A Semi-Automated Segmentation of Electron Microscopy Images for 3D Reconstruction of the Nuclear Envelope

Stephen Wood

Supervised by: Prof. Daniel Rueckert, Department of Computing, Imperial College London
Dr. Luis Pizarro, Department of Computing, Imperial College London
Dr. Lucy Collinson, Cancer Research UK, London Research Institute

MEng Computing Individual Project
Department of Computing, Imperial College London

Abstract

The nuclear envelope is an important structure in cells as they undergo mitosis, breaking down to allow the cell to duplicate and split before reforming around each daughter nucleus. This importance is also underlined in cancerous cells where certain components of the nuclear envelope take central roles in cell functions that affect tumour development and progression. Recently there has been a marked interest in studying the nuclear envelope and the role it takes in mitosis, but the scientific community is yet to agree upon an exact model for the structures and interactions that govern its reformation.

Visualising and investigating the nuclear envelope requires its accurate segmentation from a set of serial images of the cell obtained through electron microscopy. As a manual process this requires a great amount of expert knowledge and training, whilst also being very tedious and time consuming, taking on average a couple of months to segment all the structures in a full data set. Once the segmentation is achieved a 3D reconstruction of the nuclear envelope can be created and interpreted to study its characteristics and relationships to other structures in the cell.

Attempts to use state of the art segmentation software have failed due to its generality, and no other research into nuclear envelope segmentation is known to have been carried out. This report presents a semi-automated algorithm for segmenting the nuclear envelope of nuclei at a late stage of mitosis when it is almost completely reformed around the daughter nuclei. A pipeline of image processing techniques finds the nuclear envelope starting from the identification of salient points inside the nucleus, and a number of manual corrections are necessary to allow for the complexities of nucleus shape and the presence of noise in the microscopy process.

The total time for the segmentation of the nuclear envelope is reduced from around three weeks to a single day, taking into account other required manual processes that are difficult to automate. Consequently, this has a large impact on the throughput of data from the electron microscope to the biologists requiring the data to lead their research. The work represents a huge step towards an algorithm capable of segmenting the nuclear envelope at all stages of mitosis, where no reliance on the shape of the nuclear envelope can be made, and builds a strong base for this further research.

Acknowledgements

Many thanks to Dr. Luis Pizarro, whose office became a place for lengthy discussions and puzzled thoughts; I am extremely grateful for the amount of time you spared and the support you offered throughout the project. My thanks also to Prof. Daniel Rueckert for his experience, ideas and guidance overall.

I am privileged to have been able to cross over from Computer Science and gain a taste of research in another field, particularly one that has such an impact on so many people. To Dr. Lucy Collinson, Prof. Banafshe Larijani and Dr. Christopher Peddie, I'd like to thank you for your ideas, enthusiasm and the time you have spent explaining concepts that are second nature to you such that I could understand them. Having a firm grasp on the background was essential to the success of this project, and I hope this outcome has a significant impact upon your research in the future.

My final thanks go to anyone who has helped me during my time at Imperial, the lecturers I have seen out of normal hours and the friends and family who have put up with me during exam periods; it has been well worth it!

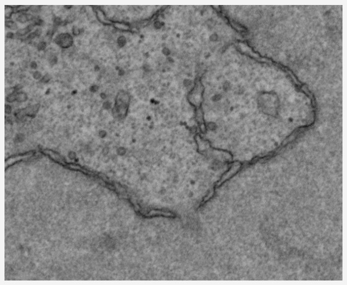
Contents

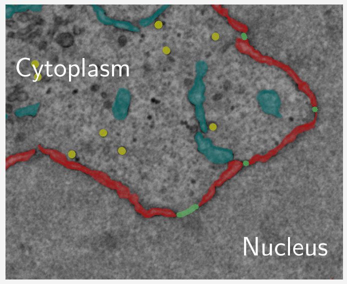
1 Introduction

The nuclear envelope is a sub-cellular structure that surrounds the chromosomes of cells, separating them from the cytoplasm and other structures. During mitosis, the dividing of a cell into two daughter cells, the nuclear envelope is broken down to allow the genetic information to duplicate and separate, and is reformed around the daughter chromosomes to form the nuclei of the daughter cells.

Correct reformation of the nuclear envelope is critical to all cells that go through mitosis [29]. Defects in its formation have been linked to cancers as well as other human diseases; for example, increased malleability of the nuclear envelope is observed in small-cell lung carcinomas. Its function is clear, but agreement of the research community upon exactly how its formation is regulated has not yet been achieved; knowing this may give clues as to why it differs in cancerous and diseased cells.

Electron microscopy allows imaging of the cell and the structures relevant to the nuclear envelope, and segmentation can be performed to identify these structures throughout the different stages of mitosis. Any manual segmentation is a lengthy process that involves a high level of expertise and corroboration, and will intrinsically be susceptible to human error. By automating parts of the segmentation we can save significant time and effort spent by the expert user. An example segmentation of part of an image obtained by electron microscopy is shown in Figure 1.1.


(a) Sample EM image



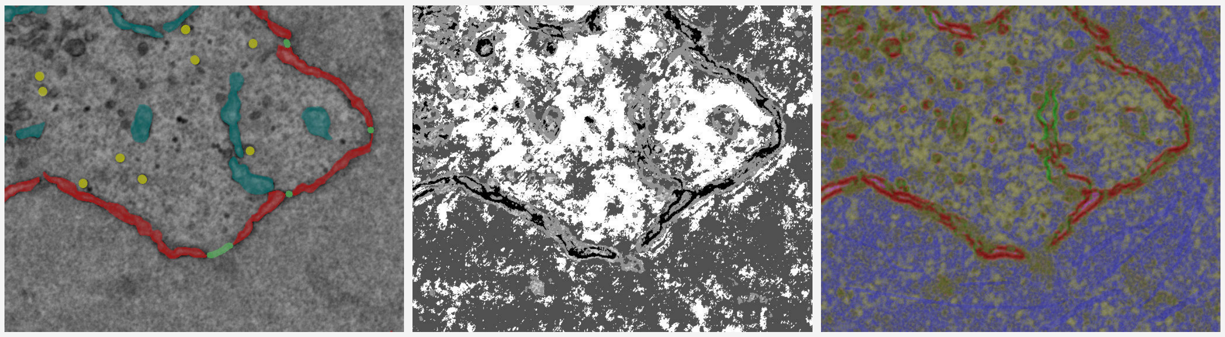
(b) Manual segmentation

Figure 1.1: Example segmentation of EM image. The nuclear envelope is the red structure in (b).

Computational biological image segmentation is a hot topic at the moment, for only recently have the techniques and power required to prove its worth been demonstrated. The imaging tool FIJI [39] ships with many segmentation plugins, whilst Ilastik [44] has also proven popular within the community. All these techniques have a machine learning element and provide segmentation based on a semi-automatic approach, allowing the user to interactively identify samples of a number of classes within an image by painting over regions. A classifier is trained from this labelling, which can then be applied to further images with user interaction only to refine the classifier.

Whilst there are tried and tested methods available for generic segmentation with FIJI plugins and Ilastik, none have had success at providing a correct or usable segmentation of the nuclear envelope. An example comparison between the ground truth and results obtained with FIJI's Advanced Weka Segmentation plugin and Ilastik are shown in Figure 1.2. Because of the nature of these programs as generally available pieces of software, the techniques used are too general and no consideration can be given to the semantics of the object being segmented. There has been research into segmenting specific parts of the sub-cellular structure, but none has focused on the nuclear envelope.

The aim of the project is therefore to provide the ability to segment the nuclear envelope from EM images of cells using a semi-automated approach from which a 3D reconstruction of the nucleus can be built. On average the current manual segmentation process takes 2 to 3 weeks to segment the nuclear envelope from a data set of images of a single cell. By reducing the time spent obtaining the segmentation the throughput of valuable research data can be increased.



(a) Ground truth **(b)** FIJI **(c)** Ilastik

Figure 1.2: A comparison of segmentations obtained using FIJI and Ilastik with the manual segmentation giving a ground truth. We can see that in general the nuclear envelope is picked up well, but there is confusion between it and the endoplasmic reticulum which have very similar features. The areas of cytoplasm and nucleus are also fairly well segmented, correctly forming the majority of pixels in those areas, however in general the classification produces quite considerable uncertainty.

This aim is complicated by a number of characteristics of the nuclear envelope:

- At different stages of mitosis the nuclear envelope can contain gaps that make overall shape description difficult.
- The nuclear envelope breaks down and reforms during mitosis; its shape changes over time.
- Different diseases and cancers affect the shape and completeness of the nuclear envelope.
- No two nuclei have easily identified relationships in their shapes.

These complexities eliminate the use of shape features which have recently been successful in automating the segmentation of mitochondria in EM images. Images obtained through EM techniques are also greyscale, therefore a successful algorithm cannot utilise colour and will have to rely solely on textural information. There is however a key characteristic of the nuclear envelope: the texture of the chromosomes it contains when compared with the cytoplasm on the outside of the nucleus. This distinctive feature is relied upon heavily in the algorithm presented in this report.

We also only focus on a cell at a late stage of mitosis when the nuclear envelope is almost fully formed. This is a simplification which enables a smaller problem to be solved, with the idea that this can be extended through further research to apply to nuclei at all stages of mitosis. Some of the techniques we present do generalise well to the nuclear envelope at other stages of mitosis, so it is certain that this algorithm contributes a solid base to a further investigation.

There are steps in the manual process, such as registration and brightness equalisation, that are difficult to automate and are outside the scope of this project. These take considerably less time than the segmentation and are thus not the bottleneck in the process. Furthermore, the segmentation will always need at least one manual step to deal with noise in some EM images, so only a semi-automated algorithm is achievable.

Whilst not specifically the subject of this project, the vision of the end-users at CRUK is to have a suite of algorithms to perform segmentations of all the sub-cellular structures detailed in Figure 1.1. These structures go through some interactions with the nuclear envelope as the cell goes through mitosis, and these interactions may hold the clue to correctly modelling the reformation of the nuclear envelope.

1.1 Contributions

This report is intended to be read and understood by any person regardless of their understanding of the relevant computing or biological subject material. As such, not all chapters are required to understand the contributions of the project and the work that has been carried out; which chapters are required depends on your interests and expertise. The contributions of this report are as follows:

- EM images are susceptible to noise and other intricacies which can significantly affect their resulting segmentations. Understanding how these images are obtained and therefore how these problems can arise is important for the discussions of how they affect the algorithm later in the process. Chapter 2 provides the background to the electron microscopy process and the relevant biological knowledge, whilst Chapter 5 looks in detail at these problems within the sample dataset from which the algorithm was developed.
- A background to the computational techniques considered and used in this project is presented in Chapter 3 and this is built upon when looking at current research into segmentation of EM images in Chapter 4.
- The algorithm presented is a pipeline of well documented image-processing techniques. Chapter 6 gives a suitable overview of this pipeline and details how points on the inside are first identified before being used to locate the nucleus, a step that is key and underpins the rest of the algorithm. This chapter is the most useful to gain insight into the algorithm without delving into implementation specifics; these are presented in Chapter 8 for those who wish to understand the operation of the algorithm at the computational level.
- The end goal of being able to segment all the sub-cellular structures has been taken into account when developing the project, and the software is designed as a framework for the development of further segmentation algorithms that operate upon EM images similar to those examined here. Chapter 7 details this framework and other general implementation features, including the use of FIJI and third-party plugins.
- Most of the initial work carried out looked at using machine learning techniques and augmenting state of the art classification techniques. This ultimately led to the hand designed algorithm presented in this report, which may seem a step back from the focus of current research. A discussion of the algorithm and the processes that led to this change is presented in Chapter 9.
- Having to rely purely on texture and grey level information gives rise to a number of problems that can occur in the pipeline as discussed in Chapter 10. These can result in parts of the nuclear envelope not being segmented, or other structures being erroneously segmented as part of the nuclear envelope, and this shows the necessity of manual corrections before the 3D reconstruction.
- Even with these problems, Chapter 11 shows that the reduction in time to perform the segmentation, inclusive of manual corrections, gives an overall processing time reduction from almost one month to one day at most. Furthermore a considerable improvement in segmentation quality over the closest state of the art methods is shown.
- There are many possibilities to extend this project not only to identify the other structures in the cell, but to investigate refining the algorithm to cater for cells at earlier stages of mitosis. A GPU implementation of the algorithm may also significantly reduce execution time; these and other ideas for further work are detailed in Chapter 13.

1.2 Collaboration

This research is pursued in conjunction with Cancer Research UK (CRUK) who have provided the EM datasets and biological background and insight.

Dr. Lucy Collinson heads the Electron Microscopy Unit at the CRUK London Research Institute. This team is comprised of four experienced electron microscopists, including Dr. Christopher Peddie, who provide a complete range of electron microscopy services to research scientists using state of the art equipment and techniques.

Prof. Banafshe Larijani heads the Cell Biophysics group, also at the London Research Institute, whose research interest lies in the proteo-lipid regulation of nuclear envelope assembly.

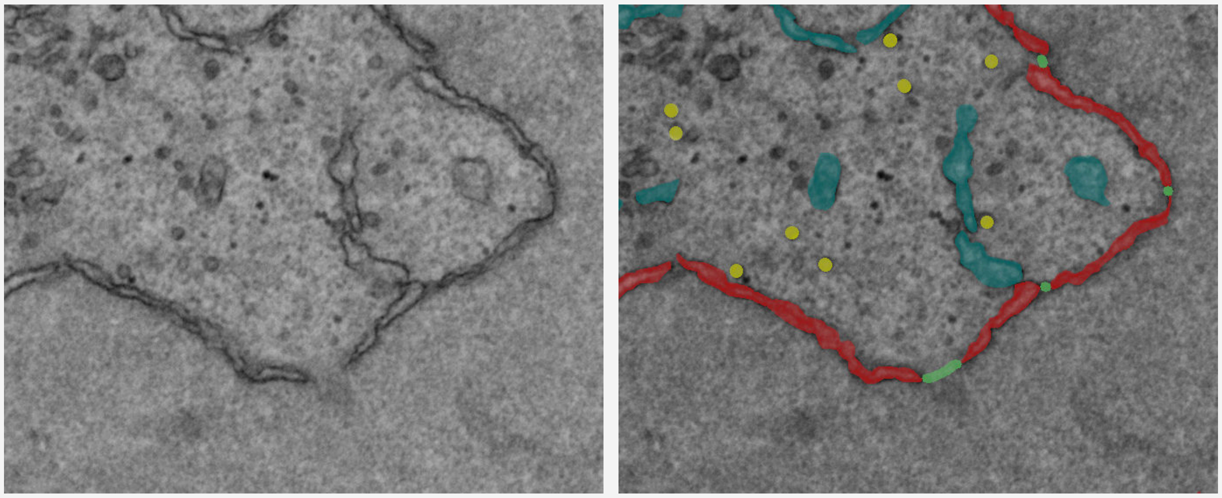
Supervision of the project within the Department of Computing was by Prof. Daniel Rueckert, head of the Biomedical Image Analysis Group, and Dr. Luis Pizarro. Prof. Rueckert's research focus is upon the development of algorithms for image processing and analysis and their translation to applications in medical image computing. Dr. Pizarro's research interests are image processing, computer vision, machine learning and biological and medical imaging.

2 Background: Cell Biology

This chapter gives the necessary knowledge to understand the biological content of the project.

2.1 Sub-cellular Structures of Interest Observable with EM

A section of an example electron microscopy image of the type this project seeks to segment is shown along with its manual segmentation in Figure 2.1.



(a) Sample EM image **(b)** Manual segmentation of sample

Figure 2.1: Example segmentation

The key biological features are [1, 2, 3]:

Nucleus To the bottom and right of the sample image and easily identified by eye as having a fairly regular texture and low variance of grey value. The nucleus contains the genetic material of the cell as well as other structures that have important roles in mitosis.

Nuclear Envelope (red) Contains the nucleus and characterised by two dark edges at a regular distance apart, a double lipid bi-layer structure, with the gap between the edges called the perinuclear space. A lipid bi-layer is a thin membrane made of two layers of lipids, a general category of molecules which function as signalling molecules, for passing information between cells, as well as modulators of membrane morphology. The nuclear envelope is visually identified by marking the boundary between the different textures inside and outside the nucleus. It is typically 40nm thick and encloses the nucleus separating it from the cytoplasm. Gaps in the nuclear envelope structure are shown in green.

Nuclear Pores Identified by a single dark line that fills a gap between two close together sections of nuclear envelope. They enable the transfer and exchange of materials between the nucleus and the cytoplasm and are typically 100-125nm in diameter, being roughly circular.

Endoplasmic Reticulum (teal) Similar to the nuclear envelope, however it is not a container for the nucleus and is therefore identified as being surrounded by cytoplasm. The endoplasmic reticulum is one structure with the outer lipid bi-layer of the nuclear envelope and often expands to the membrane separating the cell from its outer environment. It is also another important compartment that plays a large part in the reformation of the nuclear envelope during mitosis.

Cytoplasm Identified by the more coarse grained and less regular pattern outside of the nucleus.

Vesicles (yellow) Small dark circles in the cytoplasm, with a very regular size and intensity profile.

2.2 Mitosis

Mitosis is the division of a nucleus into two daughter nuclei following the duplication of the genetic material in the parent nucleus. It is described by a number of discrete observable stages, or phases [10]:

Interphase This is the state in which the cell is most stable and the nuclear envelope is completely formed, during which the cell replicates all its genetic material and structures that are essential for the daughter cells to subsequently undergo mitosis. An example of a cell at interphase is shown in Figure 2.2a.

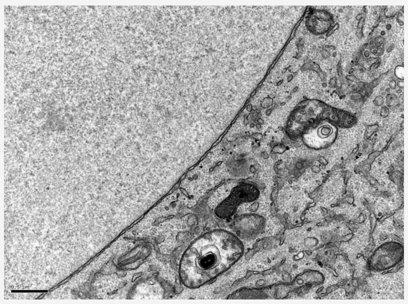
Prophase Chromatin, genetic material combining DNA and proteins, begins to condense to form chromosomes which can then be observed through a microscope.

Prometaphase The nuclear envelope breaks down, the chromosomes are fully formed in pairs and begin moving towards the middle of the nucleus.

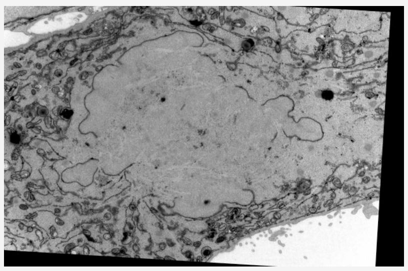
Metaphase The chromosomes align along the middle of the nucleus, ensuring that in the next phase each daughter receives a single copy of each chromosome.

Anaphase Paired chromosomes separate and move to opposite sides of the cell. An example of a cell at anaphase is shown in Figure 2.2b.

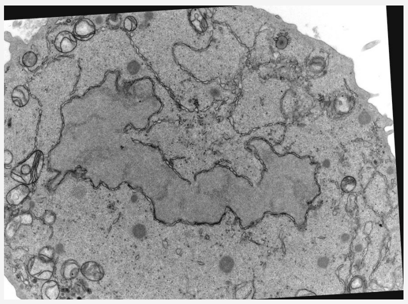
Telophase New membranes form around the daughter nuclei as their nuclear envelopes, the chromosomes also disperse and are no longer visible under a microscope. An example of a cell at telophase is shown in Figure 2.2c



(a) A high magnification image of a cell at interphase; the nuclear envelope is completely formed around the nucleus to the top left of the image.



(b) An image of a cell at anaphase; the nuclear envelope is distinguishable but does not form a continuous boundary around the nucleus.



(c) An image of a cell at telophase; the nuclear envelope almost forms a complete boundary around the nucleus and only a few gaps remain.

Figure 2.2

2.3 Electron Microscopy

Traditionally, cell biology has relied on light and fluorescence microscopy in order to analyse cells and tissues, however ultimately these techniques are limited in resolution by the wavelength of light. Electron microscopes offer a much increased resolution due to the much shorter wavelength of electrons. In electron microscopy, a beam of electrons replaces the traditional beam of light, and the behaviour of the electron as it hits a sample is captured. The interaction of the electron beam and the sample is what defines the image that is seen.

Electron microscopy allows biologists to analyse sub-cellular structures such as mitochondria and nuclei. The resolution of these images has also recently made EM images a popular topic for advancing the state-of-the-art in terms of image processing and segmentation techniques, as the sub-cellular structures are often complex and consequently require complex segmentation techniques.

This section is meant as a brief introduction to electron microscopy in order to understand how the images associated with the project are obtained. Here we take a look at the two main types of electron microscopy actively used in current research.

2.3.1 Transmission Electron Microscopy (TEM)

In TEM, a beam of electrons is incident upon an ultra thin sections [34]. The behaviour of the electrons as they hit the sections is dependent upon its properties; some electrons are absorbed, others are scattered or pass through.

The design of the TEM consists of an electron gun, a series of lenses and a charge-coupled device (CCD) camera. A condenser lens focuses the electron beam on the specimen. Three other lenses work to focus the transmitted electrons onto the camera to record the image. The overall quality of the image is dependent upon the lenses and their configurations. Magnifications of up to 1,000,000 times are possible with TEM.

The ultra thin sections must withstand a vacuum in the electron microscope [2]. To do so they are fixed to an epoxy resin after staining with heavy metals and dehydrating with ethanol. Sections around 70nm are taken from the specimen such that they are thin enough for the electron beam to pass through, however for large structures this requires many serial sections to be taken and imaged. Therefore, to create a full 3D reconstruction images for each section must be aligned and segmented.

2.3.2 Scanning Electron Microscopy (SEM)

The primary concept of SEM is that of a focused electron beam being scanned across a specimen in a raster approach, that is from top left to bottom right in horizontal lines. As with TEM, when the beam is incident upon the specimen there are many possible behaviours [34]. The most common imaging mode collects the low energy secondary electrons that are ejected by the atoms of the specimen when interacting with the incident electrons.

An alternative mode measures backscattered electrons, those from the original beam that are repelled when interacting with the atoms of the specimen. Heavier atoms (those with higher atomic number) will backscatter more strongly that lighter atoms, and this provides the contrast in the image.

A couple of variants of SEM exist [50], namely focused ion beam SEM (FIB/SEM) and serial block face SEM (SBF/SEM). The particular advantage to these methods is that the complete sample is in situ within the chamber of the microscope, as opposed to slices of the specimen being placed individually into the microscope to be imaged. From a 3D reconstruction perspective, this means that there are no problems with registration of multiple 2D slices as the specimen is sliced within the microscope itself and is stationary at all times.

The two methods vary only upon how they incrementally remove slices of the specimen. In FIB/SEM, a gallium ion beam is used to 'mill' away material, whereas the SBF/SEM uses a diamond knife to slice away layers of material. A backscattered electron detector is used to image the specimen with the same scanning beam as standard SEM techniques.

In biological applications, TEM is more popular than SEM, however SEM is used particularly effectively for discovering the surface topology of cells, bacteria and viruses.

3 Background: Computer Vision

This section gives an overview of traditional computer vision techniques that aid understanding of the current state of the art. Section 3.1 provides nearly all the knowledge required to understand the algorithm by introducing some standard and well known image processing techniques. Algorithms specific to the segmentation problem and methods for evaluating segmentation quality are detailed in Sections 3.2 and 3.3.

Many state of the art methods utilise aspects of machine learning and obtain good results, but this is often heavily dependent on the choice and descriptive power of features that can be extracted from images. An overview of feature extraction is presented in Section 3.4. Machine learning techniques and their relationship with segmentation algorithms are detailed in Section 3.5.

3.1 Image-processing Techniques

3.1.1 Image Representation

An image is in most cases represented as a two-dimensional array of intensity values within a range defined by the particular application or method for obtaining the image. The simplest possible image representation is a **binary** image, where each pixel has a value of either 0 or 1. Binary images require very little storage space but are usually only the result of some image-processing techniques that aim to identify salient features.

A **greyscale** image has values in the range 0 to 255. For a moderate amount of detail each value can be stored in a byte and can take any integer value between 0 and 255; such an image is an **8-bit greyscale** image. More detail is obtained by increasing each value to be represented by 16-bits, with the drawback of increased image storage space required; this is a **16-bit greyscale** image. Greyscale images are more commonly found in applications than binary images and often require more complex processing techniques to aid their understanding.

Full **colour** images in the RGB model have three components of colour; red, green and blue. Each colour component is in the range 0 to 255, giving each pixel value a total of 3 bytes to store. Having colour images available can often simplify the application of computer vision, however again more complex processing techniques may be required.

An alternative representation of an image is a histogram of pixel intensities. For a greyscale image, the histogram is a plot of intensity (in the range 0-255) against the frequency of that particular intensity value in the image [45]. The intensity histogram can be useful as a simple representation of global information in the image, from which particular values can be extracted, and a visualisation of the effects of image transformations.

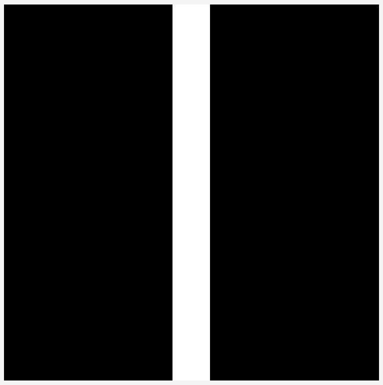
3.1.2 Binary Image Processing

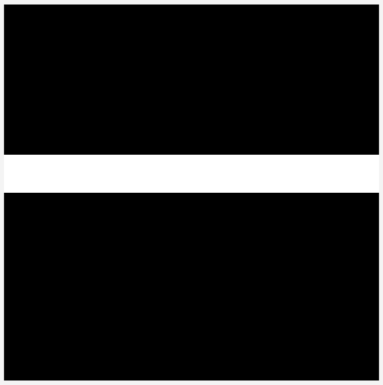
Binary Operators

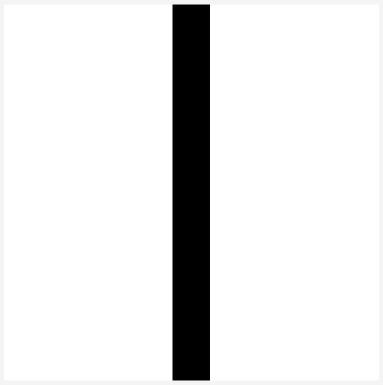
Given a number of binary images we can define several simple binary operations upon these images.

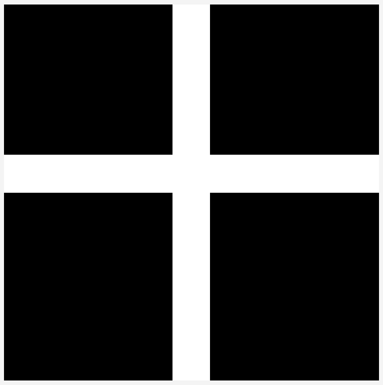
Inversion (negation) For each image, replaces each pixel intensity with the opposite value.

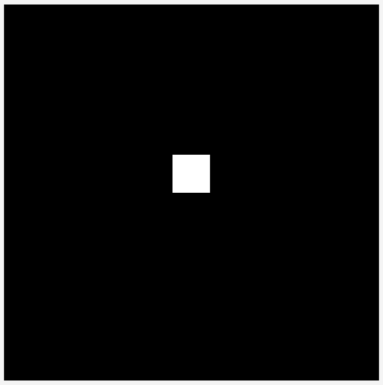
And (intersection) A pixel has value 1 in the output image if both the input images have value 1 in the same position as the pixel.


(a) Binary Image 1


(b) Binary Image 2


(c) Inversion of Image 1


(d) Image 1 or Image 2


(e) Image 1 and Image 2

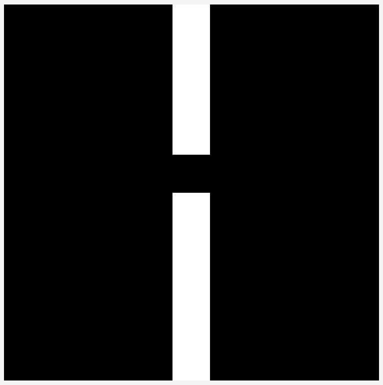

(f) Image 1 subtract Image 2

Figure 3.1: Binary image operations.

Or (union) A pixel has value 1 in the output image if either of the input images has value 1 in the same position as the pixel.

Subtract (difference) A pixel has value 1 in the output if the first input image had value 1 and the second input had value 0 in the same pixel position, otherwise the output has pixel value 0.

These operations are simple, fast to compute and easily show relationships between two images. Examples of these four operations are shown in Figure 3.1.

Distance Transform

The resulting image of a distance transform applied to a binary image is a greyscale image where pixel values relate to the distance of that pixel from its nearest background pixel in the original image. It is also possible to compute distance from object pixels by performing an inversion before the distance transform.

The individual distance values can be computed using any particular distance measure [23]. The city-block distance measure counts the number of horizontal or vertical hops necessary to get to the nearest background pixel. The Euclidean measure takes the straight line distance to the nearest background pixel. An example of the distance transform applied to a sample binary image is shown in Figure 3.2.

$$\begin{matrix}0&0&0&0&0&0&0\\0&1&1&1&1&1&0\\0&1&1&1&1&1&0\\0&1&1&1&1&1&0\\0&1&1&1&1&1&0\\0&1&1&1&1&1&0\\0&0&0&0&0&0&0\end{matrix}$$

(a) Image with object pixels denoted by 1

$$\begin{matrix}0&0&0&0&0&0&0\\0&1&1&1&1&1&0\\0&1&2&2&2&1&0\\0&1&2&3&2&1&0\\0&1&2&2&2&1&0\\0&1&1&1&1&1&0\\0&0&0&0&0&0&0\end{matrix}$$

(b) City-block distance transform

Figure 3.2: Distance transform of a binary image. The city block distance transform in (b) gives the number of horizontal or vertical hops from an object pixel in (a) to its nearest background pixel.

3.1.3 Convolution and Neighbourhood Operations

A convolution is a simple linear filtering operation whose mathematical definition is found in [45] and nearly all image-processing texts, but is skipped here as it is not necessary for understanding the concept and its application to image processing. An image intensity function is convolved with, or multiplied by, another typically smaller intensity function termed a filter or convolution kernel.

In practice a kernel is defined as a square matrix with odd dimensions such that there is a unique centre entry. This kernel is placed with each pixel in the image located at this unique centre, and this pixel's intensity is updated to be the sum of the intensities in the original image multiplied by those in the kernel where the image and kernel overlap. Convolution is therefore simply viewed as updating a pixel's value with a weighted sum of those in its immediate neighbourhood, where the size of the neighbourhood is defined by the size of the convolution kernel. Kernels should have normalised weights to prevent intensity values occurring outside the range defined by the image.

An example of a convolution kernel that calculates the mean of a 3x3 neighbourhood around the centre pixel is shown in Figure 3.3. When this is centred at a particular pixel each of its immediate neighbour's pixel values are multiplied by $\frac{1}{9}$, the sum of these values is therefore the mean over a 3x3 neighbourhood. The centre pixel's value is then updated with this calculated value.

$$\begin{bmatrix}\frac{1}{9}&\frac{1}{9}&\frac{1}{9}\\\frac{1}{9}&\frac{1}{9}&\frac{1}{9}\\\frac{1}{9}&\frac{1}{9}&\frac{1}{9}\end{bmatrix}$$

Figure 3.3: Mean convolution kernel

We can extend this notion to general operations on local neighbourhoods of pixels, each of which can again be defined over an arbitrary odd neighbourhood size. The most commonly found neighbourhood operations are the maximum, minimum, variance and median. The results of a maximum and variance neighbourhood operation being passed over a sample image is shown in Figure 3.4.

$$\begin{bmatrix}2&5&10&5&4\\5&2&4&12&8\\5&2&12&8&8\\2&6&9&10&1\\2&1&1&9&8\end{bmatrix}$$

(a) 5x5 array of pixel intensity values.

$$\begin{bmatrix}5&10&12&12&12\\5&12&12&12&12\\6&12&12&12&12\\6&12&12&12&10\\6&9&10&10&10\end{bmatrix}$$

(b) Pixel values after a maximum operation over a 3x3 neighbourhood.

$$\begin{bmatrix}3&8.7&14.7&11.4&12.9\\2.7&12.7&15.8&9.6&7.9\\3.5&12.6&15.4&12.8&13.8\\4&15.3&17.3&14.5&10.3\\4.9&10.7&16.8&17.5&16.7\end{bmatrix}$$

(c) Pixel values after a variance operation over a 3x3 neighbourhood.

Figure 3.4: Showing neighbourhood operations.

3.1.4 Noise

Any images that are obtained, transmitted or processed are susceptible to random errors that are most often called noise [45]. The noise can also be dependent on the content of the image. When looking at individual pixel intensities and their neighbourhoods, noise can be seen as intensity values that are quite different to those within the pixel's neighbourhood. Thus techniques for dealing with noise are commonly associated with smoothing techniques.

Median and Mean Filters

The neighbourhood operations of median and mean are simple and quick ways to remove noise as previously described. The mean convolution can be represented as an equally weighted convolution kernel, as shown in Figure 3.3, whereas the median of a neighbourhood is a more general neighbourhood operation.

The median operation requires a sorting of the neighbourhood intensity values and as such is slightly less efficient than the mean convolution. For both operations, the size of the neighbourhood considered affects the amount of smoothing present in the resulting image.

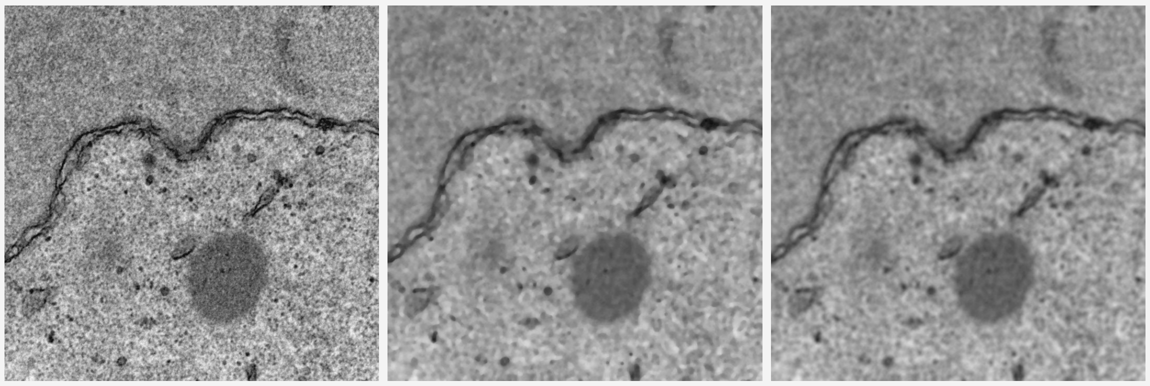

(a) Sample image **(b)** Median filter over a 5 pixel neighbourhood **(c)** Mean filter over a 5 pixel neighbourhood

Figure 3.5: Smoothing an image with mean and median filters

Gaussian Filter

Noise is often described by its probabilistic characteristics, and as such can be associated with the Gaussian distribution as a very good approximation to noise that occurs in many practical cases [45]. The Gaussian distribution is characterised in one dimension by a bell-shaped curve, in two by a bell shape and has one parameter, σ, the standard deviation of the distribution.

For this distribution to be used as a smoothing operator we modify the parameter σ to give the approximate neighbourhood size that we wish to consider. Larger values of σ correspond to wider bell shapes and thus larger neighbourhoods over which smoothing takes place. Since the image is discrete but the Gaussian distribution is continuous, the kernel we define is a discrete approximation to the continuous Gaussian for a particular value of σ. Given the Gaussian distribution, pixels further from the centre of the resulting convolution kernel have less influence on the centre value than those closer to it; the effect created by the bell-shape. The discrete convolution kernel for an approximate Gaussian smoothing with $\sigma = 1.0$ is shown in Figure 3.6, with an example of its application to an image in Figure 3.7.

$$\frac{1}{273}\begin{bmatrix}1 & 4 & 7 & 4 & 1\\ 4 & 16 & 26 & 16 & 4\\ 7 & 26 & 41 & 26 & 7\\ 4 & 16 & 26 & 16 & 4\\ 1 & 4 & 7 & 4 & 1\end{bmatrix}$$

Figure 3.6: Approximate Gaussian filter with $\sigma = 1.0$ [8]

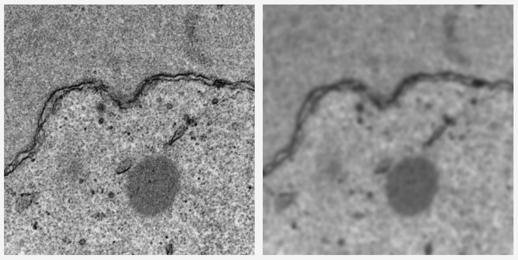

(a) Sample image **(b)** Image smoothed with Gaussian, $\sigma = 5$

Figure 3.7: Gaussian convolution kernels

3.1.5 The Fourier Transform

One particularly well documented method of mathematical analysis is that of the Fourier transform. The aim of such a transformation is to decompose a periodic function into a spectrum which gives the absolute magnitudes of contributions to the original function of a set of sinusoidal curves. The original periodic function can then be recreated by the superposition of each sinusoidal curve contributing the amount determined by its value in the spectrum.

The Fourier transform has several useful features:

- The spectrum of a given function is independent from its phase.
- The spectrum can be normalised to remove dependence upon the individual magnitudes and make the spectrum values relative.
- Comparison of two spectra for similarity is simpler than comparing their respective periodic functions.

This technique relates to image processing for analysis of shape as well as texture. For recognising similar shapes we can consistently choose a reference point within an object that we loosely call the centre of mass. If we then take a series of vectors with increasing angle relative to the x-axis and project them from this centre to the object boundary, we obtain a periodic function of vector length plotted against angle. The Fourier transform of this periodic function gives a descriptor of the object's shape.

Given that the shapes are already defined by segmentations or masks we can use this technique to find similar shapes within any image by using the same process and comparing spectra. Alternatively, if we know a particular shape we are looking for we can identify locations of these shapes in an image using a predefined spectrum. The advantages of spectra being independent from phase and normalised means that we can identify shapes regardless of any rotation or scaling transformations.

