Universiteit Leiden
The Netherlands

Opleiding Informatica & Wiskunde

Automatic Action Detection in Cell Imaging

Chika Opdam

Supervisors:
Daan Pelt (LIACS) & Roeland Merks (MI)
Daily Supervisor:
Tessa Vergroesen (IBL)

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl
Mathematical Institute (MI)
www.universiteitleiden.nl/en/science/mathematics

13/12/2023

Abstract

In this thesis, we introduce a two part sequential program which attempts to both track cells visible within a dataset of frames of microscopic cells. The program also aims to find the actions each cell performs within this dataset, noting which cell performed what action, and when this action was performed. We use an object detection network, to detect the cells within each frame and find the path each cell has walked. Using these paths, we look at each cell individually with an action recognition network to find the actions performed. We have created our own annotated dataset of cell actions and cells with their locations to be able to train and test the networks used in this project. We have found our approach to action detection and cell tracking to be fairly inaccurate and unreliable, particularly when dealing with data containing a large number of densely distributed cells. As a result, we propose different strategies to enhance our program and make it more reliable.

Contents

1 Introduction

Microscopic video data analysis plays a crucial role in various scientific domains, particularly in the field of cell biology. The ability to accurately detect and track cellular actions from these videos can provide valuable insights into the behavior and functioning of cells. However, manual analysis of these videos is a time-consuming and labour-intensive task, prone to human error and subjectivity. Therefore, there is a growing need for automated methods that can effectively detect and classify actions in microscopic video data.
In this bachelor thesis conducted at LIACS and the MI under the supervision of Dr. D.M. Pelt, Prof. Dr. R.M.H. Merks and T.M. Vergroesen MSc we investigated the application of machine learning in microbiological video data analysis to determine the viability of using this tool for this purpose.

1.1 Problem Statement

In this thesis, we explore the application of neural networks for automatic action detection in microscopic video data of cells. By leveraging the power of deep learning techniques, we aim to develop an efficient and reliable system that can accurately identify and classify various cellular actions, thereby reducing the burden of manual analysis and enabling researchers to extract meaningful information from large-scale video data.
We developed a two-part consecutive program which aims to achieve this goal. In the first program, we use the Fast R-CNN object detection network[20] to find cells within a given image sequence. Using the Intersection over Union (or Jaccard Index) [11] as a sort of reverse distance measure, we link the cells of two consecutive frames in such a way that the combined distance between the cells is as small as possible, essentially creating an assignment problem [19]. Repeating this process over multiple frames gives us a path following each cell. We tried experimenting with optical flow [5] to predict the location of cells in consecutive frames in order to increase tracking accuracy.
The second program takes the tracking information from the first program and uses the C3D[24] network to perform action recognition over video clips of each individual cell. This gives us a list of actions each cell has performed and when these actions were performed.

1.2 Thesis overview

This thesis is organised as follows:
Section 1 Provides an overview of the research problem and the structure of the thesis.
Section 2 Discusses research and algorithms necessary for understanding this thesis.
Section 3 Describes the research methodology and gives a technical explanation of the proposed neural network based system for action detection in cells.
Section 4 presents an evaluation and analysis of developed system.
Section 5 Provides a discussion of the findings and the limitations of the program.
Section 6 Discusses future research directions and concludes the thesis.

2 Background

2.1 Cell Tracking

Cell tracking is a fundamental aspect of biology, crucial for understanding cell behaviour in various environments. In microscopic video data of cells where only a few cells are visible, it relatively easy and straightforward to perform cell tracking. But given data where a few dozen, or even hundreds of cells are visible in each frame, tracking cells becomes a very tedious and time-consuming task. For this reason, a lot of research has been done on automated cell tracking programs ([22, 18, 13]), Emami et al. discusses different approaches to cell tracking in [3]. We take the tracking-by-detection approach[9], where we first detect the cells visible in each frame, and then attempt to link cells across each frame to track their movement over time.

2.2 Optical Flow

To be able to link cells across multiple frames, we can try to predict the location of the cells in the next frame; this can be done by using optical flow. Optical flow is an estimation of velocity of objects between image sequences, allowing to guess where objects will be in future frames. This is done by taking the intensity of each pixel at a specific point in time, denoted as $I(x, y, t)$. The assumption is that the intensity of an object will not change over time, thus an object at point x, y at time t will have moved to $x + \delta x, y + \delta y$ at time $t + \delta t$, and we assume $I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t)$. By taking the-first order Taylor expansion, we obtain the equation

$$\frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t = 0$$

Which simplifies to

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0$$

Here V_x, V_y represent the optical flow of $I(x, y, t)$. Solving for the optical flow can be done in multiple ways[27, 1, 5]. We will make use of the Farneback algorithm proposed by Farneback [5] which looks at the intensity changes of a neighbourhood of each pixel to calculate the magnitude and direction of flow.

2.3 Assignment Problem

To actually link cells across frames, we make use of the assignment problem. The assignment problem [19] is a combinatorial optimization problem where, given a number of agents and tasks, we want to assign each task to an agent such that the total cost required to perform all tasks is minimal. There has been a lot of research on various methods for the assignment problem, several of these methods are compared by Ramshaw and Tarjan in [2].
In our case, we consider the cells in a certain frame as the agents and the distance to a cell in the next frame as the cost. We want to assign each cell in the next frame to a cell in the current frame. For this task, we use the Hungarian algorithm [8].

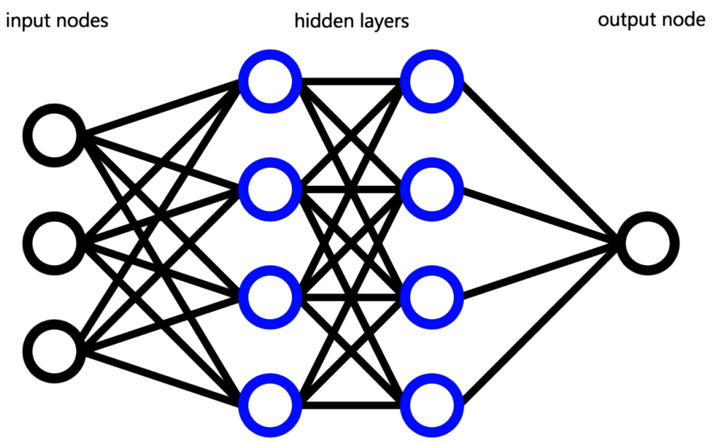


Figure 1: A visual representing the structure of NNs

2.4 Intersection over Union

The Jaccard Index, also known as the Intersection over Union, is a metric used to assess the similarities between sample sets [11]. When given two sample sets A and B, the Jaccard Index is defined as

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

Resulting in a value between 1 and 0 where 1 means that $A = B$ and 0 means that $A \cap B = \emptyset$. Since we measure the similarity of two sets, it is conventional to take $J(A,B) = 1$ when $A = B = \emptyset$ even though $A \cap B = \emptyset$.
This allows us to measure the accuracy of a calculated prediction compared to the given truth. In case of two areas in an image, say area A and area B, we can simply take the area of the overlap of A and B, divided by the area of the union of A and B.
In this thesis we refer to this measure as the Intersection over Union (IoU)

2.5 Neural Networks

Neural networks (NNs) are powerful tools used for machine learning tasks, including image processing tasks. The structure of these NNs is inspired by the structure of a brain, using artificial neurons and layers to extract meaningful patterns and features from data and allowing the NN to draw conclusions on the inputs given to the network. Therefore NNs are well-equipped for processing visual data, predicting what is depicted in said visual data and labelling it accordingly.
An NN consists of an input, several hidden layers, and an output, where each layer consists of a certain amount of nodes (Figure 1).
A single node takes a certain number of inputs, makes some calculations, and produces an output which is then fed to nodes in the next layer as an input value (Figure 2). Through this process each node detects certain patterns or features found in the output of the previous layer.
If we consider the j'th node in a certain layer of an NN with n input values, we can express the

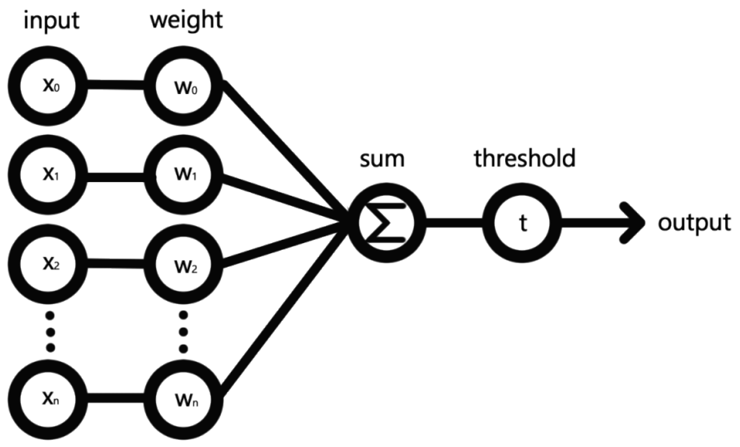


Figure 2: A visual of the structure of a node with n inputs

formula for the output of the node as

$$g_j(\overline{x}) = \max\{\overline{x} \cdot w_j + b_j, 0\} \tag{2}$$

Here we take $\overline{x} = (x_1, ..., x_n)$ to be the inputs of the node, $w_j = (w_{j,1}, ..., w_{j,n})$ the pre-determined weights and b_j the bias of this specific node. The weights are essential to make sure that each node is able to properly extract patterns from the input. They are adjustable parameters which determine the strength of the connection between nodes in different layers. Each weight essentially represents the importance of the input connected to it and determines how much each input contributes to the neuron's output.

After we take the inner product of the inputs and the weights, we add a bias value which determines how strong the output signal of a node must be for the node to activate and send its output to the next layer. We feed this value to an activation function. In our case the used activation function is the rectified linear unit (ReLU) as suggested by Hington and Nair [17] which simply checks whether the input value is greater than zero or not, as seen in (2).

Given that an NN feeds information sequentially through its layers consisting of these nodes, we can write the output of an NN ($f(x)$) with N layers, given an input vector x as

$$f(x) = G^N \circ G^{N-1} \circ \cdots \circ G^1(x)$$

Where $G^i(x) = (g_1^i(x), \ldots, g_j^i(x))$ is the output function for the i'th layer, and g_j^i the node function for the j'th node in the i'th layer as described in (2). For ease of notation, we say that if x_k is not an input for a specific node, that the weight w_k for this node is equal to 0.

We can see how the output of an NN heavily depends on the choice of weights and biases at each node. Since a bias is essentially a weight where the input value is always 1, we will from now on treat the bias to be part of the set of weights.

If we have a dataset that is already labeled, referred to as a training set, we can determine how accurate an NN is by comparing the predicted output of the NN $f(x)$ with the label corresponding to the given input y. From now on we will denote y as the expected output of x. This is done by means of a loss function $L(x, y) = l$ which calculates the loss (or error) l and thereby quantifies the difference between the predicted output and the actual label, providing a measure of how well the NN is performing.

