Deep Neural Networks and Image Analysis for Quantitative Microscopy

SAJITH KECHERIL SADANANDAN
Understanding biology paves the way for discovering drugs targeting deadly diseases like cancer, and microscopy imaging is one of the most informative ways to study biology. However, analysis of large numbers of samples is often required to draw statistically verifiable conclusions. Automated approaches for analysis of microscopy image data makes it possible to handle large data sets, and at the same time reduce the risk of bias. Quantitative microscopy refers to computational methods for extracting measurements from microscopy images, enabling detection and comparison of subtle changes in morphology or behavior induced by varying experimental conditions. This thesis covers computational methods for segmentation and classification of biological samples imaged by microscopy.

Recent increase in computational power has enabled the development of deep neural networks (DNNs) that perform well in solving real world problems. This thesis compares classical image analysis algorithms for segmentation of bacteria cells and introduces a novel method that combines classical image analysis and DNNs for improved cell segmentation and detection of rare phenotypes. This thesis also demonstrates a novel DNN for segmentation of clusters of cells (spheroid), with varying sizes, shapes and textures imaged by phase contrast microscopy. DNNs typically require large amounts of training data. This problem is addressed by proposing an automated approach for creating ground truths by utilizing multiple imaging modalities and classical image analysis. The resulting DNNs are applied to segment unstained cells from bright field microscopy images. In DNNs, it is often difficult to understand what image features have the largest influence on the final classification results. This is addressed in an experiment where DNNs are applied to classify zebrafish embryos based on phenotypic changes induced by drug treatment. The response of the trained DNN is tested by ablation studies, which revealed that the networks do not necessarily learn the features most obvious at visual examination. Finally, DNNs are explored for classification of cervical and oral cell samples collected for cancer screening. Initial results show that the DNNs can respond to very subtle malignancy associated changes. All the presented methods are developed using open-source tools and validated on real microscopy images.

Keywords: Deep neural networks, convolutional neural networks, image analysis, quantitative microscopy, bright-field microscopy

Sajith Kecheril Sadanandan, Department of Information Technology, Division of Visual Information and Interaction, Box 337, Uppsala University, SE-751 05 Uppsala, Sweden.
Department of Information Technology, Computerized Image Analysis and Human-Computer Interaction, Box 337, Uppsala University, SE-75105 Uppsala, Sweden.

© Sajith Kecheril Sadanandan 2017

ISSN 1651-6214
ISBN 978-91-513-0080-1
urn:nbn:se:uu:diva-329834 (http://urn.kb.se/resolve?urn=nbn:se:uu:diva-329834)
To my family
List of papers

This thesis is based on the following papers, which are referred to in the text by their Roman numerals.


*These authors contributed equally to this work.
Reprints were made with permission from the publishers.
For Paper I, Sadanandan, S. K. is the main contributor to the design and implementation of image analysis and track analysis part of the paper. Sadanandan, S. K. wrote the paper with inputs from the co-authors.

For Paper II, Ranefall, P. is the main contributor to the image analysis method. Sadanandan, S. K. did the quantitative comparison of the proposed method with other methods discussed in the paper. Ranefall, P. wrote the paper with inputs from the co-authors.

For Paper III, Sadanandan, S. K. is the main contributor to the design and implementation of image analysis, and deep neural network design, training and testing. Sadanandan, S. K. wrote the paper with inputs from the co-authors.

For Paper IV, Sadanandan, S. K. is the main contributor to the design and implementation of image analysis, and deep neural network design, training and testing. Sadanandan, S. K. wrote the paper with inputs from the co-authors.

For Paper V, Sadanandan, S. K. is the main contributor to the design and implementation of image analysis, and deep neural network design, training and testing. Sadanandan, S. K. and Ranefall, P. equally contributed to the development of the CellProfiler - Caffe bridge to use a trained deep neural network in CellProfiler. Sadanandan, S. K. wrote the paper with inputs from the co-authors.

For Paper VI, Ishaq, O. and Sadanandan, S. K. equally contributed to the design and development of the methodology. Ishaq, O. and Sadanandan, S. K. wrote the paper with inputs from the co-authors.

For Paper VII, Wieslander, H. and Forslid, G. are the main contributors and Sadanandan, S. K. is the main supervisor for the Master thesis project, which resulted in this paper. Sadanandan, S. K. designed the experimental procedures for training the deep neural networks. Wieslander, H. and Forslid, G. implemented, trained and tested the deep neural networks. Wieslander, H. and Forslid, G. wrote the paper with inputs from the co-authors.
Related work

In addition to the papers included in this thesis, the author has also written or contributed to the following publications:


Contents

1 Introduction .................................................................................................................. 15
  1.1 Objectives .................................................................................................................. 15
  1.2 Digital images .......................................................................................................... 16
  1.3 Computerized image analysis .................................................................................. 16
  1.4 Deep neural networks for image analysis ................................................................. 16
  1.5 Quantitative microscopy ......................................................................................... 17
  1.6 Thesis outline .......................................................................................................... 17

2 Microscopy Image Analysis ........................................................................................... 19
  2.1 Microscopy .............................................................................................................. 19
    2.1.1 Bright-field microscopy ..................................................................................... 19
    2.1.2 Phase contrast microscopy ............................................................................... 20
    2.1.3 Fluorescence microscopy .................................................................................. 20
    2.1.4 Datasets ........................................................................................................... 20
    2.1.5 Digital microscope ............................................................................................ 23
  2.2 Image analysis techniques used in this thesis ......................................................... 23
    2.2.1 Curvature based contrast enhancement .............................................................. 24
    2.2.2 Full width at half maximum ............................................................................. 24
    2.2.3 Singular value decomposition for de-noising ..................................................... 25
    2.2.4 Wavelet reconstruction for boundary enhancement ......................................... 25
    2.2.5 Average shape using distance transform ......................................................... 26
    2.2.6 Morphological geodesic active contours ......................................................... 27
    2.2.7 Segmentation evaluation .................................................................................... 27
    2.2.8 Percentage of segments v/s F-score plot ......................................................... 29

3 Deep Neural Networks .................................................................................................. 31
  3.1 Background .............................................................................................................. 31
    3.1.1 Deep neural networks ....................................................................................... 32
    3.1.2 Deep neural network training .......................................................................... 33
  3.2 Convolutional neural networks ................................................................................. 35
    3.2.1 Convolutional layers ......................................................................................... 36
    3.2.2 Dilated convolution ......................................................................................... 37
    3.2.3 Deconvolution .................................................................................................. 37
    3.2.4 Fully connected layers ..................................................................................... 38
    3.2.5 Pooling layers .................................................................................................. 38
    3.2.6 Activation function layers ................................................................................. 40
    3.2.7 Fully convolutional neural networks ................................................................. 42
3.2.8 Residual connections ........................................ 43
3.3 Loss functions ..................................................... 43
  3.3.1 Cross-entropy loss ........................................... 44
  3.3.2 Maximum likelihood estimation ............................ 45
  3.3.3 Kullback-Leibler divergence ............................... 45
  3.3.4 Softmax activation ......................................... 46
  3.3.5 Numerically stable softmax ............................... 47
  3.3.6 Spatial cross-entropy loss ................................. 47
  3.3.7 Weighted spatial cross-entropy loss ..................... 47
3.4 Regularization of deep networks ............................... 48
  3.4.1 Parameter norms ........................................... 48
  3.4.2 Dataset augmentation ...................................... 49
  3.4.3 Early stopping ............................................. 49
  3.4.4 Drop out .................................................. 50
  3.4.5 Batch normalization ...................................... 50
3.5 Optimization of deep neural networks ......................... 51
  3.5.1 Gradient descent ........................................... 51
  3.5.2 Batch gradient descent ................................... 51
  3.5.3 Stochastic gradient descent ............................... 52
  3.5.4 Mini-batch gradient descent ............................... 52
  3.5.5 Adaptive methods .......................................... 52
  3.5.6 Generative adversarial networks ......................... 54
3.6 Reflection of the deep neural network algorithms ............. 56

4 Overview of papers and summary of contributions ............... 57
  4.1 Segmentation and Track-Analysis in Time-Lapse Imaging of
   Bacteria (Paper I) .............................................. 57
    4.1.1 Background ............................................. 57
    4.1.2 Curvature based contrast enhancement .................. 57
    4.1.3 Object segmentation .................................... 58
    4.1.4 Object filtering ........................................ 58
    4.1.5 Object smoothing ....................................... 59
    4.1.6 Cell tracking ........................................... 59
    4.1.7 Cell track filtering .................................... 59
    4.1.8 Post-tracking segmentation correction .................. 59
    4.1.9 Run time evaluation .................................... 60
    4.1.10 Discussion of results and contributions ................ 60
  4.2 Fast Adaptive Local Thresholding based on Ellipse Fit for
   Bacteria Segmentation (Paper II) ............................. 61
    4.2.1 Background ............................................. 61
    4.2.2 Component tree based segmentation of *E. Coli*
      bacteria .................................................. 61
    4.2.3 Run time evaluation .................................... 61
    4.2.4 Discussion of results and contributions ................. 61
4.3 Feature Augmented Deep Neural Networks for Segmentation of Cells (Paper III) .................................................. 62
  4.3.1 Background .................................................. 62
  4.3.2 Deep neural network architecture ......................... 62
  4.3.3 Feature augmentation ..................................... 63
  4.3.4 Segmentation and post-processing ......................... 63
  4.3.5 Run time evaluation ...................................... 63
  4.3.6 Discussion of results and contributions .................. 63
4.4 Spheroid Segmentation using Multiscale Deep Adversarial Networks (Paper IV) .................................................. 64
  4.4.1 Background .................................................. 64
  4.4.2 Segmentation network architecture ....................... 64
  4.4.3 Adversarial network ...................................... 65
  4.4.4 Execution time ............................................. 66
  4.4.5 Discussion of results and contributions .................. 66
4.5 Automated Training of Deep Convolutional Neural Networks for Cell Segmentation (Paper V) ........................................ 67
  4.5.1 Background .................................................. 67
  4.5.2 Automatic training set generation ......................... 67
  4.5.3 Segmentation of cells ..................................... 68
  4.5.4 Discussion of results and contributions .................. 69
4.6 Deep Fish: Deep Learning-Based Classification of Zebrafish Deformation for High-Throughput Screening (Paper VI) .......... 69
  4.6.1 Background .................................................. 69
  4.6.2 Dataset ...................................................... 70
  4.6.3 Deep neural network architecture and training .......... 70
  4.6.4 Ablation studies .......................................... 70
  4.6.5 Discussion of results and contributions .................. 70
4.7 Deep Convolutional Neural Networks for Detecting Cellular Changes Due to Malignancy (Paper VII) .................................. 71
  4.7.1 Background .................................................. 71
  4.7.2 Datasets .................................................... 72
  4.7.3 Deep convolutional neural networks ....................... 72
  4.7.4 Discussion of results and contributions .................. 72

5 Conclusion and future perspectives .................................. 73
  5.1 Conclusion ..................................................... 73
  5.2 Future perspectives .......................................... 75

Summary in Swedish ...................................................... 77

Acknowledgments .......................................................... 79

References ............................................................... 81

Errata ........................................................................ 85
1. Introduction

A better understanding of biology helps in improving the living standard of human beings. To study the behavior of complex diseases like cancer, microscopy images can provide valuable clues, but it is often necessary to experiment with a large number of drugs on several samples. Traditionally, experiments are evaluated visually requiring many years of human effort. Today, with the advancement of new types of microscopes and imaging systems it is possible to observe events that were previously unknown to us. High-throughput and high-content screens are used to analyze events on large scales so that statistically testable conclusions can be derived. Such screens generate large volumes of data that are infeasible for human observers to analyze quantitatively. Therefore, automated methods are necessary for analyzing these large datasets.

1.1 Objectives

The main objective of this thesis has been to develop methods to analyze digital images captured using microscopes. This work consists of development for two purposes: segmentation of cells and classification of phenotypes. Two kinds of methods have been used: traditional image analysis and deep learning neural network based methods. The specific objectives and methods as presented in the appended papers can be summarized as follows:

I To develop methods to segment *E. coli* cell colonies using traditional image analysis.

II To develop a faster method to segment *E. coli* cells using traditional image analysis.

III To improve the segmentation of *E. coli* cells using a combination of traditional and deep neural network-based image analysis.

IV To develop methods to segment spheroid cell cultures using deep neural networks.

V To develop and prototype methods for automatic training of deep neural networks to segment cells in bright-field microscopy images.

VI To develop methods to classify zebrafish phenotypes in bright-field microscopy images using deep neural networks.

VII To develop methods to classify malignancy associated changes in cervical and oral cells from bright-field microscopy images using deep neural networks.
1.2 Digital images

With the advent of digital cameras we are able to acquire digital images. The digital cameras consist of image sensors that convert the real world scene to a digital version that can be viewed on a computer screen. The digital version of a scene consists of very small entities known as picture elements or pixels. The size and number of pixels in an image depends on the sensor used to capture the scene. Typically, a mobile phone camera today generates images with 18 mega pixels or more.

Digital images are used almost everywhere in the modern society from self-driving cars to screening of diseases. A camera mounted on a car digitizes the scenes in front of it for further processing. If we place a microscope in front of a camera then we can capture a magnified view of the scene under the microscope. The camera mounted on a microscope is suitable to generate images of microscopic objects such as cells or embryos of small organisms like zebrafish. All images used in this thesis were obtained using cameras mounted on microscopes.

1.3 Computerized image analysis

Once we have acquired digital images using cameras, the next challenge is to analyze them. Analyzing huge amounts of images is tedious and requires experts who can interpret the domain specific data. Computerized analysis means using programs or algorithms on a digital computer to analyze the data. Computerized image analysis is necessary to have an unbiased and quantitative analysis as the computers never get tired and we can repeat the program any number of times and get the same result.

1.4 Deep neural networks for image analysis

Deep neural networks or artificial neural networks can be used to analyze image data. Neural networks is an approach to image analysis that has been discussed for several decades, but it is only the last few years that the increased computational power and memory capacity of computers have made it possible to develop networks that really work well in solving real world problems.

When we use deep neural networks for image analysis we usually train the networks on a set of annotated images. It is typical to treat image analysis and deep neural networks as two separate domains for solving a problem. However, in this thesis we treat both as general tools for solving our problems and we often combine the approaches to obtain the desired results. The selection of a specific approach or a combination depends on the corresponding image data. Deep neural networks are discussed in more detail in Chapter 3.
1.5 Quantitative microscopy

Quantitative microscopy means using computers to analyze digital images acquired using microscopes. The results of quantitative analysis can for example be segmentation of microscopic objects such as cells, measuring the growth or migration of cells over time, or detection and localization of intracellular organelles and finally phenotype classification. All these various outcomes can be expressed as numbers so that an unbiased comparison among experiments is possible.