The same transform can be used to classify texture. If we consider a window over an image, it can be seen as periodic in two dimensions, x and y, where the greyscale intensities represent the function value. If we apply the Fourier transform to the horizontal and vertical scan lines of this window we obtain a 2D spectrum which can be used to describe the texture. Spectra can be compared as before to identify similarly textured regions.

Band-pass filtering

Noise in images can also be thought of as the presence of high frequency components in the periodic function representing greyscale intensities. Using the Fourier transform for 2D windows we can remove any non-zero entries in the spectrum for sinusoidal curves that have frequencies above a certain threshold. The Fourier transform is an invertible operation so we can reconstruct the window given the spectrum; as we have removed the spectrum entries for high frequency components the reconstructed image will appear smoother.

The same process can be used to filter out low frequency components of the spectrum, which has the effect of removing larger structures from the image. We can now see that the definition of low and high frequency thresholds determines a number of structure sizes that are preserved. Structures smaller than the low threshold and higher than the high threshold are filtered out; this is a band-pass filtering.

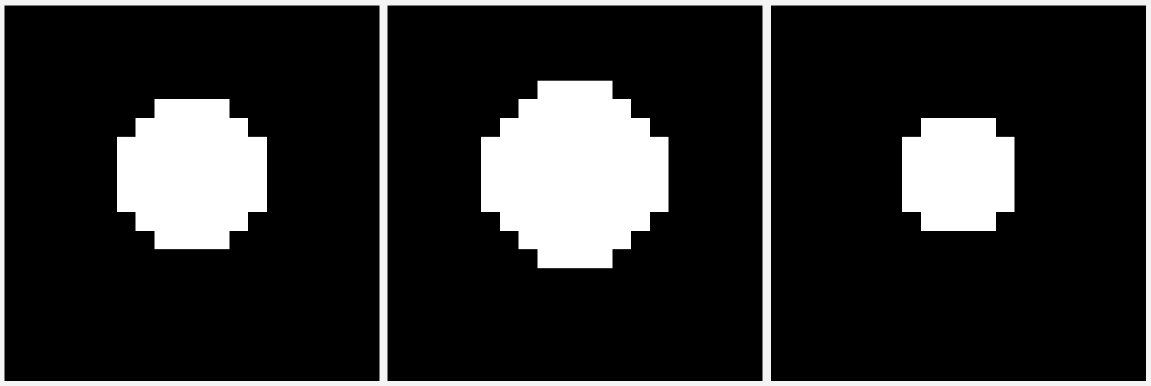
3.1.6 Morphology

Morphological operations are slightly different from the convolution operations defined previously as they primarily operate on point sets, connectivity and shape rather than pixel intensity values and distributions [45]. They are typically used to simplify shapes, filter noise or modify the object structure in a meaningful and well defined manner. These operations also require a relation that identifies which pixels in the image are considered as objects and which as background. For the simpler binary image case that we look at here, we could consider white pixels to be objects and black to be background or vice versa.

The morphological transformation of an image requires the definition of an additional smaller structuring element expressed with respect to a local origin. This structuring element is moved systematically across the entire image with each pixel placed at the local origin of the element, and the output of the operation is stored in a separate output image. Typical structuring elements include squares, circles and lines, but can also be arbitrarily shaped according to the application.

There are four primary morphological operations that can be defined based on this notion. A **dilation** places the structuring element over each object pixel in the original image. Each pixel which is then defined as an object pixel by the structuring element being placed there is defined as an object pixel in the output image. Hence this can be seen as an increasing or growing transformation.

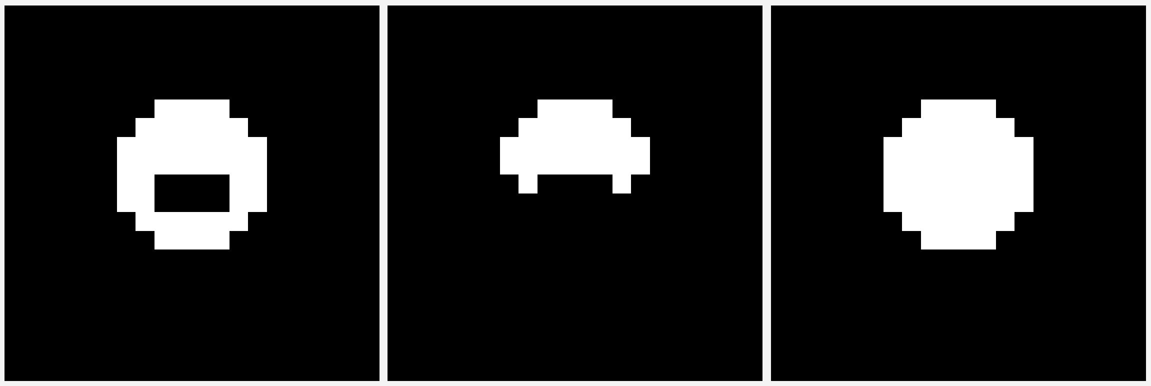
The dual operator of a dilation is **erosion**, where an object pixel is only carried over to the output image if all object pixels in the structuring element are present as object pixels in the input image. An erosion can thus be seen as shrinking or reducing the input image according to the structuring element. Example of the erosion and dilation operations are shown in Figure 3.8.



(a) Sample binary image **(b)** Dilation **(c)** Erosion

Figure 3.8: The erosion and dilation operators using a 1-pixel radius circular structuring element with the local origin at the centre of the circle.

Two further morphological operators are **opening** and **closing**; an opening is an erosion followed by a dilation and a closing is a dilation followed by an erosion. These operations in general are used to eliminate image details that are smaller than the structuring element without distorting the overall shape of objects in the input image [45]. The closing operator is useful for filling small holes, connecting close objects and smoothing the outline of objects, with the opening operator effectively performing the inverse by opening holes. Examples of the opening and closing operators are shown in Figure 3.9.



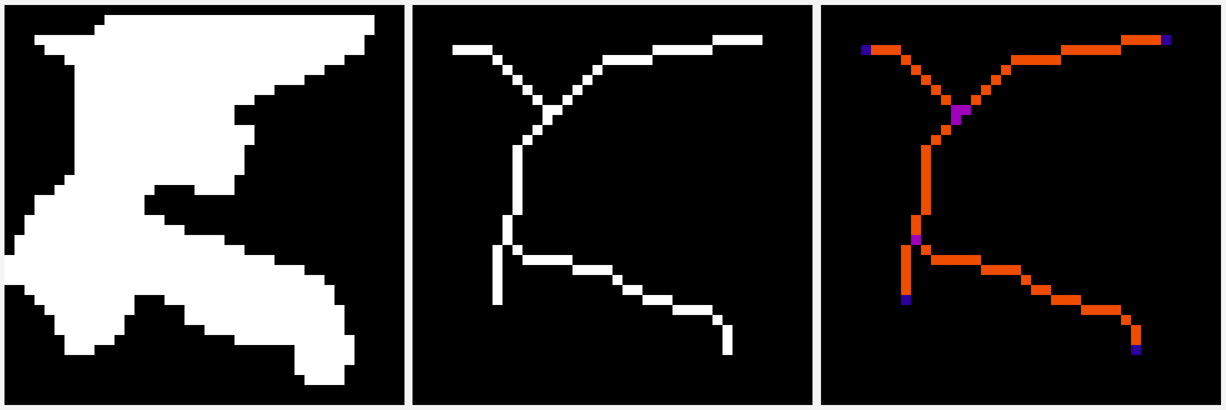
(a) Sample binary image **(b)** Opening **(c)** Closing

Figure 3.9: The opening and closing operators using a 1-pixel radius circular structuring element with the local origin at the centre of the circle.

Morphological operators are useful for refining regions that may have small holes due to noise, or for general manipulation of object and region shapes. Their extension from binary to greyscale images is simple using minimum and maximum operations. Erosion assigns the minimum intensity value found in the neighbourhood of a pixel in the input image, and dilation the maximum value. The structuring element in this case simply defines the neighbourhood over which intensity values are considered.

Skeletonisation

The skeleton of objects in binary images is a stick-figure like representation, rather like the skeleton of the human body, where all points in the skeleton are at a locally maximum distance from the boundary of the object. For example, the skeleton of a circle is the point at the centre of the circle, and the skeleton of a square is a cross positioned at the centre of the square, extending to its vertices. The skeleton can either be found through successive symmetrical erosion operations, performed until any more erosions would change the topology of the structure, or through local maxima in a distance map. Figure 3.10 shows an example of the skeleton operation upon a random binary shape.



(a) Sample binary shape **(b)** Skeletonisation of sample shape **(c)** Skeleton analysis

Figure 3.10: The skeleton representation of a binary shape

An alternative way to think about a skeleton in a continuous environment is consisting of a number of maximal balls [45]. That is, an object can be described as an infinite number of points and radii of circles placed at these points. However, in a discrete environment such as an image, the point-radius representation would only be an approximation to the real object.

The skeleton can also be used as an efficient representation of an object's topology by converting it to a graph [14]. For the 2D skeleton case we can classify pixels in the skeleton into three different classes based on their connectivity in the horizontal, vertical and diagonal directions:

- End-point pixels that have less than two neighbours.
- Junction pixels that have more than two neighbours.
- Edge pixels that have exactly two neighbours.

An example showing an analysis of the skeleton in Figure 3.10b is shown in Figure 3.10c. This analysis of the skeleton can give one or a number of graphs, depending on the number of separate objects present in the image, that can be further analysed to deduce properties of the skeleton and the original object.

3.1.7 Image Registration

Image registration is a critical technique for integrating and comparing images of the same objects that have been transformed in some way [51]. This transformation could be due to a number of reasons:

- Images are captured with different imaging processes, cameras or sensors.
- Images are captured at different times when the object has moved.
- Images are captured from different viewpoints.

The problem becomes one of searching for the best transformation between images such that the objects are registered in the same position in each image.

The first step in image registration is to decide upon a feature space between which different images can be compared, and this choice of features is one of the main factors affecting the success of the registration algorithm. Transformations are based upon this space and defined as a combination of many smaller image manipulations including rotation, scaling, shearing and translation. If these are not sufficient to describe the changes in the objects between images then higher level transformations can be used to give a more free form deformation.

Given the possible transformations to apply, the image registration algorithm becomes a search for the optimal combination of these transformations between the two images, given some measure of similarity in terms of the feature space. An example of the registration of two images from the dataset used in this project is shown in Figure 3.11. Although the two objects are quite dissimilar in shape it is noticeable that they can be better aligned.

3.1.8 Volume Rendering

The aim of volume rendering is to visualise volumetric data in some way that is meaningful and allows a human to interpret what the data represents. Many techniques exist, the most common being direct volume rendering where colour and transparency values as chosen for each voxel, a three-dimensional pixel. Volumetric data can arise from many applications, so we require application specific transfer functions to perform this mapping from voxel values to colours and transparency values based upon the structures and information that is required to be visualised. For the example of visualisation of a binary volumetric dataset one could choose to make black voxels transparent and give white voxels an arbitrary colour in order to visualise objects represented by the white voxels.

Figure 3.12 shows an example volume rendering using white pixels of binary images as object pixels and making black pixel transparent.

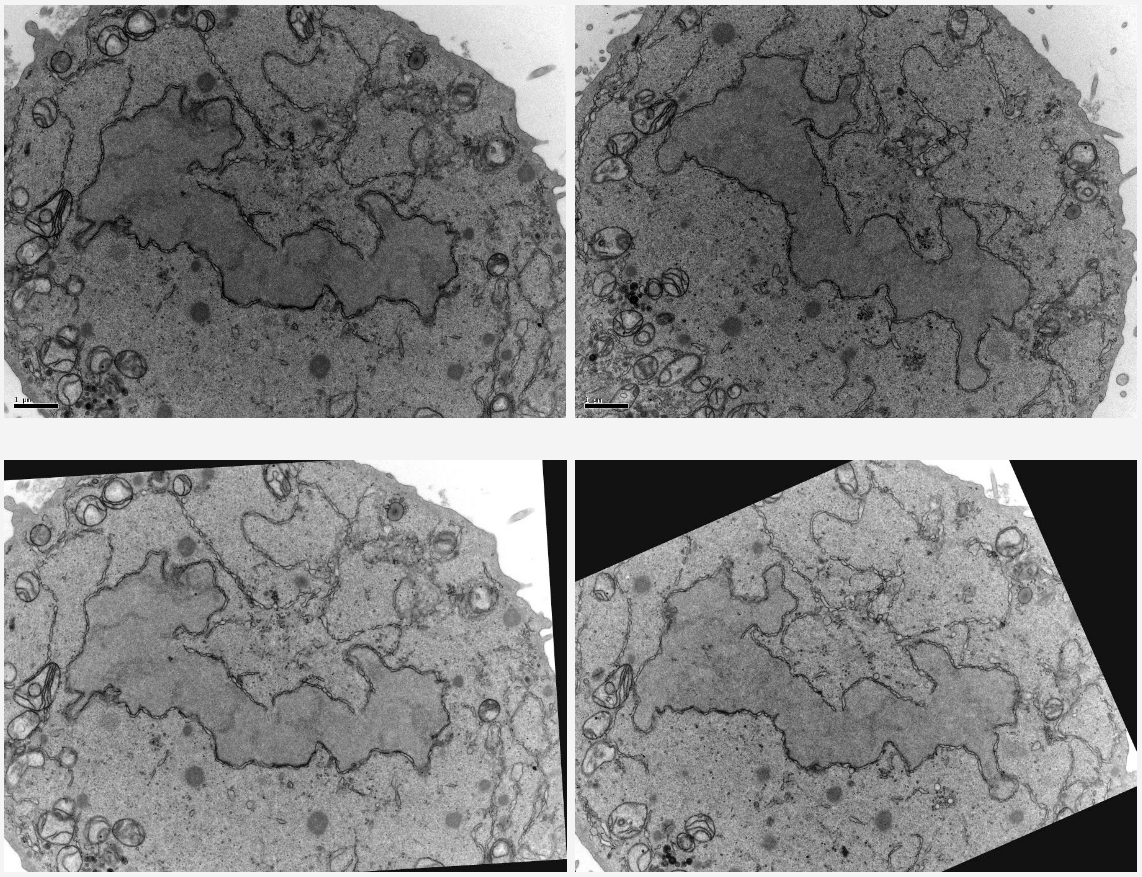


Figure 3.11: Image registration example. The top two images are original images from the dataset before they are registered as in the second row.

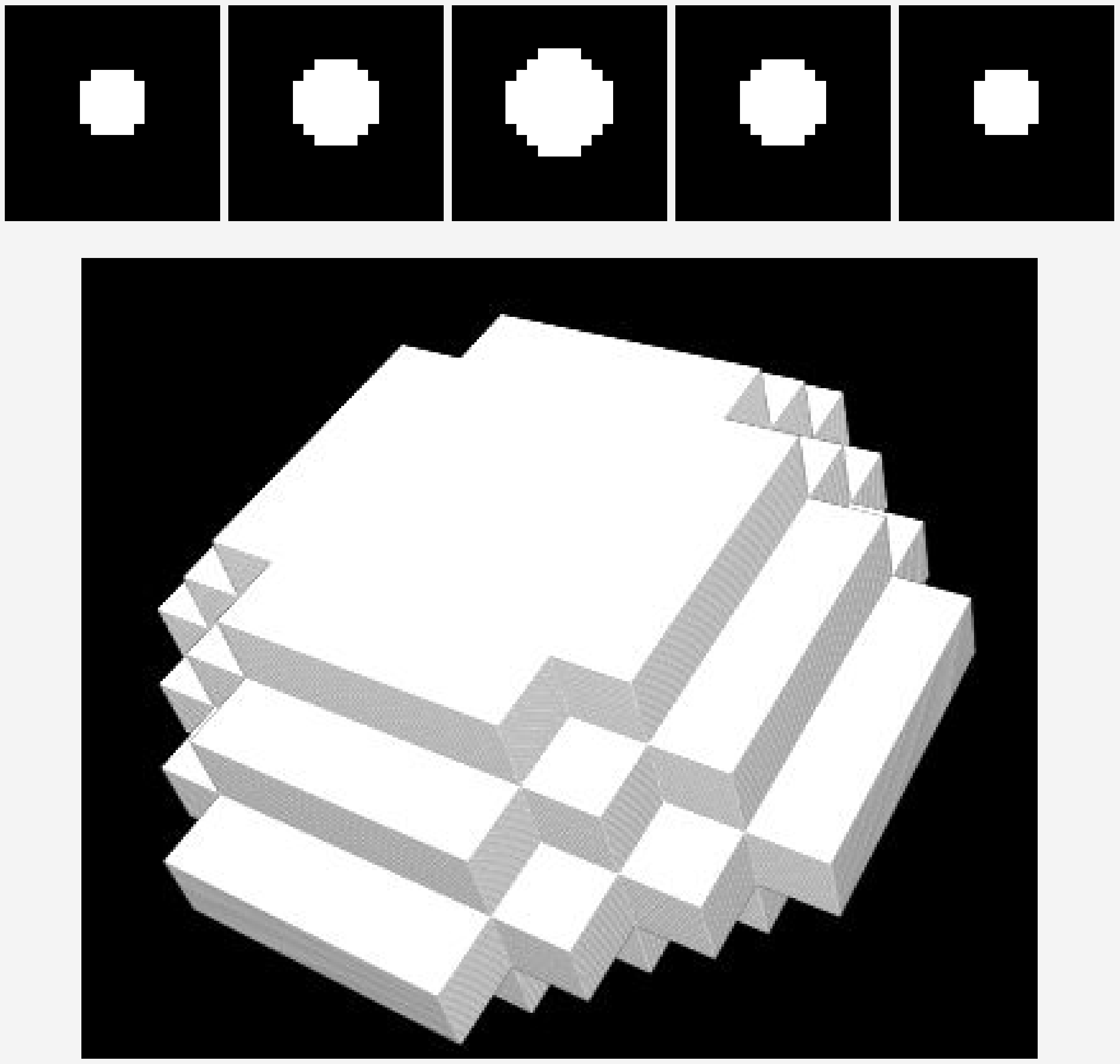


Figure 3.12: An example of volume rendering. The first 5 images are stacked together and rendered in 3D.

3.2 Segmentation Algorithms

All, bar one, traditional segmentation algorithms come in two flavours [45]. There are those that attempt to find a boundary map from the given input image through edge detection or similar methods. Other algorithms attempt to segment through defining regions within the image using a similarity measure between pixels based upon both local and global characteristics. These two representations are synonymous; a region can be represented by its closed boundary, and a closed boundary by the region that it defines. It is important to consider that whilst there is this synonymity, the different algorithms can be expected to produce different results and therefore different information. This is potentially useful as a good segmentation could result from combining the best of both strategies.

3.2.1 Thresholding

Objects can often be characterised by their colour or intensity as a way to distinguish them from the background within a particular image. Thresholding is the process of maintaining pixels whose intensity falls within a particular band or set of bands and removing pixels that fall them. The process produces a binary image where 1 indicates an intensity value within the thresholds and 0 outside the thresholds.

Thresholding is an inexpensive and fast operation that can often be done in real time using specialised hardware [45], however it is certainly not without its limitations. The application of this method needs to be simple; different objects within the same intensity bands cannot be segmented, and the specific thresholds required for a particular segmentation may not be readily found.

3.2.2 Edge Detection

A significant class of algorithms utilise the edges found from an image that are defined by changes in colour, intensity or texture as a gradient within the image. Among the most popular methods are edge detecting operators, such as the Laplacian, Prewitt and Sobel convolution filters. All these operators convolve a small square kernel, often of dimension 3x3, with the image to compute the gradient or second derivative at each pixel in the image, as shown in Figure 3.13. The resulting gradient map can then be thresholded to decide which edges are significant.

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

(a) Prewitt operator for gradient in the x-direction

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

(b) Prewitt operator for gradient in the y-direction

$$\begin{bmatrix} 5 & 6 & 52 & 18 & 5 \\ 5 & 6 & 52 & 18 & 5 \\ 5 & 6 & 52 & 18 & 5 \\ 5 & 6 & 52 & 18 & 5 \\ 5 & 6 & 52 & 18 & 5 \end{bmatrix}$$

(c) Sample 5x5 image

$$\begin{bmatrix} 3 & 141 & 36 & -141 & -39 \\ 3 & 141 & 36 & -141 & -39 \\ 3 & 141 & 36 & -141 & -39 \\ 3 & 141 & 36 & -141 & -39 \\ 3 & 141 & 36 & -141 & -39 \end{bmatrix}$$

(d) Derivative of sample image

Figure 3.13: Example showing the Prewitt edge detecting kernels passed over a sample image that has evidence of a strong edge in the x-direction. Notice that its gradient in the y-direction is 0; there is no change in intensity between adjacent pixels in that direction.

The results of such operators most often cannot be used directly for segmentation, instead additional processing is required to form a continuous edge map from the convolved image. Noise in the original image is also a source of errors in segmentation; edges can appear where there is no border in the original image, and vice versa. A popular technique is to smooth the image before edge detection in an attempt to reduce noise; this is most commonly achieved using a Gaussian filter.

Canny Edge Detection

Introduced by John Canny in 1986, a commonly used algorithm is that of Canny edge detection [45]. It is based upon three criteria that Canny aimed to satisfy to find an optimal edge detector:

- Important edges should not be missed, and there should be no spurious responses.
- Distance between the actual and located position of the edge should be minimal.
- Any given edge should only be marked once and noise should not create false edges. If there are two responses to any given edge, one should be marked false.

The first stage of his algorithm is to perform noise reduction by a convolution with a Gaussian filter. From here, estimates of the edge normal directions are calculated for each pixel using an edge detection operator, which are then rounded to one of the four directions that represent vertical, horizontal and diagonal gradients. This is then used to perform *non-maximum suppression* of edges; a pixel is determined to be an edge if its gradient magnitude is greater than those of both its immediate neighbours in the same direction as the gradient magnitude.

At this stage we have a binary image consisting of edge points. An important step is to now go back over the image and compute the magnitude of each edge point such that thresholding can be applied; edge points with higher gradient magnitude are more likely to be edges than those with a smaller magnitude. *Thresholding with hysteresis*, requiring the definition of two thresholds, is then used to identify the edges. A high threshold determines those pixels that are definitely an edge whereas a low threshold determines pixels that are also an edge if they are connected to any pixels above the high threshold. This is useful as low gradients often correspond to noise in images, however being connected to a high gradient pixel should increase the likelihood that it is part of an edge.

An optional final step in Canny's algorithm is to perform the previous steps at a number of different spatial resolutions by changing the standard deviation of the Gaussian used to smooth the image, and to then compile the information from each resolution into the result. Whilst this can produce an improved edge detection, it is common for an implementation to choose just one value for the standard deviation based upon the objects in the image and omit this final step.

Algorithms that rely on prior information

Many more algorithms for detecting edges exist [45], however they rely on prior information that is not necessarily available in most modern applications.

- **Border tracing** takes a predefined set of regions in an image and traces around the region to define its boundaries in terms of edges.
- If prior information such as the start and end point for borders, smoothness and curvature is known then the detection of the complete border can be transformed into a **graph search** problem.
- A **Hough transform** can also be used to detect features within a image based upon the identification of pre-described shapes within the image. While this process is not too sensitive to noise and can be used to segment objects of arbitrary shape, it is the fact that it requires the definition of the shapes that limits its use in natural segmentation applications.

3.2.3 Region-based Segmentation

Regions and edges have a synonymity as discussed earlier. In this section we look at algorithms that aim to grow regions directly from the original image. For these algorithms an important property to consider is that of homogeneity [45], and the goal of region-based segmentation is to divide an image into areas that maximise this property. Homogeneity can be defined using any features of the image, such as colour, intensity, shape or texture, and which features are used can greatly influence the complexity of shape and the accuracy of regions in the output.

Region Merging

The simplest way to grow regions is to begin with the original pixel values and regard pixels as regions which are then merged based upon some criteria that measures homogeneity. When regions of multiple pixels are defined, the region descriptors are most commonly based upon a statistical representation of the region, for example an intensity histogram, or the mean grey value of the region. Once regions are merged, the descriptor of the new larger region is recalculated as part of an iterative process which terminates when no more adjacent regions can be merged according to the specified criteria.

A particularly simple region merging algorithm is a flood fill algorithm. Given a set of regions and a starting pixel, this algorithm finds all connected pixels in the horizontal, vertical and diagonal directions that have either the same intensity value or are above or below a certain threshold. Alternatively some implementations can find all such maximal regions in an image that satisfy the criteria.

Region Splitting

As opposed to merging, region splitting considers the original image as a whole region and seeks to split this down into smaller regions, again using some similarity descriptor between sub-regions. While in theory a dual to merging, just applied in a reverse order, this method will most likely produce a different segmentation.

The most simple way to define sub-regions is to examine the 4 quadrants of a region to see if any can be split further. This is a quadtree approach [52], whereby a tree is formed with the root node denoting the original complete image, and leaves representing homogeneous regions. Any dissimilarity between sub-regions requires a node to be split and 4 child nodes created.

The quality of the segmentation produced by this method is very sensitive to the choice of split as well as the homogeneity criterion, and would require tuning at an application specific level. Two cases where the quadtree representation fails to segment regions properly or most efficiently is when the regions are not square, or a region overlaps more than one quadrant.

Watershed Segmentation

Algorithms for watershed segmentation are based upon the well known topographical concept that when water falls upon land, it will always fall to the lowest point it can reach by monotonically decreasing altitude. By applying this concept to segmentation, we say that individual grey levels correspond to an elevation, hence local low gradient regions correspond to geographical lowest points [48]. In general, algorithms attempt to group pixels into regions based on the catchment basins a drop of water would reach if it were to fall on each individual pixel. Regions are then homogeneous in the sense that pixels within an region all share the same local minimum.

The most basic approach to finding a segmentation as such is to find a downstream path from each pixel to its local minimum based on the local gradients. This method is however computationally expensive and susceptible to inaccuracies; an approach presented by Vincent and Soille [48] makes the idea attractive. Rather than starting from each individual pixel to find minima, the first step is to sort all pixel values by ascending grey values. A flooding step is then carried out using a fast breadth-first search of all the pixels in this order so as to attribute them to their local minimum. Conceptually this fills the basins from the bottom up, causing new pixels to reach more recently added pixels first instead of having to descend all the way to the local minimum.

Superpixels & SLIC

Superpixels are a computationally efficient representation of the pixels in an image which take into account and have a better fit to the objects and structures within the image [9]. Pixels are grouped into larger regions based on similarity of colour or texture, and savings come from calculating features for just superpixels and not all the pixels it contains. Little global information and fine detail is lost by moving to this representation as boundaries of structures of interest most often correspond to large changes in the colour or texture feature with which the superpixels were formed. The superpixel representation can come close to an accurate segmentation, but is most frequently used as a preprocessing step for a different segmentation algorithm.

One technique for finding a fast superpixel segmentation is **simple linear iterative clustering** [12], or SLIC. Its performance has been shown to be better than many other techniques for generating superpixels, and has an advantage of only requiring one parameter, the target number of superpixels in the output.

The approach taken by Achanta et al. takes into account the colour similarity of pixels as well as their proximity in the original image. From this they define a 5D space and a novel distance measure within this space that also takes into account superpixel size; this allows control over cluster sizes. This distance measure also works to limit the maximum proximity of pixels that is considered by the algorithm. Rather than compare all pairs of pixels in the image, SLIC only looks for pixels within an area relative to the current superpixel size. This is one reason that it is amongst the fastest superpixel segmentation algorithms.

The iterations are initialised by placing regularly spaced seeds onto the original pixel map and moving them to the lowest gradient position in a 3x3 neighbourhood. This avoids placing seeds at an image edge and reduces the chances of the algorithm being affected by noise. The first step is to associate all pixels with their nearest seed point, that is the seed that minimises the novel distance measure and resides within the search proximity of the pixel. Iterations then recalculate the centre point of each superpixel based upon the average of their pixel's 5-dimensional vectors, and all pixels are re-associated with their closest centre point.

The algorithm produces a regularly sized and equally spaced segmentation of the original image into superpixels at a low computational cost. These are desirable qualities when using the algorithm as a preprocessing step before the application of a more specific segmentation algorithm.

3.2.4 Template Matching

Template matching is the simple process of overlaying templates on an image to find all the occurrences of specific objects or patterns represented by the template within the image. The best match for a given template on the image is given by some measure of optimality, based on the properties and relationships between the object template and the image. One simple example of a measure would be the proportion of pixels that match between the template and the section of the image onto which the template is placed.

Whilst a simple and widely used method, the applicability of template matching to image segmentation is limited due to requiring the shape of the objects to segment be known in advance. Its effectiveness is also highly dependent on the measure of optimality; higher level image descriptors, such as correlation, can be applied to increase this.

3.2.5 Active Contours (Snakes)

An active contour is a set of points which aims to enclose a target feature [36]. A good analogy also taken from [36] is that of a balloon placed around a shape so as to enclose it fully. If enough air is taken out of the balloon it will surround the object that it encloses; active contours aim to describe a shape by enclosing it in this manner.

A snake is defined as an energy-minimising spline, and its energy is defined by its shape, location in the image and certain other image properties. The minimisation task is to find local minima of energy which then correspond to desired image features. The snake is initialised at some position in the image and is subsequently deformed to match the nearest contour.

The definition of energy depending on the shape of the snake limits its applicability to recognising shapes that are intrinsically smooth in their shape. Shapes that have sharp changes of direction in their border may struggle to be matched exactly by active contours. Further limitations upon general purpose use arise from requiring user input to position the initial snakes; active contours are a deformable model of the shape, so some segmentation is already done by a user in order for it to be refined using this method. The outcome of the model will therefore also be sensitive to how it was initialised by the user.

3.2.6 Graph Partitioning Methods

One way to represent an image is that of a weighted undirected graph, where the nodes are pixels or voxels and edge weights correspond to the similarity (or dissimilarity) between two nodes. The graph can then be partitioned using standard methods, and resulting partitions are considered as segments of the original image. Graphs of this nature in the image segmentation domain are sometimes termed *affinity graphs*, with the edge weights called *affinities* [25].

One particular advantage to this representation is the ease with which relationships between distant pixels or voxels can be expressed. The segmentation is no longer limited to examining just the neighbourhood of pixels as is the case in a boundary labelling or edge detection method. Instead, affinities between any arbitrary pixel or voxel in the original image can be expressed in a graph representation. The result is the ability to define multiple segments of objects that are disconnected in the image.

Once a graph is defined over the whole image space, the idea is to partition it into sets of pixels, where the similarity between pixels in partitions is high but the similarity between partitions is low [41]. What sets apart different algorithms for partitioning graphs is their criteria for a good partition and the ability to find efficient algorithms to evaluate these criteria.

Max-flow Min-cut

A **cut** is defined as a partition of a graph into two disjoint subsets of vertices. The measure of dissimilarity between the two partitions is the sum of the weights of edges that had to be removed to create the partition. The optimal partitioning, in terms of image segmentation, is therefore the cut which minimises the value of this sum.

Finding this optimal cut can however be mapped to the problem of finding the maximum flow in a graph [42]. Given source (s) and sink (t) nodes in the graph, we can define the capacities of each edge as equal to their weights. The maximum flow from s to t will then give the minimum cut.

Minimum Spanning Tree

Another method to find a graph partitioning is based upon Kruskal's greedy algorithm for finding a minimum spanning tree of any given graph, as presented by Felzenszwalb [20]. The set of edges is sorted and evaluated in increasing order of weight given by a predefined similarity measure. An edge is added to the final partitioning if it does not create a cycle and is similar to those pixels that are connected to its edge points. The result is a series of disjoint minimum spanning trees, each of which defines a segment of the original image.

Normalised Cuts

The normalised cut is another criterion for measuring the quality of an image segmentation as a graph partitioning. Shi and Malik [41] explain that using the standard minimum cut as the partitioning criteria lends itself to frequently segmenting small sets of nodes that become isolated in the resulting partitioning. This is not ideal for the application of graph partitioning to image segmentation.

Their idea is to normalise the cut function using the total edge connections to all nodes in the graph. The standard cut function is weighted according to the relative totals of the edge weights from within each of the two partitions to the complete graph.

The resulting partitioning algorithm then no longer favours small isolated sets of nodes, and performs well for image segmentation applications. Whilst the minimising of the normalised cut is an NP-complete problem, Shi and Malik [41] also show that an approximate discrete solution can be found efficiently.

3.2.7 Summary

We see here that there is a lot of choice when it comes to algorithms to apply to segmentation problems. Many require user input or careful choice of input parameters, and these factors greatly influence the accuracy of the resulting segmentation. These traditional algorithms are well suited to simple segmentation problems.

With increasingly complex images being subjected to segmentation, particularly with medical applications, more advanced techniques are required. A recent drive in algorithm design involves machine learning techniques to effectively learn parameters of some of the previous algorithms. This can also be extended to completely learning an algorithm based on a hand crafted set of features, or even to have a complete end-to-end learning scheme, where both the features and algorithm are learnt.

This results in a spectrum across which the degree of applied machine learning techniques can be varied. After some further background, they are introduced in Section 3.5.

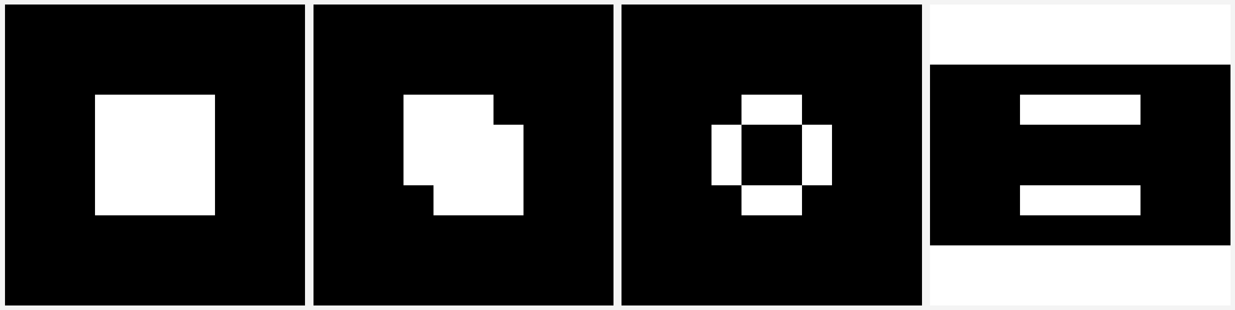
3.3 Segmentation Quality Metrics

Metrics are an important and useful tool for evaluating the quality of segmentations. As well as being used for comparing the performance of different segmentation algorithms, they are also particularly useful as the objective function for machine-learning classifiers as they act as the function or value to minimise during the learning process, as explained in Section 3.5.

Most commonly quality metrics are used to compare a pair of segmentations, one coming from a human and one from a computer as a result of applying a segmentation algorithm. If the human segmentation, or ground truth, is regarded as correct then the metric measures the error in the computed segmentation.

3.3.1 Pixel Error

The most simple errors to consider are those at an individual pixel level. Given a human and computer segmentation, the pixel error is simply the proportion of pixel positions that disagree on their labelling.



(a) Ground truth labelling, a 4x4 square in a 10x10 image **(b)** Pixel error = 0.002 **(c)** Pixel error = 0.008 **(d)** Pixel error = 0.48

Figure 3.14: Evaluation of the pixel error quality metric on some sample segmentations. The error in (d) is high, showing that all object and background pixels are taken into account.

The problem with such a naive metric [25, 24] is that it does not consider the topological result of the segmentation, the defined segments and how they are connected to each other. Two different segmentations can achieve the same pixel error, but may have very different topological results which also may or may not match the topography of the human segmentation.

Ideally, more appropriate metrics should be less sensitive to small changes in boundary positions and penalise heavily differences in topography. This being said, even the use of the naive pixel error achieved better segmentation results as an objective function in a classification technique than standard techniques such as Canny edge detection.

3.3.2 Rand Error

The Rand error is one such metric with the desired properties of penalising topological disagreements more than slight differences in boundary location [38]. It was originally designed as a method for evaluating similarity between two data clusterings but has recently been used to evaluate segmentations, as a segmentation can be viewed as a data clustering.

The metric is defined as the count of the number of distinct pairs of pixels that are in the same segment in one segmentation and not in another, divided by the number of such pairs. It is therefore the frequency with which the two segmentations disagree between pairs of pixels. When both completely agree the Rand error is 0, and when both completely disagree the Rand error is 1.

3.3.3 Jaccard Index

The Jaccard index is a fairly new metric that has only recently been used for evaluating segmentation quality. It is defined as the ratio of the areas of the intersection of the ground truth and the segmentation and their union, considering only object pixels. It is written as [31]:

$$JaccardIndex = \frac{TruePos}{TruePos + FalsePos + FalseNeg}$$

Where *True Pos* is the area of intersection of the ground truth and the segmentation, the area within which the segmentation was correct. *False Pos* is the area that the segmentation identified as belonging to an object that it did not belong to in the ground truth, and *False Neg* is area belonging to an object that the segmentation failed to identify. Therefore, *True Pos* + *False Pos* + *False Neg* is union of the ground truth and the segmentation.

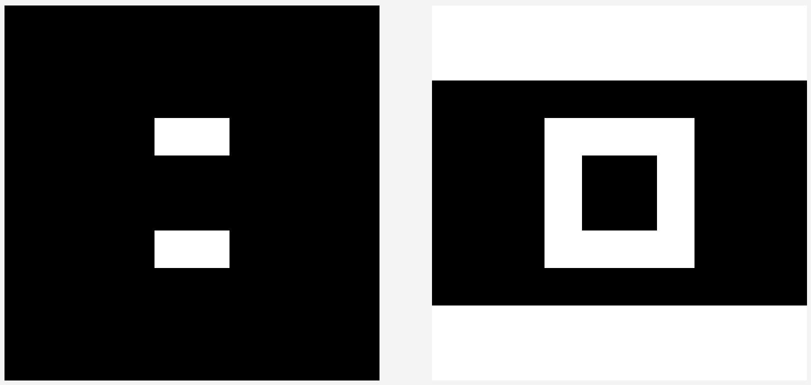

(a) Intersection **(b)** Union

Figure 3.15: Taking Figure 3.14d as the ground truth and evaluating the segmentation quality of Figure 3.14c using the Jaccard Index. The number of object pixels in the intersection is 4 and in the union is 52, giving a segmentation quality of $\frac{4}{52} = 0.08$ with the Jaccard Index.

