesis in 50% of cases.



(a) (b)

Figure 25: The plots visualizes the estimated localization error σ for tracks. In (a) the distribution of σ is shown, where count is the number of tracks in each interval, with mean 0.1 µm. (b) shows correlation between track length and average σ, where the filled area is a a 95% confidence interval. Longer tracks tends to have a smaller position error σ.

4.4 Image generation routines

Three improved algorithms for image generation were developed:

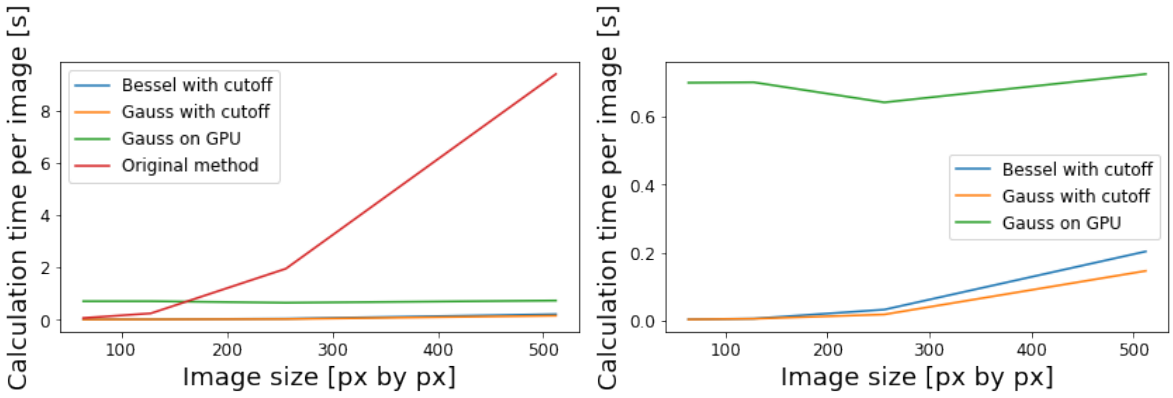
- The original DeepTrack algorithm but with the Bessel function only being calculated for elements near particle centers, defined as elements within a distance of three particle radii in either x- or y-direction. This threshold of distance from particle center is called a *cutoff*, not to be confused with the cutoff used when extracting particle information, introduced in chapter 3.3.1.
- The same as above but calculating the Gauss function instead of the Bessel function.
- Calculating the Gauss function using the GPU. This is done without a cutoff since the bottleneck of this approach is the amount of data being sent to the GPU and using a cutoff either means sending a larger number of matrices to the GPU or a larger amount of array accesses.

The image parameters used were:

Parameter	Value
Image size	128 by 128 px (when not varied)
Particle number	1-6 (when not varied)
Particle radii	1-3 px
Particle intensities	0.5-0.7
Bessel orders	1 (if used)
Gradient intensities	0.25-0.75
Background intensities	0.3- 0.5
Noise	Poisson
SNR	2-10

Table 4: The image parameters and their respective values used when measuring speed of different image generation routines.

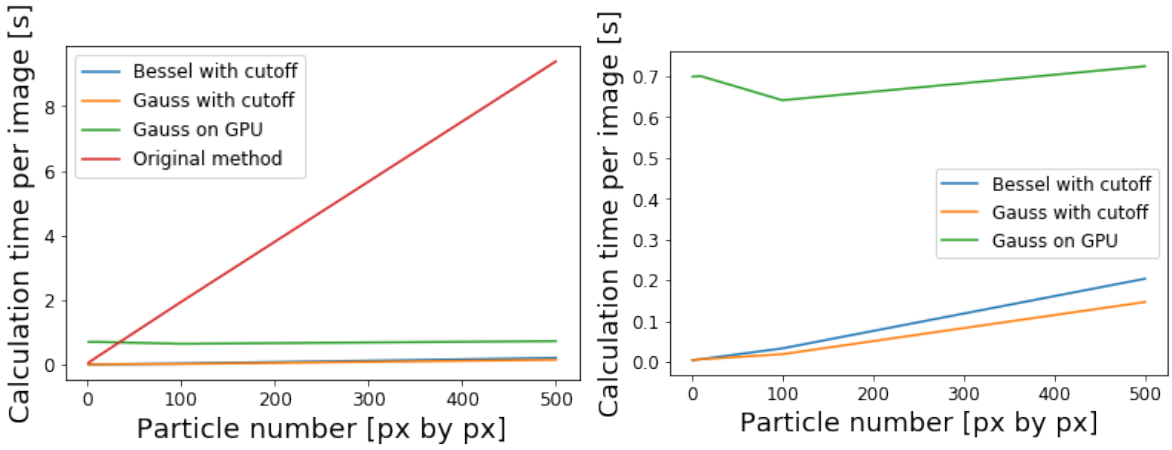
The reason for choosing these values is that they more closely resemble the ones used by DeepTrack than the ones used in the rest of this report. The calculation times per image for the different algorithms were measured as below[12]:



