ŽILINSKÁ UNIVERZITA IN ŽILINA
Faculty of Management Science and Informatics

Automated video processing for development and verification of computational models of biological cells

Dissertation Thesis

Žilina, 2020 **Ing. František Kajánek**

ŽILINSKÁ UNIVERZITA IN ŽILINA

Faculty of Management Science and Informatics

Automated video processing for development and verification of computational models of biological cells

Dissertation Thesis

28360020203002

Study Programme: Applied Informatics
Field of Study: Informatics

Department: Department of Software Technologies
Faculty of Management Science and Informatics
Žilinská Univerzita in Žilina

Supervisor: prof. Mgr. Ivan Cimrák, Dr.
Associate supervisor: Ing. Peter Tarábek, PhD.

Žilina, 2020

Ing. František Kajánek

Annotation

Type of thesis	: Dissertation Thesis
Academic Year	: 2019/2020
Name of Thesis	: Automated video processing for development and verification of computational models of biological cells
Author	: Ing. František Kajánek
Supervisor	: prof. Mgr. Ivan Cimrák, Dr.
Associate Supervisor	: Ing. Peter Tarábek, PhD.
Language:	: English
Number of Pages	: 69
Number of Images	: 19
Number of Tables	: 8
Number of References	: 49
Keywords	: Red Blood Cell, Machine Learning, Detection, Tracking, Data gathering, Neural Networks

Acknowledgements

I would like to thank prof. Mgr. Ivan Cimrák, Dr. and Ing. Peter Tarábek, PhD. for their infinite patience when being tasked with helping me with this immense topic. I would also like to extend gratitude to the whole Cell-In-Fluid team who were always supportive even when there seemed to be no light at the end of the tunnel. Big thanks also goes to Dmitri for programming support when in need and to Anne when two equal heads were needed.

Abstract

KAJÁNEK, František: *Automated video processing for development and verification of computational models of biological cells.* [Dissertation thesis] University of Žilina. Faculty of Management Science and Informatics. Department of Software Technology. - Supervisor: prof. Mgr. Ivan Cimrák, Dr., Žilina: FRI ZU, 69 p

This dissertation thesis deals with the topic of automated image processing for the purpose of automated data gathering for validation of simulations of microfluidic devices. The goals of this thesis are to evaluate existing methods for detection and tracking of red blood cells and propose improvements to both problems. Detection of objects is a topic of great interest and analysing state-of-the-art approaches and applying it to red blood cells shows promise for improving performance. Tracking on the other hand provides wider possibilities for experimentation using convolutional neural networks. The thesis also proposes several metrics for utilizing gathered data for immediate validation of simulations.

Keywords: Red Blood Cell, Machine Learning, Detection, Tracking, Data gathering, Neural Networks

Abstrakt

KAJÁNEK, František: *Počítačové spracovanie obrazu pre vývoj a verifikáciu výpočtových modelov biologických buniek.* [Dizertačná práca] Žilinská Univerzita v Žiline. Fakulta riadenia a informatiky. Katedra softvérových technológií. - Vedúci dizertačnej práce: prof. Mgr. Ivan Cimrák, Dr., Žilina: FRI ZU, 69 p

Táto dizertačná práca sa zaoberá automatizovaným zberom dát z videí za účelom validácie simulácií mikrofluidických zariadení. Cieľom tejto práce je prejsť existujúce metódy na detekciu a trasovanie červených krviniek a navrhnúť nové vylepšené metódy. Detekcia objektov je širokou témou o ktorú je veľký záujem. Analýza najmodernejších prístupov a aplikácia na túto úlohu je dobrým spôsobom na zlepšenie detekcie. Trasovanie na druhej strane umožňuje viac možností na experimentáciu pomocou konvolučných neurónových sietí. Práca taktiež navrhuje niekoľko spôsobov na vyhodnotenie simulácií pomocou získaných dát.

Kľúčové slová: Červené krvinky, Strojové učenie, Detekcia, Trasovanie, Zber dát, Neurónové siete

Contents

List of Figures

List of Tables

Introduction

When modeling blood flow in artificial microfluidic devices, the exploration of behaviour of different cells, as well as understanding the physical properties of such a device is necessary. For this purpose a lot of data is required for both design and testing of such a model. Due to this, the data gathering process is very important and needs to be as efficient as possible. In today's age this means automated processing on computers.

Most common experiment data is available in video or image form. As a result advanced computer vision techniques are required, to process high volumes of data. Different algorithms already exist in computer vision, and utilizing existing algorithms or designing own algorithms is difficult. For our purpose, we need to design system capable of analysing video data, and through cell detection and cell tracking acquire data for model validation.

In this thesis we will be dealing with the problems posed by automated image processing. Firstly, we will provide a quick overview of existing methods. Next, we will evaluate existing methods in order to provide a baseline and we will introduce our dataset. After that we will delve deeper into detection of Red Blood Cells and our solutions for the problem. After elaborating on detection, we will go over our existing tracking algorithms and our proposed method for improved results. We will conclude with a short discussion about current problems and future possibilities.

1 Data Extraction

Modeling of blood flow in microfludic devices is a way of approaching tasks, which are hard to solve using other means, for example conducting real world biological experiments. Computational modelling helps to predict outcomes in circumstances that are difficult to achieve in laboratory [5]. It is an effective tool in optimization and design [2, 1]. In order for simulations to provide valid results, fidelity and quality of the simulation are critical for modeling elasticity, interaction and motion of red blood cells (RBC). This is due to, e.g. hematocrit of blood being very high (45%), which means that realistic modeling of RBCs is key for further improvement. RBC models have been used to model processes inside microfluidic devices [4, 7]. The underlying model for cell's membrane is built upon the knowledge about real behaviour of cells in biological experiments. To validate the models, experiments with single cells may be used to assess the biomechanics of individual cells, such as in [8], where the stretching of individual cells is performed with optical tweezers. To validate macroscopic phenomena such as cell-free layer, experiments with many cells can be used. For reference see [9] and references therein. In the latter case, data extracted from video sequences are crucial for the process of validating a model.

With the increase in available data, and with the increased need for automated processing in many scientific areas, and not just for the purpose of validating simulations. Scientists commonly gather videos for further examination as well as for other parties to use during simulation configuration and validation. Our interest lies in cell experiments in general, or specifically red blood cell experiments. Validation can be performed after acquiring data from a video. Video footage can be either processed manually, which is feasible for a very small amount of footage, or automatically, by extracting key information important for further experimentation. Example of automated processing can be cell detection and cell tracking.

Cell tracking can be divided into two types of methods. The first type is contour matching methods, which segment cells in the first image and then continue to evolve their contours in subsequent images/frames.[21][22] This solves both the segmentation problem and the tracking problem simultaneously. Second group can

be characterised by first detecting all cells and subsequently tracking found cells through images/frames through temporal information.[23][24][25] In this thesis we will be mainly exploring various augmentations and improvements to the second group.

1.1 Object Detection

Cell detection or rather the general problem of object detection is a common task with many applications not just related to tracking. Even though this task has had considerable advancements in recent years, it is still considered to be a challenging task due to changes in illumination, partial occlusion, object overlap or object rotation. One approach to object detection is through shape-based methods, like Hough-transform [10][11]. This method in particular expects certain geometric shapes, such as lines, circles or ellipses, and is then able to identify these shapes in images. Despite the lower success rate of such methods, the big advantage is that the method can be used without any machine learning, and can potentially be used for automated data gathering.

Most state-of-the-art approaches rely on hand-crafted feature extraction (local binary patterns [34], Haar-like features [20], histograms [35], HOG descriptors [12]). These however, are not very well suited for all general objects, and require a lot of testing to determine, which feature extractor is most suited for the task. In addition to that, these descriptors are generally used in connection with machine learning algorithms. This means, that they require a training dataset and additional tweaking for actual use.

Neural networks present a change in the common paradigm of using hand-picked custom feature extraction for each task. Neural networks directly learn features from raw data, which resulted in state-of-the-art results in complicated tasks, such as image classification [36], object recognition [37], detection and segmentation [38]. Neural networks perform exceedingly well on tasks, which are able to supply a great amount of training data. Object detection specifically utilizes convolutional layers in order to provide the highest quality of detection. In recent years, convolutional neural networks are widely used and significant improvements have been made to the use and propagation of information, which greatly improved

their usability. Such examples are [40][39]

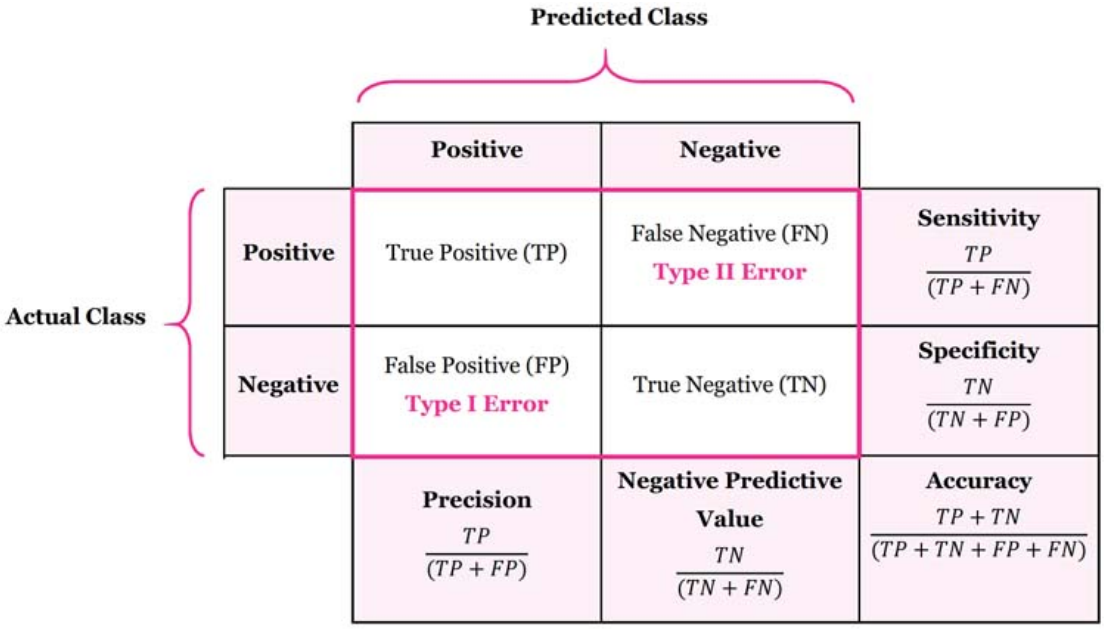


Figure 1: Example of first video. Source: [49]

Detection results can be interpreted through a confusion matrix (Figure 1). In turn, this information is very useful in computing the Precision/Recall metric. Precision is the percentage of valid detections in a given image and can be calculated as true positives/(true positives + false positives). Recall on the other hand is the percentage of all objects found within a given image and can be calculated as true positives/(true positives + false negatives).

1.2 Object tracking

The idea of general object tracking with input taken from object detection is very simple: connect together all bounding boxes found by the detection step so that they form one consecutive track. This track can then be used for various purposes, like computing average velocity, showing the most common trajectory, highlighting object characteristics and changes in shape, size, etc.

A very common approach is to utilize object velocity in order to estimate the location of a given object in the next frame of a video. Such an approach is

described in [26]. It utilizes a keyhole searching mechanism which gives estimates on where to look for the next bounding box.

Another approach utilizes particle filtering. Particle filtering is a method used for estimating the internal states of dynamical systems when partial observations are made and deviations are present in the system. In object tracking this translates to objects being close each other, potentially overlapping or crossing their paths, and connecting objects into tracks with the highest likelihood. Such an approach is described in [27].

