Wrocław University
of Science and Technology

Faculty of Pure and Applied Mathematics
Field of study: Applied Mathematics
Specialty: Data Engineering

Master's Thesis

CHANGE POINT DETECTION IN SINGLE-PARTICLE TRAJECTORIES USING MACHINE LEARNING METHODS

Marcin Kostrzewa

keywords:
change point detection, deep learning, anomalous diffusion, single-particle motion

short summary:
This work aims to develop a deep learning based method for change point detection in single-particle trajectories. The proposed approach is precisely described and extensively trained and hypertuned to ensure high accuracy and robustness. Using a set of synthetic trajectories, the method is benchmarked against the baseline technique, highlighting its superior performance. Additionally, this method is a part of a solution submitted to the 2nd AnDi Challenge and is competitive with other contributions.

Supervisor	dr hab. Janusz Szwabiński	.....	.....
	Title/degree/name and surname	grade	signature

*For the purposes of archival thesis qualified to:**

a) category A (perpetual files)

b) category BE 50 (subject to expertise after 50 years)

** delete as appropriate*

stamp of the faculty

Wrocław, 2024

Contents

Introduction

In recent years, single-molecule imaging and single-particle tracking techniques have contributed to significant advancements in experimental data acquisition, for instance, in studies of telomeres, macromolecular complexes, proteins, and organelles within living cells [25, 32]. This, in turn, has led to the development of various new methods for analyzing particle motion, most of which take advantage of machine learning approaches [23].

One of the currently, actively researched problems is the change points detection (CPD) in trajectories of particles, i.e. identifying the moments when there is a significant change in the motion pattern of a particle, indicating a transition between different states or behaviors in its movement. Many methods have been recently proposed, and to assess and compare them, the 2$^{\text{nd}}$ AnDi Challenge was organized [24].

The objective of this work is to develop a method for change point detection utilizing a machine learning approach, specifically through deep learning techniques [30]. The proposed method aims to be robust, accurate, and competitive with the latest state-of-the-art techniques. It should also demonstrate strong generalization capabilities across various types of data and scenarios, ensuring broad applicability to different particle motion studies. For a comprehensive evaluation, the method will be tested using simulated data provided by the AnDi Challenge organizers.

The method introduced in this thesis constitutes a part of the solution submitted by researchers from the Hugo Steinhaus Center to the 2$^{\text{nd}}$ AnDi Challenge. This gives a possibility of an external performance evaluation as the organizers publish rankings for submitted solutions.

This thesis consists of four chapters. In the first one, we introduce anomalous diffusion, which is commonly used to model the behavior of particle motion. We also describe the models used to simulate the data and the problems the data imposes. The second chapter introduces both the baseline and proposed methods for detecting change points, detailing their mechanisms and components. In addition, we discuss the metrics used to evaluate the performance of change point detection methods. The next chapter is focused on training and hypertuning of the proposed method and selecting a proper neural network architecture. Finally, in the last chapter we conduct a thorough comparison between the baseline and the proposed method and also analyze the impact of the data characteristics on the results.

All simulations and computations were performed using the Python programming language [35]. The detailed list of used packages can be found in the Appendix. Additionally, code used to implement, train and evaluate the proposed method is available on GitHub in the following repository: `https://github.com/Manik2000/Masters-thesis`.

Chapter 1

Single particle motion

In this chapter, the concept of anomalous diffusion is introduced as it is a commonly chosen framework for modeling the motion of single particles. Then, different data generation mechanisms used in the AnDi Challenge 2 are shown and the difficulties imposed on CPD are described.

1.1 Anomalous diffusion

Anomalous diffusion represents a fundamental departure from the classical Brownian motion [21, 26]. A deviation from the standard Brownian model is observed in the behavior of mean squared displacement (MSD), which for a continuous stochastic process $X(t)$ is defined as:

$$\mathrm{MSD}(t) = \mathbb{E}\left(X(t) - X(0)\right)^2.$$

The traditional 1D Brownian motion is described by MSD that grows linearly with time, $\mathrm{MSD}(t) = 2Kt$, where K is the diffusion coefficient. However, in many complex systems, such as those encountered in biological, ecological, and material contexts, the observed particle trajectories do not conform to this linear relationship, indicating an anomalous diffusion process.

Anomalous diffusion is typically characterized by MSD which in one dimension is given in the following power-law form:

$$\mathrm{MSD}(t) = 2Kt^{\alpha},$$

where $\alpha \neq 1$ is the anomalous diffusion exponent and K is the generalized diffusion coefficient. Depending on the value of α, the diffusion process is classified as subdiffusion ($\alpha < 1$) or superdiffusion ($\alpha > 1$). Figure 1.1 shows the MSD for three different types of diffusion.

These deviations from normal diffusion can be described by several stochastic processes that embody the core characteristics of anomalous diffusion. In the first AnDi Challenge anomalous diffusion was generated by a variety of stochastic processes, including: the continuous-time random walk [31], the fractional Brownian motion [19], the Lévy walks [13], the annealed transient time motion [20], and the scaled Brownian motion [17]. In the second edition, a fractional Brownian motion and its extensions are considered.

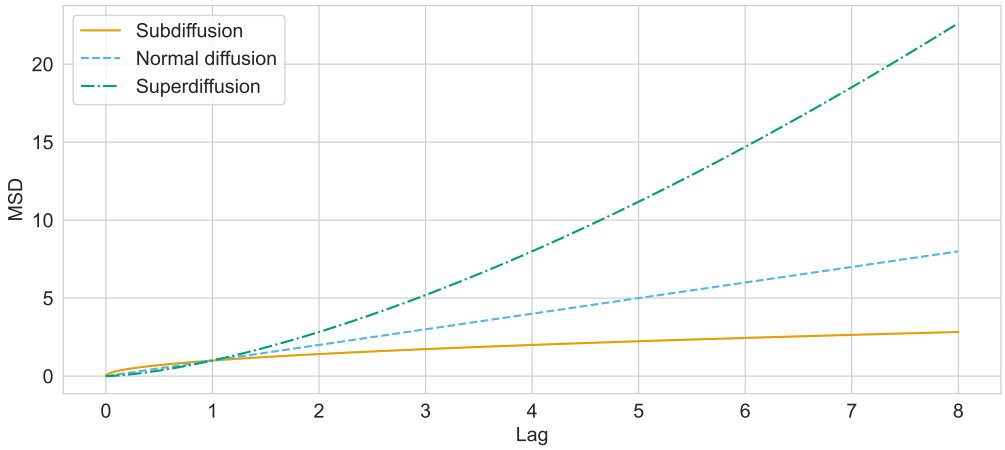

Figure 1.1: MSD for different types of diffusive motion.

1.2 2nd AnDi Challenge data

The base model of the particle motion in the 2nd AnDi Challenge is a 2D fractional Brownian motion (FBM) [24]. It is a continuous-time Gaussian process with stationary increments and the following correlation function:

$$\mathbb{E}\left[B_H(t)B_h(s)\right] = K\left(t^{2H} + s^{2H} - |t-s|^{2H}\right),$$

where K is a constant with units length$^2 \cdot$ time^{-2H} and H is the Hurst exponent.

The 2D FBM process $R(t)$ is represented as $\{X(t), Y(t)\}$ with X and Y being independent FBMs along the x and y axes, respectively. The MSD for unconstrained 2D FBM scales as:

$$\mathrm{MSD}(t) = 4Kt^{\alpha},$$

where the diffusion exponent α is related to the Hurst exponent as $\alpha = 2H$. When $\alpha = 1$, FBM simplifies to a Brownian motion. Consequently, FBM can exhibit subdiffusion for $0 < H < \frac{1}{2}$ $(0 < \alpha < 1)$, Brownian diffusion for $H = \frac{1}{2}$ $(\alpha = 1)$, and superdiffusion for $\frac{1}{2} < H < 1$ $(1 < \alpha < 2)$.

There are five different models for simulating data (from now on referred to as data models) in the 2nd AnDi Challenge. They can be used to generate trajectories in which the diffusion properties are piecewise constant along segments of varying duration and undergo sudden changes according to different mechanisms. Due to these mechanisms, different segments of generated trajectories have different values for coefficients α and K, which are sampled from normal distributions with means and variances set before simulating.

In case of each data model, simulations are performed considering particles diffusing in a square box of size L with reflecting boundary conditions, and the sampling time is equal to 1. The trajectory coordinates are corrupted with noise from a Gaussian distribution with zero mean and standard deviation σ_N.

The five data models are described below, and Figure 1.2 presents exemplary trajectories. More details on the data used in the 2nd AnDi Challenge can be found in the official manuscript prepared by the organizers [24].

Single-state model (SSM)

This model represents particles diffusing according to FBM with both a constant generalized diffusion coefficient K and a constant anomalous diffusion exponent α. For each trajectory, the values of K and α are drawn from the specified normal distribution. The trajectories produced by this model will be used to evaluate the false positive rate of change point detection methods, as they do not undergo any changes.

Multi-state model (MSM)

The multi-state model is a Markov model that describes particles undergoing FBM with diffusion properties that can change randomly. The number of states S is constant for a given experiment, as are the parameters that define the distributions of K and α for each state. For each trajectory, the S values of α and K are drawn from the distribution of the respective states. At each time step, a diffusing particle has a certain probability of changing one of its diffusive parameters (either α or K). The probability of switching is determined by a transition matrix M. From now on, it is assumed that $S = 2$.

Dimerization model (DIM)

This model considers the scenario in which dimerization, i.e. the temporary binding of two particles, can occur and alter the diffusion characteristics of both particles. We consider N circular particles with radius r. For each trajectory, the values of α and K are sampled from the distributions corresponding to the unbound state. If two particles are within a distance $d < 2r$, they have a probability P_b of binding. The dimer formed by the two particles moves with equal displacements, governed by a generalized diffusion coefficient K_{dim} and an anomalous diffusion exponent α_{dim} drawn from the distributions related to the dimeric state. At each time step, the dimer has the probability P_u of breaking its bond, allowing the two particles to revert to their original motion parameters.

Transient-confinement model (TCM)

This model considers an environment containing N circular compartments with a radius of r. The compartments are randomly positioned within the environment in such a way that they do not overlap. The particle enters the compartment with the probability of binding P_b and leaves with the probability of unbinding P_u. The diffusion rates differ inside and outside the compartments, thus creating two distinct diffusive states. For each trajectory, two values of α and two values of K are drawn from their respective distributions, representing the movement inside and outside the compartments.

Quenched-trap model (QTM)

This model simulates the diffusion of particles within an environment containing N stationary traps of radius r. The parameters α and K are drawn for each trajectory from their respective distributions and determine its free movement. When a particle enters the region defined by a trap, it has a probability P_b of binding to the trap and becoming temporarily immobilized ($K = 0$, $\alpha = 0$). At each time step, a trapped particle has a probability P_u of unbinding and being released from the trap, resuming its unrestricted motion. A particle cannot be trapped again until it takes a new step.

