


Classification of diffusion modes in single-particle tracking data: Feature-based versus deep-learning approach

Patrycja Kowalek, Hanna Loch-Olszewska, and Janusz Szwabiński 

Faculty of Pure and Applied Mathematics, Hugo Steinhaus Center, Wrocław University of Science and Technology, 50-370 Wrocław, Poland



(Received 27 February 2019; published 20 September 2019)

Single-particle trajectories measured in microscopy experiments contain important information about dynamic processes occurring in a range of materials including living cells and tissues. However, extracting that information is not a trivial task due to the stochastic nature of the particles' movement and the sampling noise. In this paper, we adopt a deep-learning method known as a convolutional neural network (CNN) to classify modes of diffusion from given trajectories. We compare this fully automated approach working with raw data to classical machine learning techniques that require data preprocessing and extraction of human-engineered features from the trajectories to feed classifiers like random forest or gradient boosting. All methods are tested using simulated trajectories for which the underlying physical model is known. From the results it follows that CNN is usually slightly better than the feature-based methods, but at the cost of much longer processing times. Moreover, there are still some borderline cases in which the classical methods perform better than CNN.

DOI: [10.1103/PhysRevE.100.032410](https://doi.org/10.1103/PhysRevE.100.032410)

I. INTRODUCTION

Recent advances in single-molecule microscopy and imaging technologies have made single-particle tracking (SPT) a popular method for analyzing dynamic processes in a range of materials [1,2]. In a typical SPT measurement the molecules of interest (e.g., proteins in a living cell) are tagged with fluorescent dye particles. After illumination by a laser, the labels produce light and their positions may be determined with a microscope. Using lasers that flash at short time intervals allows for tracking of the movement of the molecules over time. The recorded positions are used to reconstruct trajectories of individual molecules. These trajectories are then analyzed in order to extract local physical properties of the molecules and their environment, such as velocity, diffusion coefficient (or tensor), and confinement (local density of obstacles) [3].

The SPT method is of particular importance for fundamental biology. It bridges the gap between biology, biochemistry, and biophysics and allows for at least a partial understanding of living cells on a microscopic basis. It has helped already to unveil the details of the movement of molecular motors inside cells [4,5] and target search mechanisms of nuclear proteins [6].

The analysis of SPT trajectories is not a trivial task due to the stochastic nature of the molecules' movement. It usually starts with the detection of a corresponding motion type of a molecule, because this information may already provide insight into mechanical properties of the molecule's surroundings [7]. Four basic types of motion are observed in SPT experiments: normal diffusion (ND) [8], directed motion (DM) [9–11], anomalous diffusion (AD) [12], and confined diffusion (CD) [13]. The most common analysis method uses mean square displacement (MSD) curves [11]. Within this approach one fits the theoretical curves for various physical models to the data and then selects the best fit with statistical analysis [13]. However, in many cases the actual trajectories

are too short to extract meaningful information from the time-averaged MSDs. Moreover, the finite localization precision adds a term to the MSD, which can limit the interpretation of the data [11,14,15]. Consequently, several alternative methods have been introduced to overcome these problems. For instance, the full distribution of displacements may be fitted to a mixed model in order to extract differences in diffusive behavior between subsets of particle ensembles [16]. The moment scaling spectrum method can also be used to categorize various modes of motion [17,18]. The distribution of directional changes [19], the mean maximum excursion method [20], and the fractionally integrated moving average (FIMA) framework [21] may efficiently replace the MSD estimator for classification purposes. Hidden Markov models (HMMs) have been proposed to check the heterogeneity within single trajectories [22,23]. They have proved to be quite useful in the detection of confinement [24]. Particle filtering may also be used to locate binding sites for the processes with transient confinement [25].

An alternative approach to an analysis of trajectories, rooted in computer science and statistics, is also possible. Due to algorithmic advances combined with increased data availability and more powerful computers, machine learning (ML) methods may already outperform human experts at some tasks including classification, i.e., the problem of identifying to which category a new observation belongs on the basis of a training data set containing observations with a known category membership. Since the detection of the motion falls into the domain of classification, one may try to tackle this problem with machine learning algorithms. This approach is very appealing, because it would enable an automated analysis of many hundreds or even thousands of trajectories with a reduced amount of manual intervention and initial data curation.

Several attempts to analyze SPT trajectories with ML methods have already been carried out. For instance, Monnier *et al.* [13] used a Bayesian approach to MSD-based

classification of motion modes. Dosset and co-workers [26] used a simple back-propagation neural network to discriminate between different types of diffusion. Wagner *et al.* [27] built a random forest classifier for normal, anomalous, confined and directed diffusion. Although each of these attempts uses a different machine learning classification algorithm, they all belong to the class of feature-based methods. Each trajectory within this approach is described by a set of human-engineered features, and only those features were provided as input to a classifier model.

In contrast, deep-learning methods extract features on their own from raw data, without any effort from human experts. They have gained popularity in recent years and have already been successfully applied to computer vision [28–30], speech recognition [31,32], and natural language processing [33,34]. Among the popular methods are convolutional neural networks (CNNs) [35], which excel in image classification. They have already been applied to single-particle recognition in microscopy experiments [36,37]. However, although some attempts to do time series analysis with CNNs are known [38–40], to the best of our knowledge they have not been applied yet to the problem of classification of motion types from raw trajectories.

Thus, the goal of this paper is to propose an approach to SPT trajectory classification based on the CNN deep-learning method and to compare its performance with two popular feature-based methods: random forests [41,42] and gradient boosting [43]. Since all of these methods require large training data sets with trajectories labeled already with a corresponding motion type, we will use synthetic data to train and validate the models. For the traditional methods, we will follow the approach of Wagner *et al.* [27] and use their set of features for classification purposes.

The paper is divided as follows. In Sec. II we introduce basic types of diffusion and briefly discuss the mean square displacement curves as a common tool of trajectory analysis. Classification methods are introduced in Sec. III. In Sec. IV we summarize methods for computer generation of synthetic trajectories. Features used by the traditional classification methods are introduced in Sec. V. Results of our analysis are presented in Sec. VI, followed by some concluding remarks.

II. DIFFUSION MODES AND THEIR ANALYSIS

We seek to classify SPT trajectories into four basic motion types: normal diffusion (ND) [8], directed motion (DM) [9–11], anomalous diffusion (AD) [12], and confined diffusion (CD) [13]. A standard way of identifying them is based on the analysis of the mean square displacement (MSD) of particles [44]. The MSD is defined as

$$\hat{\rho}(t) \equiv \langle [X(t) - X(0)]^2 \rangle = \frac{1}{M} \sum_{j=1}^M [X_j(t) - X_j(0)]^2, \quad (1)$$

where $X_j(t)$ is the position of the j th particle after time t and M is the number of particles (i.e., independent trajectories). MSD is an ensemble average of the square displacement over the probability distribution of $X(t)$. However, due to a limited number of trajectories in many single-particle tracking experiments, the ensemble averaged MSD is usually replaced

by the time-averaged MSD (TAMSD) calculated from a single trajectory. Given a trajectory in the form of N consecutive two-dimensional positions $X_i = (x_i, y_i)$ ($i = 1, \dots, N$) recorded with a constant time interval Δt , the TAMSD at time lag $n\Delta t$ is defined as

$$\rho(n\Delta t) = \frac{1}{N-n} \sum_{i=1}^{N-n} (X_{i+n} - X_i)^2. \quad (2)$$

It is worthwhile to mention that for an ergodic process with stationary increments the TAMSD converges to the ensemble averaged MSD in the limit $N \rightarrow \infty$.

According to Saxton [14], for the four basic modes of diffusion we have

$$\begin{aligned} \rho_{\text{ND}}(n\Delta t) &= 4Dn\Delta t, \\ \rho_{\text{AD}}(n\Delta t) &= 4D(n\Delta t)^\alpha, \\ \rho_{\text{DM}}(n\Delta t) &= 4Dn\Delta t + (vn\Delta t)^2, \\ \rho_{\text{CD}}(n\Delta t) &\simeq r_c^2 \left[1 - A_1 \exp\left(\frac{-4A_2 D n \Delta t}{r_c^2}\right) \right]. \end{aligned} \quad (3)$$

Here, $\alpha < 1$ is the anomalous exponent, v is the velocity in the directed motion, the constants A_1 and A_2 characterize the shape of the confinement, and r_c is the confinement radius.