(a) (b)

Figure 26: Calculation times as function of image size. (a). All the new image generation routines are included and the graph dispays that the old routine is the slowest. (b). The same figure as (a) but with the plot of the original routine excluded. The graphs suggest that the routines implementing a cutoff are the most time efficient.

[12]Computer specifications: Processor: Intel Core i7-9750H 2.60 Hz, RAM: 8GB DDR4 2400 MHz, GPU: NVIDIA GeForce GTX 1050 3 GB.



(a) All routines included.

(b) Original routine excluded.

Figure 27: Calculation times as function of number of particles. (a). All the new image generation routines are included and the graph displays that the old routine is the slowest. (b). The same figure as (a) but with the plot of the original routine excluded. The graphs suggest that the routines implementing a cutoff are the most time efficient.

It is clear that all new routines outperform the original one and that the routines implementing a cutoff on distance from particle center are the most efficient.

5 Discussion

The chapter of discussion contains the same subchapters as the results chapter and the last part of the method chapter. This is because the considerations elaborated on here generally arise from the thought of *"If a different choice had been made during the construction of the method, how would the results be affected?"*. The matters handled in this chapter are thus often based on details from the method or theory chapters, but are approached from a point of view determined by the results chapter.

5.1 Evaluation of network performance

The choice of evaluating performance of position, radius and intensity prediction without taking into account whether U-Track actually predicts that the pixel belongs to a particle can lead to misleading results. For example, assume that the probability of a pixel being a true positive decreases with the distance from the center of the particle while the radius prediction performance increases. Then U-Track's radius prediction might perform significantly worse when applied to real data compared to our evaluation.

Another cause for concern is our definition of the SNR, which we inherited directly from DeepTrack. The SNR is defined as:

$$SNR = \frac{\sqrt{\mu}}{\sigma}, \tag{23}$$

where μ is the mean of the pixel intensity and σ is its standard deviation. In image processing, the SNR is usually defined as $\frac{\mu}{\sigma}$ [15]. This makes the evaluation difficult to compare to other projects. Example images for chosen SNRs are shown in figure 10, but this only serves as a way to get a feeling for the

network's performance and cannot be used for a quantitative analysis. A simple conversion algorithm could be to multiply our SNR by $\sqrt{\mu}$. However, because of the gradient and particle intensities, this is a highly approximate method. Our evaluation has a mean $\mu \approx 0.2$, so our "breaking point" around SNR = 10 would then be converted to $\text{SNR}_{converted} \approx 5$.

As mentioned in chapter 3.5.2, the performance of both U-Track and DeepTrack depend greatly on the choice of their parameters and also, the optimization methods used for optimizing said parameters are not proven to always reach an optimum. Both the dependence and suboptimality likely manifests themselves in the results of the evaluation of network performance as well as in the results of the benchmarking against DeepTrack. More precisely, the fluctuations in figures 11 and 12 could be the results of the optimization method failing to converge towards the optimum. An argument for this hypothesis is that the data points in general follow a relatively clear pattern and that these fluctuations appear to be somewhat erratic and of inconsistent.

