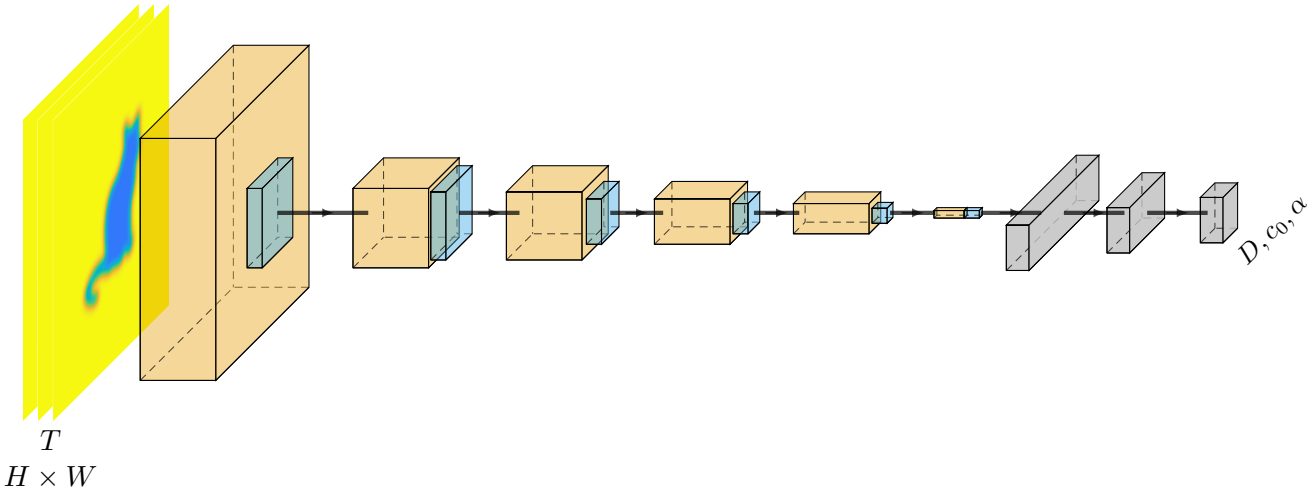
CHALMERS
UNIVERSITY OF TECHNOLOGY



Parameter Estimation in FRAP using Deep Learning

Temporal and Spatio-Temporal Models for Estimating the Diffusion Coefficient in Fluorescence Recovery After Photobleaching

Master's thesis in Complex Adaptive Systems

VICTOR WÅHLSTRAND SKÄRSTRÖM

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Parameter Estimation in FRAP using Deep Learning

Temporal and Spatio-Temporal Models for Estimating the Diffusion Coefficient in Fluorescence Recovery After Photobleaching

VICTOR WÅHLSTRAND SKÄRSTRÖM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2020

Parameter Estimation in FRAP using Deep Learning
Temporal and Spatio-Temporal Models for Estimating the Diffusion Coefficient in Fluorescence Recovery After Photobleaching
VICTOR WÅHLSTRAND SKÄRSTRÖM

Supervisor: Magnus Röding, Research Institutes of Sweden
Examiner: Mats Granath, Department of Physics, Chalmers University of Physics

Master's Thesis 2020
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of the downsampler spatio-temporal neural network applied to simulated FRAP data with a cat-shaped region of interest.

Typeset in LaTeX
Gothenburg, Sweden 2020

Parameter Estimation in FRAP using Deep Learning
VICTOR WÅHLSTRAND SKÄRSTRÖM
Department of Physics
Chalmers University of Technology

Abstract

Fluorescence recovery after photobleaching (FRAP) is a method used in optical microscopy for determining properties of diffusion in organic and inorganic solutions, including cells, membranes and gels. FRAP may be used to determine parameters such as the diffusion coefficient and binding rates of particles in a sample, and is used in wide-ranging disciplines such as the medicine, soft materials and food science. In FRAP, fluorescently labelled particles in a sample are bleached in a region of interest using a high-intensity laser. The recovery of the mean fluorescence intensity in the region of interest is known as the recovery curve. Conventional methods for inference rely on least squares and models for the recovery curve, but recent work has come to use the entire spatio-temporal image data for estimation.

In this work, we have implemented a set of deep neural network architectures for estimating parameters such as the diffusion coefficient in FRAP. This is to our knowledge a novel approach with some potential advantages over conventional methods. We have implemented a set of both spatio-temporal and purely temporal neural network models, where operating on the full image data gives the best performance in terms of error on simulated data. The downsampler neural network model is easy to implement and parallels the extraction of the recovery curve, and can be trained from numerically simulated data. We show that the downsampling neural network can be trained on limited computational resources, using the combined power of continuously generating training data and batch-mode optimization. The neural networks demonstrate a robustness against noise and computational speed unlike the conventional least squares methods.

The performance of the neural networks versus the conventional methods is tested on simulated FRAP data and finally validated on experimental data, yielding good agreement with the expected values of the parameters and those obtained from the conventional methods.

Keywords: confocal microscopy, diffusion, fluorescence recovery after photobleaching, deep learning, machine learning, neural networks,.

Acknowledgements

I would like to thank and acknowledge the work and dedication Magnus Röding at RISE Agrifood & Bioscience has put into supervising my work. I enjoyed every bit of his candour, criticisms and encouragement, providing educational and challenging discussions not only on the thesis topic. I can only wish him all good fortune in his future endeavours. I would furthermore want to thank the rest of my colleagues at RISE Agrifood & Bioscience for providing an open, light-hearted and stimulating working environment, and especially Annika Krona for her invalueable assistance and instructions with the FRAP experiments. I can only hope to find such a great workplace again.

Finally, my greatest supporter has been, and always will be, Sara Almehed.

Victor Wåhlstrand Skärström, Gothenburg, January 2020

Contents

List of Figures xi

List of Tables xv

1 Introduction 1
- 1.1 Purpose . . . 1
- 1.2 Scope and Outline . . . 2

2 Background 3
- 2.1 Diffusion . . . 3
- 2.2 Fluorescence Recovery after Photobleaching . . . 7
  - 2.2.1 Experimental Procedure . . . 7
  - 2.2.2 FRAP Models . . . 8
    - 2.2.2.1 The Recovery Curve . . . 8
    - 2.2.2.2 Models for the Full FRAP Image . . . 9
- 2.3 Machine Learning for Regression . . . 11
  - 2.3.1 Supervised Learning and Regression . . . 11
  - 2.3.2 Artificial Neural Networks . . . 13
    - 2.3.2.1 Learning by Backpropagation . . . 14
    - 2.3.2.2 Modern Frameworks for ANNs . . . 15
  - 2.3.3 Convolutional Neural Network . . . 16
  - 2.3.4 Recurrent Neural Networks and LSTM . . . 18
  - 2.3.5 Convolutional LSTM Neural Network . . . 21

3 Methods 23
- 3.1 Simulating the FRAP Experiment . . . 23
- 3.2 Deep Neural Network Architectures . . . 25
  - 3.2.1 Previous Work . . . 25
  - 3.2.2 Neural Network for the Recovery Curve . . . 26
  - 3.2.3 2D Downsampling CNN . . . 26
  - 3.2.4 Convolutional LSTM . . . 29
  - 3.2.5 Early Trials . . . 29
- 3.3 Training Environment . . . 30
- 3.4 Hyperparameter Optimization . . . 31

4 Results 33
- 4.1 Model Training, Validation and Test Results . . . 33
- 4.2 Comparison between Neural Networks and Least Squares . . . 36
  - 4.2.1 Results for the Diffusion Coefficient . . . 37
  - 4.2.2 Results for the Bleach Parameter . . . 37
  - 4.2.3 Bias in the Parameter Estimation . . . 45
  - 4.2.4 Execution Time . . . 45
  - 4.2.5 Experimental Validation . . . 46

5 Conclusion **51**

References **53**

List of Figures

2.1 Image data from a FRAP experiment, pre-bleach, a small time after bleaching and post-bleach interval times. The fluorophores in the sample first diffuse, and are then bleached by a high intensity laser in the dashed circle. In the ensuing moments, there is a diffusing flow of bleached and unbleached particles in the circular ROI. 7

2.2 A typical recovery curve for a FRAP experiment. Imaging starts during a pre-bleach period, after which the fluorophores in the ROI are photobleached for a certain duration. Post-bleaching, the particles diffuse in and out of the ROI, leading to a recovery of the fluorescence intensity F. 8

2.3 A multilayer perceptron with input variables $\boldsymbol{x} = [x_1, x_2, x_3]^T$, scalar output $\boldsymbol{y} = y$ and hidden states h^ℓ, where the superscript ℓ indicates the layer. The weights w_{ij} from node i to node j in the affine transformation are illustrated with the edges of the graph. 13

2.4 Two equivalent representations of a perceptron neural network. The left is a graph with variables $\boldsymbol{x}, \boldsymbol{y}$ and operations $\boldsymbol{w}^T\boldsymbol{x}$ as nodes, with edges indicating forward passes and outside labels indicating trainable parameters. The representation to the right uses variables $\boldsymbol{x} = \{x_1, x_2, x_3\}, \boldsymbol{y} = y = \boldsymbol{w}^{(2),T}\boldsymbol{h}$ and intermediate hidden states $h_i = w_{ij}^{(1)} x_j$ as nodes, with edges indicating multiplication by the respective weight. 15

2.5 Illustration of the convolution operation on a 10×10 image with one channel, using two filters with 3×3 kernels and no stride. The discrete convolution is equivalent to an element-wise matrix product, and the result is an image with two channels. The size of the image also depends on hyperparameters such as stride and dilation. 17

2.6 Some authors prefer to define the convolutional layer as the sequence of a convolution, normalization and a non-linear activation function layer, but in this thesis, it is called a standard convolutional block. 17

2.7 A recurrent unit, taking the preceding hidden state $\boldsymbol{h}_{t-1}$ and features $\boldsymbol{x}_t$ of a sequence as input. 19

2.8 A recurrent neural network as a recurrence in perceptron-style (left) and unfolded in time $t = 1, ..., \tau$ with recurrent units from Figure 2.7 (right). . . 19

2.9 A long short-term memory unit (LSTM) is an improved recurrent unit with a hidden state $\boldsymbol{h}_t$ and cell state $\boldsymbol{c}_t$. These are updated in time by means of the input (i), forget (i) and output (o) gates. The learnable weights and biases from eq. (2.64)-(2.69) have been excluded to avoid cluttering. 20

3.1 Comparison between the first post-bleach frame in experimental FRAP (right) and numerically simulated FRAP (right), given parameters estimated using pixel-wise least squares from the experimental sequence. 24

3.2 Detailed architecture of the Convolutional LSTM and its downsampling block 27

3.3 Detailed architecture of the 2D Downsampling CNN (left) and the neural network for the recovery curve (right). 28

3.4 Visualization of the 2D downsampling CNN. The neural network takes a sequence of 110 frames, downsampling the frame size in steps using 2D convolutional blocks followed by maxpooling. The data is reduced to a sequence of 110 values which is fed to a series of fully connected layers. . . 29

4.1 The validation and training loss for the downsampling network (first from the top), ConvLSTM (second) and neural network for the recovery curve (bottom) after 2000 epochs. The downsampler loss is noisy, but mostly due to its small magnitude. The training is also notably underfitted. The ConvLSTM on the other hand converges notably slower, with noisy validation. The neural network for the recovery curve converges quickly and with little noise. 35

4.2 The mean squared error of D (in $\log_{10} \mathrm{px}^2/\mathrm{s}$) versus the noise level a, for $\alpha = 0.5$. The MSE increases monotonically with noise and diffusion coefficient for the least squares methods. The neural networks have slightly higher MSE in general, but seem more robust to noise and value of D. 39

4.3 The standard deviation of the estimations of D (in $\log_{10} \mathrm{px}^2/\mathrm{s}$) the noise level a, for $\alpha = 0.5$. The results indicate that the neural network makes more consistent estimations of D compared to the recovery curve method, and that the standard deviation increases with higher noise levels. 40

4.4 The mean squared error of D (in $\log_{10} \mathrm{px}^2/\mathrm{s}$) versus the noise level a, for $\alpha = 0.9$. The results indicate as expected that the MSE increases monotonically with noise and diffusion coefficient for the least squares methods. The neural networks have slightly higher MSE in general, but seem more robust to noise and value of D. 41

4.5 The standard deviation of the estimations of D (in $\log_{10} \mathrm{px}^2/\mathrm{s}$) versus the noise level a, for $\alpha = 0.9$. The results indicate that the downsampler neural network makes more consistent estimations of D compared to the recovery curve method, and that the standard deviation increases with higher noise levels. 42

4.6 The MSE of a few values of α as a function of noise a, for a diffusion rate of $10^{-12}\,\mathrm{m}^2/\mathrm{s}$. The least square-based estimates obtain a lower error than the neural networks in general, but the neural networks shows robustness to noise, while the least square methods get worse estimates at higher noise levels. 43

4.7 The MSE of a few values of α as a function of noise a, for a diffusion rate of $10^{-9}\,\mathrm{m}^2/\mathrm{s}$. In high diffusion rates, the recovery curve-based least squares gets a higher error than both pixel-based one and the downsampler. The neural network MSE values do not increase with the noise, indicating a robustness against the noise level. 44

4.8 The downsampler neural network estimates versus the pixel-based least squares estimates of D (left) and α (right), where $D = 1.2386\ \log_{10} \mathrm{px}^2/\mathrm{s}, \alpha = 0.7, a = 0.05$. The downsampler consistently overestimates D compared to LS, and less so underestimates α. 45

4.9 The mean execution time in milliseconds for 500 estimations for the least squares-based estimations (LS-PX, LS-RC) and neural networks (NN-PX, NN-RC). Since the method of least squares is an iterative method, it executes up to 2 orders of magnitudes slower than the neural networks. 46

4.10 Mean and standard deviations of the diffusion rate estimates of fluorescein on experimental data with 32 wt% and 56 wt% sucrose, indicating good agreement across methods. 48
4.11 First post-bleach frame ($T = 10$) from a sample experiment (first from the left), and the residuals from predictions with pixel-based least squares (second), recovery curve-based least squares (third), and the downsampler neural network (fourth). Since the downsampler makes slightly worse estimations of the bleach depth, this is also visible in the centre residuals. . . . 49
4.12 Recovery curve fit for the least squares methods and the downsampler neural network, indicating overall good fits, but an especially low estimate for the bleach parameter α from the downsampler. 49

List of Tables

4.1 Distribution of target parameter values in the training and validation data, chosen to cover common values encountered in FRAP experiments. 33
4.2 Values for the hyperparameters used in generating the data, training and optimizing the neural networks. 34
4.3 The test MSE on 2048 samples of the neural networks, given as parameter-wise MSE and total MSE. The downsampling 2D CNN outperforms the ConvLSTM and recovery curve network, but the ConvLSTM is notably just as good at estimating the bleach parameter α. 34
4.4 The test MSE on 2048 samples of the NN-PX and NN-LS methods, given as parameter-wise MSE and total MSE. 36
4.5 Mean and standard deviation for 32 wt% experimental samples, for the least square fits to recovery curve (*LS-RC*) and pixel data (*LS-PX*), as well as recovery curve network (*NN-RC*) and downsampler CNN (*NN-PX*). 47
4.6 Mean and standard deviation for the 56 wt% experimental samples, for the least square fits to recovery curve (*LS-RC*) and pixel data (*LS-PX*), as well as recovery curve network (*NN-RC*) and downsampler CNN (*NN-PX*). . . . 47

1 Introduction

In many areas of industry and science, understanding the microscopic properties of products and materials is highly important. Mass transport processes such as diffusion govern particles' movement and dissolution in solvents, and studying these processes is useful in wide-ranging subjects such as medicine, food science and soft material studies. Fluorescence recovery after photobleaching (FRAP) is an experimental method used in microscopy to estimate the mobility properties of particles in a solvent. One such property is the *diffusion coefficient*, which characterizes the rate at which particles disperse in the solution. In FRAP, fluorescent particles are bleached in a region with a laser. The time evolution of the fluorescence concentration in the bleach region and the moving particles in it is then used to estimate the diffusion coefficient.

Another common use for FRAP is diffusion and binding, where a dissolved species may bind and create a fluorescent complex molecule. The binding and unbinding rates, as well as diffusion coefficient are then reflected by the recovery of fluorescence, which makes an important tool for the study of proteins and enzymes in cells. FRAP has been applied as a non-destructive method to study particles in pharmacology, life science and food science, to name a few [1]. For example, the method is used in medicine to study controlled drug delivery in cells where mucus and biofilms may form barriers to this delivery [2, 3]. FRAP has also shown promise in food science, Svanberg et al. [4] studied the fat bloom of chocolate, the gray surface which may arise when stored for example in a refrigerator. The group used FRAP to characterize the diffusion of oils from the filling of the chocolates to the outer shell at a micro-level, which is known to cause the fat bloom.

1.1 Purpose

It is in the interest of FRAP users to make quick and accurate estimations of the diffusion coefficient and related parameters. The acquired data from FRAP is a sequence of images, i.e. spatio-temporal. This data is often reduced to a purely temporal sequence of the recovery of the mean intensity over time in the bleach region, known as the *recovery curve* [1]. Conventional methods fit a model to either the recovery curve or the full spatio-temporal sequence, for example using least squares. Fitting to the recovery curve is quick and easy to implement, but effectively discards a lot of information in the microscopy image. Other methods try to model the image sequence from the statistical distribution of fluorescence before bleaching, to the dynamics during and after bleaching, and finally image acquisition. A *pixel-based analysis*, where each image pixel is fit to a model used for inference, is more accurate, but also computationally more demanding than the recovery curve [5, 6, 7].

The computational drawbacks of pixel-based analysis with least squares bears the question if other methods could make use of the full spatio-temporal data more efficiently. The aim of this thesis is to use modern machine learning methods and specifically neural networks to estimate the diffusion coefficient, and compare this approach to conventional methods.

1.2 Scope and Outline

In this thesis we restrict ourselves to simple motion from diffusion, and will not consider diffusion with binding. However, any neural network method is easily extendible to more parameters. Likewise, only the neural networks and least squares methods of machine learning will be considered.

In Chapter 2, the background required to understand this thesis is presented. It covers diffusion, the principles behind FRAP and finally the theory on the methods of estimation and machine learning. Chapter 3 describes the tool used to numerically simulate FRAP experiments as training and test data for the results, as well as the implementation details of the neural netowrk architectures used for estimating the diffusion coefficient and related parameters. Lastly, this chapter covers the training environment used to train the neural networks in terms of hardware and optimization. Finally the results and discussion thereof is given in Chapter 4. The results are divided into three parts: The training and validation results of the chosen neural network architectures, a comparison between the best neural networks and least squares, and lastly a validation on experimental FRAP data. Finally, the conclusion of this work is presented in Chapter 5, together with discussions of the result and recommendations for future work.

2 Background

In this chapter, some background on diffusion will be presented, as well as a description of the fluorescence recovery after photobleaching method used to estimate the diffusion coefficient. Finally, the foundations of supervised learning, regression and deep learning will be covered.

2.1 Diffusion

Diffusion is a term for a set of similar transport mechanisms of particles and concentrations driven by thermal energy. We will consider two perspectives on diffusion: In the macroscopic view, diffusion causes a net equilibration of the particle concentration in fluids, while in the microscopic view, single particles are subject to random motion. Diffusion has attracted interest in a variety of fields, and its first documentation can be traced back to Scottish botanist Robert Brown [8], who observed seemingly random motion when examining plant pollen in an optical microscope. This inspired what would be known as *random walks* and *Brownian motion*.

The first mathematical descriptions of diffusion are often attributed to German physician Adolf Eugen Fick, who studied the flux of particle concentrations in liquids and gases due to a chemical potential gradient, e.g. a concentration gradient. This constitutes a macroscopic view of diffusion, which was originally viewed separately from the particle movements seen by Brown.

This flux diffusion was described by Fick in terms of the heat equation [9]. He postulated *Fick's laws of diffusion*; Firstly, that the flux $\boldsymbol{J}$ is proportional to the negative concentration gradient $-\nabla c(\boldsymbol{x}, t)$,

$$\boldsymbol{J}(\boldsymbol{x}, t) = -D\,\nabla c(\boldsymbol{x}, t), \tag{2.1}$$

where $\boldsymbol{x}, t$ are coordinates in space and time, and D is the *diffusion coefficient*. In Fick's view, the driving force behind diffusion was the concentration gradient. However, a more general notion is that the chemical potential ϕ drives diffusion on a macroscopic scale. Define the drift velocity $\boldsymbol{v}$ of the particles under a force $\boldsymbol{F}$ from the chemical potential as

$$\boldsymbol{v}(\boldsymbol{x}, t) = \mu \boldsymbol{F}(\boldsymbol{x}, t) = -\mu \nabla \phi(\boldsymbol{x}, t) \tag{2.2}$$

where μ is the constant of proportionality known as *mobility* [10, 11]. Since flux is the number of particles flowing through a surface per unit of time, we may write

$$\boldsymbol{J}(\boldsymbol{x}, t) = c(\boldsymbol{x}, t)\boldsymbol{v}(\boldsymbol{x}, t) = -\mu c(\boldsymbol{x}, t) \nabla \phi(\boldsymbol{x}, t), \tag{2.3}$$

This means for example that

$$D = \mu c(\boldsymbol{x}, t) \frac{\partial \phi}{\partial c} \tag{2.4}$$

and that if $\partial \phi / \partial c < 0, D > 0$ and hence the flux will be in the same direction as the concentration gradient. However, in some mixtures, the reverse may be true, which indicates

that the chemical potential is a more fundamental driving force.

In *Fick's Second Law*, he posited that the concentration changes in time according to the heat equation, such that

$$\frac{\partial c(\boldsymbol{x},t)}{\partial t} = -\nabla \cdot \boldsymbol{J}(\boldsymbol{x},t) = D\Delta c(\boldsymbol{x},t). \tag{2.5}$$

The result is an example of the *diffusion equation*, or in broader terms a *continuity equation*, conserving the flux of mass.

In chemical terms, diffusion may be seen as the random motion of individual particles driven by internal kinetic energy. At the particle level, the random walk described by Brown was shown by Albert Einstein to be completely analogous to the flux diffusion described by Fick half a century prior, but derived from a statistical rather than chemical perspective. [12]

We illustrate this derivation with a 1D random walk. Consider the probability $p(x,t) = \Pr(X_t = x)$ of finding a particle at position x at a time t as a Markov chain. Let δx denote a spatial increment in positive or negative direction, and δt be the time step. Then $p(x,t)$ is the weighted sum of the probabilites to go in either direction in the previous timestep:

$$p(x,t) = w_- p(x-\delta x, t-\delta t) + w_+ p(x+\delta x, t-\delta t) \tag{2.6}$$

where $w_+ + w_- = 1$ are transition probabilities defined as

$$w_+ = \Pr(X_{t+\delta t} = x+\delta x) \tag{2.7}$$
$$w_- = \Pr(X_{t+\delta t} = x-\delta x) \tag{2.8}$$

Further, assume that increments $X_{t+\delta t} - X_t$ from a position are independent of previous positions on the line $X_s, s < t$. In undirected diffusion a displacement is equally probable in either direction, hence $w_+ = w_-$. In directed, or biased, diffusion, there is a discrepancy $\varepsilon = w_+ - w_- \neq 0$. In this case, one direction will be favoured, and the system is said to have *drift* or *advection*.

