Observation Based Modeling of Liquid Slosh in Drop Tests

Students:
Victor Ekblad
victor.e_96@hotmail.com
Axl Roos
axlroos@live.se

LTH Supervisor:
Magnus Oskarsson
magnus.oskarsson@math.lth.se

Tetra Pak® Supervisors:
Eskil Andreasson
Eskil.Andreasson@tetrapak.com
Aurelia Vallier
Aurelia.Vallier@tetrapak.com

Examiner:
Carl Olsson
carl.olsson@math.lth.se

Abstract

Tetra Pak® is a world leading food packaging company with a large variety of paperboard beverage packages. In the development of paperboard packages, an essential component is utilizing simulation models long before physical prototypes are manufactured. This speeds up the development process and allows quick evaluation of new designs. For this to be an effective approach, confidence in the virtual models is a key aspect. In the context of paperboard beverage packages, liquid product sloshing in drop tests is currently not adequately modeled in existing simulations. In this thesis, a framework for gathering data related to liquid product motion in drop tests, in a controlled and repeatable manner, is built, using image analysis and computer vision algorithms. Both the quantitative and observational results are successfully used to improve and verify new liquid sloshing simulation models at Tetra Pak®. In parallel, the collected data is utilized to create approximate data driven models using neural networks. Three different model architectures are implemented and evaluated, U-Net, convolutional LSTM, and a graph convolutional LSTM. The resulting U-Net data driven model achieved the best performance and is shown to sufficiently approximate liquid behavior in drop tests in the setting of dropped transparent bottles. The U-Net model is found to be an adequate complement to the physics based simulations, offering faster run times but with reduced accuracy.

Keywords: *Sloshing, Fluid Dynamics, Image Segmentation, 3D reconstruction, Machine Learning, U-Net, CNN, GCN, LSTM*

Acknowledgements

We would like to take the opportunity to thank our supervisors Magnus Oskarsson from LTH and Eskil Andreasson and Aurelia Vallier from Tetra Pak® for their valuable inputs and for helping us with this project. We would also like to thank everyone at Tetra Pak® who helped us with the practical parts in this work regarding all the equipment and help we needed.

Finally we would also like to thank our friends and families for listening to us go on and on about dropping bottles.

Contents

1 Introduction

Tetra Pak® is one of the largest food packaging companies in the world with a big variety of paperboard beverage packages. To speed up the development process of new designs, virtual simulation is utilized. To further improve the current models and to account more accurately for the fluid dynamics in liquid product drop tests, this work is initiated. In this thesis work, the liquid behavior inside a transparent bottle will be observed to get a better understanding of liquid behavior during drop tests. In addition, the use of data driven models using machine learning to approximate liquid product in drop tests will be investigated.

1.1 Context and purpose

During its lifetime, a paperboard beverage package is exerted to different loading conditions during manufacturing, transport, handling and usage. Therefore, an increased knowledge is needed how the package and filled liquid product interact during respective loading scenario. At Tetra Pak®, both physical and virtual tests are being conducted on packages to get information about the robustness. However, during most physical tests, such as package drop tests, the packages are filled with water. In reality this means that packages filled with any other product such as milk, yogurt or beans, are rarely being tested for products other than water. Other products might behave differently which could cause a different load and thus yield different and unforeseen results regarding package integrity.

Virtual twins, i.e. a computer replica of a physical package, is today commonly used during the package development at Tetra Pak®. However, the virtual drop test simulation models are not yet realistic or accurate enough. In a specific simulation setting of a rigid bottle the virtual behavior of liquid product can be investigated, which can later be integrated in models of deforming paper packages. Figure 1 illustrate a physical drop test in this setting, one current model at the start of this thesis, and an older model from 2013.

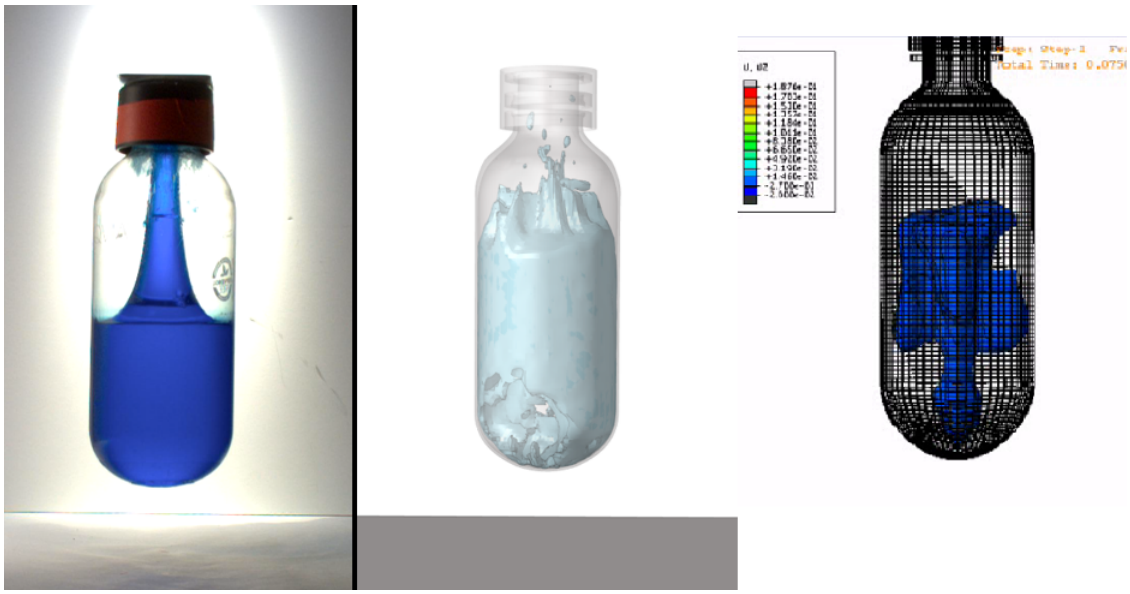


Figure 1: Illustration of a physical drop test (left), a current virtual twin (middle), and a virtual twin from 2013 (right).

To improve the virtual simulations, an understanding of how different liquids behave during a package drop test is desirable. In this thesis a transparent bottle with different types of liquids will be dropped, the sequence will be captured with two cameras, and the surface will be reconstructed.

Because of recent advances in applying neural networks to similar fluid dynamics applications, the obtained data will be analyzed and fed into machine learning models in a data driven approach, as a complement to the computationally costly physics based virtual twins.

1.2 Goal of the thesis

The aim of this thesis work is to contribute to the understanding of the liquid behavior during a package drop test. Collecting data from drop tests and analyze this could hopefully lead to a more in-depth understanding. Another aim is to investigate if machine learning methods can be used to accurately predict the behavior of the liquid in package drop tests. The results from this work can then hopefully be used to verify and improve the virtual package simulation models of drop tests.

2 Background

2.1 Properties of liquids

The understanding of how liquids behave and how they can be modeled is an essential part in many engineering applications. Sloshing for instance, is a term for liquid interacting within a partially filled container, plays an important part in the construction of spaceship fuel tanks, road tankers and water towers for example. To predict and solve these problems analytically is hard and restricted only to certain geometrical shapes of the container [8].

2.1.1 Characteristics of fluids

There are many different inherent properties of fluids that give rise to diverse physical phenomena. Concerning drops of liquids, and general flow of liquids, viscosity plays an important role. Viscosity is a measure of how resistant to flow a liquid is, meaning a low viscous liquid will typically flow faster than a highly viscous liquid. A liquid can be categorized as Newtonian or non-Newtonian depending on the behavior of its viscosity, where Newtonian liquids display constant viscosity independent on shear stress applied, and conversely non-Newtonian liquids viscosities' exhibit varying behavior relative shear stress [23].

How a specific liquid behaves can be found through rheology experiments, often approximated by models such as a power law model, Carreau- or, Cross-model, depending on its behavior. The power law models viscosity as a function of shear rate

$$\eta = m\dot{\gamma}^{n-1},$$

with constants n, m [10]. Figure 2 illustrates viscosities of liquid product relevant to this thesis modeled as power laws through rheology experiment done at Tetra Pak®.

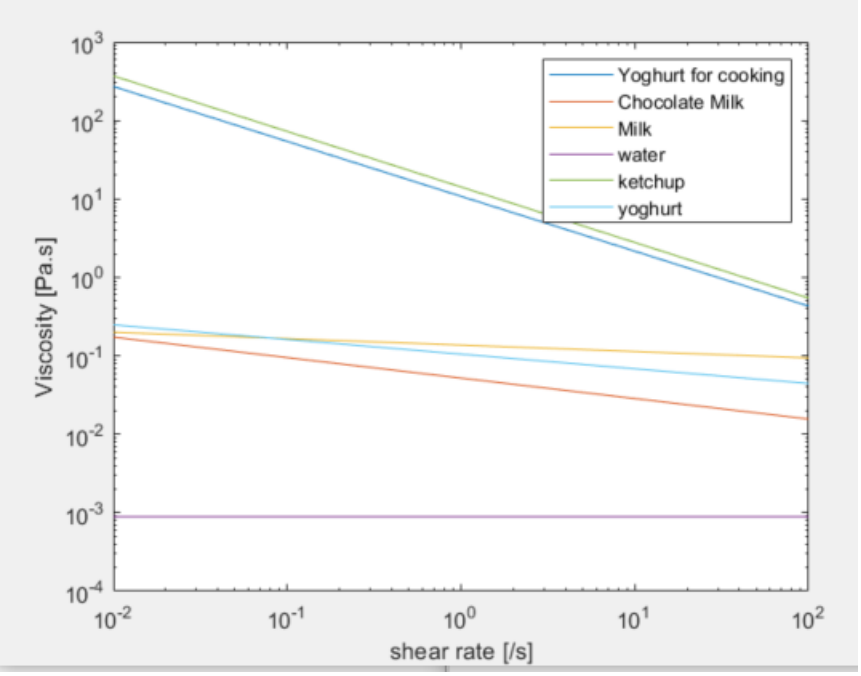


Figure 2: Viscosities of liquid product

2.1.2 Fluid dynamics

The branch of science where interaction and understanding of fluid behavior is covered is called fluid dynamics. Navier-Stokes equations are three-dimensional partial differential equations used to determine the motion of the fluid. The conservation of mass yields

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0, \tag{1}$$

in which ρ is the density and $\mathbf{v} = (v_x, v_y, v_z)$ is the velocity vector field of the flow in the fluid. Conservation of momentum for Newtonian fluids, like water, gives

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v}\right) = -\nabla p + (\lambda + \mu)\nabla(\nabla \cdot \mathbf{v}) + \lambda \nabla^2 \mathbf{v} + \rho \mathbf{f}, \tag{2}$$

where p is the pressure, λ a factor of volume compression, μ is the viscosity and $\mathbf{f}$ is the field of external forces where for example gravity is included. (1) and (2) are the Navier-Stokes equations. Further simplifications of the equations can be made if the fluid is incompressible, $\lambda = 0$ [6].

$$\nabla \cdot \mathbf{v} = 0 \tag{3}$$

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = -\frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\nabla^2 \mathbf{v} + \mathbf{f} \tag{4}$$

These equations can be solved numerically with Euler or Lagrangian approaches [22].

2.1.3 Computational fluid dynamics

Computational fluid dynamics (CFD) is a name for the numerical methods or softwares which, solves the Navier-Stokes equations numerically or approximates the physics of the fluid, to obtain a solution. Although these numerical methods can be used to solve these problems with good results they are very computationally costly. Since these methods often require small steps in time to obtain stable results and taking the complexity of the problem into account, the computational time varies in the range between hours to several days on high performing computers [11]. The computational effort linked to CFD solvers have raised the question if there are other methods that can be used to solve these problems in a shorter period of time and with good accuracy. When analyzing how machine learning techniques could be used in solving fluid dynamics problems and comparing them to the result from the CFD simulations, promising results were shown. Artificial neural networks and deep learning methods were found to obtain results faster but at the cost of reduced accuracy in the results [2].

2.2 Image Analysis and Data Processing

2.2.1 Pinhole camera model

A camera is used to project 3D objects from a real life scene to a 2D image plane. There are several ways to describe this mathematically but a simple and accurate model to use is the pinhole camera model. In the pinhole camera model the light rays passes through a single point, the camera center, before projection onto the image plane. In figure 3 the idea of the pinhole camera is shown. In a real pinhole camera the image plane is located behind the camera center, which has the effect

that the image will be upside down. However for simplicity the image plane can be assumed to be in front of the camera to get the scene to appear as it should [19,pp.103-126].

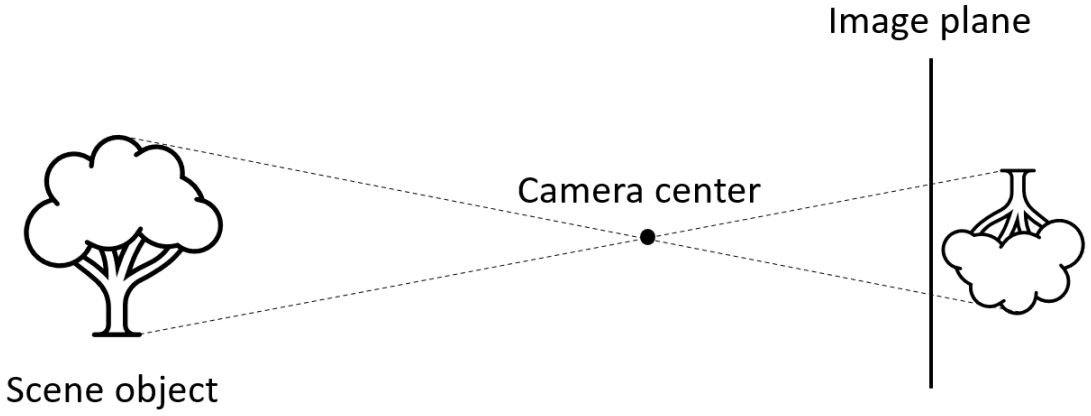


Figure 3: The pinhole camera with example rays shown in dashed black.

The pinhole camera model projects a 3D point in the scene to a 2D point in the image. This projection can mathematically be described by

$$\lambda \mathbf{x} = P\mathbf{X}, \tag{5}$$

where λ is a scalar which is needed to get the last homogeneous coordinate to have a value of one, $\mathbf{x}$ and $\mathbf{X}$ are the homogeneous image and 3D point coordinates respectively. P is the camera matrix which transforms coordinates between the 2D and 3D scene. P has dimensions 3×4 and is constructed from the intrinsic and extrinsic parameters of the camera. The matrix P is in turn the result of the matrix multiplication of two other matrices, the intrinsic calibration matrix K and the rotation matrix R and the translation vector t in the following way

$$P = K[R|\mathbf{t}]. \tag{6}$$

In (6), R has size 3×3 and $\mathbf{t}$ has 3×1. The calibration matrix consists of the inner parameters parameters of the camera

$$K = \begin{bmatrix} \alpha f & s & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{7}$$

where f is the focal length of the camera, α is the aspect ratio for non-square pixels, s is the skew and c_x and c_y is the principal point. The principal point $\mathbf{c} = (c_x, c_y)$ is often located in the center of the image, measured in pixels. The skew, s, is in most cases negligible and the aspect ratio are

in most cases very close to one [19, pp. 103-126]. With these assumptions the calibration matrix can be simplified to

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

where the only unknown parameter is the focal length.

2.2.2 Image distortions

Even though the pinhole camera model is accurate enough for most cases, the cameras used in the real world are a bit different. The pinhole camera assumes that just an infinitely small aperture is used without any lenses. That is not realistic since enough light needs to pass through to create the images. Therefore cameras use a larger aperture together with lenses to direct more light into the camera. However, these lenses distort the pinhole camera model in different ways. The two most common distortions are radial distortion and tangential distortion.

Radial distortion arises due to the shape of the lens. The effect is more distinct on cheaper cameras such as web-cameras and can also be seen in fish-eye lenses where it is a feature. Because of the shape of the lens, light rays further from the center will bend more than the rays at the center. Therefore pixels further away from the center of the image will be affected more. This can sometimes be seen on images of objects which have straight lines. These will appear to be bent in the image. The effect on each pixel can be approximated with a Taylor series expansion around the radius r from the center, usually three terms are used [1, pp. 375-378].

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{9}$$

$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{10}$$

$x_{corrected}$ and $y_{corrected}$ are the undistorted pixel values, x and y are the original values and k_1, k_2 and k_3 are three scalar parameters.

Tangential distortion is caused by wrongful assembling of the parts in the camera. If the lens is not placed parallel to the image plane these effects will occur. The effect can be calculated with the following equations where p_1 and p_2 are unknown parameters.

$$x_{corrected} = x + \left(2p_1 y + p_2(r^2 + 2x^2)\right) \tag{11}$$

$$y_{corrected} = y + \left(p_1(r^2 + 2y^2) + 2p_2 x\right) \tag{12}$$

2.2.3 Camera calibration

To use (5) when working with images, the calibration matrix must be known which means that the focal length is needed. To undistort the images and get rid of the radial and tangential distortions, another five parameters need to be calculated. Thus there are in total six unknown parameters. These parameters can be obtained by calibrating the camera.

A popular way to calibrate a camera is to take multiple images of a chessboard pattern from different angles and distances. The images are then processed with some software to extract the chessboard corners for all the images. These corners, which are in pixel units, will be stored together with the spatial coordinates for the corners. These are most times assumed to be in a fixed location in the z-direction, mostly zero is used, and then get x- and y-coordinates based on their location in a grid-based setting, (0,0,0) for the top left corner, (0,1,0) for the corner to the right and so on. With these mappings between pixel coordinates and spatial coordinates, homographies can be used. A homography is a projective mapping between the coordinates which can be carried out as a matrix multiplication. The homographies carry information about the intrinsic parameters of the cameras and by collecting several homographies and use some properties for these matrices both the intrinsic, extrinsic and distortion parameters can be obtained. For the interested reader the details can be found here [1,pp. 384-395].

With the distortion parameters being known, it is possible to compensate for the distortion effects in the images. At first a distortion map is calculated which maps the points between the distorted and corrected domain. This distortion map is based on (9)-(12). This mapping is then used on an image taken by the specified camera to obtain the undistorted result.

2.2.4 Coordinate rescaling to metric units

In images it is often desirable to extract different kinds of information. One kind of information which often times is important is the metric distances within an image. With a calibrated camera together with (5) and (6) it is possible to convert the pixel coordinates to metric values and then measure certain distances. However to do this knowledge about the distance from the camera to the object in the image is needed.