For pure trajectories with no localization errors one could actually determine their diffusion modes simply based on the shapes of MSD curves and their mathematical models given by Eq. (3). However, in cases of real trajectories there is usually a lot of noise in the data, which makes the fitting of a mathematical model a challenging task, even in the simplest case of the normal diffusion [11]. Moreover, according to Eq. (2), only the MSD values corresponding to small time lags are well averaged. The larger the lag, the smaller is the number of displacements contributing to the averages, resulting in fluctuations increasing with the lag. This constitutes a problem, in particular in cases of short trajectories, for which the fit to mathematical models has to be limited to just the first few time lags. This is the reason why we are interested in classification methods that go beyond fitting of mathematical models to the MSD curves.

III. CLASSIFICATION METHODS

Traditional machine learning is a set of methods of statistical learning where each instance in a data set is described by a set of human-engineered features or attributes [45]. In contrast, deep-learning methods extract features from raw data without any effort from human experts [46]. The representation of data is constructed automatically and there is no need for complex data preprocessing as in the case of machine learning.

The deep-learning approach constitutes nowadays the state-of-the-art technology for automatic data classification and overshadows a little bit the classical machine learning algorithms. However, in some specific situations the latter ones are still better to use. The reasons are at least threefold: they work better on small data, are financially and computationally cheaper, and usually are easier to interpret. Thus the ultimate goal of this paper is to compare the performance of machine and deep-learning algorithms applied to the recognition of

the diffusion type in single-particle tracking data. We will examine two classical algorithms, i.e., random forests [41,42] and gradient boosting [43], together with convolutional neural networks (CNNs) [35].

A. Feature-based methods

Both random forests and gradient boosting algorithms belong to the class of ensemble learning, i.e., methods that generate many classifiers and aggregate their results. In both cases, decision trees [47] are used as the basic classifier.

Decision trees are used very often for classification purposes, because they are easy to understand and interpret. And they usually do not require data preprocessing. However, they are unstable in the sense that a small variation in the data may lead to a completely different tree [48]. And they have the tendency to overfit, i.e., they correspond closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably [49]. Although methods like pruning are known to avoid overfitting, it is the main reason why decision trees are used as building blocks of ensemble classifiers rather than standalone ones.

1. Random forests

In a random forest, several decision trees are constructed from the same training data. For a given input, the predictions of individual trees are aggregated and then their mode is output as the class of the input data. A modern version of the algorithm combines the bagging idea proposed by Breiman [42] with the random subspace method invented by Ho [41,50]. Bagging repeatedly selects a random sample with replacement of the training set and fits trees to these samples. In order to avoid correlations between the trees, for each one a random subset of features is selected. Typically, in a classification problem with N features, \sqrt{N} of them are used to build one tree.

2. Gradient boosting

In contrast to random forests, the trees in gradient boosting are not independent. Instead, the single classifiers are built sequentially by learning from mistakes committed by the ensemble [43,51] (see Fig. 1 for a schematic comparison of the two methods).

B. Deep-learning methods

Deep-learning (DL) methods operate on raw data. They do not require any feature selection and extraction carried out by a human expert. Instead, they use a cascade of multiple layers of nonlinear processing units for feature identification, extraction, and transformation in order to learn multiple levels of data representations [52].

In this paper, we are going to use convolutional neural networks for trajectory classification. They have already been successfully applied to many tasks, including a time series analysis [53]. A schematic architecture of a CNN is shown in Fig. 2. Such a network has usually two components. The one consisting of hidden layers is responsible for extraction of features from the raw input data. The layers will perform a series of convolutions and pooling operations, during which

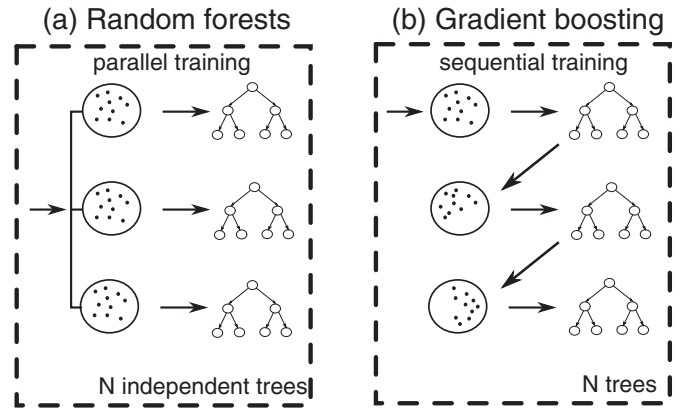


FIG. 1. Comparison between (a) random forest and (b) gradient boosting methods. In the random forest, N independent learners (trees) are built in parallel from random subsets of the input data set. In gradient boosting, the next tree is constructed from the pseudoresiduals of the ensemble and added to it.

attributes of data are detected. Each convolution uses a different filter which is sliding over the input and producing its own feature map in the form of a three-dimensional (3D) array. All the maps are then combined together as the final output of the component. The role of pooling is to reduce the dimensionality of feature maps in order to decrease the number of parameters and computations in the network. The classification part contains few fully connected layers, as in a regular neural network [54]. Flattening of data is usually required at the interface between the components, because the fully connected layers can process only 1D vectors.

IV. SYNTHETIC DATA

All three methods described in the previous section belong to the class of supervised learning, i.e., they infer a model from a set of training examples [55]. Each sample is a pair consisting of an input object (a trajectory) and a desired output value (a diffusion mode). The model is a function that maps an input to an output and can be used for classification of new input data.

Since thousands of labeled trajectories are needed to train the classifiers, especially in the deep-learning case, we will use computer-generated synthetic 2D trajectories as our training set. Simulation methods for every type of diffusion will be briefly discussed below.

A. Normal diffusion

According to Michalet [11], the probability distribution of the displacement's norm in the case of the normal diffusion is given by

$$F_d(u) = \frac{2u}{4D\Delta t} \exp\left(\frac{-u^2}{4D\Delta t}\right), \quad u \geq 0 \quad (4)$$

where Δt is the time interval during which the displacement is recorded. Mathematically, Eq. (4) is a Rayleigh distribution [56]. To simulate a trajectory, we randomly choose a start position of a particle and a random direction of displacement α and then pick up a random step length d from Eq. (4). Then

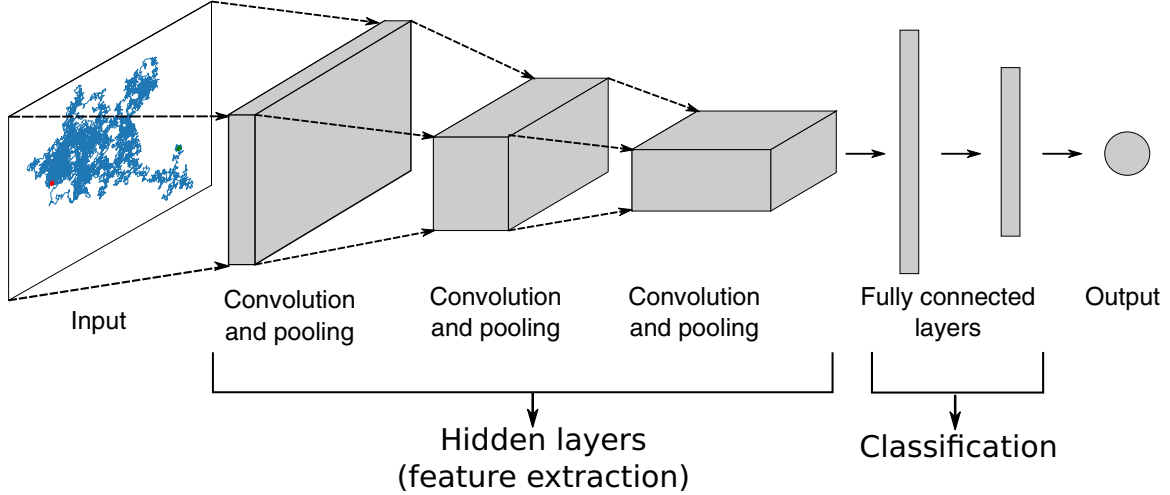


FIG. 2. A schematic architecture of a CNN. The network consists of two components: hidden layers responsible for feature extraction from input data and fully connected layers carrying out the classification.

we calculate the new position of the particle and take it as the starting point for the next step. The procedure is repeated a given number of times.

B. Directed motion

Once we have a procedure generating a normal diffusion trajectory, simulation of the directed motion is straightforward. For a given velocity \vec{v} , in each step we simply calculate a correction to the position due to the active motion,

$$\begin{aligned} dx_i &= v \Delta t \cos \beta, \\ dy_i &= v \Delta t \sin \beta, \end{aligned} \quad (5)$$

and add it to the new coordinates:

$$\begin{aligned} x_{\text{new}} &= x_{\text{old}} + d \cos \alpha + dx_i, \\ y_{\text{new}} &= y_{\text{old}} + d \sin \alpha + dy_i. \end{aligned} \quad (6)$$

The angle β in Eq. (5) is the direction of the velocity.