Now expand $p(x-\delta x, t-\delta t)$ and $p(x+\delta x, t-\delta t)$ to get

$$p(x-\delta x,t-\delta t) = p(x,t) - \delta x\frac{\partial p}{\partial x} - \delta t\frac{\partial p}{\partial t} + \\ + \frac{1}{2}\delta x^2\frac{\partial^2 p}{\partial x^2} + \frac{1}{2}\delta t^2\frac{\partial^2 p}{\partial t^2} - \delta x\delta t\frac{\partial^2 p}{\partial x\partial t} + ... \tag{2.9}$$

$$p(x+\delta x,t-\delta t) = p(x,t) + \delta x\frac{\partial p}{\partial x} + \delta t\frac{\partial p}{\partial t} + \\ + \frac{1}{2}\delta x^2\frac{\partial^2 p}{\partial x^2} + \frac{1}{2}\delta t^2\frac{\partial^2 p}{\partial t^2} + \delta x\delta t\frac{\partial^2 p}{\partial x\partial t} + ... \tag{2.10}$$

Inserting eqs. (2.9)-(2.10) into eq. (2.6) yields

$$\delta t\frac{\partial p}{\partial t} = -\varepsilon\delta x\frac{\partial p}{\partial t} + \frac{1}{2}\delta x^2\frac{\partial^2 p}{\partial x^2} + \frac{1}{2}\delta t^2\frac{\partial^2 p}{\partial t^2} + \varepsilon\delta x\delta t\frac{\partial^2 p}{\partial x\partial t} + ... \tag{2.11}$$

since $w_+ + w_- = 1$ and $\varepsilon = w_+ - w_-$. This can be rewritten (by dividing with differential elements δt) into

$$\frac{\partial p}{\partial t} = -\varepsilon p\frac{\partial x}{\partial t} + \frac{1}{2}\frac{\delta x^2}{\delta t}\frac{\partial^2 p}{\partial x^2} + \frac{1}{2}\delta t\frac{\partial^2 p}{\partial t^2} + \varepsilon\delta x\frac{\partial^2 p}{\partial x\partial t} + ... \tag{2.12}$$

By choosing $\varepsilon, \delta x, \delta t$ such that they approach the limits

$$\frac{1}{2}\frac{\delta x^2}{\delta t} \to D \tag{2.13}$$

$$\varepsilon\frac{\partial x}{\partial t} \to v \tag{2.14}$$

we arrive at a one dimensional *diffusion-advection equation*:

$$\frac{\partial p}{\partial t} = D\frac{\partial^2 p}{\partial x^2} - v\frac{\partial p}{\partial x} \tag{2.15}$$

In the case of no advection, $w_+ = w_-$, we get $v = 0$ and the ordinary diffusion equation,

$$\frac{\partial p}{\partial t} = D\frac{\partial^2 p}{\partial x^2}. \tag{2.16}$$

analogous to eq. (2.5). This result unites the two views on diffusion, the macroscopical flux diffusion and the particle-wise random walk diffusion.

The diffusion coefficient offers more perspectives on the nature of diffusion. The diffusion eq. (2.5) relates the change of concentration over time to the divergence of the concentration gradient, and D is then interpreted as a rate of diffusion. D is in general dependent on the geometry and potentially the position, but for isotropic systems it is often assumed to be constant. For the initial condition $p(x, t = 0) = \delta(x)$ the solution to eq. (2.16) becomes

$$p(x,t) = \frac{1}{\sqrt{4\pi D t}} \exp\left(-\frac{x^2}{4Dt}\right). \tag{2.17}$$

The probability of finding a particle at a certain time and position is then related to the quotient $-x^2/4Dt$, where we for a fixed distance $x = x_0$ may define $\tau = x_0^2/4D$ as a characteristic time of diffusion.

In the case of spherical particles with radius r, Einstein derived an important relation of the diffusion coefficient, commonly known as Einstein-Stokes' relation [12]:

$$D = \frac{k_\mathrm{B} T}{6\pi\eta r} \tag{2.18}$$

Given a temperature T and Boltzmann's coefficient k_B, eq. (2.18) relates the rate of diffusion of a solute to the fluid viscosity η of a solvent, to be interpreted that more viscous solvents have lower rates of diffusion. This is a special case of

$$D = \mu k_\mathrm{B} T \propto \frac{V}{F} \tag{2.19}$$

where μ is the *mobility* of a particle in the solution [13]. The mobility relates to the average drift speed V of the particle under an external force F like in eq. (2.2), which reinforces the notion of the diffusion coefficient as a measure of how fast particles spread in a solvent.

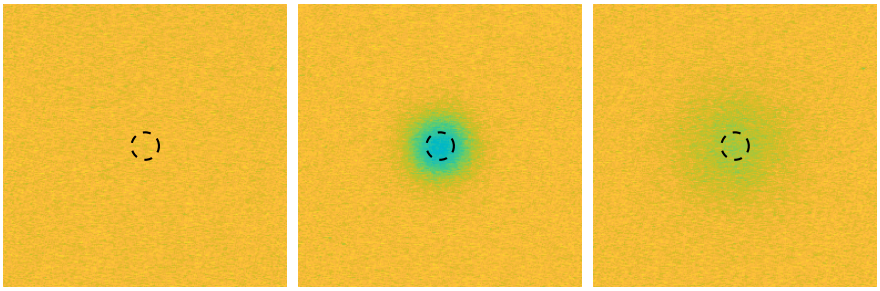
On a microscopic scale, if all particles could be tracked, it is possible to calculate the diffusion coefficient from the *mean square displacement* (MSD) from their origin. For a single particle it holds that the MSD, or second moment, is

$$\mathrm{MSD}(x) = \mathbb{E}_{p(\boldsymbol{x},t)}[X^2] = \int_{-\infty}^{\infty} \mathrm{d}x\, x^2 p(x,t) = 2Dt, \tag{2.20}$$

which may be fitted to estimate D. This is known as the *Law of Diffusion* or the Einstein-Smoluchowski relation, and indicates that the diffusion coefficient is essentially a spatial variance normalized by time [13].

2.2 Fluorescence Recovery after Photobleaching

Fluorescence recovery after photobleaching, or FRAP, is a method for determining the diffusion coefficient of particles dissolved in for example liquid suspensions, gels, cells or membranes. The FRAP experimental procedure involves a sample of fluorescently labelled particles suspended in the aforementioned solution or membrane. The experiment starts by bleaching the fluorescent particles in a *region of interest* (ROI), or bleach region, by means of a high-intensity laser. This is called photobleaching, and causes the concentration of fluorescent particles, and subsequently their combined intensity, to decrease in the ROI. In the following post-bleach time, unbleached fluorophores will diffuse into the region and bleached fluorophores will diffuse out. This causes a gradual fading of the region of interest and recovery of the mean intensity inside it [1]. See Figure 2.1 for an example of this recovery.



pre-bleach ***first post-bleach frame*** ***post-bleach***

Figure 2.1: *Image data from a FRAP experiment, pre-bleach, a small time after bleaching and post-bleach interval times. The fluorophores in the sample first diffuse, and are then bleached by a high intensity laser in the dashed circle. In the ensuing moments, there is a diffusing flow of bleached and unbleached particles in the circular ROI.*

2.2.1 Experimental Procedure

The FRAP method is most often implemented using a *confocal scanning laser microscope* (CLSM). Modern confocal laser scanning microscopy was invented in 1969 [14, 15] with various evolutions through the 1970s and 1980s. The light of a high intensity laser is collimated and split by a dichroic beam splitter. The mirrored light passes through the objective lens onto a small portion of the sample in the focal plane. This laser causes excitation of the fluorophore energy states, yielding emission of photons. The emission signal from the sample then passes through the dichroic mirror and into the conjugate pinhole. The conjugate, or confocal, pinhole is placed such that out-of-focus signals cannot pass, greatly reducing the noise and improving the resolution of the microscopy. Finally, the emitted light signal is frequently detected and amplified in a photomultiplier tube, before being projected onto the image plane. The acquired spatio-temporal signal is collected as integer counts, and is often normalized to a range $[0, 1]$. When modelling the photomultiplier it is assumed that its output is approximately linear, and thus proportional to its input intensity and thereby the particle concentration.

The noise reduction of the CLSM is vital for the fast processes observed in fluorescence recovery after photobleaching and CLSM enables imaging in different diffusion time domains. The light ray eponymously scans the sample at the desired speed, from side to

side, top to bottom. The scanning speed may thus be used to adapt the time scale of the experiment to that of the diffusion.

2.2.2 FRAP Models

The FRAP image is composed of a multitude of factors which influence the resulting image the viewer sees in the microscope. It is common to assume that the bleaching is perfectly cylindrical, which results in no net diffusion in the depth-plane and an effectively two-dimensional sample. However, more advanced models take the depth into account [1, 5, 16].

The difficulty and numerical complexity of modelling the resulting image means that the image series is in practice often reduced to a so-called *recovery curve*. However, models for the full pixel-wise data do exist, and attempt to take into account the diffusion, bleaching and the imaging processes.

2.2.2.1 The Recovery Curve

The recovery curve of fluorescence intensity, see Figure 2.2, captures the mean intensity F in the ROI. Given an initial mean intensity F_0, the sample is bleached with a bleach parameter α ending at $t = t_{\text{bleach}}$, whereupon the mean intensity in the ROI starts to increase. If the intensity of fluorescence does not recover to pre-bleach levels, this is indicative of an immobile fraction of fluorophores which does not diffuse.

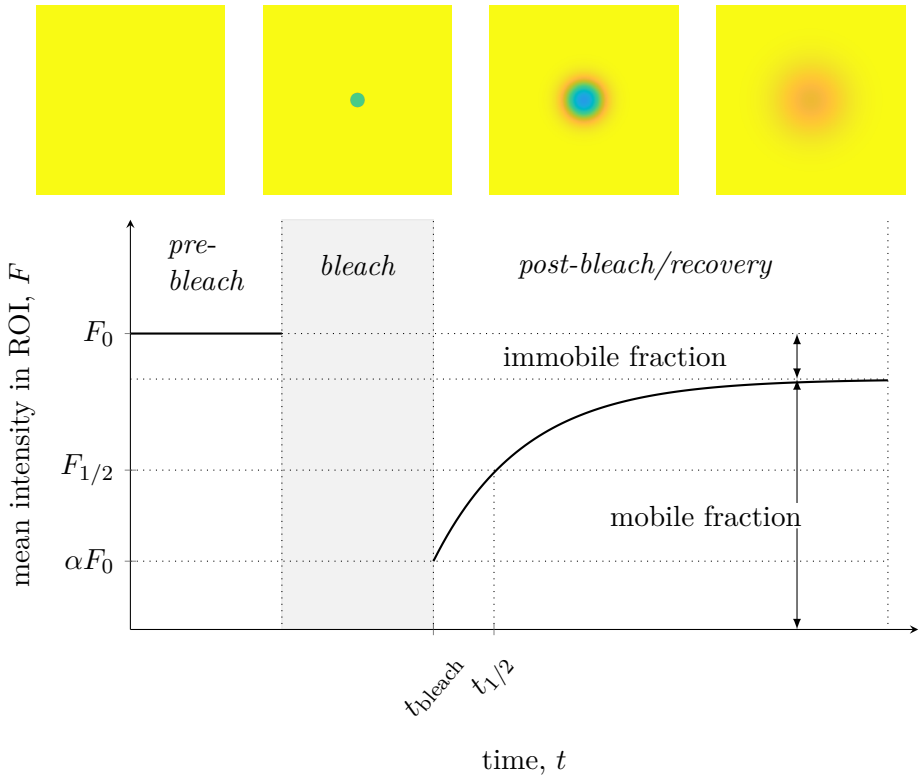


Figure 2.2: *A typical recovery curve for a FRAP experiment. Imaging starts during a pre-bleach period, after which the fluorophores in the ROI are photobleached for a certain duration. Post-bleaching, the particles diffuse in and out of the ROI, leading to a recovery of the fluorescence intensity F.*

The FRAP experiment is used to quantify the diffusive properties of the sample, such as the diffusion coefficient and mobile fraction of bleached particles. Analysing the recovery curve of the mean intensity is the dominating approach to estimate these system parameters. Kapitza et al. [17] modelled the recovery by means of an empirical exponential curve with similarities to eq. (2.17):

$$F(t) = F_0 \left(1 - \exp\left(-\frac{w^2}{4Dt}\right)\right) + b, \tag{2.21}$$

where F_0 is the initial concentration of fluorophores before bleaching, b is a constant amount of non-diffusing particles, w the radius of the ROI and D the diffusion coefficient. The parameters may then estimated by fitting the experimental fluorescence recovery curve to eq. (2.21) using least squares or maximum likelihood. A rough estimate of the diffusion coefficient can also be made by relating it to a characteristic half-time of recovery,

$$t_{1/2} = \tau \ln 2 = \frac{w^2 \ln 2}{4Dt} \tag{2.22}$$

calculated from where $F(t_{1/2}) = F_{1/2} = b/2$. Another, entirely empirical model is

$$F(t) = F_0 \left(1 - b \exp\left(-\frac{t}{\tau}\right)\right) \tag{2.23}$$

where F_0 is the initial concentration of fluorophores before bleaching, b is the fraction of bleached particles, and $\tau = t_{1/2}/\log(b/2)$ is a characteristic halft time of recovery.

Relying on empirical recovery curve models has received criticism for failing to include other parameters, such as binding rates. More advanced models exist that include for example the shape of the laser profile, but even these fail at taking into account excitation effects from the laser and are restrictive concerning the ROI shape. For an overview of models, see [1].

2.2.2.2 Models for the Full FRAP Image

Modern models for FRAP data try to include most aspects of the experiment: The shape of the initial concentration distribution, $c(x, y, t = 0)$, the temporal evolution of the concentration, the ROI bleaching, the image acquisition stage and finally noise models [1].

Assuming a homogeneous concentration of fluorophores $c(r, t)$ with a rotationally symmetric bleach region Ω, c is function of the radius r from the centre of the sample. The region of interest is then bleached with a laser with illumination intensity $I_{\text{bleach}}(r)$ for a time T, and the temporal evolution of $c(r)$ is then

$$\frac{\mathrm{d}c}{\mathrm{d}t} = -\alpha I_{\text{bleach}}(r) c(r, t), \tag{2.24}$$

where α is the photobleach parameter, the rate of bleaching of the fluorophores [1]. The concentration distribution in the sample after bleaching for a time T is then

$$c(r) = c_0 \exp\left(-\alpha I_{\text{bleach}}(r) T\right) \tag{2.25}$$

The laser bleach beam intensity $I_{\text{bleach}}(r)$ is often described as Fraunhofer-Airy or Gaussian. More accurate models also take into account the distribution of excitation states of the fluorophores due to the power of the laser. These are especially important when

modelling the sample with depth [1].

Jonasson et al. [5] also assumes a rotation symmetric bleach region and homogeneous initial concentration of fluorophores $c_0 = c(r, t = 0)$, with a time evolution according to the diffusion equation in polar coordinates,

$$\frac{\partial c}{\partial t} = D\left(\frac{1}{r}\frac{\partial c}{\partial r} + \frac{\partial^2 c}{\partial r^2}\right). \tag{2.26}$$

They further assume that the particles between the last bleach frame and first post-bleach frame have diffused enough such that the bleach profile of the sample after bleaching is approximately Gaussian. The initial concentration after bleaching is then modelled as

$$c(r, t = t_{\text{bleach}}) = a_0 - \frac{a_1}{r_0^2}\exp\left(-\frac{r^2}{r_0^2}\right), \tag{2.27}$$

where a_0 is the initial concentration of fluorophores and a_1, r_0 are constants. Once photobleached, the fluorescent particles movement in and out of the bleach region is governed by the diffusion equation, gradually fading the ROI. The solution to eq. (2.26) is then readibly obtained as

$$c(r, t) = a_0 - \frac{a_1}{4Dt + r_0^2}\exp\left(-\frac{r^2}{4Dt + r_0^2}\right). \tag{2.28}$$

This solution is analogous eq. (2.17), but in a two dimensional setting and for a wider initial distribution.

In the detection stage one adds to the model the effects of optical imaging on the resulting image. The fluorescence F as captured by the microscope may be modelled as the sum of all points on the sample, effectively the convolution of the fluorophore concentration with a detection *point-spread function* $I_{\text{detection}}$[1]:

$$F(\boldsymbol{r}, t) = I_{\text{detection}}(\boldsymbol{r}) \star C(\boldsymbol{r}, t) \tag{2.29}$$

$$= \int_{-\infty}^{\infty} \mathrm{d}\boldsymbol{r}' I(\boldsymbol{r})\, C(\boldsymbol{r} - \boldsymbol{r}') \tag{2.30}$$

where $(\star)$ denotes the convolution operation. Johansson et al. use a 3D point spread function

$$I_{\text{detection}}(x, y, z) = I_0 \exp\left(-2\frac{x^2 + y^2}{r_{xy}^2}\right)\exp\left(-2\frac{z^2}{r_z^2}\right), \tag{2.31}$$

where I_0, r_{xy}, r_z are constants. The parameters $\{a_0, a_1, D, r_0, I_0, r_{xy}, r_z\}$ of eq. (2.28) and eq. (2.29) may then be estimated numerically, for example by the method of least squares with maximum likelihood.

Finally, the image is subjected to noise. External noise is introduced due to thermal noise in the electronics and the photomultiplier tube amplification itself has a variance. An important assumption is also that the photomultiplier itself is in a linear domain, and that its output is proportional to the intensity [7, 5]. Moreover, light sources can create gradients in the sample illumination and increase thermal noise. The bleaching strength is given by the laser power output and number of bleach frames, and too strong bleaching can introduce non-linearities in the fluorescence as well as noise. Furthermore, noise models may take into account that the photon particle count in the image acquisition is Poisson distributed.

2.3 Machine Learning for Regression

Regression analysis is a set of techniques used to establish relationships between some continuous predictive variables $\boldsymbol{x}$ and a dependent outcome $\boldsymbol{y}$. There are many classical, parametric methods for regression, such as least squares estimation and in general maximum likelihood estimation. Machine learning is a type of statistical learning which may be used for regression where classical methods fail or are too computationally expensive. In this section we will cover regression and neural networks as a machine learning approach to the topic.

2.3.1 Supervised Learning and Regression

Supervised learning is a branch of statistical learning and machine learning that given an input $\boldsymbol{x} \in X$ and target $\boldsymbol{y} \in Y$ aims to learn a map $f(\boldsymbol{x}) : X \to Y$. The performance of the mapping is defined by the *loss function*, $\mathcal{L} : Y \times Y \to \mathbb{R}^+$. The goal is thus to find the most accurate representation of the function f in terms of the loss function.

This is often phrased as an optimization problem, with the objective of *minimizing the risk* R, defined as the expected loss over the joint probability distribution $p(\boldsymbol{x}, \boldsymbol{y})$ of $(\boldsymbol{x}, \boldsymbol{y})$

$$R(f) = \mathbb{E}_{p(\boldsymbol{x},\boldsymbol{y})}[\mathcal{L}(f(\boldsymbol{x}), \boldsymbol{y})], \tag{2.32}$$

where $f \in F$ is in the set of all possible representations F in some space. In conclusion, the mapping f^* is given by

$$f^* = \arg\min_f \mathbb{E}_{p(\boldsymbol{x},\boldsymbol{y})}[\mathcal{L}(f(\boldsymbol{x}), \boldsymbol{y})] \tag{2.33}$$

In practice, the true probability distribution $p(\boldsymbol{x}, \boldsymbol{y})$ is not known, and only a subset of the input and target data is available. In that case one has a dataset $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$, with $N \in \mathbb{Z}$ independent and identically distributed samples with an empirical distribution $\hat{p}(\boldsymbol{x}, \boldsymbol{y})$. The subsequent empirical risk must be minimized from the dataset, resulting in an estimated function representation $\hat{f}$:

$$\hat{f} = \arg\min_f \mathbb{E}_{\hat{p}(\boldsymbol{x},\boldsymbol{y})}[\mathcal{L}(f(\boldsymbol{x}), \boldsymbol{y})] = \arg\min_f \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(\boldsymbol{x}_i), \boldsymbol{y}_i). \tag{2.34}$$

The defining problem of supervised learning is thus to optimize the choixe of the representation function f. Supervised learning is typically split in two subdivisions, *regression* and *classification*. In regression, the target $\boldsymbol{y}$ is continuous and real-valued, whilst in classification the target is a set of discrete categories. The theoretical foundations remain the same, but in practice they are handled slightly differently. For FRAP, the targets are the real-valued system parameters of diffusion and this work will therefore only cover regression.

The most classical approach to regression is to model targets $\boldsymbol{y} \in \mathbb{R}^{M \times N}$ of N observations with M dimensions such that

$$\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{w}) + \varepsilon, \tag{2.35}$$

where $\boldsymbol{x} \in \mathbb{R}^{p \times N}$ are observations of some p features (e.g. pixels in an image or time series points), $\boldsymbol{w} \in \mathbb{R}^{M \times p}$ the parameters, and $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ is Gaussian noise. The objective

function R is then the mean squared error, MSE,

$$\mathrm{MSE}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \mathbb{E}\left[(\boldsymbol{y}_i - f(\boldsymbol{x}_i; \boldsymbol{w}))^2\right] = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{y}_i - f(\boldsymbol{x}_i; \boldsymbol{w}))^2 . \tag{2.36}$$

or the sum of squares $\mathrm{SS} = N \times \mathrm{MSE}$. The problem is now two-fold: Finding a suitable function representation f, and its corresponding parameters $\boldsymbol{w}$. A simple example is a linear model

$$\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{x} + \varepsilon, \tag{2.37}$$

where $\mathbb{E}(\boldsymbol{y}) = \boldsymbol{w}^T\boldsymbol{x}$ and the solution is known as

$$\begin{aligned}\boldsymbol{w}^* &= \arg\min_{\boldsymbol{w}} \mathrm{MSE}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) \\ &= \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N}\left(\boldsymbol{y}_i - \boldsymbol{w}^T\boldsymbol{x}_i\right)^2 \\ &= (\boldsymbol{x}^T\boldsymbol{x})^{-1}\boldsymbol{x}^T\boldsymbol{y}.\end{aligned} \tag{2.38}$$

For problems where f is non-linear, there are in general no closed form solutions. One alternative to least squares minimization is maximum likelihood estimation, where we assume a general probabilistic model, $X \sim p(\boldsymbol{x}; \theta)$, where $p(x; \theta)$ is a probability distribution of $\boldsymbol{x}$ with parameters θ. In this case, the likelihood L for N independent and identically distributed samples $\boldsymbol{x}_1, ..., \boldsymbol{x}_N$ is defined

$$L(\theta) = \prod_{i=1}^{N} p(\boldsymbol{x}_i|\theta). \tag{2.39}$$

Maximizing this likelihood is equivalent to minimizing its *negative log-likelihood*, such that

$$\arg\max_{\theta} L(\theta) = \arg\min_{\theta} -\log L(\theta) = \arg\min_{\theta} -\sum_{i=1}^{N}\log p(\boldsymbol{x}_i|\theta). \tag{2.40}$$

Observe that for a Gaussian model $X \sim \mathcal{N}(\mu, \sigma^2)$, this is the same as a linear least squares minimization up to a constant, since

$$\begin{aligned}-\log L(\theta) &= \prod_{i=1}^{N}\log\left(\left(\frac{1}{2\pi\sigma^2}\right)^{N/2}\exp\left(-\frac{\sum_{i=1}^{N}(\boldsymbol{x}_i - \mu)^2}{2\sigma^2}\right)\right) \\ &= \frac{N}{2}\log 2\pi\sigma^2 + \frac{1}{2\sigma^2}\sum_{i=1}^{N}(\boldsymbol{x}_i - \mu)^2\end{aligned} \tag{2.41}$$

where eq. (2.37) has $\mu = \mathbb{E}(\boldsymbol{y}) = \boldsymbol{w}^T\boldsymbol{x}$.