5.2 Benchmarking against DeepTrack

There were several choices made when designing the benchmarking process. To eliminate subjectivity in the decision making process, some general guidelines were formulated:

- Center the benchmarking around images containing multiple particles. This is the image type that the project focuses on, but it is not the primary focus of DeepTrack.
- Do not modify the DeepTrack package in any way when using it, as it is highly subjective where the limit of reasonable modification is.
- Use the methods as described in the original DeepTrack article or the examples provided in the DeepTrack git repository. This is both to make our tests as optimal as possible, and to save effort on designing the testing methods for DeepTrack.
- When possible, design the tests around the way DeepTrack outputs data.

These guidelines should be kept in mind when analyzing the results presented in this report, and especially when comparing them to the ones being presented in the original DeepTrack paper.

At first glance, it might seem that U-Track performs better at all cases of particle tracking, even on single particle images. This is not the correct conclusion as the single particle images used in this project are different from the ones used in the original DeepTrack paper. In the original paper the particles are always positioned near the center of the images while in our project, the particle positions are uniformly distributed in the images. This difference naturally results in a worse performance for DeepTrack which is not primarily designed for these types of images. It should be noted that DeepTrack might perform better than U-Track (and be faster) on the image type that it was originally intended for.

As stated earlier, the primary goal of this project is tracking multiple particle images, so it is not obvious that benchmarking on single particle images is needed at all. However, this part of the testing allows a more direct comparison between the networks, as it does not include DeepTrack's parameters.

In the benchmarking of multiple particles, those parameters cannot be eliminated. The tests were made as objective as possible by algorithmically optimizing DeepTrack's parameters. However, none of these parameters are proved to be optimal, so it is possible that DeepTrack's performance could be made better by further tweaking. Furthermore, a slight modification[13] of the DeepTrack package

[13]The current training routine was slightly modified, where instead of simulating images without particles by placing the particles outside of the image's borders, we simply did not place out any particle at all. The network was then

was tried, and this showed a clear improvement in performance. However, significant changes to the DeepTrack package would most likely have to be made to improve the performance to the point where it would be comparable to U-Track's on multiple particle images.

The parameters of DeepTrack discussed up until this point are only the ones used in the post-processing. Before that, there are a number of other values that have to be decided. They include the scanning box size and step, as well as the number of layers in the network and their dimensions. This means that an objective comparison between packages is hard to make, as tweaking of a parameter might lead to better results. However, the selection of parameters was made according to the examples provided in the git repository, which suggests that they should be approximately optimal.

As the two packages do not have the same direct output, the methods for translating their respective output to a common form might be prone to bias, as such methods might ignore nuances in the original output or infer new nuances not previously present. The methods used in this project, perhaps most significantly the routine outlined in section 3.3.1 and the scanning box method presented in section 3.5, aim to angle said bias in favour of DeepTrack by mostly being implemented on the output from U-Track (in the case of clustering pixels) and also being structured in a way similar to the output from DeepTrack (in the case of the scanning boxes from section 3.5).

It should also be noted that an important goal for this project was improving the network's computational efficiency for multiple particle images. Regardless of parameter choice or slight package modifications this has been achieved, as showcased in figure 19, with U-Track being around 100 times faster for images larger than 100 by 100 px.

5.3 Tracking on experimental data

The discussion of tracking first relates to the methods and results on diffusion, intensity, track lengths and bias, followed by details on specific parts of the pipeline and possible improvements.

5.3.1 Plausibility of results

The measured diffusion median value of $1.61\,\mu m^2/s$ is within the expected interval from the experiment, since some particles can be tethered by more than one DNA, see section 2.3.2. Thus the spread in measured diffusion values shown in figure 20a is because of a different number of tethered DNA to each particle, but is also influenced by statistical fluctuations of shorter tracks, and position errors from the network output. Diffusion was estimated in a direction perpendicular to the direction of the total flow $\mathbf{v}_{medium}$ as opposed to the individual flow $\mathbf{v}_f$ for each particle. This decision was made in line with the theory, where $\mathbf{v}_D$ should be in the same direction for all particles. This method is also more stable against outliers, since it is less dependent on the randomness of the motion of individual particles. However, as seen in figure 20b, this is not the case for all tracks. Some trajectories differ significantly from the flow direction, which may suggest either tracking errors or errors in the input data from sources such as an uneven surface during the microscopy, which particles may get stuck in.