Following Wagner *et al.* [27], we may want to introduce a measure of how a trajectory is influenced by the active motion,

$$R = \frac{v^2 T}{4D}, \quad (7)$$

with T being the time duration. This measure can be helpful in generating similar trajectories with different values of v and D .

C. Confined diffusion

A small modification of the normal diffusion procedure is needed to simulate confined diffusion [27]. We assume that a particle starts from the center of a 2D circular reflective boundary. We divide every step of the simulation into 100 substeps with $\Delta t' = \Delta t/100$. In every substep we carry out a normal diffusion step. The position of the particle after the substeps will be updated only if the distance from the center to new coordinates is smaller than the radius of the reflective boundary.

Wagner *et al.* [27] have introduced the boundedness parameter B , defined as the area of the smallest ellipse enclosing a

normal diffusion trajectory (with no confinement) divided by the area of the confinement,

$$B = \frac{A_{\text{ellipse}}}{\pi r_c^2} \simeq \frac{DN \Delta t}{r_c^2}. \quad (8)$$

As in the case of the directed motion, this parameter will help to evaluate trajectories independently of the actual values of D and r_c .

D. Anomalous diffusion

Anomalous diffusion was simulated with the fractional Brownian motion (FBM) [57]. FBM is a continuous-time Gaussian process $B_H(t)$ on $[0, T]$ that starts at zero, has expectation zero for all $t \in [0, T]$, and its covariance function is given by

$$E[B_H(t)B_H(s)] = \frac{1}{2}(|t|^{2H} + |s|^{2H} - |t-s|^{2H}), \quad (9)$$

where the Hurst index H is a real number in $(0,1]$. There is a simple relation between H and the anomalous diffusion exponent α introduced in Eq. (3), namely

$$2H = \alpha. \quad (10)$$

The value of the Hurst index determines the type of motion generated by the process. $H = 1/2$ (i.e., $\alpha = 1$) corresponds to normal diffusion. For $H > 1/2$ ($\alpha > 1$), the increments of FBM are positively correlated, resulting in a superdiffusion. Finally, negative correlations between FBM increments occur for $H < 1/2$ ($\alpha < 1$), leading to subdiffusion. We focus on the last case in this work.

We used a dedicated Python package called `fbm` to simulate the fractional Brownian motion [58]. The Davies-Harte algorithm [59] was utilized to generate independent trajectories of the process.

E. Adding noise

Real trajectories can be altered by various measurement noises such as localization errors, electronic noise, drift or vibrations of the sample, or postprocessing errors [60]. To account for these issues, we added normal Gaussian noise

with zero mean and standard deviation σ to each simulated position.

Let us first introduce two different signal to noise ratios (SNRs): one for ND, AD, and CD,

$$Q_1 = \frac{\sqrt{D\Delta t}}{\sigma}, \tag{11}$$

an another one for DM,

$$Q_2 = \frac{\sqrt{D\Delta t + v^2\Delta t^2}}{\sigma}. \tag{12}$$

Instead of setting σ directly in our simulations, we will prefer to set a random level of SNR first and then to calculate the standard deviation for given D and Δt from one of the above equations.

F. Simulation details

Our training data consists of 20 000 synthetic trajectories, i.e., 5000 for each diffusion type. Following Wagner *et al.* [27] we used fixed values for two of the parameters: $\Delta t = 1/30$ s and $D = 9.02 \mu\text{m}^2/\text{s}$. The time lag is a typical value in experimental setups. The value of the diffusion coefficient D corresponds to a freely diffusing nanoparticle with a diameter 50 nm in water at 22 °C. Other parameters were chosen randomly. Their values are summarized in Table I. We used our own codes written in Python to simulate the training set. The codes are available upon request.

V. FEATURE EXTRACTION

We will follow the approach of Wagner *et al.* [27] and use their nine features together with the diffusion coefficient fitted

$$\mathbf{T} = \begin{pmatrix} \frac{1}{N} \sum_{j=1}^N (x_j - \langle x \rangle)^2 & \frac{1}{N} \sum_{j=1}^N (x_j - \langle x \rangle)(y_j - \langle y \rangle) \\ \frac{1}{N} \sum_{j=1}^N (x_j - \langle x \rangle)(y_j - \langle y \rangle) & \frac{1}{N} \sum_{j=1}^N (y_j - \langle y \rangle)^2 \end{pmatrix}, \tag{13}$$

where $\langle x \rangle = (1/N) \sum_{j=1}^N x_j$ is the average of x coordinates over all steps in the random walk. We will define the asymmetry as [62]

$$A = -\ln \left(1 - \frac{(\lambda_1 - \lambda_2)^2}{2(\lambda_1 + \lambda_2)} \right), \tag{14}$$

where λ_1 and λ_2 are the principle radii of gyration, i.e., the eigenvalues of the tensor \mathbf{T} .

D. Efficiency

Efficiency relates the net squared displacement of a particle to the sum of squared step lengths,

$$E = \frac{|X_{N-1} - X_0|^2}{(N - 1) \sum_{i=1}^{N-1} |X_i - X_{i-1}|^2}. \tag{15}$$

TABLE I. Parameters of the simulation and their values. All parameters except Δt and D were randomly chosen from given ranges.

Parameter	Meaning	Range of values
Δt	time lag between steps	1/30 s
D	diffusion coefficient	9.02 $\mu\text{m}^2/\text{s}$
N	length of a trajectory	30–600
B	boundedness	1–6
R	active motion to diffusion ratio	1–17
α	anomalous exponent	0.3–0.7
Q	signal to noise ratio	1–9

from the data as the tenth one. In this section we will give a short description of the features used for training of our classifiers.

A. Diffusion coefficient

We will use the diffusion coefficient of the model given by the first of Eqs. (3) fitted to the mean square displacement curve estimated by Eq. (2).

B. Anomalous exponent

Anomalous exponent α is the exponent in the second model defined in Eqs. (3). Again, it will be fitted to the MSD curve obtained from Eq. (2).

C. Asymmetry

The asymmetry of a trajectory can be used to detect directed motion. Following Saxton [61], we will derive it from the gyration tensor, which describes the second moments of positions of a particle. For a 2D random walk of N steps it is given by

It is a measure for linearity of a trajectory and, like asymmetry, it may help to detect directed motion.

E. Fractal dimension

The fractal dimension is a measure of the space-filling capacity of a pattern. According to Katz and George [63], the fractal dimension of a trajectory can be calculated as

$$D_f = \frac{\ln N}{\ln(NdL^{-1})}, \tag{16}$$

where L is the total length of the path, N is the number of steps, and d is the largest distance between any two positions.

The measure takes values around 1 for straight trajectories (direct motion), around 2 for random ones (normal diffusion), and around 3 for constrained trajectories (confined or anomalous diffusion) [63].

F. Gaussianity

A trajectory's Gaussianity was introduced by Ernst *et al.* [64] to check the Gaussian statistics on increments,

$$g(n) = \frac{\langle r_n^4 \rangle}{2\langle r_n^2 \rangle^2}, \quad (17)$$

where the trajectory's quartic moment is given by

$$\langle r_n^4 \rangle = \frac{1}{N-n} \sum_{i=1}^{N-n} |X_{i+n} - X_i|^4. \quad (18)$$

For normal diffusion we should get Gaussianity equal to 0. Since we used FBM, which has Gaussian increments, to simulate anomalous diffusion, we expect to get the same result for AD. The other types of motion should show deviations from 0.

G. Kurtosis

Kurtosis measures the asymmetry and peakedness of the distribution of points within a trajectory [62]. For its calculation the position vectors X_i are projected onto the dominant eigenvector \vec{r} of the gyration tensor (13), yielding scalars

$$x_i^p = X_i \cdot \vec{r}. \quad (19)$$

Kurtosis is defined as the fourth moment of the set of x_i^p ,

$$K = \frac{1}{N} \sum_{i=1}^N \frac{(x_i^p - \bar{x}^p)^4}{\sigma_{x^p}^4}, \quad (20)$$

with \bar{x}^p being the mean projected position and σ_{x^p} the standard deviation of x^p .