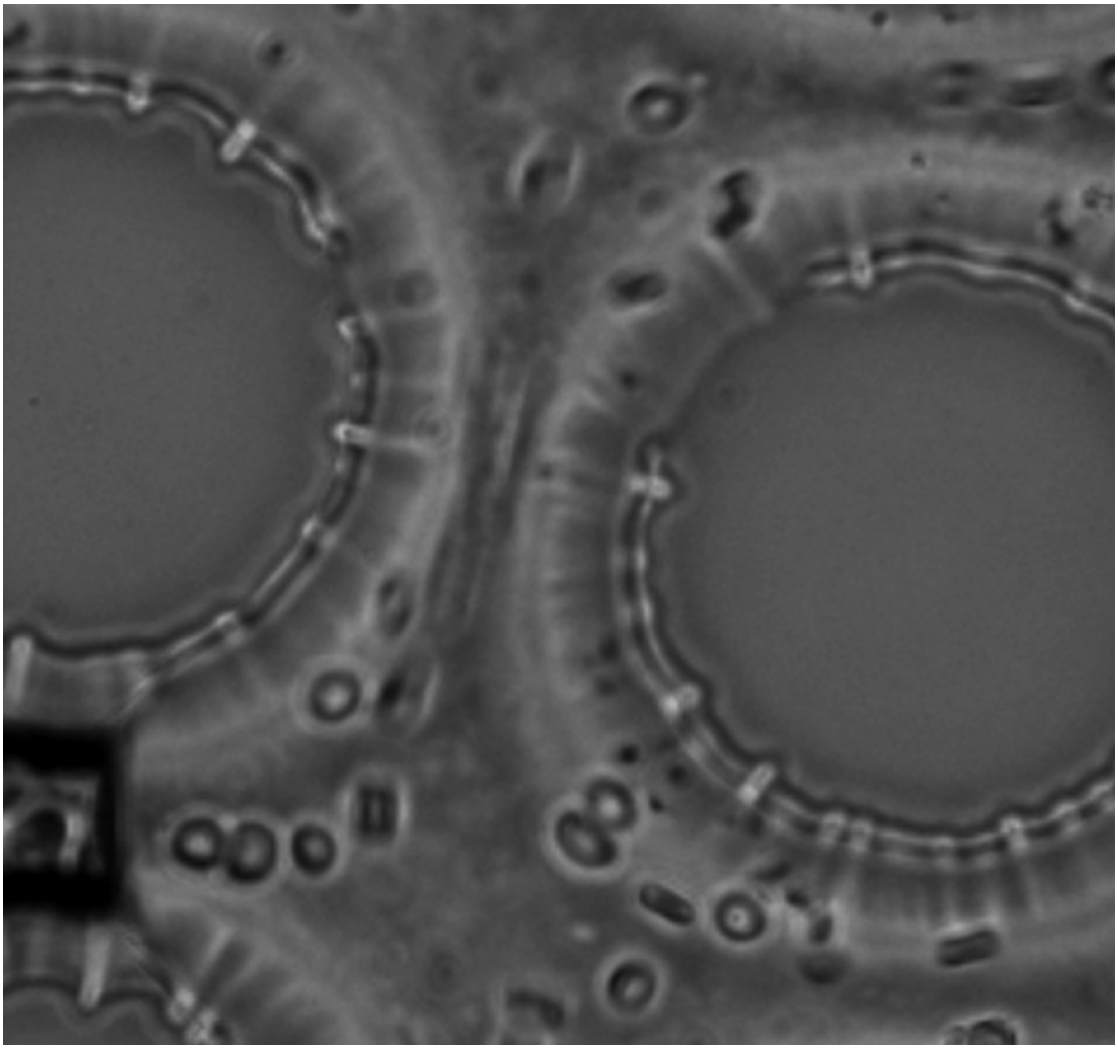


Figure 2: Example of blurred cells due to low FPS [44]

Evaluation of tracking is also a non-trivial problem. Object tracking which uses bounding boxes from detection is heavily dependant on the performance of object detection. This requires detection to be incorporated in any metric used for indicating performance increase of an algorithm. Tracking itself has various useful metrics. One such metric is for example the average length of a track, especially when compared to a dataset. Another metric is the percentage of correct track segments. Possible condensation of all these remarks can be seen for example in [28] where both detection and tracking contribute to a final score called AOGM (Acyclic oriented graphs matching) based on predefined weights adjusted according to their importance.

1.3 Common issues

Even though common algorithmic approaches already solve these problems in general, specifically cell detection and tracking has its own set of problems. When acquiring video data from biological experiments, not all data is usable. Footage from biological experiments is more often than not below average quality, the lighting is very poor, and for example colors are usually non-existing. Cells themselves tend to move at a relatively high speed comparable to their size. This causes blurring and in some cases virtual disappearance rendering certain videos not only unusable for automated processing but even for manual processing (Figure 2). There are also a few cell-specific events, such as cell division or cell death which do not occur in generic detection and tracking tasks. Cell detection in particular has to deal with an increased amount of overlap and boundary blurring due to overlap which is not usual for most objects and occurs due to the partial cell transparency.

1.4 Descriptors and Classifiers

Descriptors and classifiers are the most basic building blocks for image processing, that work in connection with each other. They enable us to tackle tasks, like image recognition, object detection, automated labeling, or filtering.

A descriptor is a representation of features in specific type of data, in our case

a visual descriptor represents features in an image. These features need to have specific properties for the successful detection/tracking in an image. Different descriptors serve a different purpose, and each computer vision task can have different descriptors. They are mostly divided into two categories. First category are low level descriptors, which give description about shape, color, texture or motion. Second category are specific domain descriptors, such as descriptors which give information about the object or the scene. Examples of such descriptors are: for face recognition (Haar Wavelets) or object detection (Histogram of Oriented Gradients). Haar Wavelets for example sum areas on an image with differing signs, and produce one number, i.e. a feature.

Classifier on the other hand is an algorithm, which is able to classify data. Its result is a discrete class for each given piece of data. Most common classifiers are binary, but also multi-class classifiers exist. Classifiers use a feature vector as input, in the case of computer vision it can either be the raw image, or more frequently the output of a descriptor. A good example of a classifier is a decision tree. A simple decision tree takes a number as input, and then by traversing along the tree to the leaves, we get a class decision.

1.5 Machine Learning

Machine learning (ML) deals with the study and construction of algorithms, that can learn from data and make predictions. Such algorithms are designed to make data-driven decisions, as opposed to static coded programs, which can only respond in one way. The tasks for which machine learning is used, tend to be difficult or unfeasible for normal explicit programs, due to various criteria, for example performance or development time. Examples of tasks for which machine learning is used are malware detection, email filtering, or computer visions problems.

There are two kinds of machine learning, or learning in general:

1. Supervised learning
2. Unsupervised learning

Supervised learning requires a labeled dataset, for which a mathematical model is constructed. Such a data set for example in cell detection will have images, which

Figure 3: Example of Haar Wavelets. The rectangles represent a specific Haar Wavelet. The squares are summed up, with black taken as negative and white as positive values, producing one feature.

are either labeled as cell or as background. It is most useful in two areas, those being classification and regression problems. Examples of algorithms which utilise supervised learning are Support Vector Machines, Linear Discriminant Analysis or Decision Trees. This approach is valid for tasks where a dataset of ground truths is obtainable, and that is not always the case.

Unsupervised learning is an approach, that describes unlabeled data. The idea is to give structure by inferring a function. The downside is that there is no straightforward way to evaluate unsupervised learning algorithm, compared to supervised learning methods. That being said, the upside of not having to supply the algorithm with any dataset of ground truths is very critical in some tasks. Algorithms, which use this approach are for example clustering methods like k-means, certain neural networks or autoencoders.

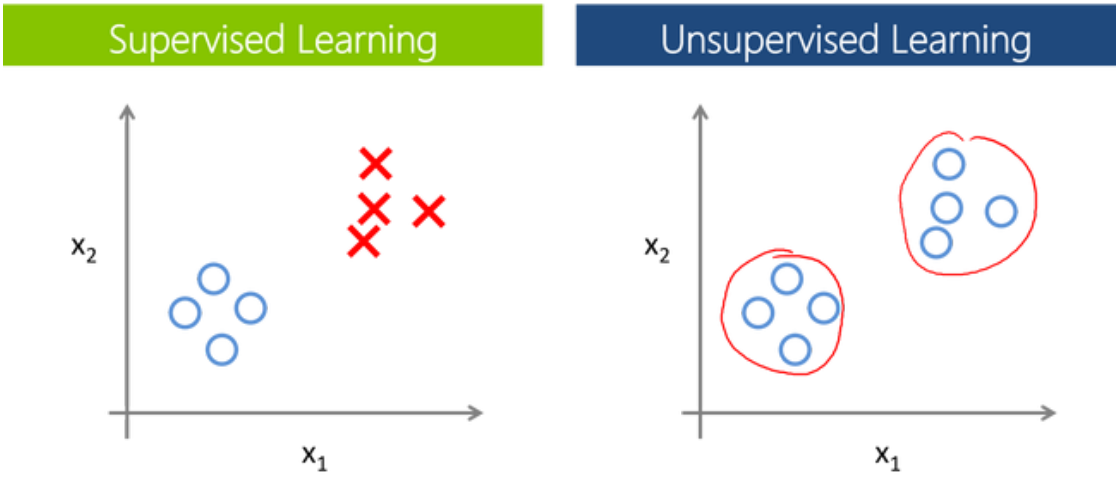


Figure 4: Example of supervised vs. unsupervised learning. Supervised learning specifically segments samples into circles and crosses, whereas unsupervised learning only groups together similar samples, without knowing what they are called. Source: [46]

1.6 Issues of Machine Learning

Machine learning has to overcome many obstacles when being used. First of such problems is overfitting. Logically one would think that more data always equals better results. This is not the case. Some machine learning algorithms try to solve classification by deciding which side of a hyperplane the sample of data is on. In such case, overfitting can be viewed like a too-detailed hyperplane, which models itself based on either wrong or extreme data. (Figure 5) The result of overfitting can be either loss in classification performance, i.e. giving wrong decisions or loss

in prediction speed, since a too-complex function likely requires more variables in its decision making process, making the process slow in comparison to an optimal function. Also another downside can be the loss of portability of such model.

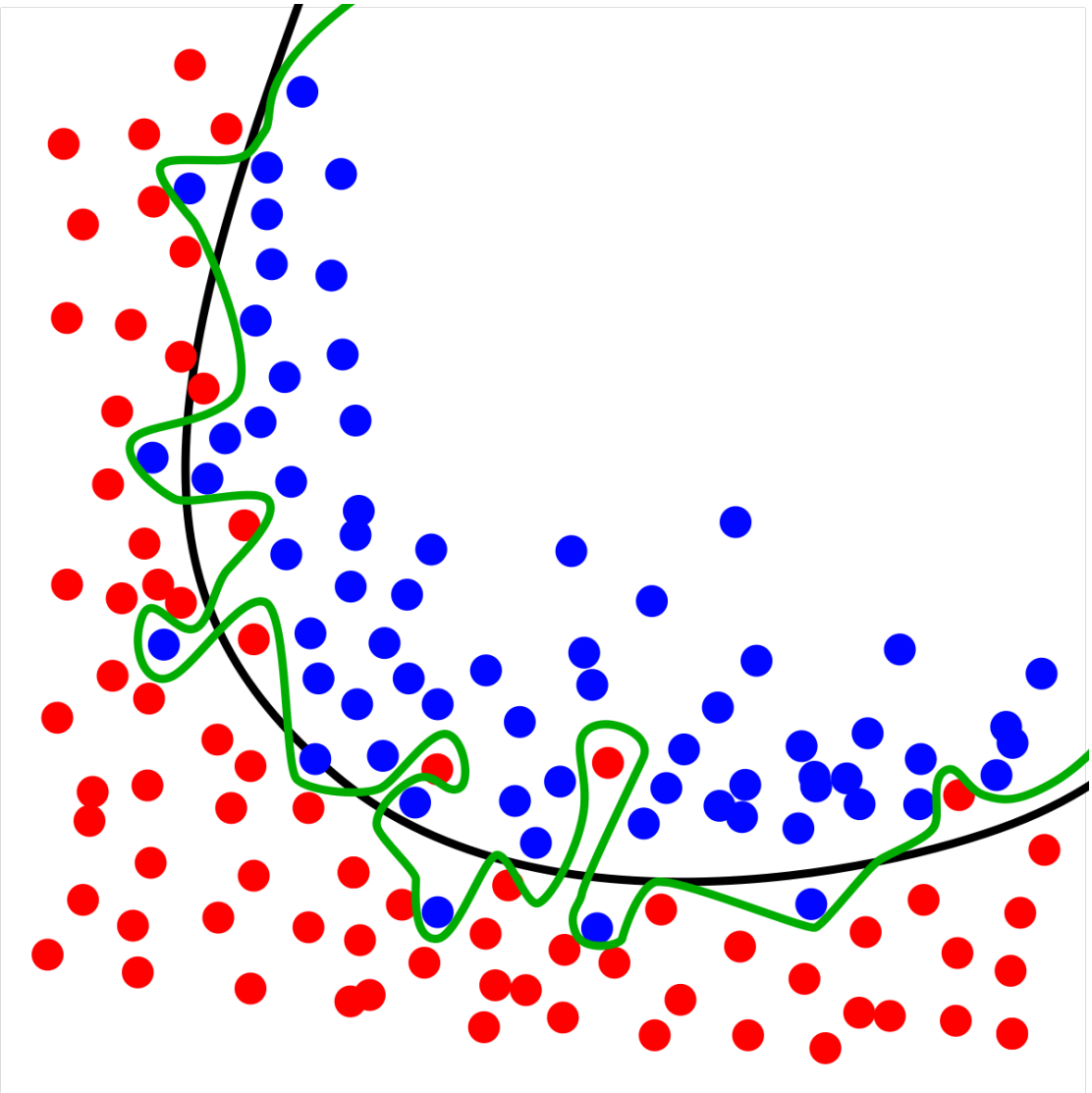


Figure 5: Example of overfitting. The green line is an example of an overtrained model, and the black line is an example of an enough trained model. The dots on the wrong side are usually noise or wrong data.
Source: [47]

Another common problem in machine learning is the curse of dimensionality.

[16] Machine learning problems tend to take a multidimensional approach to linear problems. The issue with that, is that simple logic tends to be wrong when dealing with a high order of dimensionality. Such problems include for example the usage of common euclidean distance, where measuring the distance in a multi-dimensional space does not always do what we want to achieve (nearest neighbour algorithms). Building a classifier in two or three dimensions is easy, since validation through visual inspection is possible and easy, but this is not the case in a 100-dimensional space. It is important to understand, that the dimensionality grows with the number of features. It is commonly the case that all 100 features of a given data source are relevant, and can't be reduced to less. Fortunately, there is an effect that counteracts some of these symptoms called blessing of dimensionality. The idea is that in most applications, the data samples are not spread uniformly around high-dimensional space, but are clustered in certain areas. As a result, dimension-reducing algorithms can be used or algorithms can take advantage of this effectively lower dimension.