A common modification to least squares is a penalized or regularized loss function. The penalization is implemented to impose constraints on the parameters, for example with a p-norm,

$$\mathcal{L}_{\mathrm{pen}}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) = \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{w}) + \lambda||\boldsymbol{w}||_p, \tag{2.42}$$

where λ is a constant. Using the 1-norm this is known as L^1-regularization, or weight decay in the context of neural networks, and imposes a restriction on the size of the parameters. Weight decay has been shown to reduce overfitting and improving generalization, meaning

the model makes more accurate predictions from previously unseen data.

2.3.2 Artificial Neural Networks

A relatively modern approach to supervised learning is to use *artificial neural networks* (ANNs). An artificial neural network is an algorithm for approximating a potentially non-linear function $f = f(\boldsymbol{x}; \boldsymbol{w})$, mapping an output $\boldsymbol{x}$ to $\boldsymbol{y}$ by means of optimizing a set of weights $\boldsymbol{w}$. The inception of neural networks happened in the 1940s and 1950s when McCulloch and Pitts developed a logical model for the human brain, called neural nets [18]. Other milestones include a similar structure that was invented by Hebb who showed that the neural networks had the capacity of associative memory, and the *perceptron* created by Rosenblatt, with which he was able to solve simple logical problems [19]. The perceptron is a simple neural network and important building block, defined as an affine transformation with a non-linearity, where an input $\boldsymbol{x} \in \mathbb{R}^n$ is mapped to an output $\boldsymbol{y} \in \mathbb{R}^m$ via matrix multiplication with a weight matrix $\boldsymbol{w} \in \mathbb{R}^{m \times n}$

$$\boldsymbol{y} = \sigma(\boldsymbol{h}) = \sigma(\boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{b}) \tag{2.43}$$

where σ is a non-linear *activation function*, and $\boldsymbol{b}$ is a *bias* or *threshold*. The perceptron was invented by Rosenblatt as a model of human neural functions, where a chemical signal enters a neuron, which is then activated and sends a response if the signal is stronger than a threshold. In applications, the intention is that the perceptron should learn which input features are important from this activation. The state $\sigma(\boldsymbol{h})$ is thus often called the *activation*, and the components w_{ij} $i = 1, ..., m; j = 1, ..., n$ are called *neurons*.

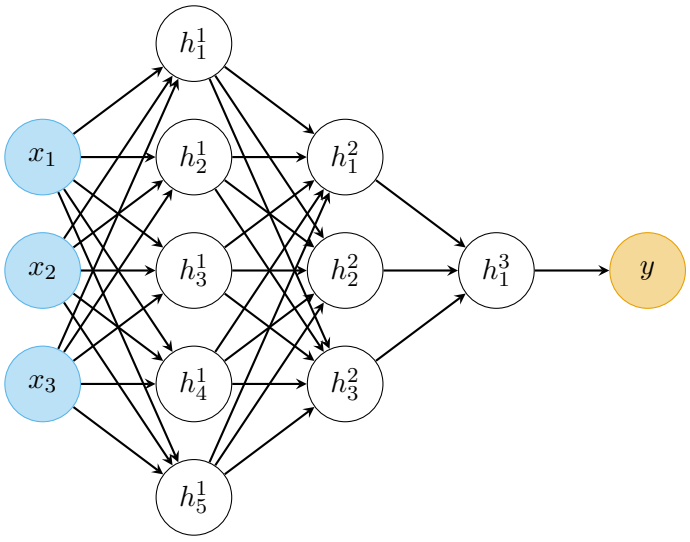


Figure 2.3: *A multilayer perceptron with input variables $\boldsymbol{x} = [x_1, x_2, x_3]^T$, scalar output $\boldsymbol{y} = y$ and hidden states h^ℓ, where the superscript ℓ indicates the layer. The weights w_{ij} from node i to node j in the affine transformation are illustrated with the edges of the graph.*

If σ were linear, this would be exactly equivalent to linear regression. Instead, common

choices are the sigmoid, hyperbolic tangent and rectified linear unit (ReLU) functions

$$\sigma(x) = \frac{1}{1+e^{-x}} \in [0,1] \tag{2.44}$$

$$\tanh x = \frac{e^{x}+e^{-x}}{e^{x}-e^{-x}} \in [-1,1] \tag{2.45}$$

$$\mathrm{ReLU}(x) = \max(0,x) \in [0,\infty) \tag{2.46}$$

The choice of activation is affected by its saturation for high inputs, and how it handles input below the threshold. In modern applications, ReLU is the most common choice [20].

The foundation of artificial neural networks is extending the perceptron with *hidden layers*. The multilayer perceptron (MLP) is a nested function where the output is fed sucessively as the hidden state $\boldsymbol{h}_{\ell-1}$ of the ℓth layer:

$$\boldsymbol{y}_\ell = \sigma(\boldsymbol{w}_\ell^T \sigma(\boldsymbol{w}_{\ell-1}^T \sigma(...) + \boldsymbol{b}_{\ell-1}) + \boldsymbol{b}_\ell) \tag{2.47}$$

or equivalently

$$\boldsymbol{h}_\ell = \boldsymbol{w}_\ell^T \sigma(\boldsymbol{h}_{\ell-1}) + \boldsymbol{b}_\ell \tag{2.48}$$

$$\boldsymbol{h}_1 = \boldsymbol{w}_1^T \boldsymbol{x} + \boldsymbol{b}_1. \tag{2.49}$$

This may be visualized as a graph with variables as nodes and the weights w_{ij}, $i = 1,...,m; j = 1,...,n$ as the connecting edges, see Figure 2.3. The number of layers expands the capacity for learning of a neural network, and allows it to solve more complex tasks. A neural network with more than two layers is known as *deep* [20, 21, 22]. Layers of MLPs are frequently called *fully connected*, *dense* or *linear layers*.

2.3.2.1 Learning by Backpropagation

The supervised optimization problem posed in eq. (2.34) is now a matter of optimizing the weights in the ANN. The data $\boldsymbol{x}$ is *forward propagated* through the network f, yielding an estimate $\hat{\boldsymbol{y}} = f(\boldsymbol{x};\boldsymbol{w})$, with a resulting loss $\mathcal{L}(\hat{\boldsymbol{y}},\boldsymbol{y})$.

The most common optimization method for neural networks is *backpropagation*, which is a variation of the gradient descent algorithm. After propagating the input through the network, the weights of a given layer are updated simultaneously with a correction $\boldsymbol{w} \leftarrow \boldsymbol{w} + \delta\boldsymbol{w}$, from the last layer to the first one. The correction is defined as the negative gradient of the expected empirical risk

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} \mathbb{E}_{\hat{p}(\boldsymbol{x},\boldsymbol{y})} \left[\mathcal{L}(\hat{\boldsymbol{y}},\boldsymbol{y})\right], \tag{2.50}$$

where $0 < \eta < 1$ is a step size parameter known as the *learning rate*. This requires explicitly calculating the gradient of the expected loss with respect to the weights of each layer [21].

This learning scheme is often improved by introducing *momentum*, a memory of the previous iteration $t-1$:

$$\delta\boldsymbol{w}_t = -\eta \nabla_{\boldsymbol{w}_t} \mathbb{E}_{\hat{p}(\boldsymbol{x},\boldsymbol{y})} \left[\mathcal{L}(\hat{\boldsymbol{y}},\boldsymbol{y})\right] + m\delta\boldsymbol{w}_{t-1}, \tag{2.51}$$

where m is the momentum rate. A further improvement is the *Nesterov accelerated mo-*

mentum, which evaluates the gradient at the point after adding momentum [23]:

$$\delta \boldsymbol{w}_t = -\eta \nabla_{\boldsymbol{w}_t} \mathbb{E}_{\hat{p}(\boldsymbol{x},\boldsymbol{y})} \left[\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \right] \big|_{\boldsymbol{w}_t + m\delta \boldsymbol{w}_{t-1}} + m\delta \boldsymbol{w}_{t-1}, \tag{2.52}$$

Both of these methods accelerate convergence to a minimum, without sacrificing accuracy. However, m must be chosen in conjunction with η.

2.3.2.2 Modern Frameworks for ANNs

In modern practice, little resembles the original ideas from the 1940s. Recent frameworks for artificial neural network implementation include PyTorch [24] and Tensorflow [25] which have support for parallelization and computation on GPUs at a high-level in several programming languages. The formulation of neural networks used by especially these two frameworks differs from the purely mathematical description above.

The neural network variables, trainable parameters and operations are expressed as a computational graph, essentially a directed acyclic graph, see Figure 2.4, illustrating the simple linear neural network $\boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$. Here, the right representation is a graph connecting inputs $\boldsymbol{x} = \{x_1, x_2, x_3\}$ with hidden states $h_i = w_{ij}^{(1)} x_j, i = 1, ..., 5$ and later the output $y = \boldsymbol{w}^{(2),T} \boldsymbol{h}$. The variables and hidden states are nodes, connected by weights in the matrix multiplication operation. This representation grows unwieldy for multiple layers and many hidden states, and often obscures the layer operations.

The left form is more compact, and feeds a node variable $\boldsymbol{x}$ to a matrix multiplication operation (*dot*) with trainable parameters $\boldsymbol{w}$, which is in turn fed as an output variable $\boldsymbol{y}$.

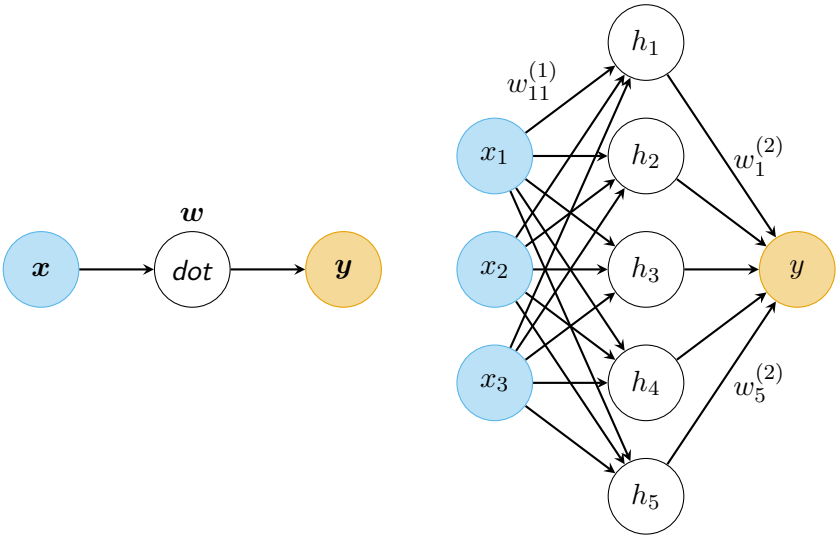


Figure 2.4: *Two equivalent representations of a perceptron neural network. The left is a graph with variables $\boldsymbol{x}, \boldsymbol{y}$ and operations $\boldsymbol{w}^T \boldsymbol{x}$ as nodes, with edges indicating forward passes and outside labels indicating trainable parameters. The representation to the right uses variables $\boldsymbol{x} = \{x_1, x_2, x_3\}, \boldsymbol{y} = y = \boldsymbol{w}^{(2),T} \boldsymbol{h}$ and intermediate hidden states $h_i = w_{ij}^{(1)} x_j$ as nodes, with edges indicating multiplication by the respective weight.*

The benefit of the compact graph representation is that it enables faster and more general calculation of derivatives than the approach covered in Section 2.3.2.1. Consider the MLP in Figure 2.3. The gradient of $\boldsymbol{y}$ with respect to the first layer weights $\boldsymbol{w}^{(1)}$ may

be expanded using the chain rule, as a product of the derivatives of the hidden states with respect to their input:

$$\frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{w}^{(1)}} = \frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{h}^{(3)}}\frac{\mathrm{d}\boldsymbol{h}^{(3)}}{\mathrm{d}\boldsymbol{w}^{(1)}} = ... = \frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}\boldsymbol{h}^{(3)}}\frac{\mathrm{d}\boldsymbol{h}^{(3)}}{\mathrm{d}\boldsymbol{h}^{(2)}}\frac{\mathrm{d}\boldsymbol{h}^{(2)}}{\mathrm{d}\boldsymbol{h}^{(1)}}\frac{\mathrm{d}\boldsymbol{h}^{(1)}}{\mathrm{d}\boldsymbol{w}^{(1)}} \tag{2.53}$$

If the derivative of the operation from one state to another is known, the calculation becomes easy [20].

The gradient in eq. (2.52) can most often not be estimated from the entire dataset $\mathcal{D}$ (or *batch*) due to either computational limitations or an insufficient dataset. A common alternative is to estimate the gradient on *mini-batches* $\mathcal{B}$, such that $\mathcal{D} = \cup_{i=1}^{N_\mathcal{B}} \mathcal{B}_i$, where $N_\mathcal{B}$ is a given number of mini-batches of size $N/N_\mathcal{B}$. One iteration over all mini-batches is then called an *epoch*.

For testing purposes, batch or mini-batch gradient descent is often split into a *training set*, *validation set* and *test set*, where only the training portion of the data is directly available during learning. The validation set is used indirectly to monitor the learning, and the test set is used after training for evaluation of the model's performance.

2.3.3 Convolutional Neural Network

A *convolutional* neural network (CNN), or a convolutional layer as part of a deep neural network, was developed to operate on data with an internal lattice structure, such as images. The use of CNNs has boomed in recent years, and was ignited by Yann LeCun using them for image recognition of US postal codes in 1998 [26].

Given an input image $\boldsymbol{x}$ with dimensions (C_in, W, H), where W, H are the width and height of the image respectively and C_in denotes the number of channels, the convolutional layer output $\boldsymbol{y}$ over this image is defined as

$$\boldsymbol{y}_c = \boldsymbol{b}_c + \sum_{c'=1}^{C_\text{in}} \boldsymbol{x}_{c'} \star \boldsymbol{w}_{cc'}, \qquad c = 1, ..., C_\text{out}. \tag{2.54}$$

Here $\boldsymbol{w}$ is a learnable weight lattice known as the *kernel*, with dimensions $(C_\text{in}, C_\text{out}, k, k)$ and a kernel size k. The kernel is equivalent to a filter in traditional image analysis, and can function as for example a blurring filter or contour-detecting filter. Here $(\star)$ denotes the discrete *convolution* operation, defined

$$y_{ijcc'} = (\boldsymbol{x}_{c'} \star \boldsymbol{w}_{cc'})_{ij} = \sum_{m=1}^{k}\sum_{n=1}^{k} x_{(i-1)s+m,\,(j-1)s+n,\,c'}\, w_{mncc'} \tag{2.55}$$

for a row pixel i, column pixel j and out channel c. The s variable denotes the *stride* hyperparameter, which allows for sampling only every s pixel, in practice downsampling the input image to a smaller size. [20]

The convolution operation in practice amounts to element-wise multiplication between input and weights, as illustrated in Figure 2.5, and is thus equivalent to a fully connected perceptron on a small subset of the image. This rectangular subset in the input image is known as the $k \times k$ receptive field of the output. However, the fundamental difference to a fully connected neural network is *weight sharing*, where weights are shared between input pixels, leading to a significant reduction in the number of operations and weights necessary compared to the former, from an order of $\mathcal{O}(NM)$ to $\mathcal{O}(kM)$, given N inputs and M outputs. [20]

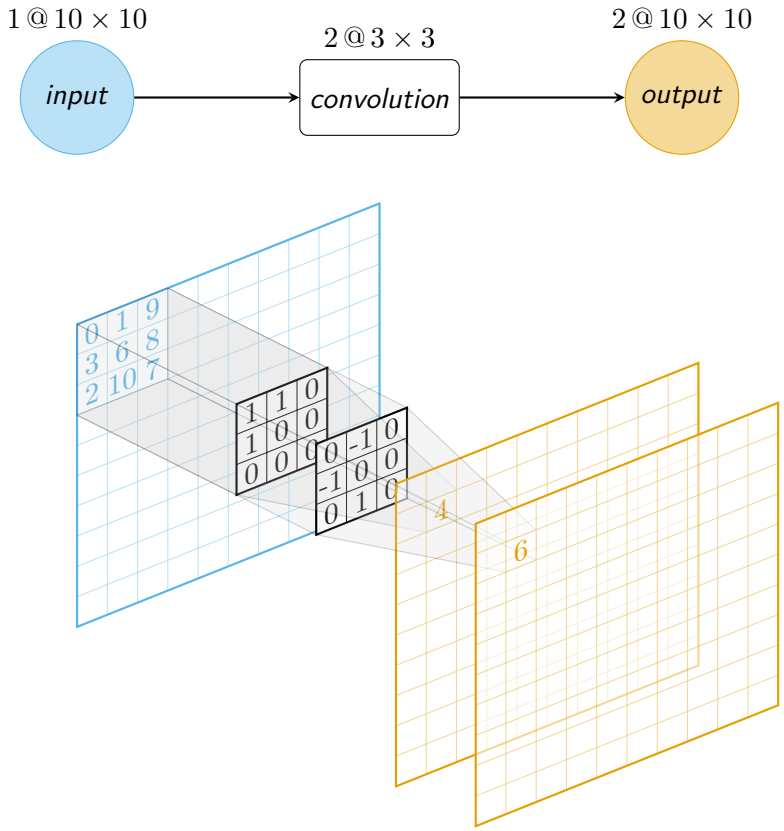


Figure 2.5: *Illustration of the convolution operation on a* 10×10 *image with one channel, using two filters with* 3×3 *kernels and no stride. The discrete convolution is equivalent to an element-wise matrix product, and the result is an image with two channels. The size of the image also depends on hyperparameters such as stride and dilation.*

The convolution operation of eq. (2.55) is also applicable to the 1D and 3D cases, which is especially relevant to this work, where a recovery curve has dimensions (C, T) and the full FRAP data has dimensions (C, T, H, W), where $C = 1$ and T is the temporal dimension.

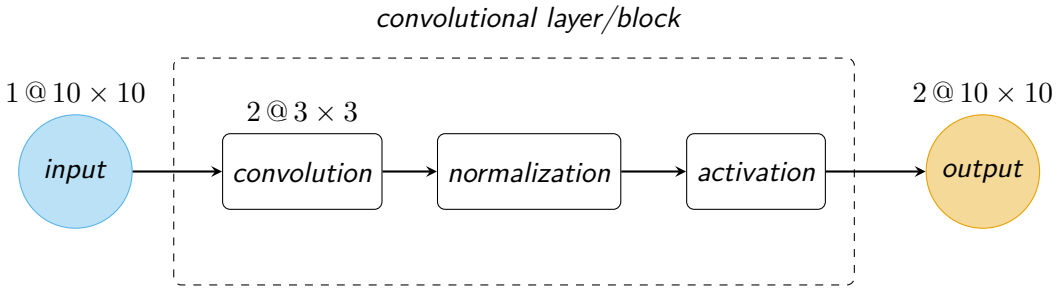


Figure 2.6: *Some authors prefer to define the convolutional layer as the sequence of a convolution, normalization and a non-linear activation function layer, but in this thesis, it is called a standard convolutional block.*

A variation of the convolution operation is the pooling operation, most notably *max pooling* and *average pooling*, the former of which is an efficient operation for non-linearity

and downsampling. If s is a stride like before, and k is the kernel size of the max filter, then

$$y_{ijc} = \text{MaxPool}(x)_{ijc} = \max_{m=1,\dots,kH} \max_{n=1,\dots,kW} x_{(i-1)s+m,\,(j-1)s+n,\,c} \tag{2.56}$$

where H, W are the height and width of the input image, respectively. Similarly, one may define an average pooling layer, such that

$$y_{ijc} = \text{AvgPool}(x)_{ijc} = \frac{1}{k^2 HW} \sum_{m=1}^{kH} \sum_{n=1}^{kW} x_{(i-1)s+m,\,(j-1)s+n,\,c} \tag{2.57}$$

Pooling layers provide a way for the network to emphasize certain regions in their input, and provides local invariance to activations. This local invariance means the network becomes less sensitive to small perturbations in the receptive fields [20].

In practice, a convolution is most often followed by a normalization and an activation, in a convolutional layer or block, see Figure 2.6. Normalization is a regularization technique used to improve generalization on the validation set of the network. A common choice is *batch normalization*, introduced in 2015, which is a feature-wise standardization to zero mean and unit variance over a mini-batch:

$$x_{ij} \leftarrow \frac{x_{ij} - \mathbb{E}\left[x_{ij}\right]}{\text{Var}\left[x_{ij}\right] + \varepsilon}, \tag{2.58}$$

where $\varepsilon > 0$ is a small tolerance to avoid division by zero and $\mathbb{E}\left[x_{ij}\right], \text{Var}\left[x_{ij}\right] \text{Var}\left[x_{ij}\right]$ are a running mean and variance over time [27]. The method has been shown to stabilize training, but the reason is not yet fully understood. During test and validation, the mean and variance is fixed to their training values.

The most common activation in modern CNNs is the ReLU function, and this is the activation used throughout this thesis unless stated otherwise.

Another regularization technique is the *dropout layer*[28], which is often applied to fully connected linear layers, defined as

$$\text{Dropout}(\boldsymbol{w}; p) = \begin{cases} 0, & \text{with prob. } p \\ \boldsymbol{w}, & \text{with prob. } 1-p \end{cases} \tag{2.59}$$

This method is meant to force the neural network to more efficiently encode information in its weights. As described by Goodfellow et al. [20], dropout may also be seen as an average of many smaller perceptrons.