In this thesis, I used various image analysis approaches and deep neural networks to obtain quantitative results. An overview of the type of microscopy data and the associated paper number is shown in Fig. 1.1. Papers I and II present two image analysis methods to segment *Escherichia coli* (*E. coli*) bacteria cells. Paper I also contains a method for tracking of the cells over time to measure the impact of growth media on the cell growth. Paper III describes segmentation of *E. coli* cells and mouse mammary gland cells using a combination of traditional image analysis methods and deep neural networks. Paper IV explains a novel deep neural network to segment spheroid cell cultures. For microscopy image analysis using deep neural networks, large amounts of manually curated images are required. This problem is addressed in Paper V by creating automated annotations using image analysis methods and then training a deep neural network to segment cells from the images.

Papers I, II, III, IV, and V discuss different types of methods that are used to segment microscopy objects, while Papers VI and VII discuss classification of phenotypes. Paper VI explains the usage of deep neural networks to classify zebrafish embryos as untreated or treated by a particular drug for high throughput applications. Cancer is one of the major causes of death worldwide. This is particularly predominant in developing countries due to the lack of proper screening programs and availability of experts. The screening programs are helpful in detecting and identifying the type of cancer so that mortality rates can be reduced. In Paper VII, we discuss the possibility of finding precancerous lesions known as malignancy associated changes from images acquired from oral and cervical cell samples.

1.6 Thesis outline

In this thesis, I discuss various image analysis methods for analyzing and quantifying microscopy image data. The image data consist of images of different types of cell lines and model organisms such as zebrafish. Chapter 2 gives a brief introduction to different types of microscopy used to capture the image data and various image analysis methods used in the appended papers. Chapter 3 describes and discusses the deep neural networks used for analyzing the image data in this thesis. Chapter 4 summarizes the results and contributions of the papers that form the basis for this thesis. Each section is focused
Figure 1.1. Thesis overview. Each image represents datasets used in one or more papers contained in this thesis. The arrows symbolically show either the samples are from human patients or from other organisms.

on one of the papers. Chapter 5 concludes the thesis and discusses future perspectives.
2. Microscopy Image Analysis

In this chapter, I give a brief introduction to the types of microscopes used to acquire our datasets and different image analysis techniques applied. I used both traditional image analysis methods and deep learning based methods for analyzing our image data. The traditional image analysis employs already known filters or algorithms, while the deep learning based approach is performed by learning the filters from a dataset. In this work, either I use traditional methods alone or I combine both approaches to solve a specific image analysis problem. A detailed description of image analysis techniques using deep neural networks is provided in Chapter 3.

2.1 Microscopy

A microscope is an instrument used to magnify very small objects so that they become visible to human eyes. The microscope consists of a light source and a complex system of lenses to magnify a sample. There are different types of microscopes depending on the magnification required and the type of samples analyzed. In this thesis, all image data were acquired using microscopes.

2.1.1 Bright-field microscopy

The cell samples collected for a particular experiment is typically transparent and it is difficult to see the intracellular structures. Therefore, the samples are usually fixed and then stained with specialized stains such as Pap stain or hematoxylin and eosin stain (H&E), depending on the type of sample and the intracellular region to be observed. The stain attaches to a particular region of interest in a cell such as nucleus or cytoplasm. Thereafter when we illuminate the samples, the stain absorbs certain wavelengths of light. The wavelengths of light passing through a region, not absorbed, determine the color of that region. The transmitted light is either observed through the eyepiece or imaged using a color or monochrome camera.

Bright-field microscopy is not suitable for imaging live cells, as the cells need to be fixed before imaging. We used bright-field microscopy images of cervical and oral cells, as shown in Fig. 2.1 (f) and (g), in Paper VII. Bright-field images of zebrafish, as shown in Fig. 2.1 (i), were used in Paper VI.
2.1.2 Phase contrast microscopy

In phase contrast microscopy, we can image samples without fixing and staining. Phase contrast microscopy is popularly used to image live samples, since it is one of the least invasive ways of imaging. It is used to acquire images of live cells or organisms. In phase contrast microscopy, contrast in a sample is generated by utilizing the shift in phase of light when passed through the sample. For imaging, the specimen is illuminated with a light source. When the light passes through the specimen, the diffraction of light causes a phase shift. This difference in phase is transferred to a difference in the amplitude of the light waves by the microscope, showing the specimen as if it was stained. Sample datasets of phase contrast images are shown in Fig 2.1 (a), (c), (d) and (h). We used phase contrast datasets in Papers I, II, III, IV, and V.

2.1.3 Fluorescence microscopy

In fluorescence microscopy, the samples are stained with a specific fluorescent dye. Then the samples are illuminated with an excitatory light, usually of shorter wavelength. After excitation, the fluorescent dye emits light of longer wavelength, resulting in fluorescence [1], which is measured to produce the image. The fluorescence microscopy is used to image specific intracellular organelles or macromolecules. This enables the study of a particular macromolecule of interest. Fluorescent stains such as DAPI and Hoechst are used to stain cell nuclei. On excitation, all the fluorescent molecules fluoresce resulting in low contrast to the region of interest. Therefore, a type of fluorescence microscopy known as confocal microscopy is typically used to get images with better contrast between foreground and background. In confocal microscopy a focused narrow beam of LASER is used to excite the sample. The beam then moves over the sample to produce the overall image of the sample. Examples of fluorescence microscopy images are shown in Fig. 2.1 (b) and (e). We used fluorescence microscopy images in Papers I and V.

2.1.4 Datasets

Datasets used in this thesis range from *Escherichia coli* (*E. coli*) cells (size \( \approx 10^1 \mu m \)) to zebrafish (size \( \approx 10^3 \mu m \)).

*Escherichia coli*

*E. coli* cells are prokaryotes, i.e., single cell organisms without a well defined nucleus. They are rod-shaped bacteria and grow by increasing their lengths. They are found in the intestines of warm-blooded animals [2]. In this study, we used live *E. coli* cells that were growing in a specialized trap. These cells were treated with different types of drugs and the effect on growth over time was monitored. The *E. coli* cells were used also for single macromolecule
Figure 2.1. Datasets used in this thesis. Approximate sizes of pixels are shown by the scale bars.

tracking experiments. These images were acquired at the Elf Lab, Department
of Cell and Molecular Biology, Computational and Systems Biology, Uppsala University, Sweden. Sample images of *E. coli* cells from our dataset are shown in Fig. 2.1 (a) and (b).

**Live migrating cells**
In addition to live bacterial cells we also used live eukaryotic cells, i.e., cells with a well defined nucleus and other intra cellular organelles. Live eukaryotic cells in our study included mouse mammary gland cells and human lung cancer cells. The mouse cells were used in a cell migration assay in which a micro scratch was made in a cell colony. Afterwards, the migration of cells to close the gap was monitored over time. The mouse cells dataset was acquired at the Department of Physiology & Pharmacology, Karolinska Institutet, Stockholm, Sweden.

We also imaged live human lung cancer cells over time to study their growth, cell division and migration over time. The human lung cells dataset was acquired at the Center for Biosciences, Department of Biosciences and Nutrition, Novum, Karolinska Institutet, Huddinge, Sweden. Sample images of mouse cells and human lung cancer cells are shown in Fig. 2.1 (c) and (d), respectively.

**Fixed and stained cervical and oral cells**
A detailed study of subcellular structures is possible after fixing (killing) and staining the cells. We studied fluorescently labeled human bone cancer cells, known as U2OS cells. The image data was obtained from the collection BBC022v1 [3], available from the Broad Bioimage Benchmark Collection [4]. These images were stained with different fluorescent stains highlighting various intracellular organelles. A sample image from this dataset is shown in Fig. 2.1 (e).

We also used fixed and stained human cervical cells and oral cells. These cells were stained with Papanicolaou (Pap) stain to visualize and analyze the nucleus and cytoplasm regions. These datasets were used to detect pre-cancerous lesions present in Cervix and mouth regions. The datasets consist of samples from benign and malignant cases. The human cervical cells dataset was acquired at the Regional Cancer Center, Thiruvananthapuram, Kerala, India. The oral dataset was acquired at the Swedish Dental Service Medical Dental Care, Södersjukhuset, Sweden. Sample images from cervical and oral datasets are shown in Fig. 2.1 (f) and (g), respectively.

**Spheroid cell clusters**
Samples from cancerous tumors are allowed to grow and divide over time in a laboratory setup to monitor the tumor formation. Such cells form clusters of cells known as spheroids, due to their spherical appearance. These cell cultures consist of cells attached to each other unlike individual cells in the above-mentioned datasets. Therefore, the cells in spheroids can interact...
with each other. Using several spheroids from a tumor enable high-throughput screening of drugs [5]. Our spheroid dataset consists of clusters of different sizes, shapes, and textures. The spheroid dataset was acquired at the Discovery Sciences, Innovative Medicines, AstraZeneca, Gothenburg, Sweden. An image from the spheroid dataset is shown in Fig. 2.1 (h).

**Zebrafish**

Zebrafish (Danio rerio) is a model organism used for the study of vertebrate development and gene function. Embryos of zebrafish develop externally to their mothers and have transparent bodies. Therefore, the development of organs can be easily observed. In our dataset, the zebrafish are treated with a drug that induces damage to nerve cells. As a result the affected zebrafish tails bend, while the unaffected zebrafish have straight tails. The zebrafish dataset was acquired at the Broad Institute of Harvard and MIT, Cambridge, MA, USA. One image from our dataset is shown in Fig. 2.1 (i).

### 2.1.5 Digital microscope

In the above section, I briefly discussed the different microscopy techniques used and the datasets used in this thesis. The next step involved digitization of the datasets. This was done using specialized digital cameras attached to the microscopes. The light from the observed samples is focused on a sensor (CCD or CMOS). The images formed on the sensor could be observed on a computer screen and can be saved on a hard disk for further image analysis. Digital cameras are also used for capturing time-lapse (images at regular intervals, typically 60s), images for live cell experiments. More sophisticated microscopes offer motorized stages that allow for tiled capturing of samples and stitching of the images so that the whole sample can be visualized later. All the images used in this thesis were directly acquired as digital images using different microscopes fitted with digital cameras.

### 2.2 Image analysis techniques used in this thesis

Image analysis refers to mathematical operations performed on digital images in order to extract some meaningful information. Usually the analysis of digital images consists of a combination of several processes or algorithms applied in a specific order. The application of a specific algorithm for segmentation or classification of biological objects such as cells, model organisms or spheroid clusters depends on the biological data and the type of imaging modality. Therefore, a careful choice has to be made on the application of such algorithms. Depending on the problem, the final goal of an experiment may be segmentation or classification. A comprehensive summary [6] of all image analysis techniques is beyond the scope of this thesis. Here, I discuss
the special techniques and algorithms that were developed and adapted for use in different studies. In addition to those, I used many of the more standard image analysis techniques.

2.2.1 Curvature based contrast enhancement

Typically, image analysis algorithms depend on raw pixel intensity values for image segmentation. Sometimes, the raw pixel values of foreground and background regions may be similar, resulting in poor segmentation. One way to handle this is to utilize the curvature information of the intensity landscape of the input image dataset. We used the curvature information to segment *E. coli* cells from bright-field microscopy images in Papers I, II and III.

The curvature of the intensity landscape of the input image, $I$, can be found using the eigen values of the Hessian matrix. The Hessian matrix, $H$, is defined as

$$H = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix},$$

where $I_{xx}$ and $I_{yy}$ are the 2nd derivatives of the image in the horizontal ($x$)- and vertical ($y$)- directions, and $I_{xy}$ is the derivative of the image taken first in the $x$-direction and then in the $y$-direction, using discrete approximations [7]. The eigenvalues, $k_1$ and $k_2$, can be calculated as follows

$$k_{1,2} = \frac{\text{trace}(H) \pm \sqrt{\text{trace}(H)^2 - 4 \times \text{det}(H)}}{2}$$

Depending on the input image we may use either the highest ($k_1$) or the lowest ($k_2$) eigen values. If the objects are darker than the background we use the lowest eigen values and vice versa. More details regarding the contrast enhancement can be found in Paper I.

2.2.2 Full width at half maximum

Typically, we assume an unknown distribution to be normally distributed. Assuming a Gaussian distribution makes the implementation simpler, even though we compromise the performance in some cases. Sometimes the data samples are relatively low and in such cases we estimate a Gaussian distribution. We can describe a Gaussian distribution using two parameters such as mean and standard deviation. We set the maximum value to be the mean of the distribution. Starting from the maximum we traverse to the left and right sides of the distribution until the value is 50% of the maximum value, as shown in Fig. 2.2. Then the distance between the left and right half maximum is found and is called full width at half maximum (FWHM). After finding FWHM the
standard deviation, $\sigma$, is estimated as [8]

$$\sigma = \frac{\text{FWHM}}{2\sqrt{2\log(2)}}$$  \hspace{1cm} (2.2)

We used FWHM to estimate the parameters in Paper I to post-process segmentation results.

![Figure 2.2](image)

*Figure 2.2.* Full width at half maximum is used to estimate the standard deviation of a Gaussian distribution.

### 2.2.3 Singular value decomposition for de-noising

Singular value decomposition (SVD) is used in data mining and image analysis [9]. SVD decomposes the data into orthogonal vectors. We can order these vectors in a decreasing order of their contribution to the data. Thus the most relevant part of the data becomes visible. Typically the vectors with small singular values can be discarded by setting the corresponding singular values to zero, to obtain truncated SVD. The truncated SVD can be considered similar to denoising as the noise components have typically small singular values. We used SVD for de-noising fluorescence microscopy images in Paper III.

### 2.2.4 Wavelet reconstruction for boundary enhancement

Wavelets are used to analyze image data and for feature extraction [10]. We used Daubechies 4 (Db4) wavelet [11] for enhancing “boundary like” structures in phase contrast images. We observed that artifacts in phase contrast microscopy illuminate regions other than edges as shown in Fig. 2.3 a. Therefore, we decomposed the input raw image to four levels and removed the approximation coefficients, by setting them to zero. After removing the approximation coefficients we reconstructed the image.

The reconstructed image consisted of relatively high frequency components. Thus the “boundary like” structures are highlighted in the final image.
as shown in Fig. 2.3 b. We trained a deep neural network for segmentation of cells from phase contrast images as explained in Paper III. The output probability map from the deep network using raw image is shown in Fig. 2.3 c, while Fig. 2.3 d. shows the probability map from the deep network using both raw and Db4 reconstructed images as input. The details regarding the deep network can be found in Paper III and Section 4.3. In this work, we used only Db4 wavelet but it could be possible that other types of wavelets give a similar or better results.

![Wavelet decomposition](image)

**Figure 2.3.** Wavelet decomposition. (a) Raw input image and the corresponding output probability map in (c), (b) the reconstructed image after Db4 decomposition and the corresponding output probability map in (d).