3.3.4 Warping Error

The warping error [24] is based upon a warping of the ground truth segmentation to attempt to match it to the computed segmentation. This is done by flipping pixel labels incrementally, subject to a couple of constraints, to bring it closer to the computed segmentation. First, the flipping of a pixel must not alter the topology of the ground truth labelling. Second, boundaries that are shifted due to the flipping of pixels cannot shift more than a certain distance overall.

Due to these constraints, there will be a time when no more pixels are eligible to be flipped. The pixel error is then calculated between the best warped image achieved in the previous steps and the computed segmentation, and this is the warping error.

3.4 Feature Extraction

Many of the algorithms that employ machine learning techniques are highly dependent on the feature set (**feature vector**) that is used to describe the image. Most commonly, features are extracted for each individual pixel based upon characteristics of the pixel itself and those of its nearest neighbours. The size of the neighbourhood from which features are extracted can also impact the values for different features, hence some approaches tend to vary the neighbourhood size to gain complete feature coverage.

The set of features chosen to describe an image is another way of introducing domain knowledge; as explained previously, designing complete algorithms by hand requires a certain amount of intuition from the designer. Applying machine learning to find the optimal algorithm for segmentation can reduce this reliance upon domain knowledge and human intuition. Allowing features to be designed can make the search for an optimal algorithm simpler and quicker, and the result may generalise to more novel input. That being said however, hand designed features have the potential to throw away important information, or even to bias the outcome.

Features can generally be categorised in three groups: colour, texture and shape.

3.4.1 Colour

The colour of an individual pixel can be broken down into a particular hue and intensity, both of which can be used as features. For a greyscale image, the only feature is the intensity of the individual pixel. In standard thresholding, colour is the only feature used to create a segmentation.

When considering a pixel's neighbourhood it is possible to provide a more statistical representation of the colour values. For example the standard deviation, minimum, maximum, median and the first and third quartiles [13] of the intensity value over an arbitrarily sized neighbourhood could be calculated for use in a feature vector.

3.4.2 Texture

Texture is important in segmentation applications as different objects in an image may well have different textures, for example a wooden table with a cloth on top, and they may help greatly in computing the segmentation. At the image processing level, texture can be described by the spatial arrangement of different colours or grey values. Texture features aim to characterise and provide quantifiable information about the arrangement of values found in an image to be segmented.

A simple feature for a particular region is its histogram, a count of the frequency of greyscale values within the region. We can normalise the histogram for varying sizes of region, and thus we can compare the histograms of different regions by their greyscale distributions. Using a measure of distance between histograms, this technique will identify regions that potentially have the same texture, but we cannot be sure that they do not have different spatial arrangements of grey values. Therefore, when describing texture it is important to capture both spatial arrangement and relative brightness values.

Co-occurrence Matrices

Co-occurrence matrices measure the likelihood that two grey values appear in a certain relative position by identifying repeated spatial patterns [36, 37]. The size of the matrices is the number of grey values in the image, and as such the original image values are often quantised into larger groups in order to keep the size and computation time low. Each co-occurrence matrix counts the number of times a pair of grey values occur in the image at a particular pixel distance and orientation. Each entry in the matrix corresponds to the count of the grey values given by its row and column indices, for example the entry at row 2 and column 5 represents the number of times that a pixel of intensity 2 has a neighbour at the chosen pixel distance and orientation with an intensity of 5.

It is common to use several different co-occurrence matrices, utilising all 4 pixel neighbour directions and varying the pixel distances through a sensible range. The end result is a good description of textures within a window or the whole image, however the resulting matrices are rather large. The accepted approach is therefore to compute a set of characteristics of the co-occurrence matrices. Essentially, a co-occurrence matrix represents a joint probability distribution, and we can partly characterise these with less coarse statistics.

To gather these statistics, the first step is to normalise each matrix such that it truly defines a joint probability distribution, $p(m, n)$. The following statistics can then be computed [37, 53, 45]:

Maximum	$\max_{m,n} p(m, n)$
Energy	$\sum_{m,n} p(m, n)^2$
Entropy	$\sum_{m,n} p(m, n) log(p(m, n))$
Homogeneity	$\sum_{m,n} \frac{p(m, n)}{1 + \lvert m - n \rvert}$
Correlation	$\frac{\sum_{m,n} mnp(m, n) - \mu_x \mu_y}{\sigma_x \sigma_y}$ where μ_x, μ_y are the marginalised means of m and n respectively, σ_x, σ_y are the marginalised standard deviations of m and n respectively.

The co-occurrence matrix itself is a very powerful descriptor of texture, however it is limited in its applicability because of the amount of information it creates. Finding statistics based upon the matrices is a good compromise, however some useful information can be thrown away to create them.

Structure Tensor

A commonly used feature for describing local texture is the structure tensor. This typically represents gradient information in the neighbourhood of a pixel, providing information about edges and oriented texture within that neighbourhood. Features such as the eigenvalues can be used to describe the structure tensor.

Hessian

The Hessian is the matrix of second-order derivatives of the image, which can be computed with a convolution filter such as the Laplacian operator. For use in feature vectors the eigenvalues of the matrix are often computed, however a variant to use is the determinant of the Hessian.

Laplacian of Gaussian (LoG)

The Laplacian operator, or convolution filter, gives the second derivative of an image. If the image is not smoothed before the convolution, then the resulting matrix is the Hessian that can be used directly to generate features. Another method (the Laplacian of Gaussian) [45] first applies a Gaussian smoothing to the image in an attempt to remove noise before convolution with the Laplacian. Features such as the eigenvalues can be used to describe the resulting matrix.

Difference of Gaussians (DoG)

The difference of Gaussians feature [45] takes the difference of two convolutions of the original image, each a Gaussian with a small variation in the standard deviation. Features such as the eigenvalues can be used to describe the resulting matrix.

3.4.3 Shape

Histograms of Oriented Gradients (HOG)

This feature, designed by Dalal and Triggs [18], uses the intuition that the distribution of edge directions in an image, or smaller parts of it, can be used to describe the shape of objects within the regions. The first step in its generation is to calculate edge gradients across the image with a convolution mask such as the Sobel or Prewitt operator. The orientations of the gradients are then binned into either 180 or 360 bins, depending on whether the sign of the gradient is taken into account. Additionally, each pixel's contribution to the descriptor is weighted by the magnitude of the gradient.

In similar ways to previous features, either the entire histogram vector can become part of the feature vector, or some statistics can be calculated from it in order to reduce its dimensionality.

Ray

Ray descriptors are a set of shape features introduced by Smith et al. [43] aimed at detecting highly deformable objects, in contrast to HOG which relies on regular shape cues at precise locations. They exploit the relative locations of objects, rather than characteristic features of the objects themselves.

The descriptor set contains four different features, all of which depend on a function which returns the location of the closest contour point to a location at a given angle. The idea is simply to measure properties of this closest contour point at a given number of evenly spaced angles in order to characterise a complete environment of any given pixel.

The first descriptor, distance, simply takes the distance from the pixel in the image to its nearest contour point. The orientation feature takes the direction of the gradient at the nearest contour point, relative to the given angle. The norm feature is the gradient strength at the nearest contour point. Finally, the most dominant feature is the distance difference that compares relative distances from a pixel location to its nearest contour points in two different directions.

3.5 Machine Learning Techniques

The aforementioned segmentation algorithms are all hand designed by a community of image processing experts over many years, with gradual improvements and many variants appropriate for different applications. Utilising the right algorithm for a particular purpose requires a good understanding of the nature of the problem. Designing an algorithm in the case that an existing one does not quite fit the bill also requires a deep understanding of why existing algorithms do not work and an intuition into a new algorithm fit for purpose.

Recently, machine learning techniques have been applied to search for new, better algorithms, and have achieved superior performance in general benchmarks. Jain et al. [25] explain that there has been some resistance in the past from researchers that prefer to design algorithms based on their intuitive understanding of image segmentation. The reason that machine learning techniques have achieved superior performance is potentially that our intuitions about image segmentation are incorrect or incomplete. Attempting to understand how our brains segment images is a complex task; we are conscious of the results of the segmentation, but unable to understand exactly the processes that lead our brains to that conclusion. Essentially, hand designed algorithms are susceptible to human intuition error and an incomplete understanding of natural segmentation by our brains.

Machine learning techniques define a space of algorithms that transform images into segmentations and learn the algorithm that optimises a particular measure of the quality of a segmentation, the objective function. A prerequisite of machine learning techniques is the availability of training data; examples of likely images sometimes with their labelled segmentations. For complex applications such as those in medicine, segmentation is carried out by more than one human expert in order to verify segmentations. This example data is the **training set**, and is used to train a classifier which will apply a label to each pixel in an image to achieve a segmentation. If a labelled segmentation is also available for every training image then the classifier can utilise this ground truth during training. This is termed a **supervised** learning approach; training a classifier without labels is therefore an **unsupervised** approach.

An example dataset for general segmentation applications is the Berkley Segmentation Dataset [33], which has been widely used in the development of modern segmentation algorithms.

3.5.1 Classifiers

One of the most popular uses of machine learning techniques is that of classification [27]. Given objects that have a feature vector describing their characteristics, classifiers attempt to associate each object with a particular class of objects. To determine the class, the classifier must learn a function that maps from features and attributes to classes; this can be designed by hand but is most often learnt using training data. The classifier can use this data to subsequently generalise to further new data, but care must be taken not to overfit the algorithm to the training data.

Boosting

Boosting is a popular algorithm for this type of learning. It makes full use of the concept that a number of weak learners used in an ensemble can create a strong learner. A weak learner is defined as "an algorithm that outputs a hypothesis that has some advantage over random guessing" [27]. Boosting is most often associated with a weak learner called a stump, a decision tree consisting of just one level which makes a decision based purely upon one feature.

A strong learner is an algorithm that generates classifications that are in general closer to the ground truth. The boosting algorithm uses many weak learners to form a classifier. The characteristic of the boosting algorithm is that incorrectly classified training examples are given higher weight than correct classifications in subsequent iterations of the algorithm. The weights of the different weak learners are also adjusted according to their individual classification accuracy. The iteration continues until the mis-classification rate is below some error threshold.

Support Vector Machines

Support vector machines (SVMs) are very popular and successful classification algorithms used extensively in current research [27]. They are especially good at dealing with large feature spaces and many training examples, common properties of segmentation problems.

The algorithm attempts to find the optimal class separating hyperplane in either the original attribute space, or in the space of transformed attributes. For a simple two-dimensional attribute space a separating hyperplane is a line or curve; in 3D it is a plane or surface. Transformed attributes are often linear combinations of other attributes, and extending the attribute space can be required if the original feature space does not discriminate between the classes such that a good separation can be found. Examples from the learning set that are closest to the separating hyperplane are called the support vectors, and the distance from the hyperplane to the support vectors is called the margin. The hyperplane that maximises the margin is the optimal hyperplane.

Finding the optimal hyperplane for separation is a quadratic optimisation problem that can be solved efficiently using existing algorithms. Once it is found and confirmed to correctly classify all training examples, it can be used for classification. Traditional SVMs as discussed here are only applicable to separating two classes. If there are more classes to be used then the problem is divided into sub-problems where one class is compared to all other classes.

Random/Decision Forests

Decision forests are ensembles of decision trees that create a strong learner as a classification technique [17]. A decision tree has a number of internal split nodes which test values based on features of the training data. At the leaves of the decision tree, after any number of split nodes, is a probability distribution across the possible labels.

Decision forests use many decision trees, typically more than 100, and the probability distributions from all trees are averaged out to give a probability map for each label from the overall forest. The averaging of all posterior probabilities has an advantage in that it is heavily influenced by the more confident and informative trees [17]. Noisy tree contributions are also suppressed.

Randomness can be introduced into the forest to good effect by reducing correlation between individual trees, thus improving generalisation. For each split node in a tree, a random subset of all parameters is made available according to some globally defined parameter, ρ. When $\rho = 1$ there is maximum randomness as only one feature is available at each node. For $\rho = numberoffeatures$ then the full feature set is available at each node, and there is no randomness in the forest.

3.5.2 Learnt Algorithm Design

Boosted Edge Learning (BEL)

Work from Dollár et al. [19] introduces a supervised learning algorithm for edge detection called boosted edge learning, BEL. This is one of the first techniques to be adopted by the segmentation community that used a machine learning element to develop an algorithm. It is based on the understanding that designing edge detectors to be general is often a hard task as the applications can be varied. It is often simple to obtain training images that define the application and correct output, so a general classification approach can be created instead.

The major focus is to learn probability distributions over image patches centred at specific pixels. If this patch is large enough then both low level and contextual information will be gathered as features. BEL again varies from other previous approaches by computing many varied simple features over a large region in order to shift the bulk of the work onto the classification algorithm. General features also allow the algorithm to be applied in many different domains.

BEL primarily uses an adaptation of the common boosting algorithm, the probabilistic boosting tree, which is similar to a decision tree except a boosted classifier (with a limited number of weak learners) splits the data at each node. The training step is recursive; at each node in the tree a boosted classifier is trained, the data is split into two sets and the left and right children are trained recursively with one of the two sets each. The algorithm is applied to a number of different datasets and shown to produce good results.

Boundary Learning by Optimisation with Topological Constraints (BLOTC)

Jain et al. [24] show that using the warping error not only as a way of evaluating segmentation quality, but also as the objective function in a supervised learning scenario, produces far greater results than basing the learning task on the pixel error metric. This research also introduced the warping error as an error metric for the first time. Using the warping error yields a couple of advantages over the pixel error:

- Minor differences in boundary location are tolerated.
- Topological disagreements give larger errors.

Both pixel error and warping error can be used directly as the objective function in a machine learning scenario, and this work highlights that the choice of objective function can greatly affect the success of the segmentation. The output of their algorithm is also shown to improve upon the state of the art by comparing current methods with BLOTC and measuring segmentation quality with both the warping and Rand errors.

Maximin Affinity Learning of Image Segmentation (MALIS)

The approach presented by Turaga et al. [47] trains a classifier to produce good affinity graphs by using the Rand index as the objective function. This graph is then simply thresholded by removing all edges with affinity less than some value; high affinity values suggest that two pixels are similar. Connected components of the thresholded graph are found to produce the resulting segmentation. As an extension to using just the Rand index, their objective function also relates to a connectivity indicator between pixels that takes into account maximin affinity.

The maximin path and edge are analogous to the minimax concepts that are well known in graph theory. Therefore, the maximin path from a set of paths is the path that maximises the minimum weight (affinity) of any of the edges along that path; this edge is also called the maximin edge and its weight is the path's maximin affinity. These maximin properties can be computed efficiently using minimum spanning tree algorithms.

The algorithm aims to improve classifier output at maximin edges because incorrectly classifying these edges leads to segmentation errors such as splits and mergers. Edges that are not maximin are often internal to a segment, rather than at a boundary between segments, and so incorrect affinity classifications are not so important at these places.

Their results show that integrating the maximin affinity into the objective function in this way leads to a good improvement in segmentation performance, even over a standard classifier that uses the Rand index as its objective function but does not take into account maximin affinities.

Learning to Agglomerate Superpixel Hierarchies (LASH)

Jain et al. present a supervised agglomerative clustering algorithm called LASH [26]. Their approach is to take an over-segmentation of supervoxels produced by some other algorithm, for example SLIC, and instead of hand crafting a function to agglomerate those supervoxels that represent the same object, the similarity function is learnt through a reinforcement learning technique.

They model their reinforcement learning approach as a Markov decision process, for which a set of states and actions for each state is required along with a policy that maps states to actions. In this case, states are clusterings of objects (supervoxels) and actions are defined as merging a pair of clusters. The goal of this technique is to find a policy that maximises the possible total reward. In this application, the reward is defined as the increase in Rand index (defined as $1-\textit{Rand error}$) calculated between the current and potential next states, as compared to the ground truth clustering.

The optimal policy for any state is given as the action that maximises the optimal action-value function, the sum of the rewards of taking an action in a state and subsequently following an optimal policy. In the case of LASH [26], the optimal action-value function is the reward function, the increase in Rand index. This is known exactly for training data, however the idea behind LASH is to train a function approximator to the reward function and hope that it generalises well to the test data. If the action-value is exactly the reward function, then agglomerative clustering is equivalent to greedy maximisation of the Rand index, and therefore the resultant clustering would be a global maximum.

LASH performs particularly well compared to techniques such as BLOTC and MALIS, mainly due to the increased window within which its features are extracted. Whilst this may seem unfair to use a larger window that other techniques, the training time of LASH compared to BLOTC and MALIS is significantly smaller, so there is a trade off. LASH is more accurate because its efficiency allows more image context in its calculations.

3.5.3 End-to-end Learning

Another option to fully exploit machine learning techniques is that of end-to-end learning; the transformation of the input image into the output segmentation is learnt in its entirety. This is a convenient technique to use if individual knowledge of a particular domain is limited, as it does not require any hand designing of features or algorithms. End-to-end learning has the drawbacks of lengthy training time and requiring more training examples than other techniques, however the classification speed once trained is often better than traditional or trained algorithms. Computing power has reached a sufficient stage to also make this approach plausible, especially by exploiting fast GPU implementations.

Convolutional Networks

The most popular end-to-end learning technique is a convolutional network [25]. It is closely related to a neural network as it consists of an input, several outputs and a number of hidden layers. Each of the layers performs a series of convolutions and pixel-wise transformations, which typically results in tens of thousands of free parameters for the machine learning algorithm to learn. These free parameters effectively define the convolutions and transformations in the layers, and the result is a specific set of convolutions and transformations for any particular training that maps an input image to a segmentation.

Convolutional networks have been shown to emulate some hand designed algorithms through a particular combination of convolutional filters. Studies have also shown that this large number of free parameters means algorithms designed by CNs have outperformed those designed by hand.

3.6 Summary

This chapter gives an overall flavour of how the field of segmentation has grown in recent times. Current research often involves combining several of the explained techniques to produce a better segmentation than an individual algorithm. Many more different individual techniques exist for segmentation, but the particular ones I have highlighted here are those that have been used in related research or within this project.

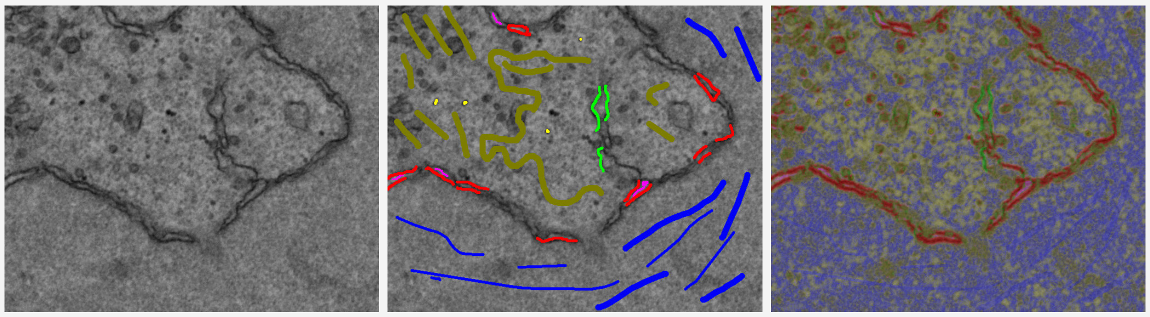
4 Background: State of The Art

This chapter takes a look at the current state of the art as a small literature review. Most of the techniques examined pull from more than one of the areas discussed in the previous background chapters.

4.1 Ilastik: A State of the Art Segmentation Tool

Ilastik [44] is an easy to use semi-automatic general purpose segmentation tool driven by user labelling of sample images to train a random forest classifier. A generic set of features is derived from each pixel's neighbourhood, including colour, edge, texture and orientation information. During the training phase, the user is allowed to refine the classifier by providing new labels on the training data in real time. The trained classifier can subsequently be saved and used as a batch process at a later stage.

For an example of the input and output of Ilastik, a small section of an example EM image has been labelled and classified by Ilastik. The sample, its labelling and the resulting segmentation are shown in Figure 4.1.



(a) Original sample **(b)** Resulting segmentation **(c)** Segmentation of a small sample image using Ilastik

Figure 4.1: The training process in Ilastik requires painting over samples of the different classes to be segmented.

A particularly positive outcome from this example is that the nuclear envelope, labelled in red, seems to be the most consistently segmented feature, however other objects outside the NE are incorrectly classified as such. I suspected that Ilastik would have problems distinguishing the endoplasmic reticulum from the nuclear envelope as is the case in this example; the characteristics of these structures are very similar. Another fairly good result is the general difference in segmentations of the nucleus, labelled in blue, and the cytoplasm, labelled in brown. However, there is still a large amount of uncertainty in these regions.

Ilastik in general achieves good results on standard EM segmentation problems [44].

4.2 Sub-cellular Segmentation

This section is an introduction to recent work on segmenting sub-cellular structures in EM images. Whilst this work may seem to be similar to the task on this project, there are critical differences that will be subsequently highlighted. The work makes some important points that really underpin why there is a focus upon specific applications of segmentation to EM images.

4.2.1 Supervoxel-Based Segmentation of Mitochondria in EM Image Stacks with Learned Shape Features

Lucchi et al. [32] begin by raising some good general points in their introduction:

- This application has a good rationale for automated segmentation; it would take months for an image stack to be analysed by hand, and potential would creep in for some of the useful data to go to waste.
- The modern imaging techniques of SEM allow for nearly isotropic 3D datasets that contain vast quantities of useful data. Many of the generic algorithms, including those introduced previously, would perform poorly on this EM data purely because of the sheer quantity of it.
- Generic algorithms also fail to take into account strong shape cues, and rely only on local statistics that can be poor discriminants with the amount of noise and texture inherent in a high detailed EM image.

Their approach to segmentation is split into three main parts. First, the original image pixels are grouped into supervoxels using the SLIC algorithm. Features are then extracted from the supervoxels, including ray descriptors as described in Section 3.4.3. Finally, the graph of supervoxels is cut using a graph-partitioning algorithm controlled by classifiers that learn the boundary appearance of the segmentation.

Superpixels/supervoxels are fast becoming standard as their popularity increases, primarily due to the decreases in computation time that they yield. The aim of the first step here is to produce an over-segmentation which still respects the boundaries within the image. Computation on the resulting graph that represents the supervoxel structure will be much more efficient than using the original image's voxels directly.

The ray descriptors then capture information about the shape of the mitochondria; the presented results compare the segmentations achieved with and without these shape cues and show they are important for this application. Given that these shape cues are learnt from a general model, this could have high applicability to segmenting other sub-cellular structures if they have a well prescribed shape, although it could only be applied to a single structure at a time.

In addition to the shape cues, instead of defining graph cuts at edges with high colour or intensity changes, the characteristics that indicate a true object boundary are learnt from training data. The graph partitioning algorithm that defines the final segmentation takes into account both the shape cues and the learnt object boundary characteristics.

Lucchi et al. show that the use of supervoxels over voxels results in a computationally efficient algorithm, whilst the combination of ray descriptors and learning object boundaries increases the segmentation quality, and brings it close to the quality of human segmentations. Their approach has a lot of applicability to segmenting other cellular structures, however mitochondria typically have a less complex shape than the nuclear envelope and so the applicability of this work to my project is limited.

4.3 Neuronal Segmentation

This section takes a look at the application of segmentation to EM images of the brain, with the aim of automatically segmenting neurons and their connections (synapses). This work is part of a bigger picture to find a human connectome, a map of all the neurons and connections in the brain. Such a task was previously infeasible but with modern computing power and the right techniques, recent work has made significant progress towards providing a method for fully automatic segmentation of neural structures.

Whilst targeting slightly different segmentation interests from this project, there are still many concepts that are applicable to other complex segmentation problems. This work is of particular interest, as there is a great focus on being able to accurately segment neurons and synapses. When one considers that the brain has somewhere in the region of 100 billion neurons and about 1000 times more synapses, finding all the connections accurately is of utmost importance.

4.3.1 Automated Segmentation of Synapses in 3D EM Data

A technical report by Kreshuk et al. [28] details an extension of the popular program Ilastik aimed at semi-automated segmentation of the synapses in FIB/SEM images. Their approach involves training a random forest classifier based on a labelled training set, as produced by the user through Ilastik's GUI. Users are able to select the most descriptive features for the classifier to use, again through an interactive GUI.

The classifier output is a probability map across classes; the probability map for the synapse class is adaptively thresholded with a user defined threshold and those pixels with a high enough probability of being a synapse are joined together to form candidate synapses. An additional post-processing step encompasses further domain knowledge by eliminating all candidate synapses that are too big or small when compared to the largest and smallest labelled synapses in the training data.

Their results show a significant improvement in the quality of results compared to other more general techniques. Almost all the human interaction is incorporated into the training phase, where the user defines the thresholds for segmenting the probability map, choosing the most descriptive features and labels the training set. The idea is that a small volume of the dataset is used for training, with the trained algorithm performing automatic segmentation on the rest of the volume.

Of concern with this approach is its reliance upon user input, which I believe to be one of the major downsides of Ilastik as a whole. Giving the user control over the features to be examined brings back the human error element that machine learning can eliminate through determining the best features for the job itself.

This being said, it is obvious that their approach to training with user guidance does produce good results. The incorporation of some domain specific knowledge by eliminating candidate synapses that are highly unlikely to be synapses is a particularly novel yet intuitive step.

4.3.2 Segmentation of SBFSEM Volume Data of Neural Tissue by Hierarchical Classification

In the paper presented by Andres et al. [13], a hierarchical approach is used to segment EM images of neural tissue to identify neurons. Their approach is hierarchical with two random forest classifiers used, the first with the original voxels and the second with an intermediate supervoxel segmentation. The first classifier takes the original voxels and outputs a probability map based upon features calculated from each voxel's neighbourhood. The probability map is then used as the input to a watershed algorithm to produce a segmentation into supervoxels.

The key idea behind using the watershed algorithm is to obtain an over-segmentation of the image that requires only supervoxel merges to produce the final segmentation. The second random forest classifier is used to learn which supervoxels should be merged as a result of the over-segmentation. This includes defining some rudimentary features about the candidate supervoxels, such as the number of voxels and the difference in this number between the two supervoxel candidates for merging. Further standard statistical features were derived from the face defined by the two adjacent supervoxels, the area over which two supervoxels meet.

The application of hierarchical classification has once again proven to generate results that advance the state of the art. This work was carried out before the SLIC [12] algorithm was published, and so it would be interesting to see if using SLIC over the watershed segmentation would prove beneficial both in terms of computation speed and accuracy of segmentation. There are also similarities to be drawn between this work and the segmentation of mitochondria in [32].

5 Background: Dataset

The dataset provided by Cancer Research UK and manually segmented by Dr. Christopher Peddie consists of 80 slices through a control cell obtained using transmission electron microscopy. The images are 8-bit greyscale and have dimensions 3625x2660 pixels and real world pixel dimensions of 3.283nm by 3.283nm. The physical distance between each slice is approximately 70nm. The cell is at late telophase (Section 2.2), which results in a nuclear envelope that is almost completely formed.

The imaging process involves taking ultra-thin slices of a cell that has been placed on a regular resin grid. Taking these serial slices of the cell sometimes causes folds and compression of the sample below the slice being taken, requiring some manual manipulation to stretch the sample back to its original shape. The fact that each slice is imaged separately, rather than the cell being imaged in situ as in SEM, and the slight compression and stretching of the sample means that registration of the captured images is required.

Registration is another primarily manual process; the images are too complex for automatic registration algorithms to work effectively. If the transformation between two slices is minimal then these algorithms work, but their use is restricted primarily to SEM where there are commonly only small transformations. For a typical dataset the registration step takes an hour or two.

A further manual processing step is equalisation of the overall brightness of the images. Images that have noise present, as discussed later in Section 5.4, tend to have lower average intensity, low contrast and the image appears dull compared to unaffected slices. This step is quite subjective and down to the individual that is performing the segmentation; the goal is to have the same structures have more or less the same intensity values throughout the whole dataset. It takes roughly half a day to perform this step manually.

Once the manual registration and brightness equalisation steps have been performed the segmentation can be performed. This step is very simple but the most time consuming part of the manual process. For each image in the stack, the best tool for the job is a pen to draw the outline of the subject region which is then filled to give a part of the segmentation. Other area selection tools exist but their use is minimal in this application. To segment just the nuclear envelope for an averaged size dataset takes 2 or 3 weeks of full time work. If all other sub-cellular structures are segmented then this can add an extra 2 or 3 weeks. The large amount of work those that carry out the segmentation have to do often means the effective throughput time for a single dataset is much larger than this estimate or 6 or so weeks.

5.1 Sample Slices and Manual Segmentations

A stack of four adjacent sample slices is shown in Figure 5.1 along with their manual segmentations. The important thing to notice here is the gradual but minimal change of shape in the nuclear envelope and nucleus between neighbouring slices.

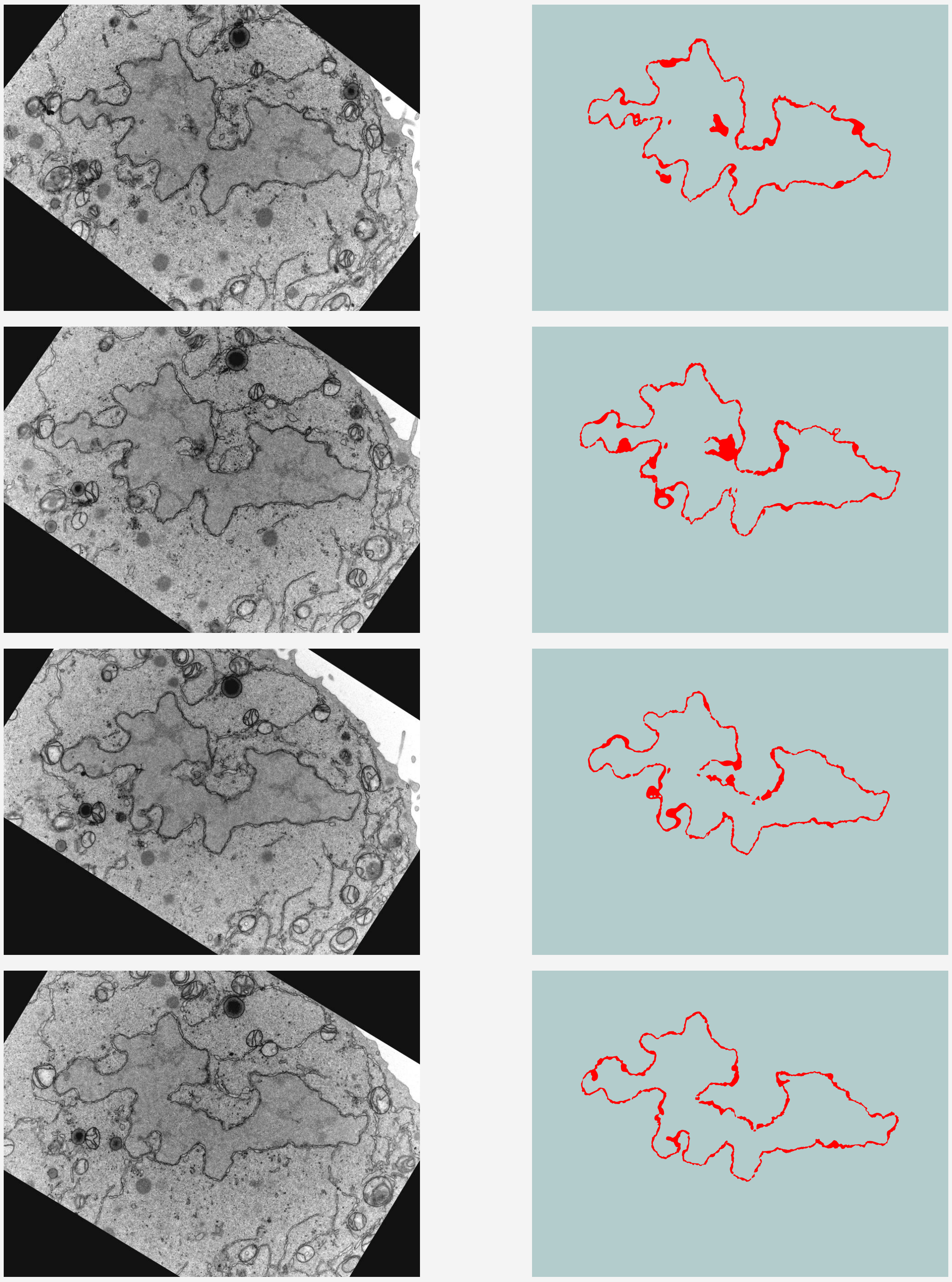


Figure 5.1: Four adjacent slices from the data set with their manual segmentations.

5.2 3D Reconstruction

Overall, the aim of the segmentation process is to produce a 3D reconstruction of the nuclear envelope which can then be studied and interacted with in 3D. The characteristics of the nuclear envelope and its relationships with other structures in the cell are critical to the mitotic process and potentially identifying cancerous cells, and studying the individual 2D slices does not provide the required contextual information.

Two images of a 3D reconstruction of the sample dataset produced from the manual segmentation of all slices are shown in Figure 5.2.

5.3 3D Structures in 2D Slices

A difficultly to consider in the algorithm is the proliferation of 2D artifacts, where the nuclear envelope is not one continuous structure in a single 2D slice due to slicing through the 3D structure. These arise due to concavities and complexities in the 3D shape as demonstrated in Figure 5.3.

In some slices these artifacts are clearly visible as small separate sections of nuclear envelope, as in Figure 5.4a, whereas in other cases they appear as small dark patches which can be difficult to classify, as in Figure 5.4b.

5.4 Noise Affected and Corrupted Images

Noise on the slices can be caused by either small particles of dust being present on the imaged sections or by the samples folding as they are placed on the resin grid. In some cases a small amount of noise on a particular slice does not affect the segmentation, but larger amounts it can make the slice unusable. For these cases the slice is considered to be corrupted and the best that can be done is to interpolate results from other neighbouring slices.

Unfortunately the line between noisy and corrupted is somewhat qualitative and hard to quantify exactly. A cautious approach would be for the user to select which slices have any noise present such that they can all be considered corrupted and discounted from the automated segmentation steps. This does however increase the workload for the end-user in terms of finding a manual segmentation for all these slices, but good results would be guaranteed.

The difficulty in choosing which slices to discount from the algorithm can be shown by comparing Figures 5.5b and 5.5d. In Figure 5.5b, the noise is significant but outside of the nucleus and nuclear envelope, key features required for the segmentation algorithm, whereas for Figure 5.5d the noise affects both of these features.

A particularly useful feature of the noise is its constant intensity value at the minimum value found in the image, which gives a quick and simple way to identify noise in images. For slices that are considered noisy this knowledge can be used to refine and restrict the segmentation algorithm, but for corrupted slices the noise affects the performance too much to obtain even average results. This knowledge could also be used to automatically classify slices as noisy or corrupted by looking at the proportion of noisy pixels in the image. This would be suitable for the cautious approach above, but for a less cautious approach the problem is that of automatically finding noise and the particular structures the noise affects, a far from simple task.

Therefore, the best method for dealing with noise is to allow the end-user to classify slices into clean, noisy or corrupted slices by examining each for noise and investigating which structures of the cell are affected. This is a small pre-processing step that is a suitable compromise to make in terms of overall segmentation time.

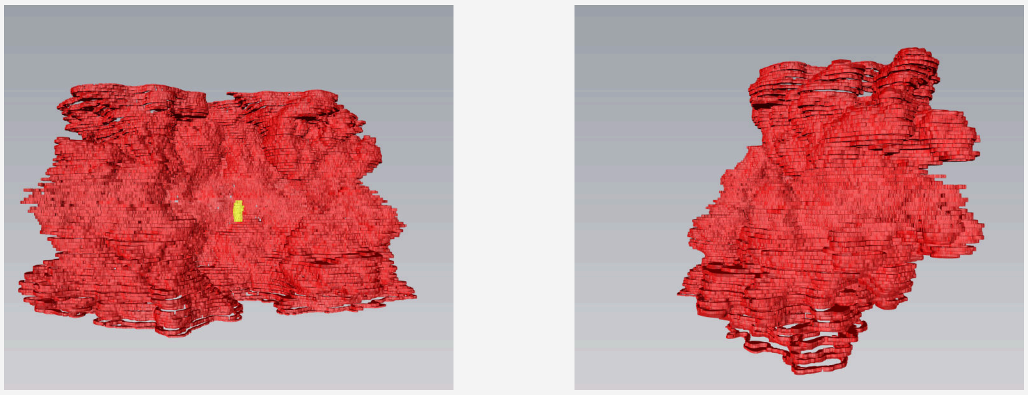
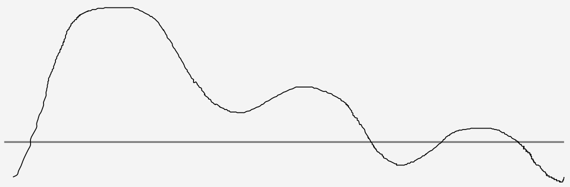
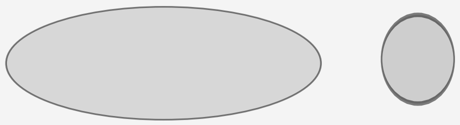


Figure 5.2: 3D reconstruction of the manual segmentation.