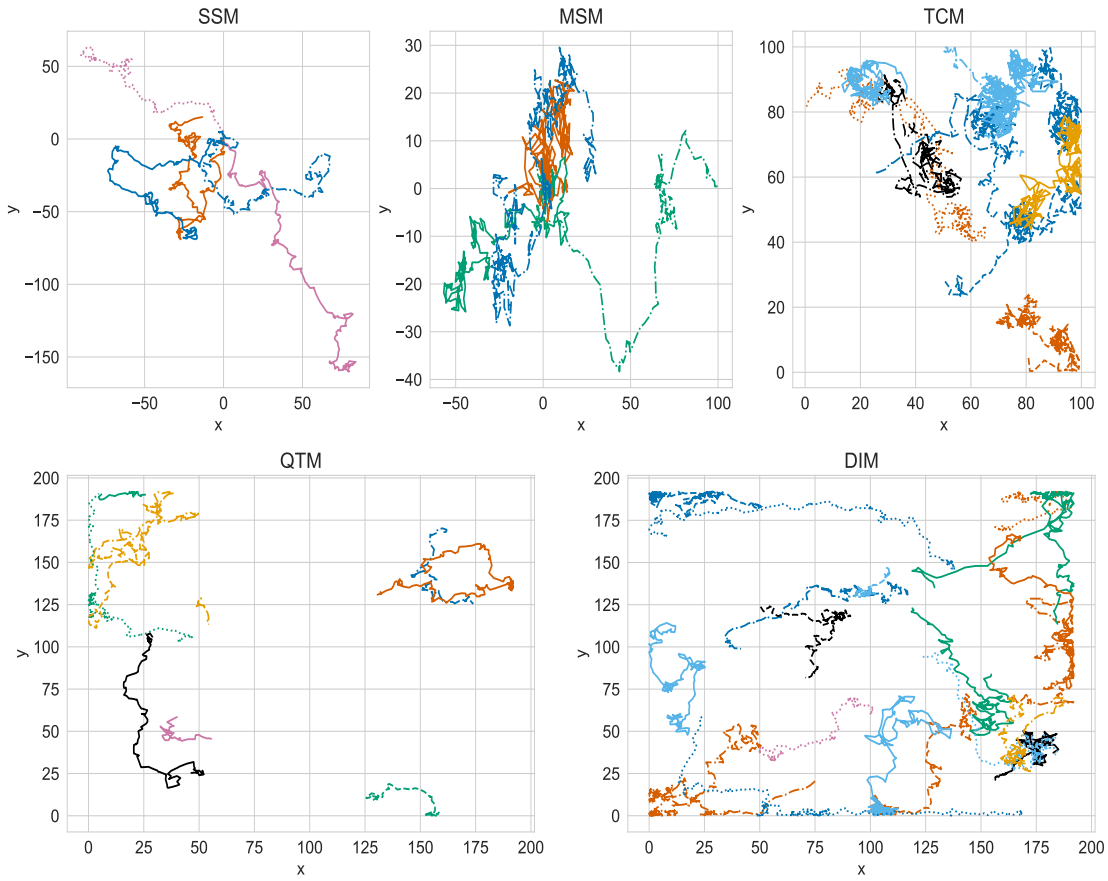

Figure 1.2: Exemplary trajectories for different particle motion models.

1.3 Difficulties imposed by data

Many existing change point detection methods work well if the observed changes stem from changes in the mean, variance, or linear trend of the trajectory. This is not our case, as the change points appear because of changes in underlying anomalous diffusion coefficients. Many existing methods are not able to detect such changes, resulting in poor performance. This can be mitigated by a change in a loss function as shown in Section 2.1 or by constructing a different method, which is the objective of this thesis.

Moreover, Figure 1.3 shows another challenge encountered. In each subplot there are x and y coordinates of multi-state model trajectories with $\alpha_1 = 0.7$, $\alpha_2 = 1.5$ and $K_1 = K_2 = 1$, but with different probabilities of a state transition. Increasing the value of p leads to a drastically increasing the number of change points. Some of the change points in the bottom plot are only three time steps apart from each other, and detecting all of them is rather impossible.

This all means that the method for detecting change points has to not only detect the changes in the underlying physical model rather than some observable measures such as mean or variance, but also be very flexible and robust. In addition, it should also work with very short segments of trajectories.

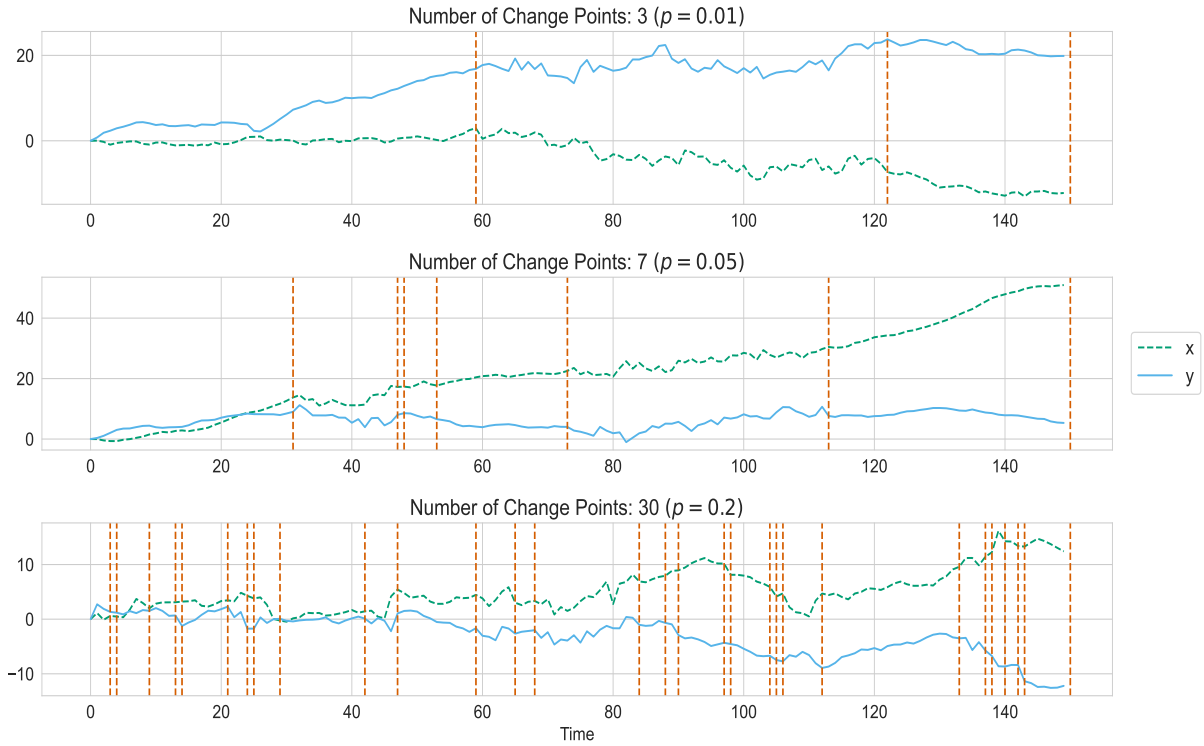

Figure 1.3: Exemplary multi-state model trajectories with different probability p of a state transition.

Chapter 2

Change point detection

This chapter is devoted to change point detection (CPD). First, we introduce a statistical-based window method that will be treated as a baseline in further comparisons. Then, a deep learning-based approach and all of its building blocks are described. Finally, we also introduce metrics that will be used to evaluate the performance of the change point detection.

Throughout this chapter, we assume that we are given a dataset of d-dimensional trajectories $\mathbf{x}^{(i)} = \left(\mathbf{x}_1^{(i)}, \ldots, \mathbf{x}_T^{(i)}\right)$, $i = 1, \ldots, N$, where each $\mathbf{x}_j^{(i)} \in \mathbb{R}^d$, $j = 1, \ldots, T$, where T is the length of each trajectory.

2.1 Baseline method

The baseline technique used in this thesis is a sliding window method utilizing a custom cost function [34]. It involves calculating the discrepancy between two neighboring windows that move along the trajectory $\mathbf{x}$. It is assumed $d = 2$, as in the case of the 2nd AnDi Challenge data. The sliding window algorithm serves as a rapid approximate substitute for optimal methods.

For a specified cost function $c(\cdot)$ and window length w, the discrepancy between two subsequences is defined as

$$d(\mathbf{x}_{t-w..t}, \mathbf{x}_{t..t+w}) = c(\mathbf{x}_{t-w..t+w}) - c(\mathbf{x}_{t-w..t}) - c(\mathbf{x}_{t..t+w}), \quad (w+1 \leq t \leq T-w),$$

where $\mathbf{x}_{a..b} = (\mathbf{x}_a, \ldots, \mathbf{x}_b)$ for any $1 \leq a \leq b \leq T$. When the two windows cover dissimilar segments, the discrepancy reaches larger values. Calculated discrepancies are collected and among them local peaks are chosen as the estimated locations of the change points.

The most important factor here is a proper cost function that takes into consideration the specifics of the data. In our case, this function should enable us to detect changes in the diffusion coefficients α and K. Thus, we propose the following cost function:

$$c(\mathbf{x}_{t-\tilde{w}..t+\tilde{w}}) = |\hat{\alpha}(\mathbf{x}_{t..t+\tilde{w}}) - \hat{\alpha}(\mathbf{x}_{t-\tilde{w}..t})| + \left|\hat{K}(\mathbf{x}_{t..t+\tilde{w}}) - \hat{K}(\mathbf{x}_{t-\tilde{w}..t})\right|,$$

where $\hat{\alpha}$ and $\hat{K}$ are estimators of α and K, respectively, and $\tilde{w}$ is the half of the sliding window length $w = 2\tilde{w}$.

Both estimators are based on empirical MSD and are taken from Ref. [14]. Assuming that we have a subsequence of the initial trajectory $(\mathbf{x}_t)_{t=A}^{B}$ of length $L = B - A$, $\hat{\alpha}$ is given by the following formula obtained by the least squares method:

$$\hat{\alpha} = \frac{n \sum_{\tau=\tau_{\min}}^{\tau_{\max}} \log(\tau) \log(M_L(\tau)) - \sum_{\tau=\tau_{\min}}^{\tau_{\max}} \log(\tau) \sum_{\tau=\tau_{\min}}^{\tau_{\max}} \log(M_L(\tau))}{n \sum_{\tau=\tau_{\min}}^{\tau_{\max}} \log^2(\tau) - \left(\sum_{\tau=\tau_{\min}}^{\tau_{\max}} \log(\tau)\right)^2},$$

where $A \leq \tau_{\min} < \tau_{\max} \leq B$ define the window used for the estimation, $n = \tau_{\max} - \tau_{\min} + 1$, and M_L is a time average MSD estimator (TAMSD):

$$\frac{1}{L-\tau} \sum_{i=1}^{L-\tau} \|\mathbf{x}_{i+\tau} - \mathbf{x}_i\|_2^2.$$

To estimate K, we use the following relation:

$$M_L(\tau) - M_L(\tau_{\min}) \approx 4K\left(\tau^{\alpha} - \tau_{\min}^{\alpha}\right), \quad \tau = \tau_{\min}, \ldots, \tau_{\max},$$

and get $\hat{K}$ by the iterative fitting procedure.