Naturally, we want to set the weights such that l is as small as possible. We achieve this by training the NN by means of backpropagation[10]. Backpropagation is done in a couple of steps:

1. *Initialization*: We first initialise the NN with random weights (and thus also biases) and feed it the training set.
2. *Loss Computation*: We then compute the loss of the output using the loss function. Here, the input and desired output are fixed while the weights are adjustable variables.
3. *Gradient Calculation*: To determine how the weights influence the output and set them to minimise loss, we calculate the gradient of the loss function with regards to w. The gradient, denoted as $\nabla L(w)$ is represented as a vecotor $= \vec{c}$, where the gradient ∇ is defined to be

$$\nabla f(\vec{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} f(x) \\ \frac{\partial f}{\partial x_2} f(x) \\ \dots \\ \frac{\partial f}{\partial x_n} f(x) \end{bmatrix}$$

   This vector $\vec{c}$ indicates what changes to the weights cause the most significant change to the value of the loss function.
4. *Weight Adjustment*: Since we want the loss function to be as small as possible, we will adjust the weights with some small multiple of $-\nabla L(w)$ to try and find a smaller value of L.
5. *Repetition*: Repeating steps 2-4 multiple times provides us an estimation of the minimum value of L with associated values for the weights and biases.

Afterwards the trained NN can be used to label unlabelled data.
Our program intends to track cells in a sequence of frames and determine the actions these cells perform. For this we make use of two types of neural networks: an action recognition neural network which has a video of certain size as input and returns the action performed within this video, and an object detection network which has an image as input and returns a set of boxes around objects the network finds within this image. Both of these networks are so-called convolutional networks, which use convolution between an input matrix and some kernel matrix to break the input down into smaller sections to analyse those before taking the results of these smaller sections to make an overall conclusion about the entire input.

2.5.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [21, 10] are often used to capture spatial information. This is done by breaking an image down into smaller areas and recognising certain patterns within these smaller areas, from which we conclude what is visible in the entire image. The breaking down of an image and finding certain patterns is done by means of convolution, therefore the layers in which this happens are called convolutional layers.
Convolution is a mathematical operation expressing how the shape of two functions (in our case matrices) impact each other. Convolutional layers consist of an $n \times n$ input matrix (X), an $n' \times n'$ kernel matrix (K) where $n' < n$, and an activation function.

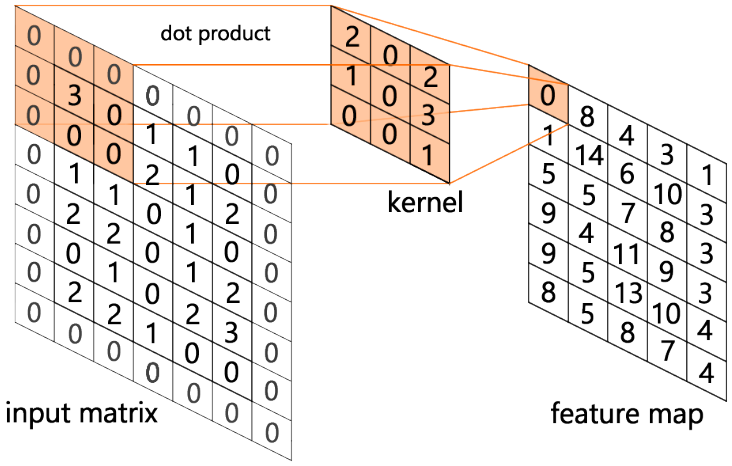


Figure 3: Visual representation of the working of a convolutional layer. Zero padding, which is the action of extending the input matrix with a border of elements of value zero, is used to make sure the feature map has the same dimensions as the input matrix.

The output of a convolutional layer is calculated by taking the two dimensional discrete convolution of X and K which is given by

$$o_{i,j}(X) := (X \star K)(i,j) = \sum_{l=1}^{n'} \sum_{m=1}^{n'} x_{i-l,j-m} \cdot k_{l,m}$$

Here, $o_{i,j}(X)$ is the value of the output matrix O at position (i,j) when given the input matrix X. O is an $n \times n$ matrix called a feature map, where each element $o_{i,j}$ in O contains information about the $n' \times n'$ area around $x_{i,j}$ from the input matrix X. A visual representation of the convolution operation between two matrices can be found in Figure 3. Note that we are essentially using a sliding window, taking the dot product of the kernel matrix and an $n' \times n'$ window over the input matrix.
In terms of a neural network layer, we consider the kernel matrix to be the weights shared by each node in the same layer, and the input of each node is then an $n' \times n'$ submatrix of the input matrix. Thus we find the node function for the node (i,j) to be

$$g_{i,j}(X) = max\{o_{i,j}(X) + b, 0\} \tag{3}$$

Where X is the input of the convolutional layer and b the threshold value which allows us to write the resulting feature map as

$$G = \begin{bmatrix} g_{1,1}(X) & g_{1,2}(X) & \cdots & g_{1,n}(X) \\ g_{2,1}(X) & g_{2,2}(X) & \cdots & g_{2,n}(X) \\ \vdots & \vdots & \ddots & \vdots \\ g_{n,1}(X) & g_{n,2}(X) & \cdots & g_{n,n}(X) \end{bmatrix} \tag{4}$$

Since patterns crucial to recognising objects or actions can be found at different scales of an image, we want to study our input at different scales. This is done by downsampling in between different

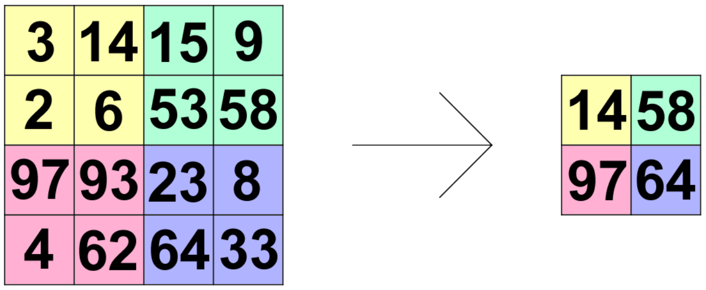


Figure 4: An example of max pooling with pooling size 2 where the maximum value of each 2×2 section of the input matrix will be the value representing this area in the down sampled output matrix

convolutional layers, essentially lowering their resolution. This allows us to take a convolution over a bigger area without increasing computational cost for each layer. The downsampling is done using max pooling layers which take a small subarea of a matrix and take the maximum value to be the new value representing this entire area as seen in Figure 4.
After alternating convolutional layers and pooling layers, we end with a fully connected layer in which every node takes the entire previous layer as input to obtain a final output. During the training of a CNN, the network aims to find the best kernel values for its task given a training dataset. The kernel size and the pooling size are set before training starts and do not change throughout training.

2.5.2 Object Detection

Object Detection is the task of finding objects within an image and labeling what these objects are. Unlike with instance segmentation tasks, where for each object the program aims to find which pixels are and are not part of the object, object detection simply provides boxes around each object found. A survey on various object detection models using neural networks has been conducted by Zaidi and Ansari et al [26]. For our purpose of object detection, we use a network called Faster Region-based Convolutional Network (Faster R-CNN), proposed by Ren et al. [20].

Faster R-CNN Faster R-CNN consists of two components; a region proposal network (RPN) which takes the input image and outputs a set of boxes called region proposals where objects might be, and the region classification network Fast R-CNN [6], which takes an image together with region proposals and classifies the object found within each proposed region (if any).
The RPN takes a sliding window over a generated feature map to find a set of region proposals where each proposal contains the coordinates of the bounding box of this region, and the estimated probability of the region containing an object and not containing an object. This is done by evaluating a set of anchor boxes at each sliding window position, where the anchor boxes are set by a predetermined set of scales and aspect ratios. Thus if we decide to use s different scales and n aspect ratios, we will have sn anchors for each window position. These boxes are resized and grouped together based on their IoU and their probability of containing an object which results in the outputted region proposals.

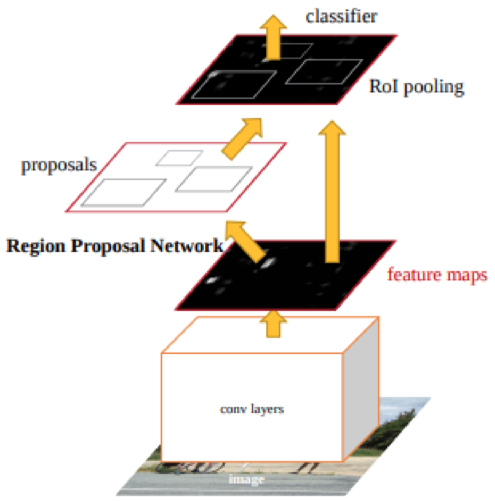


Figure 5: feature maps generated from convolutional layers are both used to generate region proposals, as well as classify what is visible within these regions (Ren et al. [20])

Fast R-CNN is a region classification network. It takes the convolutions of each proposed region given to the network and predicts the probability of certain objects being contained within these regions.
Faster R-CNN is build in such a way that generated feature maps are simultaneously used by the RPN and the Fast R-CNN, decreasing computation by sharing convolution computations over both networks as is illustrated in Figure 5.

2.5.3 Action Recognition

Action Recognition is the task of recognising what action is occurring in a short video and labeling said video accordingly. This can also be done by using convolutional networks. A lot of research has been done on video recognition, a study on multiple papers on this topic done by Zhu and Li et al can be found in [28]. One of the neural networks designed for the task of action recognition is the C3D network.

C3D To be able to detect actions visible in a video clip or series of frames, we not only need to be able to recognise spatial data and extract spatial features, but we also need to be able to extract temporal features. The Convolutional 3 Dimensional network (C3D network), proposed by Tran et al. [24] does this by expanding the kernel of a CNN to have three dimensions. This allows each node in the output to also say something about what was visible in the previous and next frames. If we take the depth of the kernel to be smaller than the number of input frames, we can slide the kernel over space and time calculating the 3 dimensional convolution between the kernel and input to capture both spatial and temporal information of the video as seen in Figure 6. Another option would be to have the kernel have the same depth as the input matrix, but this has the disadvantage that we cannot slide over time, which results in our output being two dimensional. This means that after the first convolution, temporal information will be lost, while using 3D convolution will preserve temporal information over multiple convolutions. The output of the C3D network is a list

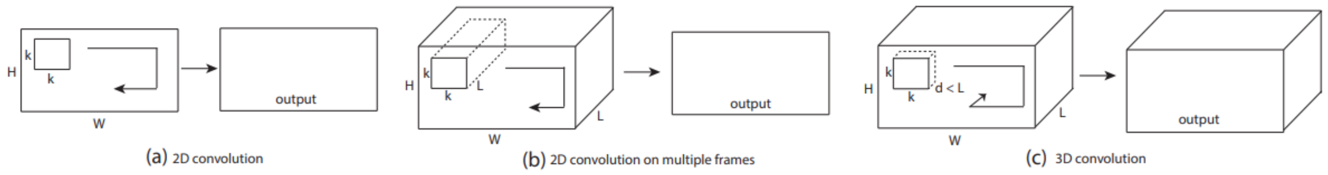


Figure 6: (a) 2D convolution over an image gives a two dimensional output (b) applying 2D convolution over multiple frames gives a two dimensional output (c) 3D convolution over multiple frames gives a three dimensional output (Tran et al. [24])

of n values for the n possible actions the network is trained to recognise, where each value gives the certainty of the network that the given input video depicts the output corresponding to this coordinate in the output.