It is possible to combine (5) and (6) to

$$\lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = K[R|\mathbf{t}] \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \tag{13}$$

where x and y represents the pixel coordinates and X, Y, Z the spatial 3D coordinates for the pixel. The calibration matrix in (8) is used and both sides of (13) is multiplied with the inverse of K

$$\lambda K^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = [R|\mathbf{t}] \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \Longleftrightarrow \lambda \begin{pmatrix} 1/f & 0 & -c_x/f \\ 0 & 1/f & -c_y/f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = [R|\mathbf{t}] \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \Longleftrightarrow$$

$$\lambda \begin{pmatrix} \frac{x-c_x}{f} \\ \frac{y-c_y}{f} \\ 1 \end{pmatrix} = [R|\mathbf{t}] \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}.$$

For this new equation further assumptions are that the rotational matrix R is the identity matrix and the translation vector t are a vector of zeros. These assumptions are possible to use since the actual position in 3D space is arbitrary and the only thing of importance is to get measurements between two points in an image. The actual positioning of the 3D-points with respect to rotation and translation around the camera center is not of importance. The simplified equation can thus be written as

$$\lambda \begin{pmatrix} \frac{x-c_x}{f} \\ \frac{y-c_y}{f} \\ 1 \end{pmatrix} = [I|\mathbf{0}] \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \Longleftrightarrow \lambda \begin{pmatrix} \frac{x-c_x}{f} \\ \frac{y-c_y}{f} \\ 1 \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}.$$

In the new equation there are four unknowns, λ and the wanted parameters X, Y and Z. As mentioned in the beginning of this subsection, if the distance to the object from the camera is known, then the variable Z is known and therefore also λ. Thus the equation to obtain the last two spatial coordinates are simplified even further to the final equation

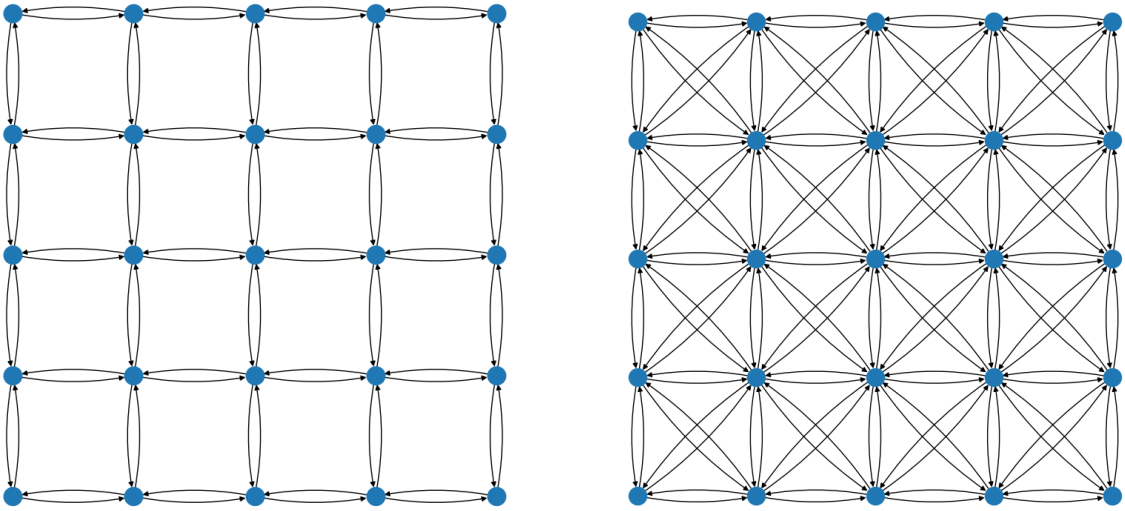
$$\begin{pmatrix} X \\ Y \end{pmatrix} = Z \begin{pmatrix} \frac{x-c_x}{f} \\ \frac{y-c_y}{f} \end{pmatrix}. \tag{14}$$

Since all the parameters on the right hand side are known, a mapping between pixel coordinates and spatial coordinates have been obtained. Thus in order to measure the distance between two parts in an image, the spatial coordinates, X and Y, can be calculated and used appropriately.

2.2.5 Segmentation using graph cuts

Segmentation in the context of image analysis entails separating a given image into two or more regions of pixels using some pre-defined algorithm. There are a multitude of methods available that are suitable depending on specific application of the segmentation. For the purposes of this work where a colored liquid is to be segmented from a plain white background, a robust algorithm that can handle input images that are slightly different with high reliability and without need to manually construct large training sets is wanted. One such method is *max-flow/min-cut segmentation* - more commonly referred to as *graph cut segmentation*. The idea behind graph cut segmentation is transposing the segmentation problem into an energy minimization problem, using the max-flow min-cut theorem to find the cut that minimizes the energy - corresponding to a separation between regions.

As the name suggests, an image is first converted to a graph - a structure with nodes connected in some way by edges. Essentially a graph is simply a way of representing data where there is some form of relation between instances of data. In relation to images a graph is most commonly constructed with nodes representing pixels and edges connecting a pixel to its neighboring pixels. Since pixels are structured in a square grid the most straight forward approach is connecting a pixel to its 4 neighbors vertically and horizontally, or its 8 neighbors - with diagonal edges as well. Figures 4a and 4b illustrates two example graphs constructed from a 5×5 pixel image with 4-neighbor and 8-neighbor edges.



(a) Graph with 4-neighbor edges

(b) Graph with 8-neighbor edges

Figure 4

Graphs are a versatile structure where in general nodes can hold essentially any type of data and edges are typically assigned a weight describing strength of the connection to its connected node. There is also the concept of directed or undirected edges, the former meaning the connection is one-way and the latter meaning the connection goes both ways. A cut in a graph constitutes removing edges.

For the purposes of the graph cuts algorithm we construct a graph with as many nodes as pixels of the input images, with bi-directional edges connecting nodes in a 4-neighbor or 8-neighbor setup. Two additional nodes, *source*- and *sink*-nodes often denoted S and T respectively, are then added to the graph with edges connecting from the source node to all pixel nodes and from all pixel nodes to the sink node. In other words these edges are directed with a clear general direction from source to sink if the graph is interpreted as a flow network. Figure 5 illustrates the constructed graph for a 5×5 input image with 4-neighbor connectivity.

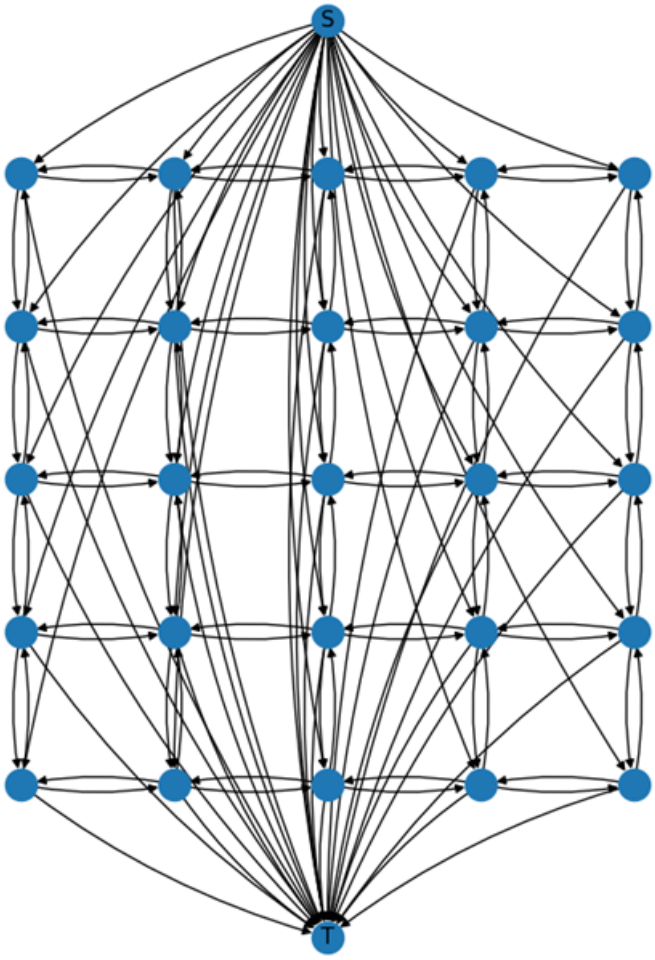


Figure 5: Graph with added source and sink nodes for a 5×5 input image with 4-neighbor connectivity.

The point of the algorithm is then to find a cut in the graph that separates the source from the sink node. The nodes contained in the two resulting graphs after the cut will correspond to foreground and background pixels of the image. To make the cut meaningful, energy is introduced as the sum of edge weights constituting a cut. For a cut, f, separating source from sink the energy of the cut is defined as

$$E(f) = \sum_{i} E_R(i) + \sum_{j,k} E_P(j,k) \tag{15}$$

where i denotes all pixel nodes and j, k denotes pixel nodes cut from other pixel nodes, with j on the inside of the cut and k on the outside [20].

Region energy E_R can be defined as,

$$E_R(i) = \begin{cases} R_S(f(S,i)), & \text{if connected to source node} \\ R_T(f(i,T)), & \text{if connected to sink node} \end{cases} \tag{16}$$

where $R_{S/T}$ is some region statistic dependent on likelihood of a pixel belonging to foreground or

background. The boundary energy term can be written as,

$$E_P(j,k) = \nu f(j,k) \tag{17}$$

with ν as some constant regulation parameter. Considering a cut f separating source from sink as illustrated by edges colored in red and green in figure 6 and marked edge weights. Red edges represent edges that are present in the E_R part of (15) and green the edges in E_P, usually referred to as data terms and smoothness terms respectively.

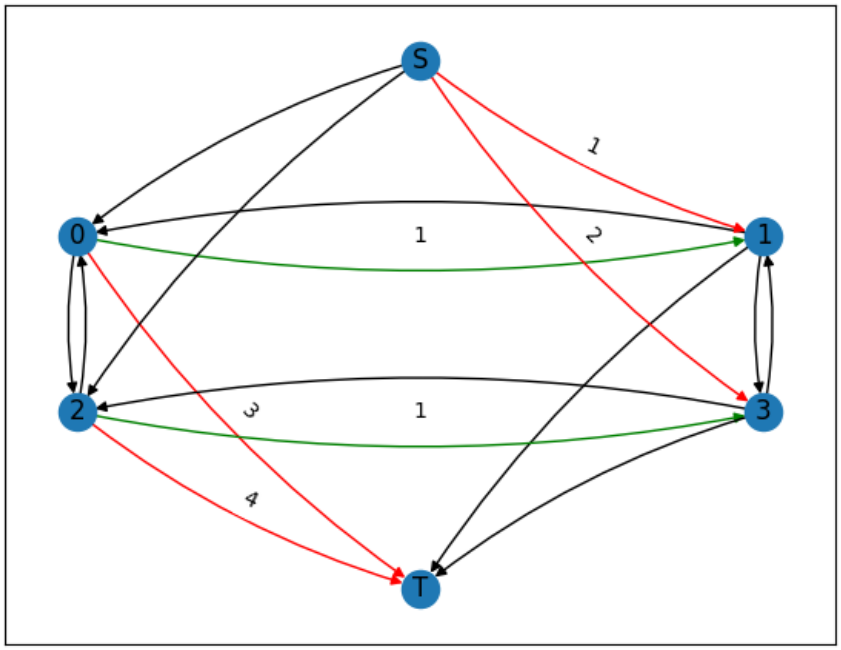


Figure 6: Illustration of a graph cut.

Calculating the energy of the cut in figure 6 with (15) would mean $i = 0,1,2,3$, $j = 0,2$, $k = 1,3$, ie

$$E(f) = \sum_{i=0,2} E_R(i) + \sum_{\substack{j=0,2 \\ k=1,3}} E_P(j,k) = \underbrace{R(f(S,1)) + R(f(S,3)) + R(f(0,T)) + R(f(2,T))}_{E_R} + \underbrace{\nu f(0,1) + \nu f(2,3)}_{E_P},$$

which with values as marked in the graph evaluates to $E(f) = 1+2+3+4+1+1 = 12$. In this example nodes $\{0,2\}$ would be segmented from nodes $\{1,3\}$. Considering that the goal is finding the cut that minimizes energy of the cut, the regulation coefficient ν determines how much the length of the cut is penalized. In other words this is a smoothness parameter that can result in jagged edges if too small, and excessive background segmentation if too large. For edges connected to source and sink in the context of finding a minimum cut means edge weights should be small in magnitude for foreground pixels connected to source and background pixels connected to the sink. Usually this is accomplished by setting edge weights using negated log-likelihoods of pixel feature

distributions that has been fitted beforehand on foreground and background examples, but can generally be that anything maps small values on edges between source and likely foreground, and between sink and likely background.

Remaining is to actually find the cut with minimal energy in a graph that disconnects the source from the sink. Historically this was done with gradient descent-based methods, which were susceptible to slow convergence and finding non-global minima [20]. Instead it is possible to make use of the *max flow, min cut* theorem which states the value of the maximum flow through a graph flow network is the same as the capacities (weights) of the minimum cut [16]. Solving the dual problem of finding the maximum flow then additionally gives the corresponding sum of weights for a minimum cut in the graph. There exists many algorithms designed for this purpose such as Edmonds-Karp augmenting paths or push-relabel algorithms, that solves the maximum flow problem in polynomial time dependent only on number of nodes and edges in the graph [4].

2.3 3D reconstruction

To create a 3D model from 2D images is a common problem in computer vision. There exists different methods to do this and in this subsection a few of these will be highlighted.

2.3.1 Stereo vision and structure from motion

Stereo vision and structure from motion are two similar methods which can be used to recreate a 3D scene from multiple images taken of the same object from different viewpoints. The biggest difference between these methods are that stereo vision tries to mimic the perception of the human eye. It uses two synced cameras to take images at the exact same time with just a horizontal difference between the two cameras. In structure from motion, the same camera is used to take the different images where the camera has been moved between the photos. In both methods, features are extracted from the images to map pixels between the images. These pixel correspondences is then used in calculations to obtain information and depth perception from the scenes.

An important aspect of stereo vision is that the cameras should have their image planes perfectly aligned. When they are aligned, the geometrical calculations and understandings of the algorithm is much easier. Figure 7 shows an example of the geometrical interpretation of the stereo vision algorithm. By using a known point correspondence in both images, knowing the distance T between the cameras and use the same and known focal length f on the cameras, the depth Z can be calculated by looking at similar triangles. The resulting formula is

$$Z = \frac{fT}{x_l - x_r}. \tag{18}$$

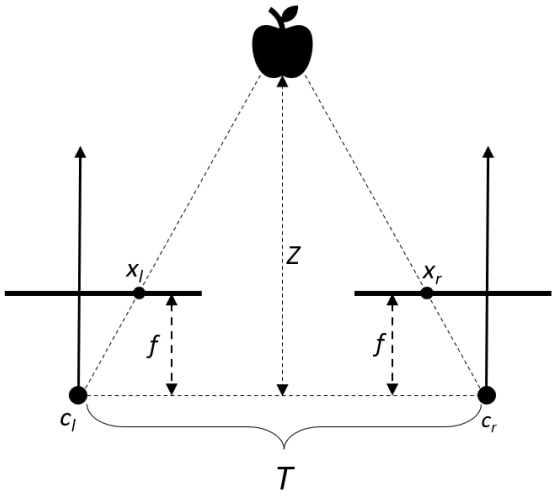


Figure 7: Geometrical interpretation of the stereo vision algorithm and what measures that are used.

The depth can then be calculated for all correspondences between the images to create a depth map. This depth map is used to create a 3D model of the scene. Although this simplified version and the assumption that the image planes are aligned is hard to achieve in reality, calibration and rectification can be used to achieve this effect. Calibrating the setup will relate the locations between the cameras in space and rectification will transform the images so that they appear to have their image planes aligned. This will allow for a faster and more accurate search for point correspondences between the images which will speed up the algorithm [1,pp. 405-458].

When instead inspecting the structure from motion algorithm the steps are a bit different. At first the point correspondences between the two images are calculated. With these points it is possible to calibrate the cameras and find the relative orientation between them. This is done with epipolar geometry which are geometrical constraints between the points which relates to the camera positions, the inner parameters of the cameras and the 3D points. The epipolar constraint

$$\mathbf{x}_2^T F \mathbf{x}_1 = 0, \tag{19}$$

is a constraint between matching points in the images. $\mathbf{x}_2$ and $\mathbf{x}_1$ are homogeneous image coordinates and F is the fundamental matrix. The fundamental matrix is the product of calibration, rotation and transformation matrices between the two cameras. However, it can also be calculated using only eight point correspondences via the eight point algorithm. Once the fundamental matrix has been calculated, the camera matrices for the two cameras can be extracted from the fundamental matrix. This is obtained by singular value decomposition and matrix multiplications. Once the camera matrices are known it is easy to use the camera equation, (5), to calculate the 3D-point for a set of image points. For the two cameras we get a system of the following form

$$\begin{pmatrix} P_1 & -\mathbf{x}_1 & 0 \\ P_2 & 0 & -\mathbf{x}_2 \end{pmatrix} \begin{pmatrix} \mathbf{X} \\ \lambda_1 \\ \lambda_2 \end{pmatrix} = 0.$$

Due to noise there might not be an explicit solution. However, an approximate least squares solution can be found with singular value decomposition to obtain the 3D coordinates, $\mathbf{X}$ [19,pp. 127-160].

2.3.2 Marching cubes algorithm

A very different approach to obtain a 3D-reconstruction is to use the marching cubes algorithm. It is widely used for medical applications to reconstruct 3D objects from CT och MRI scans. This algorithm creates a 3D model from a scalar field by making use of small cubes to identify the surface of the object.

The scalar field which describes the 3D-object to reconstruct can be thought of as a cube. Inside this cube, several smaller cubes are placed to fill the larger cube. Each of the smaller cubes has eight corners. For each cube the vertices are compared to the values in the scalar field. If they are in a region above a certain threshold τ, that vertex is considered to be inside the object and gets the value one. If the vertex is outside of the region it get a value of zero. By doing this for all vertices in the smaller cubes it is possible to label which are inside or outside the region. The edges between a vertex which is inside the object and outside are noted. This is to store information about the edges on which the triangles should be placed to create the surface. Since every cube has eight corners and every corner can be either inside or outside the object, there are 256 different possibilities. However if rotation and mirroring are taken into account there are actually only 15 unique cases for the smaller cubes. These situations are shown in figure 8 below.

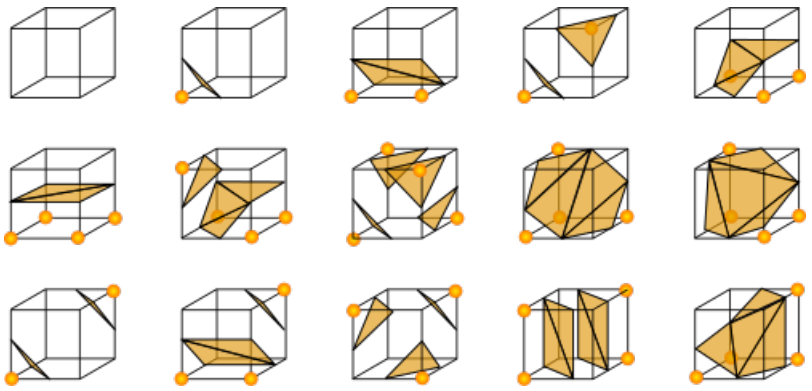


Figure 8: The fifteen unique cases being used in the marching cubes algorithm. Image source: [9]

Even though the vertices are labeled and the edges to connect to are known, it is relevant to know where on the edge the corner of the triangle should be placed. To solve this and make the mesh smoother, linear interpolation is used to find the proper distances. When all of this is done for all of the smaller cubes, a unit normal is calculated for each cube vertex using central differences and the triangle meshes and vertex normals are returned and used to plot the object as a 3D model [12].

2.4 Machine Learning and Neural Networks

Machine Learning (ML) is a general name for algorithms which can find patterns and learn from data. Two common ways in which these models are trained are via supervised or unsupervised learning. The difference between these two methods is that in supervised learning the data is

labeled, for example an image of a number has a label which states what number it is. In unsupervised learning the data is not labeled and the algorithms try to find patterns in different ways. A powerful branch of ML algorithms are the artificial neural networks.

2.4.1 Artificial Neural Networks

Artificial neural networks (ANN), is a machine learning method which got its name from the structural similarities to the neurons in the human brain. These networks have been shown to learn the behavior of complex functions with good results [5]. The easiest network consists of only one part, the perceptron, which maps a vector of inputs to an output value. The mathematical formulation formulation is

$$y = \varphi\big(b + \sum_{i=1}^{N} w_i x_i\big), \tag{20}$$

where y is the output, φ is the activation function, b is the bias term and w_i are the weights for each of the inputs x_i. A figure of the structure of the perceptron can be seen in figure 9.