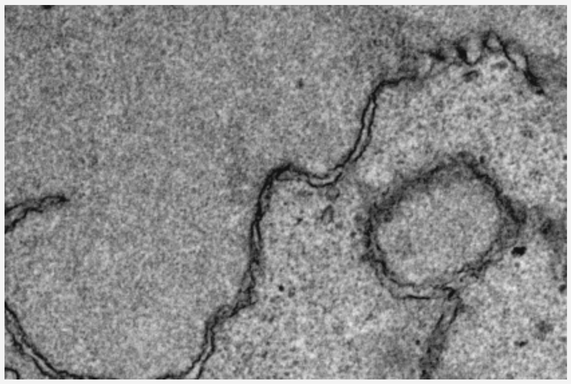


(a) Side on view of the slicing process

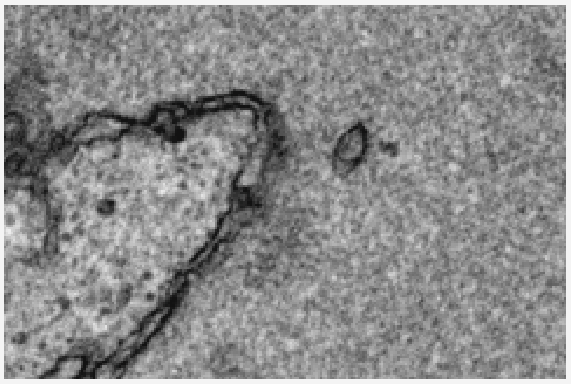


(b) Viewing the resulting slice from the top down gives two separate structures

Figure 5.3: Viewing the nucleus from side on and in 3D shows that there is only one nuclear envelope structure in the cell. However, the 2D image resulting from slicing through the nucleus causes two disconnected sections.

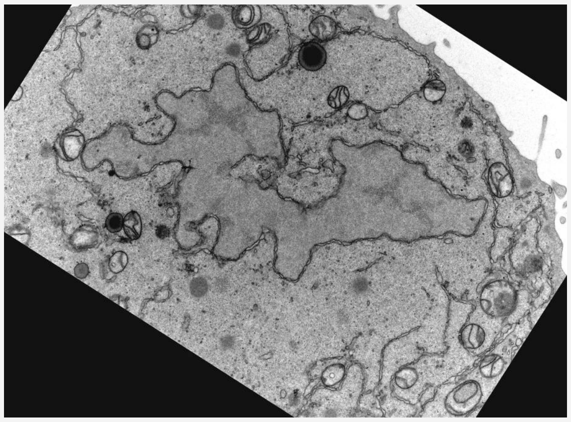


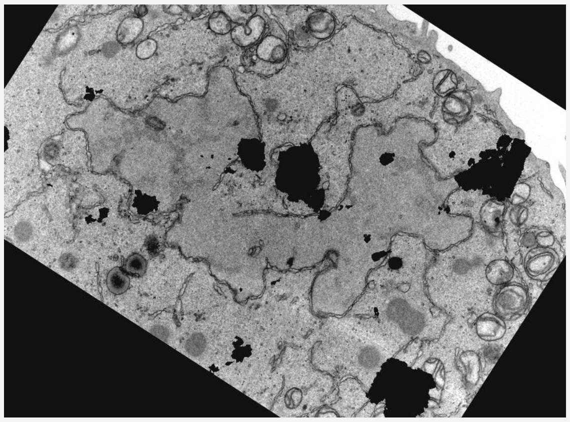
(a) A small section of nuclear envelope is visible outside the main structure

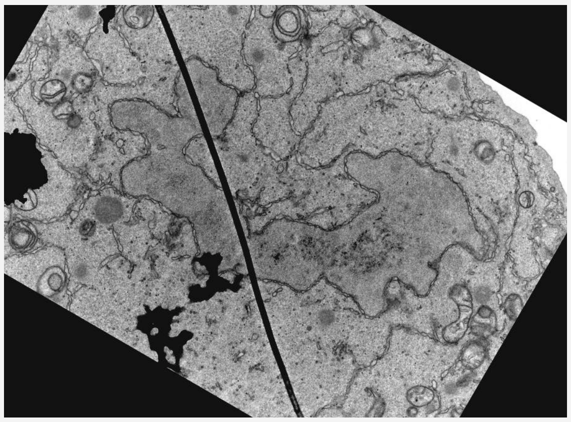


(b) A patch of nuclear envelope inside the main structure

Figure 5.4: Examples of 3D sections of nuclear envelope creating 2D artifacts.


(a) Ideal image


(b) Noisy slice due to dust


(c) Noisy slice due to folding of the sample

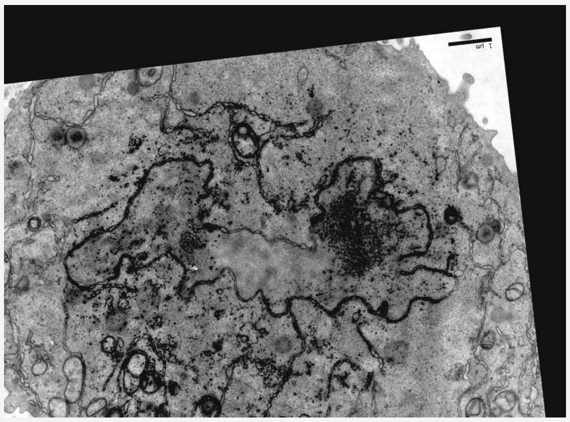
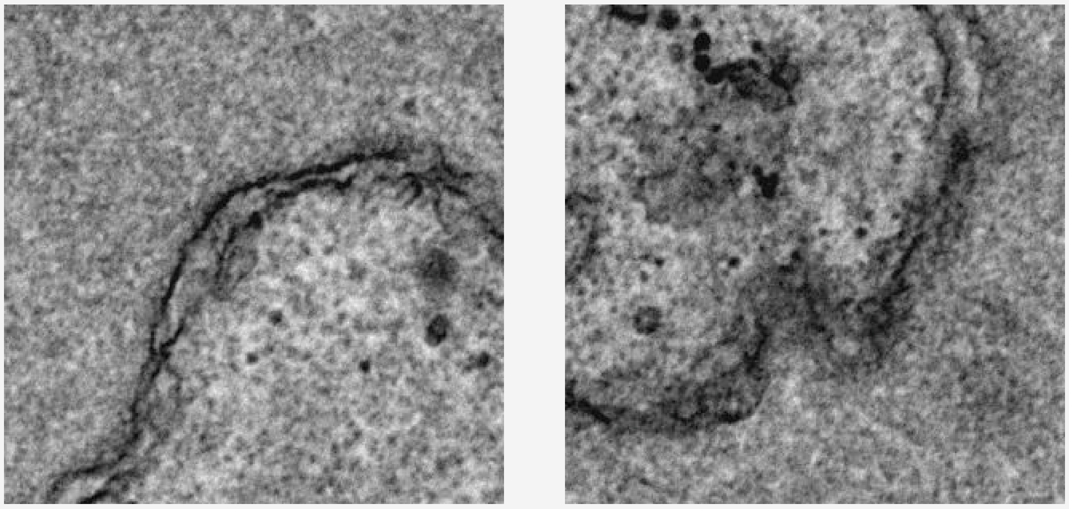

(d) Corrupted slice

Figure 5.5: Examining characteristics of noise in corrupted slices.

5.5 Blurring of the Nuclear Envelope

A particular problem that can occur with the images obtained through transmission electron microscopy is the definition of the nuclear envelope can be slightly blurred, in a visual sense, depending on the angle between the blade and the sample during the electron microscopy process. An example of this blurring in the images is shown in Figure 5.6.



(a) A well defined section of nuclear envelope where the cutting blade is roughly perpendicular

(b) A blurred section of nuclear envelope

Figure 5.6: Example showing the effect of cutting angle on the definition of the nuclear envelope

This blurring of the nuclear envelope in the images is because of the projection of a 3D structure onto a 2D image, remembering that each slice in the dataset is approximately 70nm thick. When the cutting angle and nuclear envelope are close to perpendicular then the projection of the small amount of membrane in a 70nm slice onto a 2D image will be well defined. This can be imagined as the nuclear envelope only having variation in one direction, and because that is parallel to the viewing direction we cannot see it. However, when the angle moves away from perpendicular some movement in the 70nm slice occurs in the x and y directions local to the image, therefore when this is projected we see a more blurred structure as if the section had been squashed flat.

6

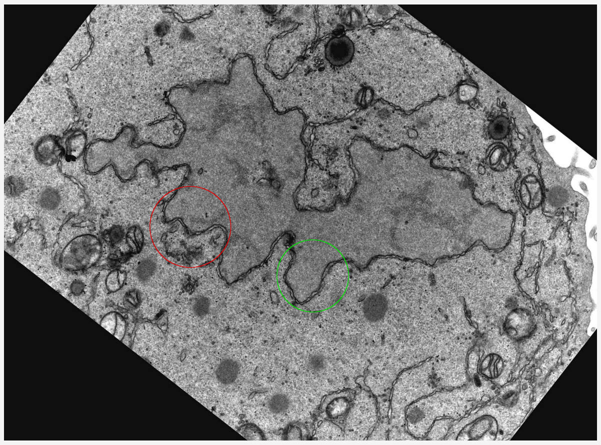
Algorithm

This chapter gives an overview of the algorithm used to achieve the segmentation of the nuclear envelope, without considering any implementation details. It focuses on the steps of the algorithm that work from the inside of the nucleus out towards the nuclear envelope. Each 2D slice of the data set is primarily segmented in 2D, but the third dimension does provide consistency between slices and can easily improve results. Corrupted slices are discounted from the earlier steps of the algorithm as the information they provide cannot be deemed reliable. Instead they are estimated from neighbouring slices in the stack at a later stage in the algorithm.

The idea behind the algorithm is to make as much use of the different textures of the nucleus and cytoplasm as possible. Figure 6.2 shows this difference, where the texture inside is more regular, smooth and is said to have **low variance**. In contrast the texture on the outside of the nucleus is much less regular and contains the other dark cellular structures such as vesicles which contribute to it having a higher variance.

Some of the steps consider the data's third dimension to ensure consistency of information across all the slices in the dataset. Key to these steps is the assumption that over the 70nm physical space between slices the nuclear envelope has moved minimally, which is reasonable given that it is only ever used to improve steps that find approximations to structures. A more precise model of movement of the nuclear envelope between slices would be required to carry out absolute calculations or interpolate very accurately between two nuclear envelope segmentations.

Figure 6.1 shows two dataset-adjacent slices. We can see effectively zero change in the global shape, however the green circles show a slight smoothing of the envelope in Figure 6.1b, and the red circles show a region of envelope in Figure 6.1b that has sharper curves than the respective region in Figure 6.1a.


(a) Dataset slice 0000

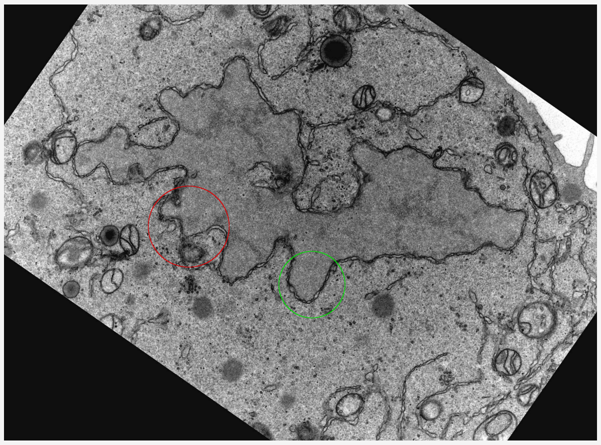

(b) Dataset slice 0001

Figure 6.1: Example slices showing the minimal movement assumption of the nuclear envelope.

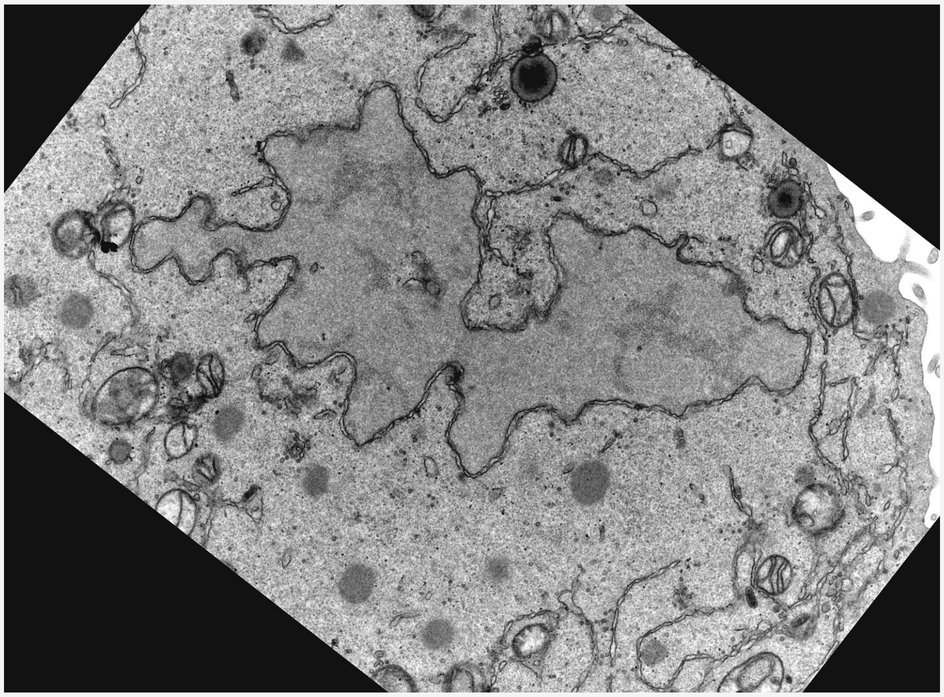


Figure 6.2: Sample image showing difference in texture between the inside and outside of the nucleus.

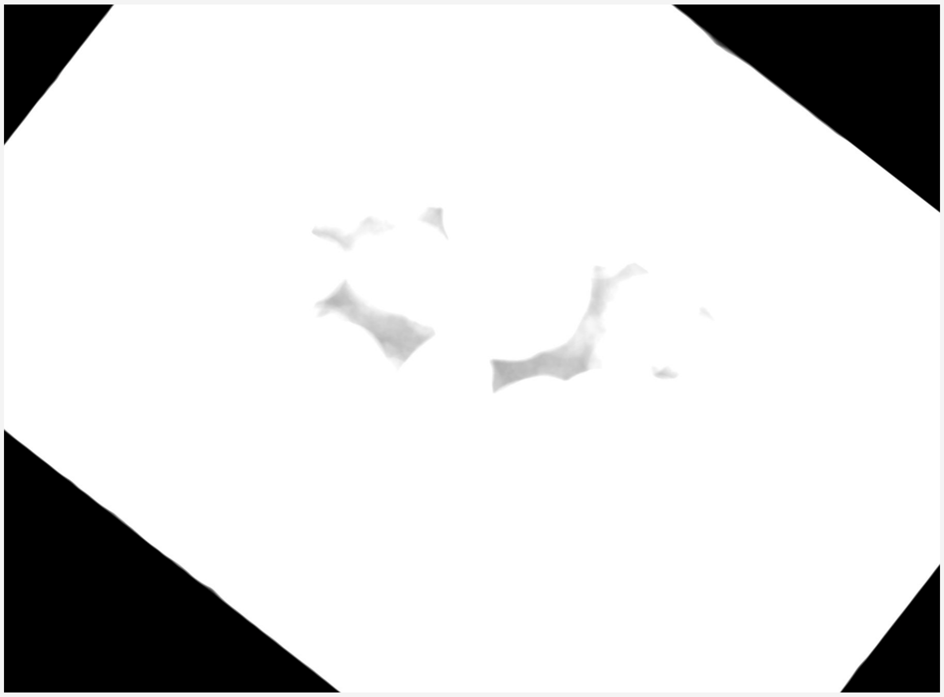
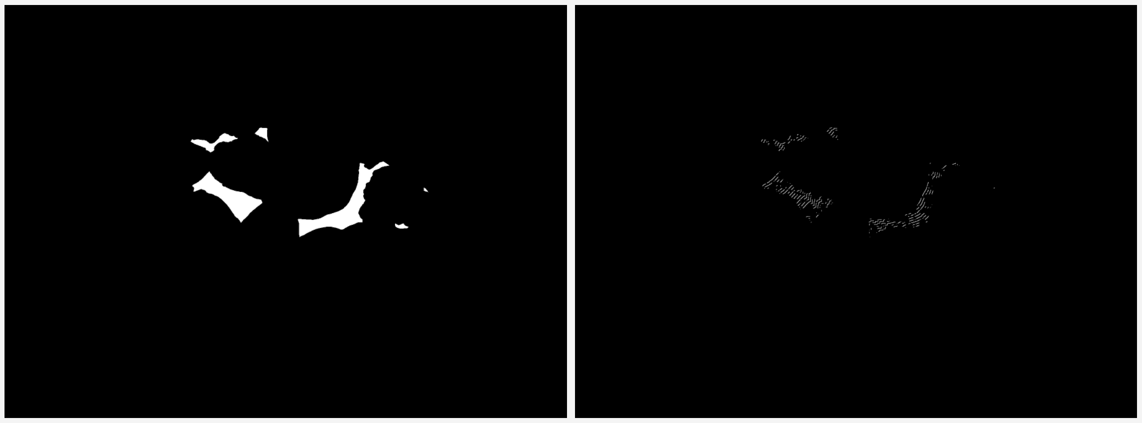


Figure 6.3: Identifying points within the nucleus using a large neighbourhood variance filter.

6.1 Identifying Nucleus Seed Points

The first step of the algorithm is to identify some pixel locations that we can be certain are inside the nuclear envelope as a way of starting to locate the nucleus in the image. Successful identification of these suitable seed points is a crucial step that underpins and affects the success of the whole algorithm.

To identify suitable seed points within the image a variance filter with a large neighbourhood size is passed over the image. Any points with a variance that is still inside the range of 0-255 are then considered as candidates for seed points and are passed on to the next step. Figure 6.3 shows the result of this operation on the sample image in Figure 6.2. We can see that the cytoplasm has been set to the maximum value in the image of 255, but some of the areas inside the nucleus have non-maximal values. At this stage we only need to find **some** points within the nucleus and not an entire representation of the nucleus as that is found using these seed points as input to a different algorithm step. The image in Figure 6.3 after finding areas of non-maximal variance is shown in Figure 6.4a, and to reduce computation time at later stages these regions are thinned to a smaller number of points as shown in Figure 6.4b. The latter representation of seed points will be used from now on.



(a) Seed regions found from large neighbourhood variance filter.

(b) Seed regions reduced to a smaller number of points to reduce computation time at later stages.

Figure 6.4: Seed points from non-maximal variance.

6.2 3D Consistency of Nucleus Seed Points

In some cases if the large neighbourhood variance filter technique is unlucky it can identify some points that are outside the nucleus, or given noise or odd shaped nuclei it can fail to find enough suitable seed points. An example of the variance filter applied to a different image in the stack where occurs is shown in Figure 6.5b. Fortunately these poor results are few and far between, so we can use the seed points found in neighbouring slices and the minimal movement assumption to refine the set of seed points for any image in the stack.

For the removal of erroneous seed points outside the nucleus we use the observation that the highly cluttered cytoplasm means there are seldom any seed points found outside the nucleus in the same place in two neighbouring slices. The strategy is therefore to consider for each seed point in a particular slice the evidence there is for that seed point, given the seed points of its neighbouring slices. If there is not sufficient evidence from neighbouring slices that a seed point is valid then it is filtered out of the set. The threshold for evidence is chosen such that strong evidence for seed points must come from at least two neighbouring slices on both sides of any particular slice. The seed points from Figure 6.5b after considering support from neighbours are shown in Figure 6.6, and if we compare this to the original slice in Figure 6.5a we can see that all seed points now lie within the nucleus.

In the case where noise or odd shapes cause there to be few seed points found by the large neighbourhood variance filter, we can use a similar strategy of considering support from neighbouring slices to identify **new** seed

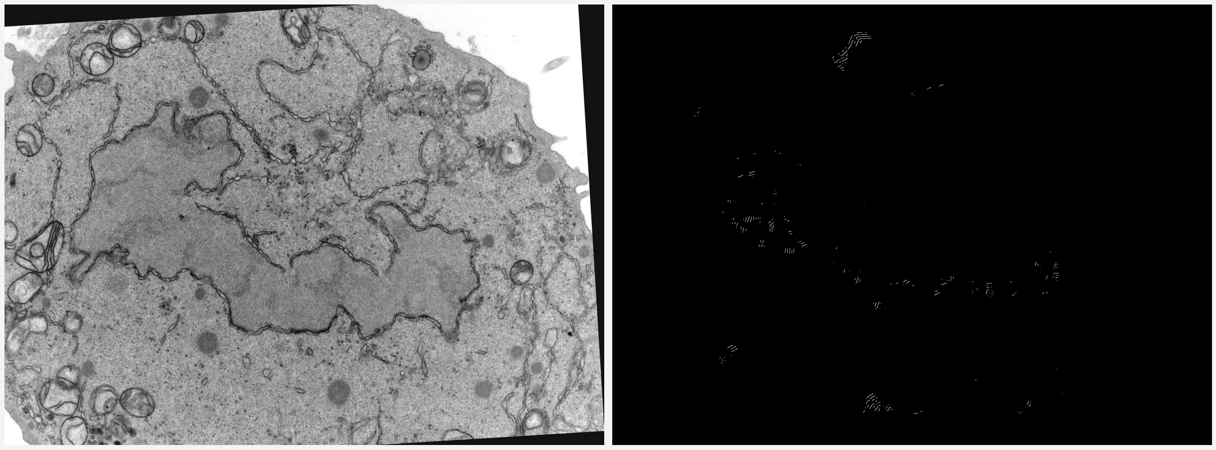

(a) Original Image **(b)** Seed points without 3D consistency.

Figure 6.5: The large neighbourhood variance filter identifies some seed points that are outside of the nucleus.

points for any slice. The aim with this step is to find new points that weren't identified previously but have very high support from all neighbouring slices. Figure 6.7 shows the seed points in Figure 6.6 along with those added points that have strong evidence from neighbour slices. We can see that we now have a much greater number of seed points, and they themselves already make a good approximation to the nucleus area in this case.

6.3 Obtaining an Approximation to the Nucleus

The purpose of finding seed points inside the nucleus is to subsequently find an approximation to the area of the nucleus. The seed points found from previous steps are used as starting points for a flood filling algorithm which captures the nucleus area by moving from the seed points inside it out towards the nuclear envelope.

One of the main difficulties in this step is to stop the flood filling at locations where there are gaps in the nuclear envelope. Individual pixel grey values do not change dramatically in a gap at the transition between the nucleus and cytoplasm texture, but the variance over a neighbourhood does. This prohibits the use of a flood fill defined simply by a grey level intensity threshold and instead a variance filter must be passed over the image before performing the flood fill. This is effectively flooding the area defined by the lower variance texture inside the nucleus, and an example demonstrates this in Figure 6.8. This goes to show the importance of the texture of the nucleus in this algorithm, as it is used both for the identification of seed points and defining the region over which to flood fill.

In some slices the nucleus has dark patches that create higher variance, preventing the flood filling algorithm from capturing that area. The flood filling algorithm has to be quite restrictive to avoid leaking through a gap in the nuclear envelope, and this requires a low threshold that consequently can cause holes to appear in the nucleus approximation. Furthermore, the slice can show that there is a divide in 2D between parts of the nucleus, an example of which is shown in Figure 6.9. This demonstrates the necessity to find as many correct seed points as possible from the previous steps; the flood filling algorithm is carried out for every seed point and all found regions are joined together to give the approximation to the nucleus. Areas of the nucleus which are not captured by this flood fill can affect the accuracy of results later in the algorithm.

Finally, there are often small holes in the approximation due to small patches of slightly darker nucleus texture. The last step is to perform a morphological closing operator as shown in Figure 6.10. Many of the smaller holes in the approximation are closed and the outline of the approximation is also slightly smoother, which in practice improves the outline of the nucleus approximation without causing it to cross the nuclear envelope at any point. Any holes left in the approximation are retained as 3D features of the nuclear envelope (Section 5.3).

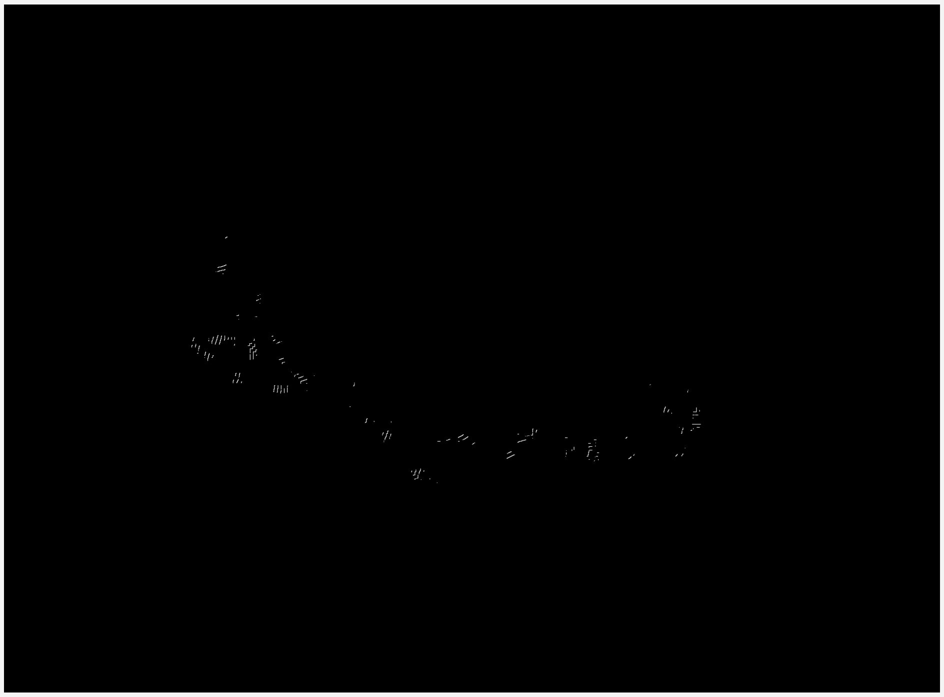


Figure 6.6: Identifying points within the nucleus using a large neighbourhood variance filter.

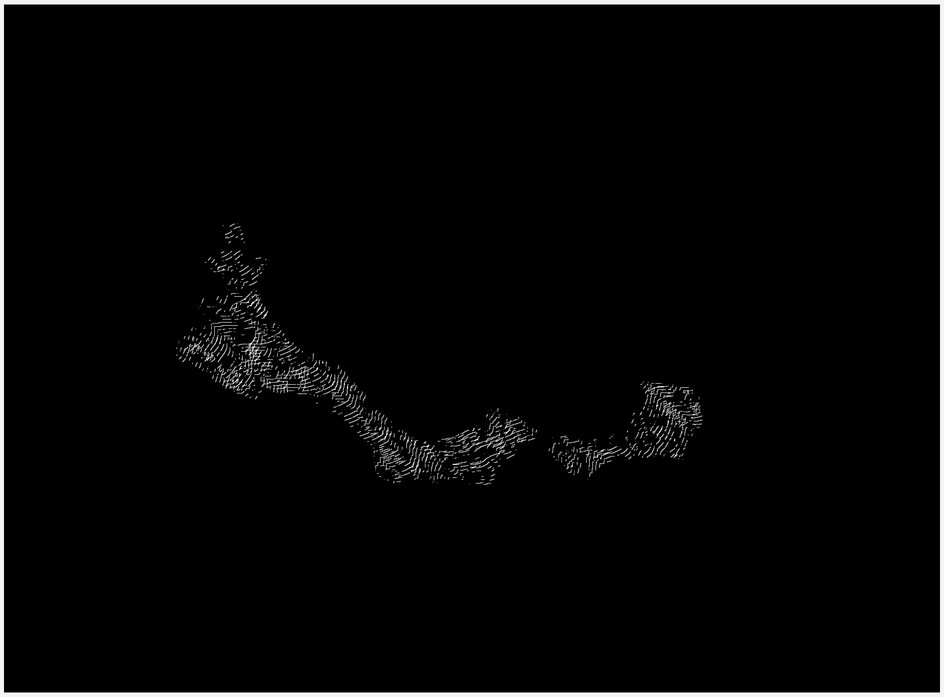
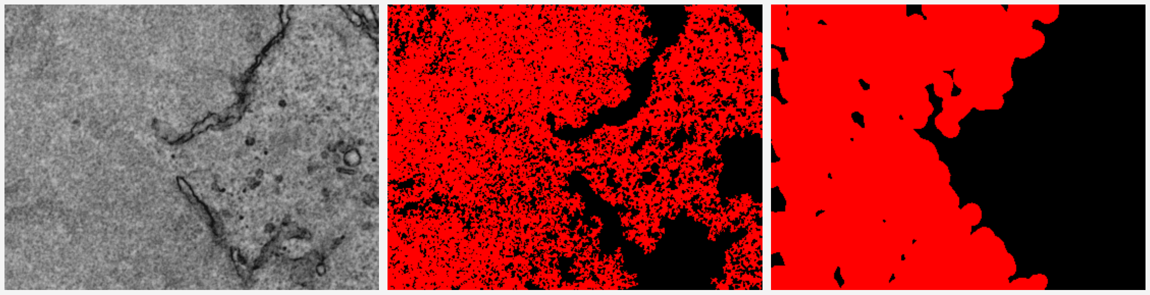


Figure 6.7: Interpolating seed points through support from neighbouring slices can greatly increase the number and quality of a slice's seed points.



(a) Section of a slice showing a gap in the nuclear envelope

(b) Flood fill based on a grey value threshold

(c) Flood fill based on the variance filtered section

Figure 6.8: Showing the need for a flood fill algorithm based upon the variance inside the nucleus, rather than on grey level values.

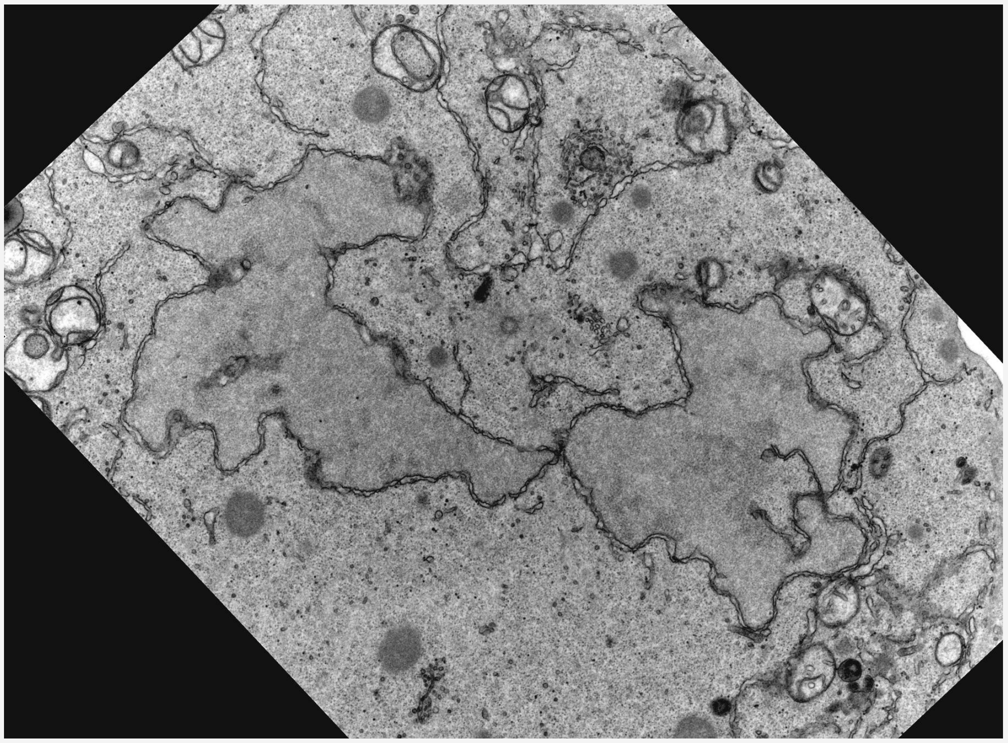
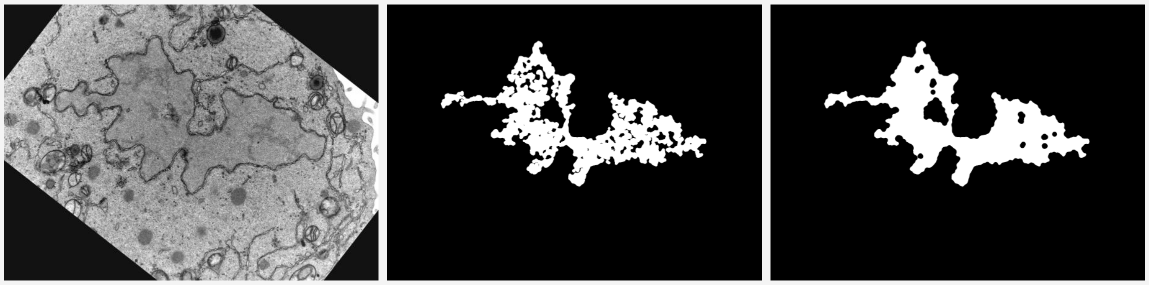


Figure 6.9: Showing disconnected regions of the nuclear envelope.



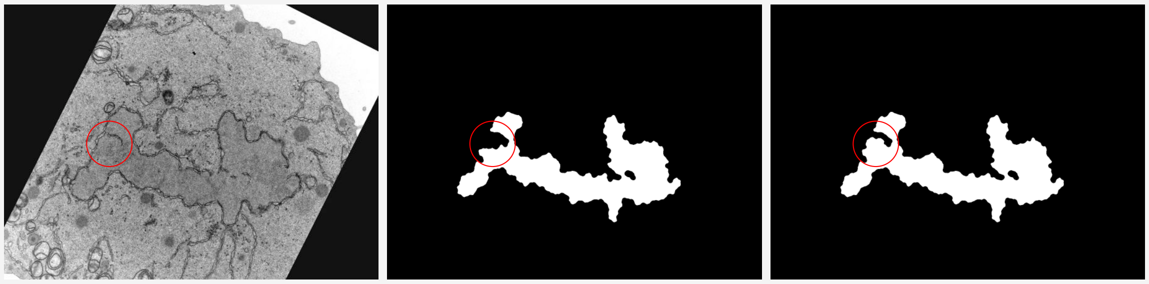
(a) Slice 0000 **(b)** Nucleus estimate after flood fill algorithm **(c)** Nucleus estimate after morphological closing operator

Figure 6.10: Showing the effect of the closing operator on the nucleus approximation.

6.4 3D Consistency of the Nucleus Approximation

Unfortunately, for some slices the flood filling algorithm does not capture the entire nucleus. This often happens when there are small loops of the nucleus with high variance values at their thin entry points that prevent its capture by the flood fill algorithm. Figure 6.10a shows a good example of a loop to the left of the nucleus, although in this case the flood fill was not blocked by a high variance entry to the loop. Missing areas in the nucleus approximation can also happen if we are unlucky with the placement of some dark patches inside the nucleus as in Figure 6.11.

Fortunately neighbouring slices are often successful in obtaining areas similar to those that weren't captured in a particular slice, albeit in a slightly different place and shape. Therefore, in a similar manner to the 3D consistency of seed points in Section 6.2, we use the minimal movement assumption of nuclei to further refine the approximation to the nucleus. This again considers evidence for regions based on the neighbour's approximations, and those regions with sufficient evidence are used to refine the original approximation.



(a) Original slice **(b)** Result of approximating the nucleus **(c)** Nucleus approximation after 3D consistency

Figure 6.11: Showing the effect of ensuring 3D consistency with neighbouring slices to obtain a better approximation to the nucleus for slice 0064. We see that in (b) there is a small section missing to the left of the nucleus. Ensuring the 3D consistency results in that section being added as shown in (c).

6.5 Finding Seed Points on the Nuclear Envelope

Once we have obtained a suitable approximation to the nucleus we use this as a base to find the nuclear envelope. The approach here is to find further seed points that lie on the nuclear envelope in order to find its area with a second flood filling step. Finding these seed points on the nuclear envelope cannot simply be done by expanding the outline of the nucleus approximation for a number of reasons:

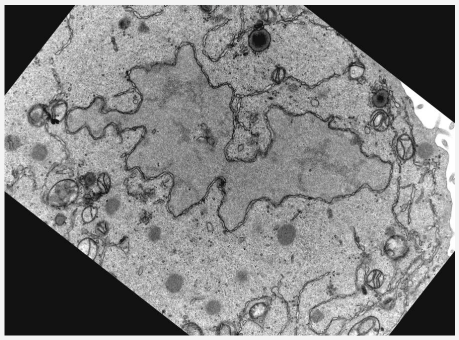
- The nuclear envelope does not have a constant thickness.
- The nucleus approximation is not always a fixed distance from the nuclear envelope; this depends on the textures around the nuclear envelope.
- The perinuclear space has a different texture to the dark lines that characterise the double lipid bi-layer structures of the nuclear envelope membranes.
- Gaps are not considered part of the nuclear envelope.

We can still use the outline of the nucleus approximation, but we have to do so in a slightly more accurate manner as we have these restrictions to consider.

Intuitively, if we travel outwards from this outline, up to a suitable maximum distance, we can expect in most cases to enter and subsequently exit the nuclear envelope structure. This technique requires some blurring of the nuclear envelope in the case that it has the well defined double-lipid bi-layer structure; an example of this is shown in Figure 6.12. This blurring works on closing the perinuclear space such that well defined dark structures of the nuclear envelope merge into one distinctive edge and have just one entry and exit point.

To find seed points inside the nuclear envelope we find the points of entry and exit on the nuclear envelope when travelling outwards and take the mid-point. An example of this is shown in Figure 6.13, where the red line shows the path travelled outwards from the nuclear envelope approximation and the green lines show the entry to and exit from the nuclear envelope. The white dot is the seed point added to the overall set for this slice.

It is also possible that multiple structures might be encountered when travelling outwards from the outline; the nuclear envelope is the first structure on such a path and all subsequent similar features encountered are ignored.


(a) Slice 0000

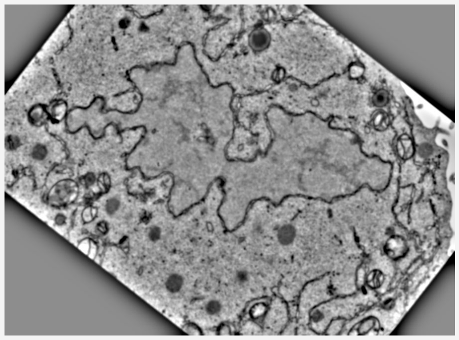

(b) Blurred edges of the nuclear envelope

Figure 6.12: Attempting to close the perinuclear space in order to merge the double lipid bi-layer structure into one distinctive edge.