2.5.4 Action Detection

While action recognition focuses on identifying a single action occurring in a short video, action detection has the task of taking a longer video, and finding when certain actions took place and what actions these are. This can be considered as a temporal version of object detection where, instead of region proposals to find locations of objects, temporal proposals are generated to find the time intervals of actions. There are multiple methods that, similarly to the RPN network discussed in 3.2, analyse multiple predetermined overlapping temporal windows of different scales. These windows are later combined and resized into action proposals [23, 7, 4]. A temporal proposal model like BSN [12] determines at each temporal location the probability of said location being the start of an action, the end of an action, inside an action instance, or outside an action instance. Using this information, they generate temporal proposals by combining moments with high starting probability to moments with high ending probability, where the moments in between have a high chance of being inside an action instance. A survey on action detection methods has been done by Vahdani and Tian [25]. In this thesis, we will instead use a sliding window technique to determine when actions took place.

3 Methodology

For this project, we developed two programs: one which uses Faster R-CNN to track cells over multiple frames and outputs this information, and one program which takes this output to perform action recognition on each cell using the C3D network. In this section, we will delve into how these programs were constructed and how the networks used in them were trained.

3.1 Object Tracking

The object tracking program performs object detection on each frame of the input data and attempts to link every cell found in a frame to a cell found in the next frame of data. To store the tracking data, we wrote a tracker class which stores the id number and location history of a single cell along with the frame in which this cell was first detected. This tracking data is stored in a .pkl file to be read by the action detection program.

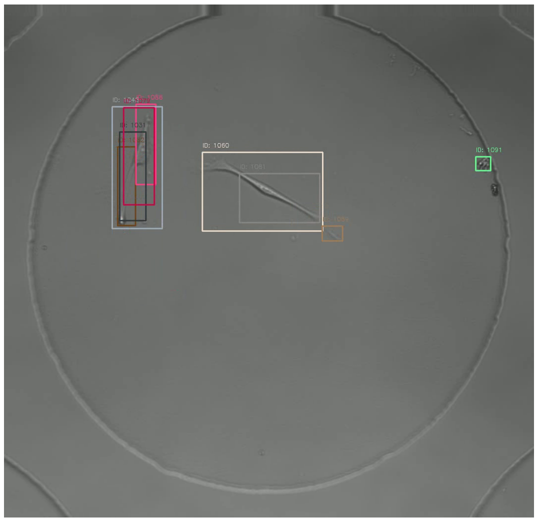


Figure 7: A frame of the output of Faster R-CNN with as input the BF-C2DL-MuSC-01 dataset where we can see that a lot of bounding boxes refer to the same cell

3.1.1 Object Detection

Object detection over each frame is performed using the Faster R-CNN model from the mmdetection Python framework [16]. The output consists of a list of bounding boxes that encase each cell found in the current frame, together with an object score indicating how confident the program is that this box contains a cell.
If left unmodified, the output of the Faster R-CNN model occasionally results in multiple cells being detected where only one cell exists, or cells being found where there are actually none (as shown in Figure 7). To address these errors, experiments were conducted to determine when two bounding boxes refer to the same cell and combining these boxes in such a way that each cell only has one bounding box referring to it. We did this by assuming that when a bounding box is completely contained within another bounding box, one of the two boxes is redundant. We also tried looking at the IoU of each set of boxes, where we assumed that if two bounding boxes are very similar, they must refer to the same cell. When the program determines that two boxes refer to the same cell, we want to keep the bounding box which best encompasses the cell and discards the other. This is done by keeping the bounding box with the larger object score. Lastly, to remove the false positive cell findings, we experimented with removing bounding boxes with small object score. After this, we are left with a list of bounding boxes which supposedly tells us the location of each cell in a certain frame. We define L_n to be the list obtained from frame n.

3.1.2 Tracking

Once the list of bounding boxes L_n is obtained, we need to update the trackers to store one of the L_n boxes as the next location where the tracked cell is found. If the current frame is the first frame of the sequence, a new tracker is created for each cell found in this frame. Otherwise, the program calculates the distance of each box in L_n to each box of the cells found in the previous frame.
This distance is determined by the IoU. If we consider two boxes $B_i \in L_{N-1}, B_j \in L_N$ in frame

$N-1$ and N respectively, we calculate the distance to be

$$d_{i,j} = -\frac{B_i \cap B_j}{B_i \cup B_j}$$

Using this we obtain a matrix of distances

$$D_{N,N+1} = \begin{pmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,m} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n,1} & d_{n,2} & \cdots & d_{n,m} \end{pmatrix}$$

Using this matrix, we want to link each cell tracked in frame $N-1$ to a cell in frame N such that the distances between these cells are as small as possible. Therefore creating an assignment problem where our goal to minimize the total distance of assigned cells in consecutive frames. This is done using the Hungarian algorithm [8].
We want to take into account the possibility of new cells appearing and other cells disappearing from view. In order to do this, we check each assigned pair to see if their IoU> 0. If not, it means that the cell in the previous frame frame is assigned to a region box it does not overlap with. In this case, it is assumed that this means the tracked cell and the cell it has been assigned to in the next frame are two different cells. This assigned pair is then removed from the list of assignments obtained using the Hungarian algorithm, and we assume the cell to have gone out of view and will thus no longer update the tracker associated with this cell.
After going through the list of assigned pairs and removing the invalid assignments, the program checks if there are any cells found in the next frame that haven't been assigned to a cell in the current frame. If this is the case, the program assumes this is a newly found cell yet to be tracked. A new tracker with a unique id is created to start tracking the newly found cell.
We tried to increase tracking accuracy by using optical flow to predict the likely location of the cells in the next frame. Using optical flow the program generated a predicted area for each cell in the next frame, and used the IoU of this predicted area and the cells found in the next frame to fill the matrix of distances and assign each cell. Thus tracking cells based on the predicted locations of each cell rather than their actual location.
After iterating through each frame using this approach, the found trackers were stored in a list and saved in a .pkl file. This file is intended to be read by the second program.

3.2 Action detection

Our action detection program takes the generated .pkl file and the associated sequence of frames as input. For each tracker, the location history of the tracked cell is used to to create a path that this cell has walked. Additionally, the maximum width and height (w_m, h_m) of the bounding boxes of this cell is used to create a w_m by h_m pixel image sequence which follows the cell throughout its existence. Since the model accepts video data with resolution of at least 122×122 pixels, we increased the width and height of these image sequences to 122 if necessary.
Since we not only want to recognise what actions each cell performs but also when and where these actions have been performed, the program takes a sliding window approach on this image sequence. The window size is 16 frames, which is the minimum number of frames accepted by the

C3D network, and a stride of 8 frames. Action recognition is performed over each window using the C3D model from the mmaction2 Python framework [14].
We once again experiment with a certainty threshold, where an action is only considered to have occurred if the C3D model outputs the action with a certainty exceeding a specified threshold. Thus a list of actions is obtained representing the actions a cell has performed. Since each frame is part of two windows, the program checks if each detected action is found in at least two consecutive windows. When an action is found to have taken place over multiple consecutive windows, a time interval is obtained where this action has been observed. The center frame of this interval is taken to represent the moment this action took place. Thus the program produces a list of trackers, and an associated list of actions and frame numbers. Using these we are able to determine where each action has taken place and which cell performed said action.

3.3 Training the neural networks

Since we want to use the neural networks for a very specific purpose, we need to train our networks appropriately to be able to recognise cells and the actions they perform. To do this, we need to construct an appropriate training set to train and test our network on. In the following section we will describe the construction of these data sets and how we trained on them.

3.3.1 Faster R-CNN

The Faster R-CNN model used in this project is derived from the pre-trained `faster-rcnn_r50_fpn_1x_coco.py` model from the mmdetection framework[16]. We changed the configuration to make the model fit to train on microscopy time lapses of cells by decreasing the number of classes to one, since we are only interested in looking at cells and no other objects, and trained it using the training script provided by the framework.
We trained our network on data from the Cell Tracking Challenge (CTC) website [13] which contains a collection of annotated cell data, specifically created for the purpose of training, testing, and bench-marking cell tracking programs. This dataset includes various cell types in various conditions and imaging modalities. Here, we took the training set and used the available segmentation files to generate our own annotation files for training and testing purposes. We do this by creating boxes around each segment in the segmentation file and storing their coordinates in appropriate format in our own annotation files. From the data available on the website we used the 2D+Time data sets, since we want our program to recognise cells in two dimensional video images. We did not use the `N3DH` footage, since in this data set only the cell nucleus is visible where we are interested in actions of the entire cell body.
We train this network with $scales = [8, 16, 32, 64]$ and an aspect ratio of $[1 : 1]$. The training was carried out on the Leiden University High Performing Computing Facility ALICE.

3.3.2 C3D

The C3D network we used is derived from the pretrained C3D model found in the mmaction2 framework [14]. For training the C3D network we used the data from the CTC website, alongside published time lapses of endothelial cells. We went through this data and created clips of minimum size 122×122 pixels and length of at least 16 frames where a specific action could be observed and

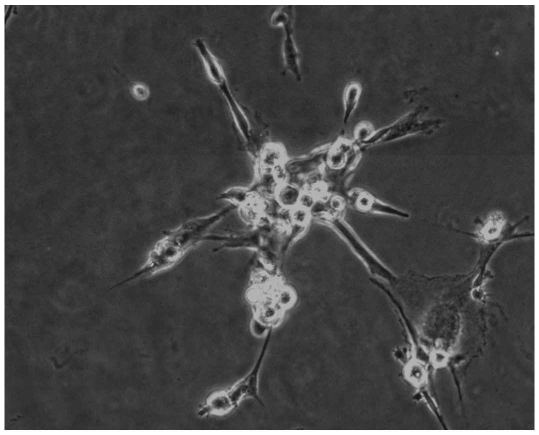


Figure 8: An image of a cluster of cells

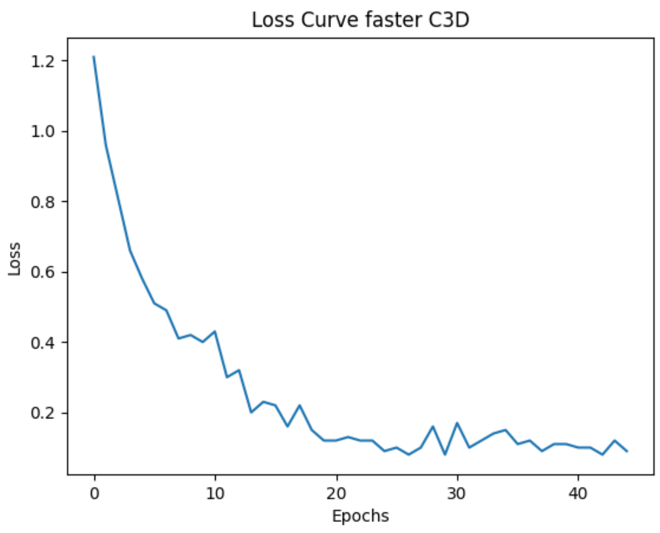


Figure 9: Loss curve for training of the C3D network

created annotation files in a format accepted by the network. In this way we build a training and testing data sets.
We trained the data to be able to recognise two different actions: clustering and cell division. Clustering is defined as the action where multiple cells group together in a distinct cluster, as depicted in Figure 8. Cell division refers to the action of a single cell dividing into two.
The loss curve associated with the training of the C3D network can be found in Figure 9,