In addition to that, machine learning is often hard to explore and examine directly. Many machine learning algorithms work with arbitrary weights which work in unison to render proper judgement, but to a human observer individual values mean nothing. As a result machine learning algorithms are often perceived as a black box. troubleshooting errors in learning can often be both computationally and time consuming. There are more issues with machine learning, some of which are more common than others. These other issues are explored in [16].

1.7 Overview of work done in the field

Visual cell processing is a common practice in computer vision. This task has the potential of helping disease detection and aiding professionals in diagnosing patients. The common workload is the detection and multi-class classification of cells and the resulting interpretation for example in [14].

The problem with detecting all different kinds of cells, means that the success rate of such an algorithm is currently much lower, than in a case where the algorithm focuses on only one kind of cell, or cells with similar characteristics. When looking further into existing research of cell detection, our focus was on tasks with

only one cell type, or specifically red blood cells.

Concerning more traditional computer vision methods, morphological methods gave good results. In [18], the proposed method uses edge detection using various algorithms, selecting canny edge detector as the most successful one. The method then plugs the detected edges into hough transform, which successfully identifies cells. In [17] this output is not just used for detection, but also for classification between normal cells and sickle cells. It is worth mentioning that both these papers utilize high quality video, and therefore does not have to deal with many challenges due to low quality and bad lighting.

In [19] the proposed method utilizes histograms and thresholding to achieve a very good cell detection algorithm. Certain color ranges are thresholded to 255, being considered cells, and certain ranges are thresholded to 0, being considered background. The image processed in this way is then analysed, and clusters of 255 value pixels are taken. Clusters which have below 100 pixels are considered noise. These clusters are then counted, and compared to manual counting of cells and existing algorithms, showing that the proposed method indeed gives improved results.

Recently the focus has shifted to the usage of neural networks, for the purpose of cell detection. Most research papers deal with the unified task of cell detection and cell tracking, utilizing neural networks for the detection part and then more traditional methods for cell tracking using the detected cells. Such an approach is used in [26] where physical properties are used for making connections. On the other hand in [13] the proposed method utilizes 2 neural networks. The first neural network gives suggestions for bounding boxes in an image. The second neural network is then online-trained for a specific cell and is used to decide which bounding box proposals are the same cells, and lastly a traditional tracking algorithm is used, which utilizes physical properties like speed, shape and so on. The paper also showcases a benchmark algorithm ISBI (International Symposium on Biomedical Imaging) cell tracking challenge, which is used for evaluating this cell tracking task.

Then in [14] again a set of two neural networks is utilized for successful cell tracking. This paper presents ways on how to approach tracking specific cells

using manual bounding box labeling and then utilizing sampling methods to train a neural network for this particular cell. Next a particle filtering method is utilized in the cell tracking stage, and then its results are fed into another neural network, which gives propabilities for it being the same cell, like in previous frames. In comparison to the [13] a completely different design of a neural network is used. Also again a benchmark data set is presented, in this case for stem cells.

When going over state-of-the-art algorithms in the field of computer vision [15], they tend to improve results by utilizing various methods at the same time. The utilized methods are both traditional morphological or machine learning methods for detection. In the cell tracking step, usually graph-based, overlap-based or distance-based algorithms, without the use of machine learning. The problem with these algorithms, is that they are specifically built for a task, and as a result there is no way of using existing algorithms if we want high detection and tracking performance.

In summary very few methods deal with red blood cells specifically, and more often than not, the presented results are made on ideal data, that does not reflect commonly available video. That being said, those algorithms which focus on a specific cell type, do give better results, than those which try to work with cells in general. It is worth mentioning that only detection stage is usually impacted by being multi-class. In tracking algorithms, at least those which utilise physics to connect objects into tracks, the results are not impacted by having a multi-class problem.

The research is lacking in areas of dealing specifically with RBCs, for both their detection and tracking. This might partially be due to RBC-only datasets with sufficient quality not being readily available. Using methods, which were utilized in other parts of this field is one avenue worth investigating. Another such possibility seems to be in utilizing deep learning not only for detection, as is quite common these days, but also for object tracking. Lastly, tightly connecting these steps, detection and tracking, and creating some sort of feedback loop is worth exploring as well.

2 Previous Research

Progress on this topic has been done within our team as well. In [42, 43], an algorithm was developed, which was composed of two steps: Detection and tracking.

The first step of this algorithm detects RBCs. Static images or video is taken as input. The output of this step are centres of cells within a given image. This step first utilizes background subtraction methods, in order to remove clutter and noise from the image. Subsequently the image is processed using an edge detection algorithm, specifically canny edge detector. The resulting edges are then used with Hough transform, which searches for two shapes: circles and ellipses. The Hough transform creates dense clusters of high intenstity points, where centres of shapes are located. A thresholding operation is then applied to the image, and qualified points are then used as output of this step.

The second step uses the result of the first step as an input. Singular cells are connected into tracks for each cell. In the initial step, where no physical data is available, neighbouring centres are connected using an initial radius. After that, the algorithm uses the speed and direction from the previous frames, to predict which bounding box should be chosen from the next frame. The direction is computed from both the previous motion, as well as the flow matrix. A flow matrix tells us what vector a cell would have in a given point, without having any other information. This is useful both when deciding what to do in the next frame, and also when the track is heavily fragmented. The current implementation is capable of skipping frames, in which there was no valid detection. The output of this step is a track for a cell, and the assortment of bounding boxes for this cell in each frame. This output can be used for statistical purposes, and is the desired output for this algorithm.

The problem with this approach is that it is still subject to many flaws of each step. The detection step has issues with lighting and shapes, that are not exact circles or ellipses, as well as setting a threshold for a particular data source, due to differences in color intensity. Unfortunately, the tracking is heavily dependent on detection input. When annotated data from our dataset was used as input for the tracking instead of the output from the detection step, the tracking algorithm was outputing very long tracks, as opposed to fragmented and short tracks when using

the detection step output (Figure 6). Going forward, this thesis will deal with the research and improvements to the detection step and the tracking step in tandem as well as extracting data from both steps suited for simulation validation.

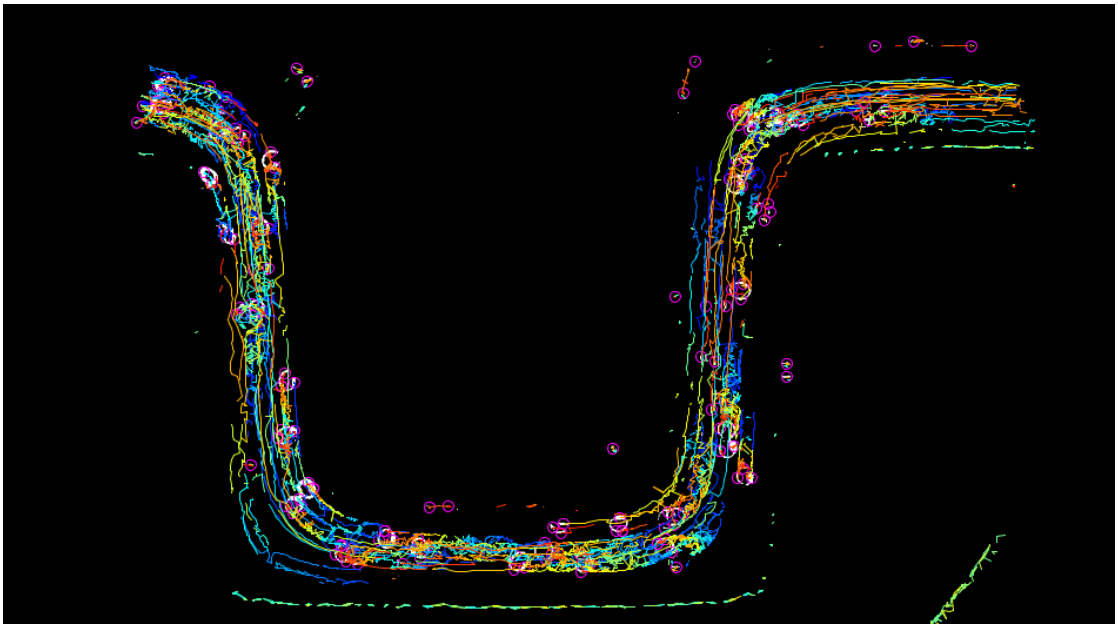


Figure 6: Example of fragmented tracks. Source: [43]

2.1 Analysis of the current situation

As was detailed in the overview, other algorithms for the purpose of cell detection do exist. The algorithms that were presented, have shown the progress in the field. Most algorithms started out with best-case-scenario data, which was well lit and easily analysed, and a single method for detection was required to provide a good result, like with Hough transform. This was the approach of my team as well. The problems start when Hough transform is applied to other videos. For example, calculating the threshold value for cell identification ended up being a big issue and not transferable between videos. Another issue is that when objects are very big, i.e. high quality video, dual edges are made out of cells, causing Hough transform to display detection artifacts. (Figure 7) Another issue is that similar round objects (Figure 8) can also be wrongly classified as a cell.

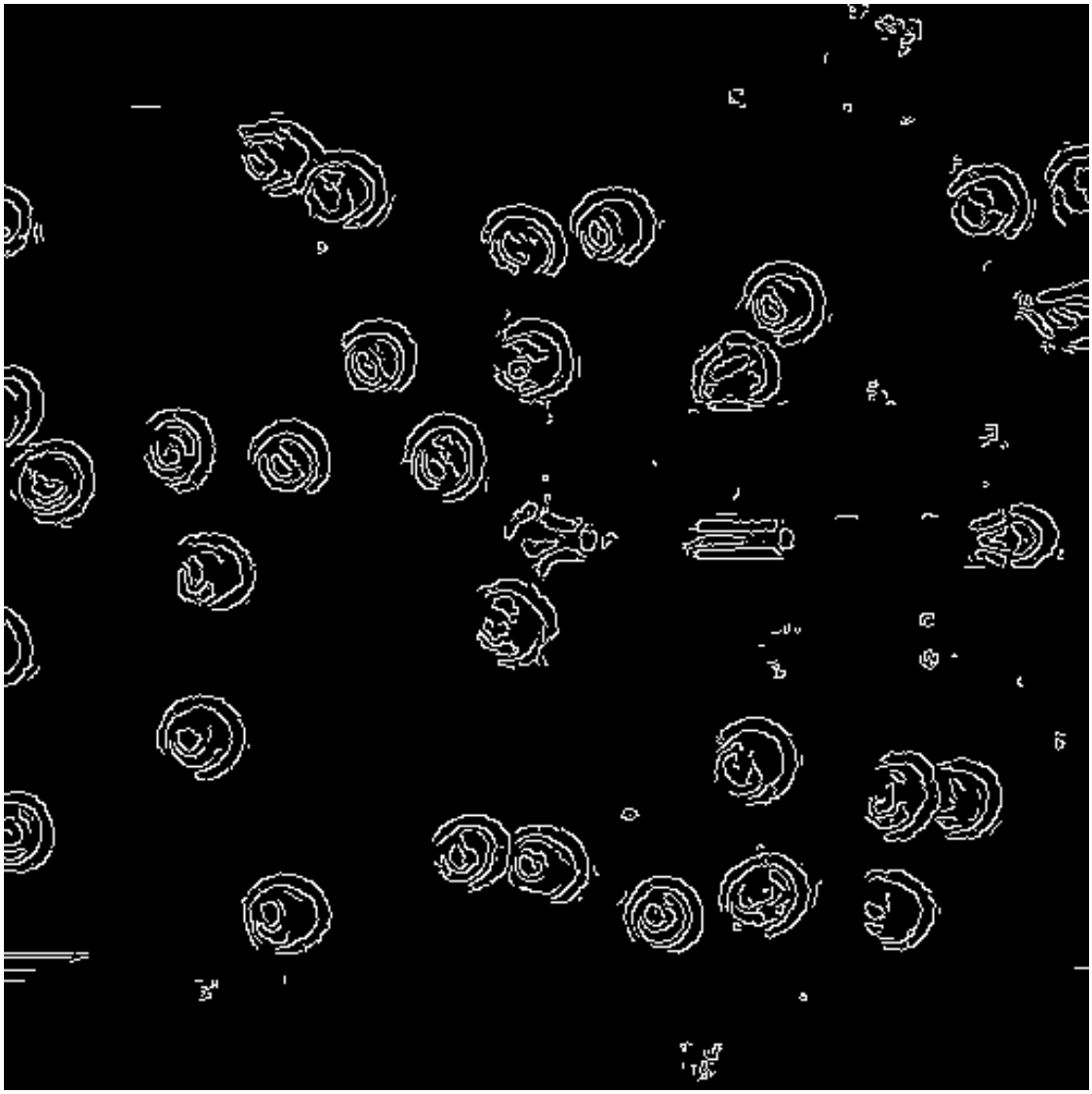


Figure 7: Example of double edges with canny edge detection. Background source: [44]

Coming back to the previous analysis, the commonly adopted solutions were as follows: 1. use multiple methods, which together alleviate the issues of one particular method, or 2. go into machine learning. We chose to go with various machine learning algorithms, with other parts of our team going into the first method potentially in the future. The reasons for choosing it were simple. Machine learning algorithms usually require a dataset for the online part of the training.