2.3.4 Recurrent Neural Networks and LSTM

A common option when modelling sequences is using *recurrent neural networks*, which are characterized by some neurons connecting not only to other neurons, but also themselves. This recurrence, or feedback, allows the network to share weights during the length of a sequence.

Given a sequence $\boldsymbol{x}_t$ in time $t = 1, ..., \tau$, a recurrent layer has a time dependent hidden state $\boldsymbol{h}_t$ which is the sum of the last hidden state in the sequence and the current input, see Figure 2.7.

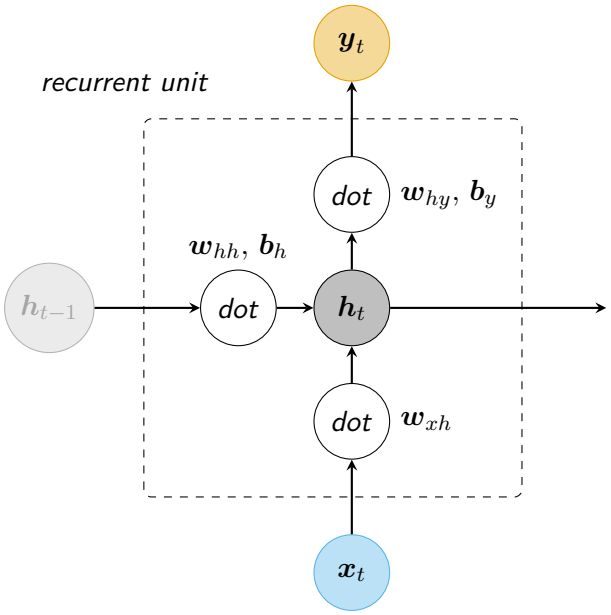


Figure 2.7: *A recurrent unit, taking the preceding hidden state $\boldsymbol{h}_{t-1}$ and features $\boldsymbol{x}_t$ of a sequence as input.*

The recurrent layer operations are

$$\boldsymbol{h}_t = \sigma(\boldsymbol{w}_{xh}^T \boldsymbol{x}_t + \boldsymbol{w}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h) \tag{2.60}$$

$$\boldsymbol{y}_t = \sigma(\boldsymbol{w}_{hy}^T \boldsymbol{h}_t + \boldsymbol{b}_y) \tag{2.61}$$

where $\boldsymbol{w}_{xh}$,, $\boldsymbol{h}_{hy}$, $\boldsymbol{h}_{hh}$ respectively connect input to hidden state, hidden state to output and from the last hidden state to the new one. The hidden state $\boldsymbol{h}_0$ as well as the weights and biases $\boldsymbol{b}_h, \boldsymbol{b}_y$ must be initialized.

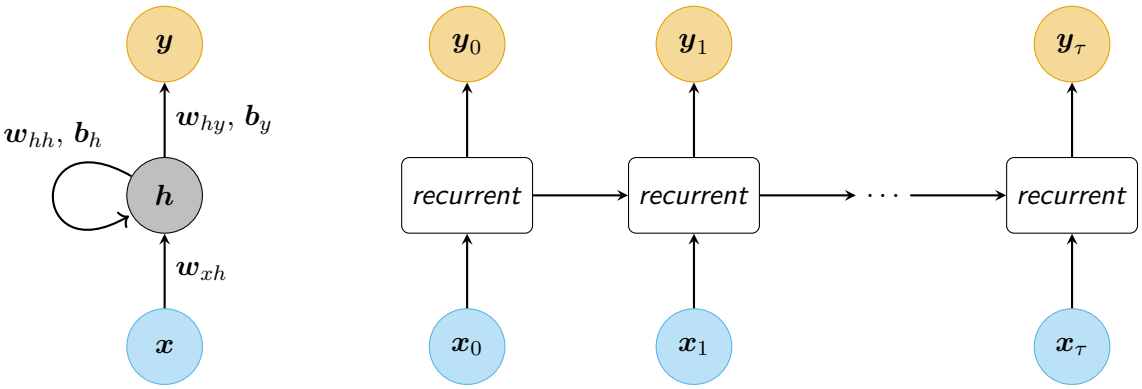


Figure 2.8: *A recurrent neural network as a recurrence in perceptron-style (left) and unfolded in time $t = 1, ..., \tau$ with recurrent units from Figure 2.7 (right).*

A recurrent neural network is often trained with *backpropagation through time*, which minimizes the empirical risk over the entire sequence:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \sum_{t=1}^{\tau} \nabla_{\boldsymbol{w}} \mathbb{E}_{\hat{p}(\boldsymbol{x})} \left[\mathcal{L}(f(\boldsymbol{x}_t; \boldsymbol{w}), \boldsymbol{y}_t) \right]. \tag{2.62}$$

In practice, this may be seen as unfolding the recurrence as connected layers in time, see Figure 2.8. This unfolding sheds light on the fact that neural networks can be described as dynamical systems, not just RNNs. However, it exposes a problem with backpropagation as a method of learning for RNNs. Approximating the sequence as continuous in a single variable, we have that

$$h(t) \approx e^{w(t)t} h_0 \tag{2.63}$$

which for $e^{w(t)} > 1$ will grow exponentially in time, and for $e^{w(t)} < 1$ will shrink exponentially, causing the network to stop learning catastrophically. This is called the unstable gradient problem [29, 22]. This is not a problem unique to RNNs, but for deep neural networks in general. However, since the depth of an RNN is related to the length of the input sequence, they are especially sensitive.

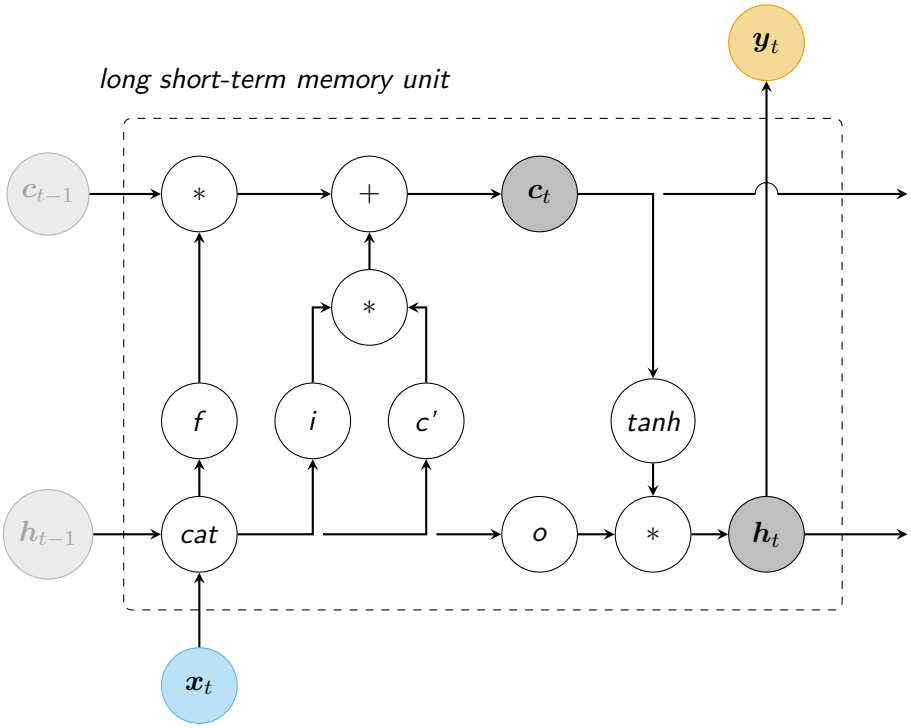


Figure 2.9: *A long short-term memory unit (LSTM) is an improved recurrent unit with a hidden state* $\boldsymbol{h}_t$ *and cell state* $\boldsymbol{c}_t$*. These are updated in time by means of the input (i), forget (i) and output (o) gates. The learnable weights and biases from eq.* (2.64)-(2.69) *have been excluded to avoid cluttering.*

Long short-term memory (LSTM) was introduced in 1997 as a remedy for the unstable gradient problem, and introduced a new hidden "cell state" as well as gating operations to only preserve important information from the input [30].

The LSTM is governed by the following set of operations

$$f(\boldsymbol{x}_t, \boldsymbol{h}_t) = \sigma(\boldsymbol{w}_{xf}^T \boldsymbol{x}_t + \boldsymbol{w}_{hf}^T \boldsymbol{h}_{t-1} + \boldsymbol{b}_f) \tag{2.64}$$

$$i(\boldsymbol{x}_t, \boldsymbol{h}_t) = \sigma(\boldsymbol{w}_{xi}^T \boldsymbol{x}_t + \boldsymbol{w}_{hi}^T \boldsymbol{h}_{t-1} + \boldsymbol{b}_i) \tag{2.65}$$

$$o(\boldsymbol{x}_t, \boldsymbol{h}_t) = \sigma(\boldsymbol{w}_{xo}^T \boldsymbol{x}_t + \boldsymbol{w}_{ho}^T \boldsymbol{h}_{t-1} + \boldsymbol{b}_o) \tag{2.66}$$

$$c'(\boldsymbol{x}_t, \boldsymbol{h}_t) = \tanh\Big(\boldsymbol{w}_{xc}^T \boldsymbol{x}_t + \boldsymbol{w}_{hc}^T \boldsymbol{h}_{t-1} + \boldsymbol{b}_c\Big) \tag{2.67}$$

$$\boldsymbol{c}_t = f * \boldsymbol{c}_{t-1} + i * c' \tag{2.68}$$

$$\boldsymbol{h}_t = o * \sigma(c_t) \tag{2.69}$$

where the cell state $\boldsymbol{c}_t$ is responsible for carrying long-term information, see Figure 2.9, and cat is a concatenation operation, e.g. $\mathit{cat}(\boldsymbol{x}, \boldsymbol{h}) = [\boldsymbol{x}, \boldsymbol{h}]$, and $(*)$ element-wise matrix multiplication. This unit requires that both $\boldsymbol{c}_0$ and $\boldsymbol{h}_0$ are appropriately initialized.

The information from the input sequence is passed through a series of gate operations. The forget gate f is responsible for determining what information to preserve from the previous cell state, and what to forget. The input gate i weights new information from a candidate cell state c', and the output gate o is a weight determining what fraction of the cell state should be forwarded as the next hidden state $\boldsymbol{h}_t$.

2.3.5 Convolutional LSTM Neural Network

A relatively recent development is the extension of RNNs to image data. Normal RNNs are unsuitable for images due to the large number of neurons required. The *convolutional LSTM* (ConvLSTM) was invented for predicting weather radar data from an image time series, but has since also been used for other spatiotemporal tasks, such as video action recognition [31, 32]. The ConvLSTM is a generalization of eq. (2.64)-(2.69) where the affine transforms are replaced by 2D convolutions like in eq. (2.55).

$$f(\boldsymbol{x}_t, \boldsymbol{h}_t) = \sigma(\boldsymbol{w}_{xf} \star \boldsymbol{x}_t + \boldsymbol{w}_{hf} \star \boldsymbol{h}_{t-1} + \boldsymbol{b}_f) \tag{2.70}$$

$$i(\boldsymbol{x}_t, \boldsymbol{h}_t) = \sigma(\boldsymbol{w}_{xi} \star \boldsymbol{x}_t + \boldsymbol{w}_{hi} \star \boldsymbol{h}_{t-1} + \boldsymbol{b}_i) \tag{2.71}$$

$$o(\boldsymbol{x}_t, \boldsymbol{h}_t) = \sigma(\boldsymbol{w}_{xo} \star \boldsymbol{x}_t + \boldsymbol{w}_{ho} \star \boldsymbol{h}_{t-1} + \boldsymbol{b}_o) \tag{2.72}$$

$$c'(\boldsymbol{x}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{w}_{xc} \star \boldsymbol{x}_t + \boldsymbol{w}_{hc} \star \boldsymbol{h}_{t-1} + \boldsymbol{b}_c) \tag{2.73}$$

$$\boldsymbol{c}_t = f * \boldsymbol{c}_{t-1} + i * c' \tag{2.74}$$

$$\boldsymbol{h}_t = o * \sigma(c_t) \tag{2.75}$$

These modifications allow the LSTM to take structured data like images in a sequence as input.

3 Methods

We are interested in learning how to estimate the diffusion coefficient and other parameters from FRAP experiments using neural networks. These experiments are simulated numerically with given parameter values, which are then estimated by least squares and by a set of deep neural networks. This chapter covers the simulation procedure and the methods of estimation.

3.1 Simulating the FRAP Experiment

The simulation tool for fluorescence recovery after photobleaching experiments was implemented in MATLAB by Röding et al.[7], and the code is available at `https://github.com/roding/frap_matlab`. The numerical simulation solves the 2D diffusion equation

$$\frac{\partial c(x,y,t)}{\partial t} = D\Delta c(x,y,t) \tag{3.1}$$

where c denotes the concentration of fluorophores, and D is the diffusion coefficient like before. Diffusion coefficients are temperature dependent, but in room temperature a common, natural upper bound of the diffusion coefficient is that of water in an ambient environment, $D = 2 \times 10^{-9}\,\mathrm{m}^2/\mathrm{s}$, in which most particles are suspended. Lower bounds vary with application, but large molecules like ribosomes may have as low as $D = 2 \times 10^{-14}\,\mathrm{m}^2/\mathrm{s}$ [33, 34].

The diffusion equation is solved with periodic boundary conditions in a box with dimensions $(H+2M)\times(W+2M)$, $H = W$, where H and W denote the height and width of the box respectively, and M is a padding used to reduce periodicity artefacts of the diffusion. The standard settings are $H = W = 256, M = 128$ pixels. The diffusion equation is then solved in the Fourier domain, where it becomes a set of independent ordinary differential equations

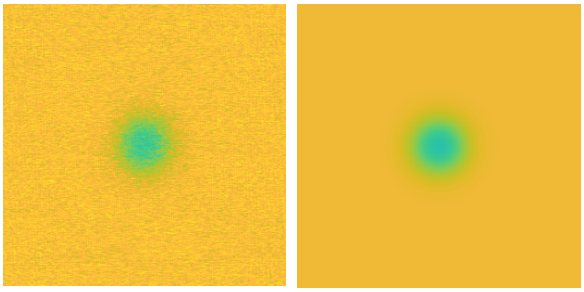
$$\frac{\partial \hat{c}(\xi,\eta,t)}{\partial t} = -(\xi^2+\eta^2)\,D\,\hat{c}(\xi,\eta,t) \tag{3.2}$$

where $\hat{c}(\xi,\eta,t)$ is the 2D Fourier transform of $c(x,y,t)$ in the spatial coordinates (x,y). The concentration is initialized in the spatial domain such that $c(x,y,t=0) = c_0$. Bleaching is done with a mask that bleaches the box in the ROI Ω with a factor α, such that

$$c(x,y) = \begin{cases} c_0\alpha, & (x,y) \in \Omega \\ c_0, & (x,y) \notin \Omega \end{cases} \tag{3.3}$$

where α is known as the bleaching depth parameter. In experiments a high degree of bleaching, $\alpha \approx 0$, is desirable for a strong signal with good recovery, but may in practice lead to non-linearities as mentioned in Section 2.2.2.2. A reasonable interval is thus $\alpha \in [0.5, 0.95]$. In a similar fashion, the initial concentration c_0 must be sufficiently high to get a good signal-to-noise ratio, but not so high that it saturates the signal acquisition. An appropriate choice is thus $0.5 < c_0 < 1$.

In order to account for the limited imaging and bleaching resolution of CLSMs, the mask



experimental ***simulated***

Figure 3.1: *Comparison between the first post-bleach frame in experimental FRAP (right) and numerically simulated FRAP (right), given parameters estimated using pixel-wise least squares from the experimental sequence.*

is supersampled and then downsampled again through an averaging filter, smoothing the edges of the ROI.

$$\hat{c}(\xi, \eta, t + \Delta t) = e^{-(\xi^2+\eta^2)D\Delta t}\hat{c}(\xi, \eta, t) \tag{3.4}$$

The solution is computed in the Fourier domain using MATLAB's Fast Fourier Transform `fft2` for a time $t_{\text{prebleach}}$ without bleaching, a time t_{bleach} of bleaching and finally $t_{\text{postbleach}}$ frames after bleaching. The sampling time between frames is set as Δt. The concentration $c(x, y, t)$ is then given by the inverse Fourier transform (`ifft2`) of eq. (3.4). Figure 3.1 shows a comparison between the first FRAP frame after bleaching of an experimental sample, compared with the respective frame of a numerically simulated FRAP sequence without noise with parameters estimated from the experiment.

Define $\theta = \{D, c_0, \alpha\}$ as the set of parameters we wish to estimate from a simulation or experiment. The parameters are then estimated using least squares via maximum likelihood, like in Section 2.3.1.

Two methods of least-squares fitting are used: Firstly, the *pixel-based method* assumes that every pixel value $p(x, y, t), x = 1, ..., W, y = 1, ..., H, t = 1, ..., T$ in the FRAP image is independent and identically normal distributed with mean $c(x, y, t)$ and variance of the form

$$\sigma^2(p(x, y, t)) = a + bp(x, y, t) \tag{3.5}$$

where a is a constant noise offset, and b is a factor of noise proportional to the intensity accounting for experimental noise from imaging and the photomultiplier. The negative likelihood is then

$$\ell(\theta) = -\frac{1}{2}\sum_{x,y,t}\log\left(2\pi\sigma^2(p(x, y, t)\right) - \frac{1}{2}\sum_{x,y,t}\frac{(p(x, y, t) - c(x, y, t|\theta))^2}{2\pi\sigma^2(p(x, y, t))}, \tag{3.6}$$

where x, y, t run from 1 to W, H, T respectively. Here $c(x, y, t|\theta)$ denotes the values generated by the model given the set of parameters, and $p(x, y, t)$ the acquired image pixels from a simulation or experiment.

Secondly, the *recovery curve method* calculates a curve F as the mean intensity in the ROI before and after bleaching ($t > t_{\text{bleach}}$)

$$F(t) = \sum_{x,y} w(x, y, t)p(x, y, t) \tag{3.7}$$

where w is an indicator matrix

$$w(x,y,t) = \begin{cases} \frac{1}{|\Omega|}, & (x,y) \in \Omega, \forall t \\ 0, & (x,y) \notin \Omega, \forall t \end{cases} \tag{3.8}$$

and $|\Omega|$ are the number of pixels in the ROI. The recovery curve noise variance is also i.i.d. normal with zero mean and variance

$$\sigma^2(F(t)) = \sum_{x,y} w(x,y,t)^2 \left(a + bp(x,y,t)\right). \tag{3.9}$$

as a consequence of the distribution of the pixels. Analogously to the concentration, F has a log-likelihood

$$\ell(\theta) = -\frac{1}{2}\sum_t \log\left(2\pi\sigma^2(F(t)\right) - \frac{1}{2}\sum_t \frac{\left(F(t) - \Phi(t|\theta)\right)^2}{2\pi\sigma^2(F(t))}, \tag{3.10}$$

where $\Phi(t|\theta)$ denotes the recovery curve generated by the model given the set of parameters, and $F(t)$ is the acquired signal from the simulation.

The negative log-likelihoods of both methods are then minimized by the `fmincon` routine in MATLAB, yielding estimates for θ.

3.2 Deep Neural Network Architectures

A broad set of neural network architectures were implemented during the course of this work, for both the temporal recovery curve data and the full spatio-temporal data. The computational limitations of the latter resulted in a pure 2D CNN approach, and a ConvLSTM. The code is available at `https://github.com/waahlstrand/frappe`.

Methods like least squares rely on statistical models of the recovery curve or individual pixels for fitting. In that sense, neural networks are *model-agnostic* and do not make assumptions about the mechanisms behind the data.

3.2.1 Previous Work

Deep learning applied to FRAP is unprecedented to our knowledge. The majority of research into deep learning applications for image data is concentrated on 2D CNNs and image recognition [**inception**, 35]. Lately however, there has been substantial work on ANNs for video data, especially in the area of video classification and action/gesture recognition. It seems natural to extend 2D CNNs to 3D, but as noted by several authors, the success of 2D CNNs has yet to be replicated for video data [36]. There have been multiple efforts to use 3D CNNs, such as C3D, Resnet-3D, and 3D Inception modules, but no method has gained traction similar to their 2D counterparts [36, 37, 38]. An uncompromisable problem with 3D CNNs is their computational complexity and number of neurons, and thus their tendency to overfit.

Many approaches thus use spatial convolutions and attempt various methods of extracting the temporal information, such as the frame-wise methods developed by Karpathy et al. [39], where the temporal information is fused either in groups of frames or separately. The results however, were known to be "disappointing" by other authors, which applied LSTMs as the temporal fusion layer instead [40]. A complement to the neural network

architecture is data augmentation and feature engineering, both of which have limited application to the FRAP data. Optical flow is used by many authors as a means of extracting pixel movement between frames, and often improves accuracy in gesture recognition by several percentage units [36, 40, 41]. Unfortunately, due to the diffusive and noisy nature of FRAP data, optical flow is an ill-suited feature and fails to extract any meaningful movement.

Using simulated data for training is also fairly uncommon, but has precedence in text recognition and 2D/3D image recognition [42, 43], but then primarily as augmentation for a natural dataset.

3.2.2 Neural Network for the Recovery Curve

The recovery curve has dimensions (T), essentially becoming a curve-fitting problem. Initial tests feeding the entire recovery curve to a deep fully connected neural network did not perform well, likely due to the discontinuity after bleaching, see Figure 2.2. We know that the information about the initial concentration c_0 is mostly contained in the pre-bleach frames, and that the ratio of the concentration in the ROI before and after bleaching is the definition of α. For this reason, the pre-bleach and post-bleach frames are fed separately to a respective fully connected neural network with 5 layers. The outputs of these networks are then fused and fed to a series of three final fully connected layers. The number of neurons were chosen by manual search. The resulting network is given by Figure 3.3.

3.2.3 2D Downsampling CNN

The recovery curve is an example of early feature engineering, which condenses the spatio-temporal pixel data into a single feature of time, which may be used to estimate the diffusion coefficient with impressive result.

We transfer this thinking to a 2D downsampling CNN. The goal of the 2D downsampling approach was inspired by the recovery curve method; to reduce the 3D data with dimensions (T, W, H) to a single-valued time series with (T) points, but the intent is for the network to learn an efficient downsampling strategy, see Figure 3.4. The method differs from for example the C3D and other 3D CNNs in that the time component is left intact, treating the time dimension as the channel dimension and applying 2D convolutions frame-wise on the sequence. This means that the spatio-temporal data must have a fixed number of frames. The FRAP data is then fed sequentially through convolutional blocks with ReLU activation and max pooling layers to produce a vector of size (T). To reduce noise, a 1D CNN is applied to the vector. The output of the 1D CNN is lastly fed to three fully connected layers with 1024 and 512 neurons and ReLU activation.