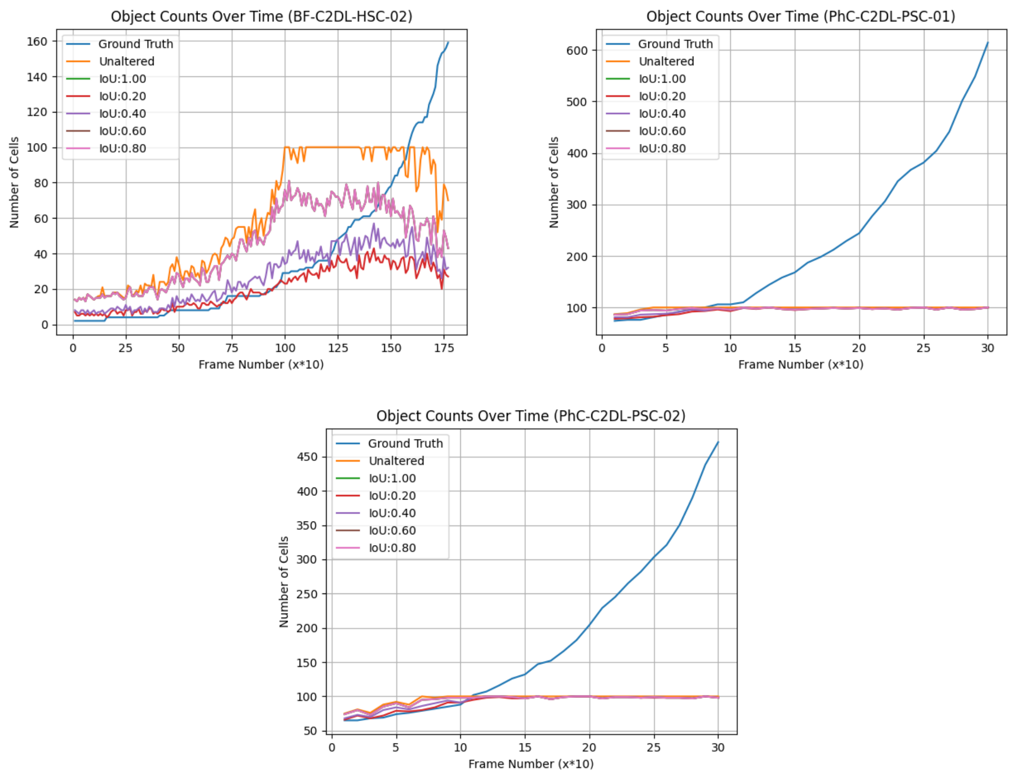


Figure 10: Number of found objects on the BF-C2DL-HSC-02, and both the PhC-C2DL-PSC data sets

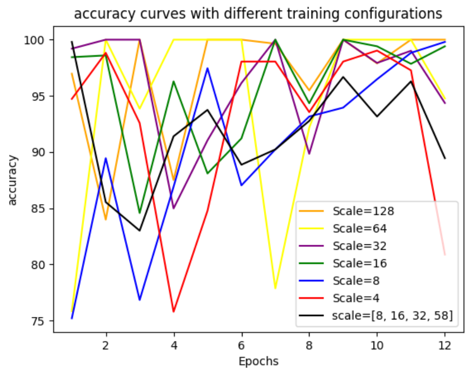
4 Experimental Results

When experimenting, we used the already annotated data from the CTC website [13]. This makes it possible to evaluate the performance of our program without needing to build a testing dataset from the ground up.

4.1 Object Detection

The network cannot find more than 100 objects per frame because of the configuration used when trained. This can easily be changed by changing its configuration, but for time preserving purposes, we keep this limitation. Therefore we will not look at the PhC-C2DL-PSC footage and the BF-C2DL-HSC-02 footage since, due to cell division, they eventually have more cells than the network can detect in a single frame resulting in an output as displayed in Figure 10. This does not tell us anything about the performance of the program itself and instead tells us there is a limit of number of cells visible in one image.

Anchor box sizes We trained the faster R-CNN network with different anchor box scales: $4, 8, 16, 32, 64, 128$, and one configuration using multiple scales: $[8, 16, 32, 64]$, the loss and accuracy values of this training process can be seen in Figure 11. In all further experiments, we used the network configured to use the four anchor box scales to perform the cell detection task.



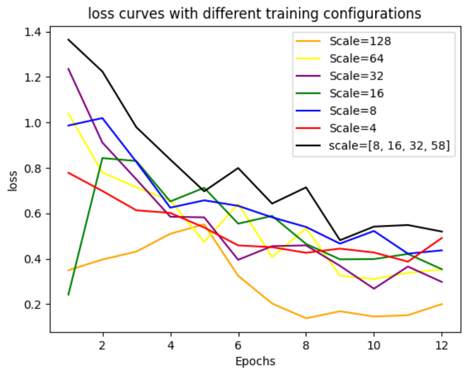


Figure 11: Accuracy and loss of rcnn network with different anchor box scales.

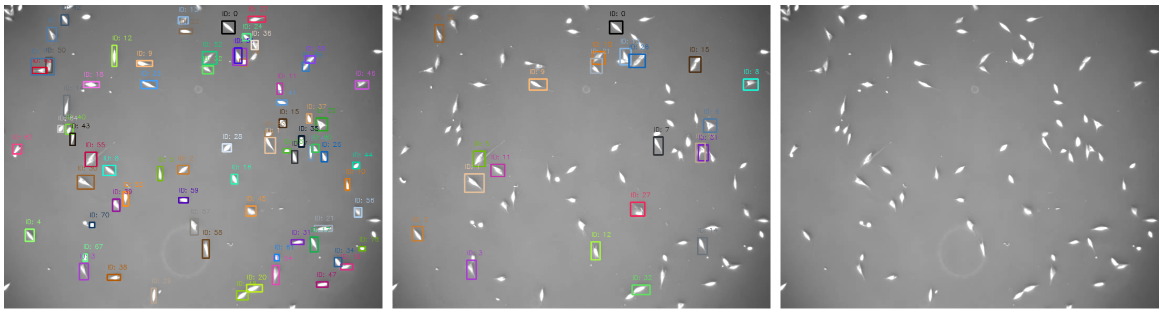


Figure 12: Result of object detection performed on frames of the PhC-C2DL-PSC dataset from the CTC[13] website with, from left to right, anchor box scales 4, 16 and 64

We also took some individual frames of data and performed object detection to gain a better understanding of how the object detection behaves on different footage (Figure 12, 13).

Data cleansing We tried to improve the object detection model by filtering out redundant bounding boxes through various methods. We evaluated the different methods of cleansing our data by assessing the number of cells the program found in specific frames of the input. It is important to note that when the number of cells detected by the program is equal to the number of cells visible in a frame, it does not necessarily mean that the program has found every cell in this frame. It is possible that n bounding boxes containing a cell have been discarded, and n redundant bounding

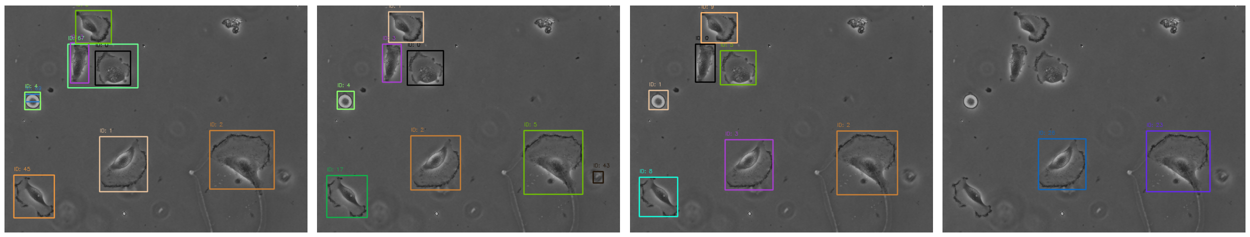
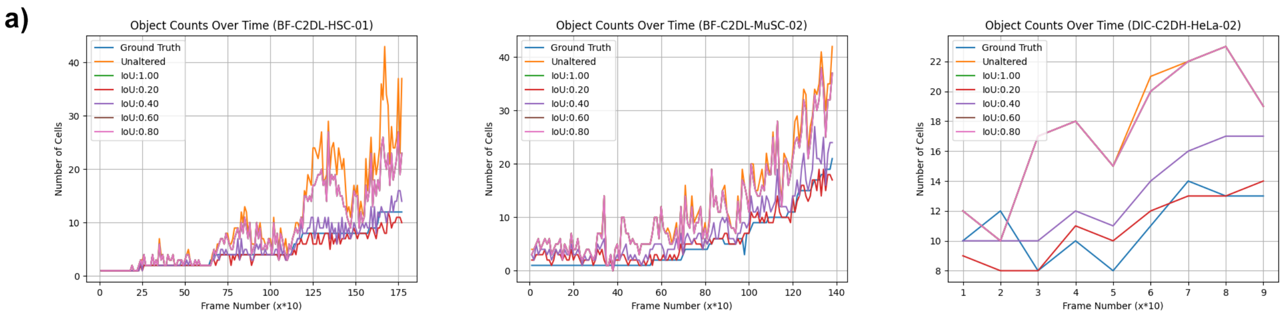


Figure 13: Result of object detection performed on frames of the PhC-C2DH-U373 datset from the CTC[13] website with, from left to right, anchor box scales 4, 16, 32 and 64

a)



b)