2.2 Deep learning method

So far, numerous machine learning approaches have been developed to address the problem of change point detection [3]. In this thesis, I will build my method using the idea of Jie Li, Paul Fearnhead, Piotr Fryzlewicz, and Tengyaofrom Wang [15]. They propose to train a classifier that will predict whether any change point is present in a given subsequence of time series and later, via a proper algorithm, use it for the problem of detecting multiple change points.

2.2.1 Neural layers

As mentioned above, the key point of the proposed method is to create a good predictive model. This is why we will consider using neural layers that are used to handle sequential data and capture temporal dependencies effectively [16]. Specifically, these are going to be:

- 1D concolutional layer [12],
- LSTM layer [9],
- attention layer [18].

In addition, dropout layers [33] are going to be used to avoid overfitting and the DAIN layer [28] for adequate data standardization.

Deep Adaptive Input Normalization (DAIN) layer

As pointed out by the authors of [28], deep neural networks have proved to be a great choice in many time-series analysis tasks. However, the performance of DL models can be severely affected if the data is not properly normalized. In case of time series, the non-stationary and multimodal nature of the data pose significant challenges. This is why

the authors of [28] decided to use a trainable neural layer, which contrary to other methods such as z-score or min-max normalization learns to perform normalization for a given task. They also show that the use of DAIN leads to significantly improved performance of different neural networks applied to time series forecasting tasks.

DAIN layer first calculates

$$\tilde{\mathbf{x}}_j^{(i)} = \left(\mathbf{x}_j^{(i)} - \boldsymbol{\alpha}^{(i)}\right) \oslash \boldsymbol{\beta}^{(i)},$$

where

- $\oslash$ is an entrywise division operator,
- $\boldsymbol{\alpha}^{(i)} = \mathbf{W}_a \mathbf{a}^{(i)} \in \mathbb{R}^d$ with $\mathbf{W}_a \in \mathbb{R}^{d \times d}$ being a weight matrix and

$$\mathbf{a}^{(i)} = \frac{1}{T} \sum_{j=1}^{T} \mathbf{x}_j^{(i)},$$

  ,
- $\boldsymbol{\beta}^{(i)} = \mathbf{W}_b \mathbf{b}^{(i)} \in \mathbb{R}^d$ with with $\mathbf{W}_b \in \mathbb{R}^{d \times d}$ being a weight matrix and

$$\mathbf{b}_k^{(i)} = \sqrt{\frac{1}{T} \sum_{j=1}^{T} \left(\mathbf{x}_{j,k}^{(i)} - \alpha_k^{(i)}\right)^2}, k = 1, \ldots, d.$$

$\boldsymbol{\alpha}^{(i)}$ and $\boldsymbol{\beta}^{(i)}$ layers are called *adaptive shifting layer* and *adaptive scaling layer*, respectively.

The final output of the DAIN layer — the output of *adaptive gating layer* — looks as follows:

$$\tilde{\tilde{\mathbf{x}}}_j^{(i)} = \tilde{\mathbf{x}}_j^{(i)} \odot \gamma^{(i)},$$

where $\odot$ is a Hadamard product and

$$\gamma^{(i)} = \sigma\left(\mathbf{W}_c \mathbf{c}^{(i)} + \mathbf{d}\right) \in \mathbb{R}^d, \quad \mathbf{c}^{(i)} = \frac{1}{T} \sum_{j=1}^{T} \tilde{\mathbf{x}}_j^{(i)},$$

with a weight matrix $\mathbf{W}_c$ and a bias vector $\mathbf{d}$, where $\sigma(x) = \frac{1}{1+e^{-x}}$ represents the sigmoid function.

1D Convolutional layer

A 1D convolutional layer [12] is a fundamental component of convolutional neural networks (CNNs) designed to process sequential data, such as time series, audio signals, or text sequences. Unlike traditional dense layers that connect every input neuron to every output neuron, convolutional layers apply convolution operations that preserve the local spatial or temporal structure of the data.

The primary operation in a 1D convolutional layer is the convolution, which involves sliding a filter (or kernel) across the input data and computing dot products at each position:

$$y[t] = (x * w)[t] + b = \sum_{i=0}^{k-1} x[t+i] \cdot w[i] + b.$$

Here, x is the input sequence, w is the filter, b is the bias term, k is the kernel size and y is the resulting feature map. For each dimension of the input trajectories, K kernels are created resulting in K feature maps y_i , $i = 1, \ldots, K$. The feature maps are averaged across the dimensions of the data. The output of the convolutional layer is thus of size (T, K). In this thesis, $k = 3$.

The convolution operation captures local patterns and features within the sequence, enabling the model to learn representations that are invariant to shifts and translations in the input. The resulting feature map represents the presence of learned features at different positions in the sequence.

After the convolution operation, an activation function is applied to the feature maps to introduce nonlinearity. The common choice is the ReLU function (rectified linear unit), which is defined as

$$\mathrm{ReLU}(x) = \max(0, x).$$

Usually, multiple convolutional layers are stacked to form an even deeper network. This aims to better capture the more complex patterns occurring in the underlying data.

Long Short-Term Memory (LSTM) Layer

Long Short-Term Memory (LSTM) networks [9] are a type of recurrent neural network (RNN) architecture designed to effectively capture long-term dependencies and mitigate the vanishing gradient problem commonly associated with traditional RNNs. LSTMs are a common building block in various sequence modeling tasks applied to time series forecasting [16], speech recognition [7], and natural language processing [27].

The LSTM layer is composed of identical cells. Each of them consists of the following transformations:

$$\begin{aligned}
\mathbf{i}_t &= \sigma\left(W_i \mathbf{x}_t + R_i \mathbf{h}_{t-1} + b_i\right) \\
\mathbf{f}_t &= \sigma\left(W_f \mathbf{x}_t + R_f \mathbf{h}_{t-1} + b_f\right) \\
\mathbf{g}_t &= \tanh\left(W_g \mathbf{x}_t + R_g \mathbf{h}_{t-1} + b_g\right) \\
\mathbf{o}_t &= \sigma\left(W_o \mathbf{x}_t + R_o \mathbf{h}_{t-1} + b_o\right) \\
\mathbf{c}_t &= \mathbf{i}_t \odot \mathbf{g}_t + \mathbf{c}_{t-1} \odot f_t \\
\mathbf{h}_t &= \tanh(\mathbf{c}_t) \odot o_t
\end{aligned}$$

where

- $\mathbf{h}_t$, $\mathbf{h}_{t-1}$ are the hidden states fo LSTM layer at time t and $t-1$; hidden states are initialized to 0,
- $\mathbf{c}_t$ is a cell state at time t,
- $\mathbf{x}_t$ is the d-dimensional input at time t,
- W, R and b terms with different subscripts are learnable weights and biases,
- σ is a sigmoid function,
- $\tanh(x)$ is a tangent hyperbolic function, $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

$\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{g}_t$, $\mathbf{o}_t$ are called input, forget, gate, and output gates, respectively. They are used to control the flow of information within the cell and maintain long-term dependencies.

The output of each cell is then propagated to the next one. By stacking many such cells, LSTMs are able to model intricate time-dependent behavior of data.

Attention mechanism

As mentioned in Ref. [18], the output of the LSTM layer can be forwarded to the attention layer to capture additional temporal dependencies. Let $\mathbf{h} \in \mathbb{R}^{T \times \tilde{d}}$ be the output of the LSTM layer. The attention layer with $\tilde{d}$ units first computes:

$$\mathbf{u} = \tanh\left(W\mathbf{h} + b\right),$$

where W is a weight matrix of dimensions $\tilde{d} \times \tilde{d}$ and b is a bias vector of size $\tilde{d}$. Next, we compute $\tilde{\mathbf{v}} = \mathbf{v} \odot \mathbf{u}$, where $\mathbf{v}$ is a context vector of size $\tilde{d}$, and the Hadamard product is applied row-wise. This vector is used to assess the significance of various features at a given timestamp. We then apply the softmax function to obtain $\mathbf{a} = \text{softmax}\left(\tilde{\mathbf{v}}\right)$, where

$$\left(\text{softmax}\left(\mathbf{G}\right)\right)_i = \frac{\exp\left(\mathbf{g}_i\right)}{\sum_{j=1}^{M} \exp\left(\mathbf{g}_i\right)},$$

with $\mathbf{G} \in \mathbb{R}^{P \times M}$, and $\mathbf{g}_i$ being the i-th row of $\mathbf{G}$. The final output of the attention layer is a vector $\tilde{\mathbf{a}}$, defined as:

$$\tilde{\mathbf{a}}_i = \sum_{j=1}^{T} \mathbf{a}_{ij}.$$

Dropout Layer

The dropout layer is an important element in neural network designs, especially to avoid overfitting [33]. This layer operates by randomly deactivating a portion of neurons during the training phase, determined by a probability p. The main objective of this method is to ensure that the neural network evolves into a more robust and generalized model by not relying too much on any single neuron or a small group of neurons.

When using a dropout layer during training, each neuron, along with its incoming and outgoing connections, is kept with a probability p or dropped with a probability $1 - p$. Consequently, each training sample experiences a slightly different network, which helps prevent neurons from co-adapting too closely. By training numerous thinned networks with overlapping architectures but slightly different neuron sets, dropout averages the predictions across these networks during the testing phase, leading to reduced overfitting.

In the forward pass, all neurons remain active; however, their outputs are typically scaled down by a factor of p to compensate for the increased number of active neurons compared to the training phase.

Dropout is particularly effective in training large neural networks, where the risk of overfitting is significant due to the large number of parameters relative to the number of training samples. The choice of p varies depending on the network architecture and the specific task; usually, $p \in (0, 0.5)$.

2.2.2 Classifier design

In this work, we utilize the neural architecture depicted in Table 2.1, referred to as `cnn_lstm_attention`. Here, B represents the batch size of the input data. The architecture includes convolutional layers with 128 and 64 output feature maps, respectively, followed by two LSTM layers containing 128 and 64 cells, and an attention layer with 64 units. Subsequently, there are three linear layers with 128, 64, and 1 neurons, respectively. The ReLU activation function is employed for all layers except the final one, which uses a sigmoid function to produce a probability output. First two dropout layers have probability p set to 0.3 and the rest to 0.2.

	Layer	Output Shape
1	DAIN	$(B, T, 2)$
2	Conv1D	$(B, T, 128)$
3	Conv1D	$(B, T, 64)$
4	LSTM	$(B, T, 128)$
5	Dropout	$(B, T, 128)$
6	LSTM	$(B, T, 64)$
7	Dropout	$(B, T, 64)$
8	Attention	$(B, 64)$
9	Dropout	$(B, 64)$
10	Dense	$(B, 128)$
11	Dropout	$(B, 128)$
12	Dense	$(B, 64)$
13	Dropout	$(B, 64)$
14	Dense	$(B, 1)$

Table 2.1: The `cnn_lstm_attention` model summary. B is the size of each data batch.