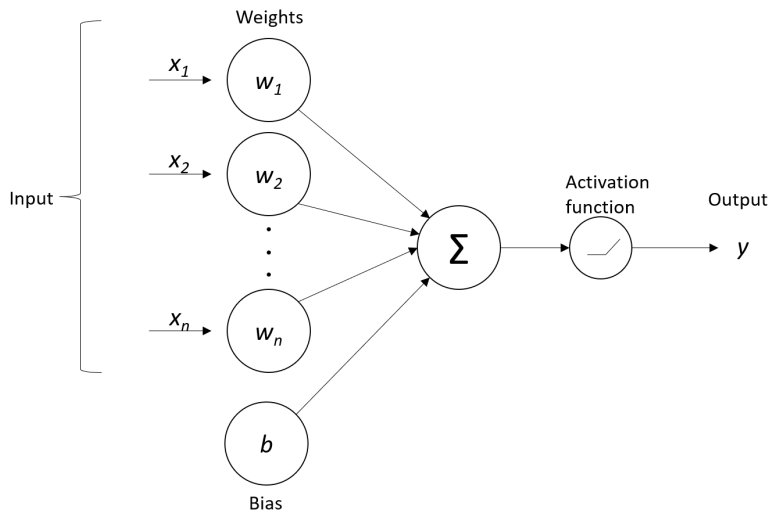


Figure 9: A perceptron with n inputs which produces a single output y.

If more nodes and layers are added, a multi layer perceptron (MLP) is constructed. The MLP can consist of an arbitrary number of input nodes which is mapped to an arbitrary number of nodes in a hidden layer. The number of layers and the number of nodes in each layer is specified beforehand and chosen with respect to the problem at hand. The final output can consist of a single node or multiple nodes depending on the application. A network where all nodes in the previous layer are connected to all nodes in the following layer is called a dense or fully connected network.

The activation function φ is used to introduce non-linearities to the network which will improve the performance. There are several options for activation functions but some of the most common are the $Sigmoid$, $ReLU$ and $Tanh$ activation functions which are shown below.

$$\text{Sigmoid: } \varphi(x) = \frac{1}{1+e^x}$$
$$\text{ReLU: } \varphi(x) = max(0, x)$$
$$\text{Tanh: } \varphi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The objective for neural networks is to minimize a loss function by updating the weights of the network. Practically this is done by training the network to minimize the loss function i.e. to make the predicted output for a certain input as close as possible to the true output. The weights are updated via backpropagation to minimize the loss. The details will not be covered in this work but can be found in [5].

2.4.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN), is a special kind of neural networks widely used when handling image data. The network uses the mathematical operation convolution to extract information from the input images. A convolutional layer consists of filters of quadratic shape, for example 3×3 or 5×5. The filters have trainable weights which will be updated via backpropagation during the training phase. These different filters move along the image and performs the operations which results in new images which highlights certain features in the image. The shape of these new image channels can be of equal size or smaller, depending on what kind of padding and stride being used. After the convolutional layer, it is common to apply some kind of pooling on the channels. Two common examples of pooling are max- and average-pooling. These operations can be seen as filters which are not trainable and are used to pick out the maximum or average value of the elements when sweeping over the image.

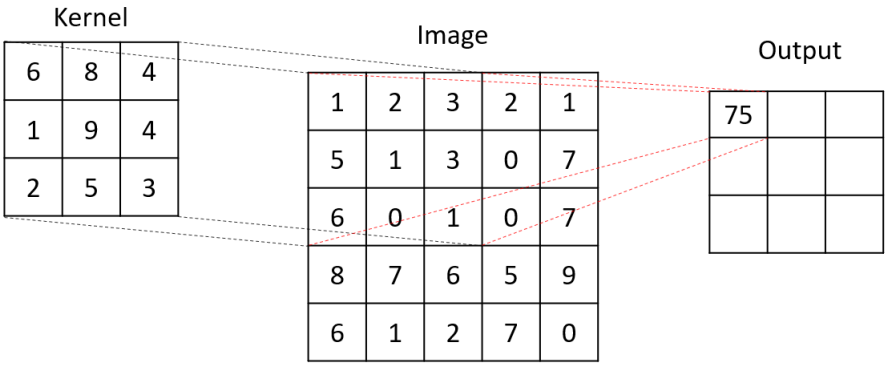


Figure 10: An example of a convolutional filter where a 3×3 kernel sweep over the first part in the image and creates a new smaller output of size 3×3. Both the number of filters (kernels) and image/output channels may be larger than one.

CNNs and the fully connected MLP described earlier are examples of feed-forward networks. This means that the information from a layer is used directly by the following layer in a forward passing sense.

2.4.3 Recurrent Neural Networks

Networks which use the information from earlier time-steps as input to later time-steps are common for time-dependent problems. Recurrent Neural Networks (RNN) is a kind of network which uses this technique, these networks are called feed-back networks. In a RNN the output from the network at time t is stored in a hidden state and serves as input together with the ordinary input vector at time $t+1$. A schematic figure of this can be seen in figure 11 where an unfolded version can be seen to the right.

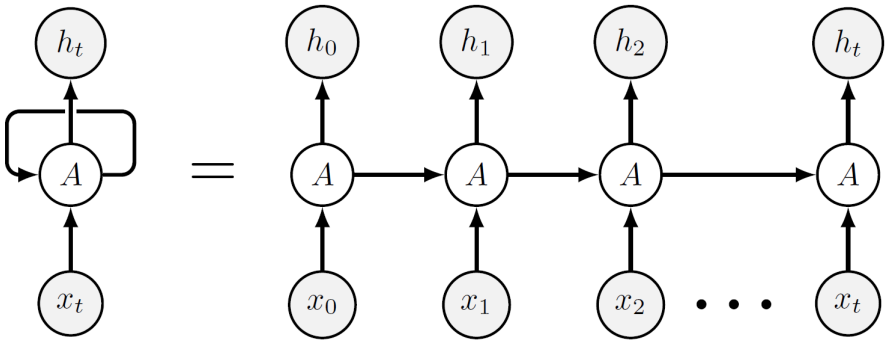


Figure 11: A simple explanation of an RNN. x_t is the input sequence and h_t is the hidden state or output. Notice the equivalence between the two ways of writing the network.

For longer time-sequences, RNNs suffer from the vanishing or exploding gradient problem. This is a problem which occurs due to the passing of information from earlier time-steps. Each time a hidden state is used in the succeeding time-step, it is multiplied by a weight. If the time-sequence is n times long, the backpropagation to update the weight is dependent on the gradient of the error functions to the power of n. If the value is below one, that value will disappear and the weight update for the earlier weights will thus be zero and the weights will not be updated. The opposite happens if the value is larger than one, then the value might tend to infinity and the gradient "explodes". This is a complicated problem which can cause the weights to update slowly or in worst case not at all [7].

2.4.4 Long Short Term Memory

A method to fix the problem with the vanishing and exploding gradients was proposed in 1997. The method is called Long Short Term Memory (LSTM). The network uses three gates, an input, an output and a forget gate to make the error flow constant when backpropagating and thus reduce the problem with vanishing or exploding gradients. The method also offers other benefits compared to the RNN as the time complexity per time-step and weight is of order $\mathcal{O}(1)$ which makes it a faster algorithm [7].

2.4.5 Convolutional LSTM

For time-dependent image sequences, an extension of the LSTM network called convolutional LSTM (ConvLSTM) can be used. The regular LSTM networks only uses a single vector as input. When using spatiotemporal datasets like image sequences, these first need to be flattened to a single vector before used as inputs. However in the flattening process a lot of the spatial dependencies are lost which leads to increased errors in the model predictions [13]. The ConvLSTM

cell however, works as an ordinary LSTM unit combined with the convolution operator and uses tensors as inputs. The inputs are processed by a number of convolutional filters before being passed on through the gates. This means that the input to such a network can be images and the dimension of the following gates will be of the same dimension as the inputs. Hence the spatial information is used in a correct manner in the model [18].

2.4.6 Graph convolutional LSTM

Depending on application, one possible drawback of the convolutional LSTM networks is the fixed structure of the input tensors. There might be farther reaching correlations in the data not fully captured by striding kernels in a square grid. As discussed in section 2.2.5, instead of images another possible data structure are graphs, which allow more general encoding of data coupling.

A similar approach as in convolutional LSTMs can be be applied to graph structures, where instead of traditional convolutional layers, there is instead a graph convolution applied to the data. The objective of the convolutional layers are still to create higher lever features for use in a dense, or in this case, LSTM layer. Defining a convolution on a graph can usually be done in one of two ways, spectral based or spatial based. The former has its foundations in graph based signal processing while the latter is more analogous to traditional 2-dimensional convolutions on images, taking the nodes connection architecture in account [24]. Figure 12 illustrates an example of a spatial graph convolutional layer on a small simple graph.

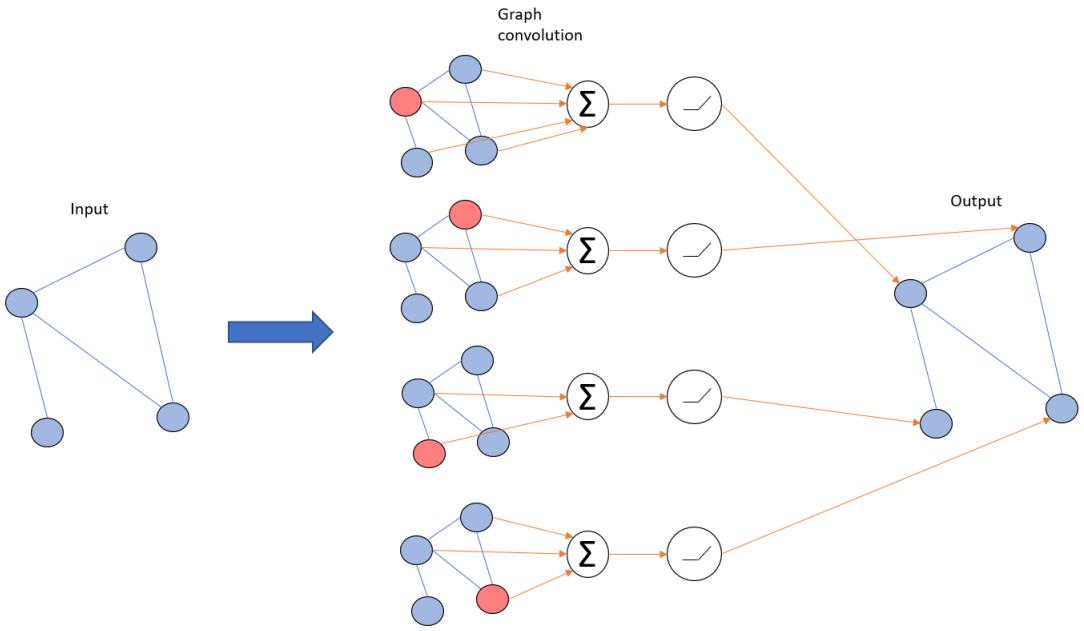


Figure 12: An illustration of a simple spatial graph convolutional layer.

The convolution operation depicted in figure 12 exemplifies how a spatial graph convolution can be applied using summation of neighboring node features. The $\sum$ operator can essentially be any way of unifying the connected nodes in a meaningful way. A simple and often used method is weighted summation of the incoming features. These weights are updated when training the model, which intuitively corresponds to strengthening or weakening a node's connection to its neighbors. A spatial graph convolution layer as a whole can be seen as message passing between

nodes where the output feature graph incorporates information from its neighbors, and by adding multiple layers information can propagate further in the graph than the original input graph [24].

Another possibility more analogous to regular CNNs is aggregating incoming node messages with for instance a mean, sum, or max operation, and then applying matrix multiplication with a matrix of learnable weights. This is essentially the same as a kernel in 2D CNNs, where shape of the kernel determines how many channels are propagated to the output graph. The GC layers can in principal be used very much the same as one would use a traditional 2D convolution layer, for example accompanied by pooling layers to downsample the output graph, and for our purposes - an LSTM layer. Specifically, the node features can hold essentially any type of data structures, like time series. A graph convolutional LSTM network is a practical architecture when modeling time series that are correlated in ways that can be described using graph structures. For instance in traffic speed forecasting on urban roads [25].

2.4.7 U-Net

A well known network architecture created for biomedical image segmentation in 2015 by Ronneberger et. al is U-Net [17]. The network consists of a contracting and an expansive part which visually makes the network look like the letter U. The contracting part consists of several successive levels consisting of two convolutional layers, each with ReLU as activation function. The resulting channels are then stored in order to be used in the expansive part later. This is called a skip-connection since the output from an early part in the network is stored and used as input to a later part. The output from the convolutional layers are besides being stored, also followed by a max-pooling operation before entering a new level. The number of channels are doubled for each new level in the contracting part. This means that the size of the input image is reduced but the number of channels will increase by each level, extracting relevant features from the image. The expansive part can be seen as an inverse to the contracting part. From the resulting feature maps at the bottom of the network, each level in the expansive part consists of an up-convolutional layer to halve the number of feature channels. It then concatenate the stored result from the corresponding contracting level with the up-convoluted result before applying two convolutional layers which use ReLU as activation function. The number of levels are the same as in the contracting part. The final layer then uses a convolutional layer of appropriate size to get the correct number of output channels [17]. An overview of the network architecture can be seen in figure 13.

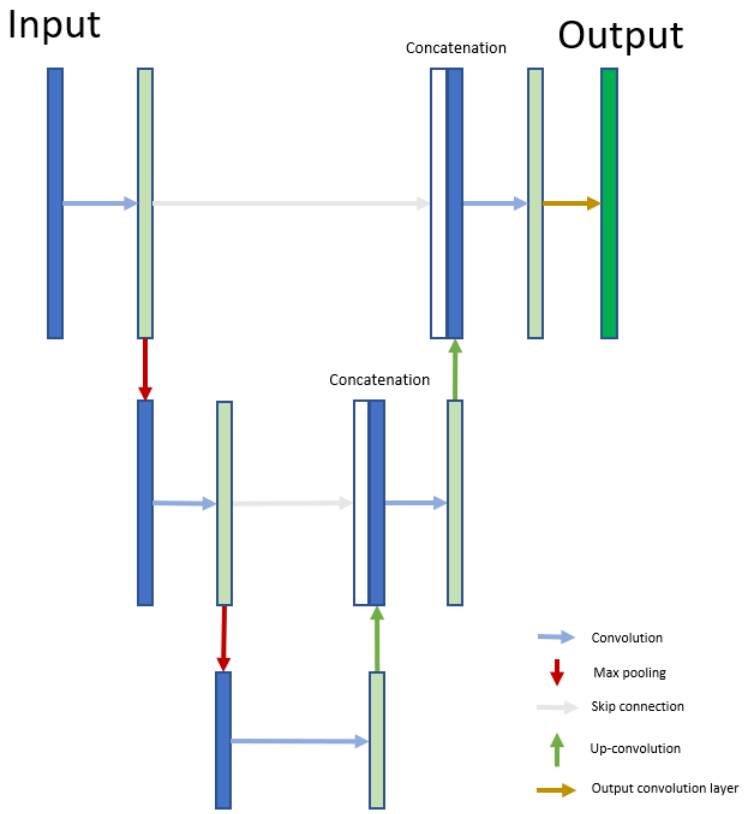


Figure 13: The structure of a U-Net with depth three. The skip-connections are concatenated to the channels in the expansive part.

The structure of the network have been proven to make the model very efficient. The U-Net model does not need a huge amount of data to work well but instead can be trained on rather few samples of data and still yield good results. The authors also believed that the network could be used for other applications [17].

Since 2015 U-Net have been implemented for other applications as well. One application which it has been tested on is the possibility to model Reynolds-Averaged Navier-Stokes (RANS) simulations around airfoil flows. Usually CFD simulations are used to calculate the pressure and velocity distributions around different airfoil shapes. However experiments have been made on several different NN architectures including U-Net with promising results [21]. Further improvements has also been made to the U-Net architecture in order to obtain even better results for the RANS prediction of airfoil flow. One proposed method is to extract other geometrical features such as angles and the Reynolds number which are concatenated at the bottom of the U-Net with a flattened version of the feature maps. This concatenated vector is then used as input to a dense MLP which outputs a vector of appropriate shape to be resized to the same shape as before the concatenation. In this way more geometrical data can be used to further improve the results and it has been proven to be more accurate than just using a regular U-Net for this specific problem [3].

2.4.8 Optimization and hyperparameter tuning

During the training of a neural network model the goal is, as mentioned previously, to minimize the loss function. The problem is thus an optimization task and during training an optimization

algorithm is used to minimize this loss by updating the weights in the network. A common algorithm upon which many extensions are based is stochastic gradient descent (SGD). In SGD the gradient of the loss function is calculated with backpropagation. As the direction of the gradient is the direction in which the function increase most rapidly, the weights are updated by taking a step in the negative direction of the gradient. The length of the step is decided by the learning rate which is a non-trainable parameter choosen beforehand [14]. Some examples of extensions of the SGD algorithm are root mean square propagation (RMSProp) and Adam.

When creating a neural network for different kinds of problems, there are a lot of choices to make. Besides the structure of the network and its layers there are other non-trainable parameters which needs to be chosen. These non-trainable parameters are called hyperparameters and needs to be selected for the specific problem at hand. Examples of hyperparameters are the number of layers in a MLP, kernel size in a convolutional layer and the learning rate for the optimization algorithm. The hyperparameters are usually determined by trial and error and evaluating the obtained result for the specific set of hyperparameters. It is also common to use grid-search to try different parameters in a certain range [14].

3 Methodology

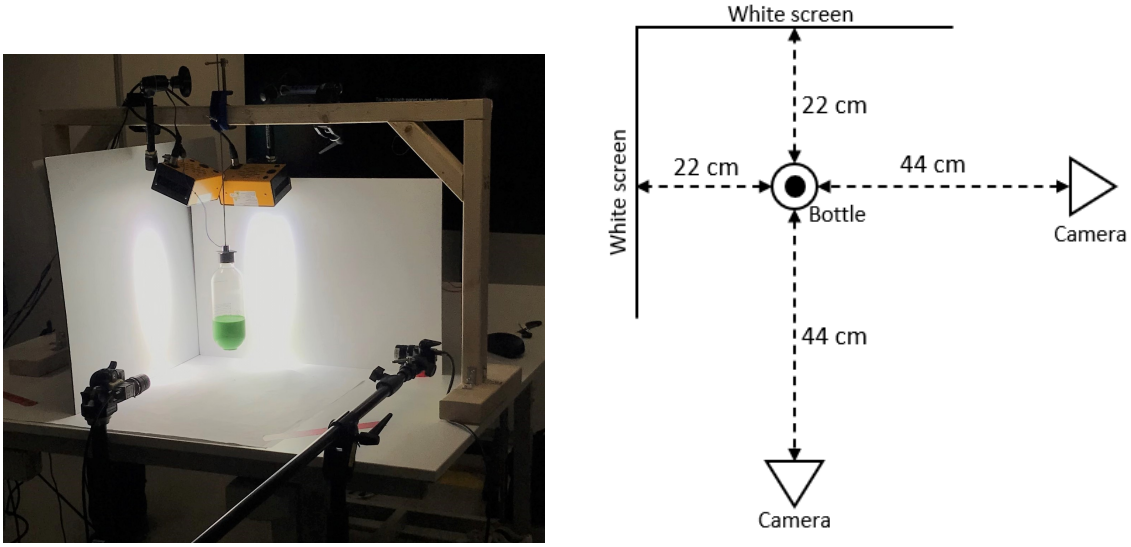
In this section the different parts of this thesis will be described in more detail. In section 3.1 and 3.2 the general framework is described. A framework where drop tests were first filmed and each sequence was stored. For each sequence the liquid was then segmented from the background with image analysis techniques. The segmented profiles from the two cameras were then used to create 3D pointclouds of the sequence. These 3D pointclouds were then used to train ML models. Section 3.3 is dedicated to the model building and training and finally in section 3.4 different assumptions and limitations are shared.

3.1 Data generation

Since the goal of the thesis is to increase the understanding of the behavior of liquids during a droptest, the data gathering was of great importance. Since there was no available dataset we wanted a robust and fast algorithm which allowed us to capture multiple sequences of droptest data and process them in a convenient way. The data could then both be analyzed manually and used in ML models. With these thoughts in mind the setup for the collection of data was constructed.