H. MSD ratio

The mean square displacement ratio characterizes the shape of the MSD curve. We will define it as follows:

$$\kappa(n_1, n_2) = \frac{\langle r_{n_1}^2 \rangle}{\langle r_{n_2}^2 \rangle} - \frac{n_1}{n_2}, \quad (21)$$

where $n_1 < n_2$. Taking Eq. (3) into account, we see that $\kappa = 0$ for normal diffusion, negative for direct motion, and positive for other types of diffusion. In our analysis we simply took $n_2 = n_1 + \Delta t$ and calculated an averaged ratio for every trajectory.

I. Straightness

Straightness is a measure of the average direction change between subsequent steps. Similar to efficiency it relates the

net displacement to the sum of step lengths:

$$S = \frac{|X_{N-1} - X_0|}{\sum_{i=1}^{N-1} |X_i - X_{i-1}|}. \quad (22)$$

J. Trappedness

Trappedness is the probability that a diffusing particle with the diffusion coefficient D and traced for a time interval t is trapped in a bounded region with radius r_0 . According to Saxton [61] it can be estimated by

$$P(D, t, r_0) = 1 - \exp \left[0.2045 - 0.25117 \left(\frac{Dt}{r_0^2} \right) \right]. \quad (23)$$

Since the radius r_0 is usually not known, we will approximate it by half of the maximum distance between any two positions along a given trajectory. For D , we will take its short-time estimate, fitted to the first two points of the MSD curve.

VI. RESULTS

We decided to use existing machine learning libraries within this project. Random forest and gradient boosting implementations available in `scikit-learn` [65], the most popular ML learning library in Python, were used to build the feature-based classifiers. And we used `mcfly` [66], a deep-learning library for time series processing, to find and train a deep classifier working with raw diffusion data. All codes were written in Python and are available on request. The computations were carried out on a cluster of 24 CPUs (2.6 GHz each) with a total memory of 50 GB. If not stated otherwise, the synthetic trajectories were randomly split into two subsets: a training set containing 70% of them and a test set.

A. Featured-based classification

The random forest classifier implemented in `scikit-learn` follows the original paper by Breiman [42]. The gradient boosting algorithm available in this library is described in Refs. [43]. The parameters of the models were optimized with a randomized search method (the `RandomizedSearchCV` function in `scikit-learn`). They are summarized in Table II.

1. Accuracy

One of the basic metrics used to assess the performance of classification models is accuracy, defined as the number of correct predictions divided by the total number of predictions.

TABLE II. Optimal parameters for the random forest and gradient boosting models trained on our data. A randomized search method was used to determine those values.

	Random forest	Gradient boosting
Number of trees	500	500
Maximum depth of a single tree	20	10
Minimum number of samples required to split an internal node	2	5
Minimum number of samples required to be at a leaf node	1	4

TABLE III. Accuracies of the feature-based classifiers.

	Random forest	Gradient boosting
Single split of data	96.43%	96.97%
Tenfold cross-validation	96.23%	96.47%

In Table III, accuracies for both feature-based classifiers are shown. The numbers in the first row correspond to the accuracy achieved after a single random split into training and test sets. For the second row we used the tenfold cross-validation method. The idea behind this technique is to randomly split the data set into ten folds without replacement and use nine of them for training and one for testing of the model. The procedure is repeated ten times, so we obtain ten models and accuracy estimates. An average of those estimates gives the overall accuracy.

Since in the gradient boosting an ensemble of the decision trees is built with the purpose of reducing the total error, we would expect that the algorithm performs much better than the random forest. From Table III it follows that its accuracy is indeed higher, but the differences are actually negligible. Both classifiers perform excellently, with an average accuracy of more than 96%.

In Fig. 3, confusion matrices of the classifiers are presented. In both cases the classifiers made a total of 6000 predictions (sum of all matrix elements), including 1500 for each type of diffusion (sum of all elements in a matrix row). As far as the random forest model is concerned, the best performance was observed for the directed diffusion: among the 1500 directed trajectories only one was wrongly classified as an anomalous one. The performance decays slightly for the anomalous and confined modes and is significantly worse for the normal diffusion. The gradient boosting model reveals similar characteristics with slightly different absolute numbers.

The data collected in the confusion matrices may be used to generate a more detailed description of the performance of the models under investigation. The results are briefly summarized in Table IV. Here, we adopted two quantities commonly used in classification problems: precision and

TABLE IV. A brief summary of the performance of feature-based classifiers. All results are rounded to two decimal digits.

	Random forest			Gradient boosting		
	Precision	Recall	Support	Precision	Recall	Support
Anomalous	0.98	0.98	1500	0.98	0.99	1500
Confined	0.94	0.96	1500	0.94	0.96	1500
Directed	1.000	1.00	1500	1.00	1.00	1500
Normal	0.94	0.92	1500	0.95	0.93	1500
Average, total	0.96	0.96	6000	0.97	0.97	6000
Accuracy	96.43%			96.97%		

recall [67]. Precision is the fraction of correct predictions among all predictions. It tells us how often a classifier is correct if it predicts a given class. Recall is the fraction of correct predictions of a given class over the total number of members of this class. Despite small differences in the numbers, each of our models is characterized by both very high precision and recall. Thus, they not only return much more relevant results than the irrelevant ones (high precision), but also yield most of the relevant results (high recall).

2. Feature importance

A nice detail of the ensemble classification methods is that they usually allow one to easily compute the relative importance of features for a given problem. Variables with high importance scores are the drivers of the outcome, and their values have a significant impact on the correctness of a prediction. Features with low importance might usually be omitted from a model, making it faster to fit and predict.

The `scikit-learn` implementations of the random forest and gradient boosting classifiers calculate the importance values on the fly during the training process and provide an interface to access them. Results are shown in Fig. 4. The linear diffusion coefficient D seems to be the most important feature in both cases, followed by the MSD ratio, straightness, efficiency, and the anomalous exponent α . There are some differences between the methods as well. For instance, the dominance of the diffusion coefficient over all other features is less

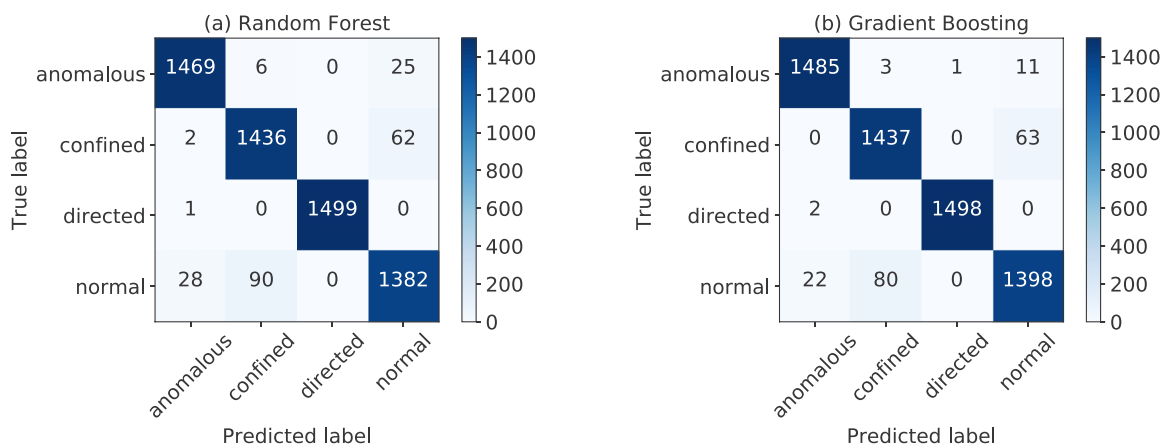


FIG. 3. Confusion matrices for (a) the random forest and (b) the gradient boosting classifiers.