### 2.2.5 Average shape using distance transform

In cell tracking applications, sometimes the segmentation algorithms fail at properly segmenting the cells due to the limitations of the segmentation algorithm or noise present in the data. We used the result from a tracking algorithm [12] to correct segmentation errors in Paper I. In this work, we correct an error in the current frame by finding the average of the corresponding segments from the previous and next frames. We create the average image in three steps. In the first step, we find the average position using the mean positions of the
previous and next segments. In the second step we find the average orientation similarly as in step 1. The third step is to find the average shape.

We found the distance transforms of the previous input segment and its complement. The difference of these distance transforms gave an approximate signed distance transform in which the cell regions had positive values and the background regions had negative values. Similarly, the signed distance transform was found for the input segment from the next frame. Thereafter we found the average of the two signed distance transforms. The average distance transform was thresholded at a value of 0 to obtain the final average shape of the segment. The post-track segmentation correction increased the number of good quality tracks for analysis as explained in Paper I. An example showing the input images from previous and next frames and the corresponding average image in the current frame is shown in Fig. 2.4.

![Image1](a) Image1 ![Image2](b) Image2 ![Average image](c) Average image

Figure 2.4. Average shape using distance transform. (a) Input segment from the previous frame of a sequence, (b) input segment from the next frame, (c) average image obtained from (a) and (b).

### 2.2.6 Morphological geodesic active contours

Morphological geodesic active contours [13] are a level set based curve evolution method. In this method, a faster way of the curve evolution is achieved using morphological operations. The morphological operations such as erosion and dilation on binary images result in a faster and numerically stable solution. We used the morphological geodesic active contours to smooth the boundaries of cells segmented in Paper I.

### 2.2.7 Segmentation evaluation

This thesis explains several methods for segmentation of cells from microscopy images. To evaluate the performance of the segmentation algorithm and to compare with similar methods a quantitative measurement is required. One
way is to compare the segmentation results with ground truths, created man-
ually or automatically. In Paper V, we explain how to create ground truths
automatically.

Let us consider that we have an image with ground truth (GT), in cyan, and a
segmentation algorithm detects the object (Seg), in magenta, as shown in Fig.
2.5. The segmentation algorithm detects some parts (pixels) of the ground
truth and these are called true positive (TP). The regions that the segmentation
algorithm fails to detect are called false negative (FN). The wrongly detected
regions in the segmentation result are called false positive (FP) and the region
outside the ground truth that the segmentation algorithm didn’t detect is called
true negative. Based on these we define the accuracy, precision and recall as
follows [14].

\[
\text{Accuracy} = \frac{TP}{TP + FN + FP} \\
\text{Precision} = \frac{TP}{TP + FP} \\
\text{Recall} = \frac{TP}{TP + FN}
\]

Accuracy is not a good metric if the foreground and background classes are
imbalanced. Therefore we use the F-score, which is the harmonic mean of
precision and recall, as follows.

\[
F\text{-score} = \frac{2 \times \text{Precision}}{\text{Precision} + \text{Recall}} \quad (2.3)
\]

F-score is a per pixel metric and is suitable if there is only one object in the im-
age. When we evaluate the performance of an algorithm that detects multiple
objects in an image, we require a per object F-score. To find the per object F-
score we compare individual objects in the ground truth with the segmentation
result. Therefore, one object in the ground truth is assigned to only one object
in the segmentation result. As a result of this, over- and under-segmentations
are penalized in the per object F-score. All the details regarding per object F-score can be found in Paper V Supplementary material. We used per object F-score based evaluation of our segmentation results in Papers I, II, III, IV, and V. Averaging the per object F-score loses the detailed information on the performance of the segmentation algorithm. Therefore, we used an F-score plot to visualize the result as explained below.

2.2.8 Percentage of segments v/s F-score plot
We already saw that per object F-score is a suitable metric for evaluating a segmentation algorithm if there are multiple objects present in the input image. The segmentation results can be interpreted by plotting the F-score values against the percentage of objects detected. First, we sort the detected segments in the decreasing order of their F-score values. After that we set a threshold, \( T \), and change its value from 0 to 1. For each value of \( T \) we find the percentage of objects detected with F-score \( \leq T \) and plot them. Similarly, we plot the F-score values for all the different segmentation algorithms.

![Figure 2.6. F-score plot showing the performance of three methods.](image)

An example in which F-score versus the percentage of segments detected is shown in Fig. 2.6. Here, three typical segmentation algorithm behaviors, methods 1-3, are presented. Method 1 detects many objects with relatively small F-score values, seen as a sharp dip in the plot (red) near the F-score 0.5. Typically, this shows a poorly performing segmentation algorithm. Such algorithms may be used for cell tracking applications in which the detection of objects is more important than their exact delineation.

The plot for Method 2 (blue) shows that the algorithm detects a small percentage of segments with relatively high F-score. Method 2 can be considered for applications such as localizing macromolecules inside cells, where a high accurate segmentation for the detected objects is required. Method 3 (ma-
genta) shows a relatively better segmentation result that can be used for both tracking and macromolecule localization problems. We used F-score versus percentage of detected objects plots in Papers I, III, IV, and V.
3. Deep Neural Networks

Deep neural networks are undoubtedly one of the most powerful technologies for machine learning applications today. The study of deep neural networks reached the current stage by traversing different research pathways. The predecessors of deep neural networks date as early as the middle of the 20th century.

3.1 Background

McCulloch and Pitts created a linear model of a neuron in 1943 [15]. Later, Rosenblatt developed an adaptive linear model that contained learnable weights [16]. An early form of the present popular convolutional network known as Neocognitron was introduced by Fukushima [17] in 1980. The convolutional network by LeCun et al. [18] was one of the first successful applications of the convolutional neural networks. The modern day convolutional neural networks share a very similar architecture to this model. The developments in neural networks were rather slow due to the lack of computational resources and the availability of large annotated datasets during the later part of the 20th century.

The deep neural network research gained momentum when Hinton et al. [19] were able to successfully train a layer-wise deep autoencoder. The advances in hardware enabled graphics processing units (GPU) to be used as general-purpose computation devices. The GPU enabled faster and cheaper computations so that deeper and larger neural networks were possible to train. The famous Imagenet Large-Scale Visual Recognition Challenge (ILSVRC), containing over a million annotated natural images served greatly in deep neural network research. In 2012, Krizhevsky et al. [20] won the ILSVRC challenge with a top-5 error rate of 15.3%. They were able to train a deep convolutional neural network using multiple GPUs. This achievement marked the turning point in the history of deep learning. Thereafter, the popularity for deep neural networks increased rapidly. Later, all the models that won the ILSVRC challenges were deep neural networks. Currently, the state-of-the-art top-5 error is less than 3%.

In the following sections, I discuss the major components required to train a deep neural network model. In this thesis, I interchangeably use notations such as deep learning, deep network or deep model to refer to a deep neural network. Even though there are other types of representation learning methods like the deep Gaussian process [21] or K-means [22], I focus my discussion on deep neural networks used in this thesis work.
3.1.1 Deep neural networks

![Deep neural network diagram](image)

Figure 3.1. A deep network with two convolutional layers, two pooling layers and two fully connected layers.

Deep neural networks are neural networks with two or more hidden layers. The depth in a deep neural network refers to the number of hidden layers in it. The layers in a deep neural network create mappings of input data to the required outputs. A typical deep neural network with five hidden layers is shown in Fig. 3.1. In the following sections, I explain the basic building blocks to create deep neural networks used in this thesis. I also discuss an advanced type of deep neural network known as generative adversarial networks, which we used to segment spheroid clusters from bright-field images in Paper IV. A trained neural network is created using three major components: 1) data, 2) a model, and 3) training, as described below.

**Data**

Let us assume that there is a data generating process, $p_{data}$, that creates the dataset. Typically the $p_{data}$ distribution is unknown or we have limited knowledge of the data generating process. In such cases we have samples, $\hat{p}_{data}$, from the original distribution. The data samples are called a dataset. The dataset consists of the raw data such as input images and their corresponding ground truths (annotations). The ground truths represent the class belongingness of the images. For a classification problem, the ground truths are scalar values showing the class a particular image belongs to. For a segmentation problem, the ground truth means a labeled image where each label indicates the class a particular pixel in the image belongs to.

For training a neural network we divide the dataset into a training set, a validation set and a test set. We train the network using the training set and perform parameter tunings using the validation set. Once optimum parameters are found we perform a final testing using the test set to evaluate the performance of our trained network model. The image dataset is usually pre-processed before feeding it to a neural network. For image datasets the typical pre-processing consists of normalizing the images. The normalization can be done in different ways depending on the problem and the availability of the dataset. In this work, we usually apply mean (or median) subtraction, followed by a division by standard deviation or the max value of the image. After normalization, the input images to the network are in the range $[-1, 1]$. 

32
Model
Once we have pre-processed the dataset we design a deep neural network, $p_{model}$. The deep neural network model learns about the distribution, $p_{data}$, using the normalized empirical dataset $\hat{p}_{data}$. The design of the deep neural network involves several factors such as the complexity of the dataset, the type of GPU available for training, the performance required etc. The choice of architecture greatly affects the performance of a network and typically involves many trials with various configurations. In this thesis, all the deep neural networks were designed as convolutional neural networks. The convolutional neural networks are a type of neural networks specially meant for images or regular grid like structures. The details regarding neural network models are provided in Section 3.2.

3.1.2 Deep neural network training
The training of a neural network means to find the suitable parameters so that input data can be mapped to the desired output. There are three major steps involved in this process; 1) a forward pass, 2) a backward pass and 3) a parameter update, as described below.

Forward pass

In the forward pass, we provide an input and a target or ground truth to the neural network. The target represents the belongingness class of the input. A neural network consists of an input layer, an output layer and one or more hidden layers. The layers consist of elements called neurons. A neuron receives one or more inputs, $x_i$, from the previous layer and computes a weighted sum of these inputs using weights, $w_{ij}$, and adds a bias, $b_j$, to the sum. The fi-
nal value is usually subjected to a non-linear activation function, $f$, to get the output, $y_j$. This whole operation of a neuron can be represented as

$$ y_j = f\left(\sum_i w_{ij}x_i + b_j\right) \tag{3.1} $$

The weights in each layer are also called model parameters and the outputs are also called feature maps. The output of one layer is the input for the succeeding layer. Thus, the outputs for all layers are usually computed in a sequential manner. After computing the outputs of the hidden layers, we compute the output of the final layer; the output layer. This final output, $o$, and the target, $t$, are provided to a loss function, $l$, to compute the final loss of the network. The loss is a quantification of how well the neural network is able to map the input to the given target:

$$ \text{loss} = l(o,t) \tag{3.2} $$

The whole process of computing the output of each layer and finding the final loss is called forward pass and is schematically represented in Fig. 3.2.

**Backward pass**

![Backward pass in a neural network with three hidden layers.](image)

*Figure 3.3. Backward pass in a neural network with three hidden layers.*

After the forward pass the loss function outputs a loss generated by the network. Now, we compute the contribution of each weight or parameter to the loss. This is computed by a process called backward pass or back propagation. Since the output depends on the intermediate hidden layers we compute the gradient of the output, $o$, with respect to the input, by applying the chain rule of differentiation. We compute the partial derivatives of the loss with respect to the weights, $w_{ij}$. Each gradient component is computed as:

$$ \frac{\partial l}{\partial w_{ij}} = \frac{\partial l}{\partial o} \frac{\partial o}{\partial y_j} \frac{\partial y_j}{\partial w_{ij}} \tag{3.3} $$
where $\frac{\partial l}{\partial o}$ is the partial derivative of the loss with respect to the output of the network, $\frac{\partial o}{\partial y_j}$ is the partial derivative of the output with respect to the $j^{th}$ neuron, and $\frac{\partial y_j}{\partial w_{ij}}$ is the partial derivative of the $j^{th}$ neuron with respect to the $i^{th}$ weight. If there are more hidden layers we multiply the partial derivative of the next hidden layer neuron to get the total partial derivative for the current neuron. The process of the backward pass is illustrated in Fig. 3.3.

**Parameter update**

The forward and backward passes do not change the weights or parameters in the network. After computing all the gradients during the backward pass, we change the values of all the parameters so that the network error is decreased. This is done by changing the parameter values in the opposite direction of the gradient. Typically, we make small changes in the parameter values, called step or learning rate, $\eta$. This optimization approach, when the parameters are iteratively updated utilizing information about the direction of a negative gradient, is called Gradient descent. The update is applied to all the parameters simultaneously. The details regarding the parameter updates are discussed in Section 3.5.

**3.2 Convolutional neural networks**

A convolutional neural network is a neural network with one or more convolutional layers. The convolutional layer implements the mathematical function of convolution. A 2D convolution of two functions, $I$ and $K$ is defined as:

$$S(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n) \tag{3.4}$$

where $I$ is the image, and $K$ is the convolution kernel. The negative signs for $m$ and $n$ indices of $K$ show that the kernel is flipped both in the vertical and horizontal directions and thereafter translated over the image $I$. But the use of a flipped kernel has no importance on the learning process of deep neural networks, so most of the deep learning libraries use non-flipped convolution operations known as *cross-correlation* [23]. Therefore, we can change the above Equation to:

$$S(i, j) = \sum_m \sum_n I(m, n)K(i + m, j + n) \tag{3.5}$$

This operation implies that the kernel, $K$, interacts with only a small region, specified by its size $(m \times n)$, in the input image. The parameters of the kernels are shared by all nodes in a layer, hence, reducing the number of parameters compared to fully connected neural networks (Sec. 3.2.4). The reduction in the total number of parameters enables the convolutional neural network
to train faster with reduced overfitting. All the popular deep learning tools implement the convolution operation using matrix multiplication to speed up computation. Here, for simplicity we represent all the convolutions using the regular convolution with kernels.

3.2.1 Convolutional layers
A convolutional layer implements the convolution operation. Usually the convolution function is specified by a few parameters. These parameters include the number of inputs, the number of outputs, and the width and the height of the kernel. Typically in deep neural networks we use 3D kernels, i.e., a stack of 2D kernels. The number of inputs decides the number of 2D kernels required for each kernel and the number of outputs determine the number of 3D kernels required. The outputs created by the kernels are called feature maps as they represent some features detected by the kernel.

An example is depicted in Fig. 3.4. Here, the number of inputs is three as there are three channels in the input and the number of outputs is two as there are two kernels, the green and yellow. Each kernel is of size $3 \times 3$. The green kernel moves around the input feature maps, in layer $n$, and creates one output feature map, red output here, for layer $n+1$. Similarly, the yellow kernel outputs blue pixels.

![Figure 3.4. Convolution operation using stacks of 2D kernels. The green kernel outputs red pixels and the yellow kernel outputs blue pixels.](image)

There are other parameters like the step size, or stride, and zero padding size. The zero padding is applied to deal with the uncertainty at the borders. If convolution is applied without padding then the spatial size of the output...
feature map decreases imposing a restriction to the depth of the deep network. If a kernel of size \( k \times k \) is used then a zero padding of size \( \frac{k-1}{2} \) is required on all sides of the input feature map to get an output that is the same spatial size as the input.