2.2.3 Algorithm for change point detection

Suppose that we have a trained classifier $\varphi : \mathbb{R}^{d\times w} \to [0,\, 1]$ that produces the probability that there is a change point in a subsequence of length w of a d-dimensional time series. The authors of Ref. [15] suggest to slide a moving window of length w over the time series $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_T)$ and for each window i, $i = 1, , \ldots, T - w + 1$ to calculate the probability p_i being an output of a classifier. We gather the output probabilities into a sequence $\mathbf{p} = (p_1, \ldots, p_{T-w+1})$ and then form a set $\{[s_1, e_1], \ldots, [s_{\tilde{v}}, e_{\tilde{v}}]\}$ where $[s_k, e_k] = (p_{s_k}, p_{s_k+1}, \ldots, p_{e_k})$, $k = 1, \ldots, \tilde{v}$, and each probability in these sequences exceeds a set threshold $\gamma \in [0, 1]$. We estimate the change point locations as the indices of the maximal probabilities in each subsequence. The Algorithm 1 outlines the scheme of the method.

It is reasonable to suspect that the window length w and the threshold level γ significantly influence the performance of the method. The effect of the threshold level and the impact of training classifiers on subsequences of varying lengths will be explored in the next two chapters.

Algorithm 1 Algorithm for multiple change points detection

Require: trajectory $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_T)$, $x_i \in \mathbb{R}^d$, a trained classifier $\varphi : \mathbb{R}^{d \times w} \to [0, 1]$, $\gamma > 0$

1: Form $\mathbf{X}_{[i,i+w)} := (\mathbf{x}_i, \ldots, \mathbf{x}_{i+w-1})$ for $i = 1, \ldots, T - w$.
2: Calculate $p_i = \varphi\left(\mathbf{X}_{[i,i+w)}\right)$ for $i = 1, \ldots, T - w$
3: Determine the set $\{[s_1, e_1], \ldots, [s_v, e_v]\}$, where for each segment $p_i > \gamma$ for all $i = s_r, \ldots, e_r$, $r = 1, \ldots, v$.
4: Compute $\hat{\tau}_r = \arg\max_{i \in [s_r, e_r]} p_i$.
5: **return** estimated change points locations $\hat{\tau}_1, \ldots \hat{\tau}_v$

2.3 Change point detection performance evaluation

In order to check how well a CPD algorithm works and compare it with others, measures of performance are needed. Five different metrics for the detection of change points will be used: root mean square error (RMSE), a measure α_{CP}, and Jaccard similarity coefficient, which are used by the organizers of the 2nd AnDi Challenge [24], F_1 score, and annotation error, which are commonly used metrics [2, 3, 34]. Let us introduce some notation before describing these metrics.

Let there be M_{GT} ground truth change points at locations $t_{(\mathrm{GT}),i}$, $i = 1, \ldots, M_{\mathrm{GT}}$, and M_P predicted change points at positions $t_{(\mathrm{P}),j}$, $j = 1, \ldots, M_{\mathrm{P}}$. We define the gated absolute distance as:

$$d_{ij} = \min\left(\left|t_{(\mathrm{GT}),i} - t_{(\mathrm{P}),j}\right|, \varepsilon_{\mathrm{CP}}\right),$$

where we set $\varepsilon_{\mathrm{CP}} = 10$.

We then pair predicted and ground truth change points by solving a rectangular assignment problem by minimizing the sum of distances:

$$d_{\mathrm{CP}} = \min_{\text{paired CP}}\left(\sum d_{ij}\right). \tag{2.1}$$

A Hungarian algorithm is recommended in [24], but we apply a modified Jonker-Volgenant algorithm without initialization described in [5]. Having paired the change points we can define:

- true positives (TP) — the paired true and predicted CPs with a distance smaller than $\varepsilon_{\mathrm{CP}}$,
- false positives (FP) — predictions not associated with any ground truth or having a distance larger than $\varepsilon_{\mathrm{CP}}$,
- false negatives (FN) — ground truth CPs not having an associated prediction at a distance shorter than $\varepsilon_{\mathrm{CP}}$.

Annotation error (AE) is simply an absolute difference between the number of predicted and ground truth CPs, i.e.:

$$\mathrm{AE} = |M_{\mathrm{GT}} - M_{\mathrm{P}}|.$$

This metric will be useful to assess whether the method is either over- or under-segmenting the trajectory.

Jaccard similarity coefficient (JSC) is given by a following formula:

$$\mathrm{JSC} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}.$$

JSC equal to 1 indicates a perfect match, where all detected change points are correct and no ground truth change points are missed. A coefficient equal to 0 indicates that there is no overlap between the detected and true change points. Thus, the Jaccard similarity coefficient provides a measure of the accuracy of the change point detection method by evaluating the proportion of correctly detected change points relative to the total number of unique change points.

$\mathbf{F_1}$ score, given by:

$$F_1 = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

is particularly useful when the costs of false positives and false negatives are similar. It helps to ensure that the method is not overly biased towards either detecting too many points (high recall but low precision) or being too conservative (high precision but low recall). The F_1 score ranges from 0 to 1, where 1 indicates perfect precision and recall.

$\boldsymbol{\alpha_{\mathrm{CP}}}$ coefficient is given as:

$$\alpha_{\mathrm{CP}} = 1 - \frac{d_{\mathrm{CP}}}{d_{\mathrm{CP}}^{\max}},$$

where d_{CP} is given by Equation (2.1) and $d_{\mathrm{CP}}^{\max} = M_{\mathrm{GT}}\varepsilon_{\mathrm{CP}}$. This metric is bound in $[0, 1]$, taking a value of 1 if there is an exact math between ground truth and predicted CPs.

Finally, **root mean square error (RMSE)** is defined as:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N} \sum_{\substack{\text{paired CP} \\ d_{ij} < \varepsilon_{\mathrm{CP}}}} \left(t_{(\mathrm{GT}),i} - t_{(\mathrm{P}),j}\right)^2},$$

where N is the number of paired change points. RMSE offers a way to assess the precision of change point detection in terms of the exact placement of the detected change points relative to the true change point.

Chapter 3

Method training and hypertuning

In this chapter, we train and hypertune the proposed deep learning method introduced in the previous chapter. First, we explain how the trajectory data is created and processed before being fed to the classifier. Then we use the generated data to train and assess the performance of the compared classifiers. We also examine the impact of varying the window length and threshold level on the results obtained.

3.1 Data creation and preprocessing

To train a classifier and later evaluate the CPD algorithm, we need to generate simulated trajectories. Deep learning methods benefit from a large amount of diverse data [30], so we generate many trajectories, each with randomly drawn parameters. The Python package `andi_datasets` [22], which was prepared by the organizers of the AnDi Challenge, is used for this. The different parameters used in the simulation procedure were described in Section 1.2.

In general, we assume that the particles move inside a box, the linear size of which is $L = 192$. For every trajectory, α for each segment is sampled from the normal distribution $\mathcal{N}(\mu_\alpha, \sigma_\alpha)$ with μ_α and σ_α sampled from uniform distributions: $\mu_\alpha \sim \mathcal{U}(0, 2)$, $\sigma_\alpha \sim \mathcal{U}(0, 0.15)$. Moreover, in the case of multi-state, transient-confinement and quenched-trap models the difference between anomalous exponent coefficients for two states have to differ by at least 0.05. The values of K are drawn from the normal distribution $\mathcal{N}(\mu_K, \sigma_K)$, where $\mu_K = F \cdot U$ with $U \sim \mathcal{U}(0, 10)$ and $F = 10^f$, f is a random number drawn from the sequence $\{-6, -5, \ldots, 3\}$, $\sigma_K \sim \mathcal{U}(0, 0.15)$.

In the MSM, for every trajectory, we sample a non-symmetric transition matrix:

$$M = \begin{pmatrix} 1 - p_1 & p_1 \\ p_2 & 1 - p_2 \end{pmatrix},$$

where $p_1, p_2 \sim \mathcal{U}(0, 0.15)$.

For TCM, we randomly position 50 confinement spaces with radiuses 10 inside the box. For QTM, we draw a random number of immobile traps uniformly from $\{50, \ldots, 150\}$. Their radius is set to 2. The traps are placed randomly inside the box. The binding and unbinding probabilities are drawn from the uniform distribution, $P_b \sim \mathcal{U}(0.85, 1)$, $P_u \sim \mathcal{U}(0, 0.15)$.

In the case of TCM, the binding and unbinding probabilities are also drawn from uniform distributions: $P_b \sim \mathcal{U}(0.9, 1)$, $P_u \sim \mathcal{U}(0, 0.1)$.

In this way, the trajectories are simulated and may be used for CPD evaluation. In order to prepare the data for the training of the classifiers, two additional steps are required. First, to address the challenges associated with the potential presence of numerous change points within a single trajectory, we consolidate closely situated change points into a single point. We use a greedy algorithm to group change points so that:

- in each group the distance between the location of first change point and the last one is at most 15,
- the sum of inertias of the groups is minimized; inertia of the group of change points $\tau_1, \ldots, \tau_n$ is defined as $\sum_{i=1}^{n} (\tau_i - \overline{\tau})^2$, where $\overline{\tau} = \frac{1}{n} \sum_{i=1}^{n} \tau_i$.

After merging the change points, we slide a window of chosen length w and collect each subsequence. Every subsequence gets a label assigned — 1 if there is at least one change point present inside it or 0 otherwise. For training classifiers, we set $w = 15$. Table 3.1 shows how many aubsequences with and without change points per each data model were included in the test, validation and test dataset.

Data model	Change (Y/N)	Train	Val	Test
TCM	N	106810	26895	33009
	Y	53094	13367	16896
DIM	N	106634	26694	33472
	Y	53613	13218	16569
QTM	N	106610	26659	33405
	Y	53320	13340	16677
MSM	N	106761	26672	33324
	Y	106786	26563	33408

Table 3.1: The counts of subsequences with or without change points per each data model across training, validation and test datasets.

In the MSM, the number of subsequences containing change points is significant, which prompts us to maintain a 1:1 ratio of labels. For other models, where the occurrence of change points is less frequent, we opt for an approximate 1:2 ratio of change to no-change subsequences.

3.2 Training and comparison of classifiers

We are going to compare four different classifiers, each built of the different layers that were described in the Chapter 2. The architecture of the `cnn_lstm_attention` model was also shown in Chapter 2, in Table 2.1. The other models follow a similar structure, but omit certain layers:

- `cnn` model does not include LSTM nor attention layers,
- `lstm` model does not contain convolutional and attention layers,
- `cnn_lstm` does not use attention layer.

The details of each of the models, the number of deep layers and parameters, is shown in Table 3.2.

Classifier	Number of layers	Number of parameters
`cnn`	10	156879
`lstm`	10	133135
`cnn_lstm`	12	190415
`cnn_lstm_attention`	14	194640

Table 3.2: Considered classifiers details.