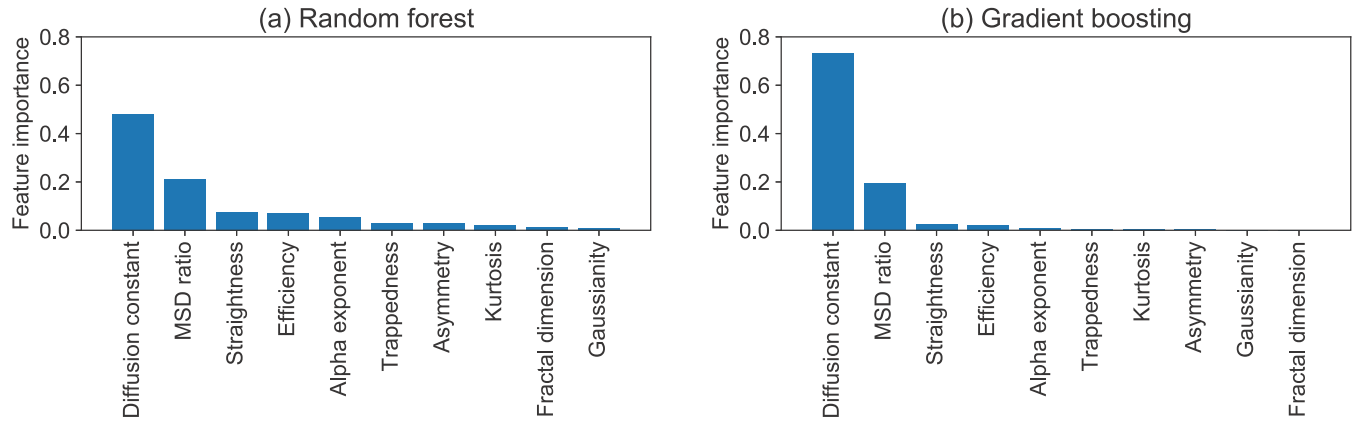


FIG. 4. Feature importance in (a) the random forest model and (b) the gradient boosting models.

pronounced in the random forest. Instead, we observe nonvanishing importance of the remaining features, i.e., trappedness, asymmetry, kurtosis, fractal dimension, and Gaussianity. In contrast, the importance values of those features are negligible in the gradient boosting case, and the difference between the first and the second rank is much higher.

To illustrate the differences between the models, in Fig. 5 we show the cumulative importance values of features. The dashed horizontal line in this plot is the 97% level of importance and could indicate a threshold for feature selection; i.e., once the level is reached, we can omit the remaining features without affecting the model very much. In order to find a value of the threshold, one should check how his model generalizes to unseen data after removing attributes for different thresholds and then chose the one not negatively affecting the accuracy of the model.

To elaborate on that issue, we first found the feature selection thresholds for cumulative importance levels ranging from 90% up to 99%. Then we trained both models again with the reduced number of features as indicated by the threshold. Results are presented in Fig. 6. As we see, gradient boosting reaches a given cumulative importance level with a smaller feature set than the random forest. Consequently, it requires fewer features to achieve high accuracies.

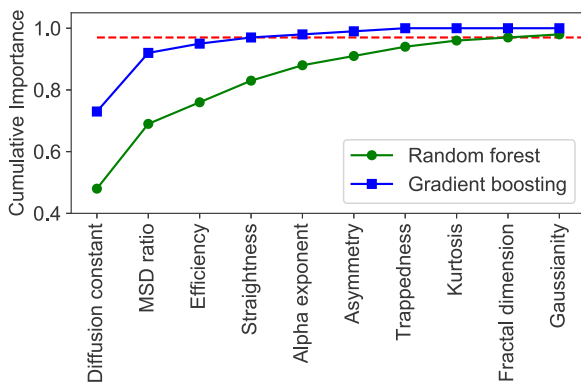


FIG. 5. Cumulative importance of features for both models. The dashed line is the 97% level of importance and indicates a threshold for feature selection.

B. Deep-learning classification

The `mcfly` package [66] used in this work for the deep-learning approach is a piece of software tailor made to a classification of time series. One of its biggest advantages is a low entry level, because it does not require a user to define exactly the architecture of a convolutional neural network and to provide all hyperparameters of the model. Instead, it carries out a search over suitable architectures and their hyperparameters to find the best performing model. Since a diffusion trajectory is nothing but a multichannel time series (2D or 3D, depending on the problem at hand), it should match the requirements of `mcfly`.

1. CNN architecture

The first step in the development of a deep-learning model is to create its architecture, i.e., to specify the following set of hyperparameters: (i) the learning rate, (ii) the regularization rate, (iii) the number of convolutional layers, (iv) the number of filters in each convolutional layer, and (vi) the number of hidden nodes (in dense layers). The learning rate scales the magnitude of weight updates in the training process in order to minimize the network's loss function. The regularization helps to prevent overfitting of the network.

As it is not known *a priori* which architecture will be optimal for classification of SPT trajectories, we performed a random search to find the best one. This procedure simply creates a number of models at random, trains them on a relative small subset of the training data and then checks how good they are. Different criteria for selecting the best model are possible. The accuracy on a validation set will be used in this work.

Once the best model is chosen, it should be trained again on the full training data set. One full pass of the data through the network in the process of training its weights is called an epoch. Usually, many epochs are required to achieve a combination of the weights that yields a good accuracy.

We used `mcfly`'s function `find_best_architecture` to perform the random search in the hyperparameter space. We had to specify only two input parameters: the number of architectures for the random search procedure and the number of epochs for training the final model. Regarding the number of architectures, we expect intuitively that the bigger it is, the

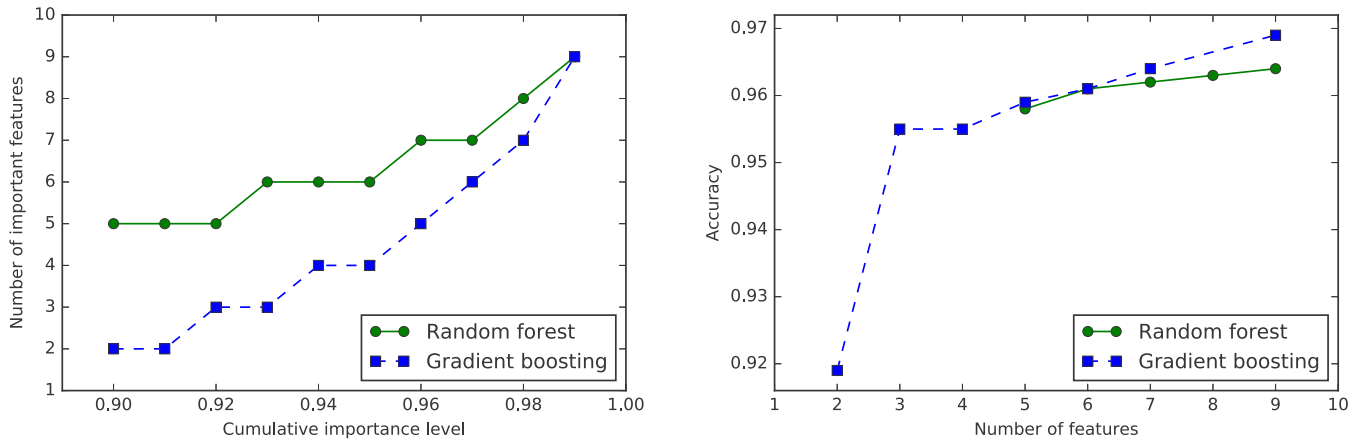


FIG. 6. Comparison of the feature-based classifiers. (a) Number of features required to achieve the given threshold of cumulative importance. (b) Accuracy of classifiers trained with the reduced number of features.

better. This is simply due to the fact that more architectures cover a larger part of the hyperparameter space. Similarly, more epochs should guarantee a better convergence of the weights to the combination which minimizes the network’s loss. Unfortunately, increasing the values of each of these parameters leads to significantly longer computing times, because the evaluation of an additional model as well as an epoch of training of the final model are very time consuming processes. Therefore, the choice of the values is usually a tradeoff between the targeted accuracy and the computation time.

In order to determine the right values of the input parameters, we checked their impact on the loss and the accuracy of the final model as well as on the total execution time. Results are shown in Fig. 7. The analysis of the number of architectures (left column) was performed for 10 epochs on the training subset. For the epochs (right column), the final model was selected from 20 initial architectures. First of all, we observe an almost monotonic growth of the execution time with the increase of both input parameters. As expected, the accuracy of the network increases with the number of architectures (middle left panel in Fig. 7). However, it remains practically constant for values between 20 and

40 architectures. Thus, 20 architectures will be used in our further investigation, as it seems to be a good compromise between the accuracy and the execution time.

The behavior of the accuracy of the model as a function of the number of epochs (middle right panel in Fig. 7) is more interesting. We see that, starting from 30 epochs, the difference between the accuracies on the training and test data sets increases. This is an indicator that the model overfits, i.e., it starts to learn the noise in the training data as an important concept, which does not apply to the new trajectories from the test set. Since there is a local maximum in the accuracy at 20 epochs, we will use this value in the following.