3.1.1 Experimental setup

In order to capture information to make a 3D-reconstruction of the liquid behaviour during the drop possible, a rig for the experiments was built. This was built to increase the repeatability of the drops as well as obtaining good conditions for extracting the relevant data. In figure 14 the setup can be seen both during a test and as a schematic overview from above.



(a) The setup being used in action.

(b) A schematic overview as seen from above. Note that the lamps and the wooden rig is not in the picture.

Figure 14

The setup consisted of the following items.

- Two Basler Ace 2 USB color cameras with identical objectives

- Two camera tripods
- A homemade wooden rig for attaching parts to
- Shelly PLUG S wifi power socket
- Electromagnet for up to 2.5 kg
- Two GSVITEC MultiLed LT lamps with 7700lm
- White screens used as background
- 1 liter SodaStream bottles
- Steel plates of size 4x4 cm
- Various brackets and rods for the mounting

As can be seen in figure 14b, the cameras were placed at an equal distance from the drop-point but spread 90 degrees apart. They were also placed on the same height of 12.5 cm above the table. All other distances such as the distance between the drop-point and the white screens are identical for the two cameras as well. The lights were placed in a setting in which they were directed directly onto the white screens to light up the background. In our opinion this was a good way of creating a contrast between the background and the liquid, as well as minimizing the reflections on the bottle. The electromagnet was placed on a metal rod placed right above the drop-point. It was possible to move the rod, and hence also the electromagnet, vertically by unscrewing a wing nut.

To film how the liquid behaves in some kind of container requires the container to be transparent. Therefore transparent SodaStream bottles with a volume of one liter was used. These were not only preferable due to the transparency but also because of their solid structure. However, these bottles have non-transparent plastic at the bottom to keep them standing upright. This part of the bottle was removed such that the whole bottle was transparent, except for some black print. In order to see the different liquids easier, food coloring was added to the liquid to make it more distinguishable. In order to drop the bottles in a convenient way, steel plates were glued to the top of the bottle so that the electromagnet could be used to hold it at a fixed distance over the surface. The metal rod holding the electromagnet was kept in a vertical position so that the bottle would hit the surface without an incoming angle. The electromagnet was in turn connected to the Shelly PLUG S power socket for activation.

The bottles were filled with three different kinds of liquids with slightly different properties. The three types were water, milk and yogurt. Their different liquid properties as well as Tetra Pak's® common use of these products qualified them as the liquids to use in this thesis. For each of the different liquids, drops were executed and filmed for different heights and with different amount of liquid in the bottle. In table 1 the different drop-heights, amount of liquid being used and also the number of drops per setting is shown.

Table 1: The different drop combinations for the different liquids

	Water	Milk	Yogurt
Drop-height (cm)	2, 4, 5, 6, 8	2, 4, 5, 6, 8	2, 5, 8, 15, 22, 26, 30
Fill level (cm)	4.5, 6, 8, 10	4.5, 6, 8, 10	4.5, 6, 8
Number of drops per setting	5	5	5 up to 15 cm drop, the rest 3

The drop-height was measured in centimeters from the table to the bottom of the bottle. The amount of liquid is measured in centimeters from the bottom of the bottle.

3.1.2 Camera calibration and specifications

When the cameras were put in the correct place they were calibrated to find the camera parameters. To calibrate a camera, 14 images of a chessboard pattern were taken from slightly different angles and distances. This was made by taping the pattern onto a flat screen which were moved between the images. The images were then processed by a Python script which used the `OpenCV` library and followed closely the example code in the `OpenCV` documentation [15]. Inner parameters, distortion coefficients and rotation and translation vectors were thus obtained for each camera.

The two cameras were identical hardware wise and even the objective were the same for both cameras. Due to the need of taking sharp photos even though the object was in motion, a short exposure time was used. Since shorter exposure times limits the amount of light the camera receives, the two strong lamps were needed. The exposure time was set to 500 microseconds for water and yogurt and 1000 for milk. The values were tested out with respect to the lighting and the different liquids such that the resulting images did not contain any reflections and that there was a clear difference between the liquid and the background. The framerate of the cameras were 160 frames per second and the images were of resolution 1200×1920.

3.1.3 Software setup for image sequence capture

After the calibration of the cameras it was time to gather the data by filming the drops. As mentioned previously in this section, it was of importance that the data gathering part was robust, fast and had a good repetitiveness. Therefore a lot of time was put into creating a Python script which with the press of only one single button could start the dropping process. The script had the functionality to turn on and off the electromagnet, changing camera settings such as exposure time for different liquids, start taking photos in the appropriate time and also know when to stop taking photos. It also saved the files in appropriate folders with clear names. In this work the data was stored as images directly to ease for later use.

When running the script, the first part is to enter the information about the drop. The information needed is the type of liquid, the height and the fill-level. This information creates a folder for this specific setting where the images will be saved. Thereafter the connection to turn on and off the electromagnet is established. The computer is connected to the Wi-Fi network of the Shelly Plug S power outlet, this enables connection between these devices. With the Python module pyShelly, the Shelly Plug can be monitored within the script. Once the connection has been established the magnet is turned on and the bottle is attached to the magnet.

The two cameras are connected to the computer via USB. The manufacturer Basler has provided a Python library, `pypylon`, to control the cameras directly within a Python script. Once the connection to the cameras within the script was established, settings such as exposure time was set automatically depending on the type of liquid being used. The cameras then grabbed images with a framerate of 160 frames per second. Capturing images synchronously for cameras on two USB ports is not supported in `pypylon` directly, instead a sequential image capture between the two cameras is unavoidable. However, by temporarily storing captures in the cameras' internal buffer the time difference between captured images became negligible.

The focus in this thesis was to model the water surface between the first and second impact. When taking the photos it was therefore pleasant that only the needed amount of photos were taken between the start of the drop and until the second bounce. For this problem a function was written to calculate the number of photos the camera should take to capture the sequence up to the second bounce. The function takes the drop height, elapsed time since the drop and coefficient of restitution as input and outputs current height and number of frames till the first and second impact. The calculations are based on the assumption of a free fall without any resistance. Since the bottle is initially at rest, the velocity and the acceleration is zero at the beginning of the drop. When dropped, the acceleration is only affected by the gravitational force, regardless of the weight of the bottle. Therefore the velocity before the impact as well as the distance moved can be described by

$$v_{before} = t \cdot g,$$
$$d_{before} = \frac{g \cdot t^2}{2},$$

and thus the time and hence also the number of frames until the first impact can easily be calculated for the different drop heights. During the first impact the bottle loses kinetic energy to the table due to the collision not being perfectly elastic. The coefficient of restitution was in our case calculated to be approximately 0.5. The velocity in the upgoing direction could with this information be calculated and the trajectory between the first and second impact was only affected by this velocity and the gravitational force. With the time from the drop to the second impact calculated it was just to multiply it with the framerate to obtain the number of images for the camera to take. In figure 15 below, the height above the table for a drop from eight centimeters is shown with respect to time.

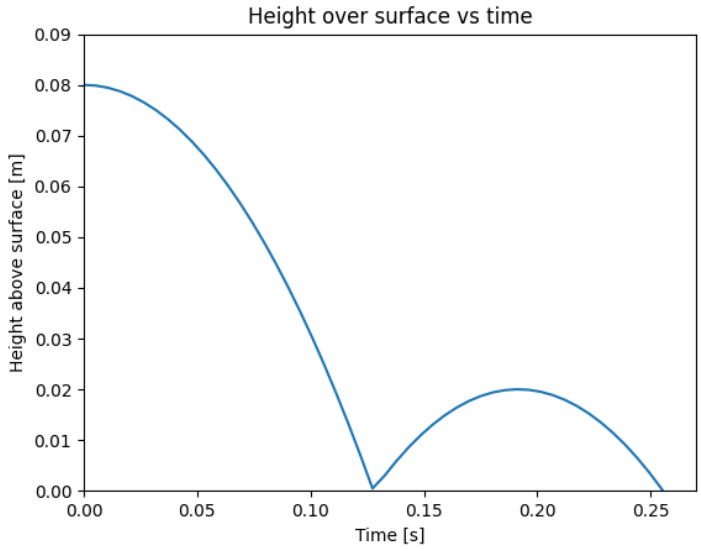


Figure 15: Height over surface vs time for a drop from 8 cm.

To summarize, the script is started and creates a folder to store the data based on the initial inputs the user types in. The electromagnet is turned on and the bottle is attached. With the press of a button the electromagnet is turned off and the cameras start taking the appropriate

amount of photos and store them in the created folder for later use.

3.2 Data handling

With the setup described above multiple drop tests can be performed in a controlled and repeatable procedure. Figure 16 demonstrate a captured image sequence of water dropped from 6 cm with a 6 cm fill level. This section will go over how data was collected from drop sequences.

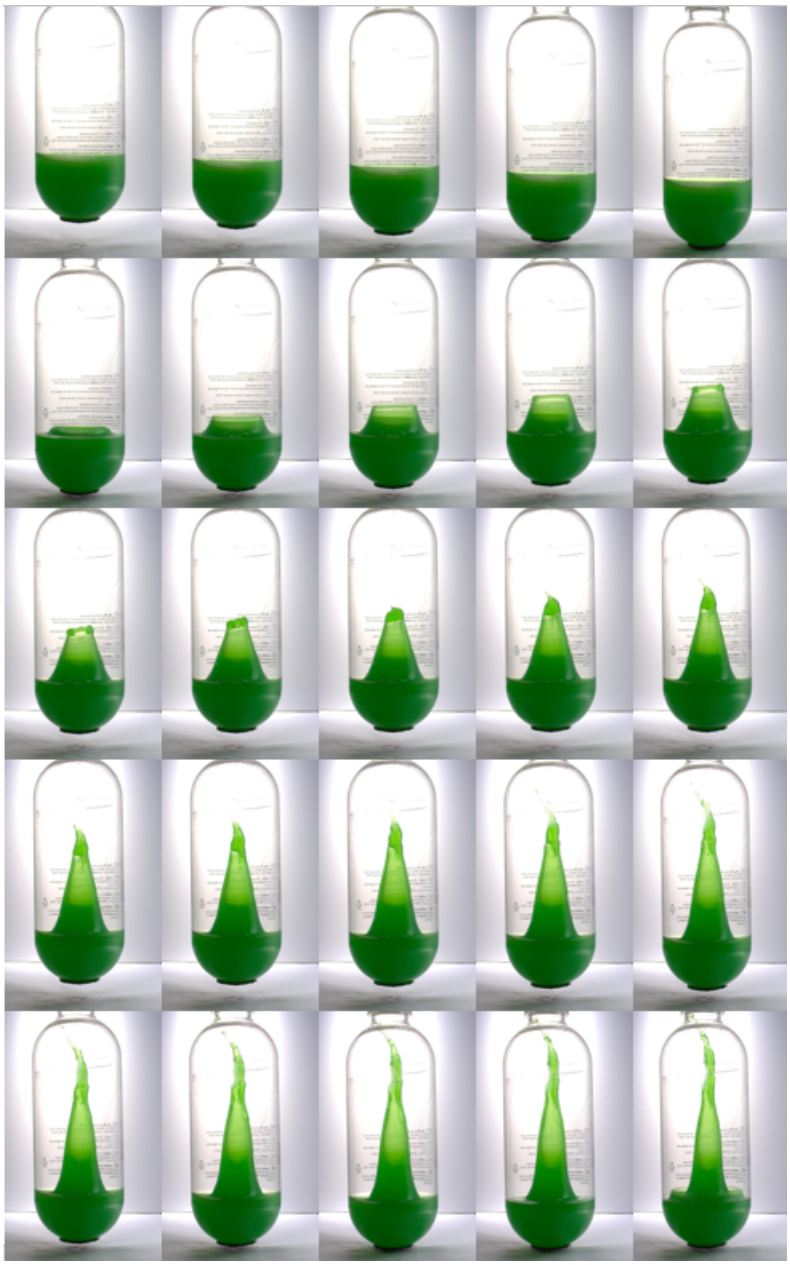


Figure 16: Image sequence of water dropped from 6 cm with a 6 cm fill level.

3.2.1 Graph cut implementation

With image sequences of drops obtained the first step in the data handling process was segmenting liquid from background. The algorithm designed for this was based on graph cut segmentation

described in section 2.2.5.

First a graph was constructed with the same number of nodes as the input image. Because the graph cut algorithm's running time is proportional to number of nodes and edges in the constructed graph, the input images were downsampled uniformly from 1200×1920 pixels to 150×240 pixels and a 4-neighborhood connectivity between nodes was used. There is a trade-off here between reasonable running times and resolution in the segmentation.We found that 150×240 pixels and 4-neighbor edges was sufficient to capture the overall shape of the liquid.

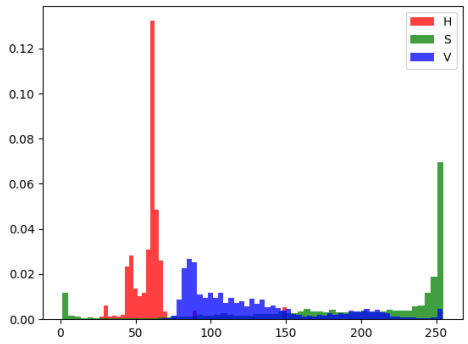
The pixel part of the graph is then created by setting bi-directional edges between pixel nodes with weight of the chosen regulation coefficient ν. Eventually it was settled at $\nu = 0.01$. The low magnitude of ν does result in segmentations that are non-smooth around the edges, however since robustness in the accuracy of the shape was valued more than fine details it was decided that a small ν was suitable.

For the edges connecting source and sink the aim is assigning small values between source and node for pixels that are likely foreground, and vice versa for edges between sink and likely background. One approach is negated log-likelihoods of normalized probability distributions fitted from pixel values of a training image. This pixel statistic will approach zero for values more likely to be of foreground/background and approach infinity for pixels not likely to be of foreground/background. Figure 17 illustrate a manually segmented typical drop image for a drop of water.

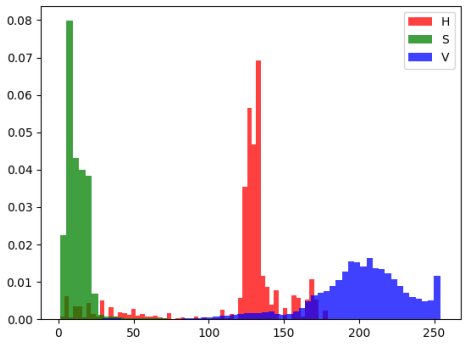
Figure 17: Segmentation training image. Original image (left), Manually segmented foreground (middle), and manually segmented background (right).

The most straightforward approach would be fitting 3-dimensional Gaussian (or other) distributions on non-white RGB-values on foreground and background pixels as illustrated in figure 17. However because of light settings in the setup and transparency of water, the overall brightness varies considerably in liquid pixels. Specifically, this means pixels that are clearly green to the human eye are not obviously green from a Gaussian distribution of RGB values point of view. Fortunately, there are other color spaces that more closely resemble how humans perceive color. HSV and CIELAB are two examples that transform the three channels of RGB images to channels that are in some sense more natural in color perception. In the HSV color space that image is

converted to hue, saturation, and value (brightness) channels. In CIELAB the transformation is to one channel describing perceptual lightness, one for distinguishing between red and green, and one distinguishing between blue and yellow. Figures 18a and 18b illustrates histograms of HSV channels for non-white pixels in middle and right of figure 17.



(a) Histograms of HSV channels for liquid pixels of water.



(b) Histograms of background pixels.

Figure 18

Inspecting the histograms with the intent of finding channels that clearly separates liquid from background in the HSV color space led to the decision to fit a Gaussian distribution on the H-channel on liquid pixels, and on the V-channel on background pixels. This because separation in the color space is adequate and the amount of outliers are low. Another choice potentially could have been to use the S channel, however outliers around zero in figure 18a appear around the edges of liquid/bottle in the input images - leading to strange results in the subsequent segmentation in theses areas of the input image.

While the H-channel and V-channel ranges 0-180 and 0-255 in the `OpenCV` implementation used, these were normalized between 0-1 for better comparisons with probability distribution values. Parameters for water segmentation were fitted as,

$$\begin{aligned} \mu_L &\approx 0.329, & \sigma_L &\approx 0.113 \\ \mu_B &\approx 0.782, & \sigma_B &\approx 0.132 \end{aligned} \quad .$$

Figure 19 illustrate the fitted distributions.

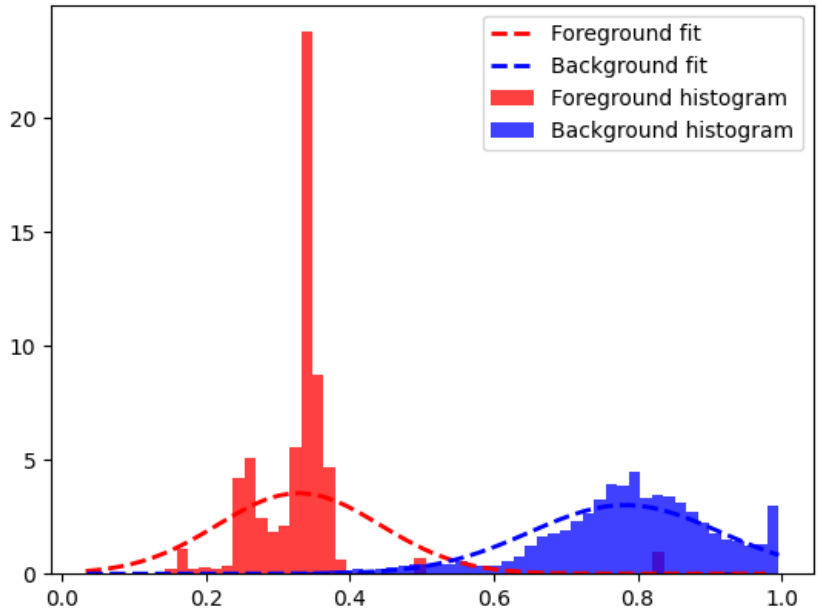


Figure 19: Histograms of foreground and background H- and- V-channels and the fitted Gaussian distributions.