Usually the convolution operation is applied to every pixel in the input feature map. We can specify the step size of the convolution operation, i.e., we can apply a convolution and skip a few pixels and perform convolution again and repeat this process. Such a convolution is called strided convolution. Strided convolution is used to reduce the spatial feature map size of the output and is computationally more efficient than applying a regular convolution followed by a pooling operation. An example of a convolution with a stride of two is shown in Fig. 3.5. A convolution with a stride of two outputs a feature map with half the spatial size as the input feature map.

![Fig. 3.5](image)

*Figure 3.5. Two consecutive positions of a 3 × 3 kernel with a stride of 2.*

### 3.2.2 Dilated convolution

Another variant of convolution operation is called dilated convolution [24]. In a regular convolution, the convolution is applied to the neighboring pixels, while, in a dilated convolution, the convolution is applied to far away pixels. The re-distribution of the values of the kernel (comparing a regular convolution) is specified by a dilation parameter. Dilated convolutions enable feature learning from a larger area without compromising the resolution even after down sampling the feature map. Multiple dilated convolutions exponentially increase the feature learning [24]. An example of a dilated convolution with a kernel of size 3 × 3 and dilation of 2 is shown in Fig. 3.6. The 3 × 3 convolution with a dilation of 2 is equivalent to a 5 × 5 filter with non-zero values only at 9 locations. The dilated convolution is used in Paper IV.

### 3.2.3 Deconvolution

Deconvolution is used to increase the spatial size of an input feature map, to compensate for the decrease in size due to max pooling (refer Section 3.2.5)
or strided convolutions. The increase of feature map size is usually done on fully convolutional networks used for segmentation (refer Section 3.2.7). Deconvolution is also known as transposed convolution or fractionally strided convolution [25]. Deconvolution can also be described by up-sampling with a linear interpolation (or other types of interpolations) to increase the image size. In actual practice, we let the kernels learn the parameters for interpolation. The deconvolution operation is usually specified by kernel size and stride parameters followed by the zero padding settings. Typically the up-sampling operation is done in two steps. In the first step, the input feature map is distributed based on the stride size. In the second step, a regular convolution is applied to the restructured data. A schematic representation of the steps of the deconvolution operation is shown in Fig. 3.7. Here a $2 \times 2$ stride is applied using a kernel of size $2 \times 2$ to get an output that is twice the size of the input.

3.2.4 Fully connected layers

In fully connected layers, all the input neurons are connected to all the output neurons. The fully connected layers are usually added towards the end of the deep network. While the convolutional layers typically act as feature extraction layers, the fully connected layers act as classification layers. The input 2D feature maps are flattened to a 1D vector by rearranging the values. The parameters of a fully connected layer are specified by the number of inputs and the number of outputs required. If the input 1D vector has a size of $m$ and the output vector has a size of $n$ then a parameter matrix of size $m \times n$ is required. A schematic diagram of a fully connected layer is shown in Fig. 3.8. Here, the input layer consists of 8 nodes and the output consists of 5 nodes, so that a matrix with $8 \times 5$ entries is required. We used fully connected layers for the classification networks in Papers VI and VII.

3.2.5 Pooling layers

Pooling layers are usually down-sampling layers added to the deep network to reduce the spatial dimensions of the feature maps. The down-sampling operation has two purposes. First, the feature map size can be reduced enabling
Figure 3.7. The steps of a $2 \times 2$ deconvolution with a stride of $2 \times 2$ to create an output feature map that is twice the input spatial size. (a) Data rearrangement to increase the spatial size of the output image. (b) Convolution to fill the missing values for the desired output.

Figure 3.8. A fully connected layer with input size of 8, output size of 5 and 40 parameters.

storage of deeper network models in the limited GPU memory. Second, the spatial area on which the convolution kernel acts is effectively increased. This is analogous to the spatial pyramid analysis popularly used in various image analysis tasks.

For pooling, we can apply linear operations, such as averaging, or order statistic operations, like maximum across a feature map. The averaging operation outputs the average value of the feature map in the region determined by
the kernel. Similarly, the max operation outputs the maximum value. Down-sampling with the max operator is used in many deep networks for segmentation and classification [20, 26]. This is due to the property that max operation preserves the largest activations in the feature maps. An example of a pooling operation with a $2 \times 2$ kernel with stride 2 is shown in Fig. 3.9. Here, the output is reduced to half the spatial size of the input.

Another type of pooling operation is known as global pooling. A popular global pooling is global average pooling. The global average pooling operator takes all the feature maps and outputs an average value per feature map. This is typically used in fully convolutional networks for classification. The fully convolutional networks are explained in Section 3.2.7.

### 3.2.6 Activation function layers

Usually after the linear layers, such as convolutional and fully connected layers, a non-linear function is applied. The layers applying the non-linear functions are called activation function layers, and the functions are referred to as activation functions. Some typically used activation functions are shown in Table 3.1. Traditionally, functions such as sigmoid or tanh were used as activation functions due to their convenience in implementation as well as easy differentiation.

**Vanishing gradient**

The sigmoid and tanh activation functions suffer from the problem of vanishing gradient. If we have a very deep network and the convolutional layers are giving very large responses then the activation functions bound them to values near 1. Therefore, the resulting gradients are very small. For example, consider that we take a sigmoid activation function as in Table 3.1. Let $x = 10$, 

![Figure 3.9. A pooling layer with $2 \times 2$ kernel and stride 2.](image)
then \( f(x) = 0.9999 \). The gradient can be computed as

\[
\frac{df}{dx} = f(x) \cdot (1 - f(x))
\]

\[
\left. \frac{df}{dx} \right|_{x=10} = f(10) \cdot (1 - f(10)) = 0.9999 \cdot (1 - 0.9999) = 9.9989 \times 10^{-5}
\]

This gradient is propagated backward and is multiplied by the local gradient in that layer. Then the total gradient becomes an extremely small value, and after a few layers the gradient goes below the machine precision, resulting in zero gradient. Therefore the network stops learning.

**Table 3.1.** Activation functions and their corresponding derivatives. ReLU is not differentiable at \( x = 0 \), and 0 is taken as the derivative at this point for practical purposes.

<table>
<thead>
<tr>
<th>Function</th>
<th>( f(x) )</th>
<th>( \frac{df}{dx} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>sigmoid</td>
<td>( \frac{1}{1+e^{-x}} )</td>
<td>( f(x)(1-f(x)) )</td>
</tr>
<tr>
<td>tanh</td>
<td>( \frac{e^x - e^{-x}}{e^x + e^{-x}} )</td>
<td>( 1 - f^2(x) )</td>
</tr>
<tr>
<td>ReLU</td>
<td>( \text{max}(0,x) )</td>
<td>1, if ( x &gt; 0 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0, else</td>
</tr>
</tbody>
</table>

**ReLU**

The Rectified Linear Unit (ReLU) [27] was introduced to overcome the shortcomings of the tanh and sigmoid activation functions. The ReLU function

\[
\text{ReLU}(x) = \max(0, x)
\]

\[
\text{ReLU}(x) = \begin{cases} 
0, & \text{if } x < 0 \\
\text{otherwise}, & \text{otherwise}
\end{cases}
\]

**Figure 3.10.** Activation functions (a) Sigmoid, (b) Tanh and (c) ReLU functions.
passes all values above 0 as they are, while all value less than 0 are set to 0. The ReLU has the advantage that it has a gradient of 1 for any value above 0. This greatly helps in propagating the gradient without vanishing since there is no upper limit of the response of this function (refer Table 3.1). However, now there is a problem of exploding activations, i.e., increase of activations as a result of several consecutive convolutions. Using proper regularization of the layers such as drop out or batch normalization can compensate this. Details regarding regularization are given in Section 3.4. We used ReLU in Papers III, IV, V, VI, and VII. There are other variants of ReLU, such as leaky ReLU [28] and parametric ReLU (PReLU) [29] that have activations for negative inputs also. A recent version of ReLU known as Scaled Exponential Linear Unit (SELU) [30] can automatically normalize the activations for very deep networks.

3.2.7 Fully convolutional neural networks

Deep networks with only convolutional layers and no fully connected layers are called Fully Convolutional Neural Networks (FCNN). FCNNs are used for both classification and segmentation. For a classification network the fully connected layers contain a very large number of parameters and often increase the size of the network model. Instead of a fully connected layer a global average pooling is typically used to reduce the feature map size. The global average pooling layer has no parameters. We used a global average pooling for the adversarial network (see Section 3.5.6 ) in Paper IV. An example of a classification network with the global average pooling is shown in Fig. 3.11 a.

In non-fully convolutional deep networks (deep networks with fully connected layers), segmentation is done as a pixelwise classification in which a patch of the input image is fed to a deep network and the network outputs one pixel of prediction. The patch-wise segmentation is repeated for the whole image. This operation is computationally very expensive since we have to redundantly compute the feature maps for neighboring pixels. The feature maps computed for a particular patch are computed again for a neighboring patch. On the other hand in a FCNN, we feed the network with a patch of an image and we get an output patch instead of a single pixel. Patch-wise pixel prediction is thus done more efficiently.

After the introduction of FCNNs for semantic segmentation [31] usually the whole image, or a large portion of the image, typically 480 × 480 sized patches, were used for segmentation. The segmentation network usually consists of a down-sampling path and an up-sampling path. In the down-sampling path, the spatial size of the feature maps decreases along with convolutions, while in the up-sampling path, the low sized feature maps are deconvolved to increase the spatial size. A softmax function (refer Section 3.3.4) is applied to the output of the last convolutional layer to get the output probability maps for
segmentation. We used FCNNs for segmentation in Papers III, IV and V. An example of a segmentation network with two convolutional, two max pooling and two deconvolutional layers is shown in Fig. 3.11 b.

3.2.8 Residual connections

We can consider a convolutional layer as a mapping from one feature space to another. Let such a mapping be $H(x)$. Instead of directly finding the mapping $H(x)$ we find another mapping (residual) $F(x) = H(x) - x$. Now the mapping becomes $H(x) = F(x) + x$ as shown in Fig. 3.12 a. He et al., hypothesize that it is easier to optimize for the residual mapping rather than the original mapping [32]. This residual mapping can be realized using skip connections as shown in Fig. 3.12. Thus the name residual, or skip connections.

During the backward pass the gradient can easily flow backwards through the skip connection rather than the non-linear path. Residual connections are widely used in most of the recent deep network architectures. We also apply various formulations of residual connections in Papers IV and V. We use two variants, one with only one non-linear mapping and the second one with concatenation rather than addition. The modifications are made considering the possible size and capacity of the model.

3.3 Loss functions

The loss function has an important role in the training of a deep neural network. The loss function provides a quantification of the error of the network prediction and drives the learning process of the network. Typically the loss functions are non-trainable and do not contain any trainable parameters. However, some special types of loss functions, such as adversarial loss, contain not
only parameters but also an entire deep neural network. The details regarding the adversarial networks can be found in Section 3.5.6. We now discuss a popular loss function known as cross-entropy loss.

3.3.1 Cross-entropy loss

Traditionally, the Mean Squared Error (MSE) was used as the loss function for classification problems. The models trained with the MSE suffered from saturations and slow learning when used with sigmoid and softmax activations (described in Section 3.3.4). Nowadays, cross-entropy loss has replaced the MSE loss function. Usage of the cross-entropy loss function is considered as one of the algorithmic factors that made deep neural networks a success [33]. The cross-entropy loss function is also known by the name softmax loss.

Consider a true data generating distribution $p_{data}$. We want to estimate it using a deep neural network model, $p_{model}(Y \mid X; \theta)$. Where $X = x_1, \ldots, x_m$ is a dataset of $m$ independent examples drawn from the $p_{data}$ and $\theta$ is the distribution of parameters of the network. Let $Y = y_1, \ldots, y_m$ be the labels corresponding to $X$. The cross-entropy loss can be computed in terms of the
training samples as

\[
\text{Cross-entropy} = -\frac{1}{m} \sum_{i=1}^{m} y_i \cdot \log p_{\text{model}}(y_i \mid x_i; \theta)
\] (3.6)

where each \(y_i\) is typically a one-hot encoded vector, i.e., only the correct class is assigned the value one and all other values are zero. A special case of cross-entropy loss for classification to two classes is known as binary cross-entropy loss.

In the following sections we will see how the maximum likelihood estimate for the model parameter \(\theta\) is related to Kullback-Leibler divergence between the training data, \(\hat{p}_{\text{data}}\), and the model distribution, \(p_{\text{model}}\). We then see the relationship between the Kullback-Leibler divergence and the cross-entropy.

### 3.3.2 Maximum likelihood estimation

The Maximum Likelihood Estimate (MLE) for \(\theta\) [33] is

\[
\theta_{\text{MLE}} = \arg \max_{\theta} p_{\text{model}}(Y \mid X; \theta).
\] (3.7)

Since all data points are independent and identically distributed,

\[
\theta_{\text{MLE}} = \arg \max_{\theta} \prod_{i=1}^{m} p_{\text{model}}(y_i \mid x_i; \theta).
\] (3.8)

This product of \(m\) terms may result in numerical instabilities so the product can be converted to a sum by taking the logarithm of the likelihood function, and the maximization gives the same value of \(\theta_{\text{MLE}}\). Thus, the above Equation can be converted to

\[
\theta_{\text{MLE}} = \arg \max_{\theta} \sum_{i=1}^{m} \log p_{\text{model}}(y_i \mid x_i; \theta)
\] (3.9)

The \(\arg \max\) remains the same even if we divide the log-likelihood function by \(m\) (or any scalar value). Then the objective function becomes an expectation with respect to the \(p_{\text{data}}\) defined by the training data.

\[
\theta_{\text{MLE}} = \arg \max_{\theta} \mathbb{E}_{x \sim p_{\text{data}}} p_{\text{model}}(y_i \mid x_i; \theta).
\] (3.10)

### 3.3.3 Kullback-Leibler divergence

MLE can also be seen as minimization of the dissimilarity between the empirical distribution \(\hat{p}_{\text{data}}\) defined by the training samples and the model distribution, \(p_{\text{model}}\) [33]. The dissimilarity is defined by Kullback-Leibler [34] (KL) divergence as

\[
D_{\text{KL}}(\hat{p}_{\text{data}} \mid\mid p_{\text{model}}) = \mathbb{E}_{x \sim \hat{p}_{\text{data}}} [\log \hat{p}_{\text{data}}(x) - \log p_{\text{model}}(y \mid x; \theta)]
\] (3.11)
where $E$ is the expectation. We can see that the first term depends on the data generating distribution and not a function of the model. So during training we have to minimize the second term only

$$-E_{x \sim p_{\text{data}}} \left[ \log p_{\text{model}}(y \mid x; \theta) \right].$$

This minimization of KL divergence is equivalent to the maximization of MLE in Equation 3.10. Furthermore, minimization of KL divergence is equivalent to minimization of the cross-entropy, Equation 3.6, between the two distributions [33]. Therefore cross-entropy, is considered as the preferred loss function in this thesis.