To summarize, our final model is the result of the random search among 20 architectures, trained for 20 epochs on the full training data set. Its architecture is shown in Fig. 8. It consist of 6 convolutional layers and 2 dense ones. Besides those building blocks, there are several others elements of the model: (i) activation layers which define the output of neurons given an input or set of inputs, (ii) batch normalization layers responsible for normalization of the activation of previous layers, (iii) flattening layers, which flatten the input without changing its size (required by the dense layers). The values

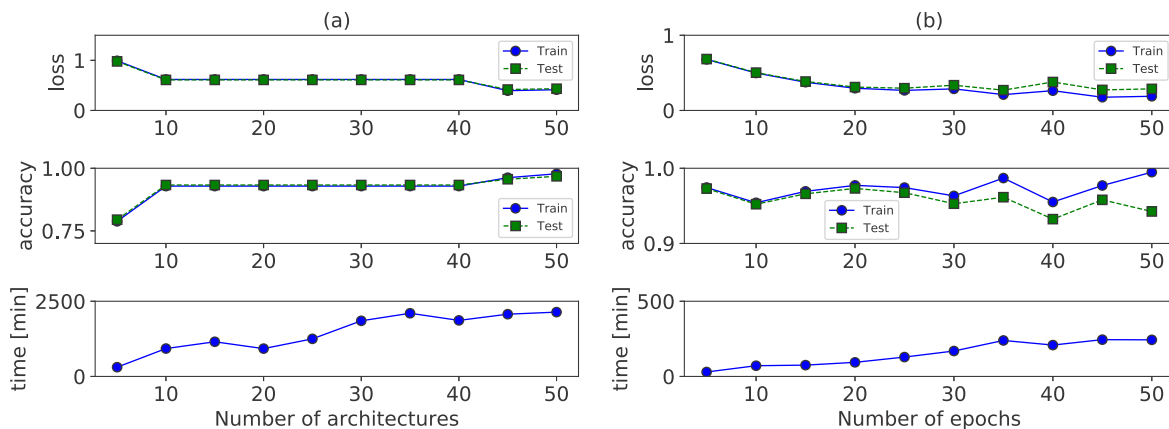


FIG. 7. Impact of (a) the number of architectures in the random search and of (b) the number of epochs in training the final model on the loss, accuracy, and execution time. In the random search procedure, 10 epochs were used to train the models on a small subset of the training data. The final model analyzed in the right column was selected from 20 architectures.

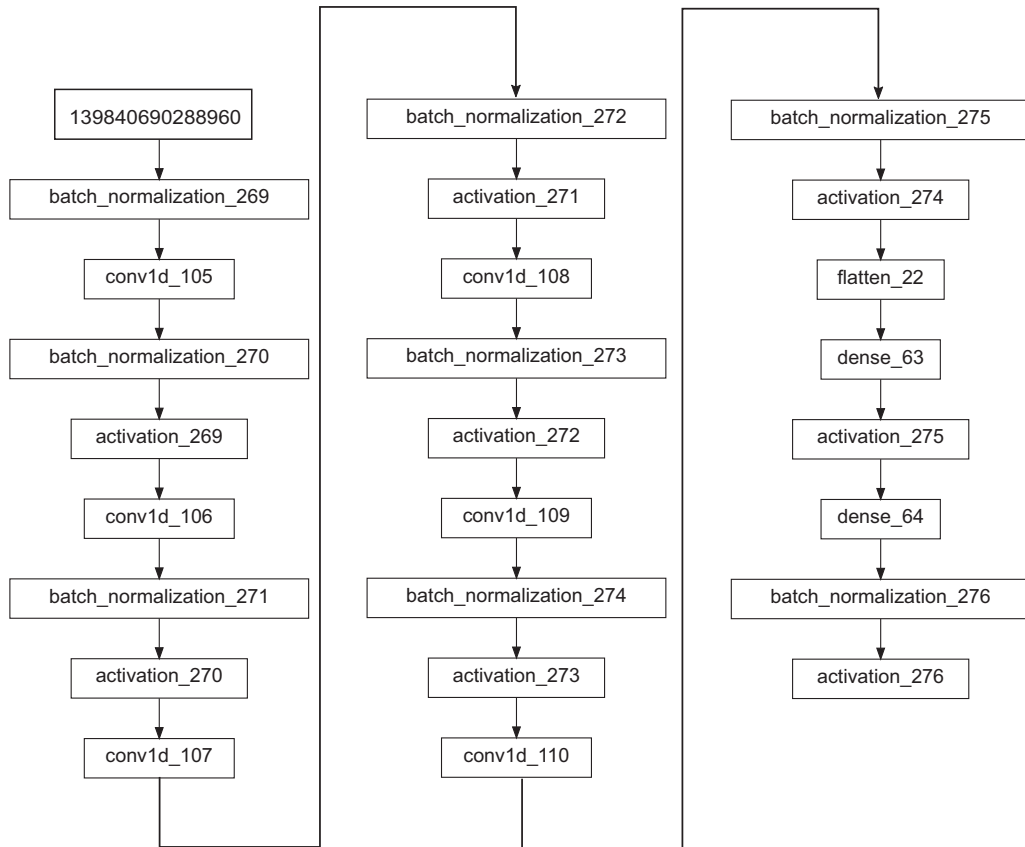


FIG. 8. Architecture of the best performing network model found by mcfly.

of the most important hyperparameters of this model are summarized in Table V.

2. Accuracy of CNN

The confusion matrix of our CNN model is shown in Fig. 9 and its performance is summarized in Table VI. Its overall accuracy turns out to be slightly better than the one of feature-based methods (see Table IV for comparison). Again, the model not only returns much more relevant results than irrelevant ones (high precision), but also yields most of the relevant results (high recall). The best performance is observed for the directed motion and the anomalous diffusion. It decays slightly for the confined diffusion in terms of precision and for the normal diffusion in terms of recall.

TABLE V. Hyperparameters of the best performing network model shown in Fig. 8.

Parameter	Value
Regularization rate	0.0014064205292043147
Number of conv. layers	6
Number of filters	[49, 36, 18, 83, 90, 27]
Learning rate	0.00021795428728036654
Hidden nodes in dense layers	1550

C. Feature-based versus deep-learning approach

Let us first juxtapose accuracies of the methods analyzed in this paper together with the required processing times. All computations were carried out on a cluster of 24 CPUs (2.6 GHz each) with 50 GB of the total memory. Results are collected in Table VII. As already pointed out in the previous sections, the deep-learning approach has a slightly higher accuracy than the feature-based methods, but at the cost of significantly longer processing times. Thus, if the time to

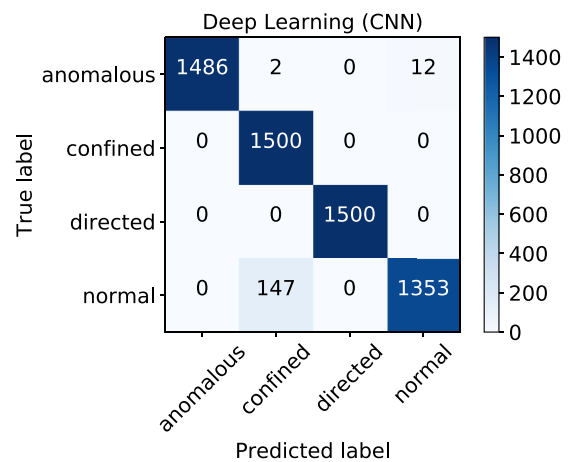


FIG. 9. Confusion matrix of the CNN classifier.

TABLE VI. A brief summary of the performance of the CNN classifier. All results are rounded to two decimal digits.

	Precision	Recall	Support
Anomalous	1.00	0.99	1500
Confined	0.91	1.00	1500
Directed	1.00	1.00	1500
Normal	0.99	0.90	1500
Average, total	0.98	0.97	6000
Accuracy	97.30%		

train the model constitutes an issue, one should rather go for gradient boosting.

It would be interesting to see how the methods perform in some limiting cases, in which we expect them to fail anyway. For instance, an anomalous diffusion with the exponent α approaching 1 should be practically indistinguishable from a normal diffusion. The same holds for a directed motion with small velocities. To elaborate on that issue, we first generated four separate validation sets for the anomalous diffusion. Each set contained 1500 trajectories with the values of α randomly chosen from a corresponding interval: $\alpha^{(1)} \in (0.55, 0.65)$ for the first set, $\alpha^{(2)} \in (0.65, 0.75)$ for the second one, $\alpha^{(3)} \in (0.75, 0.85)$ for the third one, and finally $\alpha^{(4)} \in (0.85, 0.95)$. Then we classified those sets with all three methods by making use of the already trained models. Results are shown in the left panel of Fig. 10. In case of the feature-based methods we observe an almost linear decrease of the average accuracy with increasing α . The CNN method performs better and the decrease is slower for $\alpha < 0.85$. However, in the interval close to the limiting value ($\alpha = 1$), there is a sudden drop in the performance of CNN and the deep-learning approach starts to be the worst one.