The source and sink nodes are then added to graph, with directed edges from source to pixel nodes and from pixel nodes to sink. The weights of these are added as the square of the negated log-likelihoods of the probability distribution values forced in range [0,1] - to avoid negative log-likelihoods,

$$w_{S\to i} = \left(-\log\left(\frac{\mathcal{N}(H(i);\mu_L,\sigma_L)}{\mathcal{N}(\mu_L;\mu_L,\sigma_L)}\right)\right)^2, \tag{21}$$

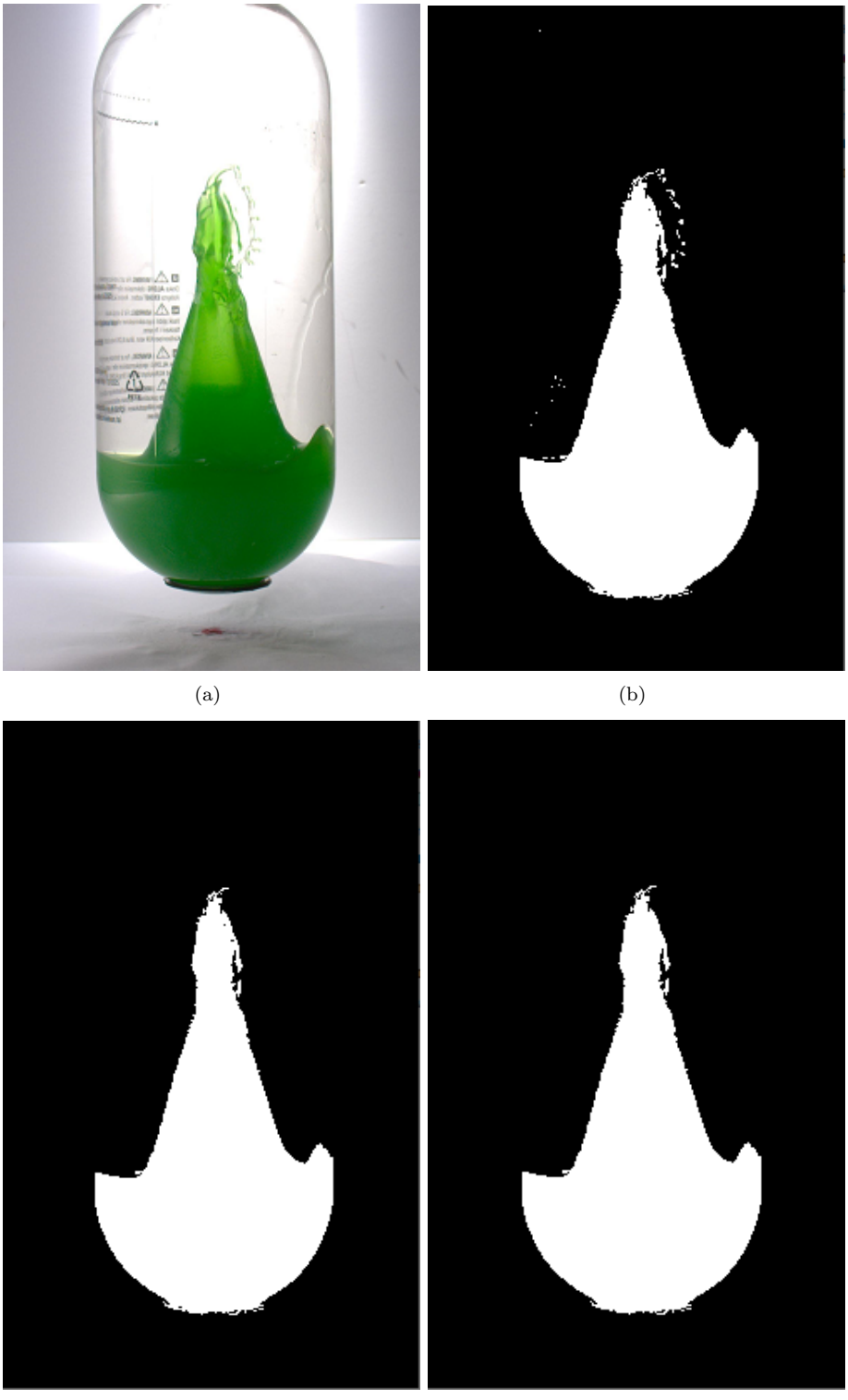
$$w_{i\to T} = \left(-\log\left(\frac{\mathcal{N}(V(i);\mu_B,\sigma_B)}{\mathcal{N}(\mu_B;\mu_B,\sigma_B)}\right)\right)^2, \tag{22}$$

where i denotes pixel nodes and $H(i), V(i)$ corresponding H and V values in the HSV color space of the input image. The weight assignments above is the region statistic R described in section 2.2.5, in which the importance is small weights connecting wanted foreground to source, and wanted background to sink. Thus (21) and (22) were empirically found to produce the most robust results. Because of shadows in some drops being segmented as foreground at the moments of impact, an added step of forcing foreground/background was also added. This can be done by entering weights of zeros instead of equations 21 and 22 when forcing foreground and background respectively. It was found that these problem areas of the input images were not distinctly separable in the HSV color space, they were however distinguishable in the a-channel of the CIELAB color space. Thus a condition of forcing background when pixels had a-channel values larger than 122 was added.

Having constructed the graph it is then sent to a maximum flow solver which returns the maximum flow from source to sink through the graph. Using `networkx`'s `minimum_cut` function it is possible to also obtain the corresponding minimum cut - the resulting segmentation is then

assigned foreground or background depending on which side of the cut a pixel node is.

In order to further improve the segmentation from the graph cut segmentation, two additional steps were added. First, because of splashes, text on the bottle sometimes being labeled foreground, and general noise in the resulting segmentation - only the largest connected segment was considered foreground. Second, because the bottle is not perfectly transparent there are sometimes reflections that are labeled background. This is mended by filling all holes in the foreground segmentation as foreground. The full chain of an example image segmentation is illustrated in figure 20.



(a) (b)

(c) (d)

Figure 20: (a) Input image (b) Segmentation after graph cut (c) Segmentation after largest segment (d) Segmentation after filling gaps

The same overall procedure is adopted for segmentation of milk drops, with different fitted distribution parameters.

3.2.2 From segmentation to 3D point cloud

To create 3D point clouds from the data, the marching cubes algorithm were used together with the segmentation from the two cameras for each pair of pictures. This method for creating a 3D version of the data was used for its robustness and easy implementation for this specific problem. To use either stereo vision or structure from motion, several features has to be extracted and matched between the two images. For our purpose we believed it would be hard to find corresponding feature matches between the two images since the surface of the liquid is not that uniquely determined with respect to lighting and reflections etc. In stereo vision only a small horizontal change would have been allowed as well which was not found to be a good enough setting in this thesis.

The use of the marching cubes algorithm in Python was very convenient. The function could be loaded from the scikit-image library and takes a scalar field as an input. In this particular case an array of shape [150,150,240] was created in which the segmented images were used for the full depth in the x- and y-direction respectively. This scalar field were then used as input in the function and the obtained results were the vertices, faces, normals and values. The most frequently used part in the rest of the thesis are the vertices which was in pointcloud format. However, one thing to note is that the resulting pointcluds will have a squared base and not a round shape. This is because of the fact that only two cameras are being used and both a cylinder and a cuboid have the same profile in this case. The squared shape was not regarded as a big problem since the overall characteristics were still captured.

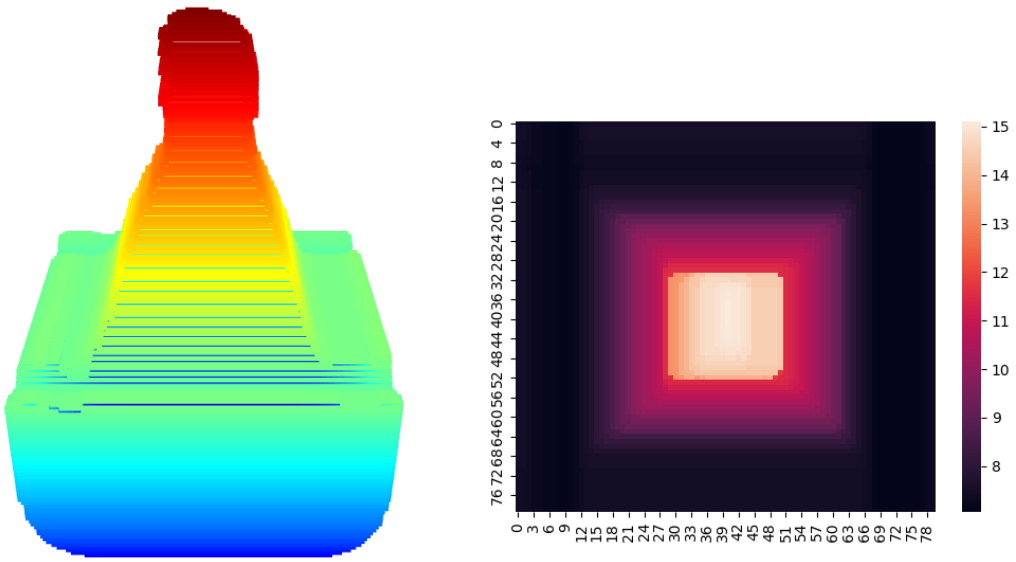
3.2.3 Rescaling of data

The pointclouds obtained from the marching cubes algorithm contain the overall shape of the liquid but the values for each of the points are in pixel-units. For an easier interpretation it would be pleasant if the values were in metric units. Since the shape of the bottle could be measured, the values in the x- and y- directions could be set to be between 0 and 8.9 cm. The height of the liquid during the droptest is a bit trickier since it changes from frame to frame and needs to be calculated for each frame individually. With calibrated cameras and with a known distance between the camera and the object, as in our case, the method from section 2.2.4 can be used to calculate the metric coordinates for the height of the liquid.

For each non-scaled pointcloud, the coordinates at the lowest and highest point was extracted. With (14) and the use of the parameters in the calibrated cameras, the pixel coordinates extracted from the lowest and highest point and the known distance Z from the camera, the metric coordinates for the two points were calculated. The difference between the Y coordinates is the height at the top of the peak in metric units if we put the height to be zero at the bottom of the bottle. Finally, since all of the metric values were known, the coordinates at the pointcloud was rescaled to these new values.

3.2.4 Heatmap creation

Even though the data now was in pointcloud format with the correct metric units, the choice of model types and their input options compelled us to store the data in a different way. The solution was to only extract the top layer for each pointcloud and store it in a 80×80 grid as a heatmap. The $x-$ and y-coordinates represents the spatial position and the value is the height in metric units. In figure 21, a pointcloud and the corresponding heatmap is shown.



(a) Pointcloud. (b) Heatmap.

Figure 21: Pointcloud and the corresponding heatmap for a drop with fill-level 8 cm dropped from a height of 4 cm.

3.3 Model building and training

3.3.1 General training framework

From the obtained data only the drops with water and milk were used during the training of the ML models. The behavior in yogurt varied too much from the other liquids, which will be shown in the next section, and it was therefore disregarded during the training phase. The data from the drops with water and milk was split up in a training, validation and test-set in order to train and evaluate the models in a correct way. At first a test set was extracted which consisted of all 40 drop sequences from both water and milk which was dropped from a height of five cm. The remaining drops were then divided into a training and a validation set. Four drops from each setup was put in the training set and the fifth in the validation set. The training set therefore consisted of 128 drop sequences and the validation set of 32. Each dataset consists of a time-series of heatmap data. Only the data after the initial impact and up to the second impact was used when training the models. This caused the drop-sequences to consist of varying samples for the different heights.

3.3.2 General model building guidelines

The models had access to four different features except for the heatmaps. These were elapsed time measured in frames, initial drop height, fill level and the viscosity of the liquid. For water a viscosity of 1 was used and for milk a value of 7.5 was used.

The main idea when creating the models was that they should be able to predict the surface of a drop sequence only based on three given features, the initial drop height, the fill level and the viscosity. For two of the proposed models this was possible while one of the models did not use any features but instead used the first frames in a sequence to predict the continuation.

3.3.3 Modified U-Net

The first model to be constructed is a modified U-Net model. It consists of a U-Net model of depth five with a dense layer at the bottom where four features are being concatenated to a flattened version of the channels at the bottom of the model. The four input features are time, viscosity, drop height and fill level. The model is then being reshaped to the correct size before being upsampled in the right part of the model. A detailed overview can be seen in figure 22 below.

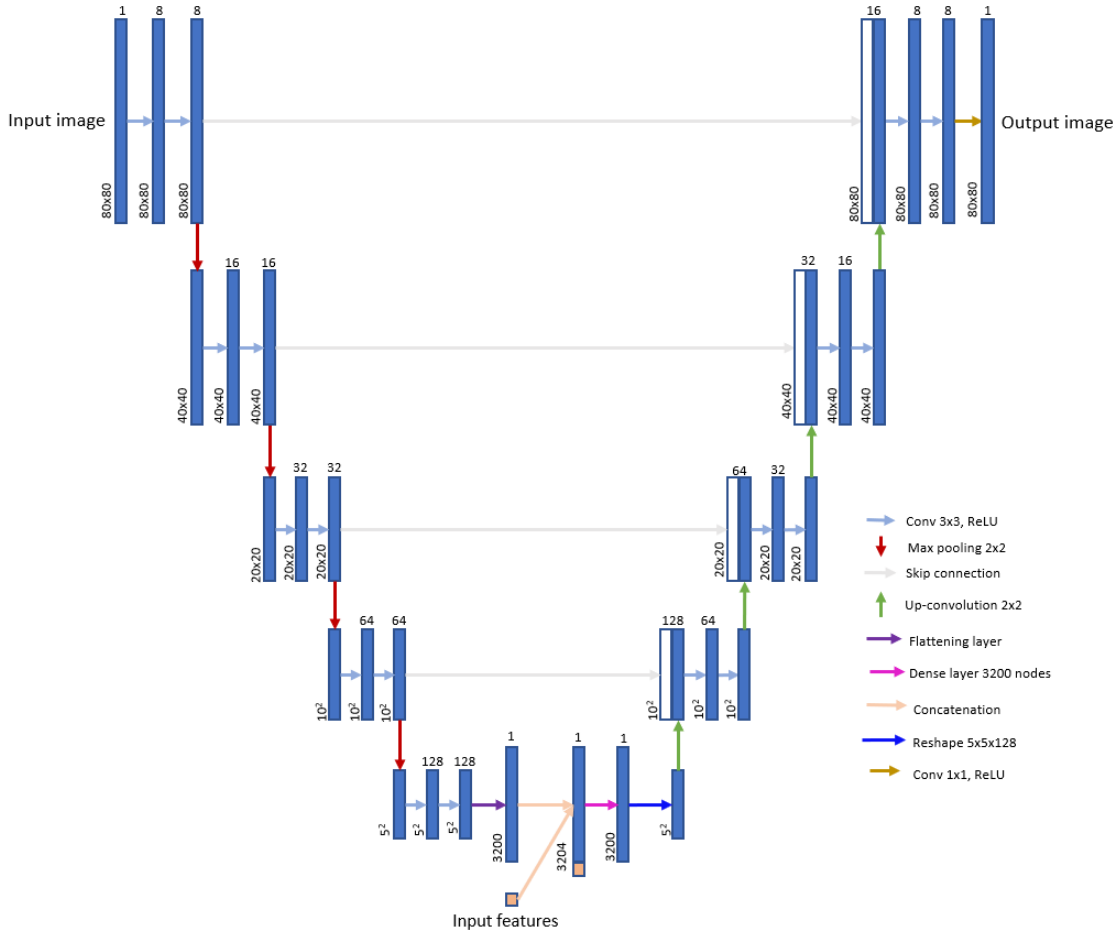


Figure 22: Detailed structure of the modified U-Net model. The goal is for the network to predict the next timestep in the sequence from an input image and the given features.

The model has depth five and uses 8 initial features in the input layer. It is trained for 100 epochs with a batch-size of 32. The loss function is mean squared error (MSE) and RMSProp is used as optimizer with an exponentially decaying learning rate which starts at 10^{-3} and decays exponentially with a factor of $0.9^{\frac{x}{10000}}$. Here x is a number of how many batches which have been used in training so far. The network has 10 796 073 trainable weights. To reduce the problem with overfitting a callback was used which saved the model with the lowest validation loss.

3.3.4 Convolutional LSTM

The convolutional LSTM network is the second model in this thesis. Due to its restricted input options the datasets had to be reshape into an array of shape [number of sequences, frames per sequence, width, height, number of channels]. The number of sequences was 128 in the training set

and 32 in the validation set. To simplify the training procedure only the first 14 frames for each sequence were being used. For drops from a height of two cm this included the full sequence while for drops from eight cm the last ten frames were disregarded. The width and height of the input images were 80 pixels as for the regular heatmaps with only one channel being used. The input data was at first normalized to have values between zero and one. The maximum and minimum values among all pixels in the training data was extracted and used in the scaling process. The minimum value was subtracted from all values in the training and validation set and the difference was then divided by the difference between the maximum and minimum value.

The model consisted of two consecutive convolutional LSTM layers each with 40 filters, a kernel of size 3×3, *same* padding and tanh as activation function. A 3D convolutional layer with one filter, a kernel size of 3×3×3, ReLU activation function and *same* padding followed the LSTM layers and provided an output of the appropriate size. During the training a MSE loss was being used together with Adam as optimizer. It had an initial learning rate of 10^{-3} and a decreasing learning rate was being used where it was decreased a factor of 0.1 if no improvement of the validation loss was seen for five epochs. The model was trained with a batch-size of five and for 20 epochs. Also in this training phase, a callback saved the model with the lowest validation error to reduce the risk of overfitting. The number of trainable weights was 175 641.

One thing to note is that no extra features are being used in this model in comparison to the other two models. The model only learns and requires the heatmap sequences to make predictions.

3.3.5 Graph convolutional LSTM

Lastly a graph convolutional LSTM was tested. Here the graph structure allows us to input features in more natural ways than a regular convolutional LSTM. The goal in this setup is to predict the next state of position for a node. We construct a graph with $80 \times 80 = 6400$ nodes, corresponding to pixels in the heatmaps used in both the U-net and convolutional LSTM models, where each node holds a time series of position of some input sequence length. These time series are created by concatenating heatmap values to for each drop sequence. The node features can be extended to hold time series corresponding to input features elapsed time, viscosity, fill level, and drop height. The series for two nodes, one situated in the top left corner of heatmaps and one situated in approximately the middle, is illustrated in figure 23 for the full training set.

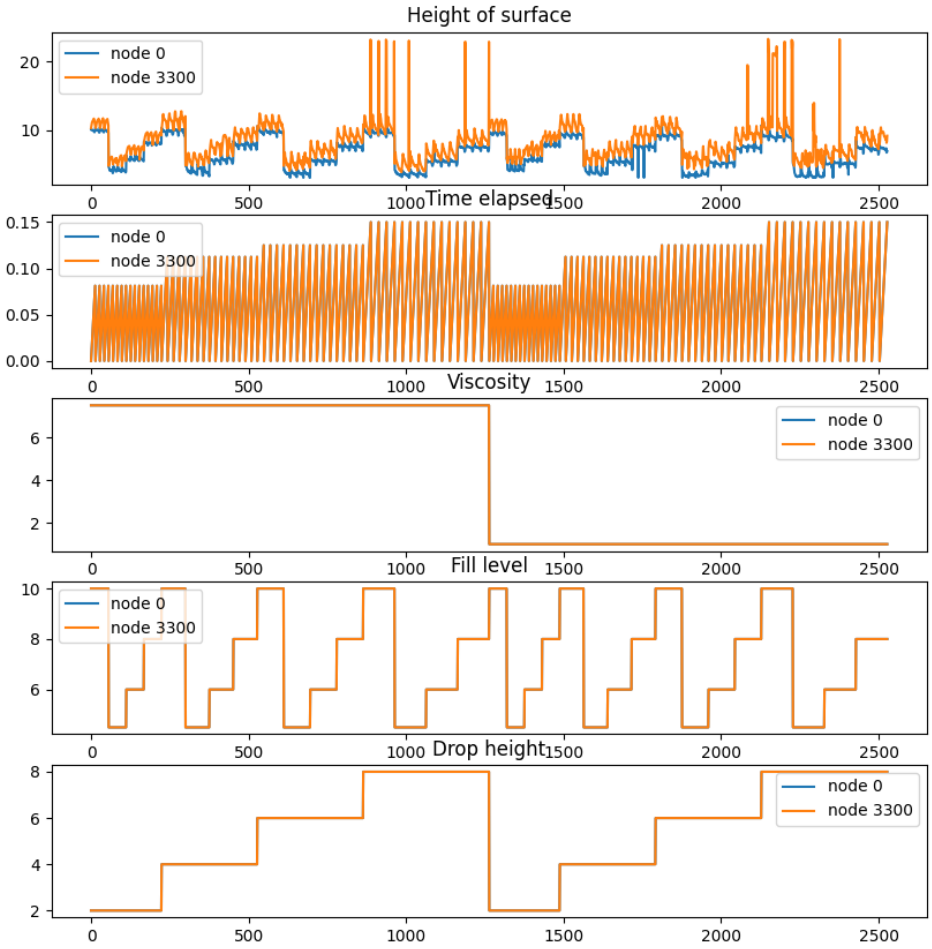


Figure 23: The node feature time series over the full training set.