The weights and biases of each model are initialized using the Glorot approach [6]. Each model is trained over 10 epochs. We use Adam optimizer to update the parameters of the networks [11]. The utilized loss function is standard binary cross-entropy. The learning rate for each case is set to 0.001. We use a batch of size 32 for a stable, though slower weight update.

Figure 3.1 presents the training and validation accuracy curves for each model. These curves indicate a stable and effective learning process for each model, with performance improving consistently over time and without evident signs of overfitting to the training data. The `cnn` model exhibits the lowest performance, with an accuracy over 1% lower than that of the `lstm` model in the validation dataset. Notably, the `cnn` model's learning arrives at a plateau after the initial few epochs, suggesting it may capture basic patterns in the trajectories, but struggles with the more complex temporal aspects due to its lack of sequential processing capabilities.

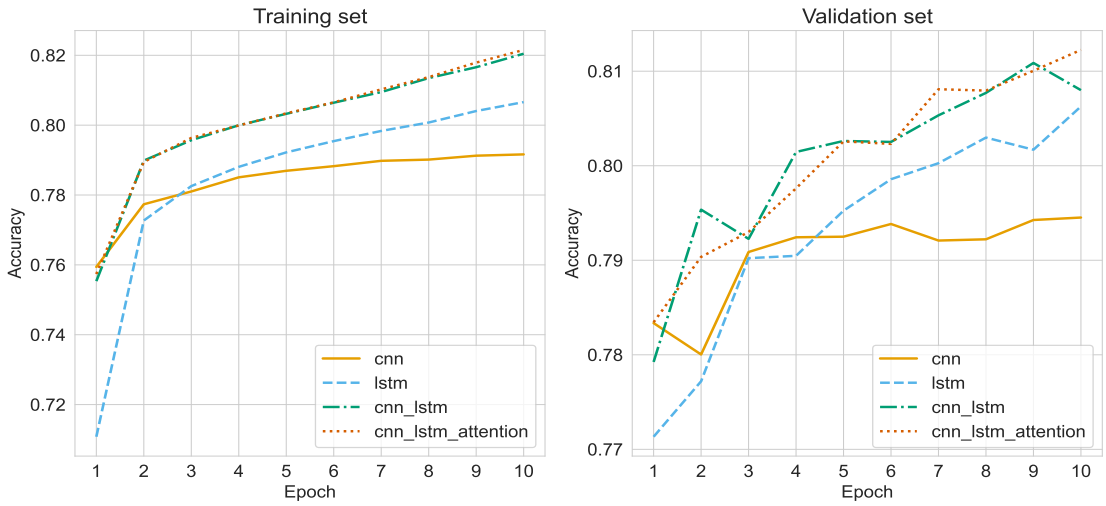

Figure 3.1: Training and validation accuracy for each classifier.

An interesting observation is that performance improves significantly when convolutional and LSTM layers are combined. This enhancement is likely because the convolutional layers effectively filter out noise and simplify the initial data, allowing the LSTM layers to extract and leverage the temporal features more effectively within the sequence. Among the models evaluated, the most sophisticated one — the `cnn_lstm_attention` model — performs the best, albeit only marginally outpacing the `cnn_lstm` model. The inclusion

of an attention layer in this model probably enables a more focused analysis of the most relevant features and time steps, significantly enhancing its ability to discern complex patterns and dependencies that are crucial for accurate task performance.

Through the analysis of the training and validation loss curves depicted in Figure 3.2, similar conclusions can be drawn as in the case of accuracy.

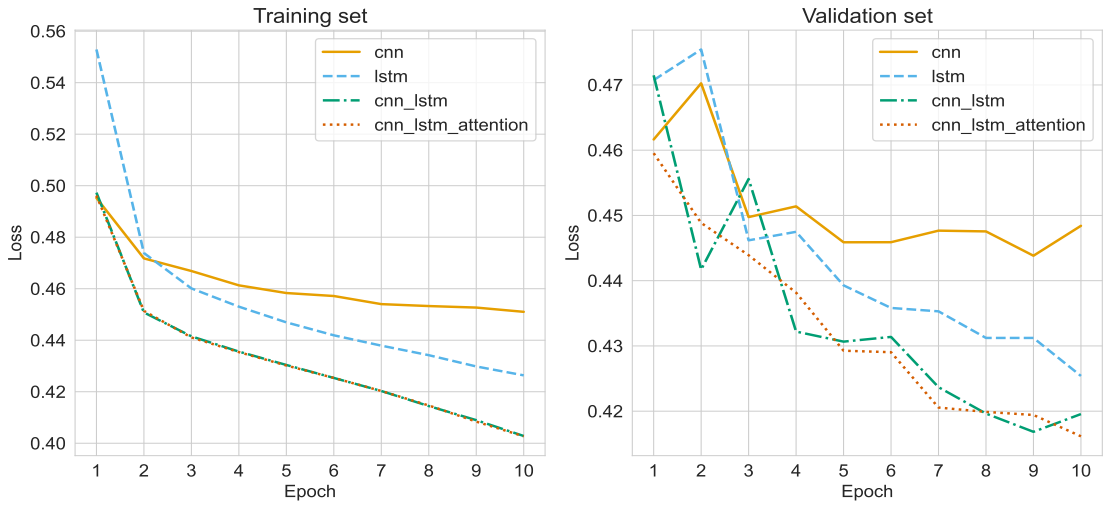

Figure 3.2: Training and validation loss for each classifier.

Performance in the test data set is presented in Table 3.3. The accuracy and F_1 score were calculated for each of the classifiers. The highest accuracy was obtained by the `cnn_lstm_attention` model, but the F_1 score equal to 0.7303 is slightly worse than for the `cnn_lstm` model. Given that the difference in the second metric is less significant, it is decided that the `cnn_lstm_attention` model will be the classifier used in the CPD method.

Classifier	Accuracy	F_1-score
`cnn`	0.7950	0.6868
`lstm`	0.8058	0.7126
`cnn_lstm`	0.8087	**0.7329**
`cnn_lstm_attention`	**0.8140**	0.7303

Table 3.3: Accuracy and F1-score for considered classifiers.

Figure 3.3 presents the confusion matrices for the classifier's predictions, distinguishing between each data model. The `cnn_lstm_attention` model shows varying effectiveness across different types of trajectories. It performs exceptionally well in detecting change points in the quenched-trap model with very good accuracy and, in contrast to results for other data models, a low false negative rate equal to 0.0929. Thus, it seems that change points in a particle trajectory caused by almost entirely stopping its motion are easily detected. The classifier achieves a very high true negative rate for the QTM and TCM models. On the other hand, for both models, an unexpectedly high false negative rate can be observed — our model predicts changes correctly in 50% of the cases. Encouragingly, the low false discovery rate suggests that when a change point is predicted, it is usually accurate. In contrast, results for the multi-state model indicate the poorest performance

in terms of true negative rate, with a true positive rate that, while better than TCM and DIM, still indicates challenges. The MSM proves particularly tough, as the classifier struggles with accurately predicting both no-change and change sequences.

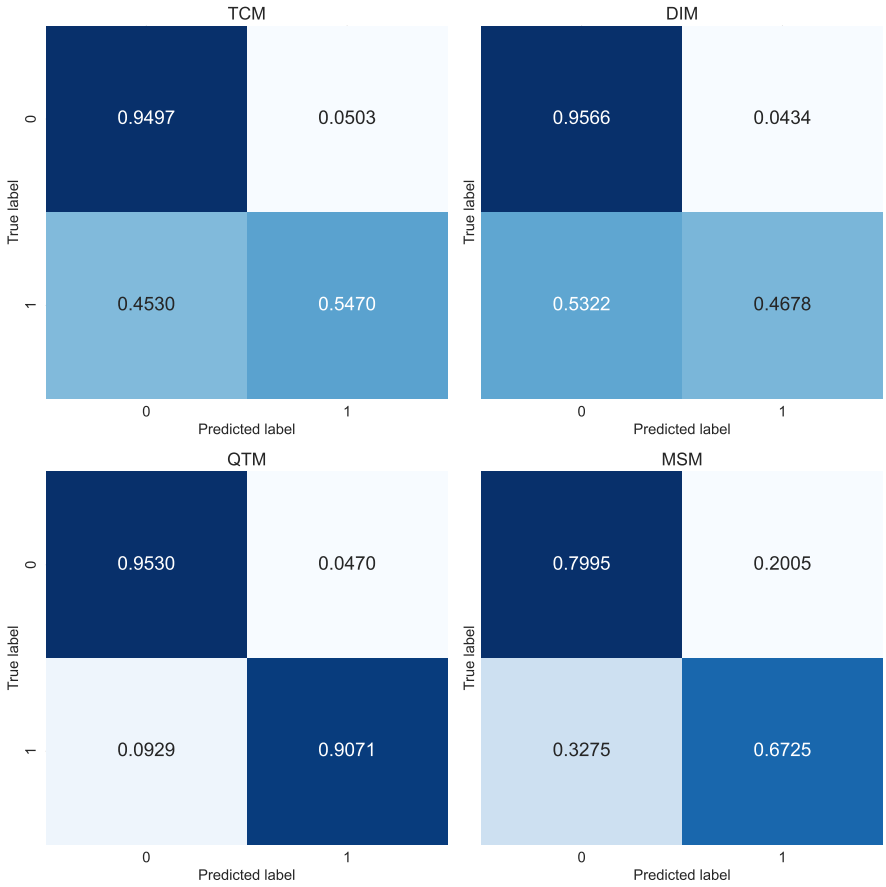

Figure 3.3: Confusion matrices for predictions made by `cnn_lstm_attention` classifier per data model.

Despite its imperfections, the classifier can still be utilized effectively within our CPD approach described in Chapter 2. The `cnn_lstm_attention` model calculates the probability p that a change point exists within a specific trajectory subsequence. Usually, if $p > 0.5$, the classifier identifies this as a change point, and all the results so far have been obtained using this approach. In our CPD approach, we introduce a hyperparameter γ — a threshold that can help filter out ambiguous classifier predictions, focusing only on those with high certainty. Section 3.4 is dedicated to the appropriate tuning of γ to optimize this process.

3.3 Different window lengths

Our classifier can be trained on subsequences of trajectories of different lengths. In the previous section, we chose 15 as the window size. Now, we will also check the results for $w = 10$ and $w = 20$.

We train the two models based on the `cnn_lstm_attention` architecture for the data created in the same way as described at the beginning of this chapter. The training procedure is also the same as for the model with $w = 15$.

Figure 3.4 shows the loss and accuracy curves for model with $w = 10$, whereas Figure 3.5 presents the same but for model with $w = 20$. In both cases, stable and efficient training can be observed.