### 3.3.4 Softmax activation

Supervised deep neural networks are typically used for classification. We can consider segmentation as a pixelwise classification in some sense. The output of a classifier is usually a categorical prediction involving $n$ unnormalized outputs, $z_1, \ldots, z_n$. Therefore to use cross-entropy as the loss function, we have to normalize the outputs to be probabilities. This is enforced by softmax activation. For the $k^{th}$ output, the softmax is

$$\text{softmax}(z_k) = \frac{\exp(z_k)}{\sum_{j=1}^{n} \exp(z_j)}.$$  \hspace{1cm} (3.12)

To compute the cross-entropy between the model distribution, $p_{\text{model}}(y_i \mid x_i; \theta)$ and the data distribution, $y_i$, as in Equation 3.6, we take the logarithm of the softmax function

$$\log \text{softmax}(z_k) = z_k - \log \sum_{j=1}^{n} \exp(z_j).$$  \hspace{1cm} (3.13)

Let us represent the $n$ log softmax outputs by $z$ and substitute it in the place of $\log p_{\text{model}}(y_i \mid x_i; \theta)$ in Equation 3.6 to get

$$\text{Cross-entropy} = -\frac{1}{m} \sum_{i=1}^{m} y_i \cdot z_i.$$  \hspace{1cm} (3.14)

The formulation of log likelihood, in Equation 3.13, has some advantages. The exponentiation is canceled by the logarithm for the current prediction $k$, therefore the loss function, in Equation 3.14, can continue learning without saturation.

When we train a deep network with the cross-entropy loss, the first term in Equation 3.13, $z_k$, tries to increase the activation of the correct output, and the
second term, \(-\log \sum_{j=1}^{n} \exp(z_j)\), tries to decrease all the outputs. If we assume that \(z_k\) is the largest activation and all other outputs are small then

\[
\log \text{softmax}(z_k) \approx z_k - \max_j (z_j) \\
\approx z_k - z_k = 0
\]

(3.15)

To conclude, a correctly classified example gives very small contribution to the loss function, while the majority of the contributions are from other wrongly classified examples.

### 3.3.5 Numerically stable softmax

The softmax activation in Equation 3.12 is numerically unstable. The softmax output gets saturated if the differences between the input values are large. To make it stable, we remove the maximum output from all of the outputs and the modified softmax function \([33]\) becomes

\[
\text{softmax}(z_i) = \frac{\exp(z_i - \max_j (z_j))}{\sum_{j=1}^{k} \exp(z_j - \max_j (z_j))}
\]

(3.16)

This numerically stable softmax was used in Papers III, IV, V, VI, and VII.

### 3.3.6 Spatial cross-entropy loss

We consider segmentation as pixelwise classification and we apply the cross-entropy loss over the whole image area. Thereafter we take the average over the image to get spatial cross-entropy loss as output. If there are \(c\) classes for segmentation, the target \(T\) and the log softmax activated output \(Y\)’s are three-dimensional with a depth dimension of size \(c\). Then the spatial cross-entropy (SCE) loss is

\[
SCE = \frac{\sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{k=1}^{c} Y(i, j, k)T(i, j, k)}{NM}
\]

(3.17)

Typically the target is one-hot encoded [33], i.e., only the correct class is one and all other classes are zero, so only the correct class contributes to the loss.

### 3.3.7 Weighted spatial cross-entropy loss

The spatial cross-entropy loss in equation 3.17 gives the same preference to all pixels in an image. In some cases the area of the foreground region may be
smaller than the background region, and sometimes we want to prioritize the network to learn more from the foreground regions. In such situation we use weighted spatial cross-entropy loss (WSCE). The WSCE can be shown as

\[
WSCE = \frac{\sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{k=1}^{c} Y(i, j, k)T(i, j, k)W(i, j)}{\sum_{i=1}^{N} \sum_{j=1}^{M} W(i, j)}
\]

(3.18)

where \(W\) is the two-dimensional weight image. The WSCE enables us to weigh individual pixels rather than individual classes. We used WSCE loss function in Papers III, IV, and V.

3.4 Regularization of deep networks

The deep neural networks we train may give excellent results on a training set but may fail to give reasonably good results on a test set. Regularization is the process that enables the model to make fewer errors on the test set.

When we train a deep network we have a data generating process creating \(p_{\text{data}}\); a training set, \(p_{\text{data}}\); and the deep network model, \(p_{\text{model}}\). During training there are three possibilities: 1) the \(p_{\text{model}}\) might deviate from \(p_{\text{data}}\) but fits to \(p_{\text{data}}\), and the network is said to be biased. 2) The \(p_{\text{model}}\) matches with \(p_{\text{data}}\) resulting in generalization. 3) \(p_{\text{data}}\) is sampled not only from \(p_{\text{data}}\) but also from many other distributions, and \(p_{\text{model}}\) learns properties or variations that are specific to these other distributions resulting in high variance. We reduce the variance of \(p_{\text{model}}\) and make it more generalizable by regularization.

3.4.1 Parameter norms

Parameter regularization norms impose restrictions on the values a network parameter can take, thereby limiting the capacity of the model and preventing overfitting. Suppose we create a deep network with the objective function \(J(\theta; X, Y)\), where \(\theta\) is the vector of parameters, \(X\) the input data set and \(Y\) the corresponding targets. Then the regularized objective function is

\[
\tilde{J}(\theta; X, Y) = J(\theta; X, Y) + \lambda R(\theta)
\]

(3.19)

where \(\lambda\) is a weight term determining the contribution of the regularization \(R(\theta)\). Typically biases (see equation 3.1) are unregularized and regularization is applied to weights only, \(R(w)\).

There are two main types of parameter regularizations: the \(L_2\) and \(L_1\) norms. For the \(L_2\) norm, the regularization function is

\[
R_{L2}(w) = \frac{1}{2} \sum_i w_i^2
\]

(3.20)
where \( w \) represents all the weight parameters except the biases. The \( L_2 \) norm tries to reduce all the weight values towards 0 as each high value will have a large impact. Therefore, \( L_2 \) norm regularization restricts the network weights from taking very large non-zero values. For the \( L_1 \) norm regularization, the regularization function is

\[
R_{L_1}(w) = \sum_i |w_i|
\]  

(3.21)

The \( L_1 \) norm tries to make the parameters more sparse, i.e., many weight values become 0. The \( L_1 \) norm regularization can be seen as a feature selection method.

### 3.4.2 Dataset augmentation

We can make the deep neural network model generalize better by applying dataset augmentation. This is based on the idea that training on more data helps the network to learn better from the data and to avoid the network to learn certain unwanted variations. If there is a specific illumination pattern in the microscopy data then the network might learn the orientation of such an illumination and the model may perform poorly on a similar dataset without the same illumination pattern.

In the case of segmentation and classification of microscopy images, the orientations of the cells are usually not important factors. We increase the total amount of data by spatial transformations such as rotations, flips and random translations, of the input images. It is tedious and expensive to create training samples for cell segmentation by manually marking several thousands of cells. We therefore manually annotate only a few hundred cells and augment the data to create enough images for training our network models. We performed data augmentation in Papers III, IV, V, VI, and VII. We also present an alternative to manual annotations for training data creation in Paper V.

### 3.4.3 Early stopping

During the deep neural network training, we divide the dataset into three parts such as a training set, a validation set and a test set. We train the network on the training set and perform parameter tunings using the validation set. The test set is only used for final evaluation.

We assume that the network model has enough capacity to overfit the training set. When training the network model for several iterations we may see that the training set error decreases and the validation set error increases showing that the network is overfitting. Instead of using the final network parameters it is favorable to use the intermediate parameters that resulted in the least validation set error. This is usually considered as early stopping. The regular-
ization by early stopping is considered similar to $L_2$ norm regularization of the parameters [33]. We used early stopping in Papers III, IV, V, and VI.

### 3.4.4 Drop out

In classification problems, using multiple classifiers or ensembles of classifiers and combining the final results, was shown to give improved performance as compared to any of the individual classifiers [35]. This involves training several deep neural networks and validating the performance on separate validation sets and performing early stopping. This process is computationally very expensive.

Drop out was introduced as an alternative approach to the ensemble classifier [36]. In the drop out regularization, the network connections between layers are randomly dropped with a probability $q$ (inclusion probability of $1 - q$) during training. During the inference (testing) time, the network weights are multiplied by a factor of $1 - q$. The random drop of the connections in every iteration means that effectively we train a different network with every iteration. Therefore, the final network is similar to an ensemble of several thousands of neural networks, provided we train the network for several thousands of iterations. We used drop out regularization in Papers VI and VII.

### 3.4.5 Batch normalization

Batch normalization is an important finding that enables the deep networks to learn faster without paying much attention to exact parameter initialization. During training with batch normalization, we take a mini-batch of a dataset and forward pass it through the network. After each linear operation, such as convolutional layer or fully connected layer, the output is normalized by removing the mean value and dividing by the standard deviation (std) of the output over the mini-batch. Usually, the batch normalization is applied to each feature map separately over the mini-batch. If $X = x_1, \ldots, x_n$, are the $n$ feature maps (each $x_k$ is formed by all the corresponding feature maps of the mini-batch) from a linear layer, then batch normalization for $k^{th}$ feature map can be expressed as

$$\hat{x}_k = \frac{x_k - \text{mean}(x_k)}{\text{std}(x_k)}$$  \hspace{1cm} (3.22)

Mean centering and dividing by standard deviation limits the representation capacity of the network so usually we apply a scaling by $\gamma$ and a shift by $\beta$. The parameters $\gamma$ and $\beta$ are set as learnable parameters. Then the batch-normalized feature map becomes

$$y_k = \gamma_k \hat{x}_k + \beta_k$$  \hspace{1cm} (3.23)
The batch normalization improved the performance of our deep network and we observed that training with high learning rates was possible in Papers IV, V, and VII.

3.5 Optimization of deep neural networks

Optimization of deep neural networks means that we minimize the value of an objective function. For deep neural networks the objective function is the loss function. For simplicity, we assume a loss function without regularization. When training the deep network we pass the data through the network and find the loss according to the loss function. The loss is then back-propagated to alter the values of network parameters. Most of the popular optimization algorithms used for training the deep networks are based on the gradient of the loss function and reduce the loss at every iteration of training by following the direction of negative gradients until convergence. Such methods are called as gradient descent algorithms.

3.5.1 Gradient descent

Gradient descent (GD) is the most popular type of optimization technique for training deep neural networks. Consider that we have an objective function \( J(\theta; X, Y) \), where \( \theta \) is the vector of parameters of the network, \( X \) the inputs and \( Y \) the corresponding targets. The GD involves the computation of the gradient of the optimization function with respect to its parameters, \( \nabla_\theta J(\theta) \). Once the gradient is calculated the parameters are updated in the opposite direction of the gradient. Usually the parameters are updated in small steps known as learning rate, \( \eta \). This process of parameter update is known as gradient descent. Based on the size of the dataset used in the GD algorithm there are three main variants: 1) Batch gradient descent, 2) Stochastic gradient descent and 3) Mini-batch gradient descent.

3.5.2 Batch gradient descent

Batch gradient descent (BGD) uses the entire training set to update the parameters.

\[
\theta = \theta - \eta \cdot \nabla_\theta J(\theta)
\]

(3.24)

The BGD outputs a monotonically decreasing loss function. Finding the gradient of the objective function for the entire training set for a single update of parameters is computationally very expensive. Typically the training set contains several millions of images and may not fit in the physical memory.
3.5.3 Stochastic gradient descent

Stochastic gradient descent (SGD) uses one sample from the dataset to make an update of the parameters.

\[ \theta = \theta - \eta \nabla_{\theta} \mathcal{J}(\theta; x_i; y_i) \] (3.25)

where \( x_i \) and \( y_i \) is the \( i^{th} \) training sample. The SGD enables training on very large datasets without running into memory issues. Using SGD for optimization results in a very fluctuating output loss.

3.5.4 Mini-batch gradient descent

Mini-batch gradient descent (MBGD) is the intermediate approach between the extreme cases of BGD and SGD. It takes a subset of the whole training set, called mini-batch, to make an update of the parameters.

\[ \theta = \theta - \eta \nabla_{\theta} \mathcal{J}(\theta; x_{(i:i+n)}; y_{(i:i+n)}) \] (3.26)

where \( n \) is the mini batch size. The MBGD results in a smoother output loss than SGD. The mini batch size can be tuned to fit into the physical memory size. The memory size and the processing cores of modern GPUs enable efficient computation of MBGD.

3.5.5 Adaptive methods

Even though MBGD is easy to implement it is not free from challenges. Choosing a proper learning rate \( \eta \) is difficult. This is often problem specific and requires careful monitoring of the deep network optimization. MBGD does not allow automatic learning rate adjustments during training. MBGD is also susceptible to be trapped in saddle points [37] due to near zero gradients around them.

To overcome the limitation of MBGD several adaptive methods were proposed [38]. A few of the popular methods are: 1) Momentum, 2) Nesterov accelerated gradient, 3) Adagrad, 4) RMSProp, and 5) Adam as described below.

**Momentum**

Momentum [39] helps in accelerating MBGD in relevant directions and damps oscillations. It is done by adding a fraction of the previous parameter updates to the current update.

\[ v_t = \gamma v_{t-1} + \eta \nabla_{\theta} \mathcal{J}(\theta) \]

\[ \theta = \theta - v_t \] (3.27)
where $\gamma$ is the momentum term, $v_{t-1}$ and $v_t$ are the previous and current updates to the parameters. The net effect of the momentum term is that the parameters with gradients pointing in the same direction are updated with higher values while the parameters with changing gradient directions are updated with smaller values resulting in faster convergence. We used a MBGD with momentum in Papers III and VI.

**Nesterov accelerated gradient**
The momentum-based algorithms result in faster convergence but at locations close to local minima the momentum term forces the update to go further over-shooting the local minimum. This can be corrected using the Nesterov accelerated gradient (NAG) [40].

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$  \hspace{1cm} (3.28)

Here, the parameters are not updated with respect to their current values but based on an approximate future position of the parameters, $\theta - \gamma v_{t-1}$.

**Adagrad**
Adaptive gradient [41] (Adagrad) extends the NAG and adapts the updates to each parameter depending on their importance. The Adagrad makes larger updates for less frequently occurring features and smaller updates for more frequent features. Each parameter, $\theta_i$, in Adagrad uses a different learning rate. The MBGD update for every parameter for time step $t$ becomes

$$\theta_{t+1,i} = \theta_t - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} \nabla_{\theta} J(\theta_{t,i})$$  \hspace{1cm} (3.29)

where $G_{t,i}$ is a diagonal matrix with each diagonal entry as the sum of the squares of the gradients of $\theta_i$ up to time step $t$ and $\epsilon$ is a small value to prevent division by zero. Thus Adagrad eliminates the need to tune the learning rate manually.