Since the end goal for the model is to continuously predict future values based on its own past predictions, a quite small input sequence length of 3 was chosen to avoid far reaching dependence on starting input values. Additionally, since a single drop ranges 11-21 frames depending on drop height, the way the data is structured far reaching input lengths would frequently overlap between drops, meaning the model would train to predict future values using inputs not related to the current drop. This phenomenon is still present for input sequence lengths of 3, but somewhat mitigated. A potential fix would be restricting batches to specific drops. This was however not investigated further.

In the graph construction, nodes can be connected in any way that seems meaningful. First an 8-neighbor connectivity was applied since the largest impact on a single nodes' position is probably its adjacent nodes' position. Additional edges were then added between nodes with high correlation, above 0.99. In total the graph had 6400 nodes and 193,606 edges.

For a single data point in the input graph, each node holds a time series of shape (3,5), corresponding to input sequence length and channels illustrated in figure 23. A kernel with 500 trainable weights of shape (5,100) was used, meaning the output graph was upsampled to 100 channels in the output graph, using mean of incoming node features as aggregating function. A relu activation functions was lastly applied.

The output graph was reshaped to a 3D tensor to be compatible with an LSTM layer. An LSTM layer with 128 units was connected to and lastly a dense layer.

There are lots of design choices and hyperparameters that can be tuned; adding more GC layers, pooling, number of LSTM filters, learning rate, etc. With relatively long training times optimizing this network is cumbersome. Eventually a network with one GC layer with kernel of shape (5,100), learning rate of 10^{-3}, RMSprop optimizer, 128 LSTM units, and a forecast horizon of 3 was built. In total this model had 169,335 trainable weights.

3.4 Assumptions and limitations

When working with this thesis several assumptions were made due to different limiting aspects where time was the largest factor.

- The data gathering part only consisted of a small amount of different liquids and from a restricted set of heights. This restricted the amount of data used to train the models.
- Only the surface was of importance.
- The interesting region was between the first and second bounce.
- Only fall with no initial angle at impact was considered.
- The viscosity was assumed to be constant for the liquids instead of a dynamic feature for non-Newtonian liquids.

4 Results

In this section the results will be presented. In section 4.1 the observational results acquired during the droptests will be introduced. This includes observations and examples of specific phenomena observed during the data gathering part. In section 4.2 the results from the proposed models will be presented.

4.1 Observational results from the images

4.1.1 Yogurt

Originally yogurt was meant to be part of the same modeling steps as water and milk. During initial drop tests it was discovered that behavior of yogurt in this setting was wildly different than the lower viscous liquids. To capture interesting data the drop heights were extended, as shown in table 1. Because it was determined not to use the yogurt drop data in the model building, no segmentation and 3D reconstruction was implemented. Instead the results in this section is purely observational.

For drops lower than 8 cm, the highest possible heights before peaks hit the top of the bottle in milk and water drops, almost nothing visual happens to the surface of yogurt except for small peaks of splashes. Figure 24 illustrate the same frame of a drop from 5 cm with 8 cm fill level for yogurt, milk, and water.

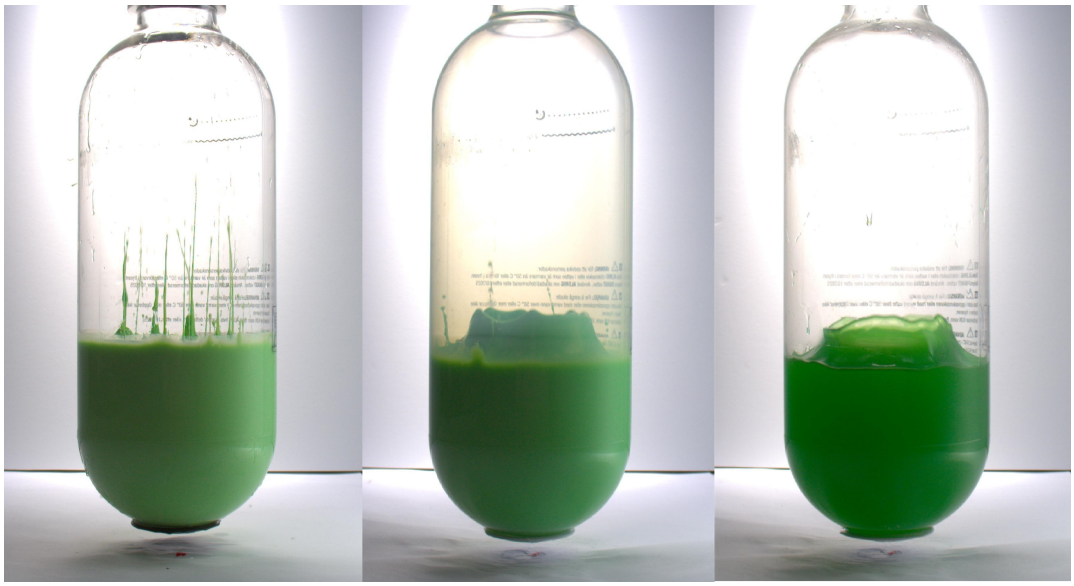
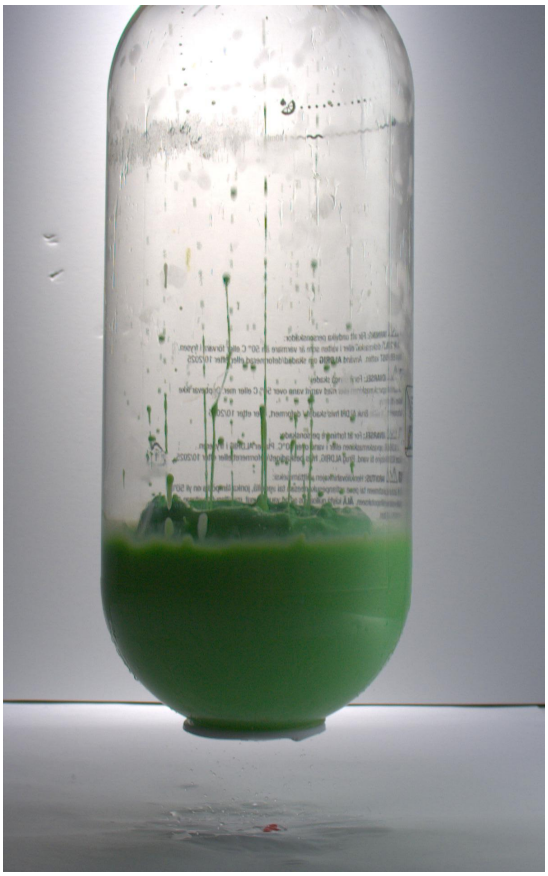


Figure 24: One frame, short after impact, visualized for yogurt (left), milk (middle), and water (right), for a drop from 5 cm and fill level 8 cm.

The behavior of sharp, seemingly random, peaks continue up until 22 cm drop height. At 22 cm and above the middle peak common for water and milk drops starts gradually appearing, illustrated in figure 25a and 25b.

(a) Yogurt dropped from 22 cm with fill level 6 cm.



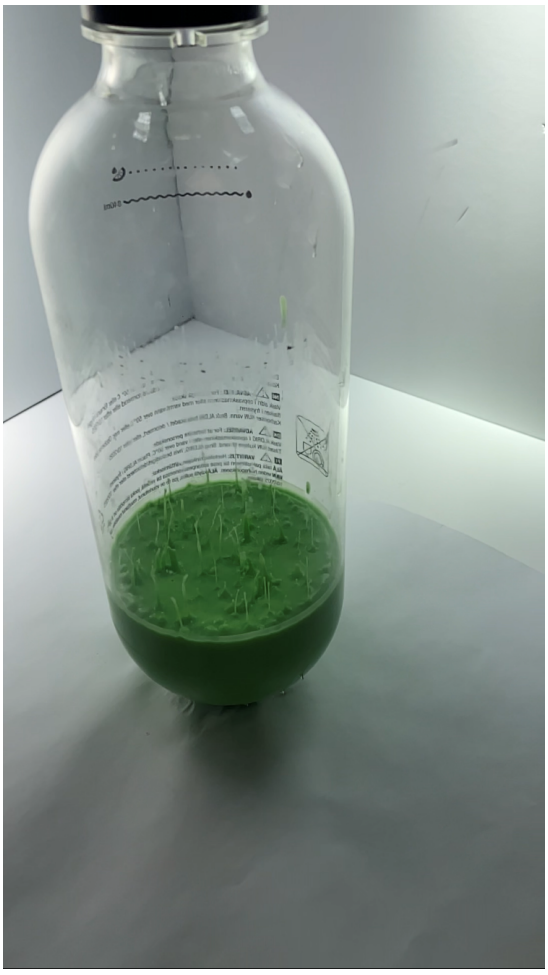
(b) Yogurt dropped from 30 cm with fill level 6 cm.

Figure 25

The small peaks are still prevalent and the peaks are still not as pronounced as in the case of water and milk drops.

Concerning the sharp peaks that appears, this was further investigated using an angle from above for a drop, illustrated in figures 26a and 26b.

(a) Yogurt drop filmed from above, before impact.



(b) Yogurt drop filmed from above, after impact.

Figure 26

Its seems that the small peaks appear, among other places, where air bubbles are trapped in the surface. We hypothesize that peaks forming where there is no clear bubble has trapped air somewhere close under the surface. Interestingly this behavior of small sharp peaks also appears when milk is shaken violently before a drop - introducing air bubbles.

4.1.2 Water and milk

The visual distinctions between water and milk are hard to find. They are generally very similar in terms of shape and drop responses, as illustrated in figure 27.

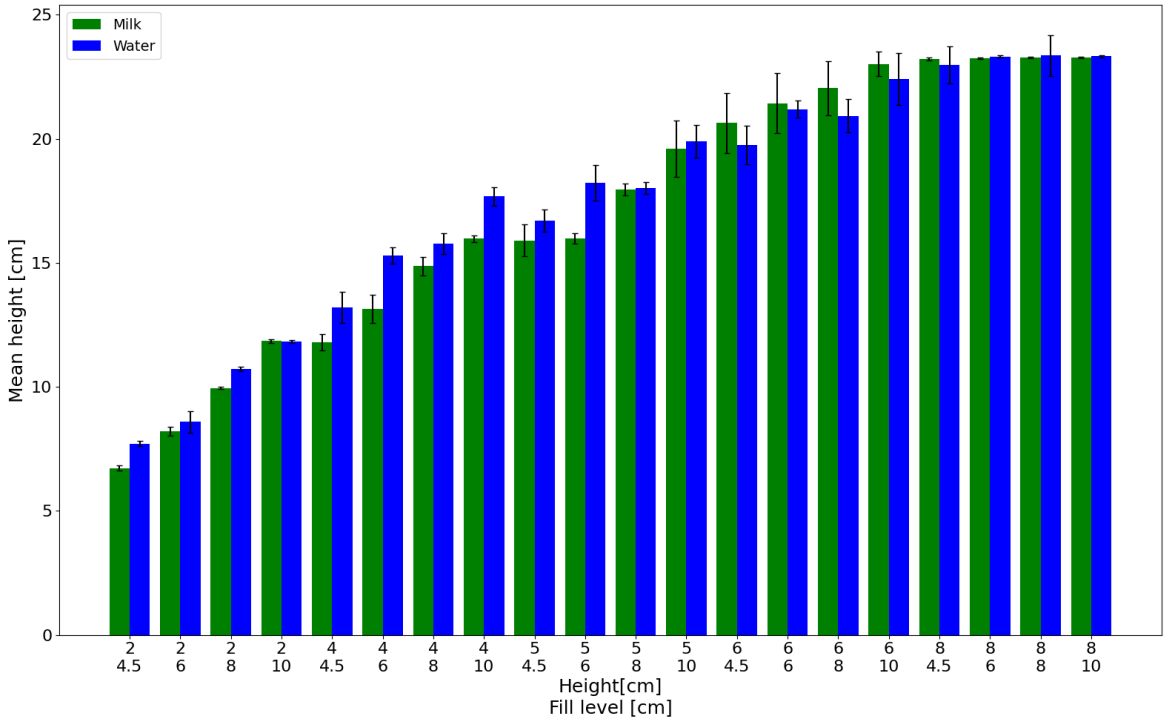


Figure 27: Average maximum height and standard deviation for the different setups of drop tests. The green bars are milk and the blue are water.

One notable difference in visual appearance is that milk tends to "clump" together more than water, figure 28 illustrates the phenomenon. Although the effect is small regarding peak heights there is a visual distinction.

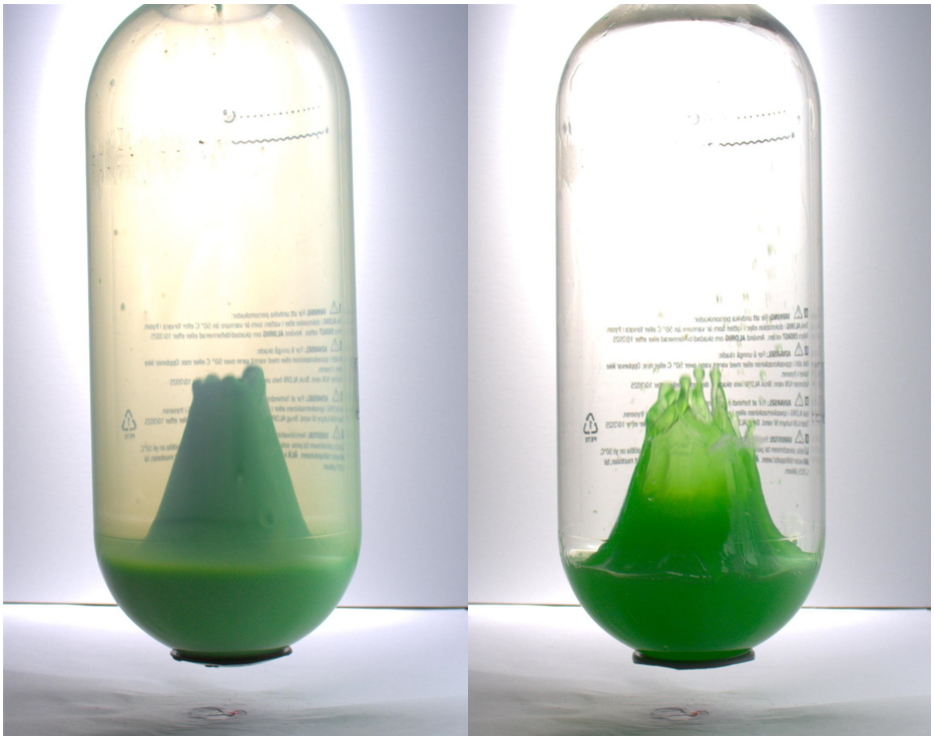


Figure 28: Milk (left) and water (right) in corresponding frames for drop height 8 cm with fill level 4.5 cm.

Another important observation, not captured by the segmentation and 3D reconstruction (that works by profiles), is that generally peaks form as a symmetric waves from the edges of the bottle combine in center. The right part of figure 28 hints at this behavior, and figure 29 show a series of images at impact.

Figure 29: Image series at impact for a drop.

Another important observation illustrated in figure 29 is that the surface of the liquid becomes curved during the fall, similar to how a spring would behave during a fall.

4.2 Results from the models

To evaluate the models, the validation set with the 32 sequences of drops as well as the 40 sequences from the test dataset will be used. For each of the models the difference between the maximum value for the predicted and the true drop for each frame will be calculated. This will be stored and evaluated as an average error per drop and per type of liquid. A MSE will also be calculated

in a similar way. Furthermore also a comparison between the predicted and true maximum height for each drop setting will be shown in a histogram.

4.2.1 U-Net

To predict a sequence with the U-Net model a uniform input heatmap consisting of the value of the fill level was needed together with the rest of the initial features. The model then makes a prediction of the next frame in the sequence which can then be used recursively to predict a full sequence. From the predicted data the maximum value for each of the predicted frames was compared to the maximum value in the corresponding frame in the validation data. The average difference per drop height could thus be calculated and the results are presented in table 2.

Table 2: The average error between the maximum value per frame in the predicted data and validation data sorted per drop height and liquid. All units are in cm.

Drop-height	Mean error water	Standard dev. water	Mean error milk	Standard dev. milk
2	0.3200	0.3114	0.2414	0.1347
4	0.6759	0.6561	0.4896	0.2832
6	1.6964	0.9996	1.4447	0.7823
8	1.0631	0.8046	1.0299	0.8993

The same comparison can be performed but with MSE on all nodes as a measure. This result is shown in table 3.

Table 3: The average MSE per frame between the predicted data and validation data sorted per drop height and liquid.

Drop-height	Avg. MSE water	Standard dev. water	Avg. MSE milk	Standard dev. milk
2	0.4861	0.6062	0.2586	0.2802
4	0.8273	0.9261	0.8096	1.0919
6	2.1591	2.5700	1.3222	1.3101
8	2.0001	2.4819	2.8699	5.8506

To further visualize the results, a plot with the maximum height for each drop in the validation set and the predicted maximum height will be shown for water and milk.

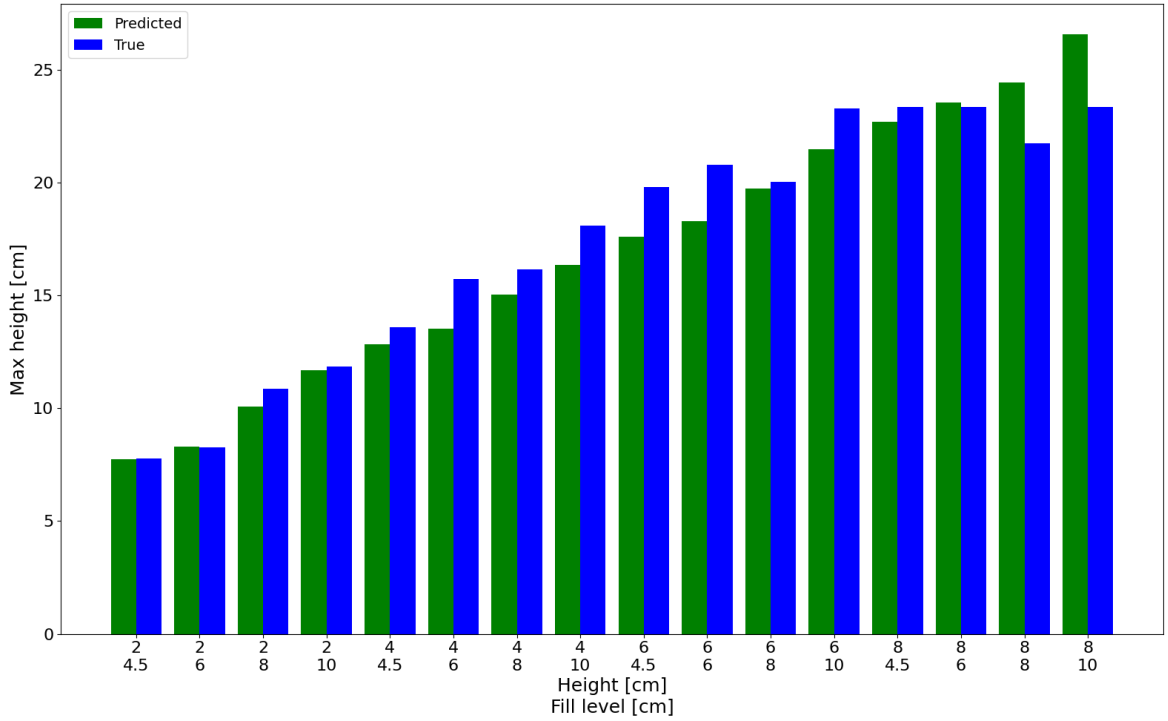


Figure 30: Predicted maximum height, in green, and actual maximum height, in blue, of the peak for water.