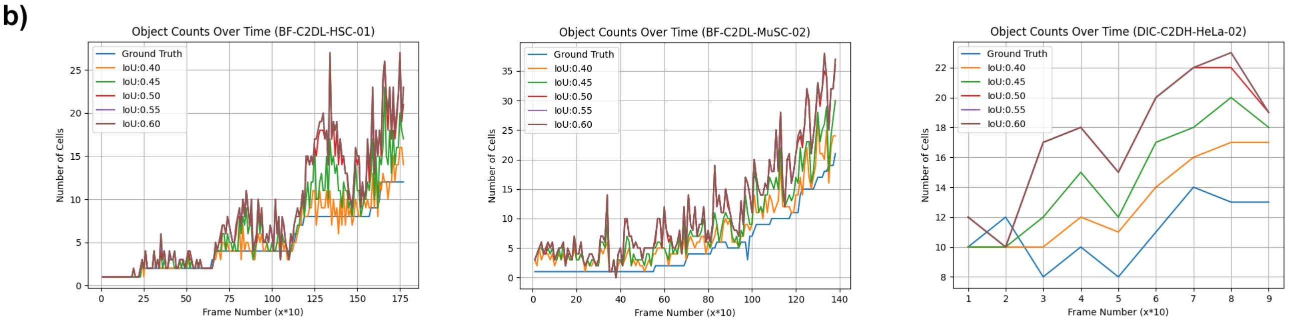


Figure 14: Number of cells found in every tenth frame in the, from left to right, BF-C2DL-HSC 01, BF-C2DL-MuSC 02 and DIC-C2DH-HeLa 02 data sets, where we discard found cells when their bounding boxes are contained within one another, or their IoU is greater than a certain threshold. Here IoU:1.00 means it only discards boxes when they are exactly the same or a box is contained in another box, Ground Truth is the actual amount of cells present in this frame, and Unaltered means we did not discard any found boxes

boxes are being kept. Still, we believe that using this method of evaluation gives us a general sense of the accuracy of the program.
One method of data cleansing involves assuming that two bounding boxes refer to the same cell when one bounding box is contained within another, or when the IoU of two bounding boxes exceed a certain threshold. When the program determines two bounding boxes refer to one cell, the bounding box with the greater object score is kept and the other one is discarded. We ran the program with different threshold values while noting the amount of cells found every 10 frames, the results are presented in Figure 14a
Upon analyses, we observed that an IoU threshold of 1.00, 0.80, and 0.60 do not significantly impact the number of found cells and that for an IoU$\leq$ 0.40 the number of boxes found can often drop below the actual number of cells (ground truth). Thus we repeated the experiment with IoU thresholds between 0.40 and 0.60(Figure 14b)
Another method of data cleansing involves assuming that bounding boxes with an object score smaller than some threshold do not actually refer to a cell. If the program determines a bounding box does not contain a cell, the program discards it. We ran our program with different object score threshold values. During this experiment we did not data cleanse using the previous method in order to only evaluate the efficacy of the object score threshold. The results of this experiment can be found in Figure 15a.
Here we find that when using a threshold value over 0.30, the number of cells the program finds can drop below the actual number of visible cells. Thus we repeated the experiment with threshold

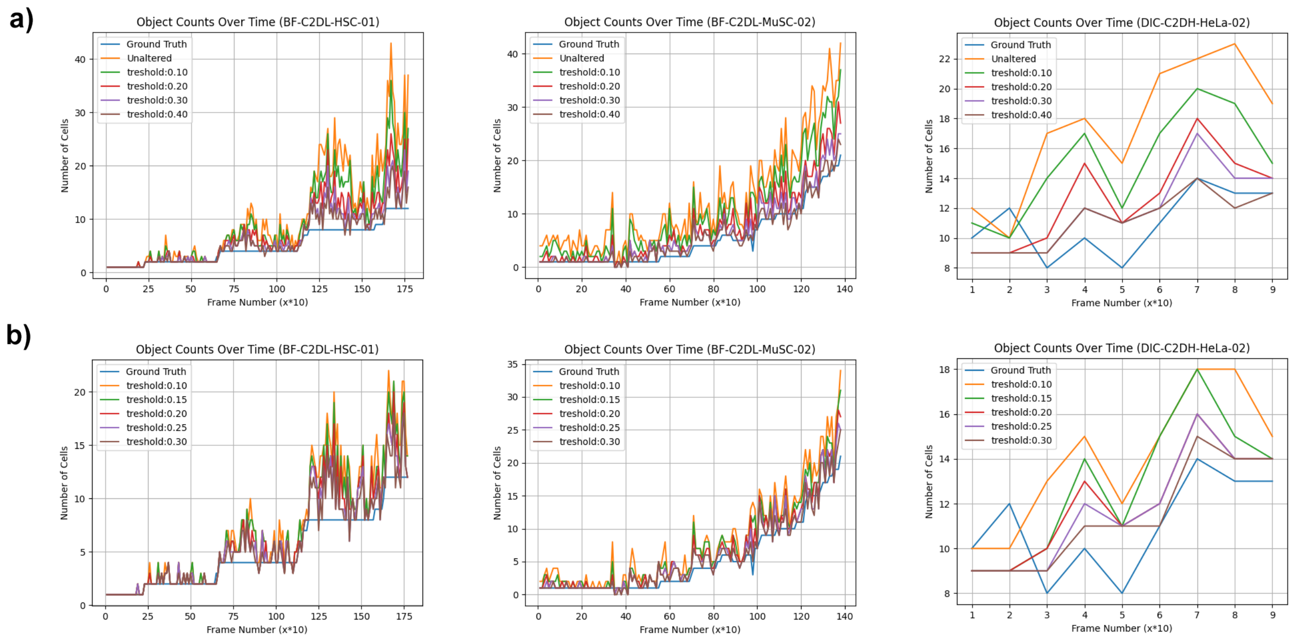


Figure 15: Number of cells found in every tenth frame in the, from left to right, BF-C2DL-HSC 01, BF-C2DL-MuSC 02 and DIC-C2DH-HeLa 02 data sets, where we discard any found bounding boxes where the confidence of the box containing a cell is less than the threshold value. In a) we do not discard any boxes based on how similar it is to another box, in b) we discard the least confident box when the IoU of two boxes are greater than 0.50, or when a box is completely contained in another box.

values between 0.10 and 0.30, this time we also used an IoU threshold of 0.50, and asserted that when a bounding box is contained within another, they refer to the same cell, the results can be found in Figure 15b. We find that, by both using a confidence threshold and an IoU threshold, the number of detected cells gets closer to the Ground Truth value than when we only use one of the two methods.

4.2 Object Tracking

Using the models we obtained we ran our cell tracking program, we experimented with predicting cell locations to aid in cell tracking using optical flow. These can be found in Appendix A Where the paths of the cells the program created without using optical flow and with using optical can be compared to the actual paths taken by the cells.

4.3 Action Detection

To test and experiment with the action detection program, we used the ground truth tracking data already available in the CTC website as input to test the action detection program. Since these sets of data only contain footage of cell division, we removed clustering from the possible outcomes to test this part of the program. To take into account the sliding window size of 16 and stride of 8, we put 8 buffer frames both in front of and behind the frames in which the cell is visible allowing

the action detection program to view the entire lifespan of the cell, and we say each action has a temporal margin of error of 8 frames.
Since the action recognition network gives for each possible action a confidence value signifying how certain the network is that the corresponding action has taken place in the input, we can experiment with using different confidence thresholds (CT). If the confidence of an action falls below this threshold we consider this action to not have taken place, even if said action has the highest probability of having occurred. To quantify the accuracy of this part of the program, we once again used the IoU [11] to measure the similarity of the actions the program found, with the actions that are actually visible. The results of this experiment can be found in Table 1

Action Detection Results						
Input data (GT)	CT	FP	TP	FS	DA	IoU
BF-C2DL-HSC 01 (11)	50%	7	2	2	4	0.100
	60%	4	1	1	2	0.063
	70%	3	1	1	1	0.067
BF-C2DL-HSC 02 (157)	50%	0	3	3	1	0.019
	60%	0	1	0	0	0.006
	70%	0	0	0	0	0
BF-C2DL-MuSC 01 (22)	50%	4	11	0	1	0.423
	60%	3	9	0	2	0.360
	70%	3	8	0	2	0.320
BF-C2DL-MuSC 02 (22)	50%	1	0	0	0	0
	60%	0	0	0	0	0
	70%	0	0	0	0	0
DIC-C2DH-HeLa 01 (8)	50%	3	1	0	2	0.091
	60%	3	0	0	0	0
	70%	1	0	0	0	0
DIC-C2DH-HeLa 02 (5)	50%	2	1	0	1	0.143
	60%	0	0	0	0	0
	70%	0	0	0	0	0
Fluo-C2DL-Huh7 01 (1)	50%	0	0	0	0	0
	60%	0	0	0	0	0
	70%	0	0	0	0	0
Fluo-C2DL-Huh7 02 (6)	50%	9	1	0	0	0.067
	60%	9	0	0	0	0
	70%	9	0	0	0	0
Fluo-C2DL-MSC 01 (0)	50%	9	0	0	0	0
	60%	9	0	0	0	0
	70%	9	0	0	0	0
Fluo-C2DL-MSC 02 (0)	50%	4	0	0	0	0
	60%	4	0	0	0	0
	70%	3	0	0	0	0
PhC-C2DH-U373 01 (0)	50%	0	0	0	0	1
	60%	0	0	0	0	1

Action Detection Results (cont.)						
Input data (GT)	CT	FP	TP	FS	DA	IoU
	70%	0	0	0	0	1
PhC-C2DH-U373 02 (0)	50%	0	0	0	0	1
	60%	0	0	0	0	1
	70%	0	0	0	0	1
PhC-C2DL-PSC 01 (570)	50%	0	1	0	0	0.002
	60%	0	0	0	0	0
	70%	0	0	0	0	0
PhC-C2DL-PSC 02 (429)	50%	0	2	0	0	0.005
	60%	0	0	0	0	0
	70%	0	0	0	0	0

Table 1: Output of Action Detection experiment where we look at the output accuracy using different CTs, Here we list every dataset used as input together with the number of cell divisions that are visible in the data set (GT), we list the number of false positives detected (FP), the number of true positives detected (TP), the number of actions detected but assigned to a cell id than the cell performing the action (FS), and the number of actions detected multiple times (DA). For example, if cell A divides into B and C and we observe cell division at that time and location in cells A, B, C and D. We will have one count in TP, two counts in DA and one count in FS. Lastly, the IoU between TP and FP is given. Here we disregard DA and consider the observations of FS to be a false positive.

We can see that in general, a lower action threshold results in a better IoU output. Furthermore, for the sake of experimentation, we ran the entire program on these datasets without omitting the action of clustering. This output can be found in Appendix B

5 Discussion

5.1 Object Detection

We can see in Figure 14 and Figure15 that it is possible to use various methods of data cleansing to have the number of cells the program detects to come closer to the true number of cells in each frame. However, the configurations which results in the number of found cells being closest to the truth also occasionally find less cells than there actually are, which indicates that the program now does not consider every cell to be a cell. At the same time, the configuration in which the number of cells never go below the truth value also have a lot of frames where the program finds a lot more cells than there actually are. Indicating that with using the data cleansing methods we used,

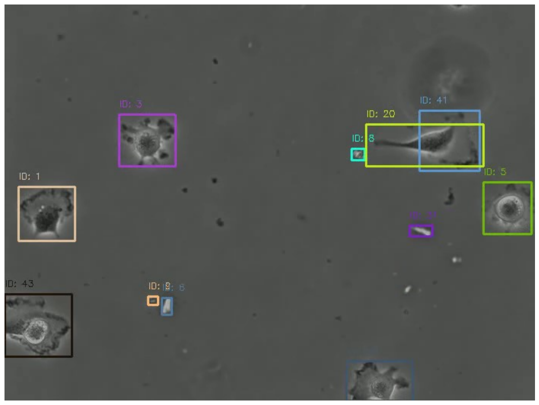


Figure 16: Case where two boxes find the same cell, but they are too different to have a large enough IoU to be considered boxes around the same cell, and one is not completely contained within another

there is a limit to how accurate the object detection model will be. This indicates that there are multiple cases when bounding boxes are dissimilar, but are viewing the same cell and both have high object score (an example of this can be found in 16), or that a bounding box not containing a cells has high object score. If the latter is often the case, it would mean that the network is not properly trained, but viewing all other results we do not believe this to be the case. The former often happens in data where cells can be of very inconsistent size, as is the case in figure 16. To further increase accuracy we would need to use different methods of data cleansing alongside the methods we used here.