The architectural search for the downsampling neural network was aimed at finding a compromise between the design goal and the size of forward and backward passes. The details of the architecture, such as kernel sizes and stride settings, are available in Figure 3.3. Results from the architecture search emphasizes that the kernel sizes of the first CNN are not larger than 4, and that larger kernel sizes for the remaining CNNs is also detrimental to performance. The neural network also performs well without regularization layers such as dropout, which only seem to introduce randomness.

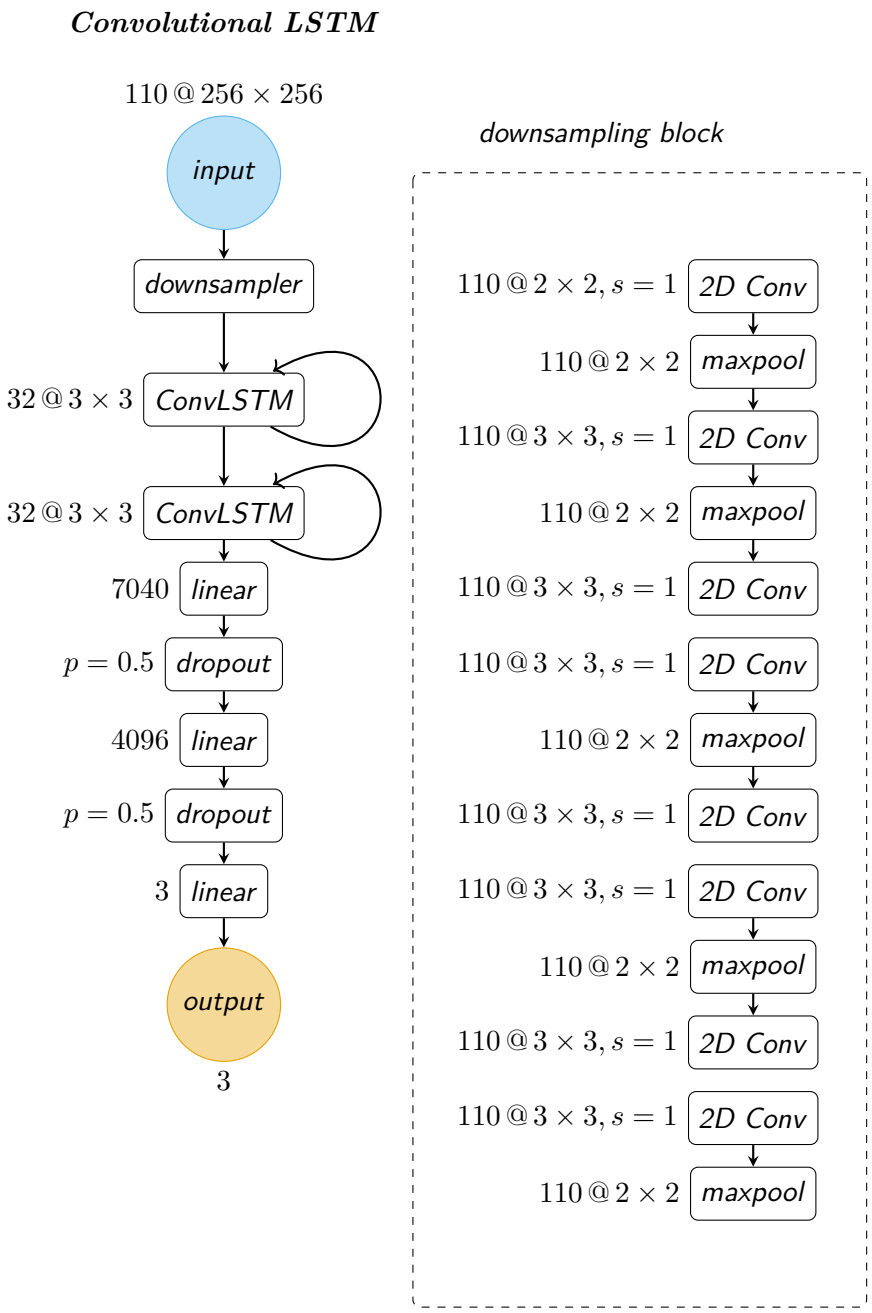


Figure 3.2: *Detailed architecture of the Convolutional LSTM and its downsampling block*

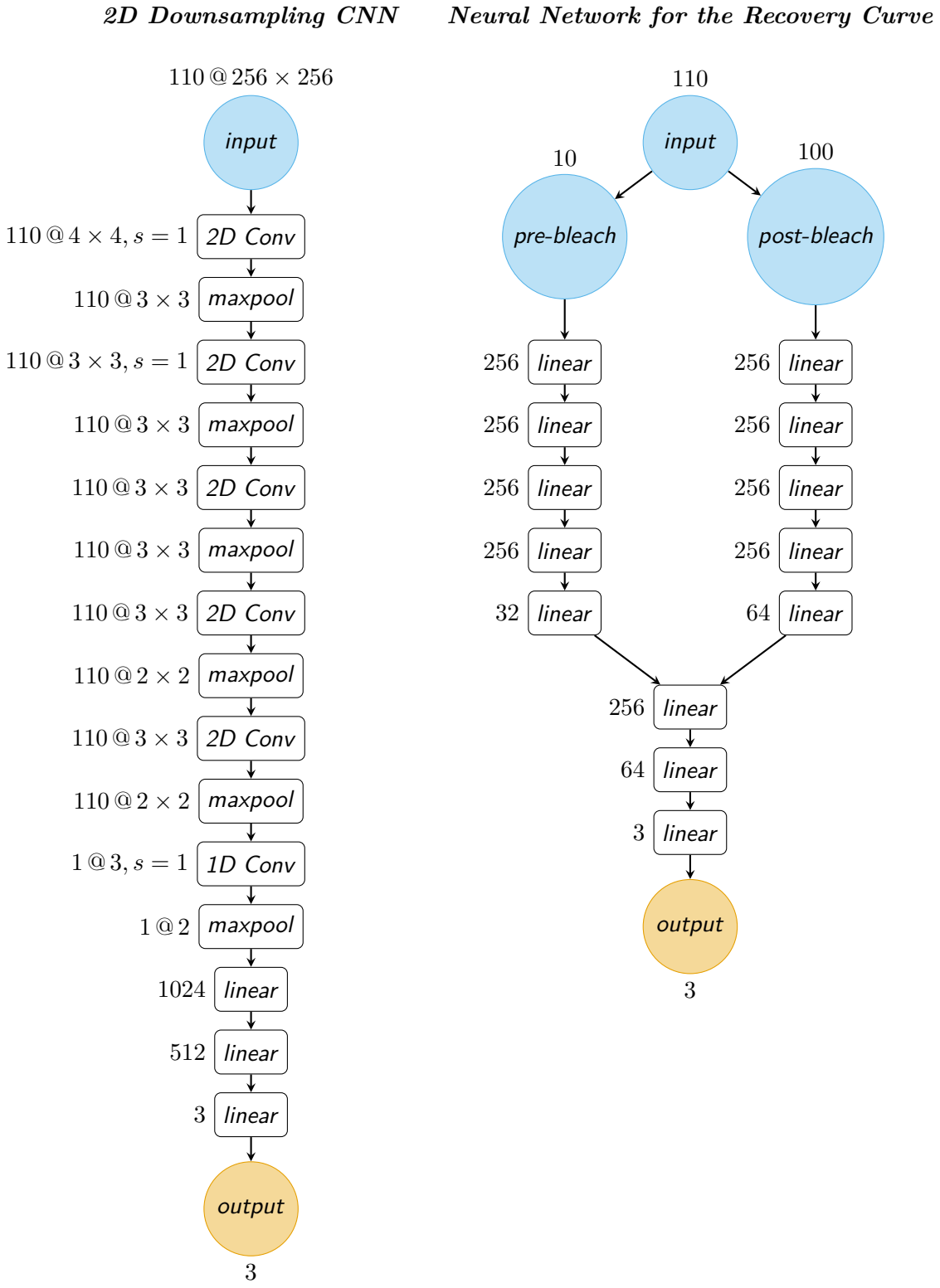


Figure 3.3: *Detailed architecture of the 2D Downsampling CNN (left) and the neural network for the recovery curve (right).*

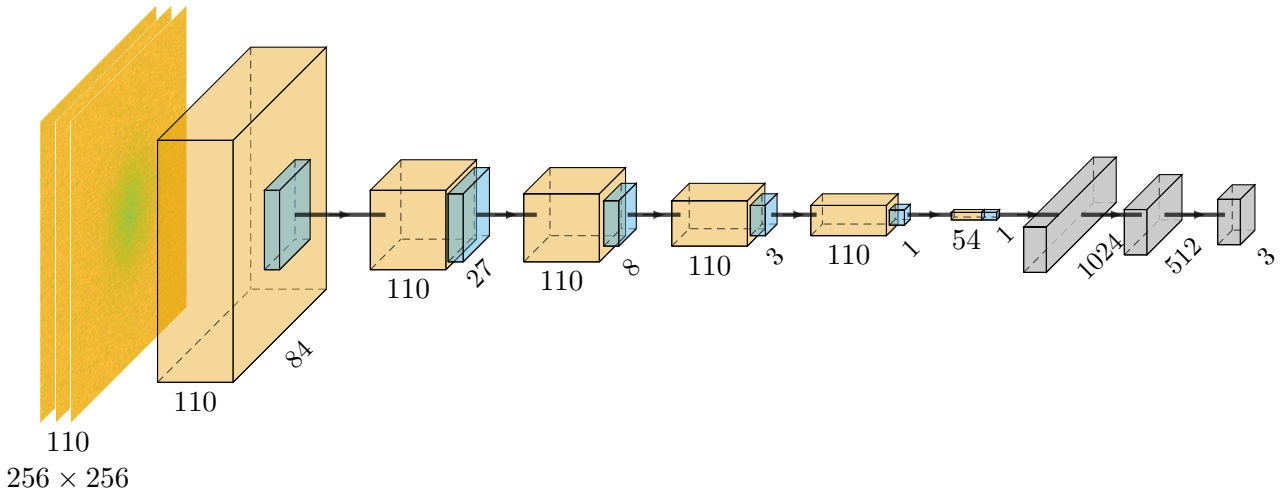


Figure 3.4: *Visualization of the 2D downsampling CNN. The neural network takes a sequence of 110 frames, downsampling the frame size in steps using 2D convolutional blocks followed by maxpooling. The data is reduced to a sequence of 110 values which is fed to a series of fully connected layers.*

3.2.4 Convolutional LSTM

A ConvLSTM neural network was constructed due to its apparent success on similar tasks in image recognition and video prediction. However, the ConvLSTM is computationally expensive for large images, in effect calculating $8 \times T$ 2D convolutions per sequence. Since $H = W = 256$ and $T = 110$ in general, the forward and backward passes occupy a lot of memory in the network.

For that reason, the sequence is first fed to a 2D downsampling block, see Figure 3.2, and the subsequent smaller features are fed to a convolutional LSTM network. The architecture of the downsampling block is an alternating series of 2D convolutions and maxpoolings, in order to extract as much information from the images as possible, while reducing the image size. The spatio-temporal data is passed from the downsampling block to two-layers of the ConvLSTM unit as pictured in Figure 2.9, in order to extract the temporal features of the sequence. Finally, the ConvLSTM is followed by two linear layers with 7040 and 4096 neurons each, each followed by a dropout layer with $p = 0.5$. The dropout was introduced to combat overfitting. Inserting dropout layers at any other position in the ConvLSTM makes learning noisy in this architecture.

The main compromise of the size of the input after downsampling was to accomodate 32 filters in the ConvLSTM. The number of filters in the ConvLSTM units were found to be very important for the performance of the neural network, but also a source of computational demand.

3.2.5 Early Trials

For the image data, trials were made using neural networks and weights pretrained on 2D images as initialization, for example resnet-18. This proved less successful than neural networks for video data with random initialization. The reason is likely the area of application: The weight kernels are learnt filters identifying corners, borders and colours present in ordinary image data. The FRAP data do not admit any of these features,

which can cause the filters to actually impede learning. Moreover, data augmentation is often introduced to achieve invariance towards lighting, noise, skewness and location in an image, but many of these augmentations are not naturally occurring in FRAP. Tests indicate that there was no significant gain or loss from introducing random occlusion in the frames, but it may have utility for real world FRAP data. In order for the neural network to perform on experimental data, it must be able to ignore errors and image artefacts from the microscope. One of these augmentation efforts is the constant noise a and proportional noise b in the simulation.

We also tried feature engineering similar to the recovery curve, extracting for example the variance of the intensity in the ROI, and calculating the mean intensity in disks with increasing radii from the centre. The main motivation for this approach was to reduce the dimensionality of the full image data while retaining more information than the recovery curve. However, fusing the engineered features in the neural networks is not trivial, and we saw no increase in loss performance.

3.3 Training Environment

The neural network models are trained on an NVIDIA Titan V and NVIDIA Titan Xp with similar performance. A simulated FRAP sample has dimensions fixed at $(C, T, W, H) = (1, 110, 256, 256)$, using a sampling time of 0.2 s and a realistic pixel size $\ell = 7.5 \times 10^{-7}$ m/px. The resulting size and dimensionality of the data is a notable limitation of the training, and a single sample occupies approximately 30 MB, and any sizeable dataset is impossible due to storage limitations. On the other hand, one $(110, 256, 256)$ video amounts to $256 \times 256 = 65536$ recovery curves with $(T) = (110)$, which is easily trained in a mini-batch scheme given by Algorithm 2. However, the computation and storage limitations of spatiotemporal data force us to reconsider the batch and mini-batch training procedures, given by Algorithm 1 and Algorithm 2. Training the network using the entire dataset $\mathcal{D}$ as a batch is not possible due to the size of the forward and backward pass. Moreover, for a smaller dataset, machine learning algorithms will find it difficult to generalize to the entire population.

Since the data is generated from the MATLAB simulation tool, our access to the distribution is unlimited. For that reason it is attractive to continuously generate more data. In the *online learning* technique in Algorithm 3 a small batch of $N_{\mathcal{B}}$ samples is generated and become available to the network sequentially. The network is trained for N_{iter} iterations, and a new small batch is generated every iteration. This would hypothetically allow the network to learn indefinitely.

However, a big problem with sequential learning for neural networks is *catastrophic forgetting*, see for example [44]. Catastrophic forgetting is the tendency to overwrite weights every iteration, and retaining too little information to generalize on the data. In the case of FRAP data, this method significantly lowered performance of the networks, likely due to this reason. In theory, regularization efforts such as dropout should alleviate catastrophic forgetting,[44] but in our experience not enough to justify its use.

We introduce a compromise between these methods in Algorithm 5, where a finite dataset $\mathcal{D}$ with approximately $N \sim 10^3$ samples is generated at the start of training. The network is then trained in mini-batch mode, with a fraction f of N new samples $\mathcal{D}_{\text{new}}$ generated at the end of each epoch. A random subset of fN samples in $\mathcal{D}$ are then updated with the values of $\mathcal{D}_{\text{new}}$. This method combines the learning benefits of batch training, with the regularizing effect of new data.

What is an appropriate value of the fraction f of new data? Setting $f = 1$ means

generating N new samples every epoch, which is computationally very inefficient. A compromise is to generate the new data in intervals of e.g. 100 epochs. However, tests indicate that this approach is very sensitive to overfitting on the first batch of data. This means f must balance overfitting on the initial batch of data and still have a regularizing effect. We find that updating $f = 0.05 = 5\%$ is a reasonable compromise between computational time and regularization.

The neural networks are all trained to minimize the mean square error loss function, as previously defined

$$\text{MSE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^2 = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) \tag{3.11}$$

for a predicted value $\hat{\boldsymbol{y}}$ and true value $\boldsymbol{y}$.

3.4 Hyperparameter Optimization

The neural networks were optimized with respect to the hyperparameters batch size $N_{\mathcal{B}}$, learning rate η and momentum m using a grid search with $N_{\text{epochs}} = 500$ epochs. The remaining hyperparameters, such as the number of layers, kernel sizes and neurons in the neural networks were chosen manually during the architecture search, taking computational complexity and loss in regard.

For the 2D downsampling net, the optimal values were found to be $N_{\mathcal{B}} = 8$, $\eta = 1 \times 10^{-4}$, and $m = 0.99$, which was in agreement with early tests. Similarly, the optimal values for the ConvLSTM were found to be $N_{\mathcal{B}} = 8$, $\eta = 1 \times 10^{-3}$, and $m = 0.99$. The best hyperparameters for the recovery curve network were found to be $N_{\mathcal{B}} = 32$, $\eta = 1 \times 10^{-5}$, and $m = 0.99$.

Data: Dataset $\mathcal{D}$ with N samples of $(\boldsymbol{x}, \boldsymbol{y})$ data.
Result: An estimated loss value.
for *epoch in* N_{epoch} **do**
  for $(\boldsymbol{x}, \boldsymbol{y})$ *in* $\mathcal{D}$ **do**
    Calculate loss;
  Update weights;

Algorithm 1: *The batch algorithm*

Data: Dataset $\mathcal{D}$ with N samples of $(\boldsymbol{x}, \boldsymbol{y})$ data.
Result: An estimated loss value.
Shuffle $\mathcal{D}$ into $N_{\mathcal{B}}$ mini-batches $\mathcal{B}$;
for *epoch in* N_{epoch} **do**
  for $\mathcal{B}$ *in* $\mathcal{D}$ **do**
    for $(\boldsymbol{x}, \boldsymbol{y})$ *in* $\mathcal{B}$ **do**
      Calculate loss;
    Update weights;

Algorithm 2: *The mini-batch algorithm*

Data: A simulation tool for the data.
Result: An estimated loss value.
for *iteration in* N_{iter} **do**
  Generate a batch $\mathcal{B}$ of $N_{\mathcal{B}}$ samples;
  for $(\boldsymbol{x}, \boldsymbol{y})$ *in* $\mathcal{B}$ **do**
    Calculate loss;
  Update weights;

Algorithm 3: *The online algorithm*

Data: A simulation tool for the data.
Result: An estimated loss value.
for *iteration in* N_{iter} **do**
  Generate a batch $\mathcal{D}$ of N samples;
  Shuffle $\mathcal{D}$ into $N_{\mathcal{B}} < N$ mini-batches $\mathcal{B}$;
  for *epoch in* N_{epoch} **do**
    for $\mathcal{B}$ *in* $\mathcal{D}$ **do**
      for $(\boldsymbol{x}, \boldsymbol{y})$ *in* $\mathcal{B}$ **do**
        Calculate loss;
      Update weights;

Algorithm 4: *The mixed online-batch algorithm*

Data: A simulation tool for the data.
Result: An estimated loss value.
Generate a batch $\mathcal{D}$ of N samples;
Shuffle $\mathcal{D}$ into $N_{\mathcal{B}} < N$ mini-batches $\mathcal{B}$;
for *epoch in* N_{epoch} **do**
  for $\mathcal{B}$ *in* $\mathcal{D}$ **do**
    for $(\boldsymbol{x}, \boldsymbol{y})$ *in* $\mathcal{B}$ **do**
      Calculate loss;
    Update weights;
  Generate a fraction f of the N samples as $\mathcal{D}_{\text{new}}$;
  Update a random subset fN of $\mathcal{D}$ to the newly generated data $\mathcal{D}_{\text{new}}$;

Algorithm 5: *Proposed training method*

4 Results

4.1 Model Training, Validation and Test Results

The downsampling CNN and ConvLSTM were trained using the proposed scheme in Algorithm 5 using $N_{\text{train}} = 4096$ initial samples for $N_{\text{epochs}} = 2 \times 10^3$ epochs and a fraction $f = 0.05$ of new samples each epoch, and a mini-batch size $N_{\mathcal{B}} = 8$. The learning rate $\eta = 1 \times 10^{-3}$ and $\eta = 1 \times 10^{-4}$ for the downsampling CNN and ConvLSTM respectively, and both used Nesterov momentum $m = 0.99$. A validation dataset with $N_{\text{val}} = 0.25 N_{\text{train}} = 1024$ was used to prevent overfitting during training. The neural network for the recovery curve was trained with $N_{\text{train}} = 65536, N_{\text{val}} = 0.25 N_{\text{train}} = 16384$ samples using Algorithm 2. As a performance test, a test set with $N_{\text{test}} = 2048$ samples was generated to compare the neural networks. The data was generated with target distributions given by the discussion in Section 2.2.2.2 and Section 3.1: The diffusion coefficient $D \sim \text{LogUniform}(10^{-12}\,\text{m}^2/\text{s}, 10^{-9}\,\text{m}^2/\text{s})$ covers the most common cases in food science with the exception of very slow diffusion, such as that of cells. Henceforth we will consider the diffusion coefficient in units of px^2/s, given the pixel size $\ell = 7.5 \times 10^{-7}\,\text{m/px}$.

The diffusion coefficient is rescaled to $\log_{10}(D)$ to ensure all target parameters are of the same order of magnitude. The initial concentration $c_0 \sim \text{Uniform}(0.5, 1)$ and the bleach parameter $\alpha \sim \text{Uniform}(0.5, 0.95)$ ensure a sufficiently strong signal. The combined vector of target parameters is defined $\boldsymbol{y} = [\log_{10}(D), c_0, \alpha]$.

Table 4.1: *Distribution of target parameter values in the training and validation data, chosen to cover common values encountered in FRAP experiments.*

parameter	distribution
D	$\text{LogUniform}(10^{-12}\,\text{m}^2/\text{s}, 10^{-9}\,\text{m}^2/\text{s})$
c_0	$\text{Uniform}(0.5, 1)$
α	$\text{Uniform}(0.5, 0.95)$

The simulated data has fixed height and width $H = W = 256$, and a padding $M = 128$ which is cropped after data generation. The sampling speed was $\Delta t = 0.2\,\text{s}$, using $T_{\text{prebleach}} = 10$ prebleach frames, $T_{\text{bleach}} = 4$ bleach frames, and $T_{\text{postbleach}} = 100$ postbleach frames, hence $T = 110$ frames in total in the sequence, omitting the bleach frames.