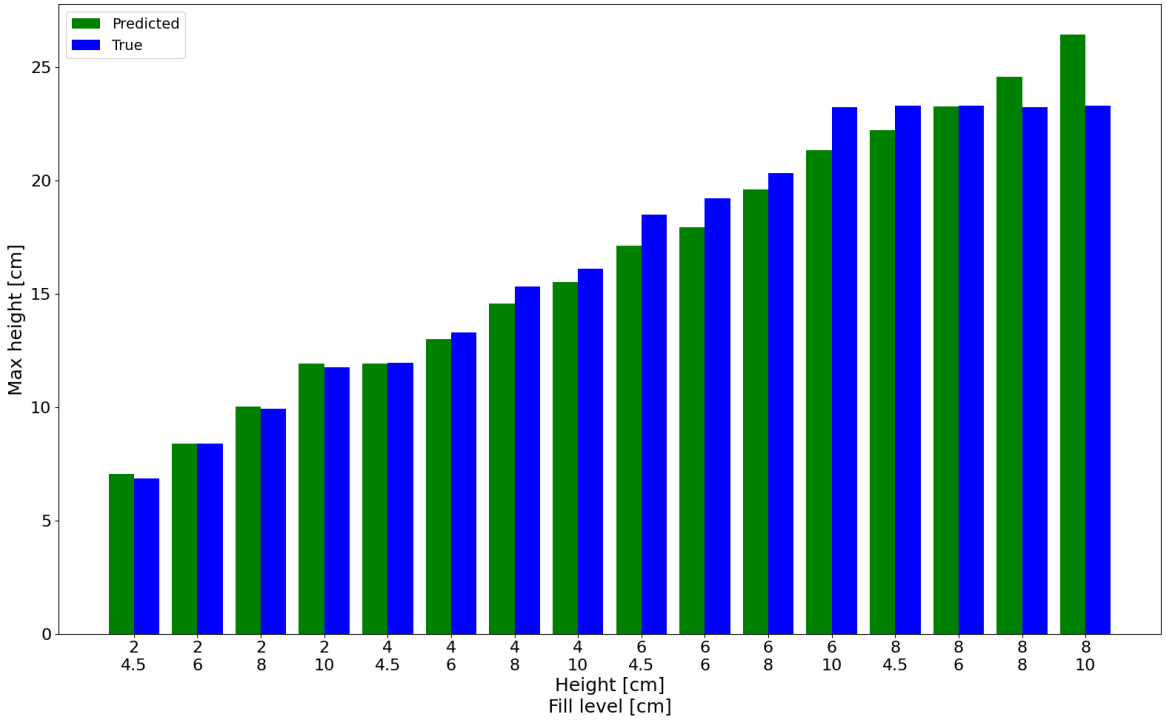


Figure 31: Predicted maximum height, in green, and actual maximum height, in blue, of the peak for milk.

The same tests is made for the test data on the unseen height of five cm. The results from that dataset is presented below.

Table 4: The average peak error and MSE for each frame between the predicted data and test data where the drop height was five cm.

Error measure	Mean error water	Standard dev. water	Mean error milk	Standard dev. milk
Peak error	1.1032	0.8778	1.3390	0.9838
MSE	1.0205	0.9221	0.9470	0.7044

In figure 32 the maximum height for the predictions and drops in the test set is compared. Since each setup in the test set consisted of five drops the mean is calculated and used.

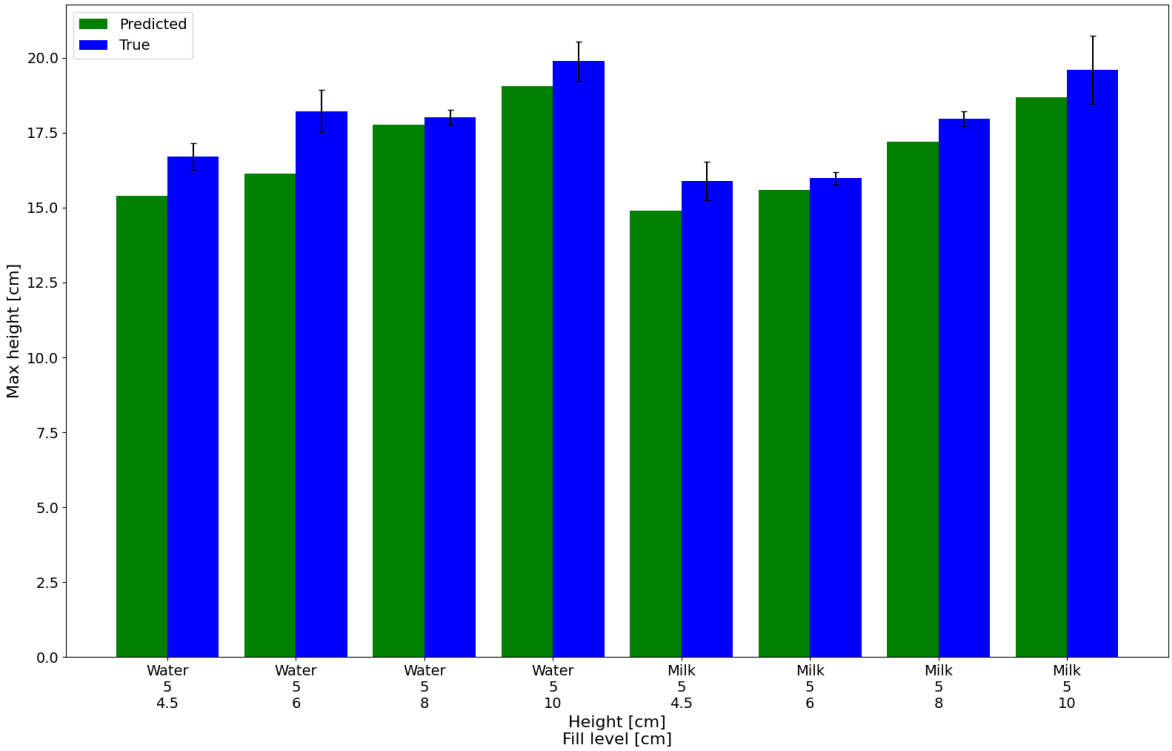
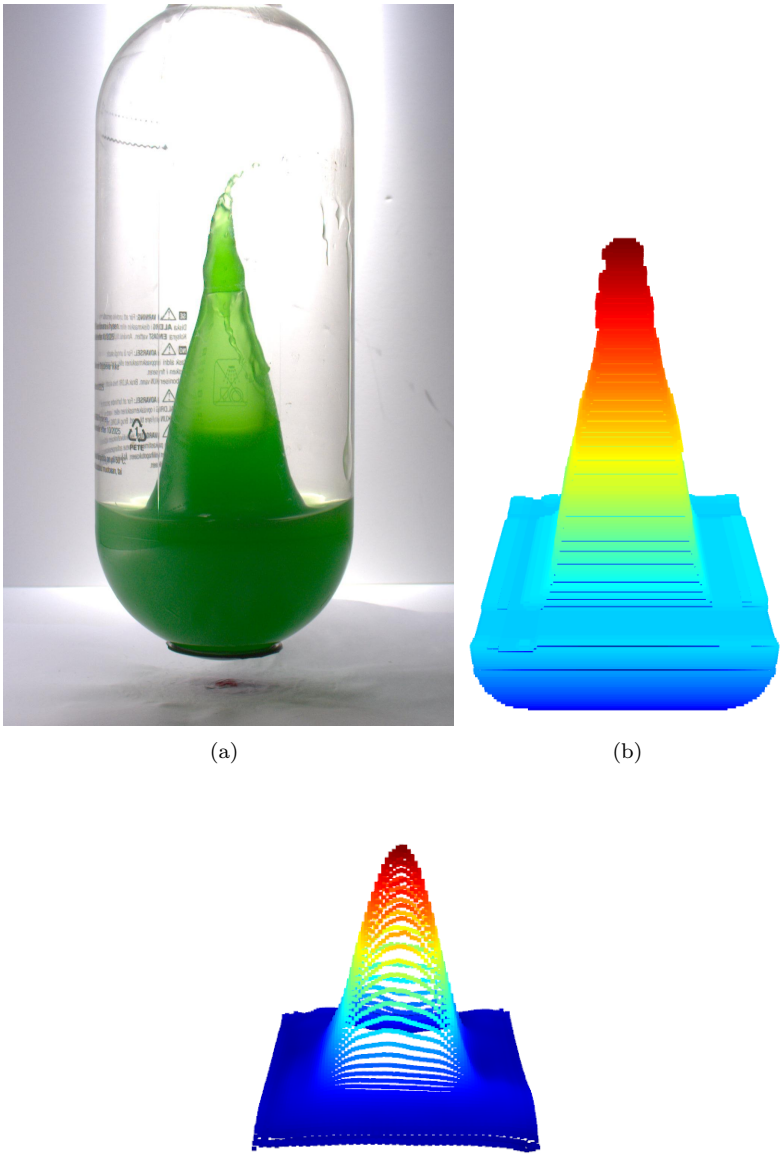


Figure 32: Predicted maximum height, in green, and actual maximum height, in blue, of the peak for milk and water.

As an example of the visual results, images from an actual drop test, the true reconstructed pointcloud and the predicted pointcloud from the model is shown in figure 33. The example frame is from a drop of water at height five and with six in fill level. Note that it is hard to quantitatively compare these images since the scales are different, instead they are being shown to compare the overall spatial appearance.



(a) (b)

(c)

Figure 33: (a) Original image (b) True 3D-reconstruction (c) Predicted reconstruction

4.2.2 Convolutional LSTM

The convolutional LSTM model was first evaluated on the validation data. Since the convolutional LSTM had a different model structure, the first five true heatmaps will be used in the initialization step and the remaining nine will be predicted by the model. In table 5 the maximum value for each of the predicted frames was compared to the maximum value in the corresponding frame in the validation set. The data is sorted with respect to drop height.

Table 5: The average error between the maximum value per frame in the predicted data and validation data sorted per drop height and liquid. All units are in cm.

Drop-height	Mean error water	Standard dev. water	Mean error milk	Standard dev. milk
2	0.9516	1.0117	1.2023	0.8149
4	2.0865	1.2343	1.8748	1.0424
6	2.1583	1.6813	1.6333	1.5064
8	3.9303	3.2775	3.4534	2.8039

The average MSE for each of the frames can also be calculated and the results is presented in table 6 below.

Table 6: The average MSE per frame between the predicted data and validation data sorted per drop height and liquid.

Drop-height	Avg. MSE water	Standard dev. water	Avg. MSE milk	Standard dev. milk
2	2.3610	3.3788	2.4476	3.3949
4	3.9641	4.6193	3.9569	5.3245
6	4.4805	4.4602	4.9194	4.5730
8	5.3311	4.0173	4.8090	4.6916

A plot with the true and predicted maximum height for each drop setup is created for both water and milk. It can be seen in the figures below.

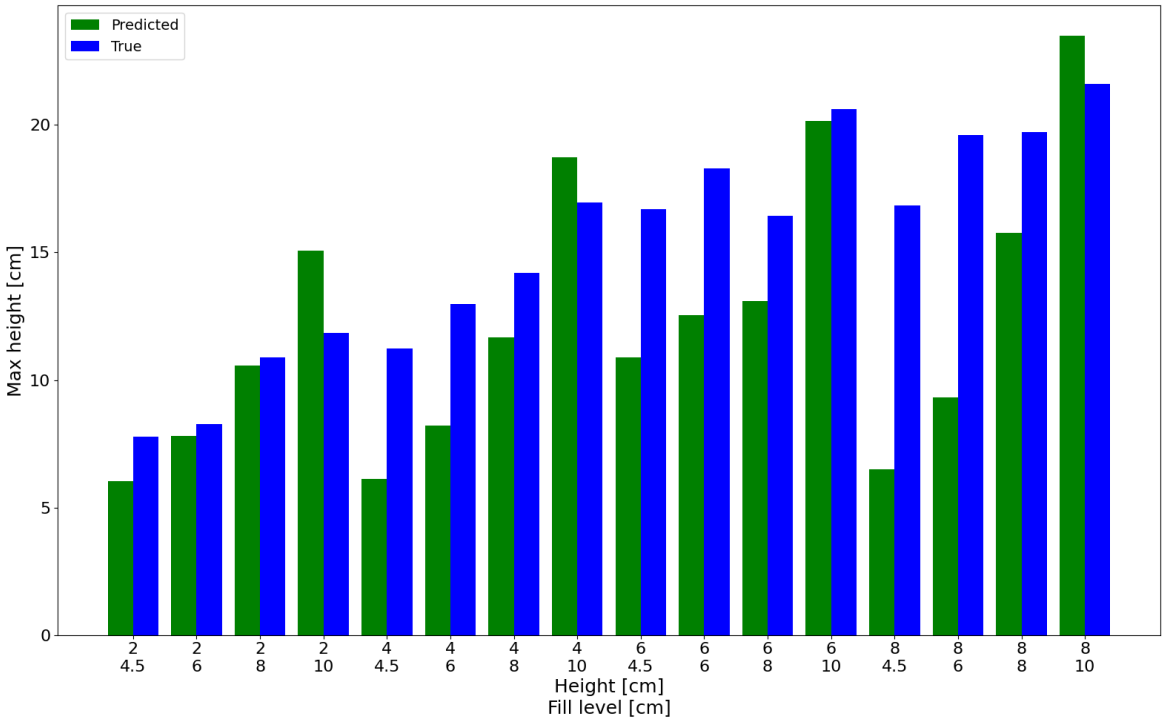


Figure 34: Predicted maximum height, in green, and actual maximum height, in blue, of the peak for water.

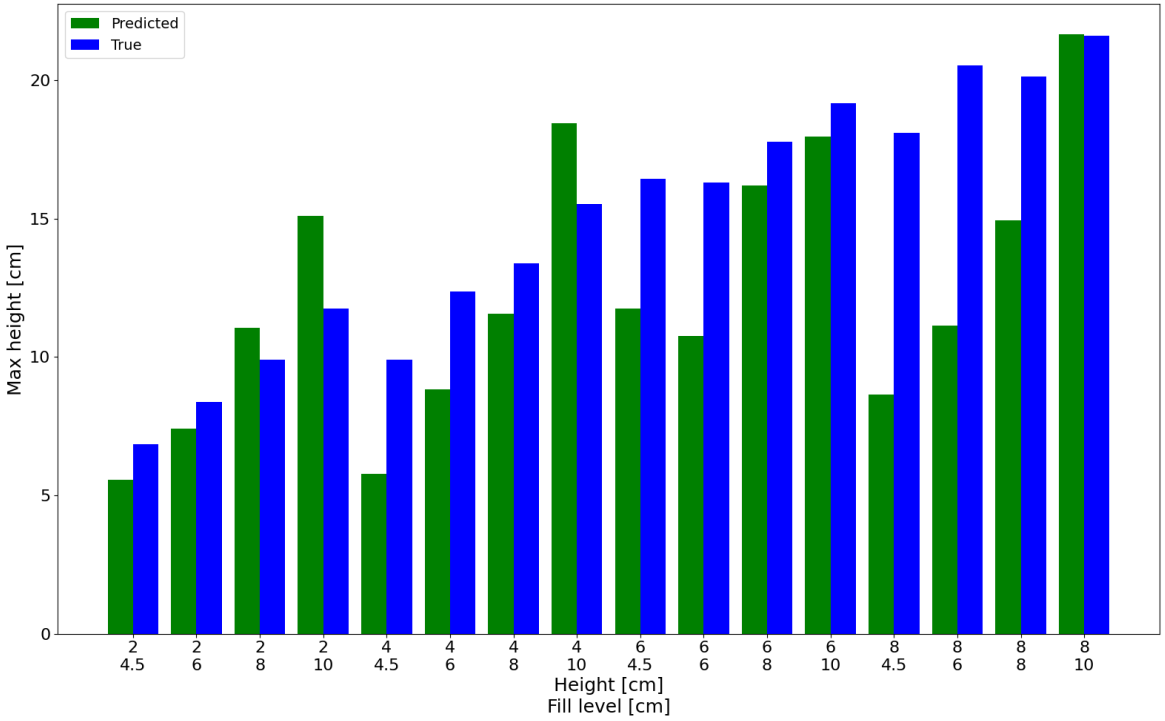


Figure 35: Predicted maximum height, in green, and actual maximum height, in blue, of the peak for milk.

The same tests is conducted for the test data on the unseen height of five cm. The results from that dataset is presented below.

Table 7: The average peak error and MSE for each frame between the predicted data and test data where the drop height was five cm.

Error measure	Mean error water	Standard dev. water	Mean error milk	Standard dev. milk
Peak error	2.3491	1.6711	1.7577	1.4062
MSE	4.3915	4.7062	4.7964	6.1160

In figure 36 the predicted maximum peak and the true maximum peak for the test set is compared per drop setting. The mean of the five drops in the test set are being used.

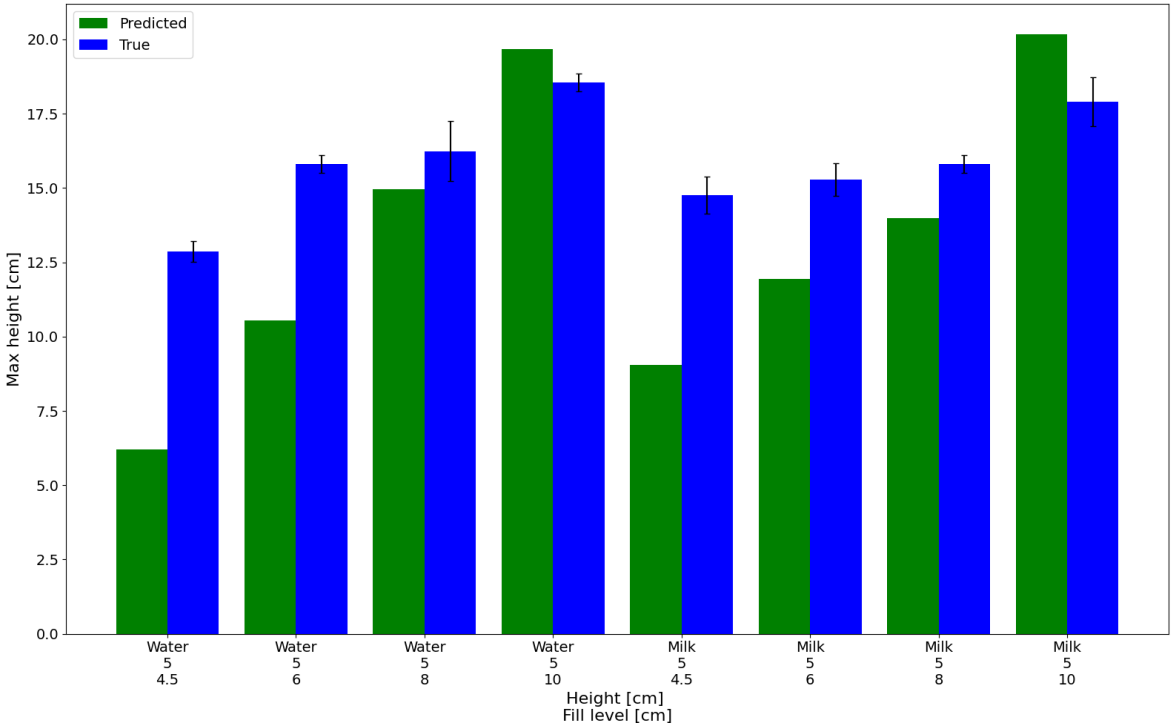
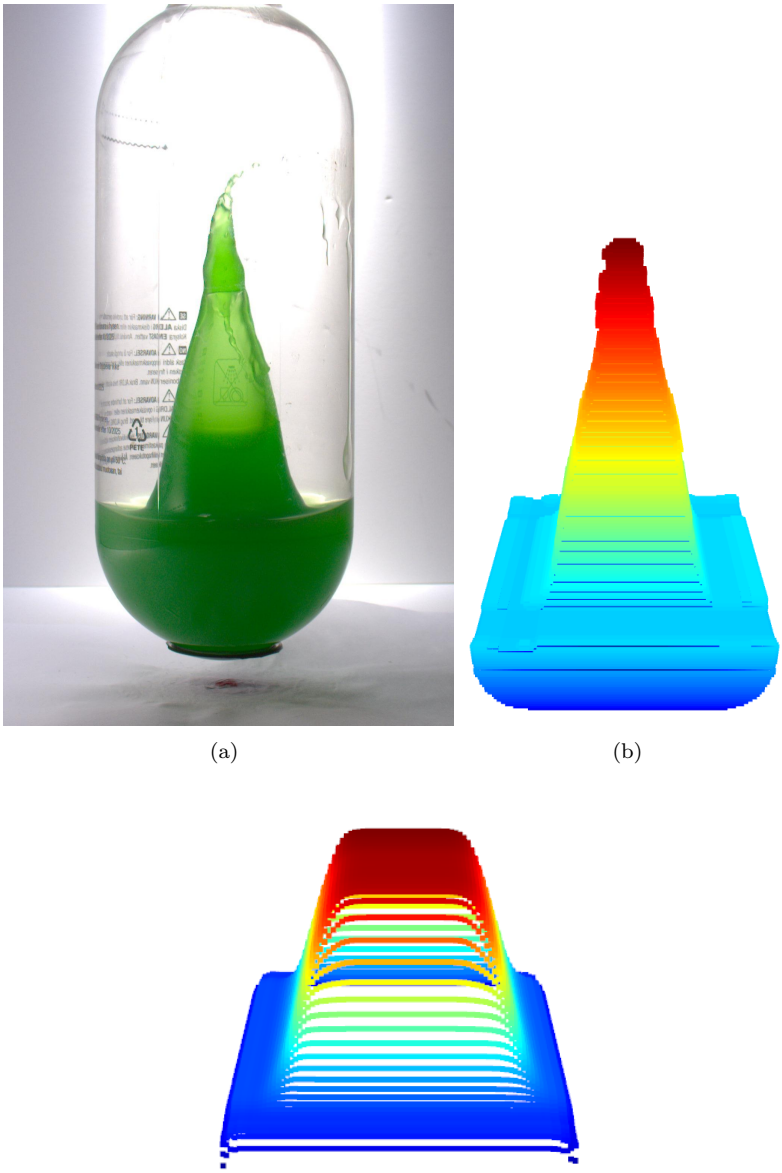


Figure 36: Predicted maximum height, in green, and actual maximum height, in blue, of the peak for milk and water.