This requirement was fulfilled when we made our dataset (more in chapter 3). The dataset was made from the same video on which the existing algorithms were implemented. The dataset contains bounding boxes of each cell, with coordinates and height + width. The dataset is still a work in progress, and there are plans on extending it, especially adding a cell tracking part, which will enable cell tracking performance benchmarks.

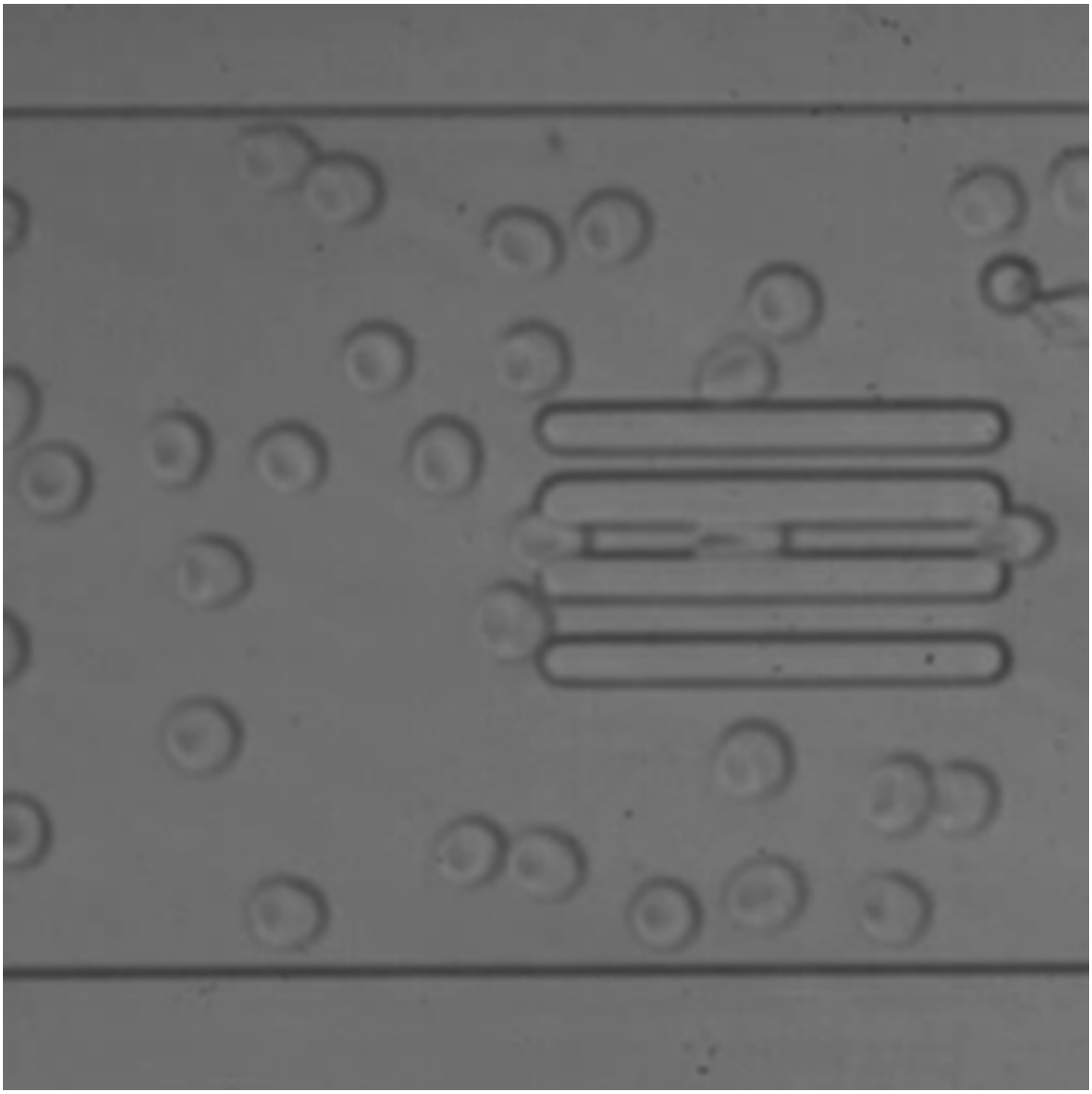


Figure 8: Example of round objects, which interfere with detection. Background source: [45]

An additional reason for choosing machine learning, was the previous positive experience with both using, and implementing additional parts of AdaBoost, namely SHOG descriptor that was added to the Viola-Jones cascade [41], as well as experience with CUDA, GPU architectures and other machine learning algorithms like SVM.

2.2 AdaBoost

Our previous experience gained during master studies led us to test the AdaBoost algorithm and to evaluate its use for our particular task. First a brief description of the algorithm and the surrounding technology. AdaBoost is a machine learning meta-algorithm, which can be used for a variety of tasks. The meta part comes from training weak classifiers, which can be any classifier, most commonly used are decision trees. AdaBoost takes these weak classifiers in bulk, assigning weight to each of them, and producing one more powerful classifier as a result.

The inputs are the same as for most machine algorithms that is a fixed size feature vector, that corresponds to the trained model. As an output, we have the assignment into one class or another. AdaBoost can be both binary and multi-class classifier, in our case it will be binary. The algorithm works in two stages, first stage is training, second stage is testing. In the training stage, labeled data is supplied into the algorithm, which trains a model composed of certain features, and their labels. These labels tell the algorithm, to which class the data belongs to. In the testing stage, a judgment is produced based on the data sample.

When it comes to features, a descriptor has to be chosen that is good for that particular task. In practice this means that different types of feature descriptors have to be tested, to determine which one is best for that task. Another thing that is important, are cascades. Cascades combine multiple classifiers, in order to improve speed of an algorithm. This enables us to put a fast, but not very rigorous classifier at the front end, and a slow, but very powerful classifier at the back end. This means that most data, will be classified as false, by the time they arrive at the slow powerful classifier. The critical behaviour of these fast classifiers though, is that they maintain a 99% or higher true positive rate, so that actual good data make it to the complete end. (Figure 9)

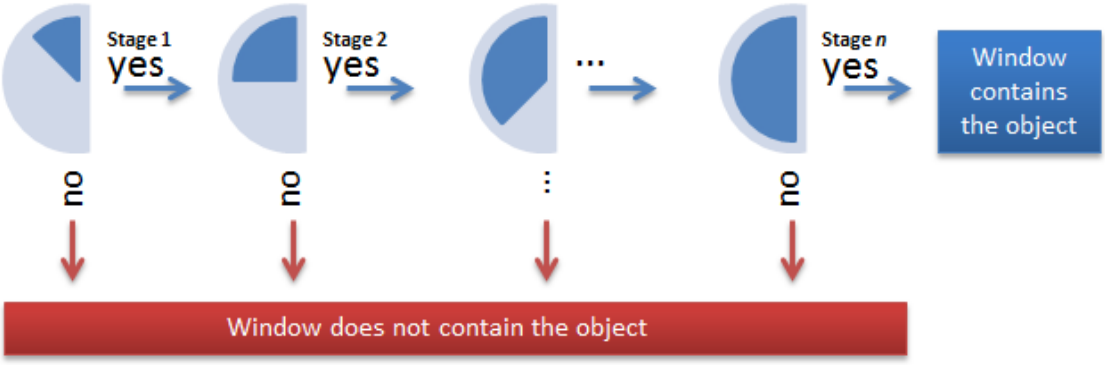


Figure 9: Example of a cascade. Each stage removes a percentage of samples, and samples which get to through the last stage are labeled as containing an object. Stage 1 is usually much faster than Stage n. Source: [48]

The particular implementation of cascades and AdaBoost in question is called Viola-Jones cascade [20]. It is part of the OpenCV library, and contains a set of tools for training and also a high performance suite for testing, as well as some implementations on the GPU. This cascade has 2 types of supported features out of the box, Haar Wavelets and Local Binary Patterns, out of which the Haar Wavelets were relevant to us. Haar Wavelets can be easily generated in thousands, which means they are usable for both fast and slow stages.

The cascade was tested using various settings by training different models and then benchmarked as highlighted in [K2]. The best model image width and height ended up being 25px by 25px, which corresponds to the usual cell sizes in our dataset, which are from 25-30px. Weak classifier count restriction was also tested, but had no performance or speed benefit, unless a very low sub-10 number was chosen, when the performance started to decay. Increased depth of weak classifiers (depth of the decision tree), had a negative impact on detection performance, as a result depth 0 was chosen for the final model.

Table 1 contains best achieved results for detection with Haar cascade. As we can see, the positive detection rate is at about 50% of total detected object count, which means that half of the objects output by the algorithm are valid detections. This might seem like a low number, but one has to take into consideration that the image had thousands of small windows to process. The false positive rate is at

Table 1: Best AdaBoost detection results confusion matrix

	Predicted cell	Predicted background	Total
Actual cell	1 320	1 290	2610
Actual background	1 322	approx. 8 000 000	X
Total	2 642	X	X

about 50% as well, but this can be partially countered by training a more rigorous cascade. In the future the algorithm might provide better results when coupled with different feature descriptors, but is not particularly well suited for our task. The only possible use would be in tandem with the Hough transform.

With these results, which have proven to be moderately successful, we decided we will try a more novel approach using machine learning, specifically deep learning. In the next chapter we will go over our initial research going forward with neural networks.

3 Dataset

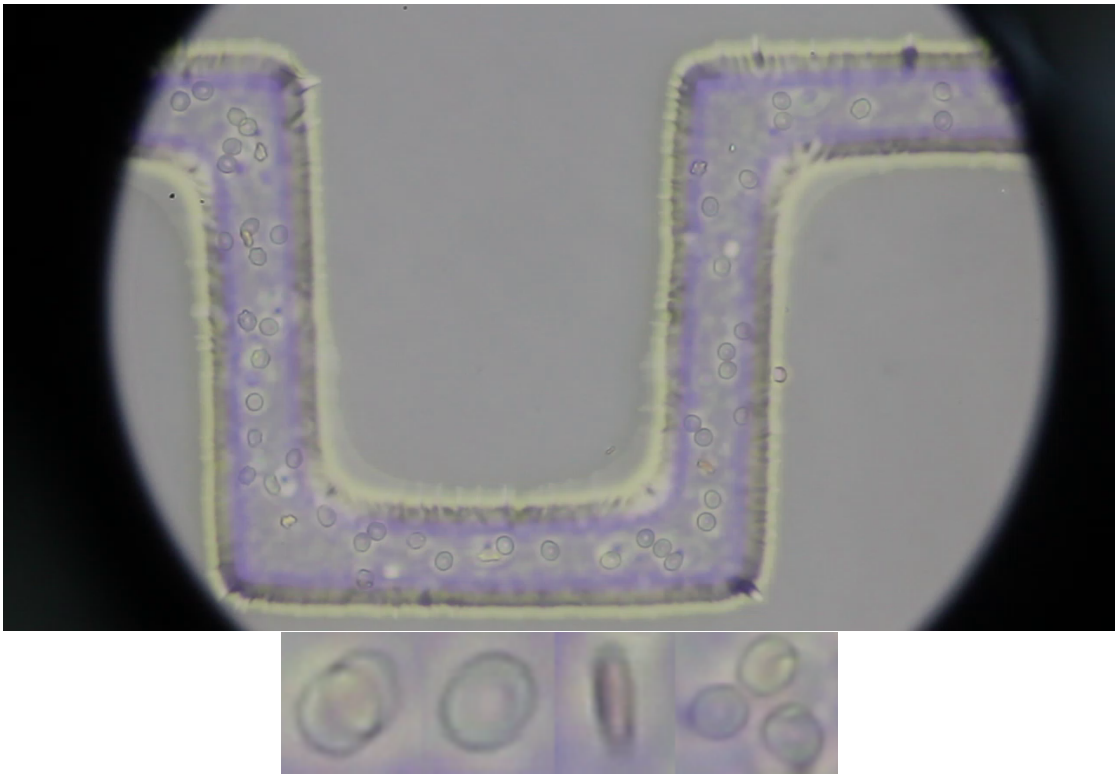


Figure 10: Example of first video and examples of cells in the video. Source: [44]

In order to be able to evaluate any algorithm, be it cell detection or tracking, we have created a dataset for both tasks. The current version of the dataset contains two videos. In the first video (Figure 10) we have annotated 300 frames for detection purposes. The video generally averages to about 50 RBCs per frame, the cells being between 25px to 35px in size and the resolution is 1280px by 720px. This video also has tracking annotations for 300 frames for tracking testing purposes. The second video (Figure 11) has 100 frames of bounding boxes annotated for detection. This video averages to about 35 RBCs per frame, cells being 35px to 45px in size and the resolution is 512px by 512px. More detailed information about the dataset videos is in Table 2

Our main goal when creating this dataset was to create a dataset that is diverse enough to encompass the general characteristics of RBCs in different experiments.