5.2 Object Tracking

In the case of object tracking, it often happens that a single cell changes id because the program finds two boxes where there is only one cell, and the old tracker gets discarded and the newly found tracker continues tracking the cell. This can be seen in figure 17. This problem would likely be largely solved if the object detection network was more accurate, since the best candidate for the next cell is correct, but since the program finds two bounding boxes for one cell, tracking becomes difficult. When viewing the first column in Appendix A, we can see that the tracking algorithm is able to find the track of the cells fairly accurately, though sometimes a tracker seems to select the wrong cell and starts tracking a different cell than it started with. We can see that this most often happens when multiple cells are right next to each other. Therefore, in datasets like PhC-C2DH-U373, where the cells are widely distributed, this seems to be less of a problem. We suspect the incorrect tracking happens when a bounding box is created containing multiple cells giving the tracker an opportunity to use the box containing the multiple cells as a bridge to start tracking a different cell. Though we have not confirmed this to be the case.
We see that using optical flow to assist in tracking results in tracks jumping from location to location (Appendix A). This jumping of the cell tracks happens when the tracker switches from one cell to another. This is very prevalent when using optical flow, we think this is because the optical flow algorithm has difficulty understanding where a cell is going. Optical flow is developed with "normal" frame rate in mind, thus anything above 1 fps, but the frame rate of the data we work

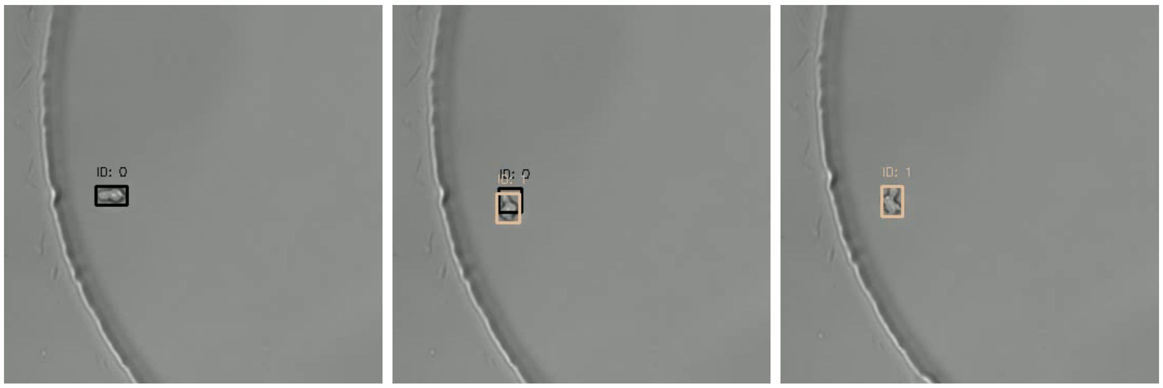


Figure 17: A cell from the BF-C2DL-HSC-01 footage[13] changing id, on the left: the cell with its original id of 0, in the middle: two boxes being found around the same cell, but different enough that one of them does not get discarded. One of them considered a continuation of the cell with id 0, the other seen as a new cell and given the id of 1, on the right: The network finds only one cell but the tracking algorithm considers the found box to be the continuation of tracker 1, discarding the original tracker with id 0

with is usually very low, having one frame per several minutes. This can result in cells drastically changing shape or direction from one frame to another which makes predicting location difficult. This results in the optical flow algorithm to think that the cell is moving differently than it is, both in direction and in speed, thus resulting in the predicted location box to be in a very different place than it should be, which causes it to be linked to a bounding box of a different cell.

5.3 Action Detection

Looking at the results in table 1 using the CTC data [11], we find that the action detection program is far from accurate. But we can see that a lower confidence threshold has consistently a higher IoU. It is noticeable that the program does not recognise the cell divisions at all in the BF-C2DL-HSC 02 and both the PhC-C2DL-PSC data sets, which contain data of many small cells very close to each other, and a lot of cell divisions. This is probably mainly due to the fact that the cells are smaller than the minimal clip dimension size, and that there are a lot of cells close to each other. This results in each clip following a single cell looking for actions said cell performs, to be also looking at a lot of other cells. This makes it hard for the program to determine what is happening at any moment, making the confidence that a division has taken place very small. This is somewhat supported by the BF-C2DL-HSC 01 output which has found more cell divisions that have taken place.
We can also see that in both the BF-C2DL-HSC data sets, it happens that cell division of a certain cell, has been linked to a different cell than the cell actually performing the action. This is probably once again due to the fact that the cells in this data set are smaller than the minimum window size required for the action recognition network, resulting in multiple cells being visible in a clip following the track of a single cell. Thus when we are looking at the action of a specific cell, say cell A, but a cell nearby, cell B, performs an action visible by the action recogniser, the program will mistakenly say the action is being performed by cell A because that is the cell we are currently observing.

When looking at the output in Appendix B, we can see that in a data input as PhC-C2DL-PSC 01, we often observe clustering. This is probably caused in a similar way as why the program has difficulty detecting cell division. When a lot of cells are visible in a clip, the program tends to observe clustering instead of cell division while clustering might not even be taking place.
In general, because the program is trained to only look at a single cell and determine the action of a single cell, it has difficulty with cells smaller than the minimal input window size. But even when we only feed cells bigger than a certain size in the action detection program, it will not be able to properly detect actions as clustering since this is an action involving multiple cells. Therefore, if we only look at a single cell it will be hard to accurately detect larger scale actions as clustering.

5.4 General Limitations

The fact that the program is split into two is a result of both the C3D and Faster R-CNN models both sharing the same mmcv [15] library. Thus when trying to initiate both models in the same program, there will be some conflict withing this mmcv library. Though having the program split in two was fairly convenient for testing purposes.
The biggest limitation in this project was the size of the training data and the hardware with which with which we could train the neural networks. Since we had to build a data set for the cell actions ourselves, we were not able to have a data set as expansive as we ideally would have wanted. For example in the CTC Dataset[13], the Fluo-C2DL-Huh7 and the Fluo-C2DL-MSC data sets were both far smaller than other data sets, both in number of frames and in the number of cells found in every frame. This limits the training we can do for both tracking and action detecting for these specific types of cells. Sadly, we also did not have access to hardware able to train the object detection model with anchor boxes of different aspect ratios, resulting in the program terminating due to a lack of memory before reaching a single epoch when trying to do so. Thus we could only train the program with anchor boxes of different sizes, but a single shape which may have limited our ability to recognise cells of unconventional shapes.

6 Conclusion and Future Research

There are several ways we would like to try to improve this program.
Firstly, with stronger GPUs, it would be interesting to see if training the object detection program with both multiple anchor sizes and shapes could result in the program being more capable of detecting cells of various shapes. It might also be interesting to look at using a masking program for object detection and tracking. This would tell exactly which pixels in the image belong to each cell instead of a general box surrounding a cell. This could be used to improve the tracking program, where we can see in more detail if the shape of the cells we wish to link are similar. It also might help increasing action recognition accuracy, as we could mask out everything that is not the cell when performing action recognition, potentially solving the problem of mistakenly recognising actions of other cells in the vicinity of the cell we are looking at.
To increase the tracking program, it may be a good idea to create a set of conditions of when a new cell can appear or disappear, ensuring that the condition for a new tracker being initialised is not only finding a candidate cell which is not linked to a cell in the previous frame, but also, for example, that cell division has taken place, or that the cell appeared on the edge of frame. The

former condition brings new problems since, given the construction of this program, we can only tell what actions have taken place after tracking all cells over the input data.
Secondly, it might be possible to develop a program that achieves the same goals as the one set in this thesis using a slightly different approach by using some sort of pattern recognition and object detection to decide on spatio-temporal candidate actions. Since object detection programs are able to detect different types of objects, we may be able to detect different behavioural patterns as different objects. For example, a dead cell can be recognised as one using the object detection program, thus the frames around when a living cell box is first linked to a dead cell box can be candidates for the action cell death which can be confirmed by the action recognition program. We could also take the appearance of new cells in the middle of the frame as a candidate for cell division. Besides this, object detection could be used to recognise actions which involve multiple cells. For example clustering: if we can see the entire image, a cluster of cells can more easily be recognised than when we only look at clips of a single cell. Therefore, to recognise actions involving multiple cells, it may be more accurate and beneficial to use the object detection part of the program to look at these actions from a larger scale.
The use of different action detection, and object detection networks instead of Faster R-CNN and C3D could also help in improving accuracy of the program.
In conclusion, it is probably possible to create a spatio-temporal action detection program which properly analyzes video data of microscopic cells. The program developed as part of this thesis was an attempt to achieve this, though we were unable to create a program accurate enough for researching microscopic cells. Despite this, there are a lot of ways that this specific approach to automatic action detection in cells could be improved on.

References

[1] S. S. Beauchemin and J. L. Barron. The computation of optical flow. *ACM Comput. Surv.*, 27(3):433–466, sep 1995.

[2] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1), jan 2014.

[3] Neda Emami, Zahra Sedaei, and Reza Ferdousi. Computerized cell tracking: Current methods, tools and challenges. *Visual Informatics*, 5(1):1–13, 2021.

[4] Victor Escorcia, Fabian Caba Heilbron, Juan Carlos Niebles, and Bernard Ghanem. Daps: Deep action proposals for action understanding. In *European Conference on Computer Vision*, 2016.

[5] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian Conference on Image Analysis*, 2003.

[6] Ross Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[7] Fabian Caba Heilbron, Juan Carlos Niebles, and Bernard Ghanem. Fast temporal activity proposals for efficient detection of human actions in untrimmed videos. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1914–1923, 2016.

[8] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 52, 1955.

[9] Laura Leal-Taixé. Multiple object tracking with context awareness. pages 17–19, 11 2014.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[11] Michael Levandowsky. Distance between sets. *Nature*, 234(5323):34–35, november 1971.

[12] Tianwei Lin, Xu Zhao, Haisheng Su, Chongjing Wang, and Ming Yang. Bsn: Boundary sensitive network for temporal action proposal generation, 2018.

[13] Maška Martin, Ulman Vladimir, and Delgado-Rodriguez Pablo et al. The cell tracking challenge: 10 years of objective benchmarking. *Nature Methods*, 2023. Article: https://doi.org/10.1038/s41592-023-01879-y, Website: http://celltrackingchallenge.net/2d-datasets/.

[14] Mmaction2(v3.0.0), 2023. OpenMMLab's Next Generation Video Understanding Toolbox and Benchmark, https://github.com/open-mmlab/mmaction2.

[15] Mmcv, 2023. MMCV is a foundational library for computer vision research, https://mmcv.readthedocs.io/en/latest/get_started/introduction.html.

[16] Mmdetection(v3.0.0), 2023. OpenMMLab Detection Toolbox and Benchmark, https://github.com/open-mmlab/mmdetection.