We did a similar analysis for the directed motion with small velocities. This time, we generated only one additional validation set with $R \in \{1, 2, 3\}$ [see Eq. (7) for the definition of R]. Again, we classified it with all methods. Results are shown in the right panel of Fig. 10. The CNN method turned out to be the best one, followed by the gradient boosting.

TABLE VII. Comparison of all three classification methods. The processing time is understood as data preparation (if required), feature extraction (if required), searching for best performing model, and finally training and validation of the classifier. A cluster of 24 CPUs with 50 GB total memory was used to perform the computations

	Feature based		Deep learning
	Random forest	Gradient boosting	CNN
Accuracy	96.43%	96.97%	97.30%
Processing time	1 h, 26 min	1 h, 9 min	3 d, 5 h, 50 min

Although the random forest still performs reasonably, there is already a noticeable gap in the accuracy to the other methods.

In this paper, we used four basic models of diffusion to generate artificial training data. However, they are not exhaustive and other models are possible for a given type of diffusion. For instance, FBM with $\alpha < 1$ is not the only model that produces subdiffusive trajectories. Continuous time random walks (CTRW) with heavy-tailed waiting times [68] or fractional Levy stable motion [69] are known to have the characteristics of subdiffusion. Similarly, FBM with $\alpha > 1$ [69] or CTRW with long-tailed spatial distribution [70] are, alongside the directed motion model, examples of a superdiffusive process.

Now one may ask, for instance, if a classifier trained on the directed motion model as the only expression of superdiffusion will recognize trajectories generated with other superdiffusive models. To check that, we took FBM with $\alpha \in (1.3, 1.7)$ to generate an additional validation set consisting of 5000 trajectories. Results of their classification with our models are shown in Table VIII. We see that all methods perform really poorly in this test. Only 28% of the trajectories are classified as directed motion (i.e., superdiffusion) by gradient boosting, 25% by random forest, and only 0.2% by CNN. Thus, the models do not generalize well to unseen models, even though they are supposed to produce the same diffusion type as the ones used for training. This constitutes an issue in

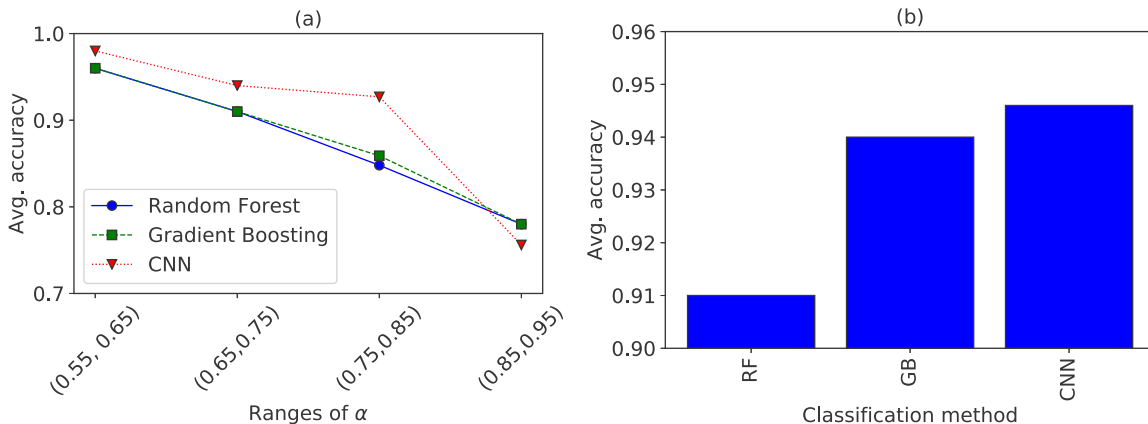


FIG. 10. Average accuracy of the methods in some limiting cases. (a) Performance of the classifiers for anomalous diffusion for four different ranges of the exponent α . The lines in the plot are used to guide the eye. (b) Same for directed motion with small velocities, corresponding to $R \in \{1, 2, 3\}$ [see Eq. (7) for the definition of R].

TABLE VIII. Classification results for superdiffusive trajectories generated with a model other than the directed motion used to train the classifiers. In this particular example, FBM with $\alpha \in (1.3, 1.7)$ was used to prepare the validation set.

	Anomalous	Confined	Directed	Normal
Random forest	206	399	1231	3164
Gradient boosting	545	784	1380	2291
CNN	0	33	9	4958

applications to real SPT data. Since it is rather impossible to provide a large set of experimental trajectories already labeled with correct diffusion types, we have to resort to artificially created training sets. As already shown, the machine learning methods work excellently on unseen and noisy data, but are not able to generalize well to unseen models. Therefore, we should include as many models as possible in our training sets in order to get some conclusive results for real trajectories.

VII. CONCLUSIONS

We proposed an approach to analysis of SPT trajectories that makes use of convolutional neural networks, i.e., one of the popular modern deep-learning methods. The biggest advantage of this approach is that it works with raw SPT data. It does not require any complex data preprocessing nor the extraction of human-engineered features from data in order to feed a classifier. Instead, it learns the features on its own from the trajectories.

Deep learning is seen already as the state-of-the-art classification method in many areas. From our results it follows

that, indeed, it has a slightly better accuracy than the traditional feature-based methods in most cases, but at the cost of significantly longer training times.

We have shown that more models of diffusive processes have to be taken into account before applying ML to classification of trajectories. All methods considered in this paper perform excellently on unseen data, provided it was generated with the models already used in the preparation of the training sets. Unfortunately, they fail to correctly classify trajectories produced with other models. Interestingly, CNN turned out to be the worst in this respect. Therefore, exhaustive data sets including as many models of diffusion as possible are needed to get conclusive classification results for real trajectories.

The excellent performance of the traditional methods observed in our experiments may be related to the fact that we assumed the movement of the particles to be homogeneous, i.e., one generated trajectory represents only one type of motion. In real experiments the type of the diffusion may change multiple times within one trajectory due to the interaction of the particle with the medium. To cope with that issue one usually divides the trajectory into short segments and then tries to classify each segment independently of the others. Classifiers trained on data with short lengths are required for that purpose. The CNN method could work better than the feature-based ones in this case, because most of the features relate to MSD estimates which are worse (much noisier) for short trajectories.

ACKNOWLEDGMENTS

H.L.-O. and J.S. were partially supported by NCN-DFG Beethoven Grant No. 2016/23/G/ST1/04083. Computations were performed on the BEM cluster in the Wrocław Center for Networking and Supercomputing (WCSS).

-
- [1] C. Manzo and M. F. Garcia-Parajo, *Rep. Prog. Phys.* **78**, 124601 (2015).
 - [2] H. Shen, L. J. Tauzin, R. Baiyasi, W. Wang, N. Moringo, B. Shuang, and C. F. Landes, *Chem. Rev.* **117**, 7331 (2017).
 - [3] D. Holcman, N. Hoze, and Z. Schuss, *Biophys. J.* **109**, 1761 (2015).
 - [4] C. Kural, H. Kim, S. Syed, G. Goshima, V. I. Gelfand, and P. R. Selvin, *Science* **308**, 1469 (2005).
 - [5] A. Yildiz, J. N. Forkey, S. A. McKinney, T. Ha, Y. E. Goldman, and P. R. Selvin, *Science* **300**, 2061 (2003).
 - [6] I. Izeddin, V. Récamier, L. Bosanac, I. I. Cissé, L. Boudarene, C. Dugast-Darzacq, F. Proux, O. Bénichou, R. Voituriez, O. Bensaude, M. Dahan, and X. Darzacq, *eLife* **3**, e02230 (2014).
 - [7] J. Mahowald, D. Arcizet, and D. Heinrich, *ChemPhysChem* **10**, 1559 (2009).
 - [8] S. B. Alves, G. F. de Oliveira, Jr., L. C. de Oliveira, T. P. de Silans, M. Chevrollier, M. Oriá, and H. L. de S. Cavalcante, *Physica A* **447**, 392 (2016).
 - [9] G. Ruan, A. Agrawal, A. I. Marcus, and S. Nie, *J. Am. Chem. Soc.* **129**, 14759 (2007).
 - [10] A. M. Bannunah, D. Villasaliu, J. Lord, and S. Stolnik, *Mol. Pharmaceutics* **11**, 4363 (2014).
 - [11] X. Michalet, *Phys. Rev. E* **82**, 041914 (2010).
 - [12] G. R. Kneller, *J. Chem. Phys.* **141**, 041105 (2014).
 - [13] N. Monnier, S.-M. Guo, M. Mori, J. He, P. Lénárt, and M. Bathe, *Biophys. J.* **103**, 616 (2012).
 - [14] M. J. Saxton and K. Jacobson, *Annu. Rev. Biophys. Biomol. Struct.* **26**, 373 (1997).
 - [15] E. Kepten, A. Weron, G. Sikora, K. Burnecki, and Y. Garini, *PLoS ONE* **10**, e0117722 (2015).
 - [16] G. Schütz, H. Schindler, and T. Schmidt, *Biophys. J.* **73**, 1073 (1997).
 - [17] R. Ferrari, A. Manfroi, and W. Young, *Physica D* **154**, 111 (2001).
 - [18] H. Ewers, A. E. Smith, I. F. Sbalzarini, H. Lilie, P. Koumoutsakos, and A. Helenius, *Proc. Natl. Acad. Sci. USA* **102**, 15110 (2005).
 - [19] S. Burov, S. M. A. Tabei, T. Huynh, M. P. Murrell, L. H. Philipson, S. A. Rice, M. L. Gardel, N. F. Scherer, and A. R. Dinner, *Proc. Natl. Acad. Sci. USA* **110**, 19689 (2013).
 - [20] V. Tejedor, O. Bénichou, R. Voituriez, R. Jungmann, F. Simmel, C. Selhuber-Unkel, L. B. Oddershede, and R. Metzler, *Biophys. J.* **98**, 1364 (2010).
 - [21] K. Burnecki, E. Kepten, Y. Garini, G. Sikora, and A. Weron, *Sci. Rep.* **5**, 11306 (2015).