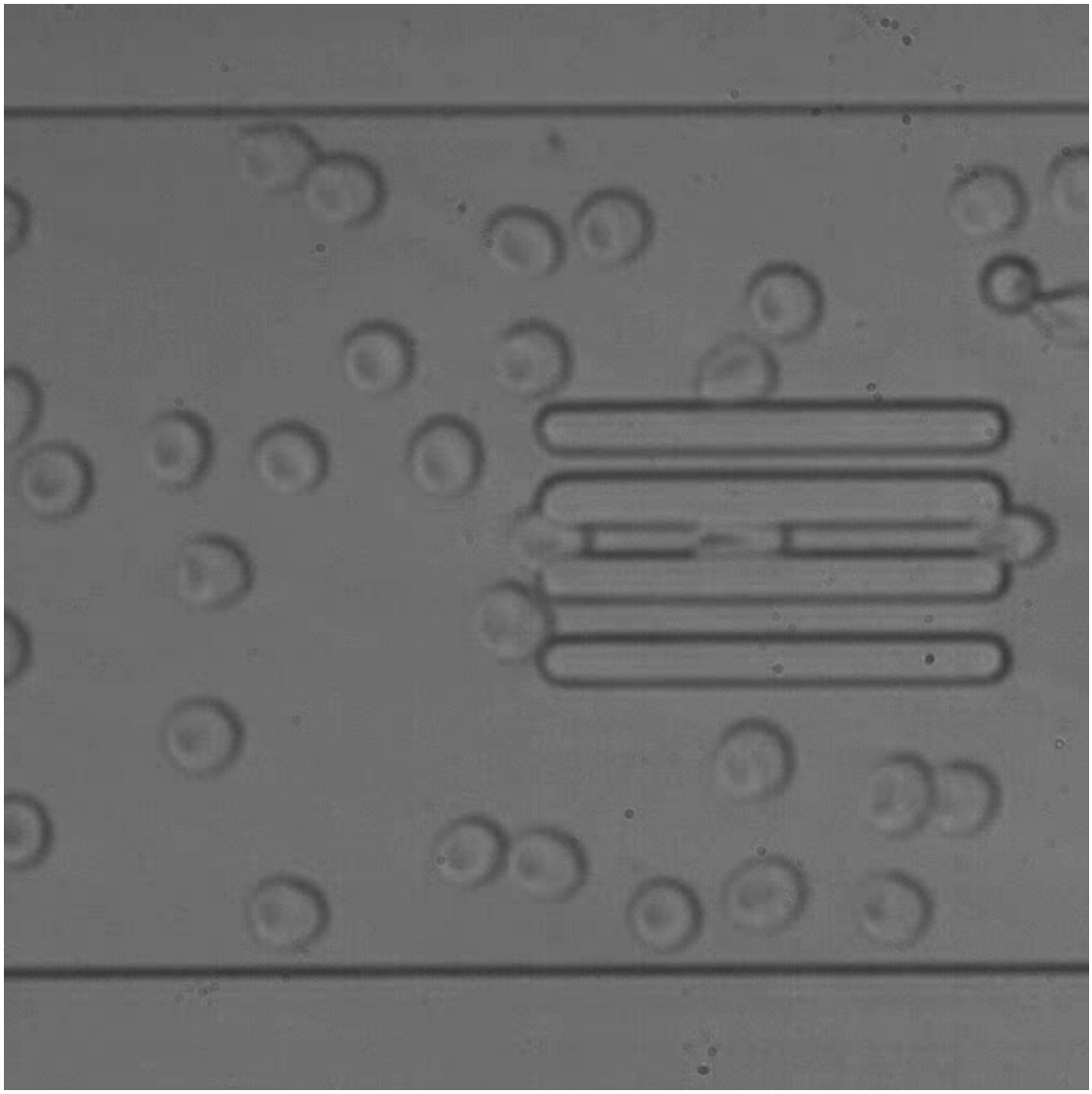


Figure 11: Example of second video. Source: [45]

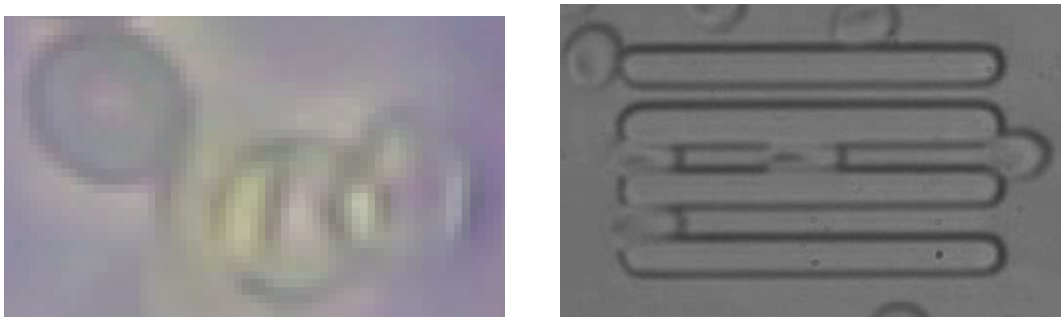
The addition of our second video to the dataset introduced us to new RBC behaviour due to the squeezing mechanism in the experiment from which the video originated. In the second video , the cells themselves are more easily discernible from the surroundings and they have different scale ratio to the whole video frame. Both videos contain cases which are difficult to discern even for humans. Such cases can be seen on Figure 12 Going forward it is of utmost importance to extend this dataset with more annotations, especially for any usage of machine learning

algorithms which greatly benefit of added variety.

Table 2: Dataset Data

	Bounding Boxes	Tracks	Average Width	Average Height	Average Track Length
First Video	15089	85	25.0147	24.4071	176.9647
Second Video	5012	265	40.1941	40.2298	18.8301

Key information about differences in the dataset. We can see that cell sizes are different and tracks are also fundamentally shorter.



(a) Clump of cells

(b) Microfluidic channel obstacles

Figure 12: Example of difficult events for detection.

The actual structure of the dataset is done through an XML file, which holds the information about file names and locations. Each image file holds information about all bounding boxes. The bounding boxes themselves contain their location and dimensions, as well as the track to which they belong if the information is available. For ease of transfer, any described algorithms are capable of using this format as input/output.

4 Convolutional Neural Networks

When it came to the initial testing of convolutional neural networks (CNNs), it became clear that we do not want to jump right into making our own implementation. As a result, we ventured into finding the correct implementation, that would serve our needs for the foreseeable future. Most of our previous codebase was either in Python for Hough transform and tracking, and then in C++ for AdaBoost. Therefore it would make little sense to focus on an additional language, and would also create problems with maintaining the code down the line.

When it comes to neural network frameworks, the most commonly referred to in 2018 are TensorFlow and PyTorch. Both frameworks use python, and both are very well maintained. TensorFlow is slightly more established, starting back in 2015, whereas PyTorch is a very recent, and is still in beta. Both are capable of utilizing GPUs for heavy workloads, most importantly CUDA technology. We opted for TensorFlow based on the fact, that the community is more ripe and the availability of tutorials is more widespread. That being said, our focus is on detection and not being rooted to one framework, so we might revisit PyTorch in the future.

Both frameworks are open-source, meaning it is possible to extend them, and contribute to the original repository. Both are written in a blend of C++ and Python, with their own building tools and contribution guidelines. In the future should we wish to improve any existing implementation, or make our own, it is possible to do within both of these frameworks.

4.1 Proof of Concept

As an initial proof of concept for delving into CNNs for detection, we constructed a simple neural network using Python and TensorFlow. The input of the CNN is going to be a frame of the video and the output will be the probability of a sample being a red blood cell. It consist of 3 convolutional layers, one flattening layer, and two fully connected layers, in this specific order. After that we apply the Adam Optimizer, which minimizes the cross entropy with the annotated data. We let the algorithm run for 1500 iterations, which is not too much, but this proof

of concept was run on a CPU, rather than on a GPU.

The convolutional layers apply the convolution operation. Each layer applies a 3x3 operation with max pooling. Max pooling takes the output of a convolution operation, and takes the maximum value out of a 3x3 matrix, thereby reducing the image size in the process and growing in channel count. The first and second layer produces 32 channels, and the third layer produces 64 channels. The strides of the filter window are 1,2,2,1 in a 4D tensor, meaning in the first dimension they move by 1 px. After the 3 convolutional layers, we add a flattening layer, which squashes the 4D feature output into a 1D feature vector with 128 features. This is then fed into the two fully connected layers, the first layer reduces the feature vector to 64 and the second creates our 2 output values. Those are probabilities for class Cells and Non-cells, and since these values are complementary, we only need one value out of these. Visual representation is on Figure 13.

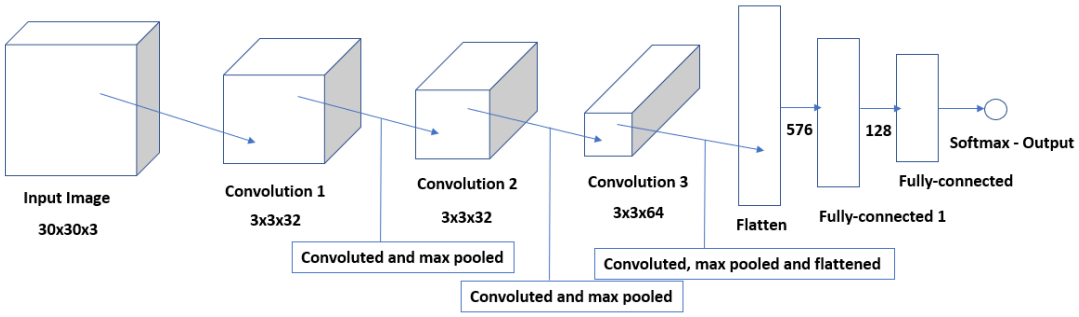


Figure 13: Diagram of our convolutional neural network.

Adam optimizer is a stochastic gradient method, that maintains a single learning rate for all weight updates, and maintains an adaptive learning rate for each network weights. The algorithm is commonly used in deep learning. When it was first designed it was applied to a logistic regression algorithm, and proved to be fast and efficient.

The training dataset consisted of 279 cell images and 549 background images which were segmented from our dataset. The testing dataset consisted of 261 cell images and 873 background images. These were selected at random from our whole dataset and the images between training and testing datasets do not overlap. The

success rate of this trained CNN is around 87% on the training data. Considering the fact that the training is not using all of the data, the training process was relatively quick (around an hour on a CPU, 20 minutes on a GPU— and the fact that this CNN was designed only for general use, and not specifically for cells, this is a very good result for our task. For these results we used 2000 iterations which equalled to about 50 epochs.

Table 3: CNN network training results

	Predicted cell	Predicted background	Total
Actual cell	6172	1058	7230
Actual background	1626	114501	116127
Total	7798	115559	123357

For the evaluation of this prototype, we used all the remaining cell images in the dataset, which amounted to 7230 cell images and 116127 background images. The results can be seen in Table 3. The resulting precision is 79% and recall is 85%. If we compare these results to our implementation of Hough transform, which has around 90% precision and 65% recall, and to AdaBoost with Haar wavelets, which has 50% precision and 50% and recall, we can already see, that our simple CNN prototype provides competitive results on the same dataset, with very little training time and a small dataset. Even though these methods have fundamental differences, the results were obtained by cross referencing the bounding boxes, or centers of them in the case of Hough transform, with the provided results of each method of detection.

Our prototype can be improved by using recent discoveries in state-of-the-art algorithms. Instead of opting for pooling methods and conventional convolutional layers, we can utilize dense layers described in [6]. This is possible through the propagation of information not being lost between layers. In addition to that, we can remove one fully-connected layer since based on other work being done in the area one is completely sufficient.

The final version is composed of 7 dense convolutional layers and one pooling layer, complemented by a fully connected layer which produces classification probabilities. We trained this final version with 10000 iterations. We did the same with

the first prototype in order to produce an equal comparison. After more thorough training, both models provide comparable results, with the final version providing 98% precision and 93% recall, and the former version 97% and 93% respectively. It is however of note that the more complex final version was also training more slowly by about 10%.

These prototypes have one major flaw however. They are quite slow and not being able to process whole images at once. As a result we need to look at existing approaches which provide performant and ready full image processing and evaluate them on our task.

4.2 Testing configuration

For the evaluation of detection we had to preprocess our dataset. We have created 3 subsets of our dataset:

1. 150 images of first video for training and 50 images for testing during training
2. 200 images of first video for training and 50 images for testing during training
3. 200 images of first video for training, 50 images of first video for testing during training, 30 images of second video for training and 20 images of second video for testing during training (this subset was also reshuffled 3 times for cross validation)

The purpose of this separation was to evaluate the impact of additional data on the performance of the neural network. The specific groups of images do not overlap within a given subset. The training process is supplied with whole images with annotated bounding boxes with valid RBCs. The rest of the image is used during the training for segmenting backgrounds.

Neural networks were trained with 100 000 iterations by default. We also evaluated training for 200 000 to see if it gives any added benefit. All configurations were selected by default for the COCO dataset which cannot be used for transfer learning in connection with our task and dataset it is very easy however to find configurations for all tested architectures for COCO, creating a perfect baseline for testing. Only change made to the configurations were the input resizers being

accommodated to our image sizes. We also tested both RGB and grayscale images for training and evaluation to see the impact of color on our images, since they are generally almost gray anyway. The input for each CNN in the evaluation stage is one frame of any size and the output is a tuple of position, class (in our case just one) and probability of each detected bounding box. For the grayscale testing, due to the framework being implemented for RGB, we converted each image to grayscale on load and then duplicated the one channel, so that the neural network was still supplied with 3 channels of an image.