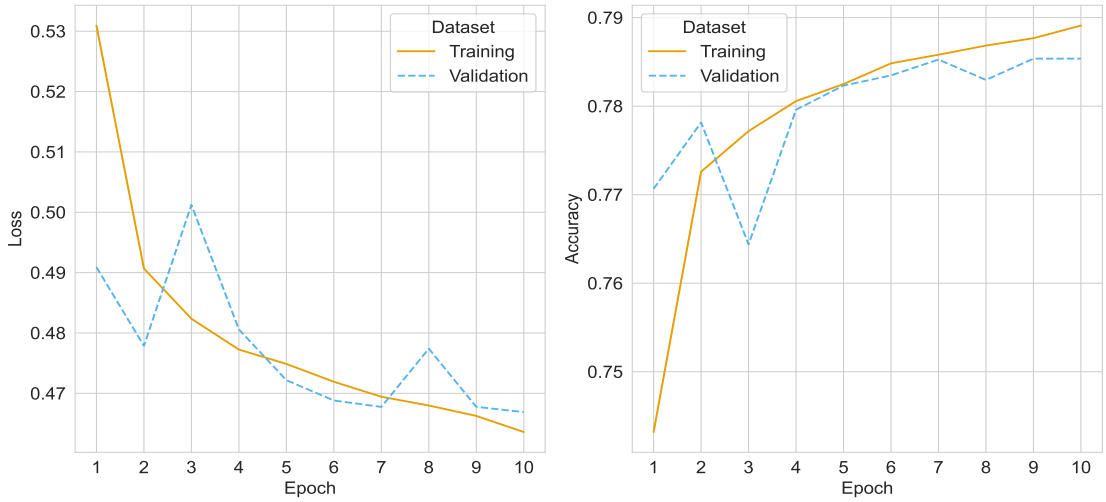

Figure 3.4: Training and validation loss and accuracy for `cnn_lstm_attention` model with window length $w = 10$.

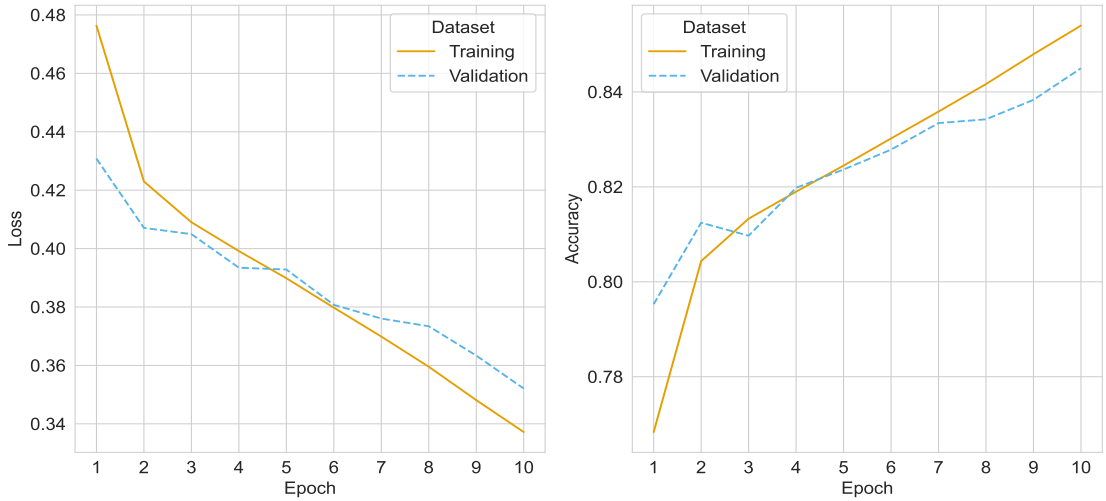

Figure 3.5: Training and validation loss and accuracy for `cnn_lstm_attention` model with window length $w = 20$.

Table 3.4 provides a comparison between the three models. Accuracy, F_1 score, recall, and precision are calculated for each data model. It can be observed that the longer the subsequences, the better the classifiers' results. In nearly every case, the `cnn_lstm_attention` model trained on subsequences of length 20 outperformed the other two models significantly. This is because longer trajectory sequences provide more contextual information and enable the classifier to capture complex dependencies more effectively. The best results can be observed for quenched-trap model, again mainly because of the specifics of the particle motion in this case. The differences between the classifiers are also less significant as in the case of the other data models.

Although the results for $w = 20$ are the best, this does not necessarily imply optimal results for the CPD method. Using a window length of 20 might hinder the method's ability to detect change points that occur frequently and close together. Consequently, methods trained on shorter sequences could potentially offer greater robustness. The implications of this will be explored further in the next chapter.

Data model	w	Accuracy	F1-score	Recall	Precision
TCM	10	0.772352	0.520934	0.371240	0.872919
	15	0.813325	0.664892	0.546993	0.847579
	20	**0.849372**	**0.743583**	**0.645144**	**0.877471**
DIM	10	0.763068	0.491799	0.343353	0.866366
	15	0.794768	0.601506	0.467801	0.842225
	20	**0.833123**	**0.694007**	**0.570815**	**0.885005**
QTM	10	0.927687	0.885422	0.847111	**0.927362**
	15	0.937722	0.906547	0.907118	0.905977
	20	**0.948868**	**0.923498**	**0.928025**	0.919015
MSM	10	0.692005	0.515784	0.400698	0.723618
	15	0.735929	0.718300	0.672504	0.770790
	20	**0.773843**	**0.750836**	**0.682026**	**0.835088**

Table 3.4: Performance of classifiers trained with different window lengths.

3.4 Hypertuning of the threshold level

As shown by the confusion matrices in Figure 3.3, the classifier had problems detecting change points in the sequences of all models except QTM. However, it was stated that maybe changing the level of threshold may help to take advantage of classifier's confidence. Figure 3.6 shows the result of such study. We trained three `cnn_lstm_attention` classifiers on three different window lengths and use them in the change point detection method. We generate 160 trajectories of length 85 for each data model with randomly chosen coefficients α and K. Then for each threshold level, we estimate the change points and calculate the F_1 score and the RMSE metrics and average their results on all trajectories.

As can be seen, clearly increasing the threshold level leads to improved performance — the F_1 score is increasing and the RMSE is decreasing. From now on, we will use the threshold equal to 0.9 for which both metrics are at the preferred level.

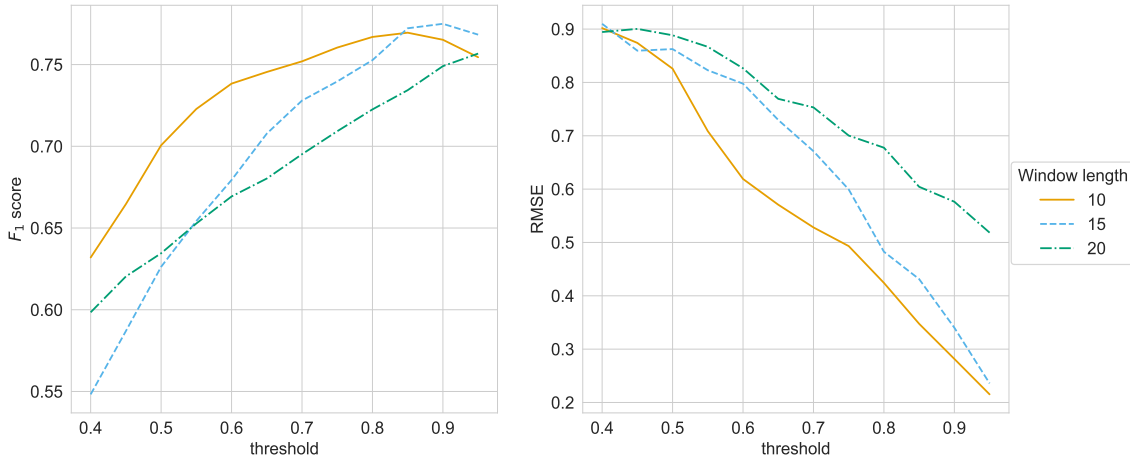

Figure 3.6: Averaged values of F_1 score and RMSE w.r.t. threshold level.

Chapter 4

Results

In the final chapter, the proposed CPD method with three different classifiers is evaluated and examined. First, we compare their results with the baseline approach. Then, we examine whether the size of differences between anomalous diffusion coefficients has impact on the performance of the method for multi-state model. Finally, the results of the method in the 2$^{\text{nd}}$ AnDi Challenge are briefly described.

4.1 Change point evaluation

Table 4.1 presents a comparison of change point detection performance across all five data models, each evaluated using a baseline detector and the proposed CPD detector in three variants of window length used, i.e. $w = 10, 15, 20$. Depending on the length of the window, the method is named accordingly CPDetector10, CPDetector15, and CPDetector20. The baseline method uses a window of length 20, as it allows the method to accurately estimate the diffusion coefficients and additionally to capture frequent change points. Performance metrics include the F_1 score, RMSE, Jaccard similarity coefficient, annotation error, and α_{CP}. The trajectories were simulated using the approach described in Section 3.1. The length of the simulated trajectories is 50.

For TCM, the CPDetectors consistently outperform the baseline across all metrics. The F_1 and JSC scores are notably higher, indicating better precision and recall. The lowest RMSE is obtained by CPDetector15, which also has the highest score for α_{CP} and is only slightly worse than CPDetector10 in other cases.

In case of DIM, similarly to TCM, all CPDetectors are significantly better than the baseline, with all the metrics scores again indicating enhanced accuracy and precision. CPDetector15 exhibits the lowest RMSE, along with the highest F1 score and the JSC, highlighting its efficiency in accurately detecting change points for this data model.

For the quenched-trap model, the CPDetectors dramatically improve over the baseline, with especially CPDetector10 showing exceptional enhancements in all metrics. The results obtained by this detector are also visibly better than those of other deep learning-based methods, indicating that the application of a length window 10 positively impacts the CPD performance. The annotation error values for CPDetectors are significantly lower than for the baseline, as in the case of TCM and DIM. This suggests that our CPD algorithm is less prone to predict spurious change points.

The results for the multi-state model present a more challenging scenario where CPDetectors work worse than the baseline method in case of F_1 score, JSC and AE. However, their RMSE values are much lower than those of the baseline method, which

Data model	Detector	F_1 score	RMSE	JSC	AE	α_{CP}
TCM	Baseline	0.6586	0.6706	0.5223	1.2127	0.8982
	CPDetector10	**0.8459**	0.1927	**0.7883**	**0.7686**	0.9859
	CPDetector15	0.8450	**0.1522**	0.7866	0.7888	**0.9907**
	CPDetector20	0.8329	0.3520	0.7709	0.8043	0.9716
DIM	Baseline	0.6757	0.5123	0.5453	1.3138	0.9403
	CPDetector10	0.8645	0.1625	0.8212	**0.7929**	0.9888
	CPDetector15	**0.8647**	**0.1336**	**0.8213**	0.8022	**0.9909**
	CPDetector20	0.8561	0.2181	0.8070	0.8331	0.9879
QTM	Baseline	0.6646	0.3647	0.5294	1.4152	0.9667
	CPDetector10	**0.9310**	0.1779	**0.9022**	**0.5166**	0.9894
	CPDetector15	0.9184	**0.1185**	0.8869	0.5911	**0.9940**
	CPDetector20	0.8936	0.1904	0.8585	0.6973	0.9884
MSM	Baseline	**0.5671**	1.6320	**0.4216**	**3.2922**	0.8932
	CPDetector10	0.4978	**0.3341**	0.3713	3.7516	**0.9804**
	CPDetector15	0.5268	0.4831	0.3972	3.6031	0.9713
	CPDetector20	0.5377	0.9689	0.4030	3.5156	0.9427
SSM	Baseline	0.6455	0.0000	0.4961	1.2304	1.0000
	CPDetector10	**0.9856**	0.0000	**0.9791**	**0.0470**	1.0000
	CPDetector15	0.9815	0.0000	0.9726	0.0580	1.0000
	CPDetector20	0.9183	0.0000	0.8848	0.3009	1.0000

Table 4.1: CP metrics results for trajectories of length 50.

proves that these methods locate change points more accurately than the baseline for all data models. In addition, the values of α_{CP} are higher than for the baseline, which, in light of the results for other metrics, suggest higher false positive and false negative numbers than for the baseline. This is also confirmed by the values of annotation error.