**RMSProp**
The squared gradient accumulates in the denominator of equation 3.29 of Adagrad and results in very small learning rates and stops the learning process. RMSProp [42] eliminates the problem of gradient accumulation. The RMSProp uses a running average of the squared gradient in a recursive manner. The average squared gradient, $E[g^2]_t$, for step $t$ is defined on the average at step $t - 1$ and the current gradient as

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$  \hspace{1cm} (3.30)

where $g_t = \nabla_{\theta} J(\theta_t)$. The parameters are updated according to

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$  \hspace{1cm} (3.31)
where $\varepsilon$ is a small value to prevent division by zero. We used RMSProp optimization to train our network in Paper V.

**Adam**

Adaptive momentum estimation [43] (Adam) also calculates adaptive learning rates for each parameter. Adam stores exponentially decaying averages [38] of past squared gradients, $v_t$, and past gradients, $m_t$ as

\[
\begin{align*}
    m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\
    v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2
\end{align*}
\]  

(3.32)

During the initial steps of the optimization, $m_t$ and $v_t$ are biased to zero since they are initialized to vectors of zeros. To compensate for this a bias correction is applied as

\[
\begin{align*}
    \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\
    \hat{v}_t &= \frac{v_t}{1 - \beta_2^t}
\end{align*}
\]  

(3.33)

The final parameter update is done as

\[
\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t
\]  

(3.34)

where $\varepsilon$ is a small value to prevent division by zero. Typical value for $\beta_1$ is 0.5 and $\beta_2$ is 0.99. We used Adam optimization to train our deep neural networks in Papers IV and VII.

So far we saw the basic building blocks of deep neural networks that were used in this thesis. Now, we discuss an advanced type of deep neural network known as generative adversarial network, used in Paper IV.

### 3.5.6 Generative adversarial networks

A generative adversarial network (GAN) [44] consists of a generator and an adversarial or discriminator network. The generator is fed with a noisy image created from a known noise distribution, $p_{\text{noise}}$. Then the generator network creates new data, $p_{\text{model}}$. The adversarial network is fed with images from a training dataset, $\hat{p}_{\text{data}}$, and the images created by the generator network. An adversarial loss is created based on how well the adversarial network is able to distinguish between the $\hat{p}_{\text{data}}$ and $p_{\text{model}}$. The adversarial loss is back-propagated to the generator network. As the training progresses the generator is able to create images similar to $\hat{p}_{\text{data}}$ and the adversarial network loses its ability to classify images. Typically, at such a stage the GAN training can be considered to have converged.
Segmentation adversarial networks
The generator network can be replaced with a segmentation network and can use adversarial loss to improve the segmentation result [45]. In such a case the segmentation network is trained using two loss functions: 1) an adversarial loss function, \( L_a \), and 2) a weighted spatial cross-entropy loss function (Section 3.3.7), \( L_{wsce} \). The \( L_a \) enables the network to learn global features that are not so obvious from a pixel wise loss such as \( L_{wsce} \). Typically, the \( L_a \) predicts the probability that the input image belongs to the class of \( \hat{p}_{data} \) (1) or \( p_{model} \) (0). So the adversarial loss function becomes

\[
L_a = \sum_{n=1}^{N} L_{bce}(a(I_n, T_n), 1) + L_{bce}(a(I_n, s(I_n)), 0) \quad (3.35)
\]

where \( L_{bce} \) is the binary cross-entropy loss (the cross-entropy loss for two output classes), \( a \) is the adversarial network, \( s \) is the segmentation network, \( I_n \) is the \( n^{th} \) input image, \( T_n \) the corresponding target image and \( N \) is the total number of images in the training set.

The segmentation network is back-propagated only when the input to the adversarial network is the result from the segmentation network. Therefore, the segmentation loss function becomes

\[
L_s = \sum_{n=1}^{N} L_{wsce}(s(I_n), T_n, W_n) - \lambda L_{bce}(a(I_n, s(I_n)), 0) \quad (3.36)
\]

where \( W_n \) is the weighted label corresponding the the \( n^{th} \) input, \( \lambda \) is a weight factor deciding the contribution of adversarial loss to the overall loss. Typically at the beginning of training, the segmentation results might be poor compared to the ground truth so the adversarial network can easily identify images from \( p_{model} \) and the gradient might be small for back propagation [44]. Instead of classifying the segmentation result being from \( p_{model} \) we change the loss function so that we check how well the images from the segmentation network are similar to the training set [45]. Hence, we change the Equation 3.36 to

\[
L_s = \sum_{n=1}^{N} L_{wsce}(s(I_n), T_n, W_n) + \lambda L_{bce}(a(I_n, s(I_n)), 1) \quad (3.37)
\]

Once the training is over, we use only the segmentation network for further segmentation. We used a segmentation adversarial network in Paper IV. In this work, we multiplied the input raw images with both the multichannel probability maps from the segmentation network and the corresponding one-hot ground truth images. Then we used these multichannel images as inputs to the adversarial network. A schematic diagram showing the training of a segmentation adversarial network is shown in Fig. 3.13.
3.6 Reflection of the deep neural network algorithms

In this chapter, I have summarized the deep neural network algorithms used in this thesis. There are many other methods, used today, to improve the performance of deep networks such as parameter initialization methods, hyper parameter tuning algorithms etc. Selection of a particular algorithm for a specific problem is highly domain dependent. A lot of trials are required to find the optimal model. Research in hyper parameter optimization may help in such cases even though the need of computational resources is very high for them. The field of deep neural networks is growing rapidly and new algorithms are being proposed every day. New algorithms are quickly replacing the existing algorithms, once they show improved performance. It is extremely likely that the algorithms discussed in this chapter may become obsolete in the future. An interested reader is advised to keep in pace with the latest developments. At the time of writing this thesis the tensor cores are replacing the vector cores in GPUs. In the future, dedicated deep neural network cores might arise and we may be able to implement algorithms and network architectures that are not feasible with the current hardware.

*Figure 3.13. A GAN training with segmentation and adversarial networks.*
4. Overview of papers and summary of contributions

This chapter summarizes the background, methods, results, and contributions presented in the appended publications. Each section summarizes one of the papers included in the thesis with focus on the main contributions.

4.1 Segmentation and Track-Analysis in Time-Lapse Imaging of Bacteria (Paper I)

4.1.1 Background

Insight into several complex biological functions can be obtained by the study of live prokaryotic cells like *E. coli* bacteria. These experiments often involve tracking of individual cells in time-lapse microscopy images. Cell tracking experiments typically involve segmentation of individual cells in the time-lapse images. After the segmentation, the individual cells are tracked throughout the sequence.

In this work, we introduce a novel cell segmentation algorithm, which is relatively accurate and fast, and can be used for routine lab experiments. We qualitatively and quantitatively compare the proposed method with two existing cell segmentation approaches. We perform cell tracking and automatically correct some segmentation errors after the tracking. We also validate the whole segmentation and track analysis pipeline for a biological experiment to quantitatively measure the growth rate subjected to various growth media.

4.1.2 Curvature based contrast enhancement

In most image processing pipelines, images are segmented based on the image intensity values. Image analysis algorithms using the intensity values give excellent results in many cases. However, if both foreground and background regions have similar intensities, raw image intensity based approaches perform poorly. In such cases a detailed understanding of the domain helps in solving such issues.

In this work, we propose to use the curvature of the image intensity landscape. We consider images as a surface in 3D with the vertical and horizontal pixels specifying the spatial locations and the pixel intensity as the height of
the landscape. The maximum and minimum curvatures at a point are known as principal curvatures. The principal curvatures are found using the eigen values of the Hessian matrix. In our dataset the E. coli cell regions are dark and the regions between them are bright. So the smallest eigen value between the cells is negative and is of relatively large magnitude. But the regions inside the cells change intensities slowly so both of the eigen values are usually positive and smaller in magnitude. Therefore the smallest eigen value image increases the contrast between the cells, while not enhancing intensity variations within the cells, functioning as an image pre-processing step.

4.1.3 Object segmentation

Our E. coli dataset consists of large numbers of tightly packed cells. Segmentation using methods such as watershed segmentation, on the normalized eigen images, usually give under-segmentations. It is often difficult to delineate individual cells, and a single threshold is not sufficient for segmenting all the individual cells. We use ten threshold levels around the mean intensity. After thresholding we assign a weight value to each segment, which is calculated using a combination of residual area ratio and convexity, serving as measures of how “bacteria like” each detected image segment is. The stack of all weighted image segments are maximum projected, keeping all unique segment labels in a preliminary segmentation mask. Segments with a “bacteria-likeness” less than 0.75 are discarded as debris. In Section 4.3, we discuss the drawbacks of introducing fixed requirements on object shape and size.

4.1.4 Object filtering

Object filtering removes the outliers that happened to be present after the object segmentation. The object filtering is based on the assumption that the raw intensities of the E. coli cells follow a Gaussian distribution. We calculate the median intensity of all segments in the raw input image. Then a normalized histogram is constructed from these median intensities and the histogram is interpolated using spline interpolation. From the interpolated distribution the peak is found and uses it as mean for the estimated distribution. Thereafter, the full width at half maximum is found with respect to the estimated mean, and is used as the estimated standard deviation.

After finding the estimated standard deviation, a threshold is set empirically at four standard deviations above the estimated standard deviation. Any segments with intensities outside this threshold are removed. This filtering strategy worked well for the phase contrast dataset but failed for the fluorescent dataset due to the large variation in the intensities of the cells.
4.1.5 Object smoothing

There are applications where the segmentation accuracy is more important than the speed of execution. In such cases we perform a smoothing of sharp corners in the segments. Here we smooth individual segments using morphological geodesic active contours, as described by Marquez-Neila et al., [13].

4.1.6 Cell tracking

After the *E. coli* cells are segmented from the time-lapse images we track individual cells over time to analyze the growth patterns. We use an iterative Viterbi algorithm [12] for tracking the cells. The cell-tracking algorithm can handle events like cell division, cell death, cells entering or leaving the field of view, over- and under-segmentations, missing cells in some frames etc. For tracking, we use Jaccard similarity index to find the cell migration when comparing the current frame with the next frame.

4.1.7 Cell track filtering

The tracking algorithm generates several cell tracks. Out of these there are some erroneous tracks created due to the noise and complexity of the dataset. For further analysis we select only those tracks that have cell divisions at the beginning and at the end of tracks ensuring that a full life-cycle is included. After selecting the tracks based on the cell division we filter the noisy tracks using two criteria. The first criterion is based on the area of the cells before and after the cell divisions, at both ends of the tracks. If the area of the current cell and the sum of the areas of the two daughter cells are within a tolerance of 1%, then it is retained.

The second criterion is based on the growth pattern of the cells. The cells grow nearly exponentially over time. Since the widths of the cells are nearly constant they increase their length over time. We take the logarithm of the major axes lengths of the segments over time, which is nearly linear in the log scale. Thereafter, we fit a line using RANSAC [46] on the log major axis lengths of the cells over the track to find the estimated growth. We find the absolute difference between the estimated growth lines and the actual measured lengths for all the selected tracks. Sudden deviations from the estimated growth indicate segmentation and/or tracking errors, and these tracks are filtered out.

4.1.8 Post-tracking segmentation correction

We found that some of the good quality tracks contain single frame errors in segmentation, which can be corrected using the cell tracks. To correct for the single frame errors we took the cell segments corresponding to the same track
in two neighboring frames. We found the average position, orientation and shape of the two segments. Average position and orientation were found as the mean of the corresponding parameters for the two segments. The average shape was found as the average of the signed distance transform for the two segments after keeping the centroids and orientations of both segments fixed.

4.1.9 Run time evaluation

We compared the proposed method with two other methods such as MicrobeTracker [47] and MAMLE [48]. Our method was 10 times faster than MicrobeTracker and 8 times faster than MAMLE on average for the four datasets we tested. We performed the comparison on a laptop with dual core Intel® Core™ i7 CPU running at 2.7 GHz with 16 Gb RAM on Windows 7 operating system.

4.1.10 Discussion of results and contributions

We quantitatively compared the segmentation performance of our proposed method with two other methods; MicrobeTracker and MAMLE on four datasets. Even though our methods perform very well on the four datasets as presented in Paper I, it is not free from limitations. Due to the limits in parameter selection for the “bacteria-likeness” criteria some cells were not detected or segmented poorly. Such an example is shown in Fig. 12 in Paper I.

The improperly segmented cells in such cases are either unusually large cells or irregularly shaped cells. To overcome this limitation we used deep neural networks for segmentation as presented in Paper III, discussed in Section 4.3. We applied the domain knowledge, we acquired through this work, in Paper III for improved performance of the deep neural networks. The segmentation correction after tracking resulted in improving the overall quality of tracks. Our data showed that the cells were growing faster in glucose than in glycerol, showing that the limitations of our segmentation approach discussed above are small enough to still enable quantification of this type growth measurements.

The main contribution of this work is the usage of curvature based contrast enhancement for segmentation of E. coli cells in a novel experimental setting. The presented work is being used by our collaborators in their experiments on a routine basis. We also provide the source code for the segmentation method as an open source python implementation [49].
4.2 Fast Adaptive Local Thresholding based on Ellipse Fit for Bacteria Segmentation (Paper II)

4.2.1 Background

Intensity based thresholding is a popular method used for segmentation. In this work, we use a fast adaptive thresholding method to segment E. coli cells. The proposed method works in quasi-linear time and uses object size based features for segmenting objects. We use the domain knowledge from Paper I and pre-process raw image data by curvature based contrast enhancement prior to segmentation. Therefore, the raw image intensity, in this case, corresponds to the lowest eigen value of the Hessian matrix.

4.2.2 Component tree based segmentation of E. Coli bacteria

We take the raw image and find the lowest eigen values of the Hessian matrix to get the contrast enhanced images for four datasets (same as in Paper I). The final segmentation result is obtained after three passes through the raw image. In the first pass, we sort the pixels in decreasing order and consider each pixel as a node in a component tree.

A priori knowledge of the data can be utilized for better segmentation of cells. Parameters like major and minor axis lengths, major and minor axis ratio, object intensity etc. can be fed to the algorithm. The algorithm can decide on an optimal parameter for each segment and can decide whether to merge nodes or not. After creating the component tree we pass through the image a second time to compute the optimal ellipse parameters for merging nodes. Sometimes the ellipse criterion for the parent node is lower than that for the child node. In such cases a third pass is made to merge the child node to a proper parent node. After the three passes through the image we get a segmented image.

4.2.3 Run time evaluation

We compared the proposed adaptive thresholding method with four other methods such as two variants of CBA as explained in Paper I, MicrobeTracker [47] and MAMLE [48]. The comparison was done on a laptop with Intel®Core™i7 laptop running at 2.7 GHz with 16Gb RAM on Windows 7 OS. The proposed method is faster by an order of magnitude than the second fastest method, CBA2, on average for the four datasets.