The testing workbench was comprised of Threadripper 2950X and GTX 1080 Ti graphics card. The training time for most of our models was around 12 hours, with some of our training taking up to 24h each.

4.3 Architectures

For testing we have selected 3 main state-of-the-art approaches:

1. SSD - Single Shot Detector [31]
2. Faster R-CNN - Region - Convolutional Neural Network [29]
3. R-FCN - Region-based Fully Convolutional Network [30]

The main selection criterion was the ready availability of these architectures as well as top performance on datasets like for example MNIST, CIFAR or SVHN.

The SSD architecture utilizes an anchor system to propose the target bounding box and then classifies that bounding box. This architecture is aimed towards real-time performance and generally has issues with small objects, but that can be adjusted through generating more anchors for each image. For detecting different sizes SSD utilizes scaling of input image, which more akin to a traditional sliding window.

Faster R-CNN is fundamentally different from SSD in that it utilizes region estimation. As a first pass the architecture identifies interesting regions in the whole image and then proceeds to only render judgement on those regions, reducing computational complexity considerably. Next, the neural network classifies a given region and outputs an estimating bounding box for the searched objects. R-CNN

used to utilize an external region estimation algorithm, whereas moving to Fast and Faster R-CNN the whole algorithm was moved to the neural network.

R-FCN is also a region estimating neural network. R-FCN tries to improve on R-CNN through simplification of certain steps providing comparable detection performance at an increased real-time performance. Even though it is similar in logic to R-CNN, the design of the neural network is different and it also responds very different to changes in shape, color or scale. As such we need to evaluate it as well to see what suits our needs best. On paper, this is the best performing architecture.

4.4 Test results

For evaluation of CNNs we used the precision and recall metrics. In order to decide whether an object is the same in annotated data as in the detected data we used Intersection over Union (IoU) with overlap value of 0.5. We also used a second parameter which puts a limit on the maximum distance of centers of bounding boxes to be 0.5 * width of annotated bounding box. When judging these criteria we also segmented IoU into three bins, 0.5, 0.3 and 0.1 and the distance into 0.5 and 0.3 bins. This provided us extra information when comparing the different architectures, specifically related to centering of detected object.

Firstly we tested the detection results of our 3 architectures on all 3 described subsets of our dataset. The results can be seen in Table 4. We can see that a fully trained network provides considerable improvements over our existing methods in two cases. SSD seems to behave very poorly on our dataset, even though it was configured for the same COCO dataset as the other two architectures.

We can see that all the neural networks perform considerably worse when trained on only 200 frames, indicating that performance still can improve with more data going forward. The more interesting metric is that when trained on both videos, even though Recall values go slightly down, the performance is still going well. We can also see that without adding data from second video, the neural network fails to detect almost anything. This implies that we need to introduce more variety into our dataset. The performance on the second video after only adding 50 frames of training/testing data is better than on the first video, which

is due to bigger cells and more easily distinguishable contour. Faster R-CNN provides the best results of the 3 architectures by far. SSD seems to have issues with small objects in the first video, but on the second video it performs admirably right away, which leads us to the anchor system limiting the performance. R-FCN seems to be more susceptible to dataset variety after adding second video to the training. We will need to enhance our dataset further to see if R-FCN performance will be fixed.

Table 4: CNN testing results

	200 frames		250 frames		250 and 50 frames	
Precision/Recall	First video	Second Video	First Video	Second Video	First Video	Second Video
SSD	90.9%/66.3%	16.7%/11.0%	94.5%/68.0%	0.0%/0.0%	95.1%/68.7%	99.2%/93.3%
SSD - Adjusted	-	-	-	-	98.7%/88.0%	99.2%/95.1%
Faster R-CNN	99.6%/89.7%	0.0%/0.0%	99.3%/97.2%	1.4%/0.1%	99.5%/94.7%	98.7%/98.7%
R-FCN	99.2%/82.1%	2.7%/0.4%	99.6%/88.6%	10.0%/0.4%	82.5%/86.5%	95.5%/97.4%

First two columns represent trained models on 200 frames of first video. Second two columns represent trained models on 250 frames of our first video. Last 2 columns represent trained models on both videos with 250 and 50 frames respectively. The values in each column represent the Precision/Recall values of a given model. The models were trained with 100 000 iterations each.

Before we go over our adjustments to the SSD framework, we need to explain the impact on our metrics. First of all, our recall is high due to us including 0.5 width distance difference bounding boxes in the "true positive" category. While these suffer from worse localization, they are still valid detections for our next step. This improved detection across the board, but it improved recall of SSD on our first video by 16%. This seems to imply, that because SSD works on much smaller images, this causes some issues with pinpointing location. On the second video, which has much smaller resolution, the difference was less than 0.5%. After tweaking the model of SSD further, we achieved comparable results on our first video. We achieved this through adjusting anchor scaling of SSD model, as well as increasing the working resolution of the model from 300px to 600px. This also fixed the localization issue of SSD in the first video. It is to be noted however, that this had an adverse effect on the speed of training of this adjusted SSD model, causing it to train twice as slow. Luckily this is not a concern for us.

Table 5: Overtraining

	First Video
SSD	75.9%/59.4%
Faster R-CNN	92.9%/96.8%
R-FCN	95.4%/88.5%

250 frames trained with 200000 iterations from first video - values represent Precision/Recall

In Table 5 we showcase the results of further training our models with 200000 iterations, up from 100000. All 3 frameworks suffer from overtraining and precision degradation. This means, that our training is as good as it gets and will only benefit from additional data and minor tweaking using error rate during the testing step.

Table 6: Background Subtraction

	First Video	Second Video
SSD	93.8%/66.7%	72.6%/98.4%
Faster R-CNN	99.3%/88.4%	5.5%/0.8%

250 frames trained from first video with background subtracted from both - values represent Precision/Recall

Next, we looked at our background subtraction evaluation. The goal here was to lessen the transferability of our models between videos with as little human intervention as possible. Here we illustrate Faster R-CNN and SSD (R-FCN behaved similarly to Faster R-CNN) in Table 6. We can see that in the case of SSD, background subtraction improved transferability considerably at the cost of precision. With additional data manipulation we can likely create a model, which will be more robust towards lighting and color. Faster R-CNN on the other hand provided no noticeable performance uplift and as a result still needs training from the other video. When comparing our two background subtraction methods, temporal and static image, both provided similar results. Our first video has an accidental camera shift, and the temporal method mitigated this issue after the frames in

mind going out of history.

Table 7: Grayscale

	First Video (Gray)	Second Video (Gray)	First Video (Color)	Second Video (Color)
SSD	97.2%/88.3%	99.4%/95.5%	97.7%/87.6%	99.4%/95.5%
Faster R-CNN	98.2%/95.9%	99.4%/96.6%	98.2%/95.2%	99.4%/96.6%
R-FCN	72.7%/80.9%	87.9%/96.9%	71.8%/73.1%	87.9%/96.9%

250 frames of first video and 50 frames of second video converted to grayscale used for training, then evaluated on both gray and colored images - values represent Precision/Recall

We wanted to evaluate the impact of using grayscale images during detection. The logic stems from all the cell imagery being gray to the human eye. We can see the results of this evaluation in Table 7. The first thing we can notice is that R-FCN suffers greatly from being trained on a grayscale dataset compared to an RGB one across the board in both precision and recall. As such this framework is not viable for use with grayscale imagery. Second thing of note is that the frameworks have a noticeable performance degradation on the first video for SSD and Faster R-CNN for precision, but Recall is better than with a color trained model. On the second video we even see SSD outperforming its color trained counterpart and Faster R-CNN suffers a minor recall degradation. In turn if we use a grayscale trained model on colored images, we get almost the same results on the first video and identical results on the second video. This leads us to believe that color is still important to RBC detection in images but we do not need to shy away from potential gray image inclusion in the training dataset.

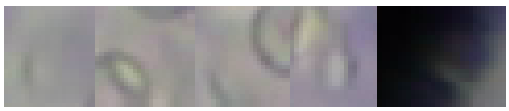


Figure 14: Example of false positives

When analysing all of our results, specifically false negatives (Figure 15), we came to a conclusion that the CNN performance is starting to outperform humans in certain cases. We took a closer look at cells which caused a Precision downgrade and when looking at multiple frames in a sequence, we noticed that manual anno-

tations for our dataset were missing certain cells. After visualisation, this enabled us to improve our dataset, further improving the results of our trained CNNs. On the different end of the scale, a percentage of false positives also turned out to not be completely invalid. In many cases the neural network joined several bounding boxes which ended up being in the middle of several cells around it. This is better than if the neural network was spitting out nonsensical data.

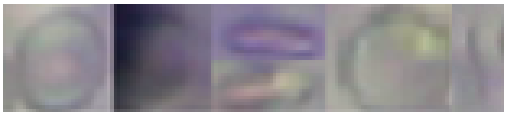


Figure 15: Example of false negatives

As last verification, we performed 4-fold cross validation on our test case of 250 images from first video and 50 images from second video. The variance of Faster R-CNN precision was 99 % ± 1 % and Recall 94 % ± 4 %. The variance of SSD precision was 97 % ± 1 % and Recall 80 % ± 4 %. The variance of R-FCN precision was 99 % ± 1 % and Recall 86 % ± 2 %. The results confirm the quality of Faster R-CNN for our task and also show that SSD is more unstable in performance compared to R-FCN.

5 Tracking and cells

We have already discussed the existing algorithm used for cell tracking. We have also discussed that it is capable of skipping gaps in detections for a certain amount of frames. In order to establish a baseline we evaluated this algorithm on our tracking dataset using frame skipping parameter set to 5. We can see the results of this baseline in Table 8. We used 300 frames of our tracking dataset to produce the results. We need to take into account that the annotated data has a Recall of 100% and it goes down to 95% for our detected data. This 5% decline amounts to having about 0.6 more holes on average within a track. The performance of the current algorithm is already showing enough potential to enable data gathering but our aim is to improve it further. We can see the visualization of these tracks in Figure 16

Table 8: Tracking performance results

Average track length	Pass-through		
Data source	First	Second	Fragment per track
Annotated	41.602/355	59.552/248	1 : 2.3
Detected	36.243/399	48.5/298	1 : 2.9

Values in each cell represent the average track length / track count. Track dataset has average track length 139.7 and 73 tracks total on 250 frames of the video. This amounts to about 2-3 holes per each track found by our algorithm.

5.1 Flow matrix

The first main issue with the current algorithm is its usage of a flow matrix. The problem with the flow matrix constructed from the existing tracks that have been detected so far is the scarcity and unevenness of data. The flow matrix even after interpolation does not have accurate data for each point. In order to solve this we setup a computational fluid dynamics (CFD) simulation of flow in the channel. For this one needs to provide the geometry of the channel and inlet velocity conditions. The channel geometry is readily available and the averaged inlet velocity can be obtained from the information of maximal fluid velocity in the channel, or using

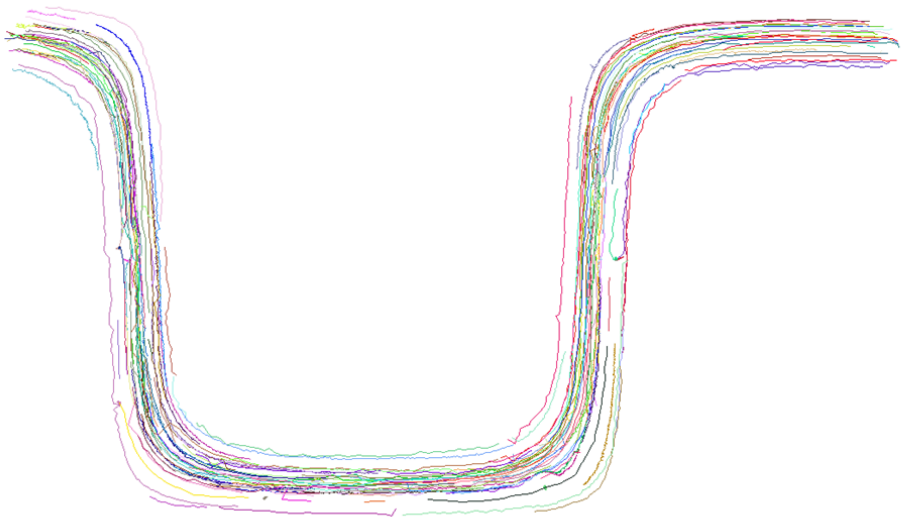


Figure 16: Example of tracks found in first 300 frames

the volumetric flow rate. The video data does not contain information about the volumetric flow rate, however, the maximal fluid velocity can be estimated using maximal velocities of the individual cells.

Our current results enabled us to extract this vital information. We extracted the maximum possible speed of cells inside our channel which turned out to be around 11 pixels per frame. This speed occurred in the direct middle of the channel.