For SSM, for which there are no change points, the advantage of using CPDetectors over the baseline method is again observed. They are clearly less prone to falsely predict any change point in a trajectory, which is reflected in the results of F_1 score, JSC and AE.

Table 4.2 presents the same metrics scores but evaluated for trajectories of length 120. These outcomes indicate the scalability of the compared methods, as longer trajectories mean more complex data. The conclusions from analyzing the results of each of the data models are the same as in the case of Table 4.1. However, a deterioration of the metrics values is observed. It is obvious for the annotation error, which is more than two times larger, as there are naturally more spurious and undetected change points in the case of all applied CPD methods. However, for the baseline, a much more drastic deterioration in metrics values than for CPDetectors is observed. This leads to the conclusion that the proposed algorithm is much more robust and reliable.

In summary, CPDetectors generally provide substantial improvements in detecting change points across considered data models compared to the baseline. In addition, these models seldom incorrectly detect change points in single-state model trajectories, much less often than the baseline. The case of MSM should be investigated further, as the results of the baseline method were slightly better than those of CPDetectors. A possible mitigation would be to try to train the classifier on larger training samples.

Another important conclusion is that even though the classifier trained on longer

Data model	Detector	F_1 score	RMSE	JSC	AE	α_{CP}
TCM	Baseline	0.5102	1.1637	0.3860	2.8450	0.7985
	CPDetector10	**0.7606**	**0.3854**	**0.6855**	1.7783	**0.9718**
	CPDetector15	0.7580	0.3998	0.6773	1.7643	0.9699
	CPDetector20	0.7593	0.7877	0.6750	**1.6884**	0.9373
DIM	Baseline	0.5043	1.0361	0.3797	3.3270	0.8483
	CPDetector10	0.7448	0.3662	0.6777	2.3365	**0.9767**
	CPDetector15	**0.7486**	**0.3436**	**0.6803**	2.3207	0.9765
	CPDetector20	0.7425	0.5963	0.6665	**2.2986**	0.9584
QTM	Baseline	0.5344	0.6962	0.4269	3.2958	0.9227
	CPDetector10	**0.9034**	0.3536	**0.8589**	**1.1094**	0.9804
	CPDetector15	0.8883	**0.2796**	0.8432	1.2388	**0.9845**
	CPDetector20	0.8397	0.3521	0.7859	1.5131	0.9764
MSM	Baseline	**0.5068**	2.2608	**0.3579**	**6.9969**	0.8310
	CPDetector10	0.3769	**0.5189**	0.2684	8.7141	**0.9753**
	CPDetector15	0.4484	0.9982	0.3219	8.1469	0.9581
	CPDetector20	0.4977	1.9269	0.3565	7.4781	0.8982
SSM	Baseline	0.4804	0.0000	0.3802	3.2938	1.0000
	CPDetector10	**0.9725**	0.0000	**0.9598**	**0.0900**	1.0000
	CPDetector15	0.9510	0.0000	0.9296	0.1675	1.0000
	CPDetector20	0.7952	0.0000	0.7199	0.8436	1.0000

Table 4.2: CP metrics results for trajectories of length 120.

subsequences was able to achieve better classification metrics, its result when applied to CPD method is worse than for the other classifiers. Using shorter windows enables one to better capture change points if they occur more frequently. In addition, setting a higher threshold level effectively helps filter out incorrect predictions.

4.2 Impact of diffusion parameters on results

In this section, we explore how variations in the diffusion coefficients — α or K — influence the performance of the CPD algorithm. It is hypothesized that larger discrepancies between these coefficients in different states may enhance the method's ability to accurately detect change points. The multi-state model is selected for this analysis due to its frequent change points within the trajectories. Furthermore, we employ the CPDetector15 with threshold $\gamma = 0.9$, which has demonstrated the best performance among all versions of our method for this data model.

In case of coefficient α, we set $\alpha_1 = 0.3$ and $\alpha_2 = \alpha_1 + r$, where $r = 0.1, 0.2, \ldots, 1.5$, $K_1 = K_2 = 1$. For each value r, 500 trajectories of length 100 are generated. We then calculate the averaged RMSE and F_1 score.

Figure 4.1 presents the corresponding results. The plots suggest that there is no clear dependence of the discrepancy in alpha coefficients on the algorithm's effectiveness. The F_1 score does not show a consistent trend, instead it fluctuates up and down as the difference between the coefficients increases. This variability suggests that the ability of the CPD method to detect change points is not significantly improved by simply increasing the discrepancy between the α_1 and α_2 coefficients.

In addition, the RMSE does not exhibit a clear trend that correlates with increased differences in the alpha coefficients. This supports the conclusion that a greater difference between alpha coefficients does not necessarily lead to more accurate or precise change point detection in the case of our CPD approach.

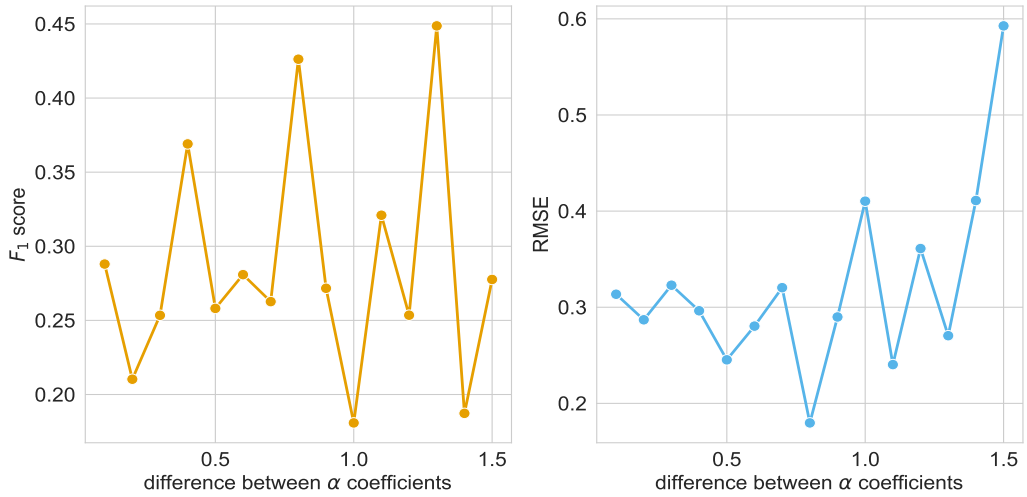

Figure 4.1: The impact of difference of α_1 and α_2 coefficient on the CPD performance.

Figure 4.2 presents the results of a similar analysis, but for coefficient K. The parameters values for different states were set in the following way: $K_1 = 10^{-3}$ and $K_2 = a \cdot K_1$, where $a \in \{2, 5, 10, 25, 50, 75, 100, 200, 250, 500, 750, 1000\}$. The rest of the generating procedure is the same as for α. The results obtained suggest a more nuanced relationship than with the coefficient α. The F_1 score shows a trend in which it generally increases as the ratio of the coefficients K increases. This pattern indicates that the CPD method better distinguishes between different states when the difference in diffusion behavior becomes more significant.

Contrary to the expected result, the RMSE also increases as the ratio of the K coefficients increases. This rise in this metric indicates that, while the classifier may be better at detecting changes, the precision of pinpointing the exact location suffers as the ratio increases.

The cause of these inconclusive results is difficult to determine. It is likely that the high variability and complexity of the data, which are not accounted for during the classifier's training, contribute to this issue.

4.3 2^{nd} AnDi Challenge results

The change point detection method developed in this thesis is used by our team from the Hugo Steinhaus Research Center (called HSC AI Team) in the 2^{nd} AnDi Challenge. As illustrated in the second table of Figure 4.3, the performance of this method secured us the second position (at least provisionally) in both the Jaccard similarity coefficient and RMSE, significantly outperforming other teams. This, in addition to previous evaluations, proves that the proposed methodology is capable of accurate segmentation of anomalous diffusion trajectories.

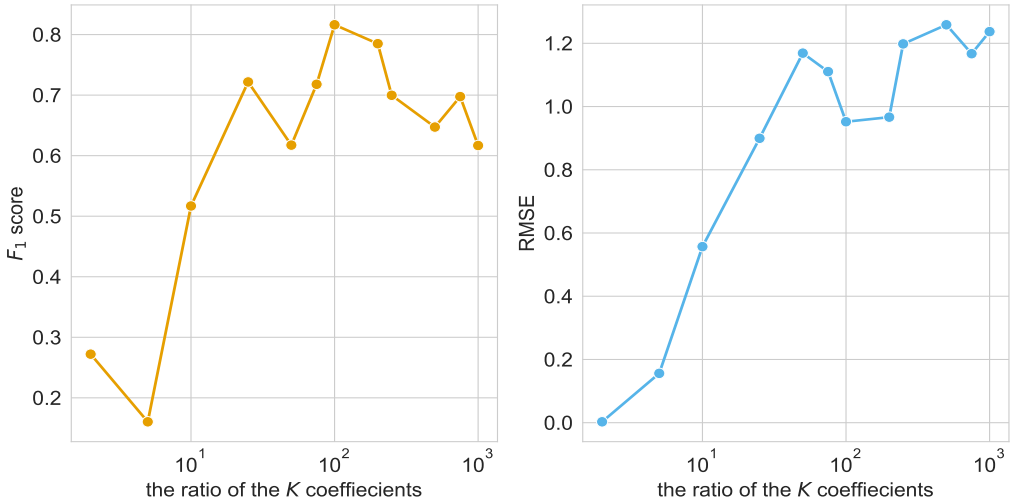

Figure 4.2: The impact of ratio of K_1 and K_2 coefficients on the CPD performance

Track 2 (trajs) - Ensemble Task

#	User	Entries	Date of Last Entry	Team Name	MRR ▲	Metrics		Detailed Results
						W1 (K) ▲	W1 (alpha) ▲	
1	malinoner	14	05/19/24	HSC AI	0.750	0.31 (1)	0.18 (2)	View
2	UnfriendlyAI	56	05/19/24	Unfriendly AI	0.750	0.65 (2)	0.18 (1)	View
3	YusefAhsni	16	05/17/24	ICSO UPV	0.333	0.69 (3)	0.35 (3)	View
4	junwoo5071	28	05/14/24	FIONA	0.250	1.35 (4)	0.48 (4)	View
5	alphayuan	15	05/10/24		0.200	1000000.00 (5)	2.00 (5)	View

Track 2 (trajs) - Single Traj Task

#	User	Entries	Date of Last Entry	Team Name	MRR ▲	Metrics					Detailed Results
						RMSE (CP) ▲	JSC (CP) ▲	MSLE (K) ▲	MAE (alpha) ▲	F1 (diffusion type) ▲	
1	UnfriendlyAI	56	05/19/24	Unfriendly AI	0.717	3.00 (3)	0.61 (1)	0.08 (1)	0.23 (1)	0.73 (4)	View
2	malinoner	14	05/19/24	HSC AI	0.550	2.67 (2)	0.53 (2)	2.57 (4)	0.37 (2)	0.78 (1)	View
3	YusefAhsni	16	05/17/24	ICSO UPV	0.517	2.08 (1)	0.29 (3)	0.18 (2)	0.47 (4)	0.77 (2)	View
4	alphayuan	15	05/10/24		0.300	3.74 (4)	0.15 (4)	2.22 (3)	0.41 (3)	0.76 (3)	View
5	junwoo5071	28	05/14/24	FIONA	0.200	4.41 (5)	0.02 (5)	190.87 (5)	2.00 (5)	0.03 (5)	View

Figure 4.3: Results of teams competing in the 2^{nd} AnDi Challenge as of May 22th.