- [22] R. Das, C. W. Cairo, and D. Coombs, *PLoS Comput. Biol.* **5**, e1000556 (2009).
- [23] P. J. Slator, C. W. Cairo, and N. J. Burroughs, *PLOS ONE* **10**, e0140759 (2015).
- [24] P. J. Slator and N. J. Burroughs, *Biophys. J.* **115**, 1741 (2018).
- [25] J. Bernstein and J. Fricks, *J. Theor. Biol.* **401**, 109 (2016).
- [26] P. Dosset, P. Rassam, L. Fernandez, C. Espenel, E. Rubinstein, E. Margeat, and P.-E. Milhiet, *BMC Bioinf.* **17**, 197 (2016).
- [27] T. Wagner, A. Kroll, C. R. Haramagatti, H.-G. Lipinski, and M. Wiemann, *PLoS ONE* **12**, e0170165 (2017).
- [28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, in *Proceedings of the Conference on Neural Information Processing Systems (NIPS12)* (NIPS Foundation, San Diego, 2012), pp. 1097–1105.
- [29] K. Simonyan and A. Zisserman, [arXiv:1409.1556](https://arxiv.org/abs/1409.1556).
- [30] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, in *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2014)*, June 2014, Columbus, OH (IEEE, Piscataway, NJ, 2014).
- [31] L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams, Y. Gong, and A. Acero, in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, May 2013, Vancouver (IEEE, Piscataway, NJ, 2013).
- [32] A. Graves, A. Mohamed, and G. Hinton, in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, May 2013, Vancouver (IEEE, Piscataway, NJ, 2013).
- [33] R. Collobert and J. Weston, in *Proceedings of the 25th International Conference on Machine Learning* (ACM, New York, 2008).
- [34] Y. Kim, in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (ACL, Stroudsburg, PA, 2014).
- [35] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Proc. IEEE* **86**, 2278 (1998).
- [36] Y. Zhu, Q. Ouyang, and Y. Mao, *BMC Bioinf.* **18**, 348 (2017).
- [37] E. Nehme, L. E. Weiss, T. Michaeli, and Y. Shechtman, *Optica* **5**, 458 (2018).
- [38] M. Långkvist, L. Karlsson, and A. Loutfi, *Pattern Recognit. Lett.* **42**, 11 (2014).
- [39] X. Qiu, L. Zhang, Y. Ren, P. N. Suganthan, and G. Amaratunga, in *Proceedings of 2014 IEEE Symposium on Computational Intelligence in Ensemble Learning (CIEL)* (IEEE, Piscataway, NJ, 2014), pp. 1–6.
- [40] J. C. B. Gamboa, [arXiv:1701.01887](https://arxiv.org/abs/1701.01887).
- [41] T. K. Ho, in *Proceedings of the Third International Conference on Document Analysis and Recognition*, Vol. 1 (IEEE Computer Society, Washington 1995).
- [42] L. Breiman, *Mach. Learn.* **45**, 5 (2001).
- [43] J. H. Friedman, *Comput. Stat. Data Anal.* **38**, 367 (2002).
- [44] H. Qian, M. P. Sheetz, and E. L. Elson, *Biophys. J.* **60**, 910 (1991).
- [45] T. Mitchel, *Machine Learning* (McGraw-Hill, New York, 1997).
- [46] N. Hatami, Y. Gavet, and J. Debayle, in *Proceedings of SPIE, Tenth International Conference on Machine Vision (ICMV 2017)*, edited by A. Verikas, P. Radeva, D. Nikolaev, and J. Zhou (SPIE, Bellingham, WA, 2018), p. 10696.
- [47] Y.-Y. Song and Y. Lu, *Shanghai Arch. Psychiatry* **27**, 130 (2015).
- [48] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning with Applications in R* (Springer, New York, 2013).
- [49] M. Bramer, *Principles of Data Mining*, 2nd ed. (Springer, New York, 2013).
- [50] T. K. Ho, *IEEE Trans. Pattern Anal. Mach. Intell.* **20**, 832 (1998).
- [51] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee, *Ann. Stat.* **26**, 1651 (1998).
- [52] L. Deng and D. You, *Found. Trends Signal Process.* **7**, 197 (2014).
- [53] J. B. Yang, M. N. Nguyen, P. P. San, X. L. Li, and S. Krishnaswamy, in *Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15* (AAAI Press, Palo Alto, CA, 2015), pp. 3995–4001.
- [54] M. Gardner and S. Dorling, *Atmos. Environ.* **32**, 2627 (1998).
- [55] S. Raschka, *Python Machine Learning* (Packt, Birmingham, UK, 2015).
- [56] A. Papoulis and S. U. Pillai, *Probability, Random Variables, and Stochastic Processes*, 4th ed. (McGraw Hill, Boston, 2002).
- [57] B. B. Mandelbrot and J. W. V. Ness, *SIAM Rev.* **10**, 422 (1968).
- [58] C. Flynn, FBM: Exact methods for simulating fractional Brownian motion (FBM) or fractional Gaussian noise (FGN) in Python, <https://github.com/crflynn/fbm>, accessed 21-02-2019.
- [59] R. B. Davies and D. S. Harte, *Biometrika* **74**, 95 (1987).
- [60] Y. Lanoiselée, G. Briand, O. Dauchot, and D. S. Grebenkov, *Phys. Rev. E* **98**, 062112 (2018).
- [61] M. J. Saxton, *Biophys. J.* **64**, 1766 (1993).
- [62] J. A. Helmuth, C. J. Burckhardt, P. Koumoutsakos, U. F. Greber, and I. F. Sbalzarini, *J. Struct. Biol.* **159**, 347 (2007).
- [63] M. J. Katz and E. B. George, *Bull. Math. Biol.* **47**, 273 (1985).
- [64] D. Ernst, J. Köhler, and M. Weiss, *Phys. Chem. Chem. Phys.* **16**, 7686 (2014).
- [65] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *J. Mach. Learn. Res.* **12**, 2825 (2011).
- [66] D. van Kuppevelt, C. Meijer, V. van Hees, P. Bos, J. Spaaks, M. Kuzak, F. Huber, J. Hidding, and A. van der Ploeg, mcfly: deep learning for time series, version v1.0.4, 2017, doi: [10.5281/zenodo.2541698](https://doi.org/10.5281/zenodo.2541698).
- [67] J. W. Perry, A. Kent, and M. M. Berry, *Am. Doc.* **6**, 242 (1955).
- [68] M. Magdziarz, A. Weron, K. Burnecki, and J. Klafter, *Phys. Rev. Lett.* **103**, 180602 (2009).
- [69] K. Burnecki and A. Weron, *Phys. Rev. E* **82**, 021130 (2010).
- [70] M. Magdziarz, R. Metzler, W. Szczotka, and P. Zebrowski, *J. Stat. Mech.: Theory Exp.* (2012) P04010.