To again visualize the results, the same frame from the drop of five cm height with a fill level of six cm is shown. This time with the predicted frame from the convolutional LSTM model.



(a) (b)

(c)

Figure 37: (a) Original image (b) True 3D-reconstruction (c) Predicted reconstruction

4.2.3 Graph convolutional LSTM

Predictions for the graph convolutional LSTM was initialized with 5 channels of length 3 containing the fill level as position, time elapsed as zero, viscosity, fill level, and drop height - for each node. After each one step prediction the position and time elapsed part of the input tensors are updated with the predicted positions and $t_{i+1} = \frac{t_i+1}{160}$, which is then passed to the model for the next step prediction until the elapsed time is greater than calculated time until second bounce.

Errors in the validation set are presented in tables 8 and 9.

Table 8: The average error for the biggest difference between the peak in the predicted data and validation data. All units are in cm.

Drop-height	Mean error water	Standard dev. water	Mean error milk	Standard dev. milk
2	0.6179	0.4263	0.6117	0.4446
4	2.0610	1.6643	1.988	1.5999
6	4.3556	3.4662	4.4070	3.4487
8	5.4133	4.034	6.1510	4.7467

Table 9: The average MSE for each frame between the predicted data and validation data. The data is sorted based on the drop height.

Drop-height	Avg. MSE water	Standard dev. water	Avg. MSE milk	Standard dev. milk
2	0.3841	0.4175	0.777	1.2310
4	0.8760	0.6908	0.9473	0.7122
6	3.4748	4.4130	2.7752	3.070
8	3.8113	4.5580	4.7476	8.0685

Statistics for the test set height of five cm is presented in table 10.

Table 10: The average peak error and MSE for each frame between the predicted data and test data where the drop height was 5 cm.

Error measure	Mean error water	Standard dev. water	Mean error milk	Standard dev. milk
Peak error	2.9853	2.4651	3.5045	2.7191
MSE	1.3726	1.3615	1.6928	1.4658

The performance can be further analyzed using maximum heights in figures 38 and 39.

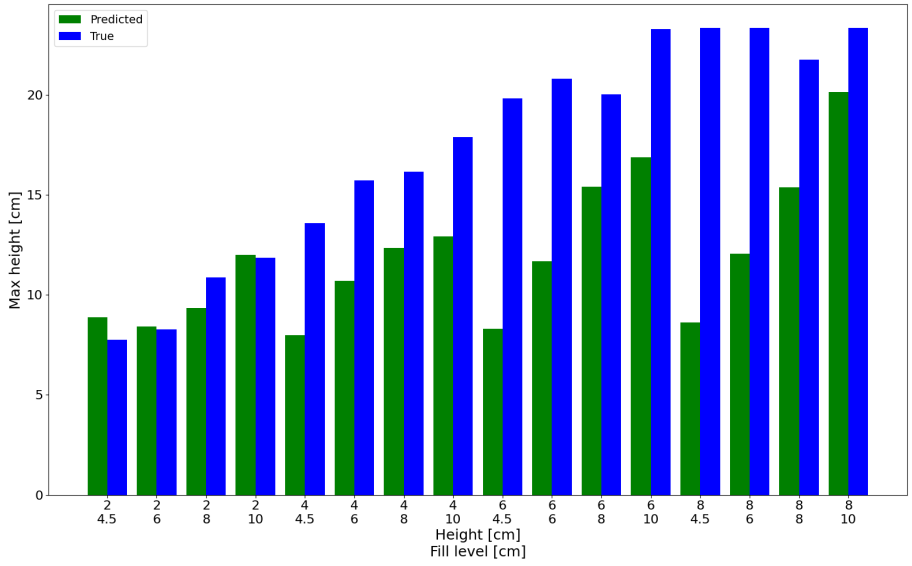


Figure 38: Predicted maximum height, in green, and actual maximum height, in blue, of the peak for water.

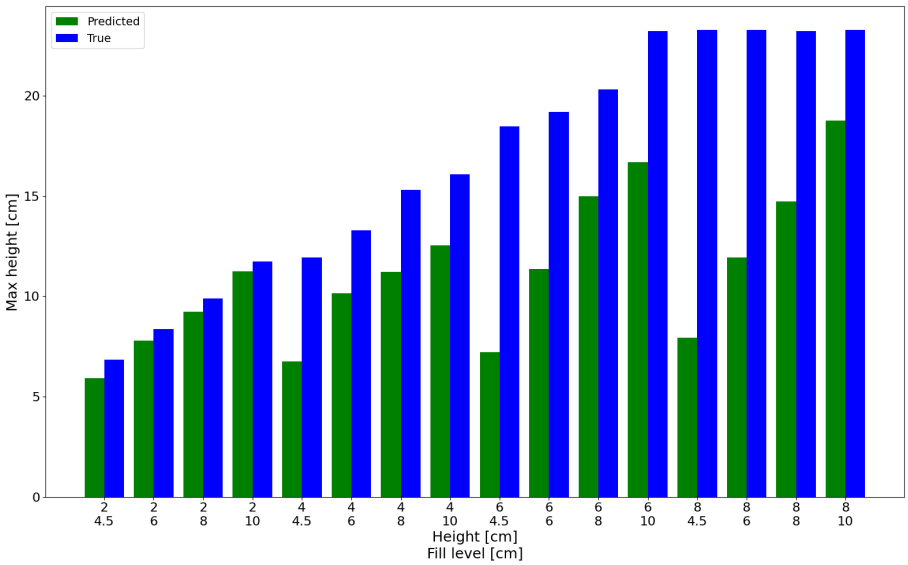
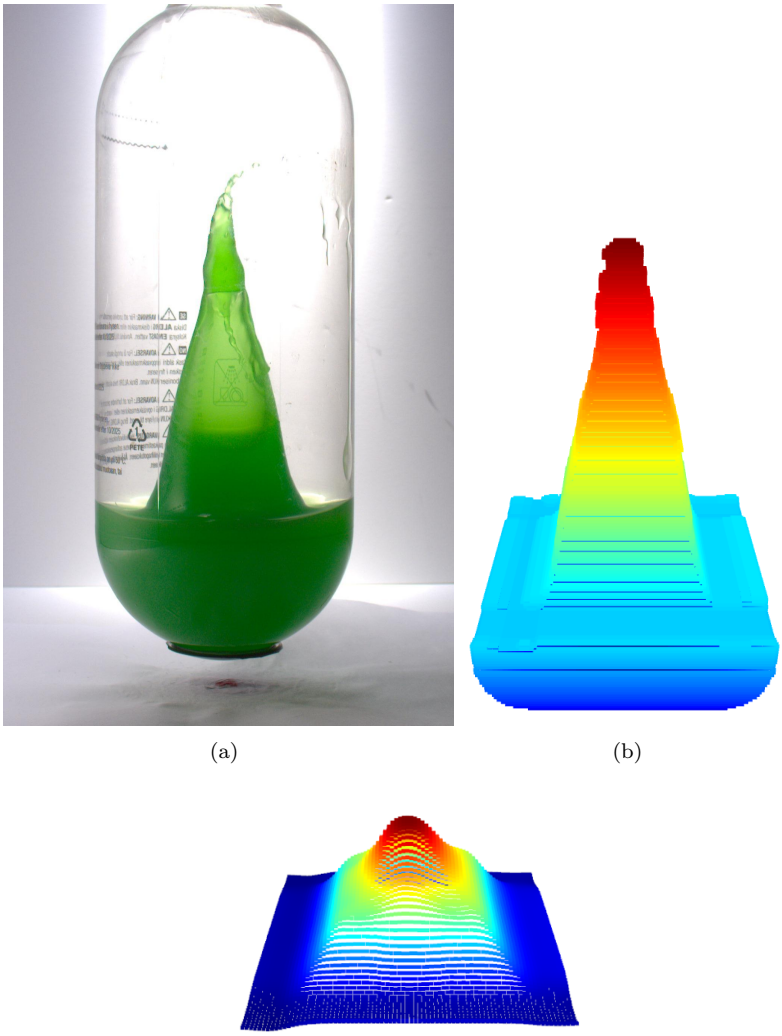


Figure 39: Predicted maximum height, in green, and actual maximum height, in blue, of the peak for milk.

The same drop visualized in the U-net and convLSTM models is illustrated in figure 40.



(a) (b)

(c)

Figure 40: (a) Original image (b) True 3D-reconstruction (c) Predicted reconstruction

5 Discussion

5.1 Analyzing observational results

One purpose of the thesis was to create a stable setup for drop tests of liquids to produce robust and repeatable data from physical tests. The main reason was to contribute with valuable insight that the virtual modeling group at Tetra Pak® can use when creating physics based models for simulation. Fluid dynamics can be notoriously unintuitive, thus real world experiments are an essential part of model verification. To this end the project was successful in reliable data gathering. There is some noise present in the reconstructed surfaces, but generally the data of drops in the same setup produce essentially the same results.

In parallel with the work done in this thesis new simulation models were created at Tetra Pak® using the gathered data to verify new model iterations, and introduce new aspects not accounted for in earlier models. One key aspect was the need to simulate the liquid before impact. The curved structure of low viscous liquids introduced during the fall makes a large difference in behavior after impact. By adding air bubbles in the isosurface, the models could simulate resemblance to the small splash peaks seen in yoghurt and shaken milk. Another addition was adding atmosphere in the simulated bottles, which did away with problem of simulated liquid separating from the bottom of the bottle. Figure 41 illustrate the same drop as in figure 1 with snapshots from two newer models.

Figure 41: Snapshots of newer models developed at Tetra Pak® during this thesis.

Another unintuitive result was how different yogurt behaved compared to water and milk. We expected it to behave somewhat similarly as the other liquids, with a smaller middle peak. Instead the small sharp peaks appeared at the same places as trapped air bubbles. A few tests where we tried to minimize the amount of air bubbles on the surface were conducted. For example the yogurt was poured more carefully into the bottle and it was also stirred with a stick to eliminate the bubbles. This did reduce the amount of peaks present in the drop. The behavior was still in general quite different compared to water and milk. In the lower viscous liquids the middle peaks were in general formed as kinetic energy is transferred symmetrically from the vertical sides of the

bottle, meeting in middle and forming the peak. We hypothesize this behavior occurs because of the curvature of the surface when the bottle impacts. This curvature is less pronounced in drops of yogurt, probably because of the higher viscosity. Subsequently the force from the impact is transferred more as a shock from the bottom, causing drop behavior quite different than the less viscous liquids. This general large difference in behavior of yoghurt further asserts the importance of conducting drop tests, both virtual and physical, with appropriate liquid product on new designs - not simply water.

5.2 Analyzing model results

The U-Net model was the first model which was implemented and tested. When the model was tuned and optimized for the validation data the results were promising not only on the validation set but also on unseen data as well. By inspecting the results from table 2 and 3 it can be seen that the average errors are relatively low when being compared to the results for the other two models. A small MSE also indicates that not only the top peak but also the general shape of the prediction is similar. It was also good at predicting the maximum height for the drops in the validation set which is confirmed by figure 30 and 31 where the predicted and true peak are close in height. However for the drops from six and eight cm height, the model error and the standard deviation increases. This effect is partially due to the liquid hitting the top of the bottle. Therefore the maximum height for the true data is around 25 cm but this restriction is not present for any of the models. When evaluating the U-Net model on the test set the results were promising. It was possible to predict the maximum height with good accuracy and obtained errors regarding peak and MSE of sizes between the drop heights four and six cm on the validation set. The overall predicted shape is also rather similar to the true shape of the examplified drop in figure 33.

The second model, convolutional LSTM, is a model which was different from the two others. It did not use any features at all but instead only required a few initial frames of the true sequence to model the outcome. This design choice was not beneficial when inspecting the results. Compared to the U-Net it performed worse when comparing the maximum peak error and the MSE. However, compared to the graph convolutional LSTM it performed better regarding the peak error but worse regarding MSE. This means that the overall shape is worse than for the graph convolutional LSTM model but it is better at predicting the maximum height. This can also be seen when comparing figures 34 and 35 to 38 and 39. Besides the results this model has the disadvantage of requiring a few true frames to start the prediction. From a generative model perspective this is not optimal. The training was also performed in a different way compared to the two other models with only the first 14 frames for each drop being used which did affect the result.

The graph convolutional network implemented did not perform as well as we perhaps hoped. While it is able to capture the general shape of the isosurfaces compared to convolutional LSTM, noticeable when comparing MSEs, the model is unable to predict the resulting height from higher drops. We conjecture that this type of model could be suitable in this type of application, they are however quite difficult to train. There are lots of design choices and hyperparameters available for tweaking, where in this specific setting it was hard to analyze what different changes resulted in. Combined with the relatively long training times and time constraints of this project led to a final graph convolutional model that is a bit underdeveloped.

Additionally, all the models are trained to predict the next position based on true input data. This is a bit different than what the final model is meant to do - multistep predictions based on

its own past predictions. The useful parts learned in training does not necessarily translate to multistep prediction.

6 Conclusion and future work

In conclusion, a data driven approach to problems traditionally handled with more physics and engineering based methods has its value. While observation based models will not capture physical ground truth - and can't be expected to, it can provide approximate solutions able to run in real time. The data gathered in this thesis provided knowledge about fluid behavior in drop tests, used to improve existing CFD models. The U-net model resulted in an approximate model that could adequately interpolate liquid characteristics in set feature space.

To further build upon this work it could be possible to gather data for more liquids of different characteristics. It could also be possible to drop the package with an incoming angle to investigate how it would behave. To fully capture more details of the drops a camera with higher frame-rate could be used, even though in this thesis it was neat to use the cameras directly via USB. Model-wise it could be of interest to try more pure generative models such as generative adversarial networks or variational autoencoders. If more types of liquids are introduced, there would probably be a need to extend the viscosity feature using dependence on shear rate in the liquid. As it stands viscosity was set constant, and a dynamic viscosity feature would connect more closely to the underlying physics present in liquids during highly dynamic conditions such as drop tests.

7 References

References

[1] G. Bradski and A. Kaehler. *Learning OpenCV*. O'Reilly, Sebastopol, 2008.

[2] G. Calzolari and W. Liu. Deep learning to replace, improve, or aid CFD analysis in built environment applications: A review. In *Building and Environment*, 206, pp.1-12, 2021.

[3] D. Chen et al. FlowGAN: A Conditional Generative Adversarial Network for Flow Prediction in Various Conditions. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pp.315-322, 2020.

[4] T. Cormen et al. *Introduction to Algorithms-Third Edition*. MIT Press, Cambridge, 2009. pp. 714-741.

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

[6] M. Grundelius. *Methods for Control of Liquid Slosh*. Department of Automatic Control, Lund Institute of Technology (LTH). pp.29, 2001.

[7] S. Hochreiter and J. Schmidhuber. Long Short-term Memory. In *Neural Computation*, 9(8), pp.1735-1780, 1997.

[8] R. Ibrahim, V. Pilipchuk and T. Ikeda. Recent Advances in Liquid Sloshing Dynamics. In *Applied Mechanics Reviews*, 54(2), pp.133-199, 2001.

[9] Jmtrivial - Own work, GPL, https://commons.wikimedia.org/w/index.php?curid=1282165

[10] P. Kennedy and R. Zheng. *Flow Analysis of Injection Molds 2nd edition*. Carl Hanser Verlag, Munich, 2013. pp. 22-24.

[11] L. Ladický et al. Data-driven fluid simulations using regression forests. In *ACM Transactions on Graphics*, 34(6), pp.1-9, 2015.

[12] W. Lorensen and C. Harvey. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *ACM SIGGRAPH Computer Graphics*, 21(4), pp.163-169, 1987.

[13] A. Mohan et al. Compressed Convolutional LSTM: An Efficient Deep Learning Framework to Model High Fidelity 3D Turbulence. *Arxiv*, DOI: https://doi.org/10.48550/arXiv.1903.00033, 2019. pp.4

[14] M. Ohlsson and P. Edén. *Introduction to Artificial Neural Networks and Deep Learning (FYTN14/EXTQ40/NTF005F)*. 2021.

[15] OpenCV documentation. Accessed: 2022-05-26. `https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html`

[16] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications Inc., Mineola, 1998. p 119.

[17] O. Ronneberger, P. Fischer and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Lecture Notes in Computer Science*, pp.234-241, 2015.

[18] X. Shi et al. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. *Arxiv*, DOI: https://doi.org/10.48550/arXiv.1506.04214, 2015.

[19] J. Solem. *Programming Computer Vision with Python*. O'Reilly, Sebastopol, 2012.

[20] R. Szeliski. *Computer Vision: Algorithms and Applications 2nd Edition*. Springer, Cham, 2022.

[21] N. Thuerey et al. Deep Learning Methods for Reynolds-Averaged Navier-Stokes Simulations of Airfoil Flows. In *AIAA Journal*, 58(1), pp.25-36, 2020.

[22] K. Um, X. Hu and N. Thuerey. Liquid Splash Modeling with Neural Networks. In *Computer Graphics Forum*, 37(8), pp.171-182, 2018.

[23] DS. Viswanath et al. *Viscosity of Liquids - Theory, Estimation, Experiment, and Data.*. Springer, Dordrecht, 2007. pp. 1-7

[24] Z. Wu et al. A Comprehensive Survey on Graph Neural Networks. In *IEEE Transactions on Neural Networks and Learning Systems*, 32(1), pp 7-9, 2021.

[25] B. Yu, H. Yin and Z. Zhu. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, 2018.