Conclusions

This thesis was focused on developing a change point detection method to analyze single-particle trajectories utilizing a deep learning-based approach. The primary goal was to improve the precision and reliability of CPD over the baseline statistical method to detect significant changes within the trajectory data that could indicate underlying physical or biological events.

To achieve this, a neural network architecture was designed that incorporates the DAIN layer, 1D convolutional layers, Long-Short-Term Memory cells, and attention mechanisms. Different neural layers played different roles and stacked led to better performance when applied in different configurations. DAIN layer was used to properly normalize the input, which was crucial in case of challenging trajectory data. CNN layers were tasked with efficient feature extraction, denoising, and pinpointing spatial features, while LSTM layers were utilized to capture temporal dependencies essential for understanding the sequence progression. Moreover, the introduction of an attention mechanism allowed for focused analysis on key features.

The effectiveness of the developed CPD method was assessed against a statistical-based sliding window baseline method using synthetic trajectories from the 2$^{\text{nd}}$ AnDi Challenge. The data models encompassed various types of diffusion processes, including transient-confinement, dimmerization, quenched-trap, and multi-state processes. These models served as rigorous tests of the robustness and adaptability of the approach.

The empirical results confirmed that the proposed method consistently outperformed the baseline one. A comparative analysis underscored substantial enhancements in detection accuracy, especially evident when managing longer trajectory sequences. Only some of results for multi-state model were slightly worse than those of the baseline. However, this problem can be mitigated by adding more training examples. The comparison was quantified using various CPD metrics, such as the F_1 score, the root mean squared error, or the Jaccard similarity coefficient. The results obtained in the 2$^{\text{nd}}$ AnDi Challenge itself also proved the method's good performance.

Future research can explore several potential directions. A promising one is the exploration of transformer architectures which have shown remarkable success in various domains of machine learning, especially in the handling of sequence data. Transformers could potentially offer improvements over LSTM-based models because of their ability to process entire sequences simultaneously and their enhanced capability for handling long-range dependencies.

In addition, the application of self-supervised learning techniques could be examined. Self-supervised learning has gained substantial traction across various fields of machine learning due to its efficiency in leveraging unlabeled data. This approach could be particularly advantageous for capturing complex, subtle patterns in trajectory data that are not immediately apparent, reducing the reliance on extensive labeled datasets and improving the model's ability to generalize from fewer examples.

In summary, this thesis has established a robust framework for change point detection in single-particle trajectory data, utilizing deep learning techniques. The promising results affirm the practical utility and enhanced performance of the developed models. Further research may refine these techniques, potentially extending their applicability and efficiency in real-world settings.

Appendix

All calculations were performed on a computer with the following technical specifications: 4-core Intel Core i7-1165G7 processor, 512GB SSD, 16GB memory RAM.

To simulate single-particle trajectory data, the `andi_datasets` package created by the AnDi Challenge organizers was used [22]. It is available at `https://github.com/AnDiChallenge/andi_datasets`.

For data preprocessing purposes, to implement CPD metrics, the Python packages `scipy` [36], `numpy` [8] were applied.

For the creation of deep learning methods, `tensorflow` [1] and `keras` [4] were used. Moreover, for all machine learning related tasks, the package `scikit-learn` [29] was utilized.

To create visualizations, `matplotlib` [10] and `seaborn` [37] were used.

The code used for the DAIN layer creation and training was downloaded from `https://github.com/llouislu/tensorflow-dain`.

Bibliography

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Aminikhanghahi, S., Cook, D. J. A survey of methods for time series change point detection. *Knowledge and Information Systems 51* (2016), 339 – 367.

[3] Aminikhanghahi, S., Cook, D. J. A survey of methods for time series change point detection. *Knowl. Inf. Syst. 51*, 2 (May 2017), 339–367.

[4] Chollet, F., et al. Keras. `https://keras.io`, 2015.

[5] Crouse, D. F. On implementing 2d rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems 52*, 4 (2016), 1679–1696.

[6] Glorot, X., Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010), Y. W. Teh and M. Titterington, Eds., vol. 9 of *Proceedings of Machine Learning Research*, PMLR, pp. 249–256.

[7] Graves, A., Jaitly, N., rahman Mohamed, A. Hybrid speech recognition with deep bidirectional lstm. In *ASRU* (2013).

[8] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T. E. Array programming with NumPy. *Nature 585*, 7825 (Sept. 2020), 357–362.

[9] Hochreiter, S., Schmidhuber, J. Long short-term memory. *Neural computation 9* (12 1997), 1735–80.

[10] Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering 9*, 3 (2007), 90–95.

[11] Kingma, D. P., Ba, J. Adam: A method for stochastic optimization. *CoRR abs/1412.6980* (2014).

[12] Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., Inman, D. J. 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing 151* (2021), 107398.

[13] Klafter, J., Zumofen, G. Lévy statistics in a hamiltonian system. *Phys. Rev. E 49* (Jun 1994), 4873–4877.

[14] Lanoiselée, Y., Sikora, G., Grzesiek, A., Grebenkov, D., Agnieszka, W. Optimal parameters for anomalous-diffusion-exponent estimation from noisy data. *Physical Review E 98* (12 2018).

[15] Li, J., Fearnhead, P., Fryzlewicz, P., Wang, T. Automatic change-point detection in time series via deep learning. *Journal of the Royal Statistical Society Series B: Statistical Methodology 86*, 2 (01 2024), 273–285.

[16] Lim, B., Zohren, S. Time-series forecasting with deep learning: a survey. *Philos. Trans. A Math. Phys. Eng. Sci. 379*, 2194 (Apr. 2021), 20200209.

[17] Lim, S. C., Muniandy, S. V. Self-similar gaussian processes for modeling anomalous diffusion. *Phys. Rev. E 66* (Aug 2002), 021114.

[18] Liu, G., Guo, J. Bidirectional lstm with attention mechanism and convolutional layer for text classification. *Neurocomputing 337* (2019), 325–338.

[19] Mandelbrot, B. B., Van Ness, J. W. Fractional brownian motions, fractional noises and applications. *SIAM Review 10*, 4 (1968), 422–437.

[20] Massignan, P., Manzo, C., Torreno-Pina, J. A., García-Parajo, M. F., Lewenstein, M., Lapeyre, G. J. Nonergodic subdiffusion from brownian motion in an inhomogeneous medium. *Phys. Rev. Lett. 112* (Apr 2014), 150603.

[21] Metzler, R., Jeon, J.-H., Cherstvy, A. G., Barkai, E. Anomalous diffusion models and their properties: non-stationarity, non-ergodicity, and ageing at the centenary of single particle tracking. *Phys. Chem. Chem. Phys. 16* (2014), 24128–24164.

[22] Muñoz-Gil, G., Requena, B., Volpe, G., Garcia-March, M. A., Manzo, C. AnDiChallenge/ANDI_datasets: Challenge 2020 release, 2021.

[23] Muñoz-Gil, G., Volpe, G., Garcia-March, M. A., Aghion, E., Argun, A., Hong, C. B., Bland, T., Bo, S., Conejero, J. A., Firbas, N., Garibo I Orts, Ò., Gentili, A., Huang, Z., Jeon, J.-H., Kabbech, H., Kim, Y., Kowalek, P., Krapf, D., Loch-Olszewska, H., Lomholt, M. A., Masson, J.-B., Meyer, P. G., Park, S., Requena, B., Smal, I., Song, T., Szwabiński, J., Thapa, S., Verdier, H., Volpe, G., Widera, A., Lewenstein, M., Metzler, R., Manzo, C. Objective comparison of methods to decode anomalous diffusion. *Nat. Commun. 12*, 1 (Oct. 2021), 6253.

[24] Muñoz-Gil, G., Bachimanchi, H., Pineda, J., Midtvedt, B., Lewenstein, M., Metzler, R., Krapf, D., Volpe, G., Manzo, C. Quantitative evaluation of methods to analyze motion changes in single-particle experiments, 2024.

[25] Norregaard, K., Metzler, R., Ritter, C., Berg-Sørensen, K., Oddershede, L. Manipulation and motion of organelles and single molecules in living cells. *Chemical Reviews 117* (02 2017).

[26] Oliveira, F., Ferreira, R., Lapas, L., Vainstein, M. Anomalous diffusion: A basic mechanism for the evolution of inhomogeneous systems. *Frontiers in Physics 7* (02 2019).

[27] Otter, D. W., Medina, J. R., Kalita, J. K. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems 32*, 2 (2021), 604–624.

[28] Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., Iosifidis, A. Deep adaptive input normalization for time series forecasting. *IEEE Transactions on Neural Networks and Learning Systems 31*, 9 (2020), 3760–3765.

[29] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[30] Sarker, I. Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science 2* (08 2021).

[31] Scher, H., Montroll, E. W. Anomalous transit-time dispersion in amorphous solids. *Phys. Rev. B 12* (Sep 1975), 2455–2477.

[32] Shen, H., Tauzin, L. J., Baiyasi, R., Wang, W., Moringo, N. A., Shuang, B., Landes, C. F. Single particle tracking: From theory to biophysical applications. *Chemical reviews 117 11* (2017), 7331–7376.

[33] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, 56 (2014), 1929–1958.

[34] Truong, C., Oudre, L., Vayatis, N. Selective review of offline change point detection methods. *Signal Processing 167* (2020), 107299.

[35] Van Rossum, G., Drake, F. L. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[36] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald,

A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods 17* (2020), 261–272.

[37] Waskom, M. L. seaborn: statistical data visualization. *Journal of Open Source Software 6*, 60 (2021), 3021.