4.2.4 Discussion of results and contributions

We quantitatively evaluated the segmentation result for the proposed method on four datasets that we used in Paper I. In Paper I, we used a repeated thresh-
olding around a fixed size window and compute the residual area ratio and convexity to segment \textit{E. coli} cells. In this work, we extended this step to segment the images faster, in quasi-linear time. We found that the segmentation results were comparable to those from Paper I. In addition to the quantitative evaluation we also evaluated the stability of the proposed method with respect to two parameters, those being the ellipse fit threshold and the number of gray-levels analyzed. The results are shown in Fig. 5 in Paper II. We found that the segmentation results are stable within a wide range of values for both the parameters.

As discussed in Sections 4.1 and 4.3, we still fail at detecting objects of unexpected size and shape. However, the method as such is an important contribution to microscopy image analysis where automated thresholding is typically based on the assumption of bimodal histograms using, e.g., Otsu thresholding. The presented method is instead focused on size and shape. In addition to experimental evaluations the proposed method was made openly available as both CellProfiler and ImageJ plugins.

### 4.3 Feature Augmented Deep Neural Networks for Segmentation of Cells (Paper III)

#### 4.3.1 Background

Traditional image analysis tasks involve hand-crafted image features for segmentation. In deep learning based segmentation, the features are learned by the network, from the dataset. It has been shown already that for small datasets, hand-crafted features perform better, and for large datasets, deep learning based features outperform traditional features.

In this work, we combine the best of these two worlds by extracting hand-crafted features from the input images, utilizing the domain knowledge, and train deep neural networks on these multi-channel images containing both raw images and images with extracted features. We use the feature augmentation strategy to segment \textit{E. coli} cells and mouse mammary gland cells. We qualitatively and quantitatively evaluate the performance of deep networks for these two datasets. In addition, we also perform modality transfer learning by training a deep network to segment cells from a phase contrast dataset and used the trained network to segment cells from a fluorescence dataset.

#### 4.3.2 Deep neural network architecture

In this work, we used an architecture that was inspired by U-net [26]. We modified the original U-net architecture by reducing the number of filters. We reduced the number of feature maps to 1/32 of the size of the original architecture. We used weighted spatial cross-entropy as a loss function. The
The network was trained for 20000 iterations with a base learning rate of 0.01 and we reduced the learning rate to 1/10 of the current value after 5000 iterations. The network was optimized using stochastic gradient descent.

4.3.3 Feature augmentation

Our main contribution of this paper is that we combine raw data with feature augmentation as input to the U-net architecture. For the *E. coli* dataset we extracted the smallest eigen value of the Hessian and truncated SVD [50], and used them as additional features. For the mouse cells dataset we used wavelet features. First, we found the wavelet transform using Daubechies 4 [51] (Db4) wavelet to four decomposition levels and then reconstructed the image after removing the approximation coefficients. These extracted images were input to the network as augmentation, together with the raw images. In addition to the data augmentation we also experimented with extracted features alone such as truncated SVD, and eigen values of the Hessian.

4.3.4 Segmentation and post-processing

The deep network required an input image of size 540 × 540 and the output probability map was of size 356 × 356. To segment larger images, we mirrored the input image on all sides and performed an overlap-tiling so that the input image tiles overlap while the output image tiles do not overlap. After creating the probability maps for the *E. coli* dataset the probability maps were thresholded at a fixed value of 0.2, followed by hole filling and small object removal. For the mouse cells dataset the output probability maps were segmented using watershed segmentation using CellProfiler [52].

4.3.5 Run time evaluation

The proposed method took 0.91 s to segment an image of size 860 × 860 on a workstation with six core Intel®Core™i7 CPU running at 3.5 GHz with 32 Gb RAM equipped with a Nvidia Titan X GPU on Ubuntu 14.04 operating system.

4.3.6 Discussion of results and contributions

We qualitatively and quantitatively compared the segmentation results from the deep learning architectures with the CBA method in Paper I. We also compared different input combinations for the deep network such as raw, eigen, raw+eigen, and SVD for the *E. coli* dataset. We found that the proposed deep neural network with raw and eigen images as input performed better for the *E.
coli phase contrast dataset, see Table 2 in Paper III. Compared to feature-based segmentation approaches where objects of deviating size and shape are lost, such objects were correctly segmented here.

Even though we got better results for our proposed method using feature augmentation, it is not always easy to find suitable features for a particular task. In this work, we inferred the domain knowledge from our previous work in Paper I. From an application point of view the methodology becomes straightforward if we can design a deep network end-to-end. Designing the best deep network is a research subject itself. The problems involving deep networks can be solved in multiple ways, ours is one solution. Therefore, if we have extensive domain knowledge then it can be used to get better results by providing the deep network with our a priori knowledge. It is also worth noting that the augmentation allowed us to reduce the network size, simplifying computations and reducing computation time. If the domain knowledge is limited then it is better to explore other network configurations such as Residual connections [32] or novel network designs such as inception networks [53] or more complex loss functions such as adversarial losses as in Paper IV.

4.4 Spheroid Segmentation using Multiscale Deep Adversarial Networks (Paper IV)

4.4.1 Background

Spheroids are cell clusters grown artificially outside the body of organisms. Cells are cultured in micro-wells and supplied with necessary environment and medium for growth and cell division. After several hours the cells form micro clusters called spheroids. The name ‘Spheroid’ is attributed to the spherical shape associated with these cell colonies.

One of the important tasks associated with the spheroids is to identify their growth subjected to the treatment using different types of drugs. The spheroids vary in size, shape, and texture depending on the biology and the local environment. Segmentation of these wide varieties of spheroids is challenging. We trained deep neural networks for segmentation of spheroids using a bright-field dataset. We used a segmentation network to segment the spheroids and an adversarial network to create adversarial loss. The architectural details of the neural networks are mentioned below. We created both the segmentation and adversarial networks from scratch.

4.4.2 Segmentation network architecture

To segment the spheroids we used a fully convolutional neural network. Since the spheroids had different sizes and were sometimes imaged at different scales, we analyzed the spheroids at two scales, i.e., at 1/8 and 1/16 the size of in-
put images. We used very deep feature extraction layers called Deeppath 1 and 2 for this in Paper IV. To preserve the spatial structures even after down-sampling we used dilated convolutions [24]. The first convolution layer created 60 feature maps. We selected the feature map size considering two factors: (1) extract as many features as possible so that the network can learn to detect many different types of textures, (2) the whole network has to fit in a GPU of 12Gb memory. The segmentation network architecture is shown in Fig. 4.1. We used weighted spatial cross-entropy, as explained in Section 3.3.7, as the loss function.

![Segmentation net](image)

*Figure 4.1. Segmentation network architecture.*

4.4.3 Adversarial network

An adversarial network discriminates between the ground truth (real data) and the output from a segmentation network (fake data). The output prediction of the adversarial network is used to create the adversarial loss. The adversarial network predicts the fake input as real data if the overall appearance of the input data is similar to the ground truth. During the start of the training, the adversarial loss for fake data might be large and as the training progresses the loss decreases.

We designed the adversarial network as a fully convolutional network. The network architecture is shown in Fig. 4.2. The network is constructed to have a dilated convolution with a kernel of size $3 \times 3$ with a stride of $2 \times 2$ and a dilation factor of $2 \times 2$ followed by a normal convolution with a kernel of size $1 \times 1$ and a stride of $1 \times 1$. This is similar to the fire module in the SqueezeNet network [54]. During the backward pass we inverted the gradient output, multiplied with a constant value $\lambda < 1$ and added it along with the
weighted spatial cross-entropy loss of the segmentation network. The details of the building blocks of the network can be found in the appended Paper IV.

![Adversarial net](image)

*Figure 4.2. Adversarial network architecture.*

4.4.4 Execution time

We trained the different network models on a workstation with a six core Intel(R) Core(TM) i7-5930K CPU running at 3.50GHz and 32Gb RAM and a Nvidia Titan X Pascal GPU with 12Gb GPU memory on Ubuntu 14.04 operating system. Training each network model took nearly one hour. The network was able to create probability maps at approximately 5 images per second, for input images of sizes $480 \times 480$.

4.4.5 Discussion of results and contributions

We compared the segmentation performance of the deep adversarial network with two other networks. The first network had same architecture as the segmentation network without adversarial loss. The second network that we compared with was U-net [26]. We performed five fold cross-validation of three models on two datasets. We performed all the hyper parameter tuning on the validation set. The final comparison was done on unseen test sets for both the datasets. The experimental results showed that the network with adversarial loss was the most stable network. Even though the adversarial network performed better we believe that further detailed studies are required to fully utilize the adversarial loss.

The main contribution of this work is the novel deep adversarial network for segmentation of spheroids. The generative adversarial networks are known for
their difficulty in training due to problems like mode collapse. The mode collapse [55] means the generator network captures only a few modes of the data generating distribution and discards other modes. This can make the generator and adversarial networks to switch between modes and fail to capture the true distribution. The linear increase in the adversarial loss used in this work is a novel strategy for training adversarial networks.

4.5 Automated Training of Deep Convolutional Neural Networks for Cell Segmentation (Paper V)

4.5.1 Background

Deep neural networks are popularly used for cell segmentation tasks. Usually for training the deep networks, a large amount of good quality annotated images are required. The annotations are created manually and it is a time consuming expensive process. Typically, the annotated samples for one experiment may not be suitable for another experiment resulting in new manual annotations for each experiment. Even within the experiments the morphology of controls and treatment groups vary widely. Therefore, separate annotations are required for controls and treatments. In short, creating ground truth manually is infeasible in many life science experiments. Here, we use an image analysis pipeline to create ground truths automatically with minimum disturbance to the biological experiment.

4.5.2 Automatic training set generation

We used two datasets for this experiment: (1) a time-lapse bright-field dataset and (2) a multi-channel fluorescent dataset. For the time-lapse dataset, multifocal bright-field images were acquired at regular intervals throughout the experiment. After the experiment was finished, we stained the cells with both nuclear and cytoplasm stains and acquired corresponding fluorescent images. In addition to the fluorescent images we acquired a bright-field image of the same field of view. We used these fluorescent and the last bright-field images to create the training data. As compared to bright-field data, specific fluorescent markers of the nuclear and cytoplasm regions are easily delineated using standard image analysis steps available in CellProfiler [52]. Once the cells were segmented from the fluorescent images they were used as labels and the corresponding bright-field images as the data for training. Similarly, we created ground truths for three micro-wells for training and evaluation.

For the multi-channel fluorescent dataset we used the openly available image set BBBC022v1 [3], available from the Broad Bioimage Benchmark Collection [4]. Different subcellular structures such as the nucleus, endoplasmic reticulum, nucleoli, golgi apparatus, plasma membrane, and mitochondria
were fluorescently stained and corresponding images were acquired. To create the ground truth we segmented the nuclear region from the corresponding fluorescent image. These segmented nuclear regions were thereafter used as labels and the rest of the image channels were used as data for training the deep neural network.

In addition to the automatically created ground truths we also created one manual annotation per well corresponding to a single site, out of the nine sites, for evaluating the final segmentation performance for the time-lapse dataset. This manual dataset was not used for training the network. The automatic ground truth corresponding to the selected site was also excluded from the training set. For the multi-channel fluorescent dataset we did not create manual ground truth, but the evaluation was done only on one of the automatic ground truths.

4.5.3 Segmentation of cells

We preprocessed the input raw images for illumination correction, cropped non-overlapping tiles of size $240 \times 240$ and created probability maps using the trained neural networks. This created artifacts at the boundaries of the tiles. To remove the artifacts we mirrored the raw images on all the four sides and passed through the mirrored image after shifting $120 \times 120$ pixels in vertical and horizontal directions corresponding to the raw image. The probability map after the second pass was cropped and added to the first probability map after a Gaussian weighting.

The Gaussian weighting removed all the tiling artifacts. Thus a final probability map for the raw image of size $2034 \times 2034$ was obtained. The crop-wise probability map creation and removing the tiling artifacts is shown in Fig. 4.3. This was followed by a seeded watershed segmentation to get the final segmentation masks. This process was repeated for all the images in the time-lapse dataset.
sequence to obtain the segmentation result for the whole sequence. The entire image preprocessing, probability map creation using the trained network and the final segmentation were done as an open source CellProfiler plugin. A similar image processing pipeline was used to segment the multi-channel fluorescent dataset also.

**Execution time**

We used the open source deep learning tool Caffe [56] for training and evaluation of our deep network. We performed all the experiments on a workstation with Intel® Core™ i7 CPU running at 3.5GHz and 32Gb RAM equipped with Nvidia Titan X Pascal GPU on Ubuntu 14.04 operating system. The training for DCNN took nearly 5.5 hours and it took six seconds to segment an image of size $2034 \times 2034$ including pre- and postprocessing.

4.5.4 Discussion of results and contributions

We quantitatively evaluated the segmentation results for the time-lapse dataset and the multi-channel fluorescent dataset. We found that the segmentation results from the deep network were similar to manual annotations showing the efficacy of automated training.

The main contribution of this work is that we can plug the additional steps to the biological experiment without disturbing the existing system. This facilitates the adoption of this novel automated training feasible in a laboratory setting. The elimination of manual annotations saves several hours of manual labor. We propose a novel deep neural network architecture for segmentation of cells from different modalities such as bright-field and fluorescent images. The different image analysis pipelines for training the neural network and segmentation of cells using the trained neural network are available as open source plugins for CellProfiler.

4.6 Deep Fish: Deep Learning-Based Classification of Zebrafish Deformation for High-Throughput Screening (Paper VI)

4.6.1 Background

Biological experiments typically involve large-scale analysis of model organisms such as zebrafish. The large scale experiments require high-content/high-throughput screenings. Automated image analysis can usually be employed in such scenarios to get unbiased quantitative results. Designing image analysis pipelines for such tasks is non-trivial.

In this work, we performed an automated classification of zebrafish phenotypes as affected or non-affected by treatment with a particular drug. Here,
we trained a deep neural network classifier to discriminate between the phenotypes. Furthermore, we performed ablation studies and verified that the network learned from the foreground regions. We also realized that the network learned from the head region rather than more visually apparent tail deformations.

4.6.2 Dataset
The dataset consisted of zebrafish embryos grown in microwell plates. There were five zebrafish per well oriented randomly. There were 79 images of treated fish and 60 images of untreated fish. We cropped all images to a size of $512 \times 512$. We partitioned the data to five folds; three folds were used for training, one fold for validation and one for testing. We kept the number of treated and untreated samples constant for all the folds. After partitioning we increased the dataset size by data augmentation by flipping and rotations at multiples of 90 degrees for the corresponding training and validation sets. No data augmentation was performed on the test set.

4.6.3 Deep neural network architecture and training
We chose a deep network structure similar to the Alexnet [20] architecture for this work. There were five convolution layers followed by three fully connected layers. We trained the network using binary cross-entropy loss function. The network was optimized using mini batch stochastic gradient descent with momentum and a mini batch size of 50. We used Caffe [56] for training the deep network.

4.6.4 Ablation studies
After training the deep network we performed ablation studies to verify that the network learns from foreground regions rather than background regions. We performed three ablation studies. In the first experiment, we removed all the fish from the images and replaced the pixel values with the immediate background intensities. In the second and third experiments, we removed the head and tail regions of the fish, respectively and performed classification using the trained network.