The loss function for all methods was the mean squared error, MSE, and the performance of an estimation $\hat{\boldsymbol{y}}$ is given as the $\text{MSE}(\boldsymbol{y}, \hat{\boldsymbol{y}})$ calculated on the test set. It can be shown that the mean squared error is the sum of the variance and bias of an estimate $\hat{\boldsymbol{y}}$:

$$\begin{aligned}
\text{MSE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) =& \mathbb{E}\left[(\boldsymbol{y} - \hat{\boldsymbol{y}})^2\right] \\
=& \mathbb{E}\left[(\mathbb{E}[\hat{\boldsymbol{y}}] - \hat{\boldsymbol{y}})^2\right] + \left(\mathbb{E}[\hat{\boldsymbol{y}}] - \boldsymbol{y}\right)^2 \\
=& \text{Var}(\hat{\boldsymbol{y}}) + \text{Bias}^2(\boldsymbol{y}, \hat{\boldsymbol{y}})
\end{aligned}$$

Table 4.2: *Values for the hyperparameters used in generating the data, training and optimizing the neural networks.*

hyperparameter	Downsampling CNN	ConvLSTM	Recovery Curve Network
N_{train}	4096	4096	65536
N_{val}	1024	1024	16384
N_{test}	2048	2048	2048
H	256	256	1
W	256	256	1
T	110	110	110
f	0.05	0.05	NA
$N_{\mathcal{B}}$	8	8	32
η	1×10^{-3}	1×10^{-4}	1×10^{-4}
m	0.99	0.99	0.99
N_{epochs}	2×10^{3}	2×10^{3}	2×10^{3}

Hence, we define the standard deviation $\text{std}(\hat{\boldsymbol{y}}) = \sqrt{\text{Var}(\hat{\boldsymbol{y}})}$ as another measure of interest, since it defines a precision of the estimates regardless of the true values $\boldsymbol{y}$, and helps us quantify the bias of the estimation.

Figure 4.1 shows the loss from training and validation of the neural networks after 2000 epochs, which translates to 9 and 10 days of training for the spatio-temporal networks and approximately 6 hours for the recovery curve network, in our environment. The downsampling network validation error follows the training error closely but with higher variance, and indicates that the model is still underfit. The validation error is approximately 7×10^{-4}. The convolutional LSTM converges significantly slower, and reaches a noisy validation error of approximately 3×10^{-3}. Lastly, the recovery curve network converges fairly quickly to 2×10^{-3}, with little noise in the validation error. This is expected from the complexity difference of fitting the spatio-temporal versus the temporal data with NNs.

Table 4.3: *The test MSE on 2048 samples of the neural networks, given as parameter-wise MSE and total MSE. The downsampling 2D CNN outperforms the ConvLSTM and recovery curve network, but the ConvLSTM is notably just as good at estimating the bleach parameter α.*

method	$\text{MSE}(\log_{10} \hat{D})$	$\text{MSE}(\hat{c}_0)$	$\text{MSE}(\hat{\alpha})$	$\text{MSE}(\hat{\boldsymbol{y}})$
Downsampling CNN	7×10^{-4}	9×10^{-4}	8×10^{-4}	8×10^{-4}
ConvLSTM	3.7×10^{-3}	2.8×10^{-3}	7×10^{-4}	2.4×10^{-3}
Recovery Curve Network	6.0×10^{-3}	1.0×10^{-5}	1×10^{-3}	2.3×10^{-3}

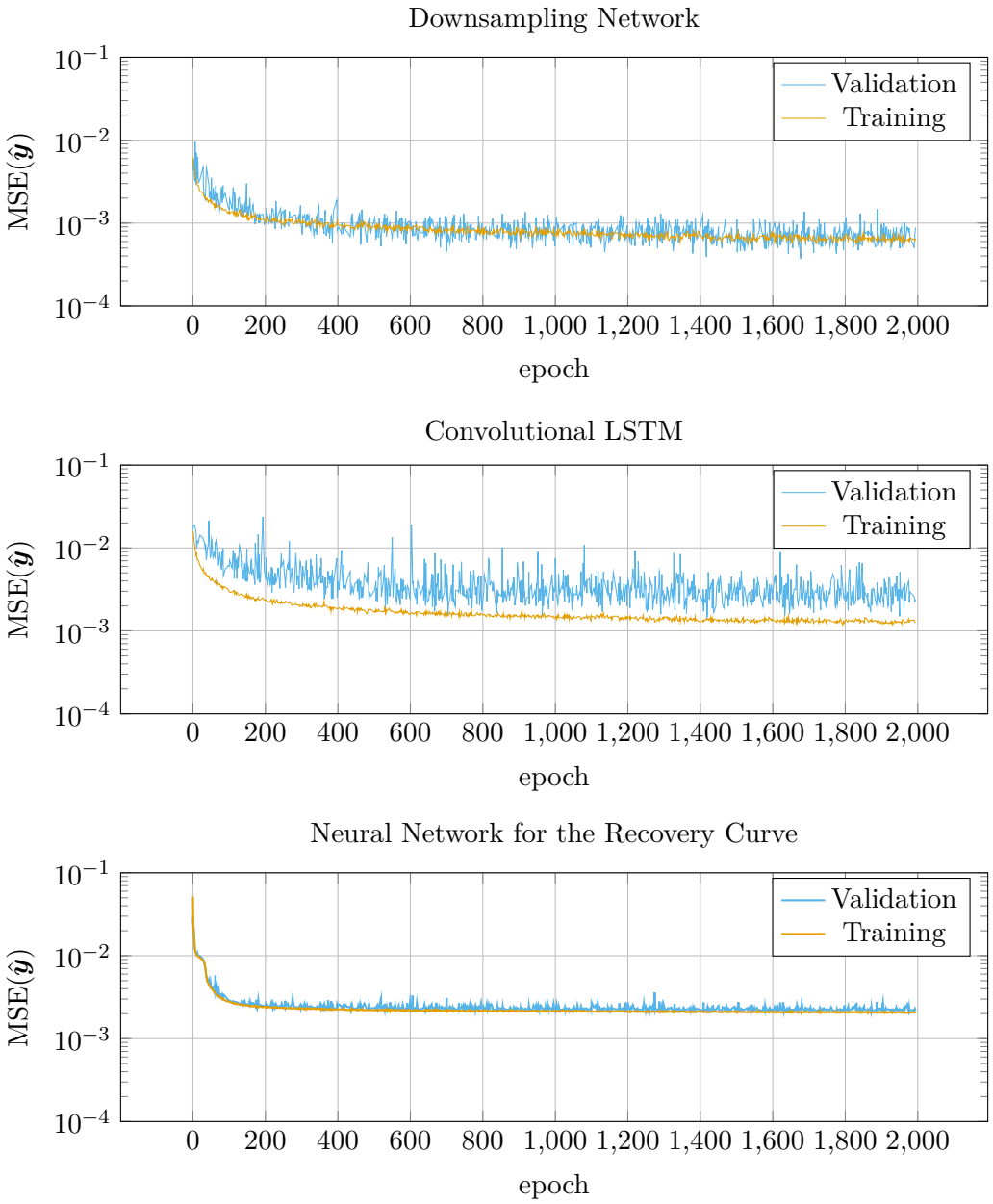


Figure 4.1: *The validation and training loss for the downsampling network (first from the top), ConvLSTM (second) and neural network for the recovery curve (bottom) after 2000 epochs. The downsampler loss is noisy, but mostly due to its small magnitude. The training is also notably underfitted. The ConvLSTM on the other hand converges notably slower, with noisy validation. The neural network for the recovery curve converges quickly and with little noise.*

The results from the test set are given by Table 4.3, divided into the parameter-wise mean squared error for D, c_0, α and the total MSE. The most interesting case is the diffusion coefficient, where the downsampling CNN yields $\text{MSE}(\log_{10} \hat{D}) = 7 \times 10^{-4}$ versus the Conv LSTM $\text{MSE}(\log_{10} \hat{D}) = 3.7 \times 10^{-3}$. Figure 4.1 seems to indicate that the downsampling CNN will retain better performance than the other neural networks after further training. The recovery curve network obtains $\text{MSE}(\log_{10} \hat{D}) = 2 \times 10^{-3}$ and thus performs

slightly better than ConvLSTM overall. LSTMs are notoriously difficult to train [20], which may be the cause of the poor convergence. It is also possibly due to poor feature extraction: Ideally, the features passed through the ConvLSTM would be the full spatial data, but memory constraints force us to downsample the image. The test set results indicate that the ConvLSTM makes other prioritizations compared to the downsampling CNN, namely a relatively good loss on α compared with the total loss. Since α is the ratio between the mean concentration in the ROI before and after bleaching, it is possible that the ConvLSTM captures this temporal relationship better. The ConvLSTM also trains slower than its 2D CNN counterpart, likely due to backpropagation through time loops over every frame as well as over each sample. The accumulation of gradient updates puts strain on the GPU. The training time combined with the poor convergence makes it ill-suited for our training environment. The ConvLSTM does have a few attractive properties, however: It could potentially be trained on variable length spatio-temporal sequences, which is useful for experimental applications. It also makes fundamentally different estimations compared with the downsampler, and seems to prioritize the bleach parameter over the initial concentration and diffusion coefficient. The downsampling neural network is limited to input sequences with the number of frames on which it is trained, but makes better estimates as a whole. Henceforth we will only consider the downsampling 2D CNN (*NN-PX*), or *downsampler*, and the neural network for the recovery curve (*NN-RC*) for comparison with least-squares on pixel-data (*LS-PX*) and recovery curve data (*LS-RC*).

4.2 Comparison between Neural Networks and Least Squares

A dataset was generated to compare the neural network approaches to the least square method in different domains of noise, bleaching and diffusion rate. The simulation settings were also updated to mimic the Leica SP5 CLSM (Leica, Heidelberg, Germany), pixel size and sampling frequency, set to $\ell = 7.598 \times 10^{-7}\,\mathrm{m/px}, \Delta t = 0.265\,\mathrm{s}$ respectively. The data was generated with the following values of the target parameter values, $D \in \{5\times10^{-12}, 1\times10^{-11}, 5\times10^{-11}, 1\times10^{-10}, 5\times10^{-10}\}\ \mathrm{m^2/s}$, which given a pixel size $7.598\times10^{-7}\,\mathrm{m}$ is equivalent to $\log_{10}(D) \approx \{0.9376, 1.2386, 1.9376, 2.2386, 2.9376\}\ \log_{10}\mathrm{px^2/s}$. The concentration takes the value $c_0 = 0.75$, the bleaching depth $\alpha = \{0.5, 0.6, 0.7, 0.8, 0.9\}$, and the level noise $a = \{0.001, 0.005, 0.01, 0.05, 0.1\}$.

The NN-PX model was trained on $N_{\text{train}} = 16384, N_{\text{val}} = 4096, N_{\mathcal{B}} = 16$, for $N_{\text{epochs}} = 1000$ and the remaining parameters like in Table 4.2. The NN-RC model was trained with the same hyperparameters as before. The neural networks were trained as in Section 4.1 with test results given by Table 4.4. We observe that increasing the number of training samples significantly improves estimation of c_0 and α. The recovery curve and pixel-based least square methods were minimized by means of `fmincon` in MATLAB. The best initial guesses for the routine were chosen by a random search.

Table 4.4: *The test MSE on 2048 samples of the NN-PX and NN-LS methods, given as parameter-wise MSE and total MSE.*

method	$\mathrm{MSE}(\log_{10}\hat{D})$	$\mathrm{MSE}(\hat{c}_0)$	$\mathrm{MSE}(\hat{\alpha})$	$\mathrm{MSE}(\hat{\boldsymbol{y}})$
NN-PX	3×10^{-4}	6.74×10^{-5}	4×10^{-4}	2×10^{-4}
NN-RC	7×10^{-3}	8.9×10^{-6}	9×10^{-3}	2.6×10^{-3}

In the following sections we will present the best and worst case scenarios for the diffusion

coefficient D and the bleach parameter α on simulated data, omitting the results for the initial concentration c_0 on which all methods are known to perform well. The MSE, standard deviation and mean of the estimates for the entire test set and all parameters is available as supplemental information online. Lastly we will validate the methods on experimental FRAP data.

4.2.1 Results for the Diffusion Coefficient

The MSE for the diffusion coefficient D is given by Figure 4.2 and Figure 4.4, as a function of the noise level a, for varying values of D, at extreme values for $\alpha = 0.5, 0.9$. The diffusion coefficient results for the intermediary values for α behave monotonically in that range. These indicate that the error for least square fits increases monotonically with higher noise, as expected. The same applies for increasing values of the diffusion coefficient, higher rates of diffusion are potentially hard to catch, and difficult to distinguish from noise. The pixel-based method also outperforms the other methods with up to two orders of magnitude. The MSE of the downsampler is in the range $[10^{-5}, 10^{-2}]$ for $\alpha = 0.5$ and $[10^{-4}, 10^{-2}]$ for $\alpha = 0.9$, but does not show the same sensitivity to the diffusion rate or the noise levels as the least squares fits. CNNs are noise reducing by design, since the neural network learns to form filters such as averaging and smoothing filters. Similarly, the recovery curve network makes estimates almost independently of the noise, albeit with an MSE in the order of 10^{-2}. The extraction of the recovery curve is noise reducing by design, but this method is more robust nonetheless.

The standard deviation of the estimates of D on the test set is seen in Figure 4.3, as a function of a and D, with $\alpha = 0.5$. Whilst the results from Figure 4.2 indicate that the MSE of the estimates does not increase with noise, the standard deviation shows that all estimates become less precise with increasing noise. This should mean that the lower MSE of the downsampler network is due to bias. It is noteworthy that the standard deviation of the neural networks is more robust in the range of diffusion coefficients than the conventional methods, and distinguishably lower than the NN-LS estimates. The high standard deviation of the least square recovery curve estimates is not necessarily surprising: Discarding a lot of information in favour of the bleach region should introduce more uncertainty.

Comparing Figure 4.2-4.4 and Figure 4.3-4.5 we see that all estimates consistently get worse with larger α. This pattern holds for $\alpha = 0.6, 0.7, 0.8$ as well, see the supplementary data for more information. Large α translates to a lower bleaching depth, which will have a less steep recovery of fluorescence compared to more bleaching. Since the recovery is essential for estimating D, it is expected that less bleaching yields worse errors.

4.2.2 Results for the Bleach Parameter

We also consider the MSE results for the bleach parameter as a function of noise for the two extreme values $D = 0.95 \log_{10} \text{px}^2/\text{s}, 2.95 \log_{10} \text{px}^2/\text{s}$. Like in the case of the diffusion coefficient, the results for α behave monotonically with D, and the extreme cases thus serve as best and worst case scenarios for all methods. The results are shown in Figure 4.6 and Figure 4.7 respectively, which again indicate that a higher diffusion rate is associated with higher MSE. Refer to the supplementary for the intermediary values of D. Like in Section 4.2.1 the pixel-based least squares estimation has lower MSE in general compared to the rest of the methods, but the recovery curve-based least squares method performs almost equally well. The NN-PX model is worse at estimating α than D, which is also verified by its test set results in Table 4.3, whilst NN-RC performs consistently across both

parameters. However, the neural networks again demonstrate robustness against noise, while the errors of the least square estimates increase consistently with the noise.

Contrary to when estimating D, the estimation of the bleaching parameter improves monotonically with higher α. This is possibly due to that a great deal of information on α is contained in the first post-bleach frame. For stronger bleaching, the difference between the last bleach frame and first post-bleach frame due to diffusion is greater, which impedes estimation of the true bleaching depth.

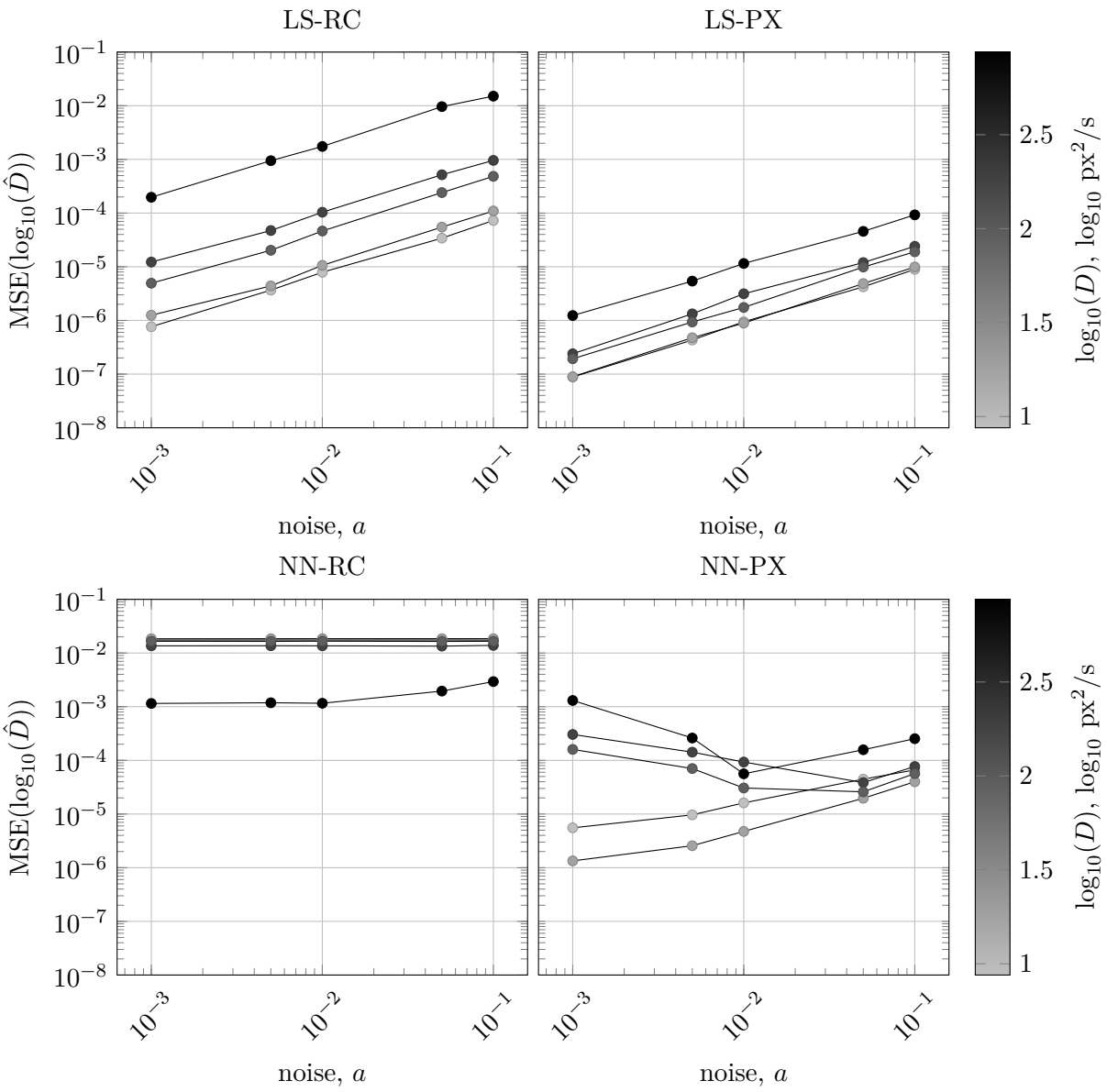


Figure 4.2: *The mean squared error of D (in $\log_{10}\mathrm{px}^2/\mathrm{s}$) versus the noise level a, for $\alpha = 0.5$. The MSE increases monotonically with noise and diffusion coefficient for the least squares methods. The neural networks have slightly higher MSE in general, but seem more robust to noise and value of D.*

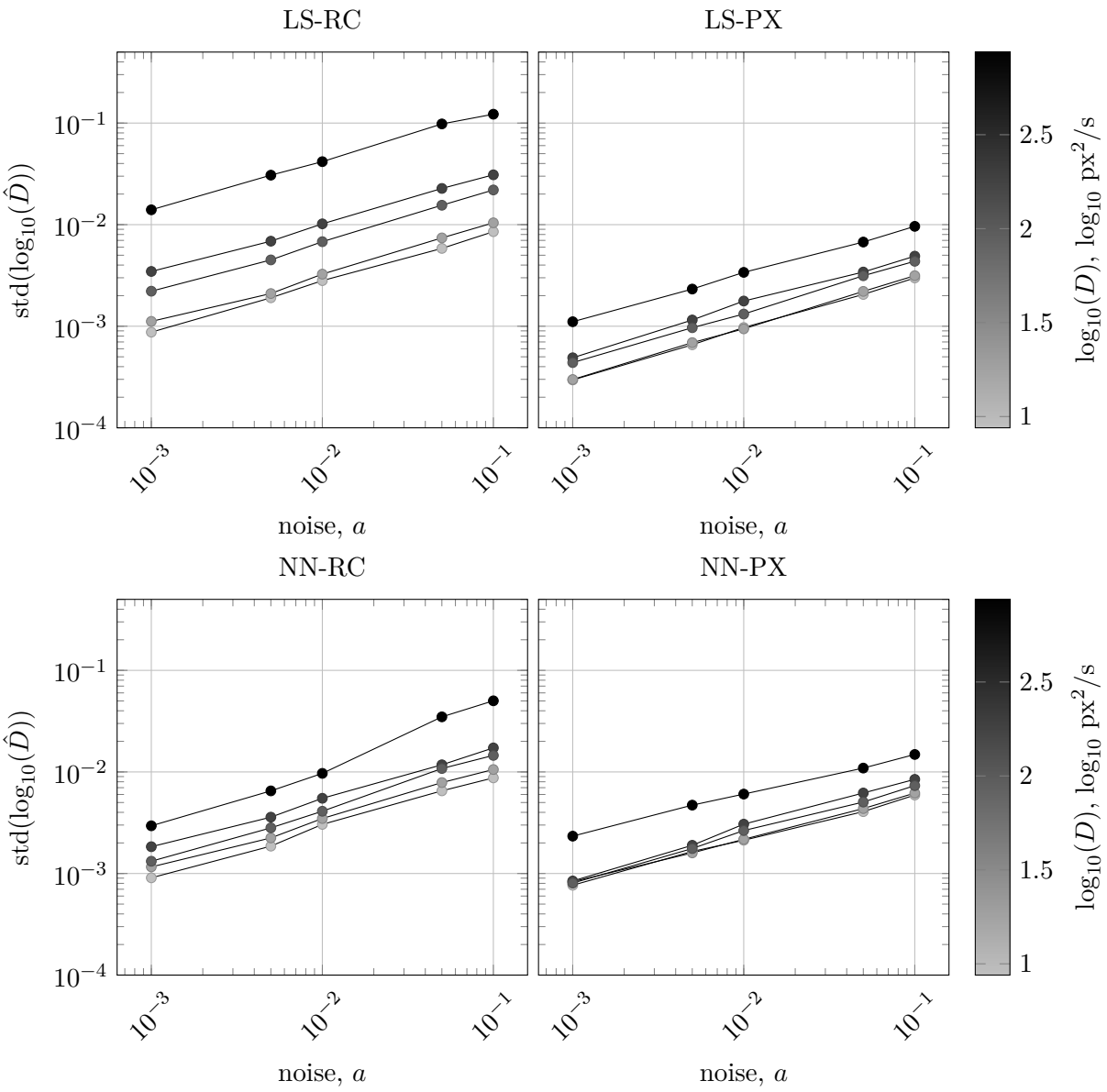


Figure 4.3: *The standard deviation of the estimations of D (in $\log_{10}\,\mathrm{px}^2/\mathrm{s}$) the noise level a, for $\alpha = 0.5$. The results indicate that the neural network makes more consistent estimations of D compared to the recovery curve method, and that the standard deviation increases with higher noise levels.*