[17] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *In:Proceedings of the 27th International Conference on Machine Learning*, 2010.

[18] Francesco Padovani, Benedikt Mairhörmann, Pascal Falter-Braun, Jette Lengefeld, and Kurt M. Schmoller. Segmentation, tracking and cell cycle analysis of live-cell imaging data with cell-acdc. *BMC Biology*, 2022.

[19] Lyle Ramshaw and Robert Endre Tarjan. On minimum-cost assignments in unbalanced bipartite graphs. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science, (FOCS).*, 2012.

[20] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.

[21] Yamashita Rikiya and Nishio Mizuho et al. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9:611–629, 2018.

[22] Karina Ruzaeva, Jan-Christopher Cohrs, Keitaro Kasahara, Dietrich Kohlheyer, Katharina Nöh, and Benjamin Berkels. Cell tracking for live-cell microscopy using an activity-prioritized assignment strategy. In *2022 IEEE 5th International Conference on Image Processing Applications and Systems (IPAS)*. IEEE, December 2022.

[23] Zheng Shou, Dongang Wang, and Shih-Fu Chang. Temporal action localization in untrimmed videos via multi-stage cnns, 2016.

[24] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[25] Elahe Vahdani and Yingli Tian. Deep learning-based action detection in untrimmed videos: A survey, 2021.

[26] Syed Sahil Abbas Zaidi, Mohammad Samar Ansari, Asra Aslam, Nadia Kanwal, Mamoona Asghar, and Brian Lee. A survey of modern deep learning based object detection models, 2021.

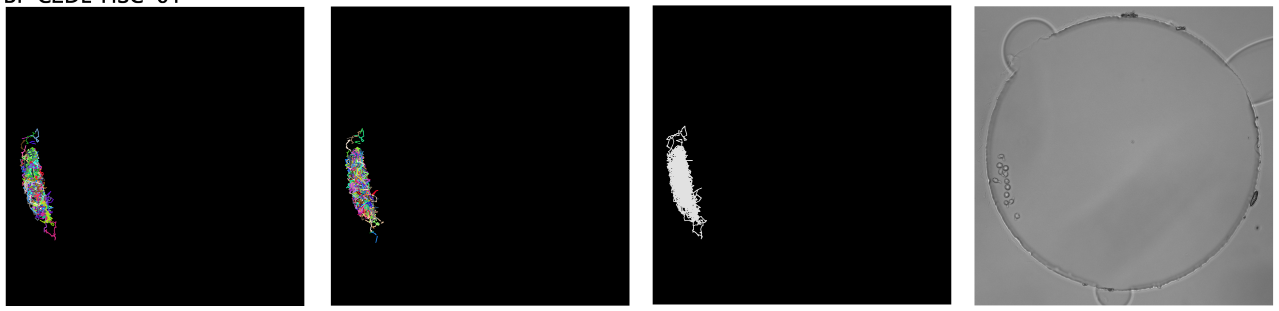
[27] Gangfu Zhang and Hubert Chanson. Application of local optical flow methods to high-velocity free-surface flows: Validation and application to stepped chutes. *Experimental Thermal and Fluid Science*, 90:186–199, 2018.

[28] Yi Zhu, Xinyu Li, Chunhui Liu, Mohammadreza Zolfaghari, Yuanjun Xiong, Chongruo Wu, Zhi Zhang, Joseph Tighe, R. Manmatha, and Mu Li. A comprehensive study of deep video action recognition, 2020.

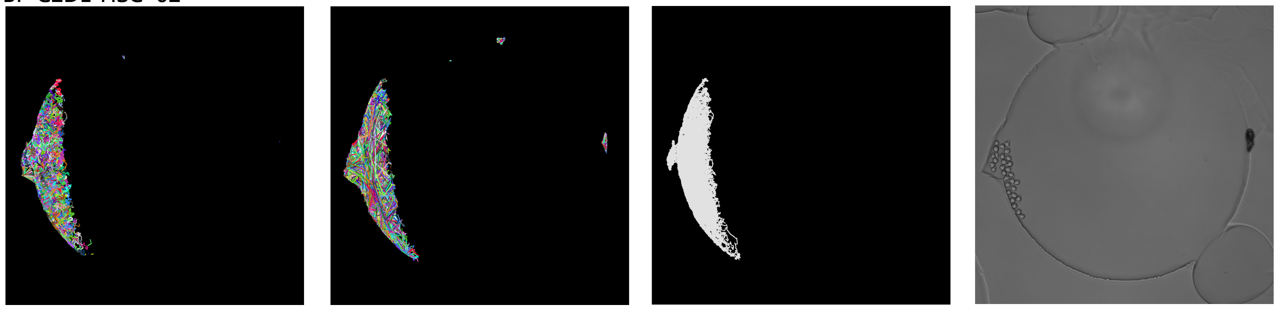
A Results Object Tracking with Multiple Anchor Box Scales

Below are the results shown of object tracking using an object detection model that has anchor box scales $8, 16, 32$ and 64. From the left to the right are displayed the tracking results of the model without using optical flow, the tracking results of the model using optical flow, the ground truth paths of the cells stored as truth values in the data set and a single frame of the input data.