4.6.5 Discussion of results and contributions
The classification performance of the five fold deep networks was tested on the independent test set. The average accuracy was 92.8%, average recall was 89.8%, average precision was 93.4%, and average F-score was 91.5%. The
drop in accuracy was mainly attributed to the presence of dead fish in the test samples. The network could be trained in less than 30 minutes. The network took 0.36 s to classify 28 test images.

An example of dead fish, which started disintegrating, can be seen in Fig. 3A, eighth column, in Paper VI. In the first ablation study, we found that the network actually learned from the foreground region. This is evident from the drop in deformation probability for the images without fish as shown in Fig. 3A second row in Paper VI. Next, for the experiment without head regions of fish the deformation probabilities were less than 50% in all the cases, indicating that features from head regions were important for classification. When the tail regions were removed, the deformation probabilities increased again indicating that features from head regions contribute to classification.

Thus the ablation studies not only confirmed that the network learned from foreground regions but also helped in localizing important regions for classification. A better understanding of the features extracted by the network was possible by different ablation studies. The proposed ablation study can be easily done on any classifier and can be used to get an approximate idea of the spatial locations of relevant features. The sub-second execution speed of the network opens the possibility of using the proposed classifier for high-throughput screenings. From the knowledge of other works with deep learning such as Papers III, IV and V, it is possible to experiment with different network architectures that are less computationally intensive and at the same time offer better performance than the proposed method, such as Googlenet [35] or ResNet [32].

4.7 Deep Convolutional Neural Networks for Detecting Cellular Changes Due to Malignancy (Paper VII)

4.7.1 Background

Mortality due to cancer is one of the main causes of death worldwide with nearly 8.8 million deaths in 2015 [57]. Previous studies showed that a large portion of the cancer mortalities could be prevented with proper screening programs. Screening of large populations is tedious, time consuming, expensive, and require large numbers of domain experts. Therefore, automated systems for screening are required to carry out the population screening programs. In this work, we created an automated system using deep convolutional neural networks to classify cell images, as benign or malignant, from oral and cervical cell samples. We also classified potential malignancy associated changes (MAC) from benign cases for oral cell samples.
4.7.2 Datasets
The dataset consists of images acquired from oral cell samples and cervical cell samples. The oral dataset was divided into two separate groups. Oral dataset 1 consisted of samples from benign cases and malignant cases. Oral dataset 2 consisted of samples from benign cases and non-tumor side of the oral cavity of malignant cases. The individual cells in the oral dataset were not manually annotated. Thus, all cells from the malignant samples were assumed to be influenced by the adjacent malignancy causing malignancy associated changes that we tried to detect even though they were not visually prominent. The benign cases were assumed to have no such changes. For the cervical dataset all individual cells were manually classified as benign or malignant by an expert cytologist.

4.7.3 Deep convolutional neural networks
The convolutional neural networks used in this study were inspired by VGG [58] and Resnet [32]. We used batch normalization for regularization and ReLU for activation for both the architectures. All the networks were trained using binary cross-entropy loss and optimized using Adam optimization [43] with a learning rate of 0.01. The networks were trained for 40 epochs.

4.7.4 Discussion of results and contributions
A five fold cross-validation of the cervical dataset gave accuracy around 86% for both VGG and Resnet. A three fold cross-validation of the oral dataset 1 gave an accuracy of 80% for VGG and 78% for Resnet. Two fold cross-validation of oral dataset 2 gave an accuracy of 80% for VGG and 82% for Resnet.

The details regarding the performance of all classifiers can be found in Fig. 7, 8, 9, and 10 in Paper VII. The patientwise cross-validation used in this paper showed that the network was not biased to irrelevant features such as the background or stain intensities. This study showed that an already existing deep neural network could be used for solving an important classification problem in microscopy. The preliminary studies for screening precancerous lesions in this work have the potential application for population screening. It is worth noting that validation of the performance of the classifier on a larger dataset is required and is planned as a future work. All the networks were trained on a modern GPU facilitating the possibility of high throughput screening of whole slide images. From the experience gained from the previous works, it is possible to use latest network architectures such as inception-v4 [53] for improved performance or Squeeznet [54] for faster execution speeds.
5. Conclusion and future perspectives

This thesis focused on different image analysis methods to quantify microscopy image data of biological samples. The algorithms ranged from traditional image analysis techniques to deep neural network based modern image analysis methods. The biological samples varied from prokaryotic *E. coli* cells to model organisms such as zebrafish. All the method developments were aimed at quantifying the image data for deeper understanding of diseases like cancer or high-throughput drug screening experiments.

5.1 Conclusion

In Papers I and II, we developed methods for segmentation of *E. coli* cell colonies using traditional image analysis methods. Our proposed methods were relatively faster than previously existing methods for analyzing these types of images. Our methods provided a similar or better performance than the other compared methods. Because of the fast execution speeds the image analysis operations were finished by the time the image dataset was acquired, facilitating quick results for the biological experiments. Our proposed methods are fairly robust and are used by our collaborators on a routine basis in their labs.

The traditional image analysis pipelines used in Papers I and II required tuning of parameters for specific experiments and due to the limitations in parameter setting some rare events could not be detected using these approaches. In Paper III, we overcame these limitations by training deep neural network based algorithms on our dataset. The deep neural networks learned the parameters directly from the data and were able to detect rare events as well. In this work, we combined our domain knowledge in traditional image analysis, gained in Papers I and II, with modern deep neural networks to give better performance.

In Paper IV, we analyzed bright-field microscopy images of spheroid cell clusters of different size, texture, and illumination. To solve this problem we created a novel deep neural network architecture based on generative adversarial networks to get better performance than two other compared deep neural network architectures. The deep neural network algorithms were created for running in GPUs facilitating high-throughput experiments. These algorithms are planned to be used in our collaborator’s lab, showing the efficacy of our methods.
One of the difficulties in adopting deep neural networks for laboratory experiments is the requirement to create several good quality annotated samples for training the neural networks. This greatly limited the usability of the deep networks in biological experiments. We tried to address this issue in Paper V. In this work, we created annotations automatically using biological clues without major changes to the experimental protocols. We acquired time-lapse images throughout the period of the experiment and stained the samples at the end of the specific experiment. Thereafter, we applied traditional image analysis methods using openly available CellProfiler software to generate annotations. The creation of automated annotations in this case is domain specific and differs with experiments. Once the annotations were made we trained the deep neural networks to segment the cells. We also created an open-source plugin to use the trained neural networks in the CellProfiler so that biologists could use them readily.

In Paper VI, we addressed a classification problem of categorizing zebrafish embryos. A previous work on classifying different phenotypes of zebrafish was done using traditional image analysis methods. This approach required tuning of many parameters that depended on the data. In the proposed work, we used a popular deep neural network for classification known as Alexnet, for classifying the zebrafish. The performance of the trained neural network was comparable to that of the previously existing method. Typically, a common criticism faced by deep neural network based methods was their bias towards the training data and learning uninformative features from background rather than the foreground region, in this case zebrafish. We performed several ablation studies and verified that the network actually learned from the zebrafish and not from background intensities. We believe that simple ablation studies are helpful in understanding the features learned by the deep neural networks, rather than considering them as ‘black-box’ methods.

In Paper VII, we classified cervical and oral cells to benign and malignant categories using deep neural networks. In this work, we studied the possibility of detecting the very subtle malignancy associated changes that have been shown to sometimes be present in the vicinity of precancerous lesions. These changes were difficult to detect with human eyes and they were similar to benign cases. The preliminary results showed that it was possible to detect those changes. We performed patient-wise cross validation to verify that the detected malignancy associated changes were not due to variations in stain intensities of different samples.

All the specific project aims were achieved to a satisfactory level in this work and resulted in the appended Papers. In addition to the Papers, the algorithms developed are either being used or planned to be used in our collaborators labs, showing the usefulness of our methods. The latest imaging and computational facilities enabled us to acquire large amounts of datasets. Such high-content/high-throughput experiments pave the way to explore future biology and helps in deeper understanding of organisms. The sizes of these
datasets are so large that it is impractical to solve such problems using only traditional image analysis methods. In this thesis, I showed that it is possible to solve two important problems in image analysis like segmentation and classification, using deep neural networks. I believe that algorithms using deep neural networks have the potential to solve many more interesting problems in biology and serving mankind in general.

5.2 Future perspectives

Application of deep neural networks for image analysis is a recent trend. The latest developments in this field show that this trend is likely to continue for the coming years. In this thesis, I showed different deep neural networks for microscopy image analysis. The design of deep neural networks for a specific problem is domain dependent and requires the tuning of several hyper-parameters. In this work, I closely followed the developments in the field of deep neural networks and modified our network architectures for improved performance. A lot of algorithmic changes are proposed on a daily basis and in the future I think that this will saturate to a general network architecture. In the future, I would like to understand more about the features learned by the deep networks by analyzing them in better ways. Latest studies on visualizing the features are promising in this direction.

In the future, I would like to explore the possibility of hyper-parameter optimization so that the deep neural networks will find the optimum network architectures. I would also like to validate our results on larger datasets, especially for the oral datasets. I would like to create a software tool for population screening that will be used by medical practitioners in their clinics. In the future, I would also like to combine traditional image analysis and deep neural network based image analysis for segmentation of spheroid clusters to get better performance.

The study of biology is evolving fast with several new types of microscopy being introduced. There are microscopes capable of imaging individual cells of a whole organism in 3D. Another interesting application is to find all the neuronal connections in the brain of organisms. Such datasets are extremely large and require large computational resources to process the data. Deep neural network based algorithms become the natural choices to analyze such datasets, overcoming the limitations of traditional hand-crafted algorithms. I believe that the future of image analysis using deep neural networks is brighter than ever.


Sammanfattningsvis visar avhandlingen att djupa neurala nätverk, i kombination med klassiska bildanalysmetoder, har stor potential för analys av mikroskopibilder där målet är att segmentera och klassificera objekt för att upptäcka och utvärdera förändringar i biologiska prover.
Acknowledgments

This thesis is the result of four years of my work at the center for image analysis. This journey would not be possible without the help and support of many people. I would like to thank the support, encouragement and care of my supervisors, colleagues, friends and family. My special thanks to

- My main supervisor, Carolina Wählby, for your support and encouragements. I appreciate your high level perspectives of research, your deeper understanding of biology and algorithms. I always wondered about your time management. I learned a lot from your prompt feedbacks on my manuscripts. Your positive criticisms molded me to approach research in a different perspective.
- My co-supervisor, Petter Ranefjäll, for giving valuable suggestions when I am stuck. Your hints and clues helped me overcome several hurdles in my research. I appreciate your exceptional patience in explaining things, even though I make the same mistakes again.
- My co-supervisor, Ewert Bengtsson, for sharing your vast knowledge in the field of image analysis. I always wondered about your knowledge in several fields of science not only on a higher level but also on very detailed algorithm level. I also would like to thank you for your nice discussions during your visit to India when I was working at CDAC.
- The Vi2 division head, Ingela Nyström, for your support and care during tough times of my life. I learned a lot from you on how to be a good boss.
- Lena Nordström, for quickly solving any problems that I have.
- Anna-Lena Forsberg, for helping me find a place in Uppsala, when I arrived here. I also would like to thank you for sorting out various administrative issues I had.
- Camilla Pajunen, for always being cheerful and helping me.
- My former colleague and friend Omer Ishaq for your company, great discussions, and ideas. Above all, being very supportive to me at needed times.
- Special thanks to our ‘fika group’, consisting of Kalyan Ram, Tomas Wilkinson, Fredrik Wahlberg, Omer Ishaq and Liang Dai, for interesting and in-depth discussions on deep learning.
- Kristína Lidayová, for your valuable tips and help in thesis writing and interesting discussions.
- My office mates in the ‘big room’, Thomas Lind, Damian Matuszewski, Fei Liu, Teo Asplund and Azadeh Fakhrzadeh for your company.
• My friend and colleague, Amit Suveer, for your nice discussions and help you gave me during needed times.
• My office mate Christer Kiselman. You are such a wonderful person.
• My colleagues and friends at the division of visual information and interaction, Alexandra Pacureanu, Vladimir Curic, Pontus Olsson, Maxime Bombrun, Amin Allalou, Anindya Gupta, Leslie Solorzano, Gabriel Partel, Eva Breznik, Alex Yuan Gao, Johan Öfverstedt, Fredrik Nysjö, Johan Nysjö and Ekta Vats.
• Seniors, Gunilla Borgefors, Anders Brun, Anders Hast, Robin Strand, and Filip Malmberg for your interesting insights into the world of image analysis.
• Nataša Sladoje, Maxime Bombrun, Ida-Maria Sintorn, Omer Ishaq, Ewert Bengtsson, Carolina Wahlby, Petter Ranefall, Ingela Nyström, and Joakim Lindblad for your comments and feedbacks on my thesis.
• My collaborators, Christina Runow Stark, Sylvie Le Guyader, Johan Elf, Johan Carlsson, and K. Sujathan, for your feedback, support and collaboration. I am lucky to have you as my collaborators.
• IT technical staff, Henrik Hedlund, Astrid Raidl and Joakim Sjöåker, for your timely help and quick responses on any issues.
• My master thesis supervisor, C. Shummuva Velayutham, who advised and motivated me to do Ph.D. My master thesis co-supervisor, D. Venkataraman, and all faculty members at the Department of Computer Science, Amritaviswavidyapeetham University, Coimbatore, India, for your encouragement and help during my studies.
• My master thesis co-supervisor, K. Sujathan, Nimi Girija, and all staff members at the department of Pathology, Regional Cancer Center, Thiruvananthapuram, India, for your support, help and supervision.
• My former colleague and friend Issac Niwas, for your motivations and helping me with everything.
• Sundararaman Gopalan, the chairperson, and all my former colleagues at the Department of Electronics and Communication, Amritaviswavidyapeetham University, Amritapuri, for your support and encouragements.
• Sudhamony S., my former section head, and all my former colleagues at Medical informatics, CDAC, Thiruvananthapuram.
• All Malayalee friends in Uppsala for your company.
• My wife Nisha, for your love and support. You taught me what is important in life.
• My daughter Navaneetha, for making my life happy everyday.
• My sister Sanooja, for being the nicest sister one could wish for.
• My mother Lalitha and father Sadanandan. Thank you for your belief, inspiration and great support. All my achievements are due to you.
References


Errata

In Paper I, page 3, Fig. 3 text written as “c) Plot showing pixel values from the same row from (a) (red line), and (b) (blue line).” should be replaced with “c) Plot showing pixel values from the same row from (a) (blue line), and (b) (red line).”

In Paper VI, page 2, Fig. 1, the scale bar is missing. Refer Fig. 2.1 for scale information.
Acta Universitatis Upsaliensis

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 1566

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title “Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology”.)

Distribution: publications.uu.se
urn:nbn:se:uu:diva-329834