We used the PyOIF module [4] within the ESPResSo package [3]. This module is capable of simulating the flow of red blood cells in a given microfluidic channel. We seeded the cell sizes roughly based on the occurrence of cell sizes in our dataset, which amounts to sizes between 25px and 35px.

The results of this simulation enabled us to gather a perfect flow matrix for experimentation within reasonable margin of error. In addition to a flow matrix we also gathered the positions of RBCs within the simulation with their cell size and velocity vector for further experimentation.

After evaluating our existing algorithm with this simulation flow matrix, the results were within 0.5 ±average length of a track compared to the flow matrix made from annotated / tracked data. That seems to indicate that the resulting

flow matrix from simulations does not impede the tracking algorithm in any way.

5.2 Neural network tracking

As the next step in improving our tracking performance, we have designed a prototype using CNNs. The idea is to take the resulting tracks from our second pass-through and try to join them with other existing tracks together. The neural network is designed to predict the location in the next frame more accurately than our physical model would.

The initial step is to convert our tracking dataset into a manageable format for our neural network. We have created 30x30x5 matrices out of our tracking dataset, centered at each given cell for which we are trying to find a segment in the future frame. First 2 channels contain the X and Y velocity of the cell respectively located at coordinates [15,15] and the rest being 0. The 3rd channel contains locations of all cell centers in the 30px x 30px vicinity from our cell. The last 2 channels contain a flow matrix of X and Y velocities for each of the 30x30 points. The velocities of in the 4 channels share a similar magnitude and were calculated from a fluid-only simulation. Our dataset contains 10058 samples for training.

The output of the neural network is two values, predicted X and Y velocity between the current and the next frame. The structure of the neural network is 12 convolutional layers with 32 filters each with 3x3 convolutions and utilizing ReLu after each. Then we have 1 pooling layer and at the end we have 1 fully connected layer which outputs our two values. For the cost function we use mean square error and Adam Optimizer for the training. We measure the accuracy of our model through the sum of absolute values of all errors after each epoch of training.

We were able to train the neural network successfully and validated it after segmenting our dataset into 3 parts: 60% for training, 20% for testing and 20% for validation. The initial test took 60% sequential data and used only those for training. This has proven inefficient, because the neural network provided bad results on the other two parts of our dataset. We approached this issue through random sampling our dataset before segmenting it into parts. This increased the

diversity of the data for the neural network and we were able to verify relatively the same performance on the latter two parts as well. Our current best performance is with error sum being at around 8 for a batch size of 32. The value is computed as a sum of all absolute values of differences between the predicted velocities and the dataset X and Y velocities. Going forward we will need to utilize this prototype in our tracking pipeline to see if the synthetic benchmark is indicative of being able to connect cells into segments using the CNN.

As a next step we trained the neural networks without the last 2 channels which represent the flow matrix. We wanted to see if the neural network performance on the training and testing dataset during training would be impacted in any way. We ran twice the amount of iterations during training and after watching how loss and error sum develops. The neural network was not even able to achieve good results on the training dataset, let alone the testing dataset. This leads us to conclude that the flow matrix is critical during training for extrapolation of information.

We also wanted to see what impact might dataset augmentation have. In an effort to add more data, we tested out rotating our data. Due to how our data is segmented for training our tracking neural network, it is not as trivial as just rotating image data clockwise by X degrees. On every rotation, all velocity vectors have to be adjusted based on the rotation. As a result we chose a simple scheme of rotating counterclockwise by 90 degrees and then doing a vector swap with sign change. The final formula is vector X becomes vector Y and vector Y becomes minus vector X. This increased our data by a factor of 4. This change negatively affected training speed. The model can still reach the same performance, or potentially better with more training, but it was generally slower by about 40%. On our synthetic benchmark we observed a minor mean error decrease in predicted velocity.

After this we used simulation RBC data for training the neural network model. We saw this as an eventual possibility of improving the diversity of our dataset. The simulation data could be acquired in a similar way to gathering a flow matrix. As a result, we had 6x more data from a single simulation than we had from our annotated dataset. On its own however, the data was not capable of predicting the movements in our videos. That being said, this might be an issue with the diversity

of the data provided by the simulation. After splicing together some samples from the simulation with our dataset, we observed no performance degradation and in the future on new videos this might provide more robust results.

As our last important step towards utilizing our new method, we had to integrate it into our existing tracking pipeline. The tracking neural network easily interfaces with it, due to velocities being readily available and flow matrix already being used by the algorithm. The neural network is capable of both looking into the future and into the past (frames with +1 and -1 indices respectively). We used it as a refinement step in the pipeline. On average, the current output of the tracking step produces tracks which have on average 2.3 holes. (Table 8) Our neural network was able to reduce that fragmentation of tracks to 1.8 holes per track, which amounts to average track length of 77 bounding boxes. This in of itself is a significant improvement in performance for our tracking algorithm.

Going forward, it shows promise to further tweak the neural network to complement the tracking algorithm even more. Other possibility is rechecking certain critical decisions in some tracks where overlap is common and using the neural network as a second judgement call with certain weight attached to it. For now the performance of the neural network, however, does not seem to indicate being able to remove the whole physical tracking algorithm in favor of solely using the neural network approach.

6 Validation with simulations

After gathering all possible data from videos we need to gather important metrics for validating simulations. The established output from the whole tracking pipeline is the list of all detected bounding boxes connected tracks, each track representing the lifetime of one cell in a given video. The first rudimentary metrics were already mentioned in the Dataset, width and height of an RBC. These are relatively easy to gather from the detection step already.

Next we are able to create a heat map of the occurrence of RBCs. This can be seen in Figure 17. The heat map gives us a detailed picture of where most RBCs flow within a given fluid flow. The heat map represents low occurrence (blue color) and high occurrence (red color) of RBCs in a given location. We can see for example in the second video due to the different nature of the obstacles there are almost no cells over the course of our 150 processed frames. This can be directly compared to a simulation to see whether the occurrence of cells is similar to the video. This is again created from the detection step.

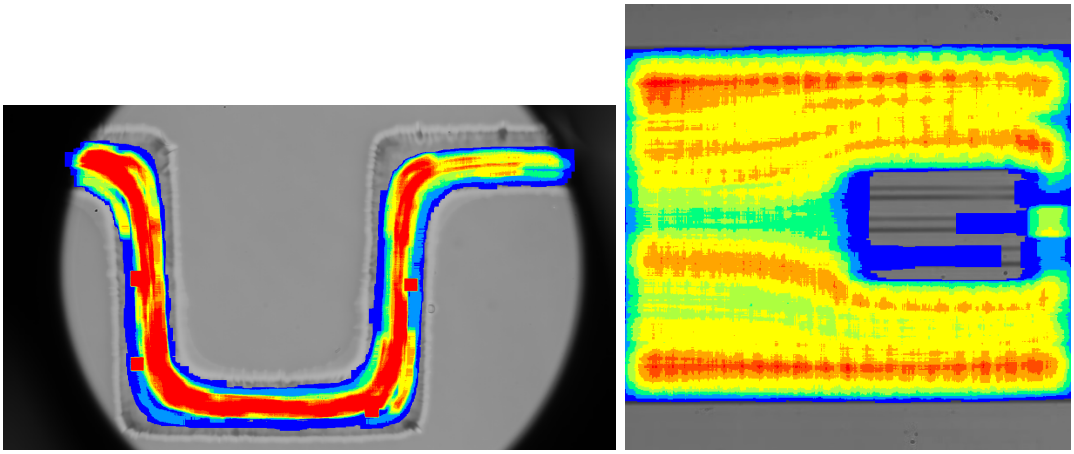


Figure 17: Heat map of the cell density in the channel. Left: First video, Right: Second video

Moving on to data gathered from cell tracking, here we have more information due to having the bounding boxes connected together. Firstly, we created a direction of movement visualisation. This can be seen in Figure 18. This particular output highlights the behaviour of cells around the squeezing obstacles. We can

see how most cells avoid them by flowing under or above the obstacle.

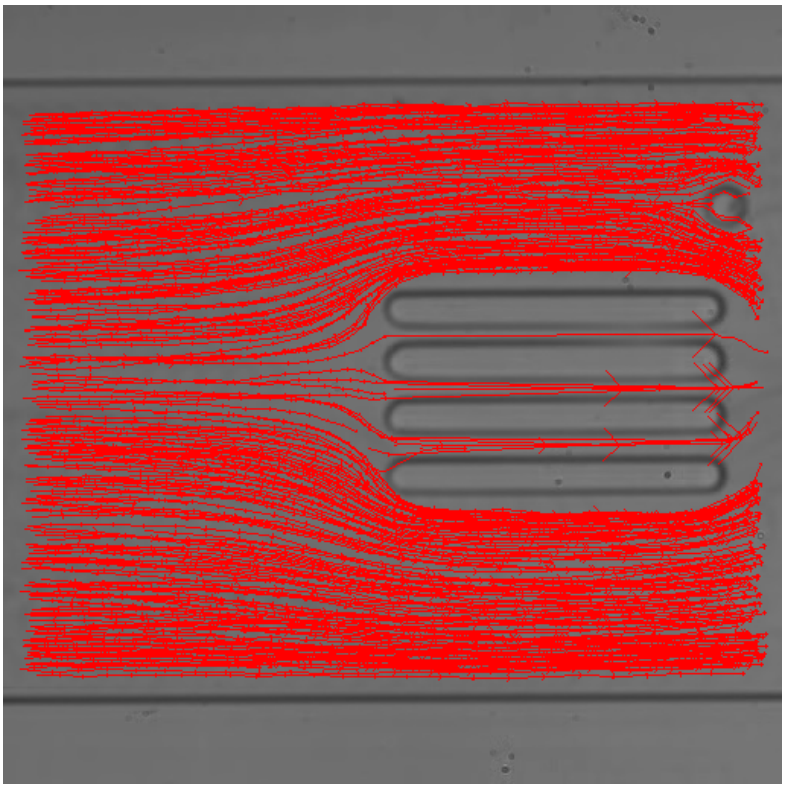


Figure 18: Flow visualisation of cell trajectories in second video.

Particularly of note are also the velocities of cells. Velocities can tell us whether the fluid is flowing the same way in the simulation, especially with added RBCs. We can designate regions within a given channel which are interesting for measurements, for example bends as seen on Figure 19. These bends are generally problematic even for the tracking algorithm. We can then construct velocity histograms from both the simulation and tracking output and directly compare their values.

Lastly, we are able to acquire partial 3D information from a 2D image. In the case of our second video that is not possible, because the microfluidic device is too thin for any overlap. However, in the case of the first video, we are able to discern the depth at which a cell is moving based on the velocity it travels at. The channel in question has highest fluid flow in the both horizontal and vertical middle and the cells slow down when near the boundaries of the microfluidic device. This is

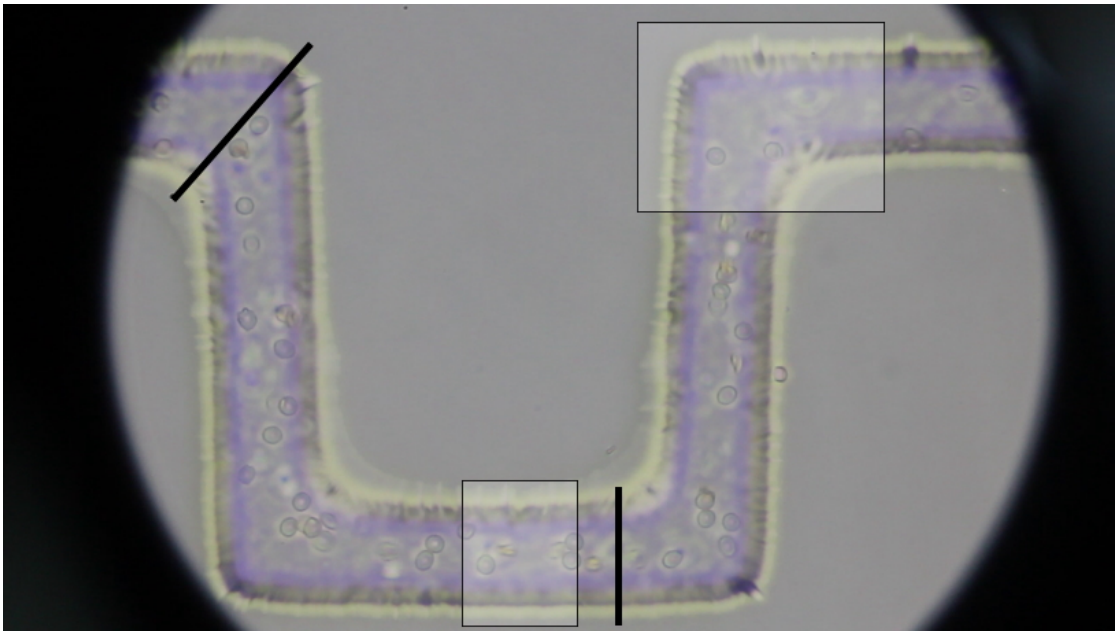


Figure 19: Two cross-sections (bold lines) and two regions (dashed) are indicated for evaluation of velocity histograms.

also critical information for calibrating a simulation since the fluid flow behaves the same way in a simulation and a single velocity value per pixel might not be enough.