The radii have been estimated by an approximation seen in equation 4. If the parameters in equation 3 were known, the radius could have been estimated more accurately. The distribution seen in the histogram in figure 21a suggests that some radii of tracked particles differ with a factor of 2 or greater. Since $I_{rel} \sim r^2$, this would imply that a difference with a factor of 4 or greater should be seen in the relative intensity distribution, figure 21b. This was not observed, as with the exception of one outlier, all tracked particles had a relative intensity differing less than a factor 1.2. This leads to the suspicion

trained to predict a large r and (x,y) = (0,0) on empty images. This led to significantly better results than the ones presented here, but still worse than U-Track's results.

that the network's predictions of intensities are not accurate. The most likely reason is that the weight corresponding to intensity $w_5 = 5$ chosen in the training is too low. An analysis of the intervals within which the attributes of the particles in the experimental data fall reinforces this suspicion, as the distances and radii are usually smaller than 3 pixels, while the particle intensities are usually smaller than 0.15. Therefore, a weight of about $w_5 = 20$ should have been chosen in the training. This would likely have led to better results. Since the quality of the intensity prediction was deemed inadequate, the correlation between the radius and intensity of tracked particles was not investigated.

The result of the plots of bias, figure 24, can be interpreted in two different ways. Either false positive detections are more likely few predicted pixels of low intensity and correctly filtered, or real particles of this type are less likely to be tracked over long times which would be biased. A clear statistical distinguishment between the two interpretations is difficult to draw from the data without looking at tracks individually and especially so when the predicted intensity might not be correct. What can be said is that the bias does at least not render the network unusable, but as the network is bound to perform better on higher SNRs, some performance increase on particles of stronger signal will always be present. The sharp change in trend around twenty frames does support the choice filtering out tracks below this length as tracks behaves differently beyond it.

5.3.2 The cost matrix

Designing the cost matrix has a large impact on the final tracking results, while also being one of the more subjective parts of the processing. Two guiding principles were therefore formed during the development to provide a structure for approaching this problem. Firstly, cost terms should describe consistent physical attributes of the tracks rather than sought after results. Particles were known to determinedly move in a direction, be relatively sparse and vary in intensity a certain amount, which was modeled through C_D and C_I. Going more into detail, the maximum allowed distance M used in the term C_D was decided through an unbiased statistical method of MSD calculation to find the separator between including feasible Δx particle displacements while rejecting noise. As a counter example, it was found that tracks tended to include particles with a similar number of pixels. This information was not included in the cost matrix, as that would reinforce the tendency further even though it is not possible to prove its existence in the initial data. In reality the separation between these opposites is not as clear cut. Likely, there is for example an overlap between distances to noise and distances to and tracks which decreases the precision of the unbiased method of MSD. A certain level of fine tuning after preliminary results might therefore be inescapable to get the best possible final results.

Secondly, weighing several continuous cost terms against each other was decided against, as results quickly become complex and unpredictable. This is the main reason why the intensity difference was used for filtering extreme cases, rather than assessing likeness of particles. Such terms are easier to combine but should still be used with caution, as extreme behaviour might be overseen with severe filtering.

A final important detail of the cost matrix is the capping of costs at c_{max} seen in equation 17. This is enforced to make all unfeasible connections equally costly for the algorithm, guaranteeing that the least costly solution contains the least amount of unfeasible connections. If this was not the case, plausible connections may be sacrificed to favour several less costly unfeasible connections, which is not intended behaviour.

5.3.3 Track length and quality

Neither of the proposed quantifications of track quality suggested inclusion of false predictions to be a problem, with a consistently small σ and displacement covariances scattered around zero. The quantifications also improved from tracks regarded as noise to long tracks, which reinforces them as suitable measurements. Critically, examining these results, both methods are based on small displacements which are in some way guaranteed from the cost matrix mentioned above.