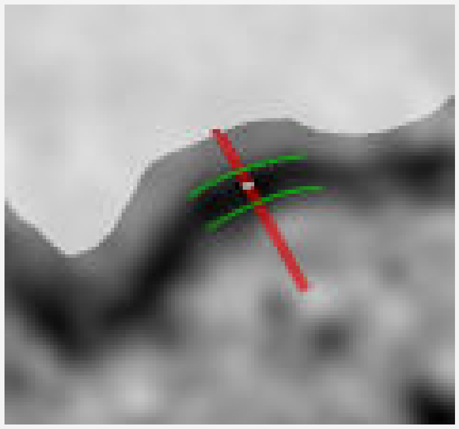


Figure 6.13: Expanding the boundary into the nuclear envelope to find seed points

6.6 Obtaining an Approximation to the Nuclear Envelope

In a similar way to Section 6.3 a flood fill is carried out from the multiple seed points located within the nuclear envelope. The same blurring of the nuclear envelope is required to find the double lipid structures and the area between them in the case that they are well defined. Additionally, increasing the contrast within the image can accentuate the nuclear envelope and obtain better results from the flood filling. This flood fill, in contrast to finding the nucleus approximation, is based upon the grey-level intensity values and the results are used as an approximation to the nuclear envelope.

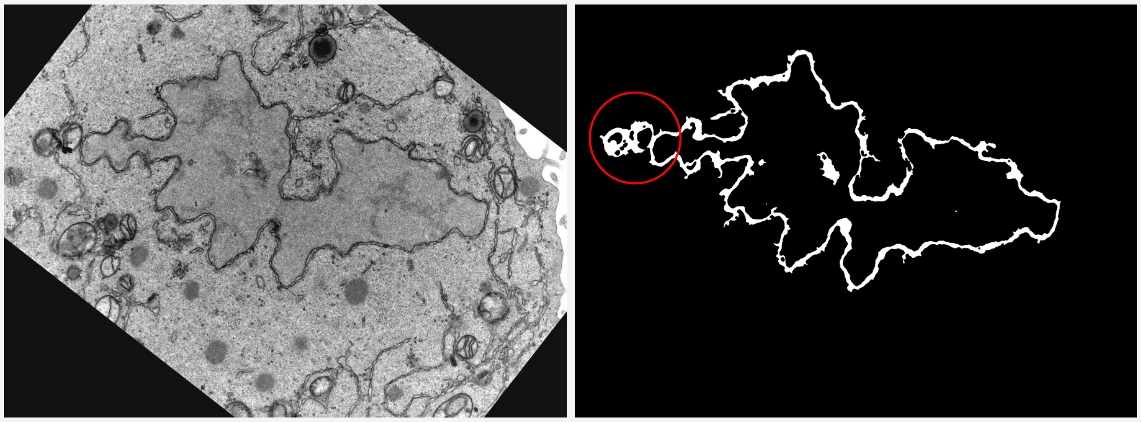

(a) Original Image **(b)** Approximation to the nuclear envelope

Figure 6.14: Flood filling of the nuclear envelope to obtain its approximation.

6.7 Pruning Connecting Structures

In some cases it is not always possible to obtain a good approximation to the nucleus at all places in each slice; it may be too far from the nuclear envelope and the maximum length travelled from the nucleus approximation outline doesn't reach the nuclear envelope. An example is shown in Figure 6.15. This then means that when seed points are obtained as in Section 6.5 we may not be able to locate them in as many places on the nuclear envelope as we would like. The flood filling algorithm in the previous step has to make up for this by utilising the connectivity of the nuclear envelope to locate regions where sparse numbers of seed points were found.

One problem with this is that there are often structures connected both physically and visually to the nuclear envelope, and these are also found by the flood filling algorithm. We can see an example of this to the top left of Figure 6.14b where two mitochondria have been segmented along with the nuclear envelope.

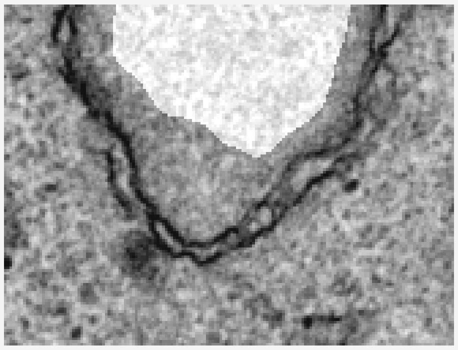


Figure 6.15: The nucleus approximation is too far from the nuclear envelope to locate seed points.

To prune off these extra connected structures we use the proximity of the structures found in the nuclear envelope estimate to the approximation of the nucleus found in Section 6.3. The intuition is that if we start at a point inside the nucleus approximation and travel outwards, the first structure we encounter is the nuclear envelope. Any further structures that we encounter after the nuclear envelope when travelling outwards cannot be nuclear envelope, and thus they can be pruned from its approximation. The nuclear envelope segmentation after the pruning of connected structures in Figure 6.14b is shown in Figure 6.16.

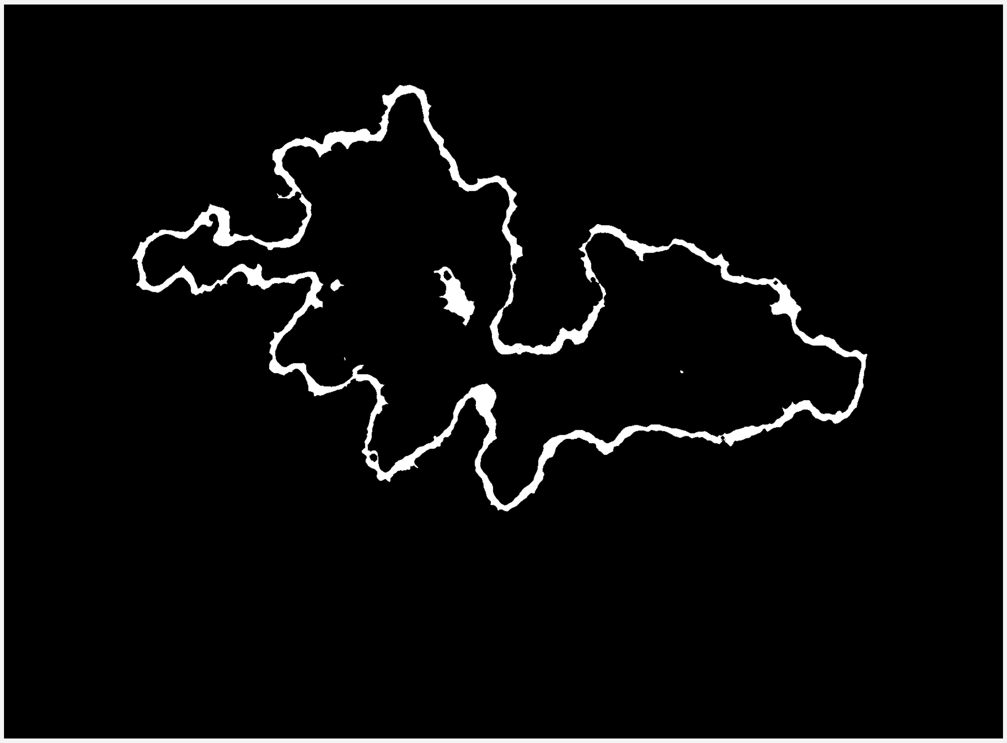


Figure 6.16: Nuclear envelope segmentation after pruning connected structures from its estimate.

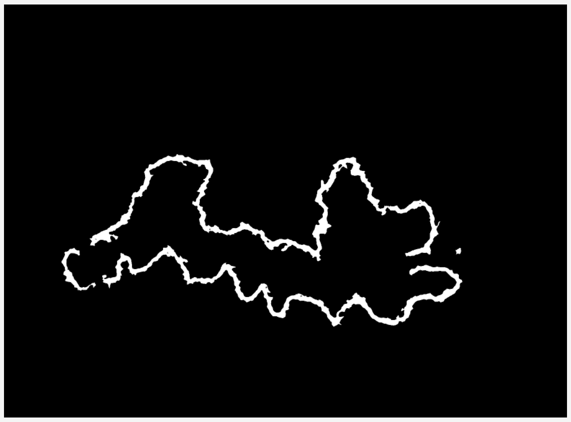
6.8 Manual Corrections

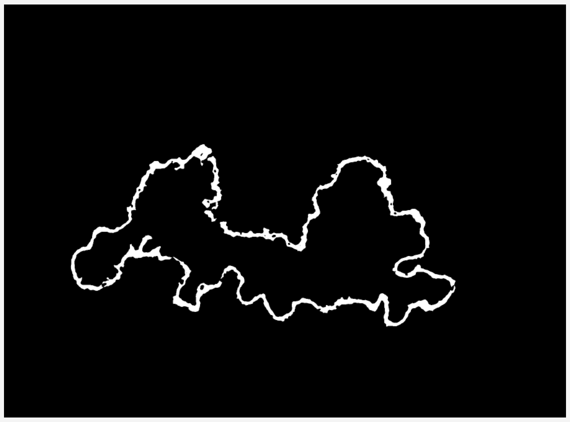
In many cases the segmentation achieved is accurate, but a number of manual corrections will always be necessary because of noise in the dataset. Furthermore, there are some complex problems with the algorithm that lead to segmentation errors, and these are explored in more detail in Chapter 10.

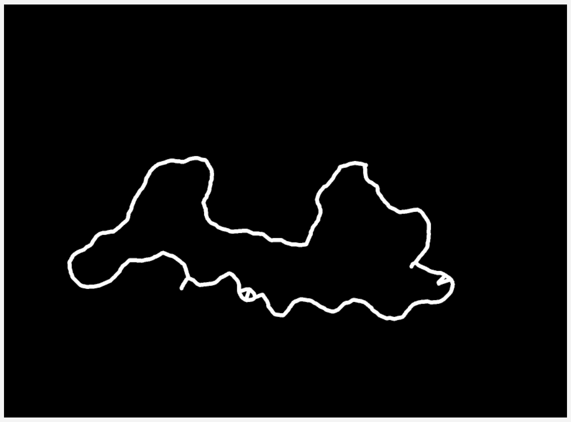
6.9 3D Interpolation of Corrupted Slices

Once manual correction of the non-corrupted slices has been carried out our attention turns to those slices that are considered corrupted. These slices require the most manual segmentation time, but given that we already have correct segmentations for all other non-corrupted slices we can use 3D considerations to obtain an estimate of the nuclear envelope for a corrupted slice. We have to use the nuclear envelope's minimal movement assumption again, but this is safe as we are only obtaining an approximation to the nuclear envelope.

To find this approximation the average position of the nuclear envelopes from the corrupted slice's immediate neighbours is found. The average position is defined by a single pixel width line, which is then expanded to create an estimate that is 40nm thick, the average thickness of the nuclear envelope. The estimated segmentation for a corrupted slice, number 0070, using its immediate neighbours in this manner is shown in Figure 6.17.


(a) Nuclear envelope of slice 0069


(b) Nuclear envelope of slice 0071


(c) Estimated nuclear envelope

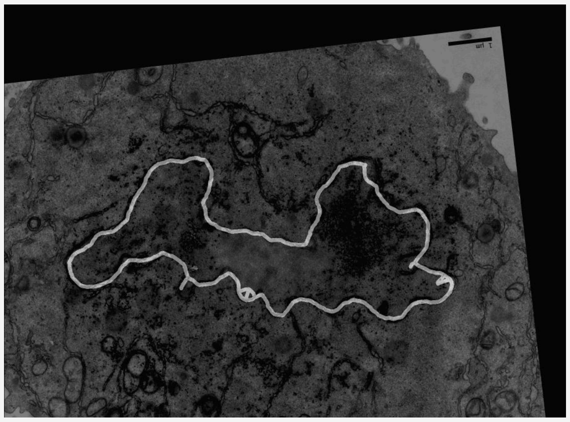

(d) Slice 0070 wit overlaid nuclear envelope estimate

Figure 6.17: Estimating the nuclear envelope segmentation of slice 0070.

6.10 3D Reconstruction

Once the corrupted slices have been manually corrected we can be certain that the segmentations we have are those of the nuclear envelope. Given the segmentations for all 2D slices we stack them in order to create a 3D volume, noting that each slice is made to be 70nm thick. Background pixels are made transparent and the volume can be rendered and interacted with in a 3D environment. An example 3D reconstruction of the project dataset is shown in Figure 6.18.

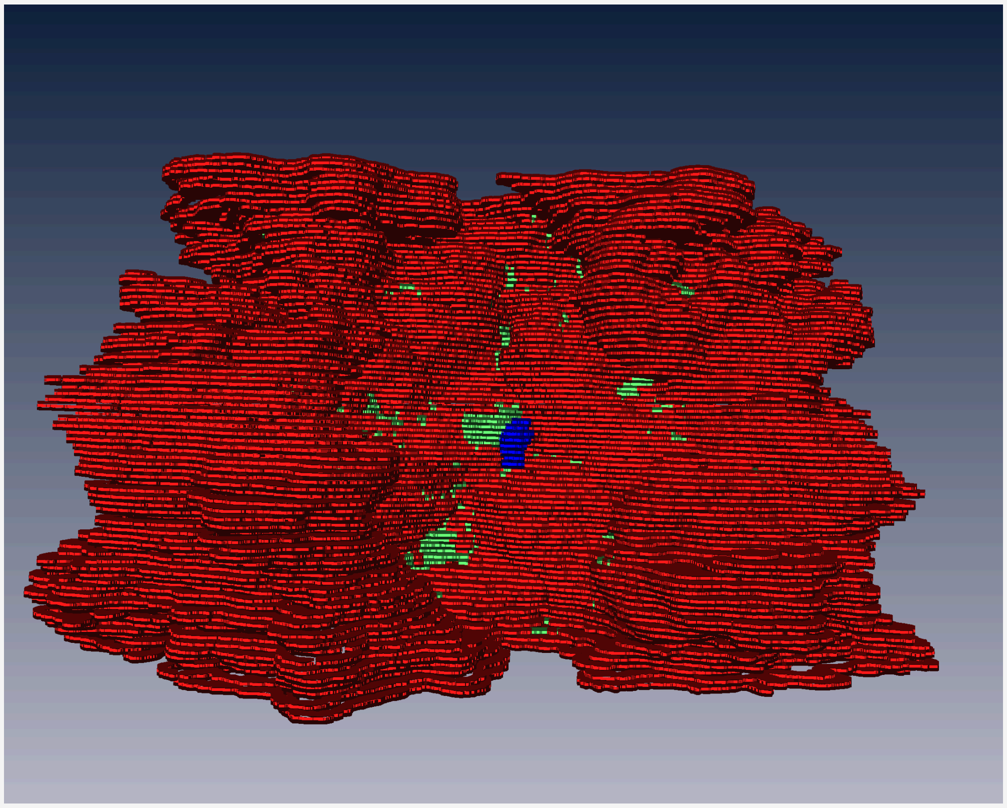


Figure 6.18: 3D reconstruction of the dataset

7 Background

Implementation

This chapter is intended to provide the required background to the implementation of the algorithm presented in the next chapter. The algorithm was developed in Java for three primary reasons:

- My familiarity with the language.
- The familiarity of the end-users with FIJI and ImageJ.
- Availability of third-party plugins for ImageJ.

7.1 ImageJ and FIJI

ImageJ [11] is an open source general purpose image processing and analysis tool written in Java. It can perform many of the operations commonly desired from a image processing package, including geometric transformations, edge detection, histogram manipulation and 3D reconstruction. Whilst it provides many of the basic and necessary techniques itself, it is also designed such that it is easily extensible with plugins. At the time of writing the number of plugins available for ImageJ is known to be at least 500.

FIJI (FIJI is just ImageJ) [39], as described by its name, is purely a repackaging of ImageJ along with a large number of popular plugins organised into a convenient menu structure. It is aimed primarily towards researchers in life sciences and many of the default plugins are provided by researchers in this area.

7.1.1 ImageJ API

The ImageJ API is a convenient collection of classes that can handle up to five-dimensional datasets. Many of the available third-party plugins make heavy use of the core functionality provided by this API. It is also well written such that there is minimal dependency on the ImageJ GUI for this functionality, and this increases its applicability to be used in standalone applications as an image processing framework. Here we take a brief look at the core classes for image management used in this project's implementation.

IJ

The IJ class provides a number of helpful static methods for commonly required actions from and environment tests upon the ImageJ application. Most of the methods interact with the ImageJ GUI and therefore require an application that runs through an ImageJ plugin, but some others provide useful general functionality:

IJ.log(String message) If a plugin is run through the ImageJ GUI this prints a message to the logging window, otherwise the message is printed to the standard output stream.

IJ.openImage(String path) Opens the given image path and returns the `ImagePlus` object that represents the image.

IJ.isLinux()/IJ.isMacOSX()/IJ.isWindows() Determines the operating system environment.

IJ.isJava16() Returns whether ImageJ is running on Java 1.6 or greater.

ImagePlus

An `ImagePlus` object is a wrapper for either a single two-dimensional image, or a higher dimensional image stack. For a single image only one `ImageProcessor` is required to store all the raw pixel values. An image stack is represented as a list of `ImageProcessors` within a single `ImagePlus`. The primary purpose for the wrapper is to store information about the images that the object contains, including metadata about the file that was loaded, a title and the real-world dimensions of pixels within the image. It also provides simple methods for showing and interacting with a loaded image.

ImageProcessor

An `ImageProcessor` is an abstract class with four subclasses representing the four different types of data that ImageJ can handle:

ByteProcessor Each pixel value is stored in one byte giving a range of possible values as all integers between 0 and 255 inclusive, thus this image is used for 8-bit greyscale images.

ShortProcessor A 16-bit greyscale image allowing for more fine grained grey values in the range 0 to 255.

FloatProcessor Holds arbitrary floating point data, where each pixel value is a 32-bit floating point number. These values are then mapped to display on the screen by setting maximum and minimum values to allow adjustment of the real values into the range 0-255.

ColorProcessor A standard 32-bit RGB image.

Roi

An `Roi` is a region of interest within a particular image upon which operations and calculations can be defined instead of operating upon all pixels in the image. Several different shape and characteristic `Rois` are provided by ImageJ:

- `Line`
- `ShapeRoi`, which relates directly to the java.awt.Shape class
- `PointRoi` represents a collection of points
- `EllipseRoi`, where a circular Roi is a special case of the ellipse
- `FreehandRoi`, usually obtained through a freehand selection within the GUI

7.1.2 ImageJ Plugins

As part of its core ImageJ also provides a number of plugins that are considered separate to the image management parts of the API. The documentation for all these plugins resides on the ImageJ wiki [7].

Wand Given a pixel within the image, the wand finds a continuous region of the same or similar intensity to the sample pixel and returns its boundary outline in the form of a `PointRoi`, ignoring any internal holes in the region.

ImageCalculator Performs simple arithmetic and logical operations including those in Section 3.1.2.

EDM Calculates the Euclidean distance map as described in Section 3.1.2.

RankFilters Implements the mean, minimum, maximum and variance filters that operate on neighbourhoods of a given size. Any variance values above the maximum value of 255 for representation in the image are mapped back to 255.

GaussianBlur Performs a Gaussian smoothing of an image given a particular standard deviation as described in Figure 3.7.

3D Viewer A 3D reconstruction plugin [40] for visualisation of volumetric data.

7.1.3 Writing Plugins

One of the easiest ways to develop a piece of software that requires all the basic image processing techniques to underpin some more complex processing is to write a plugin for ImageJ/FIJI. Plugins can either primarily use the GUI provided by ImageJ by using simple dialog boxes for configuring parameters, or can simply be an interface to access a more complex GUI and program that is more or less separate from ImageJ and built using any Java GUI library.

For a plugin that operates on just one image, FIJI provides the following sample code that implements the PlugInFilter interface:

```java
public class My_Plugin implements PlugInFilter
{
  public int setup(String arg, ImagePlus image)
  {
    // Return a bit mask that tells FIJI what images this filter can
       operate upon.
    return DOES_ALL;
  }
  public void run(ImageProcessor ip)
  {
    // Here is the action
  }
}
```

Listing 7.1: Example FIJI plugin implementing the PlugInFilter interface.

For more complex plugins, or applications that are standalone, FIJI gives sample code that implements the PlugIn interface:

```java
public class My_Plugin implements PlugIn
{
  public void run(String arg)
  {
    // Here is the action
  }
}
```

Listing 7.2: Example FIJI plugin implementing the PlugIn interface.

From the run method the application can be launched with its own GUI and functionality. All FIJI plugins must also contain a **plugins.config** file that specifies what menu entries the plugin provides and the classes they call. Finally, all source code and the configuration file are packaged into a **.jar** file that can be dropped inside the plugins directory of the ImageJ installation.

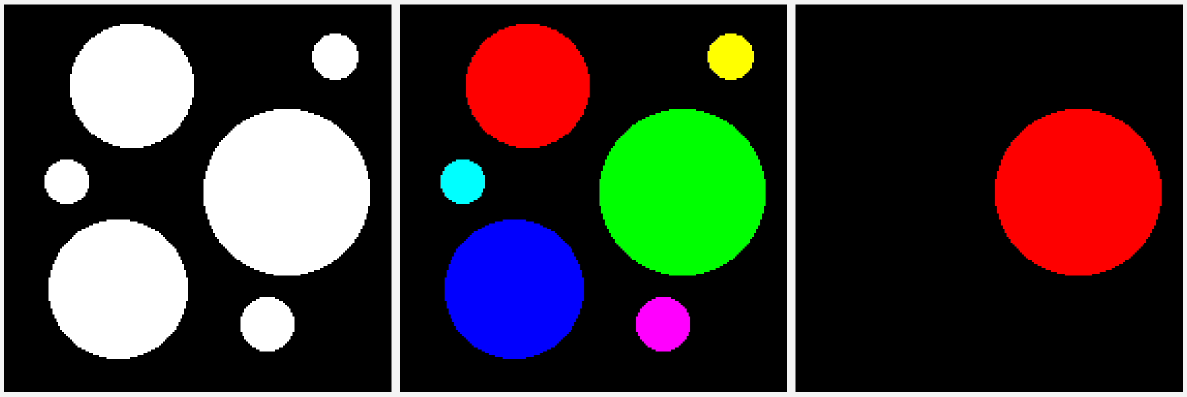
7.1.4 Third-Party Plugins

One of the main advantages to using ImageJ and FIJI is the great availability of third-party code as plugins that follow the interfaces defined in Section 7.1.3. All the code is open-sourced and has licenses allowing for free modification and adaptation to suit any particular application.

In some cases the code that implements a plugin is too tightly coupled to code that implements a GUI and there is no way to bypass this when running the plugin code through a standalone application. These plugins require refactoring by removing dialog windows for configuration options and creating a constructor to allow these options to be passed in by a standalone application.

Find Connected Regions

Find Connected Regions [30] is a flood filling algorithm, written by **Mark Longair** with contributions from Johannes Schindelin, whose operation is illustrated in Figure 7.1. It has two primary modes of operation; one is to find all connected regions within an image that are above a particular threshold (Figure 7.1b), or if a point in the image is given the maximal connected region to this particular point is found, given the threshold. This mode is shown in Figure 7.1c.



(a) Sample image for finding connected regions **(b)** Connected regions above a threshold of 100 **(c)** One connected region above a threshold of 100

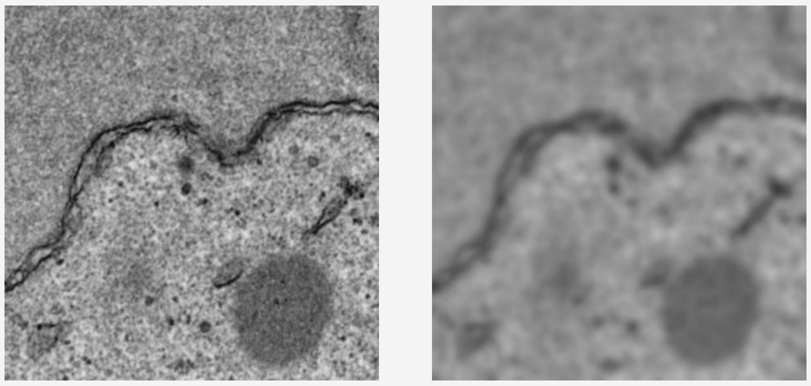
Figure 7.1: Illustration of the Find Connected Regions plugin.

FFT FIlter

A bandpass filter plugin [49] written by **Joachim Walter** that filters out structures larger than a given size and smaller than another using a Gaussian smoothing in the Fourier frequency domain. The filtering of large structures is performed by subtracting from the original image a copy of it that has been smoothed by a very large Gaussian filter, whereas the removal of small structures is carried out by a simple Gaussian smoothing. A sample section of a typical image from the dataset is shown in Figure 7.2 before and after a bandpass filter removes structures larger than 90 pixels and smaller than 7 pixels.

Skeletonize3D and AnalyzeSkeleton

The Skeletonize3D plugin [15] takes a binary image and creates its skeleton as described in Section 3.1.6. The AnalyzeSkeleton plugin [14] gives an analysis of the skeleton as also described in this section, and provides a number of methods of accessing the graphs and vertices that its defines. Both plugins are authored by **Ignacio Arganda-Carreras**.



(a) Small window of an example image before band-pass filtering

(b) Window after band-pass filtering, structures above 90 pixels and below 7 pixels are filtered

Figure 7.2: Illustration of the operation of a bandpass filter from the FFT Filter plugin.

FeatureJ

FeatureJ [35], written by **Erik Meijering** is a collection of ImageJ plugins used to compute standard features of images and regions of interest. These features are typically individual elements of feature vectors for use in a machine-learning classification based approach. The features provided by this package include:

- Derivatives
- Laplacian (second-derivatives)
- Edge detection using a Canny edge detector
- Eigenvalues of the Hessian matrix
- Structure tensor
- Statistics of values within a region of interest, including minimum, maximum, mean, median, variance and higher-order moments

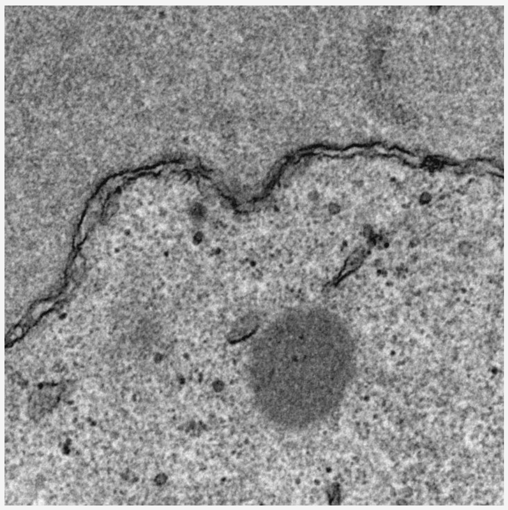
Examples of the plugin's operation are shown in Figure 7.3.

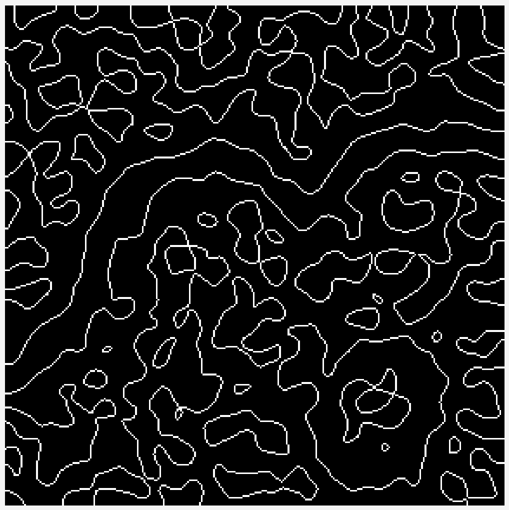
GLCM Texture

The GLCM_Texture plugin [16], written by **Julio E. Cabrera**, calculates the grey-level co-occurrence matrices and their associated features for quantifying texture within a region (Section 3.4.2). Co-occurrence can be calculated at any given pixel distance and in any of the four primary directions along the horizontal and vertical axes. A small neighbourhood region of interest around a given pixel can be used to classify local texture characteristics; if no `Roi` is specified then the co-occurrence matrices are calculated over the whole image.

FastMarching

The FastMarching algorithm is a flood fill algorithm with increased sensitivity to encountered boundaries, looking not only at individual pixel intensities but also checking that the change in pixel intensities falls below some threshold. Multiple seed points can be specified from which to propagate the region growth. The rate at which the flood fill progresses depends upon the change in pixel intensities [6]; if the change is small then the region


(a) Sample image


(b) Zero crossings of the Laplacian (second derivative) after a Gaussian smoothing at standard deviation of 6

Min	Max	Mean	Variance
15	219	119.45	1000.387

(c) Statistics calculated for the sample image

Figure 7.3: Illustration of the FeatureJ plugin.

is allowed to grow quickly, as the pixels are similar. Conversely, if the difference between intensities is large it is likely that we are getting close to a boundary and the growth of the region is this area is slowed.

The FastMarching algorithm is part of a more general LevelSets plugin [21] written by **Erwin Frise**.

Differentials

Authored by **Philippe Thévenaz**, the Differentials plugin [46] can compute derivatives of the pixel intensity in both the horizontal and vertical direction. Its operation is similar to that of the edge detection operators explained in Section 3.2.2 without thresholding their output.

7.2 Algorithm Framework

The design of the software for creating this algorithm is intended to be general enough to apply to any segmentation algorithm that could be applied to this type of image. The vision of the end-users at CRUK is to have as an end result a piece of software that is capable of segmenting all interesting features within the cell, not just the nuclear envelope. The software is intended to be loosely coupled to the specific nuclear envelope application, and provide a general framework over which a user interface or command-line program could be placed.

AlgorithmInstance

The full class definition of `AlgorithmInstance` is shown in Appendix A.1.

All algorithms require an extension of the `AlgorithmInstance` class, a container for a single instance of the algorithm i.e. one input image and images from intermediate processing steps.

One important consideration here is that with a large number of instances, and multiple processed images per instance, the memory requirement for the algorithm can be very high. Therefore, each `AlgorithmInstance` is provided with a directory into which it can save any intermediate images that are not required at particular stages of the algorithm, releasing the memory they occupy. The abstract functions `serialise()` and `unserialise()` are implemented by the specific algorithm instance, which also defines a set of bit-mask flags for determining which images to (un)serialise.

`AlgorithmInstances` also implement the Comparable interface such that they can be ordered according their position in the stack; a necessary step if doing any processing in parallel.

AlgorithmStep

An algorithm step is a part of an algorithm that can operate on individual images concurrently, and this concurrency is handled at a higher level in the algorithm. An example of simple algorithm steps would be running a Gaussian or variance filter over all images. The abstract class for an algorithm step is shown in Listing 7.3:

```java
package segmentation.algorithms;

public abstract class AlgorithmStep<T extends AlgorithmInstance>
    extends AlgorithmProcess
{
  public abstract T step(T instance)
    throws AlgorithmProcessException;
}
```

Listing 7.3: AlgorithmStep abstract class.

The generic `T` refers to the application specific `AlgorithmInstance`.

AlgorithmFilter

In contrast to an algorithm step, an algorithm filter operates upon all images at the same time, as happens in the 3D consistency parts of the segmentation algorithm. Any concurrency that can be exploited is handled within the filter itself rather than at any higher level. The abstract class for an algorithm step is shown in Listing 7.4.

```java
package segmentation.algorithms;

import java.util.List;

public abstract class AlgorithmFilter<T extends AlgorithmInstance>
    extends AlgorithmProcess
{
  public abstract List<T> filter(List<T> instances)
    throws AlgorithmFilterException;
}
```

Listing 7.4: AlgorithmFilter abstract class.

AlgorithmProcess

A superclass to both `AlgorithmStep` and `AlgorithmFilter`, an `AlgorithmProcess` is quite simply a wrapper for some debugging information and a superclass that gives the ability for steps and filters to be interleaved in one data structure.

```java
package segmentation.algorithms;

public abstract class AlgorithmProcess
{
  protected boolean debug = false;

  public void setDebug(boolean debug)
  {
    this.debug = debug;
  }

  public abstract int requiredImages();
}
```

Listing 7.5: AlgorithmProcess abstract class.

One important method that must be implemented and set correctly for all steps and filters is `requiredImages()`, as this returns the bit-mask for the application's `AlgorithmInstance` that specifies all the intermediate images and information the step or filter requires.

Algorithm

A stripped down definition of `Algorithm` is shown in Appendix A.2.

An algorithm can then be considered an ordered combination of a number of `AlgorithmStep` and `AlgorithmFilter` processes. The `Algorithm` abstract class allows for easy construction of specific segmentation algorithms, requiring only to be provided with the ordered list of processing steps and a number of instances to run the algorithm on. This information is provided by the specific `Algorithm` subclass implementing the `setup()` method.

The marshalling of instances and invocation of the different processes only needs to be handled once within the abstract class for it to be available to all segmentation algorithms. Before each `AlgorithmProcess` is invoked, each instance is serialised such that only the images necessary for the next process are still in memory.

Also provided is reporting functionality through the observer pattern, allowing for messages, progress and status updates to be listened to, decoupling the running of an algorithm from a GUI or error log, for example.

7.3 Multi-threaded code in Java

One particularly useful advantage of image processing is the parallelisation of the algorithm across the images; with multi-core processors this can reduce the runtime of the algorithm considerably. Working with multi-threaded code in Java is simple, and an example of this is given in Listing 7.6.

```java
import java.util.ArrayList;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

ExecutorService exec = Executors.newFixedThreadPool(
    Runtime.getRuntime().availableProcessors()
);
ArrayList<Callable<Object>> tasks = new ArrayList<Callable<Object>>()
   ;

tasks.add(new Callable<Object>() {
  public Object call() {
    // Do some image processing here
    ...

    // Return something of type Object
    // If the call modifies external data structures it doesn't need
       to
    // return anything
    return null;
  }
});
tasks.add(...);

// Blocks until all tasks return.
exec.invokeAll(tasks);
```

Listing 7.6: Sample for creating multi-threaded code in Java

8 Algorithm Implementation

This section gives a more detailed account of the implementation of the algorithm as presented in Chapter 6, assuming that the reader has read that chapter to gain an overview of the algorithm as a whole.

The implementation processes the images at 50% of their original size to dramatically reduce the computation time without compromising the level of detail.

At the moment there is no integration of a 3D reconstruction into the segmentation pipeline. Instead, the end-users are happy to take the results and render them in 3D in the same software used to perform the manual segmentation.

8.1 As an Instance of the Algorithm Framework

The nuclear envelope algorithm detailed herein is defined as an instance of the algorithm framework presented in Section 7.2. Algorithm steps and filtering processes are referenced to the section in this chapter where they are explained in detail.

NEInstance The nuclear envelope `AlgorithmInstance`, with intermediate images of the seed points, nucleus approximation and the output segmentation.

NEAlgorithm Extends the `Algorithm` base class, with a setup method that loads the 80 slices of the dataset and adds the processing steps listed in the order below for the algorithm.

FindSeedPoints Extends `AlgorithmStep`, Section 8.2.

FindSeedPoints3DConsistency Extends `AlgorithmFilter`, Section 8.3.

FilterSeedPoints Extends `AlgorithmStep`, Section 8.4.

FindNucleus Extends `AlgorithmStep`, Section 8.5.

FindNucleus3DConsistency Extends `AlgorithmFilter`, Section 8.6.

FindNuclearEnvelope Extends `AlgorithmStep`, Section 8.7.

PruneConnectingStructures Extends `AlgorithmStep`, Section 8.8.

FindNuclearEnvelope3DConsistency Extends `AlgorithmFilter`, Section 8.9.

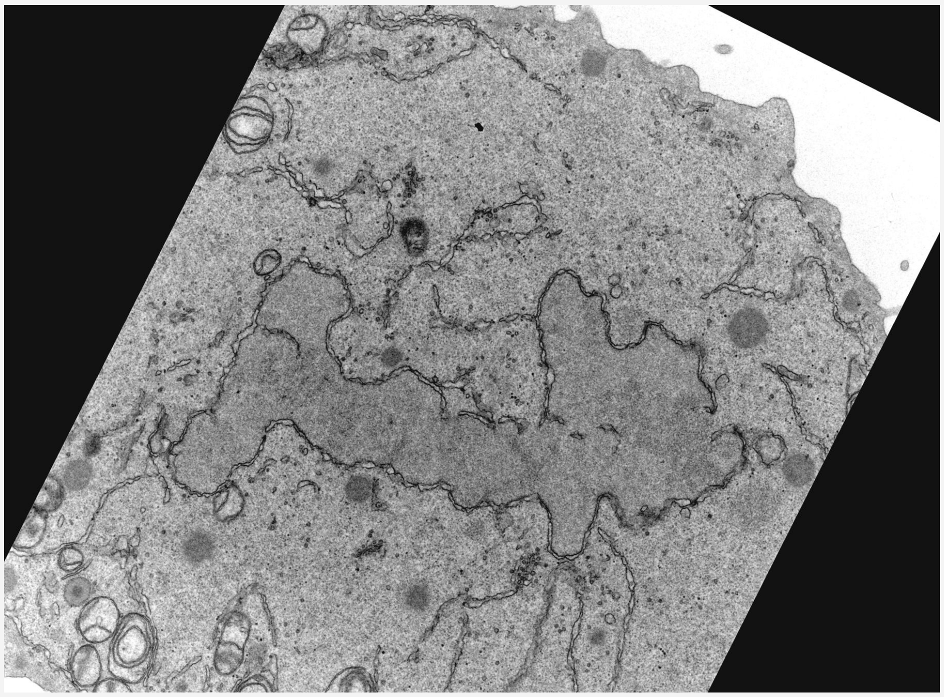


Figure 8.1: Scaled Image

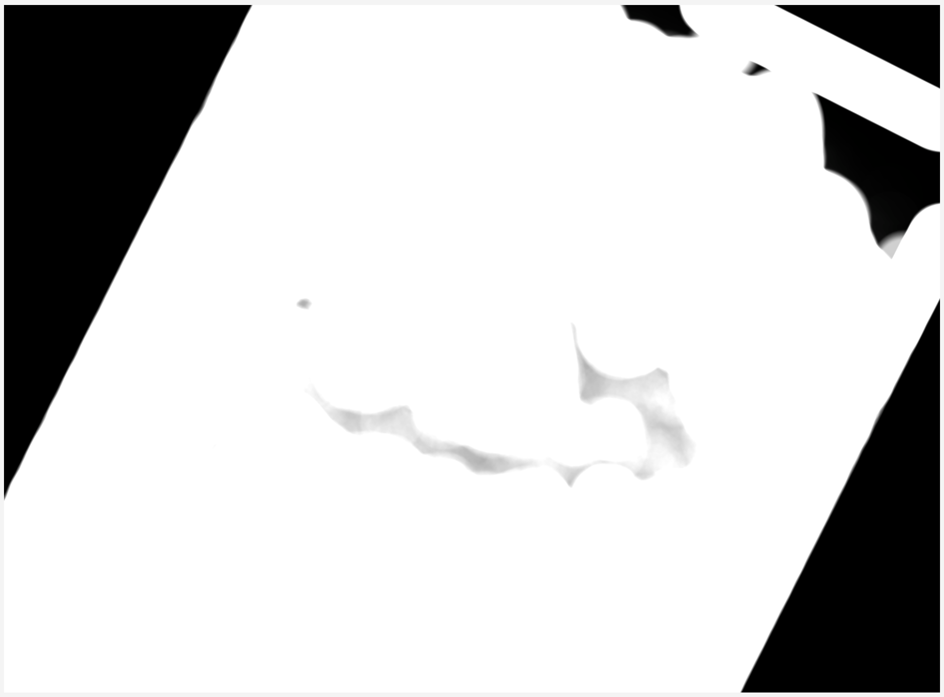


Figure 8.2: Smoothed image after filtering with a large neighbourhood variance filter

8.2 Identifying Nucleus Seed Points

The first step here is to smooth the scaled image with a Gaussian filter with $\sigma = 1$ before running a variance filter over the image with a 60 pixel neighbourhood region. We can see in the final image of this step in Figure 8.2 that some of the nucleus is defined by an area of medium variance. As the aim of this stage is simply to locate some seed points within the nucleus, rather than identify the complete nucleus, as long as we can guarantee the location of these points we can be quite restrictive with the large variance filter at this stage.