7 Future goals and closing remarks

With the advent of new methods being available very frequently in the field of computer vision and specifically neural networks, future improvements to the data gathering show a lot of promise. In terms of detection, for the future use of the described approaches, further data gathering is imperative for any progress. It is however of note, that further incremental improvements are not trivial not only from the research and data gathering standpoint, but also from the software development standpoint. A dataset of relatively small size as was described in this thesis already requires a lot of data manipulation for any simple task. By introducing any further redundancies like is often the case, in our case with the usage of a flow matrix, the complexity increases. Any simplification such as the introduction of

already mentioned dense convolutional layers over normal convolutional layers is desirable. Exploring latest state of the art approaches to detection that could be combined with the detection framework capabilities described would be the most straightforward area of improvement next to the addition of more data.

In terms of tracking, the approach of using any form of neural network is relatively new and quite often is on a per-cell basis or in our case requires help from simulation software for certain steps. As such, further research shows a lot of promise in replacing or complementing traditional methods like physical models or particle filtering. Improvements on our proposed method lie mainly in the experimentation with different information components, since they are not as straightforward as changing layers example with detection.

After describing all of our metrics and data analysis methods, there is still a lot of potential in finding new ways of how to apply the existing data to simulations. One possible approach is the automation of future approaches, which would then lead to more data as well as more validated models. Potentially automating the simulation software to work in tandem with the tracking algorithm would enable experimentation with parameters of the flow matrix generation. All in all both this area of study as well as the improvement of microfluidic devices simulations shows a lot of promise going into the future.

Conclusion

In this thesis we went over several topics, which all deal with the automated gathering of data from videos of RBCs. We performed an analysis of existing solutions for detection and tracking problems. We then went over current approaches to detection and tracking of RBCs in videos and presented improvements through the application of state-of-the-art algorithms. We also delved into the topic of data extraction through several metrics for immediate simulation validation.

In the first chapter we established a common terminology for topics in Computer Vision. We went over how most traditional and modern methods work and what is required to make them work.

In the second chapter we went over the previous research done in the field of tracking RBCs in videos. We established both the methods used world-wide but also used by the Cell-In-Fluid team specifically for validation of simulations. These methods included an existing Hough Transform method and a physics-based model for tracking.

In the third chapter we introduced our dataset and provided statistics of what data it contains. We also noted the particular problems that occur when a new dataset is composed.

The fourth chapter deals with convolutional neural networks and their application to RBC detection. We described a proof of concept, which was capable of correctly classifying RBCs in images despite having poor performance. Next, we evaluated several existing frameworks for object detection and selected one which provided the best results for RBC detection on our dataset. This chapter also remarks upon the particular differences between various convolutional neural network architectures.

In chapter five we go over the existing tracking algorithm and suggest several improvements for improving its performance. We describe a method which utilizes convolutional neural networks for reducing the amount of gaps in detected tracks. The chapter also describes several issues when designing the inputs for training such a model.

Chapter six summarizes the proposed methods and suggest future improvements in the field and highlights some obstacles which prevent further improve-

ment.

All in all, the tracking pipeline is proving to be a valuable asset for acquiring large volumes of data from real-world experiment videos. The development of this pipeline and its algorithms is not final and shows future promise for both its performance and its utilization.

References

[1] K.-K. Kleineberg, L. Buzna, F. Papadopoulos, M. Boguñá, and M. A. Serrano, "Geometric correlations mitigate the extreme vulnerability of multiplex networks against targeted attacks," Phys. Rev. Lett., vol. 118, p. 218301, 2017.

[2] J. Janacek, M. Kohani, M. Koniorczyk, and P. Marton, "Optimization of periodic crew schedules with application of column generation method," Transportation Research Part C: Emerging Technologies, vol. 83, pp. 165 – 178, 2017.

[3] F. Weik, R. Weeber, K. Szuttor, K. Breitsprecher, J. de Graaf, M. Kuron, J. Landsgesell, H. Menke, D. Sean, and C. Holm, "Espresso 4.0 – an extensible software package for simulating soft matter systems," The European Physical Journal Special Topics, vol. 227, no. 14, pp. 1789–1816, Mar 2019.

[4] I. Cimrák, M. Gusenbauer, and I. Jančigová, "An ESPResSo implementation of elastic objects immersed in a fluid," Computer Physics Communications, vol. 185, no. 3, pp. 900–907, 2014.

[5] M. Calder and et al., "Computational modelling for decision-making: where, why, what, who and how," R Soc Open Sci, vol. 5, no. 6, 2018.

[6] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," CoRR, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993

[7] I. Jančigová and I. Cimrák, "A novel approach with non-uniform force allocation for area preservation in spring network models," AIP Conference Proceedings, vol. 1648, no. 1, pp. –, 2015. [Online]. Available: http://scitation.aip.org/content/aip/proceeding/aipcp/10.1063/1.4912489

[8] M. Dao, J. Li, and S. Suresh, "Molecularly based analysis of deformation of spectrin network and human erythrocyte," Materials Science and Engineering C, vol. 26, pp. 1232–1244, 2006.

[9] D. Fedosov, B. Caswell, A. Popel, and G. Karniadakis, "Blood flow and cell-free layer in microvessels," Microcirculation, vol. 17, pp. 615–628, 2010.

[10] J. Illingworth and J. Kittler, "The adaptive hough transform," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-9, no. 5, pp. 690–698, Sep. 1987.

[11] H. K. Yuen, J. Illingworth, and J. Kittler, "Ellipse detection using the hough transform," 01 1988.

[12] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01, pp. 886–893, 2005. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2005.177

[13] S. U. Akram, J. Kannala, L. Eklund, and J. Heikkilä, "Cell tracking via proposal generation and selection," CoRR, vol. abs/1705.03386, 2017. [Online]. Available: http://arxiv.org/abs/1705.03386

[14] Y. Wang, H. Mao, and Z. Yi, "Stem cell motion-tracking by using deep neural networks with multi-output," Neural Computing and Applications, pp. 1–13, 2017.

[15] V. Ulman, M. Maška, K. Magnusson, O. Ronneberger, C. Haubold, N. Harder, P. Matula, P. Matula, D. Svoboda, M. Radojevic, I. Smal, K. Rohr, J. Jaldén, H. Blau, O. Dzyubachyk, B. Lelieveldt, P. Xiao, Y. Li, S.-Y. Cho, and C. Ortiz-de Solorzano, "An objective comparison of cell tracking algorithms," Nature Methods, vol. 14, 10 2017.

[16] P. Domingos, "A few useful things to know about machine learning," Commun. ACM, vol. 55, p. 78–87, 10 2012.

[17] S. ArunaN and S. Hariharan, "Edge detection of sickle cells in red blood cells," International Journal of Computer Science and Information Technologies, Vol. 5(3), 2014.

[18] M. Maitra, R. Gupta, and M. Mukherjee, "Detection and counting of red blood cells in blood cell images using hough transform," International Journal of Computer Applications, vol. 53, pp. 13–17, 09 2012.

[19] S. Khan and A. Khan, "An accurate and cost effective approach to blood cell count," International Journal of Computer Applications, vol. 50, pp. 975–8887, 08 2012.

[20] P. Viola and M. Jones, "Robust real-time object detection," International Journal of Computer Vision - IJCV, vol. 57, 01 2001.

[21] A. Dufour, R. Thibeaux, E. Labruyère, N. Guillen, and J. Olivo-Marin, "3-d active meshes: Fast discrete deformable models for cell tracking in 3-d time-lapse microscopy," Image Processing, IEEE Transactions on, vol. 20, pp. 1925 – 1937, 08 2011.

[22] M. Ka, O. K, S. Garasa, A. Rouzaut, A. Muñoz-Barrutia, and C. Ortiz-de Solorzano, "Segmentation and shape tracking of whole fluorescent cells based on the chan-vese model," IEEE transactions on medical imaging, vol. 32, 01 2013.

[23] E. Turetken, X. Wang, C. Becker, C. Haubold, and P. Fua, "Network flow integer programming to track elliptical cells in time-lapse sequences," IEEE Transactions on Medical Imaging, vol. PP, pp. 1–1, 12 2016.

[24] M. Schiegg, P. Hanslovsky, C. Haubold, U. Köthe, L. Hufnagel, and F. Hamprecht, "Graphical model for joint segmentation and tracking of multiple dividing cells," Bioinformatics (Oxford, England), vol. 31, 11 2014.

[25] R. Bensch and O. Ronneberger, "Cell segmentation and tracking in phase contrast images using graph cut with asymmetric boundary costs," Proceedings - International Symposium on Biomedical Imaging, vol. 2015, 04 2015.

[26] C. Reyes-Aldasoro, S. Akerman, and G. Tozer, Measuring Red Blood Cell Velocity with a Keyhole Tracking Algorithm. Springer Verlag, 01 2007, pp. 810–813.

[27] I. Smal, K. Draegestein, N. Galjart, W. Niessen, and E. Meijering, "Particle filtering for multiple object tracking in dynamic fluorescence microscopy images: Application to microtubule growth analysis," IEEE Transactions on Medical Imaging, vol. 27, no. 6, pp. 789–804, June 2008.

[28] P. Matula, M. Maška, D. Sorokin, P. Matula, C. Ortiz-de Solorzano, and M. Kozubek, "Cell tracking accuracy measurement based on comparison of acyclic oriented graphs," PloS one, vol. 10, p. e0144959, 12 2015.

[29] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," CoRR, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[30] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: object detection via region-based fully convolutional networks," CoRR, vol. abs/1605.06409, 2016. [Online]. Available: http://arxiv.org/abs/1605.06409

[31] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," CoRR, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[32] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," CoRR, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[33] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," CoRR, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[34] Z. Kalal, J. Matas, and K. Mikolajczyk, "P-n learning: Bootstrapping binary classifiers by structural constraints," Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 238, pp. 49 – 56, 07 2010.

[35] X. Jia, H. Lu, and M. Yang, "Visual tracking via adaptive structural local sparse appearance model," 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1822–1829, June 2012.

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Commun. ACM, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. Available: http://doi.acm.org/10.1145/3065386

[37] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," CoRR, vol. abs/1310.1531, 2013. [Online]. Available: http://arxiv.org/abs/1310.1531

[38] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," CoRR, vol. abs/1311.2524, 2013. [Online]. Available: http://arxiv.org/abs/1311.2524

[39] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," CoRR, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," CoRR, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[41] K. F., "Parallel implementation of feature descriptor for object detection using adaboost." [Master thesis] University of Žilina, Faculty of Management Science and Informatics, Department of Mathematical Methods and Operations Research, 61 p., 2017.

[42] F. Mučka, "Algorithms and their implementation for analysis and image processing from recordings of biological experiments," [Master thesis] - University of Žilina. Faculty of Management Science and Informatics. Department of Software Technologies, Žilina, 61 p., 2017.

[43] J. Tomášiková, "Processing and analysis of videosequences from biological experiments using special detection and tracking algorithms," [Master

thesis] - University of Žilina. Faculty of Management Science and Informatics. Department of Software Technologies, Žilina, 63 p., 2017.

[44] G. Mazza., 2017.

[45] C. T. et all, 2016.

[46] [Online]. Available: lakshaysuri.wordpress.com

[47] I. Icke. [Online]. Available: commons.wikimedia.org

[48] C. de Souza. [Online]. Available: www.codeproject.com

[49] M. SIRSAT. [Online]. Available: https://manisha-sirsat.blogspot.com/2019/04/confusion-matrix.html

Publications

[50] F. Kajánek and I. Cimrák, "Evaluation of detection of red blood cells using convolutional neural networks," in 2019 International Conference on Information and Digital Technologies (IDT), June 2019, pp. 198–202.

[51] ——, "Advancements in red blood cell detection using convolutional neural networks," in Proceedings of the 13th International Joint Conference on Biomedical Engineering Systems and Technologies - Volume 3: BIOINFORMATICS,, INSTICC. SciTePress, 2020, pp. 206–211.

[52] T. Poštek., F. Kajánek., and M. Ondrušová., "Detection of lattice-points inside triangular mesh for variable-viscosity model of red blood cells," in Proceedings of the 13th International Joint Conference on Biomedical Engineering Systems and Technologies - Volume 3: BIOINFORMATICS,, INSTICC. SciTePress, 2020, pp. 212–217.

[53] F. Kajánek, I. Cimrák, and P. Tarábek, "Automated tracking of red blood cells in images," in Bioinformatics and Biomedical Engineering, I. Rojas, O. Valenzuela, F. Rojas, L. J. Herrera, and F. Ortuño, Eds. Cham: Springer International Publishing, 2020, pp. 800–810.

[54] H. Bachratý, K. Bachratá, M. Chovanec, F. Kajánek, M. Smiešková, and M. Slavík, "Simulation of blood flow in microfluidic devices for analysing of video from real experiments," in Bioinformatics and Biomedical Engineering, I. Rojas and F. Ortuño, Eds. Cham: Springer International Publishing, 2018, pp. 279–289.

[55] F. Kajánek, "Tracking of red blood cells in videos of experiments," in Proceedings of the MIST conference 2018. CreateSpace Independent Publishing Platform, 2018, pp. 1–5.

[56] ——, "Red blood cell detection using convolutional neural networks," in Mathematics in science and technologies: proceedings of the MIST conference 2019. CreateSpace Independent Publishing Platform, 2019, pp. 40–44.