Set against the result of track length distributions, a reasonable conclusion is that the current configuration prioritizes fewer errors over longer tracks. A rough estimate for the average track length with a mean velocity of 6.17 µm/s over the frame width of 81.92 µm is 13.3 seconds or 318 frames while the current average track length is of length 44. It is easy to see how the two interests of long tracks and tracks of high quality conflict with each other, as one straight forward way to increase lengths is to loosen on the criteria for a connection between two positions to be made. A more subtle way of improving the algorithm is to imagine situations where a track would be cut off by the current algorithm and then try counteracting them. Such cases can be

- The particle making a unexpectedly large move between two frames.
- The particle not being detected by the network for one or more frames.
- The particle making a faulty connecting in one frame, throwing it of track.

Allowing for gaps in tracks similarly to the benchmarking script has potential to overcome the first two situations without relaxing conditions for other particles. One implementation would be extending the set of particles to be assigned to tracks $(\mathbf{x}_{n+1}, i)_{i=0}^{I_1}$ with $(\mathbf{x}_{n+2}, i)_{i=0}^{I_2}$, $(\mathbf{x}_{n+3}, i)_{i=0}^{I_3}$ and so on depending on the desired size of gaps and keeping unlinked particles further than one frame away for the next step of the tracking. Doing so with U-Track however, would effectively increase the amount of noise linearly with the amount of frames added, because of the false positives which are not as present in the benchmark script.

A suggested improved way would therefore be to run one original tracking and then connect track ends with track beginnings which may be some frames away through the same algorithm. As many of the original particles are not considered and short tracks are possible to filter away as noise between the two trackings, the second tracking could use a more generous cost matrix without risking inclusion of more noise.

One problem which was encountered when analysing tracks with gaps for the benchmarking script is to accurately extend the statistics of continuous tracks. For calculations of D and σ^2, values were calculated separately for each subtrack and weighted together based their respective lengths, but since averaging of displacements was advised against in the original article because of their statistical dependencies this strategy is questionable in cases where the exact result is of greater importance. Other plausible methods include ignoring gaps or substituting gaps with extrapolated data, which could be a negligible error source for long tracks.

5.4 Image generation routines

As shown in chapter 4.4, all three new image generation routines outperform the original method when measured on calculation time per image. We chose to implement the method calculating the Bessel function only near the particle centers. This is partly because it seems to be the fastest and partly, perhaps most significantly, because it retains the functionality of enabling use of other Bessel orders than 1.

In development, there were indications that neither the method chosen above or the one using the GPU were notably affected by increased image size. However, the chosen method was more affected by the number of particles than the one using the GPU, suggesting that there might be use cases with large and very dense images where the method using the GPU might be preferred to the one chosen.

It should be noted that DeepTrack's structure is such that the training is primarily done on small images containing only one particle. Thus the absolute time gained from implementing a cutoff is less prevalent for DeepTrack than it is for U-Track.

6 Conclusion and outlooks

The aim of U-Track performing at least as well as DeepTrack in the aspects of image classification and calculation efficiency for multiple particle images is to be regarded as fulfilled. For such images, U-Track consistently scores higher in precision and recall and lower in DULROC and MAE. It also requires less calculation time both for predictions (in most cases) and for image generation. It should be noted that handling this type of images is not the primary focus of DeepTrack, and this is likely part of the reason for the improvement demonstrated by our method.

The development of a framework for post-processing of network predictions into particle tracks further improves the usability of the library. This is exemplified by its application on experimental data with useful results. Several considerations for creating and analysing the tracks have been discussed, including prediction cutoff threshold, cost matrix calculation, and minimum track length filtering.

The next step in the development of U-Track could be to change the architecture to a Recurrent Neural Network, which retains information from the previous frame for the prediction of the next one. This would most likely lead to better performance than predicting each frame separately. The biggest challenge with this approach is the simulation of data, as realistically simulated movement and flickering of particles is crucial for the approach to work.

References

[1] S. Helgadottir, A. Argun, and G. Volpe, "Digital video microscopy enhanced by deep learning", *Optica*, vol. 6, no. 4, pp. 506–513, Apr. 2019. DOI: 10.1364/OPTICA.6.000506. [Online]. Available: http://www.osapublishing.org/optica/abstract.cfm?URI=optica-6-4-506.