We also notice that there are areas towards the edges of the image that have very low variance. In the cases of the black triangles at the corners of the image, these are introduced when the images are registered. They can be removed from this step of the algorithm by noting that these areas in the image have an intensity value of 15, the lowest within the image, and that this value rarely, if ever, occurs within the cell and the area of interest.

The more natural looking black areas towards the top right of the image are caused by the area of the slice in the original image that does not contain any of the cell. This area sometimes has a small amount of noise and some other small blobs of matter, so to remove them we pass a maximum filter with a neighbourhood size of 20 over the original image. The result of this is shown in Figure 8.3, and pixels in this result that have an intensity value greater than 250 are considered to have come from these areas and are discounted as seed points. The low variance and medium grey-level values of the area of interest again makes this a safe assumption.

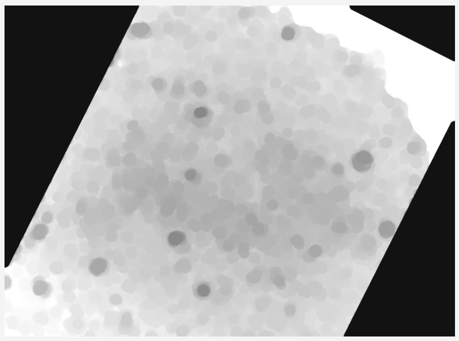


Figure 8.3: Scaled image filtered by a maximum filter of neighbourhood size 20. We can see that the area to the top right has intensity greater than 250 and will be discounted.

To then transform the image in Figure 8.2 into a set of seed points we loop over all the non-maximal pixels in the image, those pixels whose variance is less than 255. A further refinement step is performed, where a grey-level co-occurrence matrix is computed over a neighbourhood of 10 pixels, with co-occurrences being counted at a 3 pixel distance along the positive x-axis direction. This is a fairly time-consuming operation and so is only computed at pixels that are candidates for seed points that were not excluded in previous steps. The contrast of the co-occurrence matrix is computed and thresholded as a further test that candidate seed points must pass. Whilst this may discount valid seed points, in practice enough are retained such that the algorithm performance is not impacted whilst the robustness of this step is increased with the addition of a further test.

There is an argument to present here that only one direction of co-occurrence needs to be considered. The texture within the nucleus has low variance and no particular preference to direction, and in practice the thresholds for the contrast values for this test are chosen and passed by nearly all possible directions of co-occurrence inside the nucleus. We can therefore save time by only evaluating one co-occurrence matrix.

If a pixel passes all these tests then it is considered to be a seed point for the next step of the algorithm. The final image that represents the seed points is shown in Figure 8.4.

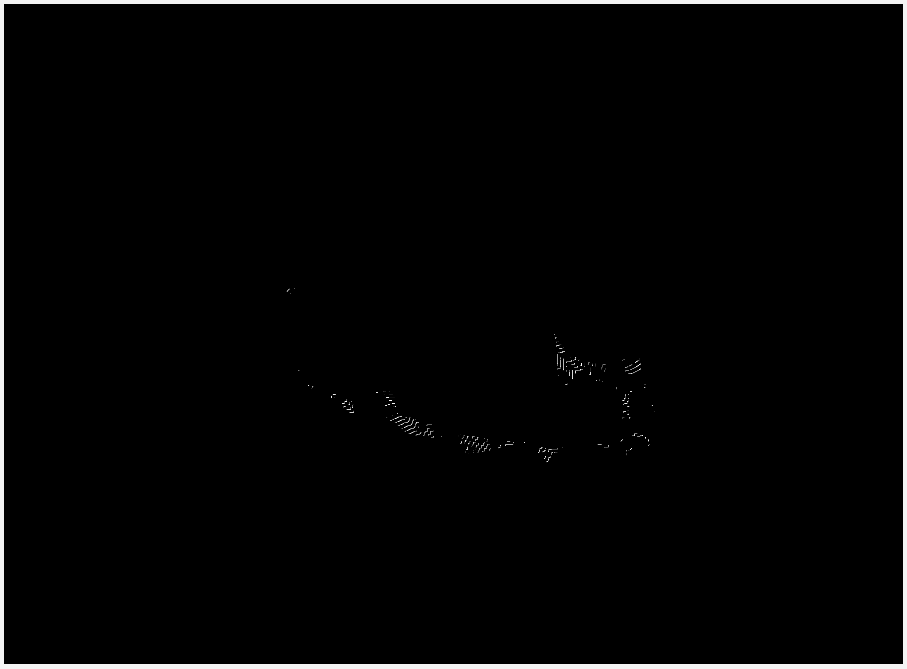


Figure 8.4: Seed points

8.3 3D Consistency of Nucleus Seed Points

The first of the 3D consistency steps is to consider the location of seed points between neighbouring slices, as sometimes the identification step can return seed points that are outside the nucleus that should be discounted. Oftentimes these are from similar patches of texture to the inside of the nucleus that occur in the cytoplasm, and given the grey-scale nature of the image it is very difficult to tell these apart. However, the busyness of the surrounding structures compared to the inside of the nucleus is what enables a 3D consistency algorithm to perform well. These erroneous seed points rarely, if at all, appear in neighbouring slices.

To begin the 3D consistency a measure of support from other slices is calculated, such that each pixel in a particular slice has a value between 0 and 1, showing how much evidence it has from these other slices for being a seed point. For a particular slice we consider the 3 slices to either side in the whole stack to compute the measure of likelihood. The stack does not wrap around, so in the edge cases in the stack we use as many slices as possible.

To account for the movement of the nucleus and the fact that we have a set of points rather than regions, a variance filter of neighbourhood size 20 is passed over all the seed point images as a preprocessing step. The result of this is shown in Figure 8.7, and we can see that this moves towards a region based representation that allows for movement in the nuclear envelope between slices.

The likelihood of a pixel in a particular slice is then calculated as the sum of the proportions of white pixels in a neighbourhood centred at the same pixel in the neighbouring slices. This use of the neighbourhood highlights the need to move from a point to a region based representation of the seed points. To account for the movement of the nucleus, the neighbourhood size that is considered is increased with distance (in terms of slices) from the focus slice. Furthermore slices closer to the focus slice are also weighted more than slices further away, as it is important to consider those nearer slices to be “better” representations of likelihood for the focus image, noting as well that the movement between immediately neighbouring slices is minimal.

As a trivial example, suppose that we are calculating the likelihood of a seed point at (2,2), and we consider only one neighbour. If this neighbour has no seed points in a 3x3 neighbourhood centred at (2,2), then it provides little evidence that (2,2) is a valid seed point. However, if this neighbour has a complete 3x3 neighbourhood of seed points centred at (2,2), then that is very strong evidence that (2,2) is a valid seed point. The same happens

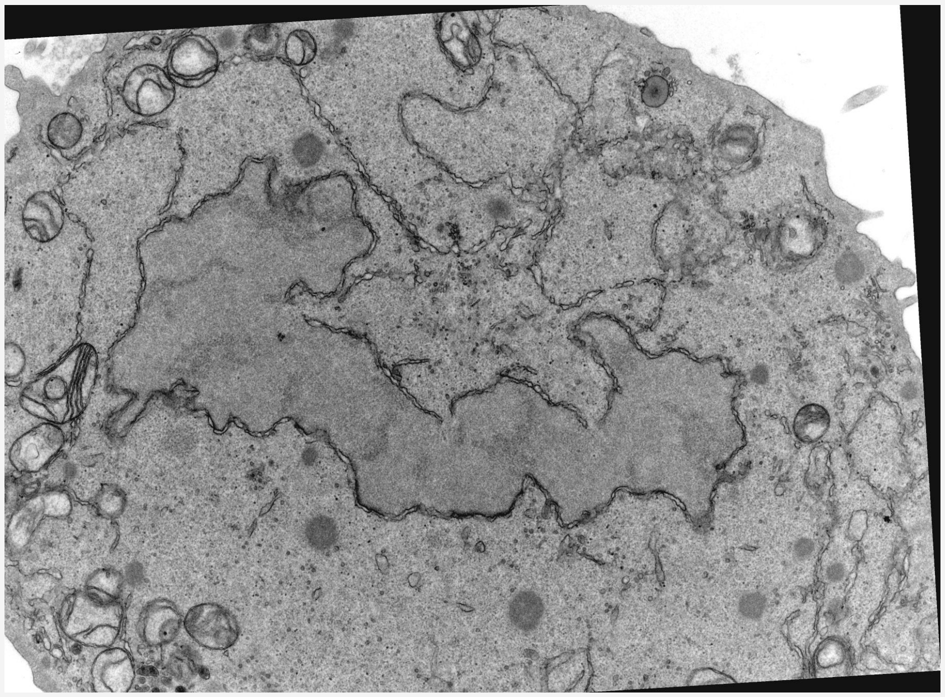


Figure 8.5: Scaled original image of slice 0049

Figure 8.6: Set of seed points output from previous step

here, only with more neighbours and larger neighbourhoods.

Putting all this together gives a likelihood value for each seed point which can then be suitably thresholded to eliminate those that have little evidence. The immediate neighbour slices to slice 0049 are shown in Appendix B.1. The likelihood map of support from neighbouring slices for slice 0049 is shown in Figure 8.8, and a comparison with the set of seed points in Figure 8.6 shows that there is little evidence for those seed points that lie outside the nucleus, and the evidence that there is to the bottom left of the nucleus also falls below the threshold.

In some cases the method for finding seed points in individual slices can fail to produce many good seed points, and we can effectively increase their quality and quantity by interpolating seed points of high likelihood from neighbouring slices. For this application having a complete likelihood map would be particularly useful; it could be thresholded to see where there is very high evidence for seed points from neighbours and add all those places onto the current set of points. However this is an expensive and wasteful operation, and could be simplified by only considering seed points that are present in either immediate neighbours of the focus slice. It is reasonable that the definition of "very high evidence" necessary for this interpolation requires that both immediate neighbours contribute highly to this evidence, and so we do not lose any information by considering points only in the union of the two neighbours rather than the whole image.

The result of this step can be seen when comparing Figure 8.9 with Figure 8.10. The set of seed points is much less sparse and covers more of the nucleus area than before, and the advantages of this will become apparent in further algorithm steps.

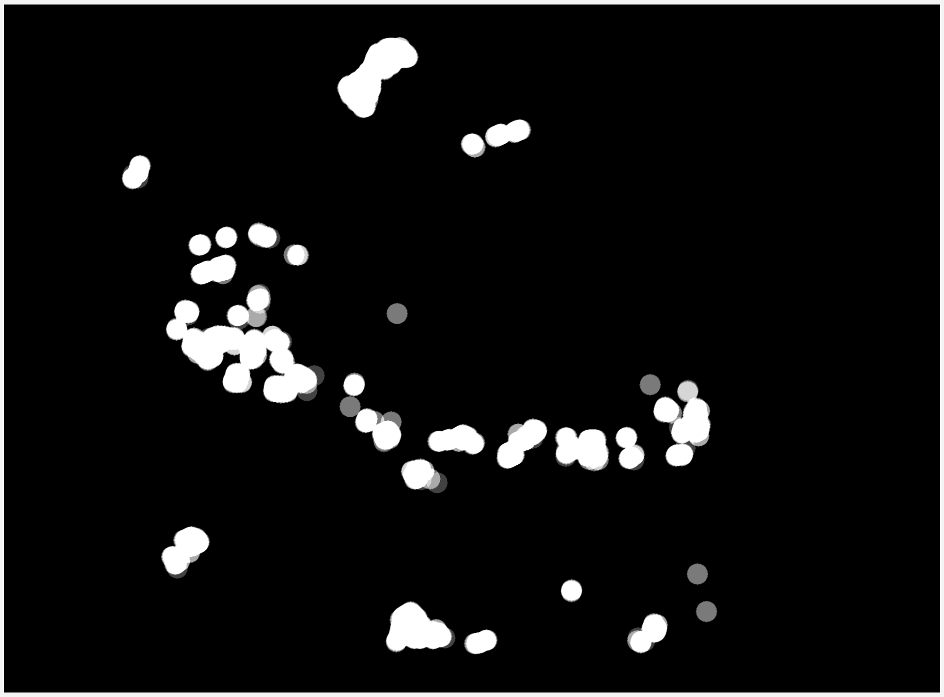


Figure 8.7: Seed point image after variance filtering

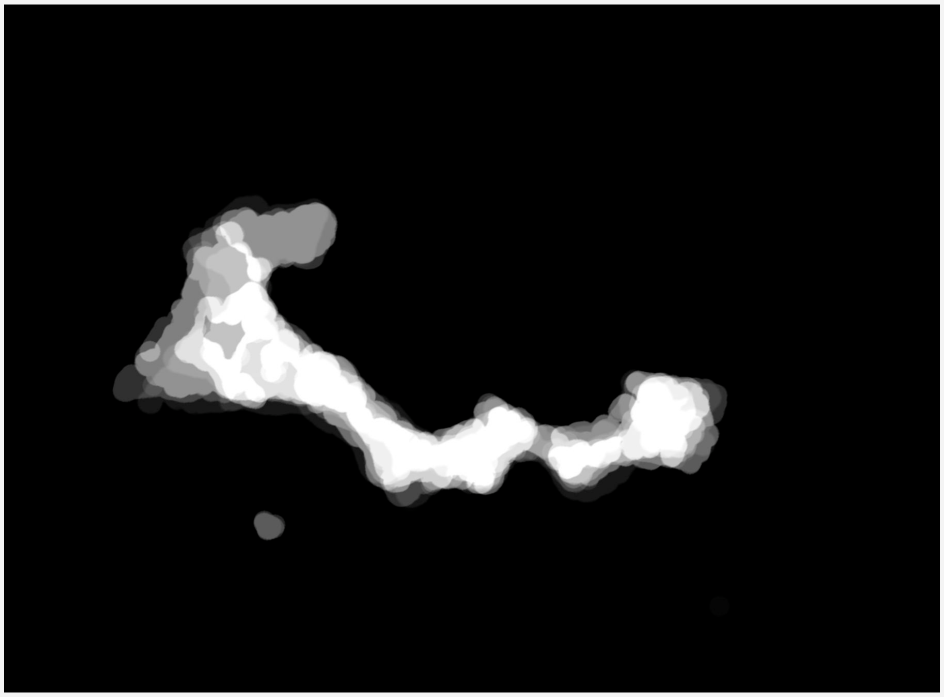


Figure 8.8: Likelihood map for seed points in slice 0049.

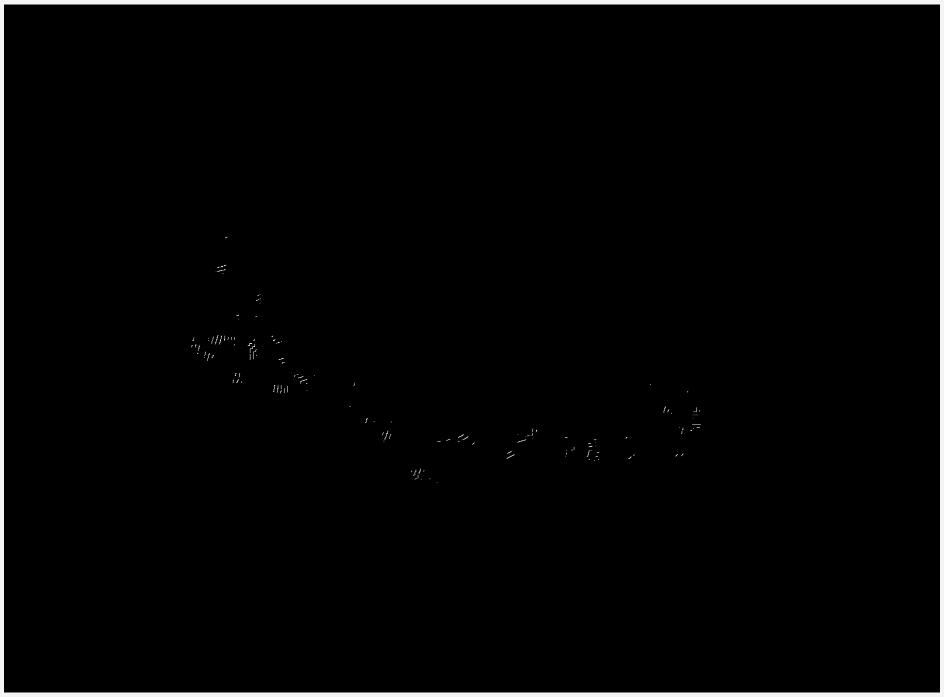


Figure 8.9: Seed points for slice 0049 after ensuring consistency with neighbouring slices.

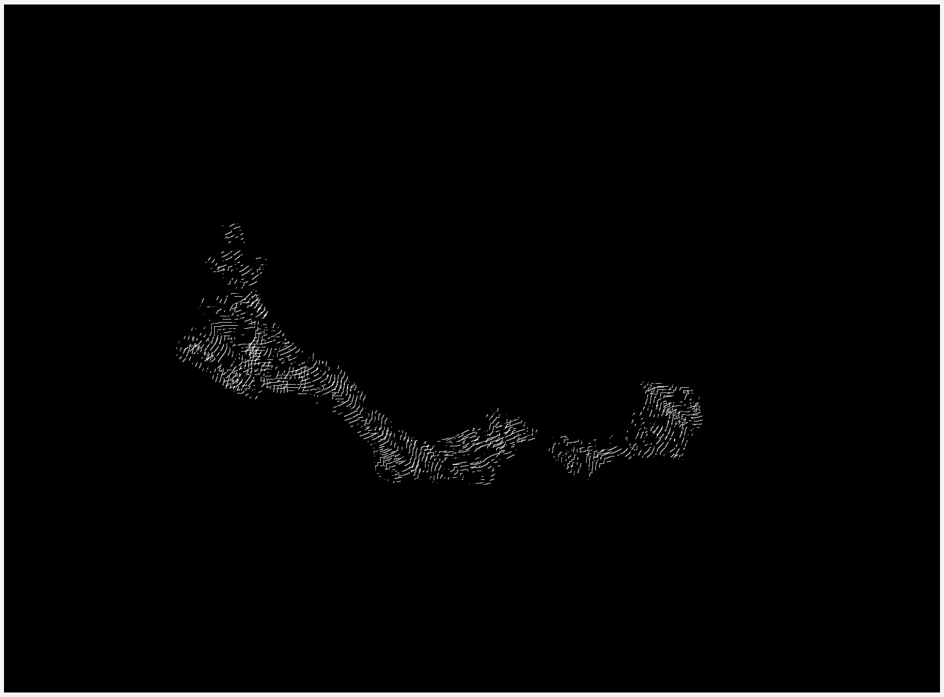


Figure 8.10: Seed points for slice 0049 after interpolating seed points of high likelihood from neighbouring slices.

8.4 Filtering Nucleus Seed Points

It may seem a little strange that the previous seed point consistency step had an interpolation aspect to improve the number and quality of seed points, and now there is a step that is defined as filtering seed points. The seed points are used as starting points for a flood fill algorithm, and so the important aspect is to try and obtain seed points in as many different regions of the nucleus as possible. How many seed points there are in these regions is not terribly important; in effect we are trying to ensure as great a global coverage as possible, with local coverage not being as important. The flood filling algorithm to find the nucleus deals with local coverage well, but can be impeded by some structures or noise within the nucleus that prevent full global coverage.

Effectively, if the number of seed points can be reduced whilst still retaining a global coverage the computation time of the flood filling algorithm can be reduced. However, this advantage is only attainable if the filtering step can be performed in less time than is saved by having less seed points. Fortunately there is such an operation that retains global coverage whilst reducing local coverage; the skeleton transform.

By performing a skeletonisation of the seed point image we obtain a much smaller representation of the seed points. A further reduction in size can be achieved by taking just the junction and end pixels of the skeleton, and ignoring the pixels that form branches between them in the skeleton. Again this is an acceptable step because we retain the important global information that the skeleton provides and remove the local connectivity information.

For the particular case of slice 0049, 13142 seed points remain after the 3D consistency step, which are then filtered down to 2510, an 80% reduction, and in practice the reduction across all images is between 70% and 90%. This process works very well as the skeletonisation is a fast operation, and the savings in later stages can be considerable.

Exemplifying the output of this stage is omitted here, purely because the seed points cannot be seen on paper.

8.5 Finding the Nucleus

Another step of great importance to the success of the segmentation is that of obtaining an approximation to the inside of the nucleus, primarily identified by its texture of low variance compared to other structures within the cell. An identification of these regions utilises this low variance and the seed points from previous steps using a flood fill algorithm to obtain an approximation to the nucleus.

A naïve flood fill criteria would be that of a grey-level threshold defined on the original image, however this is susceptible to leaking outside the nucleus if there are gaps in the envelope.

The solution is to transform the original image into an alternative representation based upon local neighbourhood variance. A preprocessing Gaussian smoothing is performed at $\sigma = 2.1$, which has the effect of smoothing the inside of the nucleus whilst retaining much of the variance outside of the nucleus due to the cluttered structures. After the Gaussian filter we perform a first iteration of a variance filter with a 6 pixel neighbourhood and a minimum filter with a 5 pixel neighbourhood; the results of both stages are shown in Figures 8.11 and 8.12 respectively. The effect of the the variance filter followed by the minimum is to first find the areas of low variance within the nucleus, but then use a minimum filter to propagate the low variance values back towards the nuclear envelope. The minimum also removes small areas of high variance within the nucleus.

A second pair of variance and minimum filters are then used to amplify the low variance values within the nucleus, making them more distinct from those variance values outside the nucleus. The second variance filter has a neighbourhood of size 12 pixels, and the minimum a neighbourhood of size 6 pixels. The intuition behind this is that the first minimum filter (Figure 8.12) reduces the amount of high variance locations inside the nucleus, but the outer structures retain a higher variance in general. Taking the variance of this image over a larger neighbourhood provides greater definition to the texture within the nucleus and accentuates the boundary between the nucleus texture and the cytoplasm. The result of this step is shown in Figure 8.14, and we can see that there is a much better definition to the area inside the nucleus.

The result is inverted as a final step to enable the use of the flood fill algorithm, FindConnectedRegions (Section 7.1.4), that finds regions above a given lower threshold. A simple application of this algorithm would

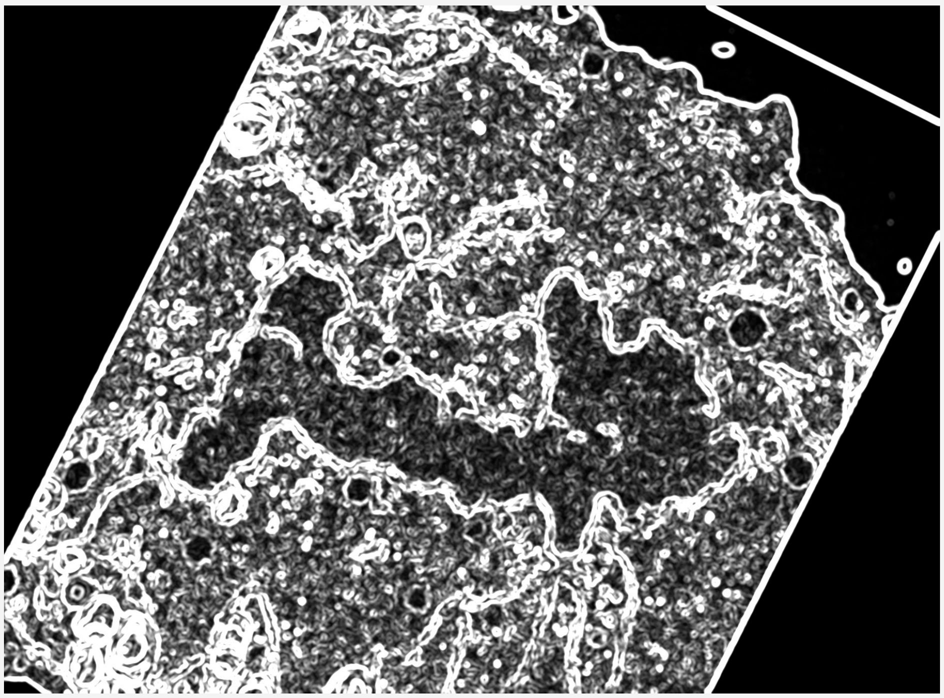


Figure 8.11: Variance filter of 6 pixel neighbourhood passed over original of slice 0064 processed with a Gaussian filter.

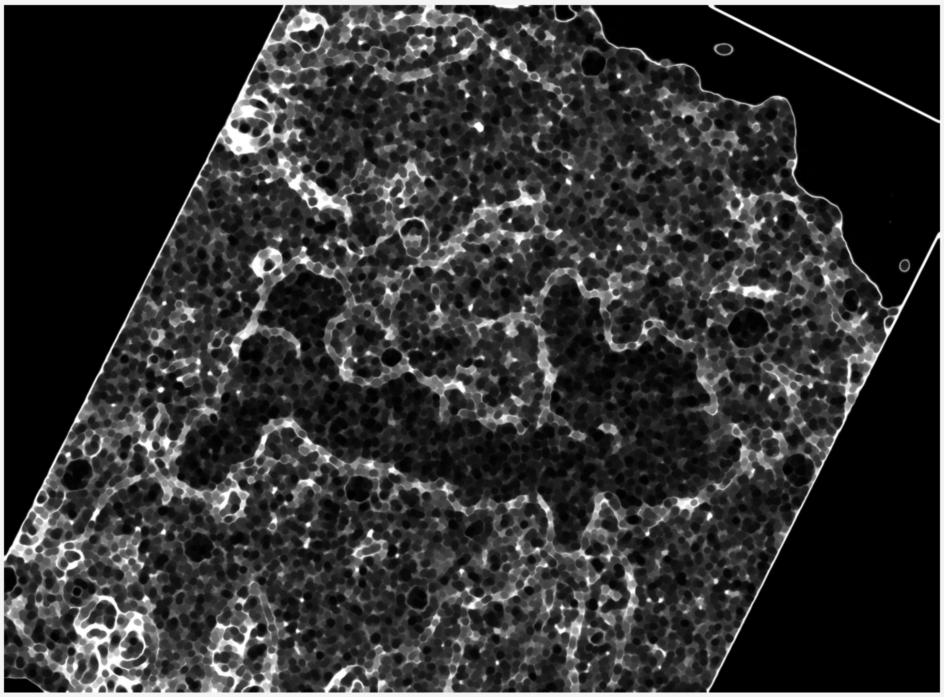


Figure 8.12: Minimum filter of 5 pixel neighbourhood passed over 6 pixel neighbourhood variance filtered image.

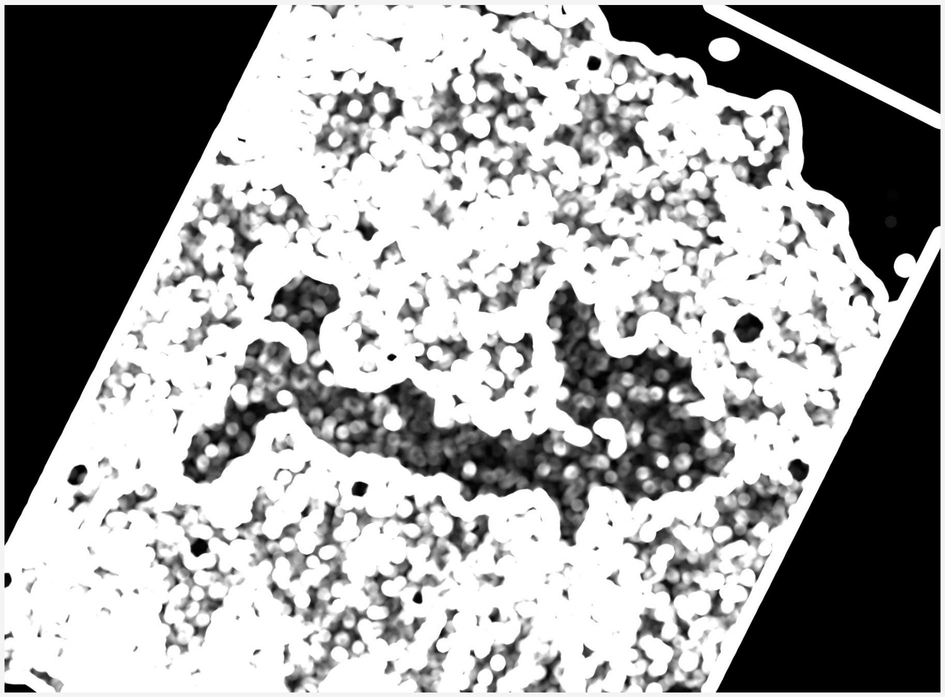


Figure 8.13: Variance filter of 12 pixel neighbourhood passed over output of previous variance/minimum step

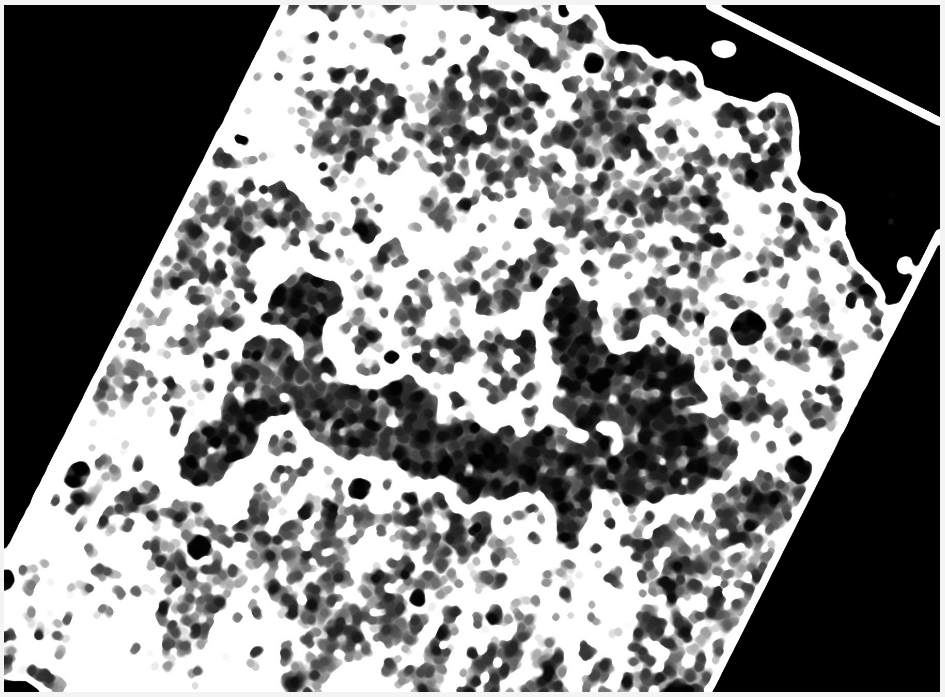


Figure 8.14: Minimum filter of 6 pixel neighbourhood passed over variance filtered image

find many regions above this threshold which are not part of the nucleus, so this is where the seed points found earlier come into play.

Each seed point is considered in turn, with a flood fill from it above a threshold of 180 on the result of the two variance/minimum filter steps, remembering that the image has been inverted so areas above 180 have lower variance in the original image. The combination of all found regions gives an approximation to the nucleus, and it is this step that shows the importance of obtaining the best global coverage of seed points possible. Additionally, in order to speed up the computation, if a seed point is found to be contained within a region that is already discovered then we skip the flood fill from this point; the region that would be found is the same as that for any other point within the region, so it is wasted computation.

The output of the union of all regions found is shown in Figure 8.15, and whilst this captures the overall shape there are a lot of holes, disconnected parts and it is overall not very smooth. A solution to this, shown in Figure 8.16, is to perform a morphological closing operator with a circular structuring element of radius 12 pixels, which closes the holes and smooths the boundary. Any holes that still remain after this operation are considered to be legitimate holes as a result of the 3D nature of the nucleus (Section 5.3).

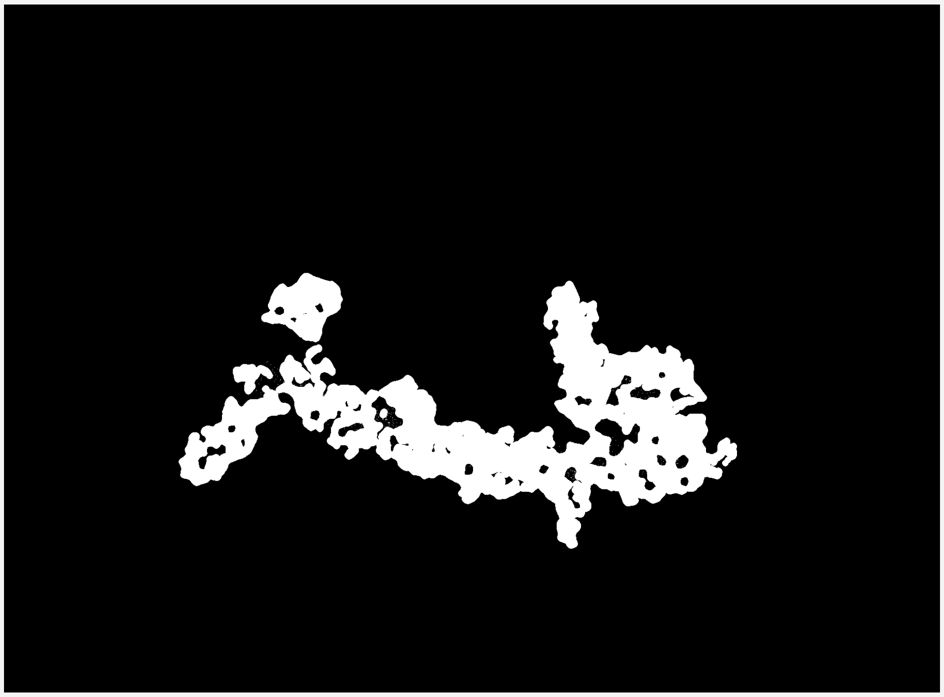


Figure 8.15: Combination of all regions found by a flood fill from all seed points

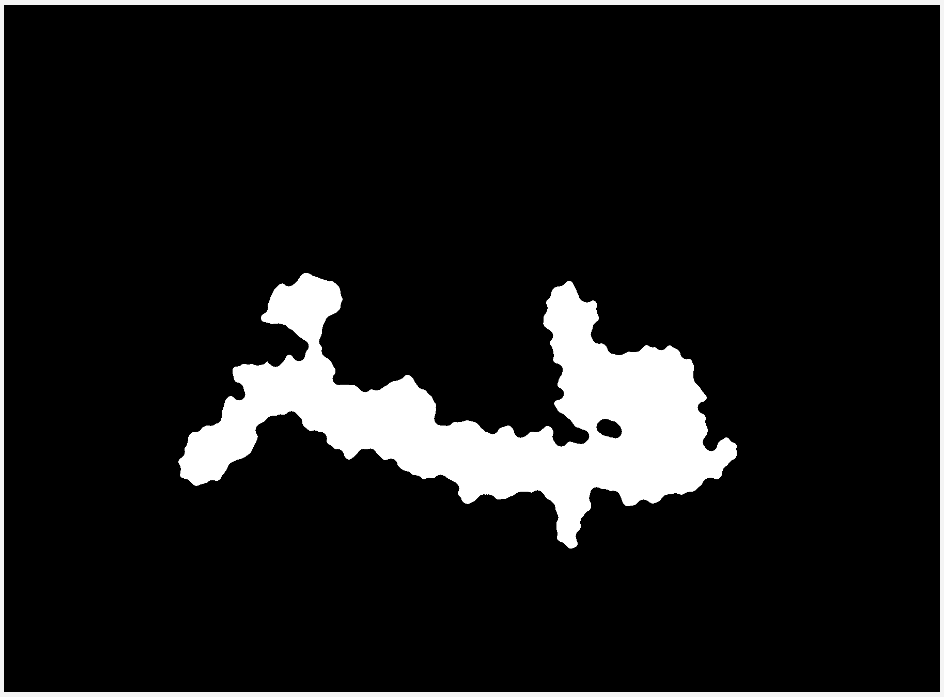


Figure 8.16: Morphological closing operator closes holes within the approximation

8.6 3D Consistency of the Nucleus Approximation

Similarly to the 3D consistency of seed points, we can use the same idea of support from surrounding slices to refine the nucleus approximations found in the previous step. This becomes useful in cases where some noise or dark patches within the nucleus prevent the flood fill reaching all parts of the nucleus; again demonstrating the importance of finding as many seed points as possible.