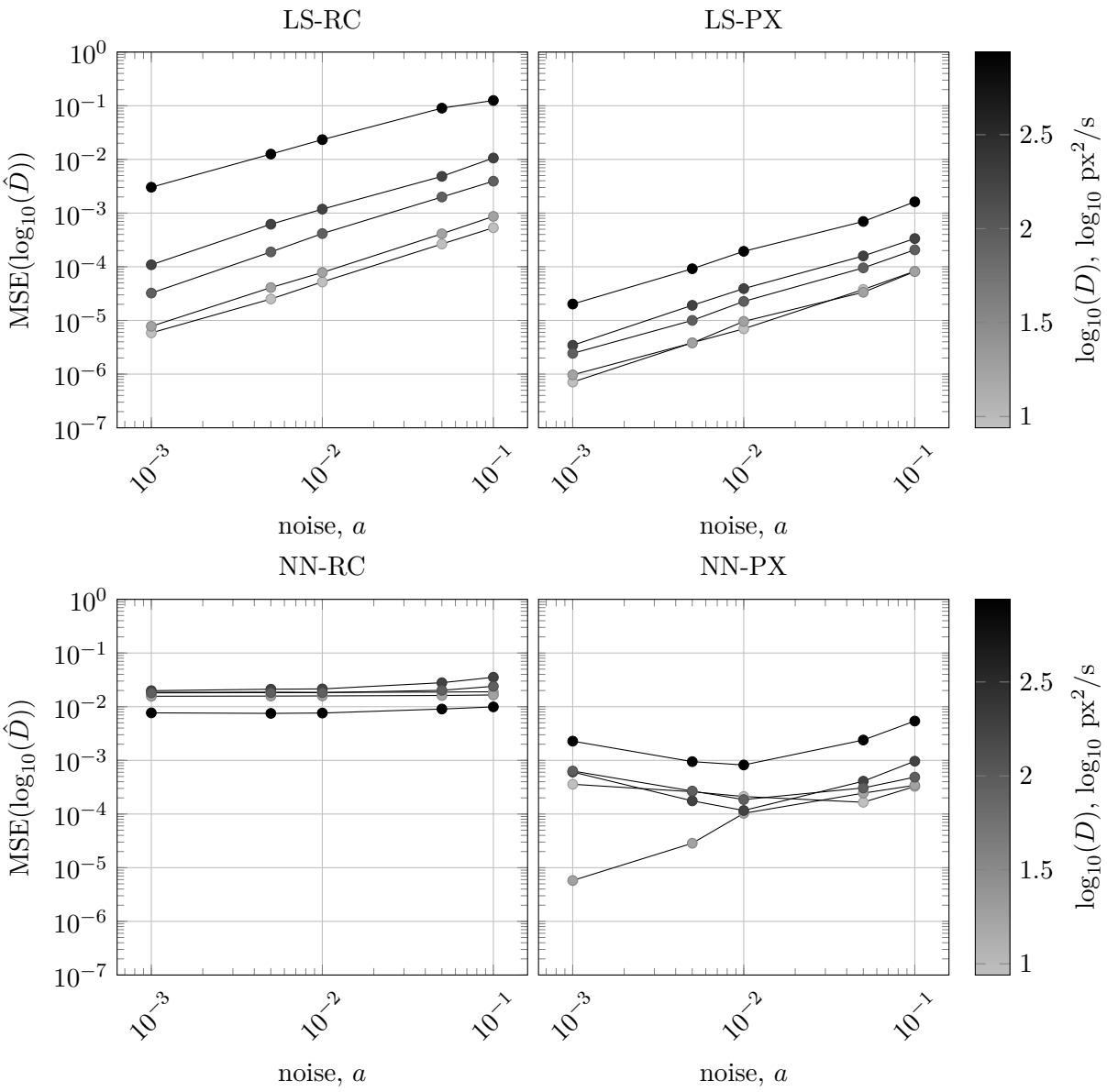


Figure 4.4: *The mean squared error of D (in $\log_{10}\,\mathrm{px}^2/\mathrm{s}$) versus the noise level a, for $\alpha = 0.9$. The results indicate as expected that the MSE increases monotonically with noise and diffusion coefficient for the least squares methods. The neural networks have slightly higher MSE in general, but seem more robust to noise and value of D.*

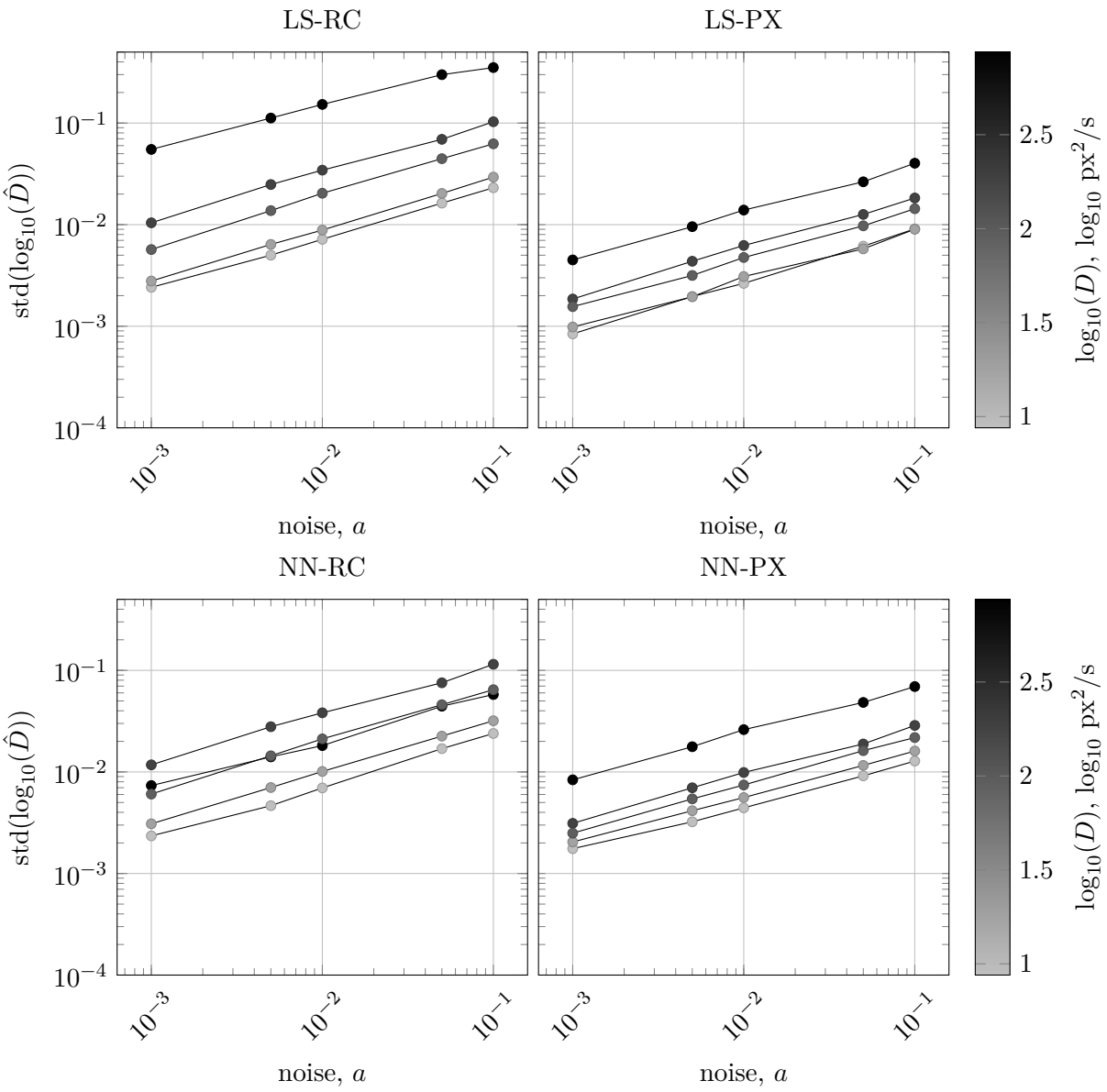


Figure 4.5: *The standard deviation of the estimations of D (in $\log_{10}\mathrm{px}^2/\mathrm{s}$) versus the noise level a, for $\alpha = 0.9$. The results indicate that the downsampler neural network makes more consistent estimations of D compared to the recovery curve method, and that the standard deviation increases with higher noise levels.*

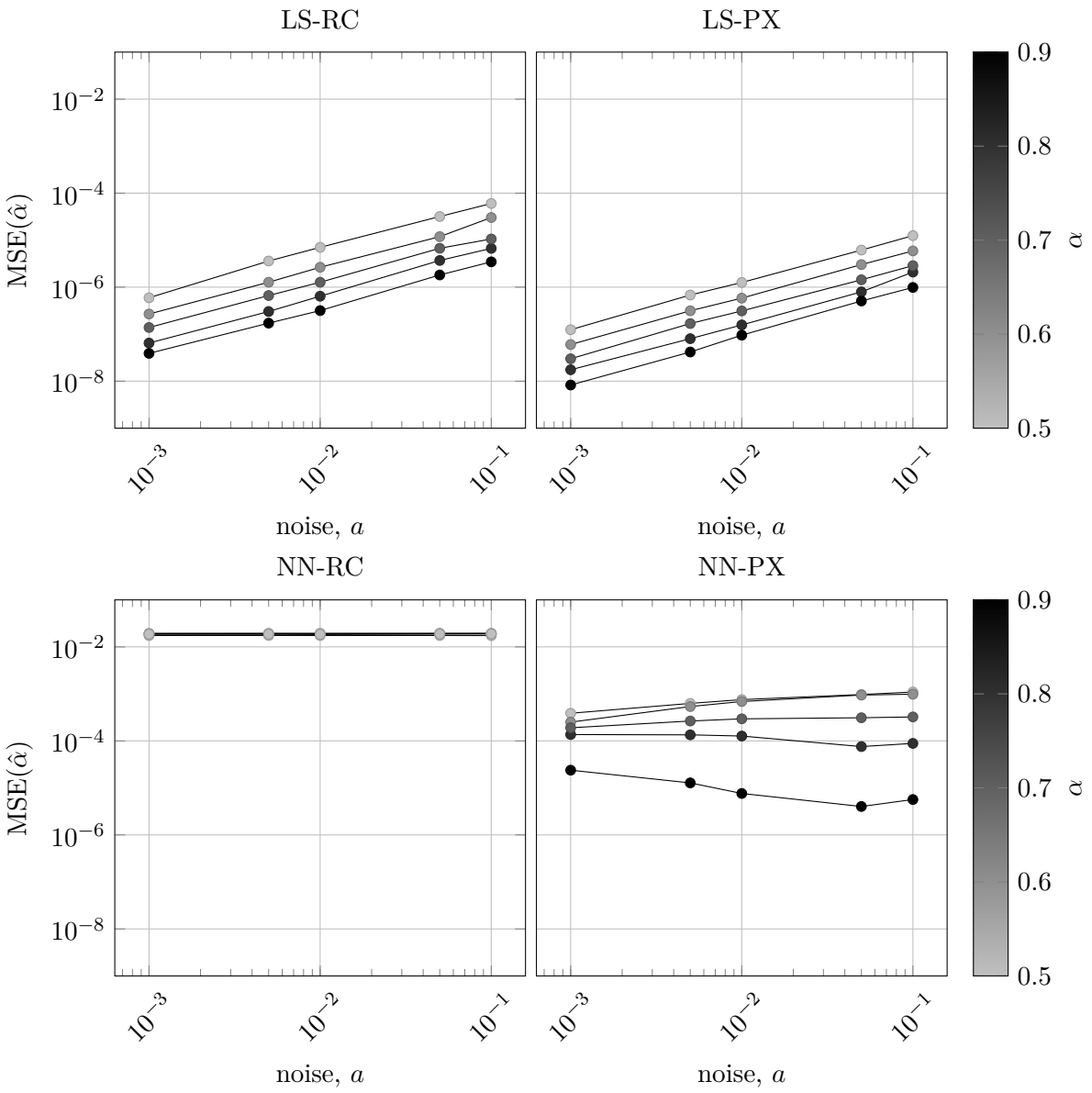


Figure 4.6: *The MSE of a few values of α as a function of noise a, for a diffusion rate of $10^{-12}\,\mathrm{m}^2/\mathrm{s}$. The least square-based estimates obtain a lower error than the neural networks in general, but the neural networks shows robustness to noise, while the least square methods get worse estimates at higher noise levels.*

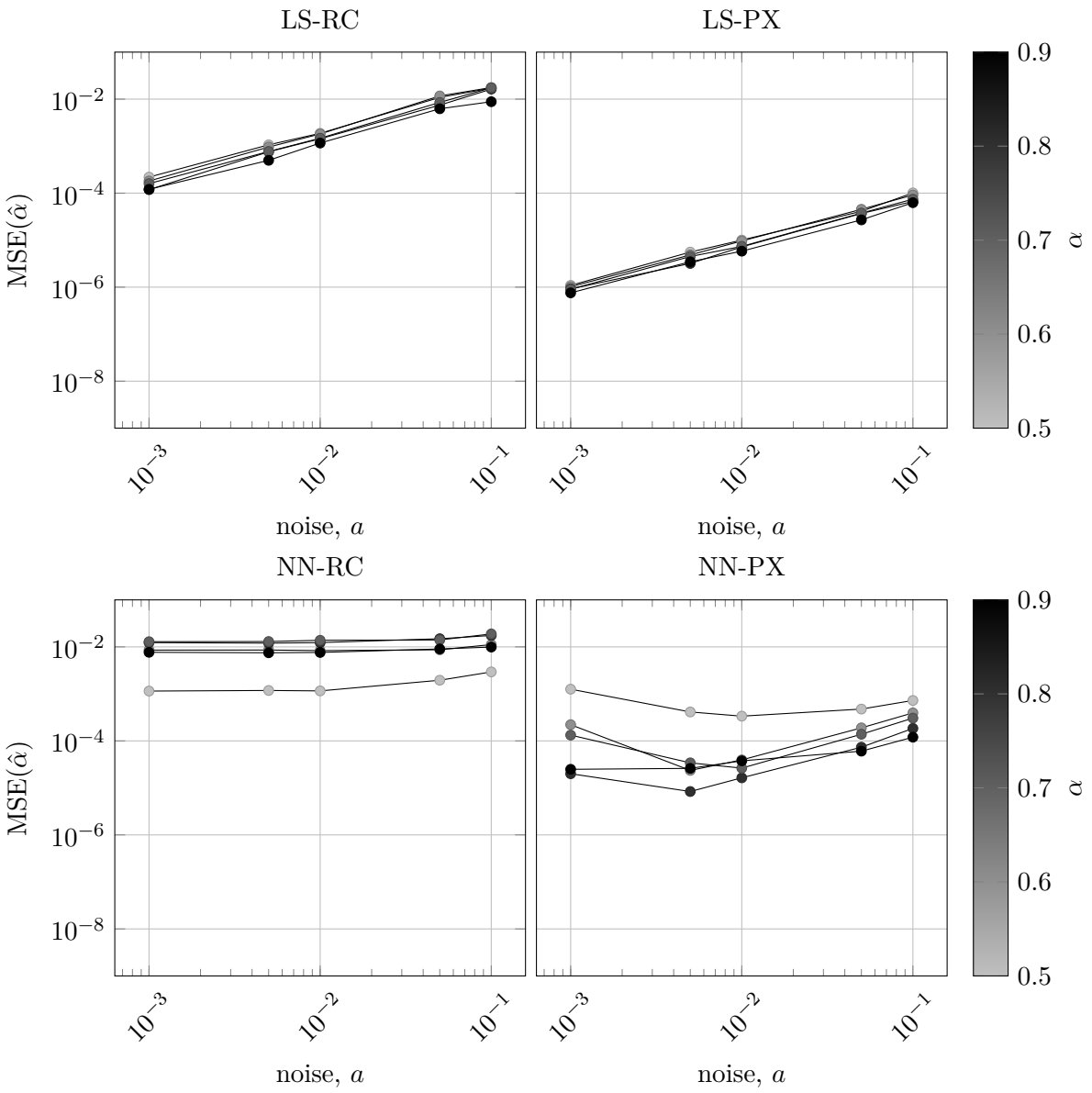


Figure 4.7: *The MSE of a few values of α as a function of noise a, for a diffusion rate of $10^{-9}\,\mathrm{m}^2/\mathrm{s}$. In high diffusion rates, the recovery curve-based least squares gets a higher error than both pixel-based one and the downsampler. The neural network MSE values do not increase with the noise, indicating a robustness against the noise level.*

4.2.3 Bias in the Parameter Estimation

Figure 4.8 shows a sample of estimates $\hat{D}_{\text{NN-PX}}$, $\hat{D}_{\text{LS-PX}}$ and $\hat{\alpha}_{\text{NN-PX}}$, $\hat{\alpha}_{\text{LS-PX}}$ for the downsampler and pixel-based least squares respectively, where $D = 1.2386\ \log_{10} \text{px}^2/\text{s}$ and $\alpha = 0.7$ respectively, and $a = 0.05$ for both parameters. While the least squares method consistently makes better estimates than the downsampler, the latter seems to have a bias towards lower values of the diffusion coefficient, and higher values of the bleach parameter α, which confirms the suspicions about bias raised in Section 4.2.

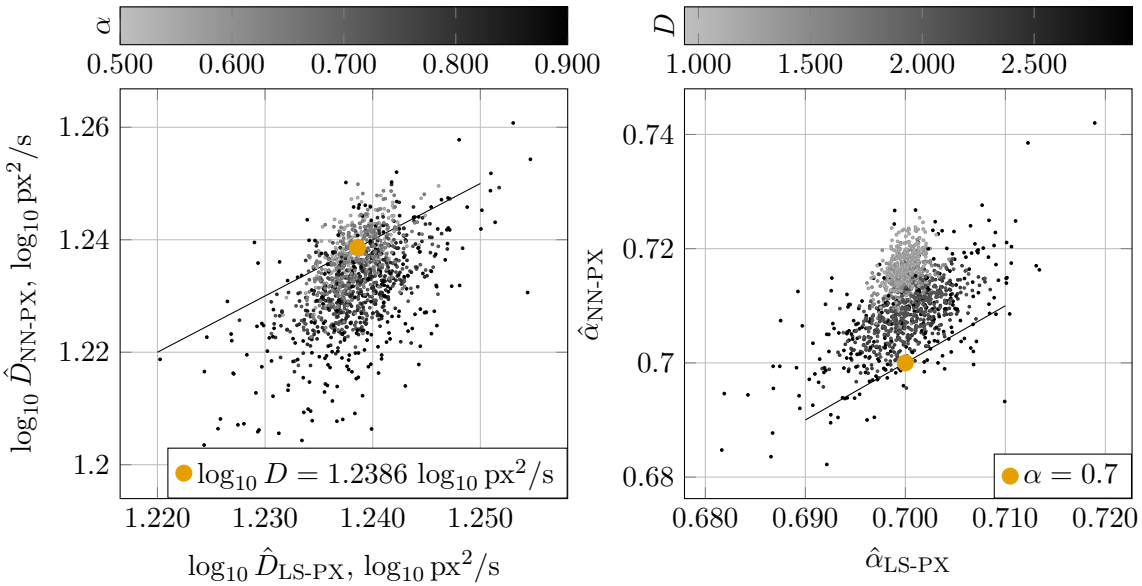


Figure 4.8: *The downsampler neural network estimates versus the pixel-based least squares estimates of D (left) and α (right), where $D = 1.2386\ \log_{10} \text{px}^2/\text{s}, \alpha = 0.7, a = 0.05$. The downsampler consistently overestimates D compared to LS, and less so underestimates α.*

The exact cause of the bias is unknown. However, more bleaching and higher rates of diffusion are reasonably more difficult to estimate: Higher rates of diffusion means more loss of information between FRAP frames, and likewise does more bleaching lead to higher loss of information between the last bleach frame and first post-bleach frame, one time step apart.

4.2.4 Execution Time

A benefit of the neural networks over least squares is that the latter is an iterative method, and must be initialized reasonably closed to a minimum to converge. Rpeatedly testing random initial guesses is costly in time, and even then the final iterations to the minimum are required. A pretrained neural network only requires a forward pass of the data. The mean execution time of the neural networks were calculated by $n = 500$ repeated predictions, with the results shown in Figure 4.9. The downsampler is a large CNN and executes in 152 ± 3 ms, while the smaller recovery curve network runs in just 1.01 ± 0.137 ms. For the least squares estimations, $n = 500$ simulations with the same settings as in Table 4.1 and Table 4.2 were performed, yielding 79631 ± 10819 ms for the pixel-based method and 59070 ± 6485 ms for the recovery curve-based estimation. The neural networks are thus significantly faster, and potentially easier to use a priori, when very little is known beforehand on the range of the parameters.

Speed may be not be a priority in real world applications, however, where accuracy is most important. Pixel-based least squares is more accurate in terms of all the target parameters, but choosing an appropriate initial guess is still non-trivial: The accuracy and speed of the downsampler solution makes it an appropriate initial guess for least-squares.

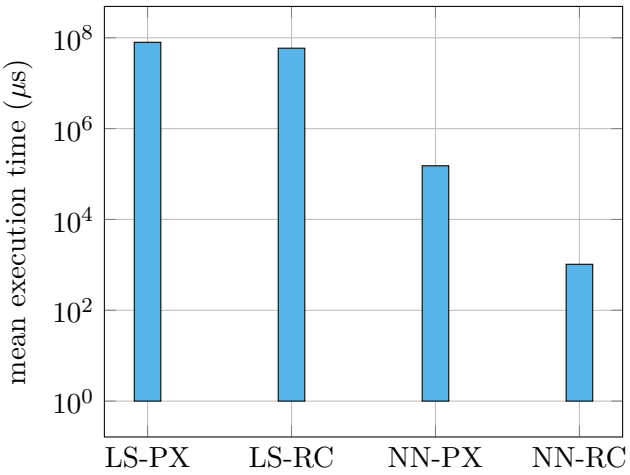


Figure 4.9: *The mean execution time in milliseconds for 500 estimations for the least squares-based estimations (LS-PX, LS-RC) and neural networks (NN-PX, NN-RC). Since the method of least squares is an iterative method, it executes up to 2 orders of magnitudes slower than the neural networks.*

4.2.5 Experimental Validation

It is interesting to validate the performance of the least square methods with the neural networks in a controlled setting.

The experiment is similar to that of Röding et al. [7]. The sample solution consisted of water and 0.01 wt% sodium fluorescein salt (Sigma-Aldrich, St. Louis, MO) with a diffusion coefficient $D_{\text{ref}} = 4\times10^{-10}\,\text{m}^2/\text{s}$ [45, 46], where wt% denotes *percentage by weight*. To validate the estimation methods in different diffusion rate scales, two sample solutions were prepared: 32 wt% sucrose, and the second with 56 wt% sucrose, expected to lower the fluorescein diffusion rate by a factor 4 and 40 respectively. The true diffusion coefficients were then calculated to be approximately $D_{32\,\text{wt\%}} = 1 \times 10^{-10}\,\text{m}^2/\text{s}$ and $D_{56\,\text{wt\%}} = 1 \times 10^{-11}\,\text{m}^2/\text{s}$. A total of 20 and 21 replicate measurements were taken at different locations in the 32 wt% and 56 wt% samples respectively.