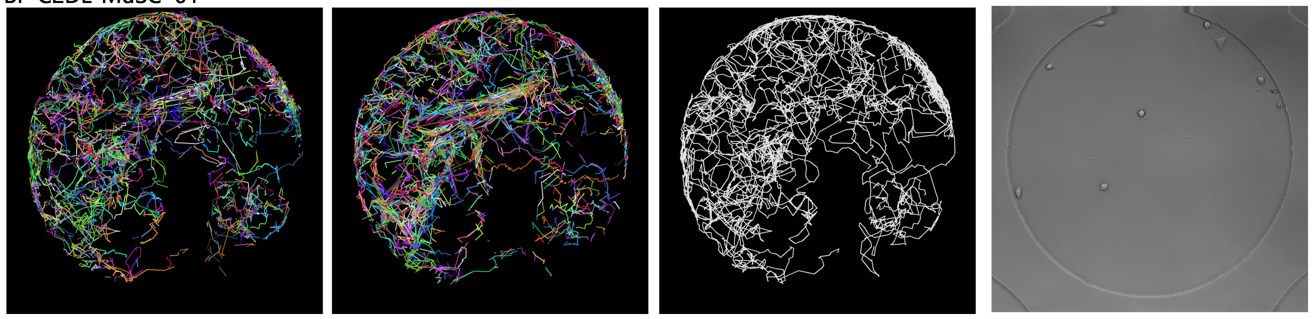
BF-C2DL-HSC_01



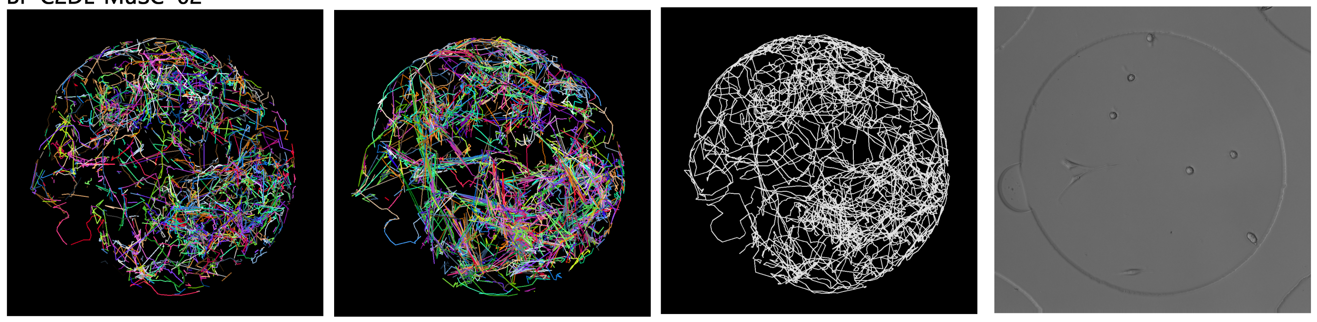
BF-C2DL-HSC_02



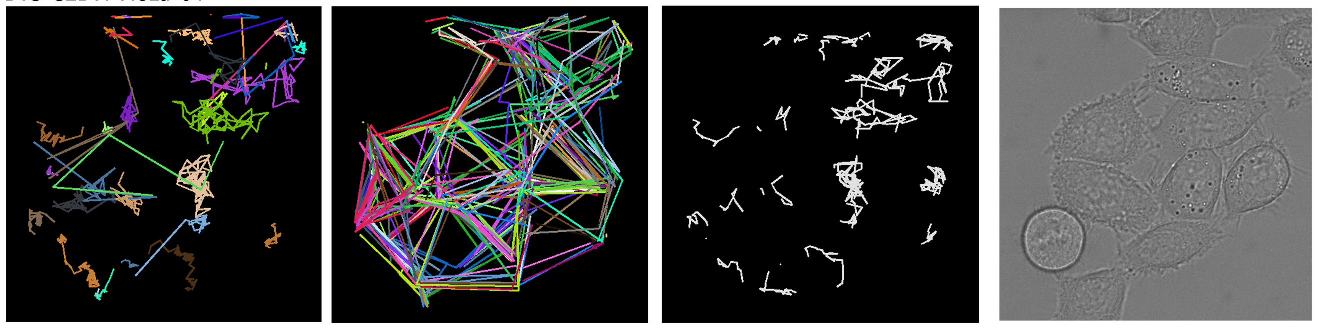
BF-C2DL-MuSC_01



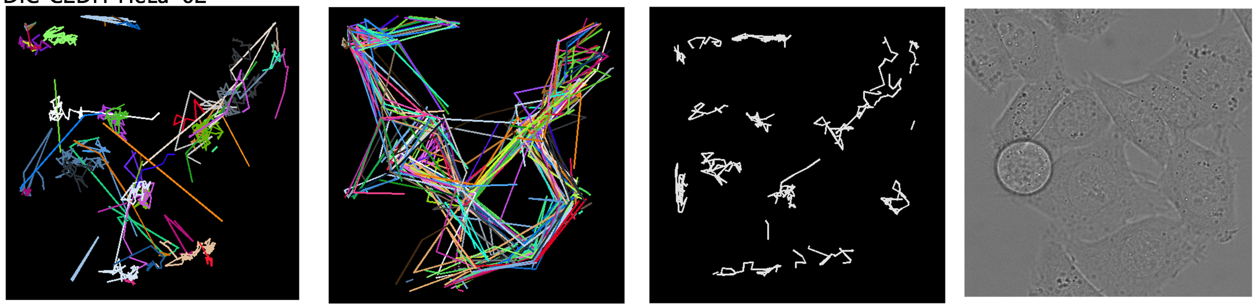
BF-C2DL-MuSC_02



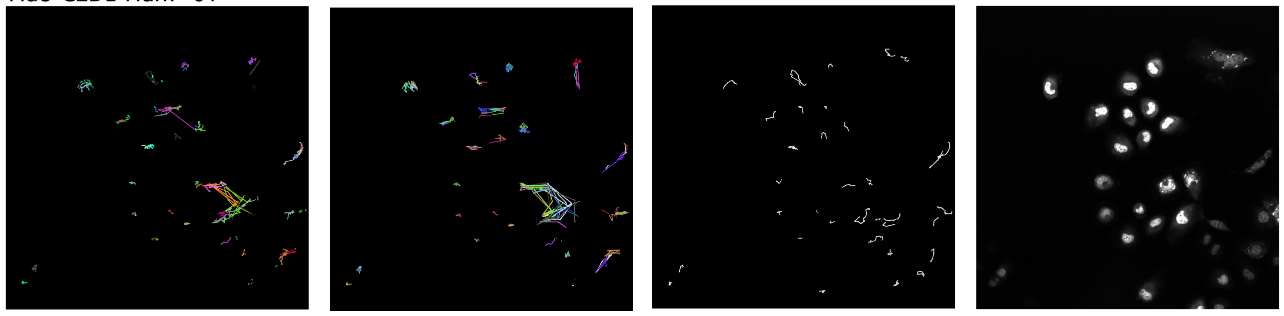
DIC-C2DH-HeLa-01



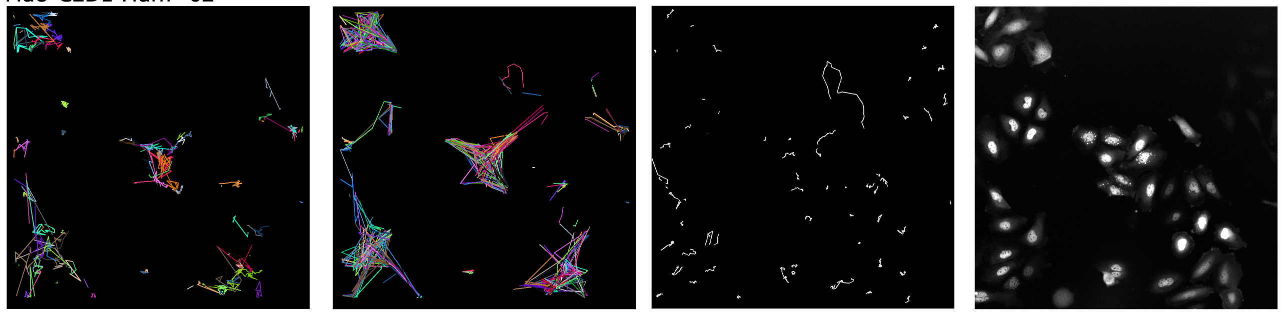
DIC-C2DH-HeLa 02



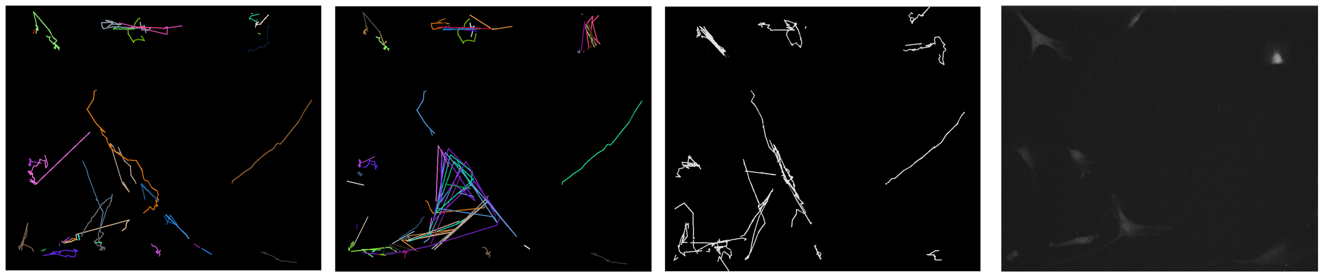
Fluo-C2DL-Huh7 01



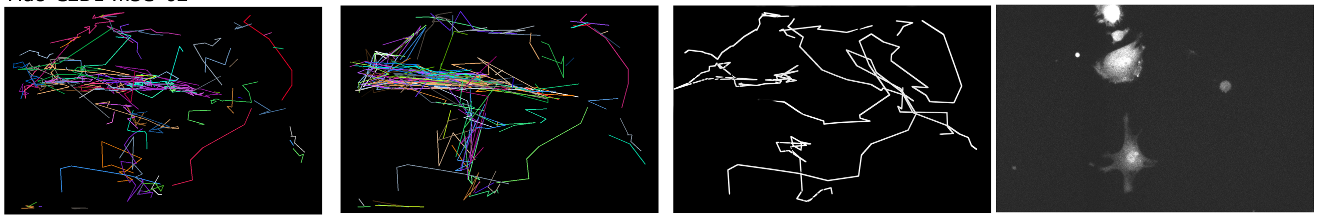
Fluo-C2DL-Huh7 02



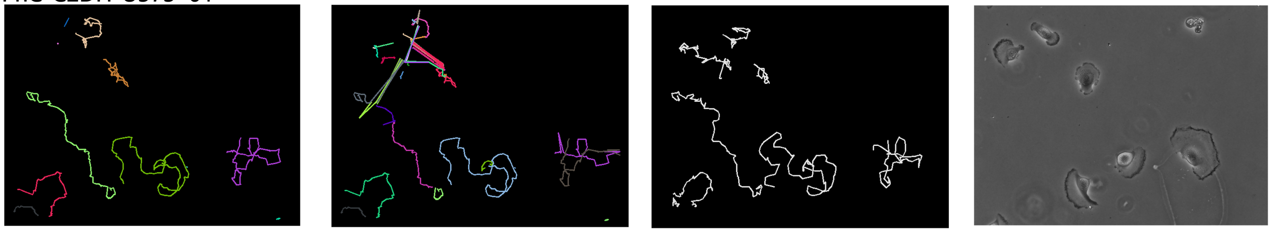
Fluo-C2DL-MSC 01



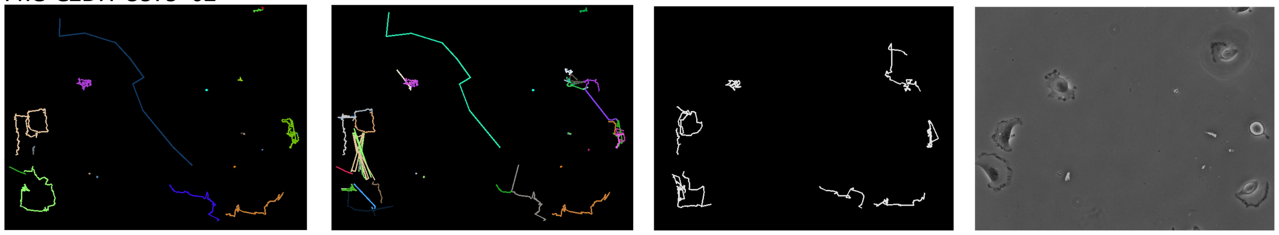
Fluo-C2DL-MSC 02



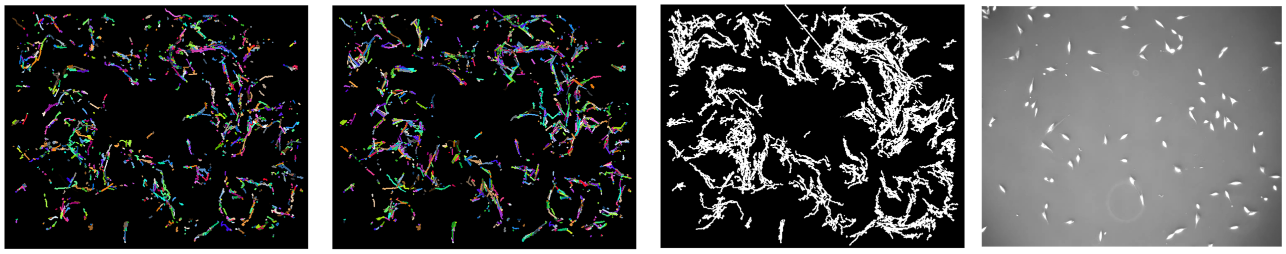
PhC-C2DH-U373 01



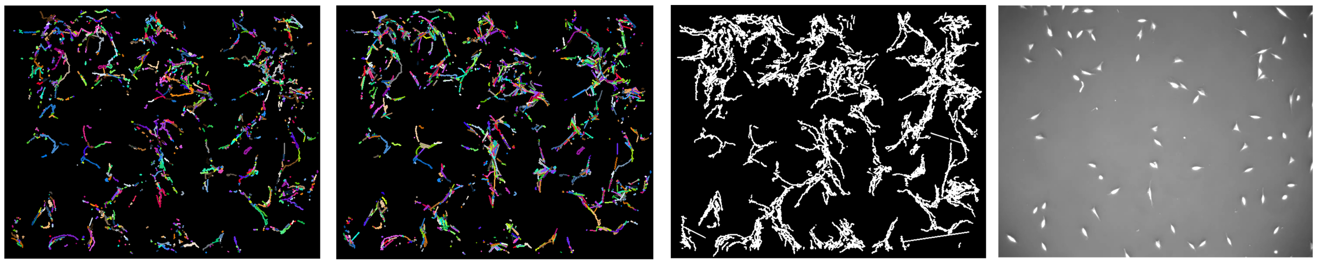
PhC-C2DH-U373 02



PhC-C2DL-PSC 01



PhC-C2DL-PSC 02



B Results Complete Program

Output when certain data sets are put through the entire program

PhC-C2DL-PSC-01.txt

```
cellid: 201
        action: clustering
                frame: 20
                coordinates: 162, 347
        action: clustering
                frame: 24
                coordinates: 165, 350
cellid: 20
        action: clustering
                frame: 71
                coordinates: 203, 311
        action: clustering
                frame: 45
                coordinates: 204, 316
cellid: 269
        action: clustering
                frame: 19
                coordinates: 233, 350
cellid: 571
        action: clustering
                frame: 2
                coordinates: 171, 350
        action: clustering
                frame: 9
                coordinates: 172, 353
cellid: 1655
        action: clustering
                frame: 4
                coordinates: 166, 355
        action: clustering
                frame: 9
                coordinates: 165, 355
cellid: 3065
        action: clustering
                frame: 1
                coordinates: 174, 227
```

DIC-C2DH-HeLa-02.txt

```
cellid: 3
        action: cell division
                frame: 25
                coordinates: 181, 197
cellid: 2
```

```
        action: cell division
                frame: 25
                coordinates: 336, 213
cellid: 6
        action: cell division
                frame: 28
                coordinates: 141, 264
cellid: 1
        action: cell division
                frame: 53
                coordinates: 307, 442
cellid: 77
        action: clustering
                frame: 11
                coordinates: 34, 329
cellid: 85
        action: cell division
                frame: 1
                coordinates: 384, 134
cellid: 28
        action: cell division
                frame: 32
                coordinates: 202, 472
cellid: 0
        action: cell division
                frame: 49
                coordinates: 424, 304
cellid: 4
        action: clustering
                frame: 25
                coordinates: 92, 42
        action: cell division
                frame: 53
                coordinates: 95, 64
cellid: 7
        action: cell division
                frame: 24
                coordinates: 418, 157
        action: cell division
                frame: 55
                coordinates: 433, 123
```