A naïve way to do such consistency is purely to take the intersection of the two immediate neighbours of a particular slice and add it onto the nucleus approximation for that slice. This method does however break down when locally the nuclear envelope "contracts" from one neighbour to the focus slice before moving back out again to the next neighbour; in this case the intersection may have areas that would be on or outside the nuclear envelope for the focus slice.

We can however still utilise the intersection of the immediate neighbouring slices by using the points it defines as those we examine for significant support from further neighbouring slices. The idea is the same as the 3D consistency, where points with significant support in belonging to the nucleus move on to the next step. However, instead of simply adding these points onto the nucleus approximation we can be a bit more accurate.

The idea here is therefore to take those points that have significant evidence for being part of the nucleus approximation and use them as seed points for a second set of flood fills in exactly the same conditions as the last algorithm step. This is essentially finding more seed points for the nucleus approximation, but this time from 3D consistency of the nucleus approximation itself. Again, as used in the filtering of seed points, a skeleton transform is used on the set of seeds to reduce the local clustering but retain the same global coverage before the flood fills are run.

Figure 6.11c shows the nucleus approximation in Figure 8.16 after using the 3D consistency steps to improve its accuracy. There is a noticeable addition to the approximation for this slice. Looking back to Figure 8.1 we can see that the reason this area wasn't captured to begin with was because of the dark patches in the original slice causing too high a variance for the area to be flood filled in the nucleus approximation step. The nucleus approximations for the immediate neighbours of slice 0064 are shown in Appendix B.2.

8.7 Obtaining an Approximation to the Nuclear Envelope

Now that we have a good approximation to the inside of the nucleus for all slices we can use it to locate the nuclear envelope. However, it is not suitable to simply expand the boundary by a set amount and assume that can be used as the nuclear envelope segmentation. It is not possible in all cases to obtain a nucleus approximation that is accurate enough to do so.

Instead, we can look at the outline of the nucleus approximation and consider the local edge normal to it; the directions that propagate out of the nucleus perpendicular to its outline. If we traverse a "ray" that is cast along this direction and plot the grey values encountered we should find that they are roughly an inverted bell curve, with salient points in the curve where the ray crosses into and out of the nuclear envelope. By thresholding this plot we could find the approximate locations along the ray where it entered and exited the nuclear envelope, and use these to define a mid-point which can be added to a second, different set of seed points that lie within the nuclear envelope. There may be multiple threshold crossings within the ray depending on the structures in a particular image, however given that the nucleus approximation is always contained within the nuclear envelope the first set of crossings that we find when moving away from the nucleus is taken as the nuclear envelope.

A small but useful amount of preprocessing is done to improve the reliability and ease of detecting points that lie within the nuclear envelope, the purpose of which is to close the space between well defined nuclear envelope edge and create just one distinct edge. To do so a band-pass filter is used, where structures smaller than 7 pixels and larger than 90 pixels are filtered. This has the effect of smoothing the dark areas that define the nuclear envelope without drastically blurring its shape. The effect of this, coupled with a contrast filter, is shown in Figure 8.17.

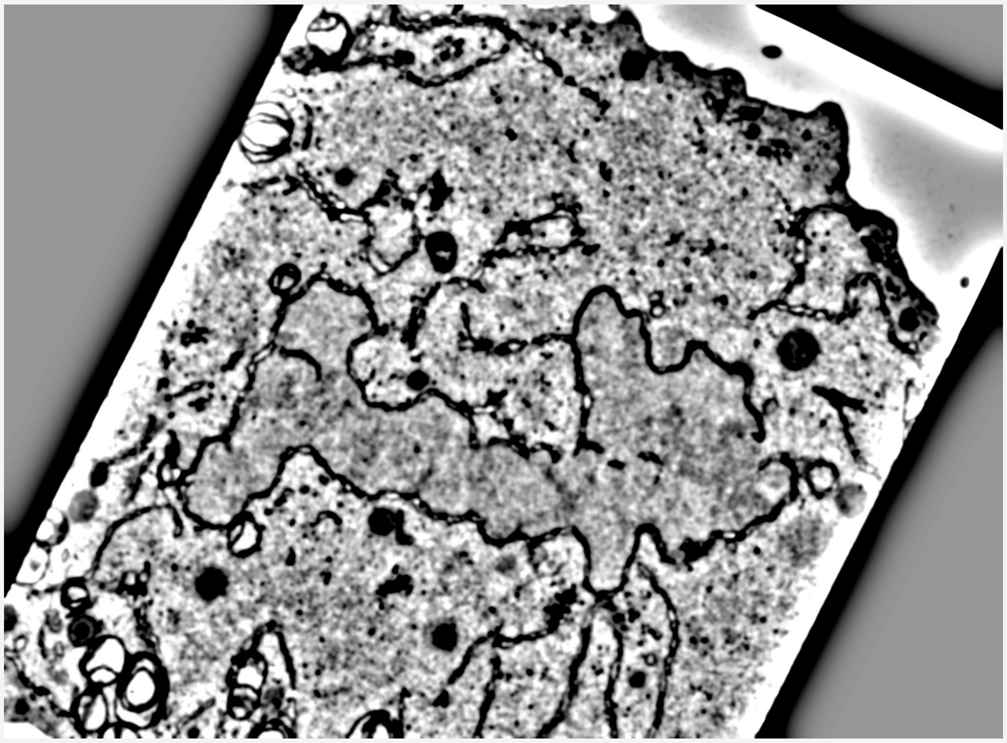


Figure 8.17: Slice 64 preprocessed with a band-pass and contrast filter

The ImageJ Wand is then used to select a region connected to a particular point, and we can again use the earlier seed points as this required input. Given a region, the wand object returns the set of points that lie on the boundary of the found region, which is exactly what we need to perform the expansion of the outline into the nuclear envelope.

In order to find an estimate to the gradient vector that we need to perform this expansion, the finite different approximation is used over a point's neighbour's x and y co-ordinates. This gives a vector that is approximately the local edge orientation at the particular point, and the perpendicular vector can easily be found from this. This method gives a vector that defines a direction, but it does not guarantee whether the ray will travel into the nucleus approximation or out into the nuclear envelope. A simple traversal of the ray in either direction confirms whether the ray goes from the boundary point to the inside of the nucleus, and if so the direction vector is negated to point out towards the nuclear envelope.

Now for each point we have a direction vector to travel out towards the nuclear envelope from, and we can begin to traverse them to find the nuclear envelope. A gradient map of the image is created by passing the original image through a Canny edge detector; peaks in the gradient values along a ray indicate crossing into or out of structures.

Each ray is traversed until it reaches length 50 at which point it is terminated; this prevents too many additional structures within the cell from being associated with seed points when they are actually too far from the original nucleus approximation. This does however require the approximation of the nucleus to be sufficiently close to the nuclear envelope, and in practice this proves to be the case. An examples of rays being traversed in this manner is shown in Figure 8.18.

To locate a seed point inside the nuclear envelope we find the midpoint along the ray between the peaks in the gradient value where it enters and exits the nuclear envelope. In the example in Figure 8.19 seed points are identified by white dots, overlaid onto the nuclear envelope. The seed points found by this method are used as the input to another flood fill algorithm that can operate upon multiple seed points, the Fast Marching algorithm. A processed image similar to the one in Figure 8.17 with a slightly lower contrast is used as the source, and a

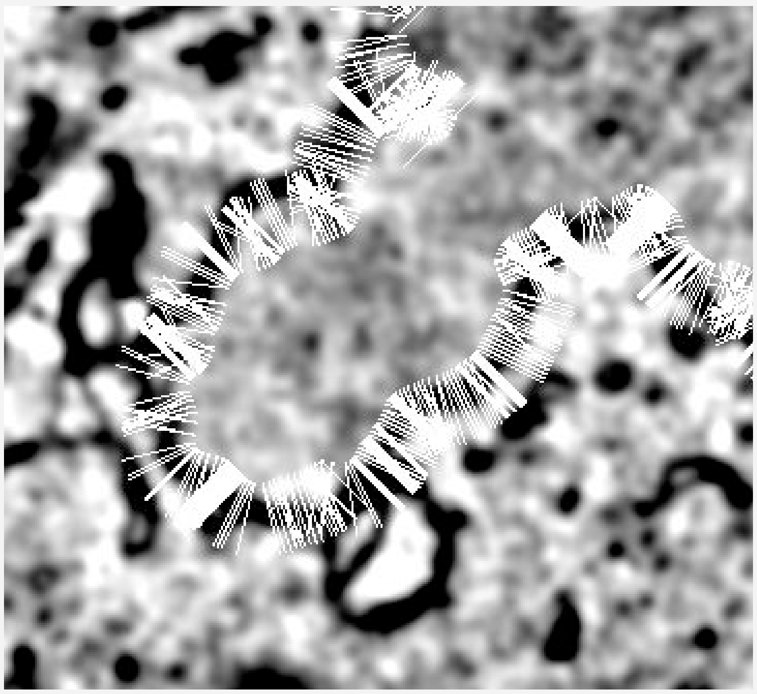


Figure 8.18: Rays traversed when searching for seed points within the nuclear envelope

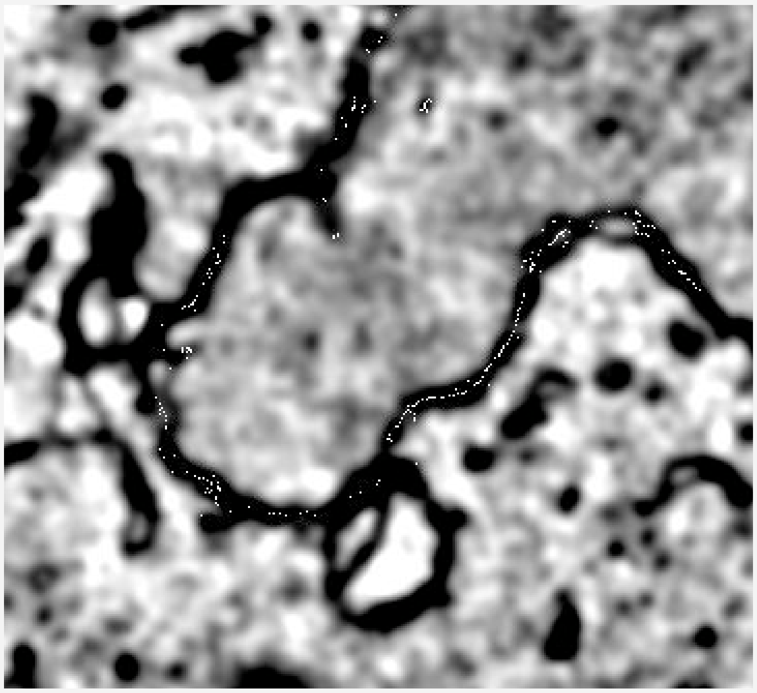


Figure 8.19: Location of seed points within the nuclear envelope

threshold specifies that all grey-level values lower than 60 should be captured in the regions.

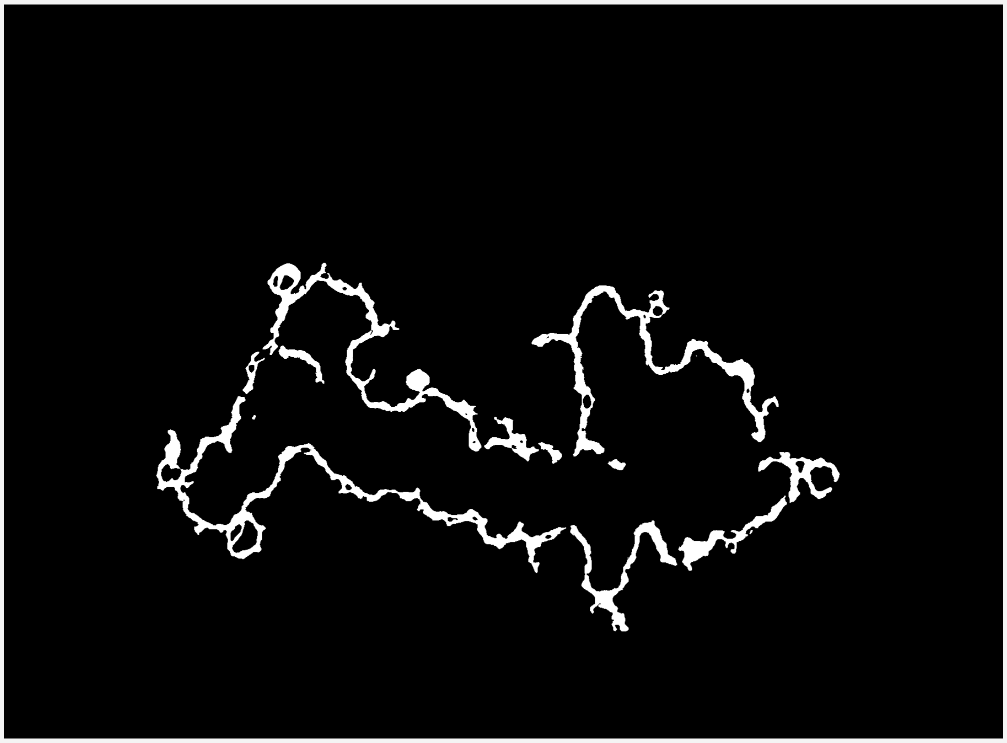


Figure 8.20: Result of flood filling from nuclear envelope seed points

A slight caveat to the use of the ImageJ Wand to find the outline of a region is that it ignores any holes within the region, something which due to the 3D nature of the nucleus (Section 5.3) we need to account for. To do this we use the wand's selection and search it for black pixels; these black pixels become the source point for a further Wand operation which gives us the outline of the hole. From there the same boundary expansion process is applied to find any seed points in the hole inside the nucleus approximation. These holes are typically small, so the length of the vector projected out from the boundary can be made shorter than those of the vectors for the main nuclear envelope. We are also looking for points inside a hole, so typically there is some unnecessary overlapping between vectors and therefore the shorter the vectors are made the less wasted computation there is.

8.8 Pruning Connected Structures

One problem that is apparent from the result shown in Figure 8.20 is that there are sometimes structures that should not be segmented as nuclear envelope in the segmentation. This is due to the fact that, even if not physically separated, the discrete nature of the visualisation means that they are not visually separable and little can be done to restrict the flood fill from spilling over into these areas. Furthermore, a more restrictive flood fill may solve this problem but could potentially prevent the segmentation of all the nuclear envelope in cases where the nucleus approximation and nuclear envelope seed points were not quite good enough. It is better to achieve an over-segmentation with the guarantee that all nuclear envelope has been found, and subsequently refine the segmentation, than to run the risk of not locating all the nuclear envelope.

This final step in the pipeline does just that; it prunes off the structures that are visually connected to the nuclear envelope but should not be part of the segmentation. It uses the intuition that the nuclear envelope is the closest structure to the approximation of the nucleus found earlier, and therefore any structures that have other structures in between themselves and the nucleus approximation should be removed.

To find any structures between a particular structure and the nucleus approximation, we need to find a vector between a point on the structure and the nearest point to it on the nucleus. We can find a suitable approximation to this vector using the differential of a distance map applied to the nucleus approximation. The distance map for the nucleus approximation of slice 0064 is shown in Figure 8.21, where the inverse of the individual pixel intensity gives the minimum distance of that pixel from the nucleus.

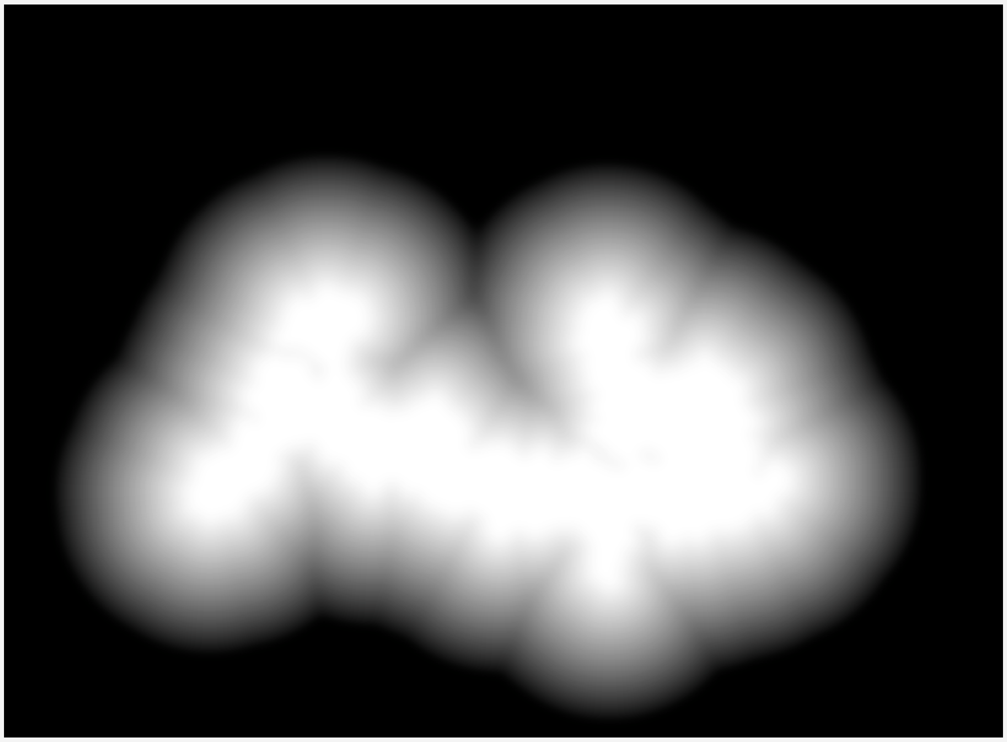


Figure 8.21: Distance map where pixel intensity shows inverse distance to the nucleus approximation.

By finding the gradients of the distance map we can obtain an approximation to the vector from any point to its nearest point on the nucleus approximation. These gradients are calculated using the ImageJ Differentials plugin, which provides methods to calculate differentials in the x and y directions separately.

Considering the projection of every pixel in the over-segmentation onto the nucleus approximation would be an expensive and unnecessary operation. A much better approach is to again utilise the skeleton of the over-segmentation and use the analysis of the skeleton to find its end and junction pixels. These are the important points to consider, and we can subsequently discount complete branches of the skeleton based on the status of the pixels at either end of the branch.

The basic idea is therefore, for each junction or end pixel (henceforth termed a vertex), traverse the vector from its location to the nucleus approximation and:

- If a maximum distance along the vector has been reached and no structure has been encountered then discount the vertex.
- If the vector crosses another part of the skeleton then it is a connected structure, not nuclear envelope, and is discounted.
- If the first structure encountered is the nucleus approximation then this is a valid part of the nuclear envelope, so preserve that vertex.

The example in Figure 8.22 makes this clear. The projection of the vertices to the top left of the image intersect with another part of the skeleton of the over-segmentation, and as such these vertices are discounted. Those vertices that project onto the nucleus without crossing another part of the skeleton are kept in the skeleton structure.

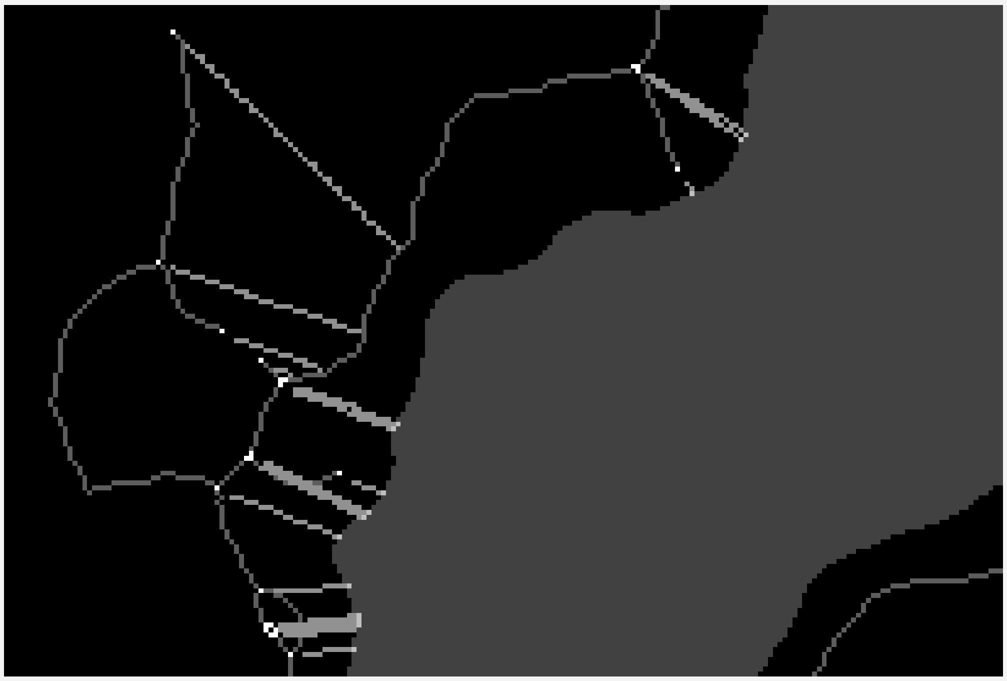


Figure 8.22: Discounting of skeleton vertices based on crossing structures on path to nucleus

This algorithm can be improved in practice by making two further tweaks; the first of which is to remove some of the structure in the local area around a vertex. The purpose of this is to avoid the case where the projection of a vertex crosses a branch of the skeleton that is connected to itself. Branches connected to immediate vertex neighbours of a particular vertex are also removed from the skeleton if the neighbour is at a similar depth to the vertex currently being considered. The removal of these structures when considering each vertex works primarily due to the intuition that the vertices that lie on the nuclear envelope are typically at a similar depth value, and the connecting structures that we are trying to filter out typically join onto the nuclear envelope at an angle perpendicular to it.

The second tweak to improve the quality of the results is to consider a number of different vectors from each vertex to the nucleus by performing a number of rotations of the vector found by the method described above. Each rotation is found by the standard matrix equation for rotation of a vector by an angle θ:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Using the rotation equation we can perform a projection of the vertex by rotating the initial vector through angles in the range -40 to 40 degrees, in steps of 10 degrees. If any of the projections intersect with the nucleus then we consider the vertex to be valid and retain it in the skeleton, otherwise if no projections reach the nucleus then the vertex is deleted as before. The effect of this is shown in Figure 8.23.

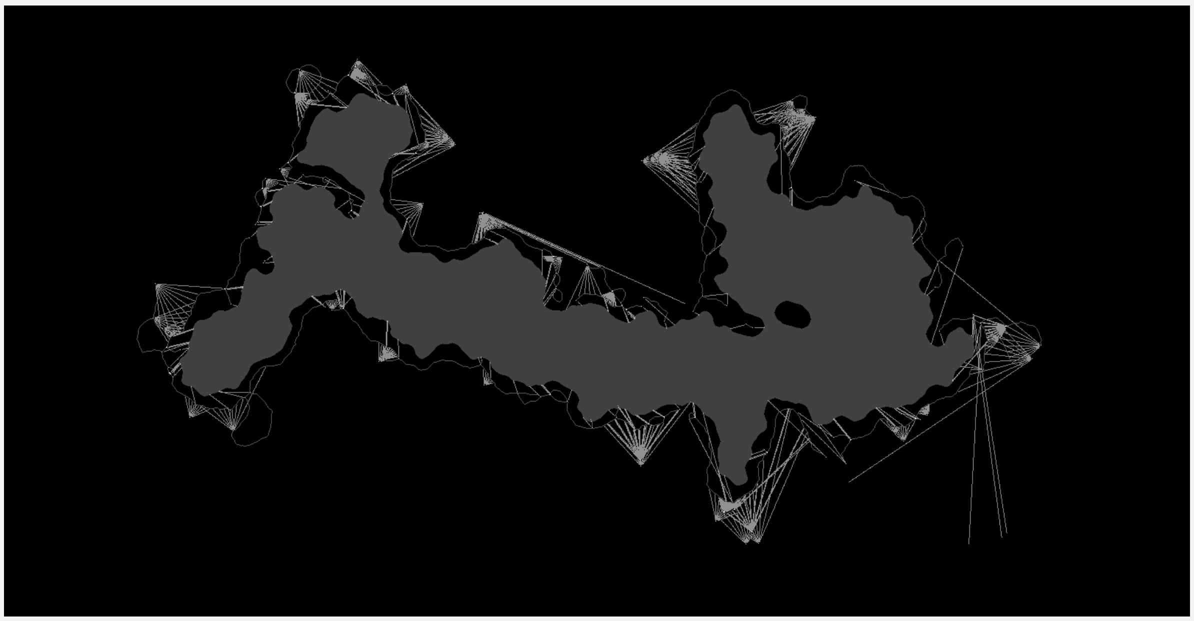


Figure 8.23: Vertex projection algorithm operating on a number of rotations of the initial vector

8.9 3D Interpolation of Corrupted Slices

After manual corrections have been applied to the non-corrupted slices we can use these to estimate the nuclear envelope of corrupted slices as an initialisation of a further set of manual corrections. Several techniques have been considered for characterising the movement of the nuclear envelope between two slices:

- **Optical flow** techniques attempt to estimate a velocity vector at each point in one image based upon the changes in pixel intensity between images, which gives the location that pixel moves to in the second image.
- **Graph matching** attempts to find the best match, or transformation, between two graphs. We could define a set of graphs of one nuclear envelope segmentation as its skeleton and attempt to match vertices and end-points to other skeletons of segmentations.

Both techniques have difficulties with the images for which we are trying to find the transformations between. Optical flow is difficult to compute on binary images, and it is arguable whether the results from the original greyscale images would be accurate enough due to the number of similar greyscale values and structures in the images. It is also a reasonably expensive computation to perform.

The skeletonisation of the binary images that represent the segmentation of the nuclear envelope is a fast operation and its analysis can give a graphical representation of its end, edge and junction points. The problem with graph matching of these skeletons is that it will be sensitive to changes in topology, for example where a gap is present in one segmentation but not in the adjacent slice. It will also be sensitive to where the vertices of the graph are located; small changes in nuclear envelope thickness can cause a vertex to be found in one image with none nearby in an adjacent image. For these reasons I believe the results obtained through such a method may not be of sufficient quality.

A much simpler method has been devised that utilises the distance transformation from the object pixels of the nuclear envelope; the results are good, and it is unlikely that the more advanced methods considered will do much better. In any case some manual changes will be necessary so a sacrifice of accuracy for a faster and simpler method is acceptable. This estimation again requires the assumption of minimum movement of the nuclear envelope.

The first step is to calculate the distance transform of the two nearest neighbour slices of the corrupted slice, which gives two images where pixel value shows the distance from the nuclear envelope segmentation. Figures 8.24 and 8.25 show these distance transformations for slices 0069 and 0071, 0070 being the corrupted slice.

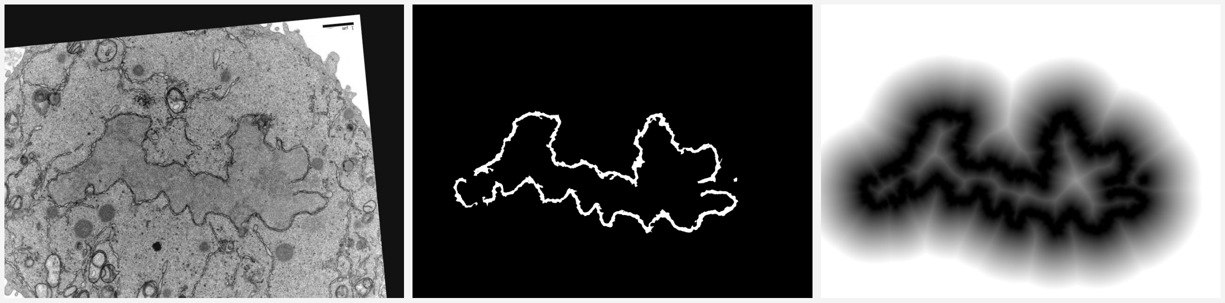

(a) Original image **(b)** Nuclear envelope segmentation **(c)** Distance map from the nuclear envelope

Figure 8.24: Slice 0069

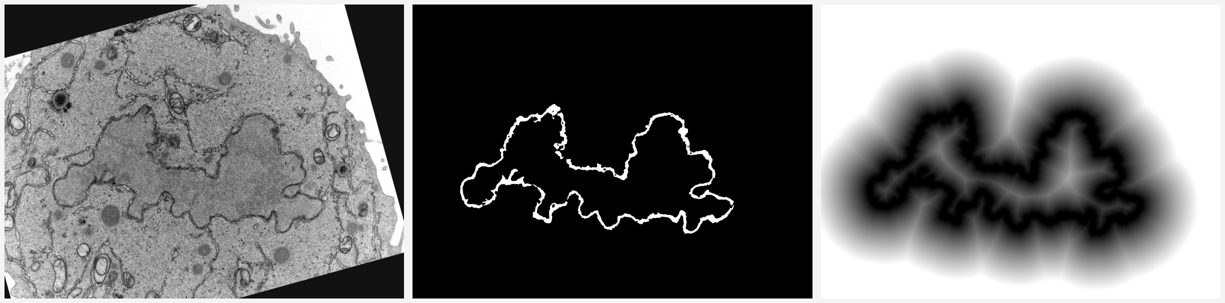
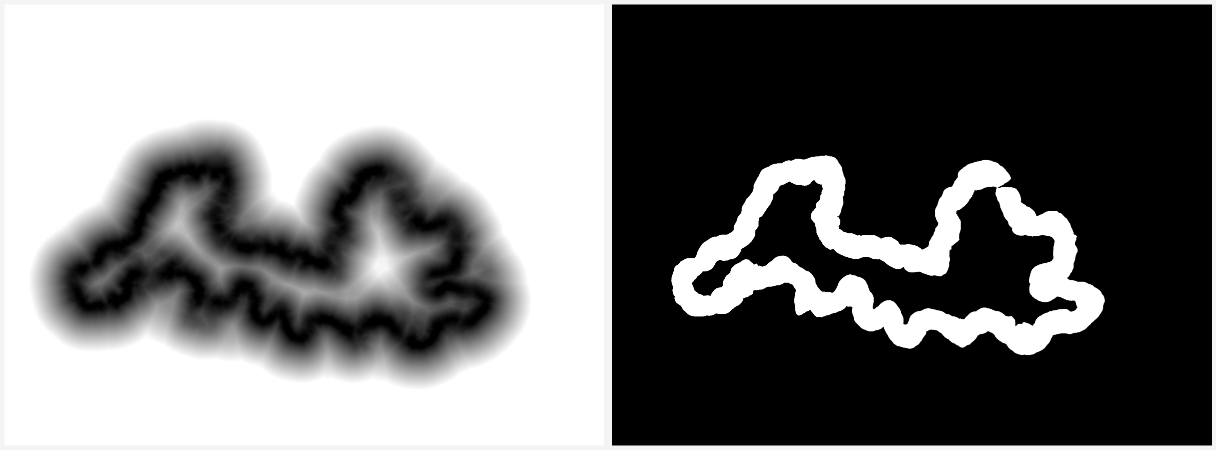

(a) Original image **(b)** Nuclear envelope segmentation **(c)** Distance map from the nuclear envelope

Figure 8.25: Slice 0071

After finding the distance transforms of the neighbouring images they are added together to give a further distance map, which is effectively that of the nuclear envelope estimate of the corrupted slice. To allow for the maximum movement between the slices either side of the corrupted slice this added distance map is thresholded for a maximum cumulative distance of 45; this threshold lies roughly where the dark and light areas change most steeply. The sum of distance maps and the resulting of applying the threshold are shown in Figure 8.26.

Once we have the thresholded depth map as a estimate of the nuclear envelope we need to refine it to have similar width to a normal nuclear envelope. It is not possible to perform a morphological erosion step because the movement, or lack of movement of the nuclear envelope between slices means that approximations of the kind in Figure 8.26b will have a varying thickness, so we cannot apply a constant erosion to the image. Instead, a skeletonisation followed by a morphological dilation with a circular structuring element of radius the same as the average thickness of nuclear envelope (40nm) is applied. At the resolution the algorithm operates at, 40nm is roughly 12 pixels. The skeletonisation and result of the morphological dilation are shown in Figure 8.27, and the result overlaid on slice 0070 is shown in Figure 8.28.

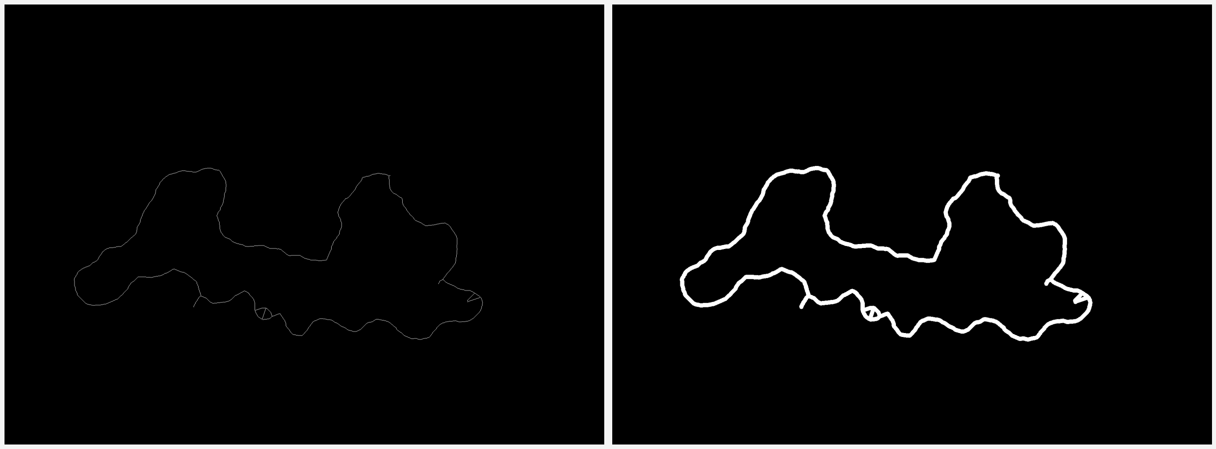
We can see that the result has a close overall resemblance to the nuclear envelope of slice 0070 albeit with a few small local differences; the global shape is captured well. After all this is an estimate as a starting point for manual corrections, so we only require it to be quite close to the actual envelope.


(a) Distance map as addition of neighbouring slice's distance maps

(b) Thresholded distance map

Figure 8.26: Finding an approximation to the location of the nuclear envelope estimation

Slice 0070 is one case where the noise does not greatly effect the ability for us to pick out the nuclear envelope easily, but there are some cases where the noise completely covers large parts of the nuclear envelope. In these cases the best that can be done by the manual segmentors is to guess where the envelope would be based upon its neighbouring slices [1]. This is essentially the same strategy adopted here for estimating the nuclear envelope for corrupted slices.


(a) Skeletonisation of thresholded distance map

(b) A dilation operation gives a thickness of the average nuclear envelope

Figure 8.27: Finding an approximation to the location of the nuclear envelope estimation

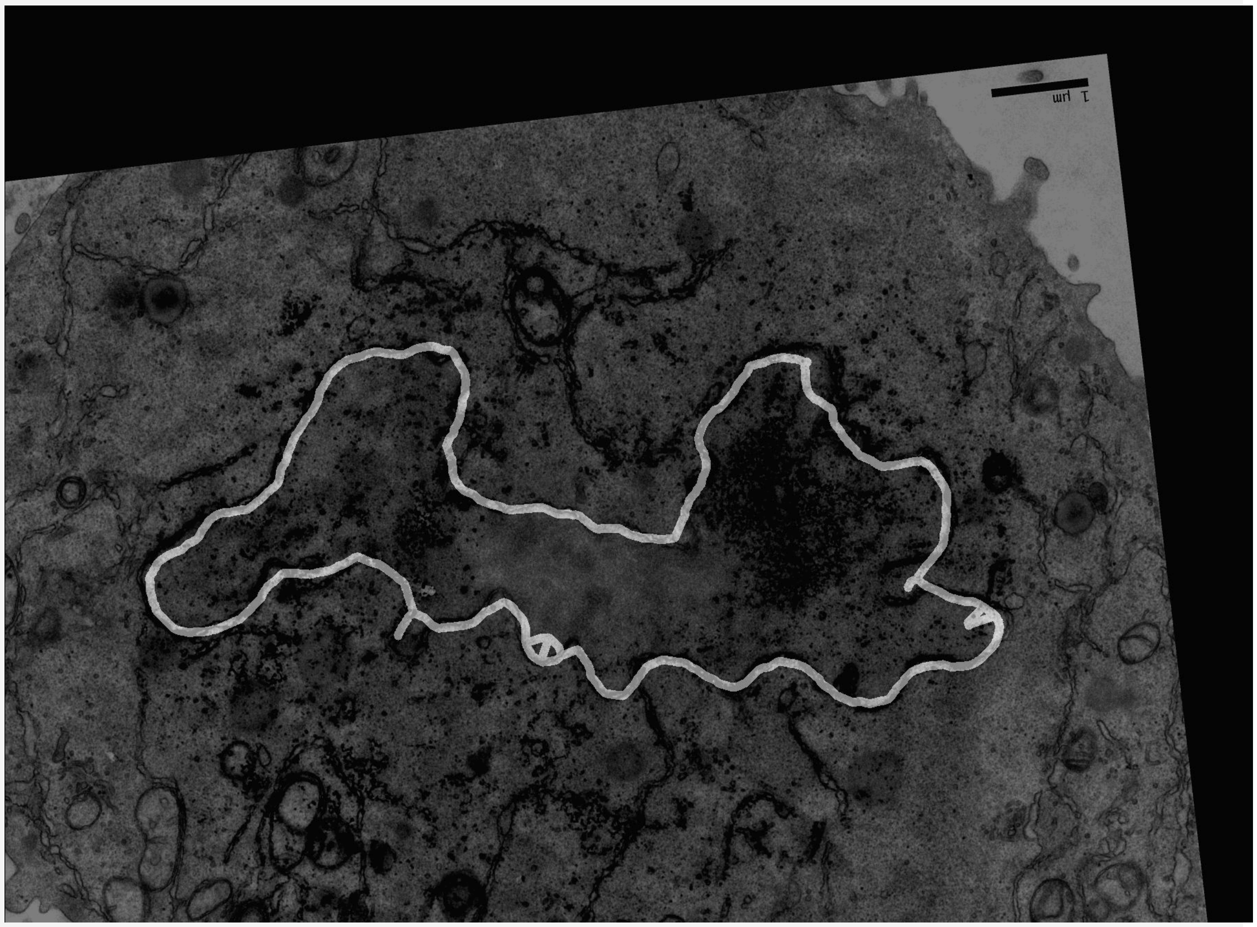


Figure 8.28: Result of approximation the nuclear envelope for a corrupted slice, number 0070

9 Algorithm Discussion

The chapter presents a discussion of design choices and the process that lead to the algorithm detailed in the previous chapters, kept separately to avoid cluttering its presentation.