[2] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation", in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III*, N. Navab *et al.*, Eds. Cham: Springer International Publishing, 2015, pp. 234–241, ISBN: 978-3-319-24574-4. DOI: 10.1007/978-3-319-24574-4_28. [Online]. Available: https://doi.org/10.1007/978-3-319-24574-4_28.

[3] G. Litjens *et al.*, "A survey on deep learning in medical image analysis", *Medical Image Analysis*, vol. 42, pp. 60–88, 2017, ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2017.07.005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1361841517301135.

[4] S. Jõemetsa, "Supported lipid membranes and their use for the characterization of biological nanoparticles", PhD thesis, Chalmers University of Technology, 2020.

[5] (2020). U-track repository, [Online]. Available: https://github.com/FredrikMeisingseth/U-Track.git (visited on 05/13/2020).

[6] T. Falk *et al.*, "U-net: Deep learning for cell counting, detection, and morphometry", *Nature Methods*, vol. 16, Jan. 2019. DOI: 10.1038/s41592-018-0261-2.

[7] D. M. Powers, "Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation", *Journal of Machine Learning Technologies*, pp. 37–63, 2011. [Online]. Available: http://hdl.handle.net/2328/27165.

[8] C. Vestergaard, P. Blainey, and H. Flyvbjerg, "Optimal estimation of diffusion coefficients from single-particle trajectories", *Physical review. E, Statistical physics, plasmas, fluids, and related interdisciplinary topics*, vol. 89, p. 022 726, Feb. 2014. DOI: 10.1103/PhysRevE.89.022726.

[9] Y. Yuan, M. Chao, and Y.-C. Lo, "Automatic skin lesion segmentation using deep fully convolutional networks with jaccard distance", *IEEE Transactions on Medical Imaging*, vol. 36, no. 9, pp. 1876–1886, 2017, cited By 115. DOI: 10.1109/TMI.2017.2695227. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85027980102&doi=10.1109%2fTMI.2017.2695227&partnerID=40&md5=9a387f4da29843912d3c1fdf3dd378d7.

[10] R. Byrd *et al.*, "Sample size selection in optimization methods for machine learning", *Mathematical Programming*, vol. 134, no. 1, pp. 127–155, 2012, cited By 119. DOI: 10.1007/s10107-012-0572-5. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84865685824&doi=10.1007%2fs10107-012-0572-5&partnerID=40&md5=a9b6209cd7eb9e6675576fee24510f39.

[11] D. Midtvedt *et al.*, "Size and refractive index determination of sub-wavelength particles and air bubbles by holographic nanoparticle tracking analysis.", *Analytical chemistry*, 2019.

[12] H. W. Kuhn, "The hungarian method for the assignment problem", *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955. DOI: 10.1002/nav.3800020109. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800020109. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109.

[13] J. A. Nelder and R. Mead, "A simplex method for function minimization", *The Computer Journal, Volume 7, Issue 4*, pp. 308–313, 1965. [Online]. Available: https://doi.org/10.1093/comjnl/7.4.308.

[14] R. Parthasarathy, "Rapid, accurate particle tracking by calculation of radial symmetry centers", *Nature Methods*, vol. 9, 7. [Online]. Available: https://doi.org/10.1038/nmeth.2071.

[15] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. USA: Prentice-Hall, Inc., 2006, ISBN: 013168728X.

[16] No author. (2020). Keras documentation, [Online]. Available: `https://keras.io/` (visited on 02/09/2020).

[17] X. Wu *et al.*, "Faultseg3d: Using synthetic data sets to train an end-to-end convolutional neural network for 3d seismic fault segmentation", *Geophysics*, vol. 84, no. 3, pp. IM35–IM45, 2019, cited By 20. DOI: `10.1190/geo2018-0646.1`. [Online]. Available: `https://www.scopus.com/inward/record.uri?eid=2-s2.0-85064681933&doi=10.1190%2fgeo2018-0646.1&partnerID=40&md5=0e0dbeb1a0438e02f50100d4f37d6f12`.

[18] C. Fiorio and J. Gustedtb, "Two linear time union-find strategies for image processing", *Theoretical Computer Science, Volume 154, issue 2*, pp. 165–181, 1996. [Online]. Available: `https://doi.org/10.1016/0304-3975(94)00262-2`.