The experiment was performed in ambient temperature and pressure on a Leica SP5 CLSM (Leica, Heidelberg, Germany), with a Leica HCX APO 20x/0.5 water immersion lens with zoom 4 and pinhole 6 Airy units, giving a field of view of 193.75 μm with pixel size 0.76 μm. The laser used 10% power at 488 nm, and 16-bit 256×256 images were acquired in the 500-650 nm range using an 1% acousto-optic tunable filter and photomultiplier tube with 436 V gain. The bleach region radius $r = 15\,\mu\text{m}$, and the scanning rate was 1000 Hz, with a sampling time lag of $\Delta t = 0.265\,\text{s}$. Like in the network training, $T_{\text{prebleach}} = 10$ prebleach frames, $T_{\text{bleach}} = 4$ bleach frames, and $T_{\text{postbleach}} = 100$ postbleach frames were used.

To eliminate shadows and gradients in the image, we apply a Gaussian filter ($\sigma = 5$) to the mean of the pre-bleach frames, and subtract it from the pre- and post-bleach frames.

We test the least squares fit on both the recovery curve and pixel data, test the neural

network for the recovery curve, and the downsampler network on the pixel data. The results for 32 wt% and 56 wt% given by Table 4.5-4.6, given as mean and standard deviation of the estimates. The LS-PX and LS-RC methods yield $1.001 \pm 0.005 \times 10^{-10}\,\mathrm{m}^2/\mathrm{s}$ and $0.884 \pm 0.002 \times 10^{-10}\,\mathrm{m}^2/\mathrm{s}$ for the 32 wt% solution, wheareas we obtain $1.045 \pm 0.005 \times 10^{-10}\,\mathrm{m}^2/\mathrm{s}$ and $1.264 \pm 4.64 \times 10^{-10}\,\mathrm{m}^2/\mathrm{s}$ from NN-PX and NN-RC. The diffusion coefficients and the remaining parameters are all close to the expected values, with similar standard deviations.

Table 4.5: *Mean and standard deviation for* 32 wt% *experimental samples, for the least square fits to recovery curve (LS-RC) and pixel data (LS-PX), as well as recovery curve network (NN-RC) and downsampler CNN (NN-PX).*

method	$\hat{D}$, $(\mathrm{m}^2/\mathrm{s})$	std($\hat{D}$), $(\mathrm{m}^2/\mathrm{s})$	$\hat{c}_0$	std($\hat{c}_0$)	$\hat{\alpha}$	std($\hat{\alpha}$)
LS-PX	1.001×10^{-10}	5.14×10^{-12}	0.799	0.0183	0.747	0.0091
LS-RC	0.884×10^{-10}	2.12×10^{-12}	0.798	0.0189	0.779	0.0060
NN-PX	1.045×10^{-10}	5.19×10^{-12}	0.787	0.0143	0.785	0.0097
NN-RC	1.264×10^{-10}	4.64×10^{-12}	0.800	0.0167	0.775	0.0079

Table 4.6: *Mean and standard deviation for the* 56 wt% *experimental samples, for the least square fits to recovery curve (LS-RC) and pixel data (LS-PX), as well as recovery curve network (NN-RC) and downsampler CNN (NN-PX).*

method	$\hat{D}$, $(\mathrm{m}^2/\mathrm{s})$	std($\hat{D}$), $(\mathrm{m}^2/\mathrm{s})$	$\hat{c}_0$	std($\hat{c}_0$)	$\hat{\alpha}$	std($\hat{\alpha}$)
LS-PX	1.105×10^{-11}	3.26×10^{-13}	0.817	0.0100	0.699	0.0064
LS-RC	0.936×10^{-11}	3.50×10^{-13}	0.816	0.0106	0.736	0.0021
NN-PX	1.126×10^{-11}	2.27×10^{-13}	0.794	0.0086	0.721	0.0052
NN-RC	1.287×10^{-11}	4.71×10^{-13}	0.818	0.0107	0.732	0.0015

For the 56 wt% solution, we acquire similar results, but with slightly more variation between mean estimates. The least squares methods give $1.105 \pm 0.03 \times 10^{-11}\,\mathrm{m}^2/\mathrm{s}$ and $0.936 \pm 0.04 \times 10^{-11}\,\mathrm{m}^2/\mathrm{s}$ for the pixel-based and recovery curved-based methods respectively. The neural networks yield $1.126 \pm 0.002 \times 10^{-11}\,\mathrm{m}^2/\mathrm{s}$ and $1.287 \pm 0.004 \times 10^{-11}\,\mathrm{m}^2/\mathrm{s}$ for pixel data and recovery curves respectively. Since all estimates are close to the reference parameters, it would seem the neural perform similar to the LS methods. Using LS-PX as the gold standard, we can observe in Figure 4.10 a pattern of underestimating and overestimating the diffusion coefficient for LS-RC and the neural networks respectively.

However, it is known from Section 4.2 that the downsampler makes worse estimates of the bleach parameter α, and that the recovery curve network struggles with high MSE in general. Figure 4.11 shows the residuals from the first post-bleach frame, $T = 10$, from simulating the spatio-temporal data with the estimated parameters of an experimental sample with 32 wt% for all methods but NN-RC. Both LS methods have very evenly distributed, small, residuals. The downsampler residuals in the ROI indicate that the bleach depth prediction is not entirely correct. The experimental recovery curve in Figure 4.12

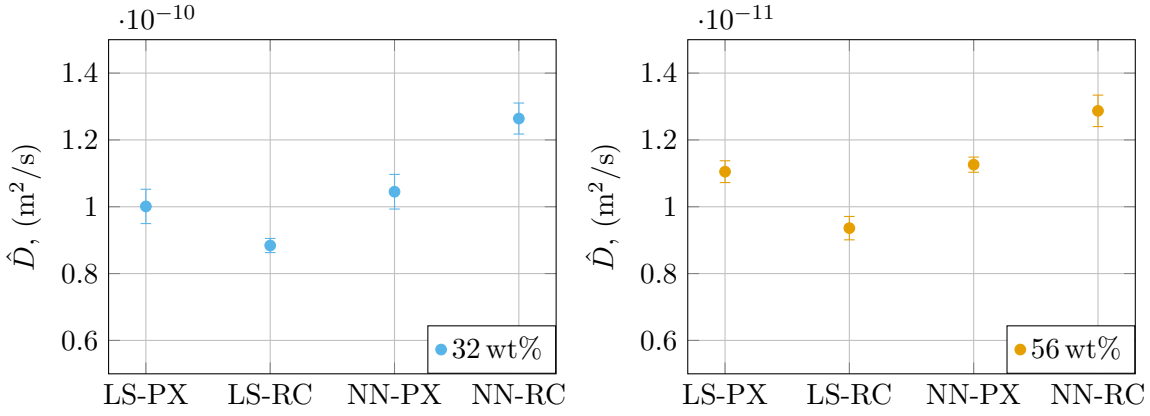
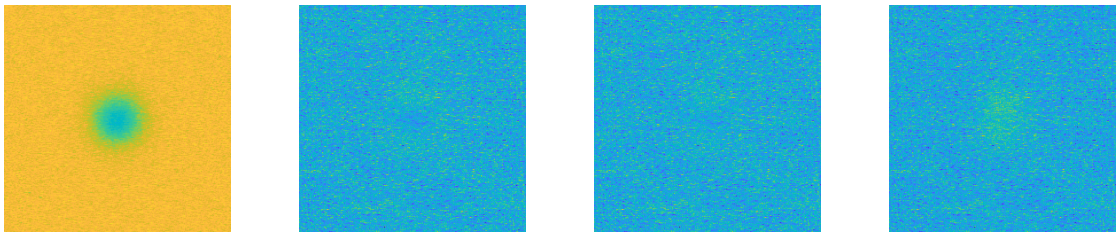


Figure 4.10: *Mean and standard deviations of the diffusion rate estimates of fluorescein on experimental data with* 32 wt% *and* 56 wt% *sucrose, indicating good agreement across methods.*

again indicates the the downsampler underestimates bleach depth like in Figure 4.8, and has a slightly different recovery compared to the LS methods. Since the recovery curve does not contain the full information of D, it is hard to say which estimation is the best. Considering Section 4.2.3, we do know that the downsampler tends to overestimate D compared to pixel-based least squares.

Finally, a possibility of the simplicity and speed of the neural networks is to use its estimate as the initial guess for least-squares. Using the downsampler parameter estimates for the 20 replicate 32 wt% experiment as the guess for `fmincon` in MATLAB, we get $\hat{D} = 1.0029 \pm 5.13 \times 10^{-12}\,\mathrm{m}^2/\mathrm{s}$, $\hat{c}_0 = 0.7986 \pm 0.0183$, $\hat{\alpha} = 0.7467 \pm 0.0091$, which are all very close or identical to the first estimates in Table 4.5. This means the downsampler could potentially be used to simplify the workflow of the least squares estimations.



first post-bleach, $T = 11$ **LS-PX** **LS-RC** **NN-PX**

Figure 4.11: *First post-bleach frame ($T = 10$) from a sample experiment (first from the left), and the residuals from predictions with pixel-based least squares (second), recovery curve-based least squares (third), and the downsampler neural network (fourth). Since the downsampler makes slightly worse estimations of the bleach depth, this is also visible in the centre residuals.*

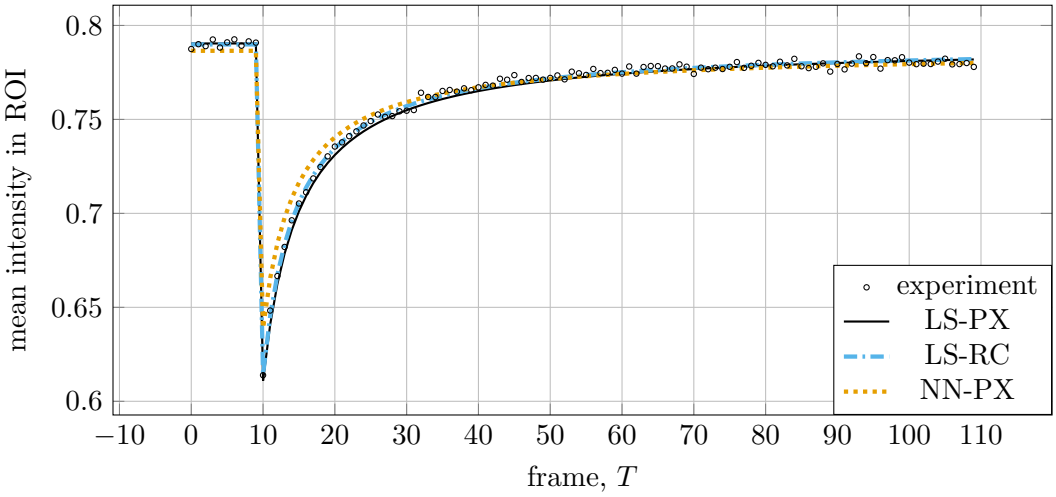


Figure 4.12: *Recovery curve fit for the least squares methods and the downsampler neural network, indicating overall good fits, but an especially low estimate for the bleach parameter α from the downsampler.*

5 Conclusion

The purpose of this thesis was to investigate if deep learning as a method for estimating the system parameters in FRAP was possible and how it compares to the conventional least squares methods. A set of neural network architectures were considered, namely two spatio-temporal models; a convolutional LSTM and downsampling 2D CNN, and a fully connected model for the recovery curve. Numerically simulated FRAP data has served as the training, validation and test data for the networks. However, the spatio-temporal models struggle with the computational weight of the full image data, which becomes a clear bottleneck for training, both when it comes to storage and working memory. A number of training schemes were devised to combat this problem, each with their own limitations - training neural networks online on continuously generated data can cause catastrophic forgetting, a case of overwriting the network weights with each batch. This shows that having access to in principle infinite data does not necessarily make training easy. By generating a moderately large dataset which is updated by a fraction of new samples each epoch, we counteract some of the limitations concerning data storage and catastrophic forgetting.

After comparing the neural networks to the conventional methods on simulated data we conclude that the downsampler performs best in terms of loss compared to the other neural networks, but slightly worse compared to least squares, especially on the bleach parameter. The training results show that the downsampler test results may be improved by using more training data and more epochs. The neural networks do have a few advantages: They do not require an initial guess for the parameters and are robust to noise and different parameter magnitudes. The pixel-based least squares method produces estimates with very low error, and the temporal recovery curve-based least squares method is easy to implement and interpret. However, they both require initial guesses for the parameters in order to minimize the error, a processes which is potentially time consuming, especially if little is known about the system beforehand. We show that applying the downsampler parameter estimate as the initial guess for least squares speeds up calculation and does not require a hypothesis on the parameter values. This is a potential future application of our findings.

The biggest disadvantages of the the neural network models is its limitation to the microscope parameters on which it is trained, for example the size and shape of the ROI, number of frames and sampling time lag. Some of the drawbacks of the downsampler could be remedied by the convolutional LSTM, namely extracting both temporal and spatial information for estimating the bleach parameter, but also potential independence of the number of frames. However, such work needs considerable computational resources or to otherwise find a solution to the size of the data. The ConvLSTM has shown promise on similar problems in weather forecasting, but the number and size of the frames force a compromise on the batch size and the number of filters during training. This is computationally very heavy and puts a constraint on the loss convergence.

Future work on the topics of this thesis should aim to generalize the neural networks for different microscope settings. Initial tests indicate that the neural network is able to learn small variations in ROI size and position in the training data. In a similar fashion, the network can be taught to be invariant to shadows and occlusions, which may occur in

real world applications, by means of data augmentation. The sampling time lag defines the fundamental time scale of the diffusion coefficient, and significantly affects the bleach parameter estimate, since it is determined by the ratio of fluorescence in the ROI before and after bleaching. However, a diffusion coefficient prediction from data with time lag Δt by a neural network trained on a time lag $\Delta t'$ could theoretically be rescaled as $D \leftarrow D\Delta t/\Delta t'$. Unfortunately, such a linear scaling would not be possible for α. This also translates to a possible strategy for handling a variable number of frames; if the input sequences have more frames than the network is trained to handle, the pre- and post-bleach frames may be linearly sampled and the diffusion coefficient rescaled.

The downsampler and neural networks in general are designed to learn a method of feature extractions. However, investigating the alternate feature extractions from the images could circumvent the need for the pixel-based estimators, if they preserve enough information. Possible approaches include calculating the mean intensity in rings or sections with increasing size, or supplying higher order statistical moments of the image ROI than the mean and variance. Albeit, these methods are most appropriately benchmarked in a least squares and maximum likelihood and not by means of neural networks.

Lastly, this work is easily extendible to more FRAP parameters, such as the image bleaching or more interestingly the on and off binding rates in diffusion and binding.

References

[1] N. Lorén et al. "Fluorescence recovery after photobleaching in material and life sciences: putting theory into practice". In: *Quarterly Reviews of Biophysics* 48.3 (2015), pp. 323–387. DOI: 10.1017/S0033583515000013.

[2] H. Deschout et al. "FRAP in Pharmaceutical Research: Practical Guidelines and Applications in Drug Delivery". In: *Pharmaceutical Research* 31.2 (Feb. 2014), pp. 255–270. ISSN: 1573-904X. DOI: 10.1007/s11095-013-1146-9.

[3] S. C. De Smedt et al. "Studying biophysical barriers to DNA delivery by advanced light microscopy". In: *Advanced Drug Delivery Reviews* 57.1 (2005). Advances in Fluorescence Imaging: Opportunities for Pharmaceutical Science, pp. 191–210. ISSN: 0169-409X. DOI: https://doi.org/10.1016/j.addr.2004.06.003.

[4] L. Svanberg et al. "Effect of Pre-Crystallization Process and Solid Particle Addition on Cocoa Butter Crystallization and Resulting Microstructure in Chocolate Model Systems". In: *Procedia Food Science* 1 (2011). 11th International Congress on Engineering and Food (ICEF11), pp. 1910–1917. ISSN: 2211-601X. DOI: https://doi.org/10.1016/j.profoo.2011.09.281.

[5] J. K. Jonasson et al. "A pixel-based likelihood framework for analysis of fluorescence recovery after photobleaching data". In: *Journal of Microscopy* 232.2 (2008), pp. 260–269. DOI: 10.1111/j.1365-2818.2008.02097.x.

[6] J. K. Jonasson et al. "Pixel-based analysis of FRAP data with a general initial bleaching profile". In: *Journal of Microscopy* 239.2 (2010), pp. 142–153. DOI: 10.1111/j.1365-2818.2009.03361.x.

[7] M. Röding et al. "A Highly Accurate Pixel-Based FRAP Model Based on Spectral-Domain Numerical Methods". In: *Biophysical Journal* 116.7 (2019), pp. 1348–1361. ISSN: 0006-3495. DOI: https://doi.org/10.1016/j.bpj.2019.02.023.

[8] R. Brown. "XXVII. A brief account of microscopical observations made in the months of June, July and August 1827, on the particles contained in the pollen of plants; and on the general existence of active molecules in organic and inorganic bodies". In: *The Philosophical Magazine* 4.21 (1828), pp. 161–173. DOI: 10.1080/14786442808674769.

[9] A. Fick. "Über Diffusion". In: *Annalen der Physik* 170.1 (1855), pp. 59–86. DOI: 10.1002/andp.18551700105.

[10] Kofstad J. and T. Norby. *Lecture notes on chemical diffusion in the Defect Chemistry and Reactions course*. Feb. 2009.

[11] K. A. Dill and S. Bromberg. "Molecular Driving Forces: Statistical Thermodynamics in Chemistry and Biology". In: Garland Science, 2003. Chap. 18, p. 327.

[12] A. Einstein. "Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen". In: *Annalen der Physik* 322.8 (Jan. 1905), pp. 549–560. DOI: 10.1002/andp.19053220806.

[13] G. Wahnström. *Lecture notes on Brownian dynamics in the Computational Physics course*. Dec. 2016.

[14] P. Davidovits. "Scanning Laser Microscope". In: 223.5208 (Aug. 1969), p. 831. DOI: 10.1038/223831a0.

[15] P. Davidovits and M. D. Egger. “Scanning laser microscope for biological investigations”. In: 10.7 (July 1971), pp. 1615–1619. DOI: 10.1364/AO.10.001615.

[16] H. Deschout et al. “Straightforward FRAP for quantitative diffusion measurements with a laser scanning microscope”. In: *Optics express* 18.22 (2010), pp. 22886–22905.

[17] H. Kapitza, G. Mcgregor, and K. A. Jacobson. “Direct measurement of lateral transport in membranes by using time-resolved spatial photometry.” In: *Proceedings of the National Academy of Sciences of the United States of America* 82 12 (1985), pp. 4122–6.

[18] W. S. McCulloch and W. Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259.

[19] F. Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain”. In: *Psychological Review* (1958), pp. 65–386.

[20] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[21] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[22] B. Mehlig. *Artificial Neural Networks*. Lecture notes in Artificial Neural Networks. 2019. arXiv: 1901.05639.

[23] Y. E. Nesterov. “A method for solving the convex programming problem with convergence rate $O(1/k^2)$”. In: *Dokl. Akad. Nauk SSSR* 269 (1983), pp. 543–547.

[24] A. Paszke et al. “Automatic Differentiation in PyTorch”. In: *NIPS Autodiff Workshop*. 2017.

[25] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[26] Y. LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[27] S. Ioffe and C. Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[28] G. E. Hinton et al. *Improving neural networks by preventing co-adaptation of feature detectors*. 2012. arXiv: 1207.0580 [cs.NE].

[29] Bengio Y., Simard P., and Frasconi P. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (Mar. 1994), pp. 157–166. DOI: 10.1109/72.279181.

[30] S. Hochreiter and J. Schmidhuber. “Long Short-term Memory”. In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[31] Xingjian Shi et al. “Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting”. In: *CoRR* abs/1506.04214 (2015). URL: http://arxiv.org/abs/1506.04214.

[32] B. Zapata-Impata, P. Gil, and F. Torres. “Learning Spatio Temporal Tactile Features with a ConvLSTM for the Direction Of Slip Detection”. In: *Sensors* 19.3 (Jan. 2019), p. 523. ISSN: 1424-8220. DOI: 10.3390/s19030523.

[33] R. A. Robinson and R. H. Stokes. *Electrolyte solutions*. Courier Corporation, 2002, pp. 13–14.

[34] F-X. Theillet et al. “Physicochemical properties of cells and their effects on intrinsically disordered proteins (IDPs)”. In: *Chemical reviews* 114.13 (2014), pp. 6661–6714.

[35] K. He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[36] K. Hara, H. Kataoka, and Y. Satoh. “Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet?” In: *CoRR* abs/1711.09577 (2017). arXiv: 1711.09577. URL: http://arxiv.org/abs/1711.09577.

[37] D. Tran et al. “C3D: Generic Features for Video Analysis”. In: *CoRR* abs/1412.0767 (2014). arXiv: 1412.0767. URL: http://arxiv.org/abs/1412.0767.

[38] S. Xie et al. “Rethinking Spatiotemporal Feature Learning For Video Understanding”. In: *CoRR* abs/1712.04851 (2017). arXiv: 1712.04851. URL: http://arxiv.org/abs/1712.04851.

[39] A. Karpathy et al. “Large-Scale Video Classification with Convolutional Neural Networks”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. June 2014, pp. 1725–1732. DOI: 10.1109/CVPR.2014.223.

[40] J. Donahue et al. “Long-term recurrent convolutional networks for visual recognition and description”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 2625–2634.

[41] J. Carreira and A. Zisserman. “Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset”. In: *CoRR* abs/1705.07750 (2017). arXiv: 1705.07750. URL: http://arxiv.org/abs/1705.07750.

[42] J. Tremblay et al. “Training deep networks with synthetic data: Bridging the reality gap by domain randomization”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 969–977.

[43] T. A. Le et al. “Using synthetic data to train neural networks is model-based reasoning”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 3514–3521.

[44] M. R. French. “Catastrophic forgetting in connectionist networks”. In: *Trends in Cognitive Sciences* 3.4 (1999), pp. 128–135. ISSN: 1364-6613. DOI: https://doi.org/10.1016/S1364-6613(99)01294-2.

[45] G. Perale et al. “Drug release from hydrogel: a new understanding of transport phenomena”. In: *Journal of biomedical nanotechnology* 7.3 (2011), pp. 476–481.

[46] C. Soeller et al. “Application of two-photon flash photolysis to reveal intercellular communication and intracellular Ca2+ movements”. In: *Journal of biomedical optics* (2003).