9.1 Machine-learning Based Algorithm

The first stage in development of the algorithm was to investigate machine learning techniques that have proven to be successful in recent literature, with a particular focus on Ilastik [44]. The applicability of the other machine learning techniques presented in Chapter 4 is limited by the inability to use shape descriptors effectively for segmentation of the nuclear envelope.

The general purpose machine learning algorithms provided by the Weka [22] collection were used, along with features from the ImageJ plugin FeatureJ to give a set of features similar to those used in general purpose segmentation applications [13, 44]. A random forest classifier was trained and the algorithm developed using one sample slice and ground truth from the dataset to being with.

I knew from the outset that any machine learning algorithm would need application specific features in order to perform well at segmenting the nuclear envelope; this was clear from the performance of existing tools presented in the introduction. This lead to developing a feature based on the proximity of the nuclear envelope to the texture of the inside of the nucleus. The idea behind such a feature is that it should be able to discriminate between nuclear envelope and endoplasmic reticulum, which are visually the same, by their locality with respect to the nucleus. To obtain the region that represents the nucleus a naïve flood fill algorithm based on pixel intensity was used, which sufficed as the problem of gaps in the NE was not considered at this point.

After development of this feature, its testing showed that it was not quite enough to improve the quality of the segmentation sufficiently; even a descriptive feature of locality to the nucleus breaks down because of variable thickness of the nuclear envelope.

I think that the reason for the classifier based approach being ineffective overall is that the structures in the data are too complex to be described by image features based purely on grey-level values. Shape plays a big part in segmentation of EM images containing mitochrondria or neurons, but the complexity in shape of the nuclear envelope means that shape descriptors cannot be leveraged to improve the classification results.

A further difficulty with the nuclear envelope is that it cannot generally be described by one edge or one area; the double lipid bi-layer structure is described by two dark lines at an average distance apart, with a bright region between them. Saying this however, sometimes the blurring of the nuclear envelope (Section 5.5) leads to a single region of similar intensity. For machine learning techniques attempting to classify individual pixels this feature description is too complex for good results to be obtained; the successes for machine learning techniques in neuronal segmentation (Section 4.3) occur because the boundary between neurons is consistently described by a near-constant intensity region.

There are more disadvantages to the machine learning approach that were slowly being realised as I developed an algorithm:

- Each change to the feature set requires a new classifier to be trained; this is a lengthy process and really hinders research and development on a restricted timescale.

- When the algorithm is extended and tested on EM images of cells with diseases or at different stages of mitosis further training data will probably be required, and that data may not be available or will take time to produce.

9.2 Expanding the Nucleus into the Nuclear Envelope

Preliminary work lead to requiring an approximation of the nucleus as a proximity feature for points on the nuclear envelope, and it became clear that having such an approximation gave a good estimate of the position of the nuclear envelope. Given this, I started to investigate techniques that could expand outwards from the approximation to find the region containing the nuclear envelope. As we can see in the algorithm, similar techniques to those used to find the nucleus could also be found to identify the nuclear envelope and immediately the results were very positive.

It did not take much time after developing the nucleus feature to drop the classification approach completely and move onto a hand-designed algorithm. This also enabled faster development as the overheads of training classifiers and calculating features were no longer present. As soon as the classification approach was dropped, the attention focus upon the pipeline of hand-designed algorithm steps led me to notice possibilities such as the 3D consistency between slices, which had not been considered before, but which have a massive impact upon the success of the segmentation algorithm.

9.3 Edge-based vs. Region-based segmentation

Whilst containing elements of both types of segmentation algorithm, I would classify this algorithm as a region-based segmentation because of its nature to identify the nucleus and nuclear envelope through flood fill algorithms. This being said, a quick look at a sample image for this application could give the idea that an edge-based segmentation algorithm would be a good method to investigate. The nuclear envelope could be defined by detected edges, for example by Canny edge detection, but this approach is difficult for many reasons.

Firstly, there are a great number of small changes in gradient within the image; thresholds would have to be specified to pick out those gradients we can associate with the nuclear envelope. Identifying such a threshold is difficult as there are many edges that are similar in intensity to those at the nuclear envelope. Smoothing may simplify both the tasks of identifying suitable thresholds and performing the edge detection but we may sacrifice some important edge information in the process.

Secondly, even if it were possible to reliable detect those edges that belong to the nuclear envelope, it is likely that we would also find edges that are associated with the endoplasmic reticulum. Therefore we again need a way of distinguishing those edges that are part of the nuclear envelope from those that are endoplasmic reticulum, and that is only achievable through the identification of the region of low variance as an approximation to the nucleus.

Finally, suppose that the edges representing the double lipid bi-layer structure of the nuclear envelope were obtainable. The target of the segmentation is to cover the region including the gaps between these structures, so another region finding algorithm such as the flood fill would have to be used to achieve this. In effect, having to find the nucleus approximation to localise which edges are those belonging to the nuclear envelope eliminates the need to find those edges in the first place, as we already have an estimate to the contour of the nuclear envelope. As a region is required and can be obtained through a flood fill we only require some points inside the region; the region found by filling from the edge points would be the same.

9.4 Algorithm Framework

Whilst obviously not the main focus of this project, I felt that it was important to add some structure to the code developed for this project. This was mainly inspired by the end-user's vision of a software suite of segmentation algorithms for all the different sub-cellular structures. The framework is not complex by any means and I expect it will go through many more iterations before it is integrated into the end product. However, for ease of development it is extremely useful to define the running of an algorithm, including handling of multi-threaded code, only once, and simply plug together the different components of an algorithm. Individual parts also become much easier to isolate and test with this framework.

Problems 10

10.1 Disconnected Nucleus Sections

Sometimes due to the 3D nature of the nuclear envelope there are small sections in an individual 2D slice that are separate from the main part of the nucleus in that slice. An example of this is shown in Figure 10.1.

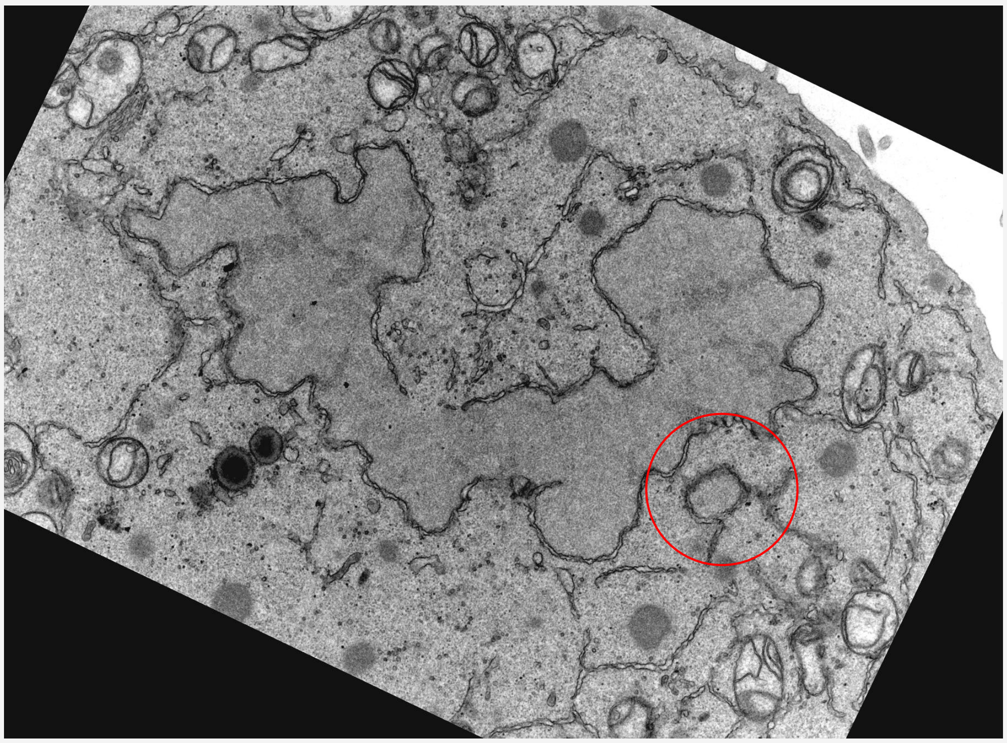


Figure 10.1: Slice 0014, showing a small disconnected section of nuclear envelope.

We see that circled to the bottom right of the main nucleus there is another small section of nucleus. This appears disconnected in an individual 2D slice due to a small offshoot of the nucleus being sliced through to create the image (Section 5.3). Further on in the image stack the small section gets larger and eventually joins on to the main nucleus structure.

These small sections do not get segmented by the automated part of the algorithm because they are too small for the large neighbourhood variance filter to capture seed points inside them. The subsequent flood fill to find the approximation to the nucleus therefore doesn't fill any of the area within these small sections, and the subsequent steps of the algorithm fail to find the small area of nuclear envelope.

10.1.1 3D Flood Fill

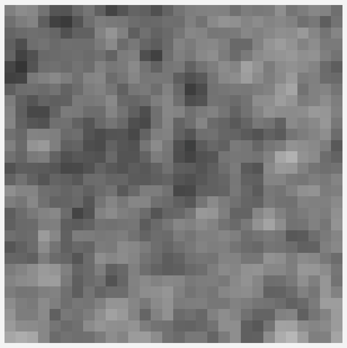
One potential solution to this problem is to consider the flood fill algorithm in 3D, but the problem with such an algorithm is ensuring that it does not fill outside the nucleus. For this not to happen there could never be a case where a pixel inside the nucleus in one slice is a pixel in the cytoplasm in a neighbouring slice. As soon as this happens in just one place the knowledge that the region we have flood filled is contained inside the nucleus is no longer valid. This is something that can never be guaranteed, and unless the 3D flood fill can be better controlled it would not solve this problem.

10.1.2 Seed Points from a Smaller Neighbourhood

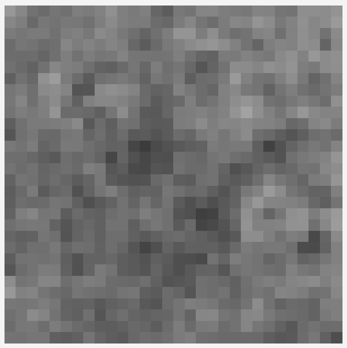
Being able to identify seed points inside the nucleus from a smaller neighbourhood would allow these smaller disconnected sections to be identified. Unfortunately, the smaller the neighbourhood used the more likely it is that textures within the cytoplasm become similar to the texture inside the nucleus. The reason that the large neighbourhood variance filter works well is because in general the cytoplasm is cluttered with many different structures giving it a high variance over a large neighbourhood. As soon as the neighbourhood size is reduced we start to run the risk of identifying areas in between these structures which are harder to distinguish from the texture inside the nucleus.

Take for example the 30 pixel neighbourhoods in Figure 10.2. This is just one example where the variance of a 30 pixel window in the cytoplasm is less than the variance of a 30 pixel window inside the nucleus; the means are also quite similar so there is no real discriminatory potential there. Our human processes for visually distinguishing textures also fail to see much difference in these two samples, which doesn't bode well for an automated process.

It may however be possible to find a threshold such that we can be sure all points that pass through are inside the nucleus. The trouble is that the number of such thresholds that we could choose from becomes smaller as it becomes more difficult to separate the textures. Our choice of threshold would have to be very restrictive which means that we may not find seed points in all regions and we may still miss these small disconnected structures.



(a) 30x30 piece of nucleus, mean = 99 and variance = 408



(b) 30x30 piece of cytoplasm, mean = 94 and variance = 286

Figure 10.2: Showing similar regions within the nucleus and cytoplasm.

10.2 Nuclear Pores

Nuclear pores are structures closely related to the nuclear envelope, but the twp should be segmented separately. Their characteristics are shown in Figure 10.3, where we see two well defined sections of nuclear envelope being joined by a single dark line, the nuclear pore. This is one of the clearest examples of the nuclear pore to show, as due to blurring (Section 5.5) it can be difficult to distinguish between nuclear pore and nuclear envelope.

Nuclear pores are roughly circular with diameter between 100 and 125 nm, and as the slicing process takes 70nm slices sometimes the pore is quite wide if it is cut through at its thickest point. If the slice cuts through near the top or bottom of the circle the nuclear pores appear thinner in 2D and are thus harder to distinguish.

The current segmentation algorithm includes nuclear pores in the segmentation due to the flood fill capturing areas of similar intensity value to the nuclear envelope. Whilst it may be possible to identify some of the well defined

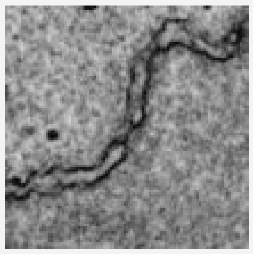
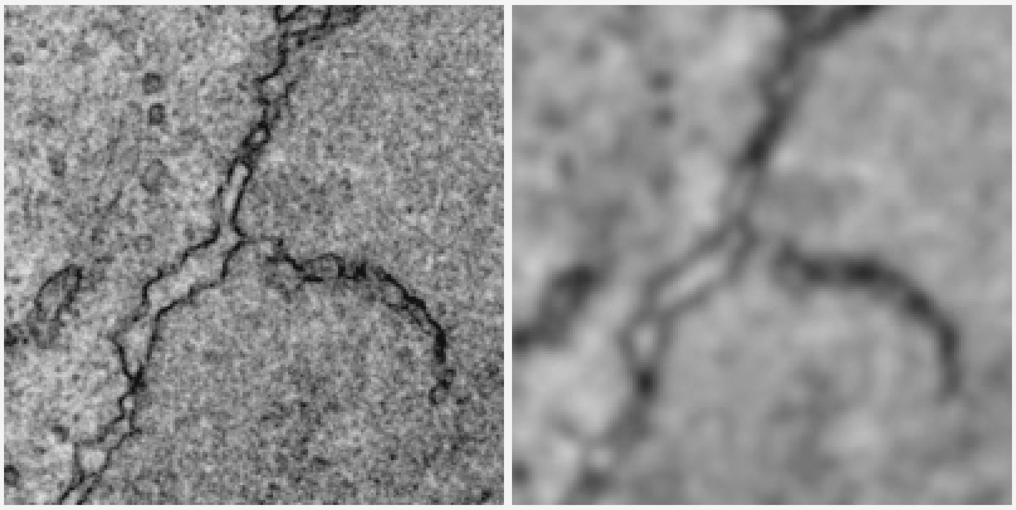


Figure 10.3: A nuclear pore visually connects sections of the nuclear envelope

nuclear pores by looking for single line structures within the nuclear envelope, including them in the segmentation does not affect the overall characteristics of the nuclear envelope when viewing the 3D reconstruction.

10.3 Perinuclear Space

As previously explained in Section 5.5, the angle of the cutting blade to the nuclear envelope affects whether the double lipid bi-layer structure is well defined. An example of a well defined nuclear envelope is shown in Figure 10.4; the perinuclear space is the area between the lipid bi-layers that define the nuclear envelope.



(a) A well defined section of nuclear envelope from slice 0064

(b) Result of running the band-pass filter in the sample in a

Figure 10.4: A large perinuclear space prevents the segmentation from covering the whole area inside the nuclear envelope.

To a large extent these gaps are filled by the band-pass filter used in the nuclear envelope flood fill stage, but in some cases, as in Figure 10.4, the perinuclear space is larger than the average thickness of the nuclear envelope and so the two edges remain distinct with a gap between them. When it comes to the flood fill of the nuclear envelope this can cause:

- A large gap in the nuclear envelope that should be segmented.
- Only one or neither edge is captured as they are quite thin, which increases the risk of the flood fill being stopped too early. When the two edges can be blurred together it is more likely that the flood fill will succeed and move through the wider area.
- If both edges are found then there is a risk that the outer edge is removed in the pruning of connected structures. A suitable change to this step to account for just this case is possible, where we look at distance to the nucleus, but we run the risk of not pruning off some structures that are not nuclear envelope.

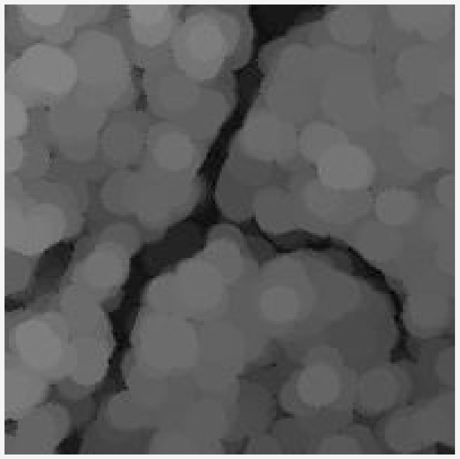


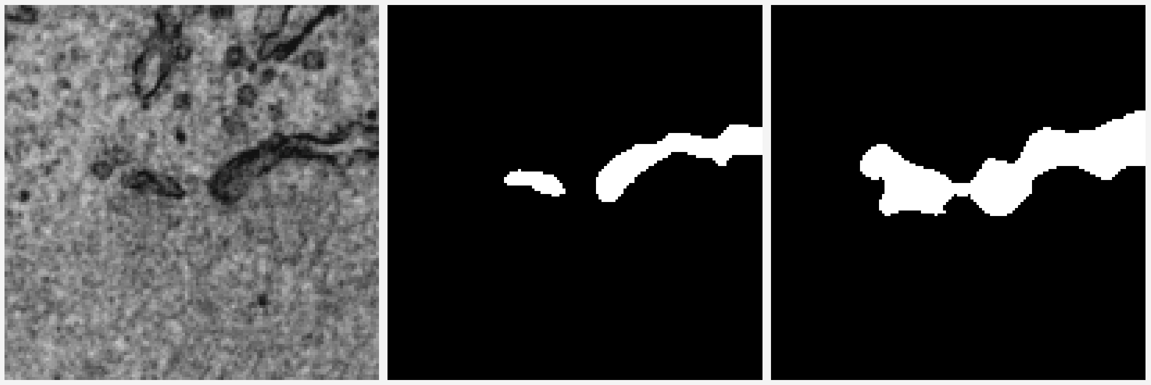
Figure 10.5: Morphological closing operator applied to sample in Figure 10.4a

I believe the best solution to this problem would be to find a better way of closing the perinuclear space regardless of its size. One possible way to do so is to use a morphological closing operator of a suitable size, the result of which is shown in Figure 10.5. This may seem to work, but the problem with this transformation is that it cannot take into account that we only want to smooth the inside of the nuclear envelope and not any structures around it. If, for example, there were vesicles or other structures close to the nuclear envelope they could be smoothed and joined to the nuclear envelope. It is likely that they would not be removed when pruning connected structures as they would just make the nuclear envelope estimate thicker, rather than appearing in the skeleton of this estimate as a separate branch.

A solution therefore could take into account both proximity to the nucleus and the local edge orientation of the nuclear envelope. As we can obtain some seed points inside the nuclear envelope at many places, but not all, an algorithm step could start at these points and repeatedly move along the nuclear envelope performing local closing operations. The target is for the perinuclear space can be closed without the morphological closing being applied to any structure that is not nuclear envelope.

10.4 Small Gaps in the Nuclear Envelope

A further side effect of the strategy employed to close the perinuclear space (Section 10.3) by using a band-pass filter is that very small gaps in the nuclear envelope may also be closed. These gaps are distinguishable from the perinuclear space by the fact that the smoothing that causes them to be segmented occurs along the edge direction of the nuclear envelope rather than perpendicular to it. An example of a gap that is closed in shown in Figure 10.6.



(a) Sample of slice 0014 **(b)** Manual segmentation of the sample **(c)** Automated segmentation of the sample

Figure 10.6: Showing the erroneous closing of a gap in the nuclear envelope.

10.5 Pruning Sections of Nuclear Envelope

In general the pruning of connected structures works well to reduce the nuclear envelope approximation to a better segmentation, but it is susceptible to trimming off parts of the nuclear envelope that should remain. This step depends greatly on the quality of the nucleus approximation, and it is a failure to obtain an approximation of the complete nucleus that results in some erroneous pruning of the nuclear envelope.

Figure 10.7 shows an example of this problem. We can see in Figure 10.7c that the approximation to the nucleus didn't manage to capture the small loop in the centre of the original image. Looking at the entrance of the loop there is a small light dot and a dark dot close together, and when the variance filters are run over the image prior to the flood fill for the nucleus approximation these two dots increase the variance sufficiently to block off the flood fill into this loop.

This error in the approximation to the nucleus has a knock on effect when the algorithm reaches the pruning of connected structures, which uses the notion of a clear path between a structure and the nucleus approximation as meaning that structure is nuclear envelope. Any structure that has to cross any other structure to reach the nucleus cannot be nuclear envelope and is therefore pruned from the nuclear envelope segmentation. This is what happens to the example in Figure 10.7 as shown in Figure 10.8.

The white vertices in the red circle in Figure 10.8 form part of the sections of nuclear envelope that are found by the flood fill from seed points on the nuclear envelope. If the nucleus approximation was more accurate and filled this loop then the nearest point in the nucleus to these vertices would be inside that loop, the paths to this nucleus approximation would not be blocked by another structure and they would be retained as nuclear envelope. Instead, the nearest point on the nucleus approximation for these vertices is on a completely different part of the nucleus, as shown by the direction of the grey radial lines originating at the vertices. These lines have to cross another structure to reach the nucleus and are therefore pruned from the segmentation.

This example shows how important it is to obtain a good approximation to the inside of the nucleus, and that in turn requires finding as many good seed points as possible. If we could find seed points over a smaller area (as discussed in Section 10.1.2) it is likely that some seed points would be located inside this loop that the flood fill for the nucleus approximation failed to reach. The flood fill would find the area inside the loop from these seed points and consequently the large section of nuclear envelope would not be pruned from the nuclear envelope approximation.

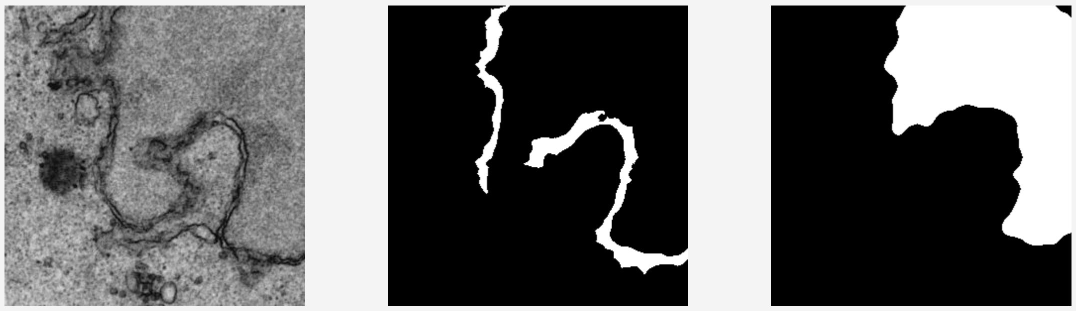
10.6 Failure to Prune Connected Structures

Another problem with the pruning of connected structures is that of structures not being pruned when they are not nuclear envelope. This happens when there are gaps in the nuclear envelope and we are unlucky that the shortest path from a structure to the nucleus passes through this gap or the flood fill for the nucleus approximation leaked out into the cytoplasm. Examples of these cases are shown in Figures 10.9 and 10.10.

These are both difficult problems to solve. For the leaked estimate of the nucleus in Figure 10.9, there really is not much that can be done as the texture is too similar to that of the nucleus and is captured by the flood fill. Placing harsher restrictions on the flood fill negatively impacts other slices in the stack by preventing important regions of the nucleus from being found. These kinds of errors are however easy for a manual segmentor to spot and fast to remove, and given that these cases happen fairly rarely, having the manual corrections solve this problem is a suitable compromise for now.

The case in Figure 10.10 of structures being retained due to gaps is also particularly difficult to solve. I have thought about closing gaps in the nuclear envelope by performing a local search guided by the outline of the nucleus approximation, which would work well for the example case but doesn't generalise well to all slices. For example, consider attempting to close the gaps in the nuclear envelope in Figure 10.9. If we follow the outline of the nucleus approximation we would end up joining gaps between the nuclear envelope and structures that aren't the nuclear envelope. We also cannot make the naïve assumption that a small distance between points on the nuclear envelope means that there is a gap that can be closed, as we cannot ignore cases where there are thin branches in the nuclear envelope as in Figure 10.7a.

Both problems could be solved if a better descriptor for the texture inside the nucleus is found, but this a similar problem to attempting to find seed points over a smaller neighbourhood (Section 10.1.2) and one that is certainly difficult and left to thought and further research.



(a) Original image **(b)** Pruned segmentation **(c)** Nucleus approximation

Figure 10.7: Errors in pruning connected structures from the nuclear envelope approximation

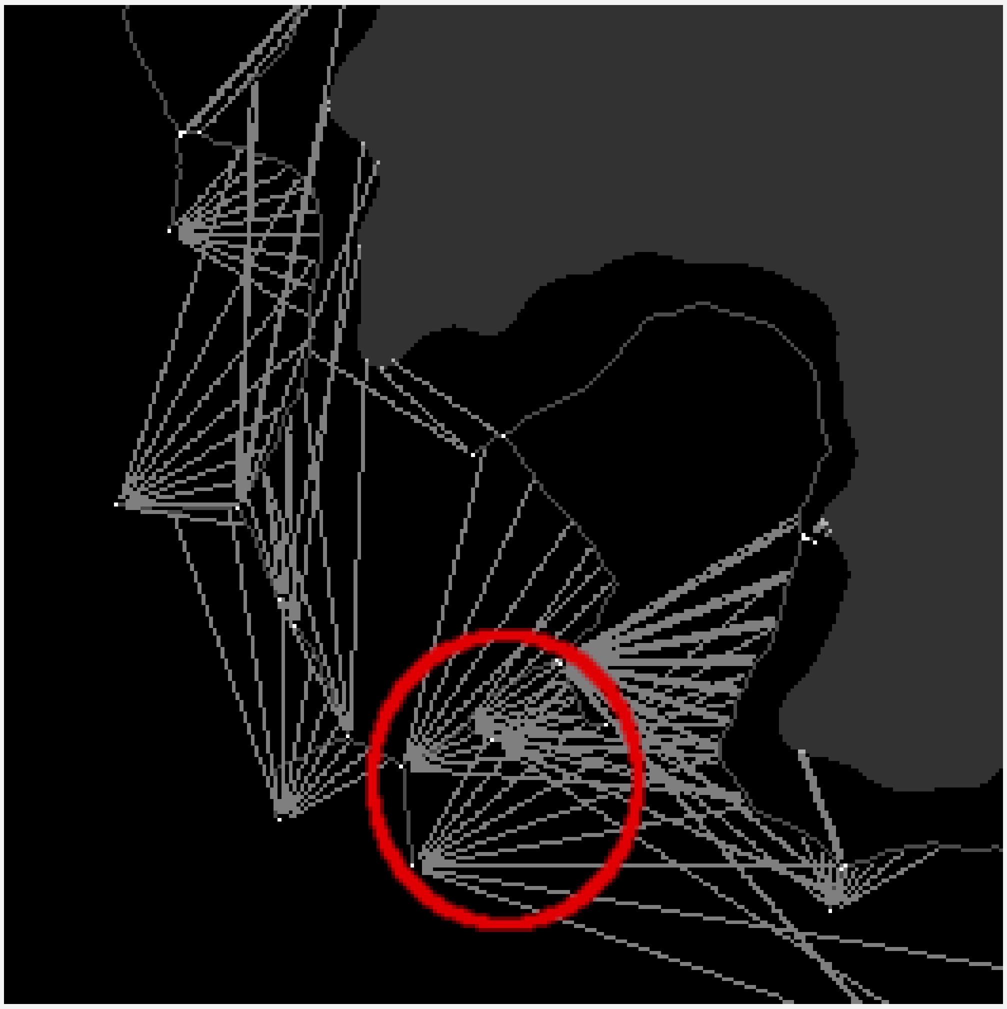
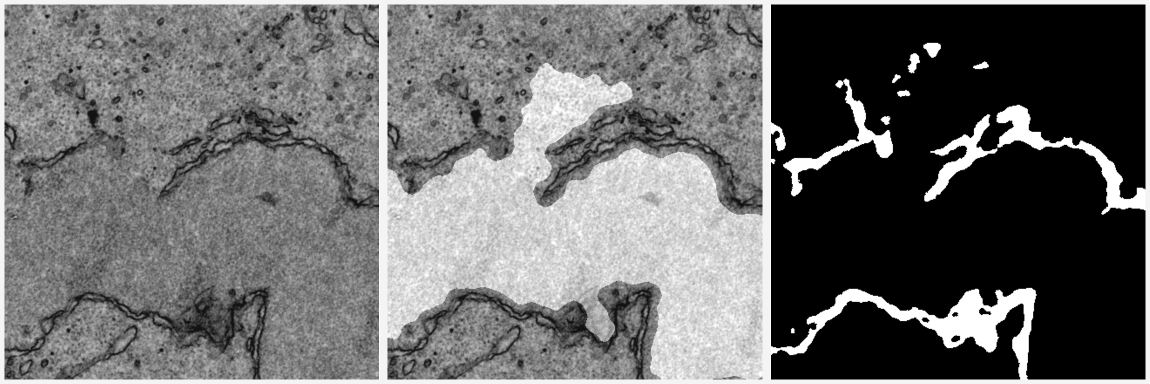


Figure 10.8: Showing the pruning of the connected structures

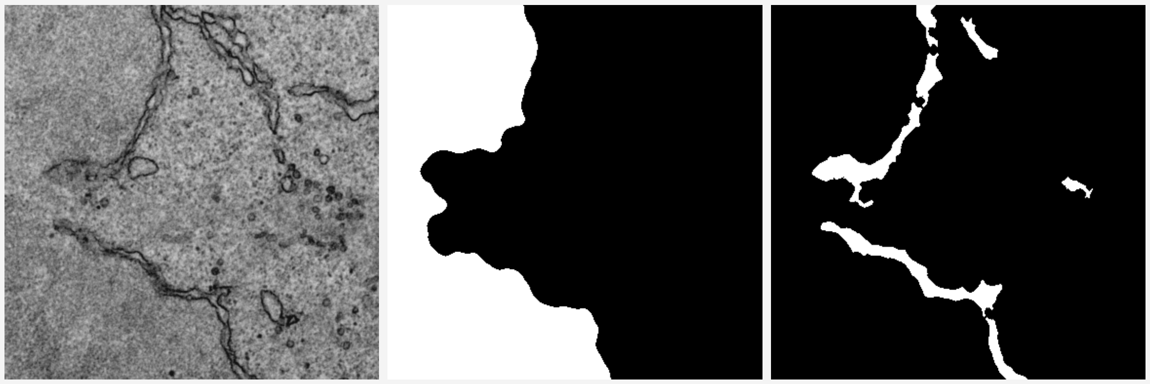


(a) Original image

(b) Nucleus flood fill leaks into cytoplasm

(c) Nuclear envelope segmentation after pruning

Figure 10.9: Leaking of the nucleus approximation causes failure to prune some structures that are not nuclear envelope



(a) Original image

(b) Nucleus flood fill that doesn't leak through the gap

(c) Erroneous structures not pruned from the nuclear envelope segmentation

Figure 10.10: Holes in the nuclear envelope allow some erroneous structures to be retained in the nuclear envelope segmentation

11 Evaluation

The main objective of this project is to dramatically reduce the overall time it takes to segment a dataset of the type we looked at in Chapter 5. There are several elements of the process that are difficult to automate, such as registration and brightness equalisation, due to the complexities in the data being expressed only in greyscale values. These processes are also not where the main time is spent to obtain the segmentation: outlining and picking out the nuclear envelope is estimated to take 2-3 weeks on average [1] and so that was the focus of this project.

The segmentation algorithm presented herein has a runtime of about one hour on a quad-core processor, so a rough estimate puts it at two hours to complete on a typical computer used in the labs at CRUK. These computers have plenty of memory to handle the large number of images, but they only sport a dual-core processor. Dr. Christopher Peddie estimated [1] that it would take another couple of hours to half a day to perform the manual corrections across all slices, including those that are corrupted. The total time for the complete process including necessary manual steps and the automated algorithm is therefore approximately half a day to one day. Cutting down 2-3 weeks to just one day is a significant reduction and one that is very much welcomed by the collaborators at CRUK.

We now take a look at a more detailed evaluation of the algorithm performance from both quantitative and qualitative perspectives.

- A qualitative analysis of the performance of Ilastik is given in Section 11.1.
- Segmentation metrics are calculated for the results and compared against those achieved with Ilastik in Section 11.2.
- In Section 11.3, end-user scoring gives an idea of the good and bad slices in the results.
- The performance of the algorithm and its applicability to the wider biological audience is evaluated by the end-users and included verbatim as feedback in Section 11.4.
- A comparison of the 3D reconstructions of the manual and automated dataset is presented in Section 11.5.
- Finally, in Section 11.6 the algorithm is run on a cell at an earlier stage of mitosis.

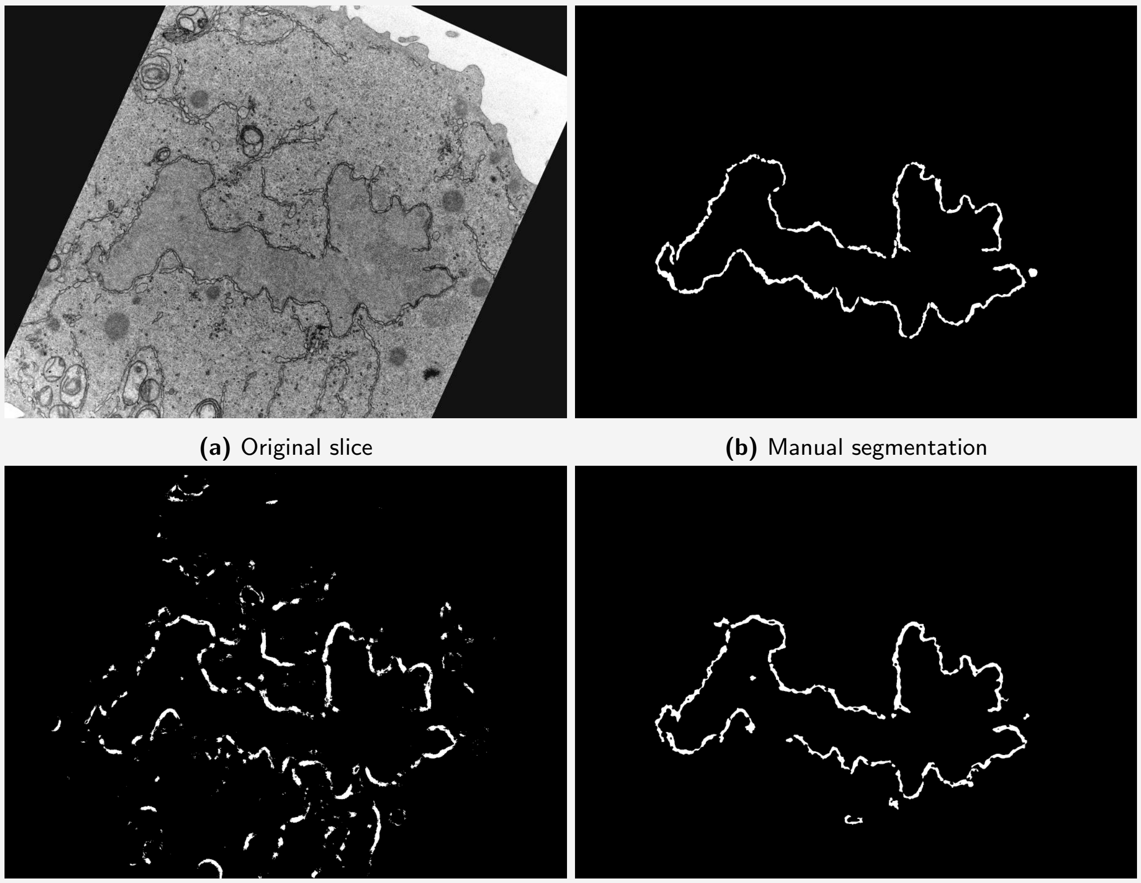
The end-user scoring a set of 36 images taken by choosing every other image from the original dataset and removing corrupted images. This is mainly to reduce the workload for the evaluation by hand carried out by the collaborators. Examples of the problems discussed in Chapter 10 are found in images throughout these samples, as well as the rest of the results, and as such no loss of applicability to the complete dataset is assumed.

11.1 Comparison to Sample Trained with Ilastik

As no other work has focused solely on the nuclear envelope there is nothing to provide a direct comparison, however a classifier trained with Ilastik should provide a suitable comparison with the state of the art. As expected we see a marked reduction in the segmentation quality for those produced with Ilastik.

Ilastik does not yet provide the functionality to import labels along with images, so I did as best I could to match the manual segmentation of the nuclear envelope by painting on class labels. A random forest classifier was trained with colour, edge, and texture features over a large range of neighbourhood sizes according to the labels shown in Figure 11.2, chosen so as to cover as many structures of the cell as possible. The use of only one image and set of labels as the training data may seem too little, but I would argue that the sample image is suitably representative of the rest of the dataset such that any general problems or difficulties experienced by the classifier would not be solved by simply using more training data.

Figure 11.3 shows the outcomes from all segmentation methods on slice 0022, one of the best results from the nuclear envelope algorithm. Figure 11.1 shows the outcomes for slice 0066, the slice that achieved the worst result from the algorithm presented herein.



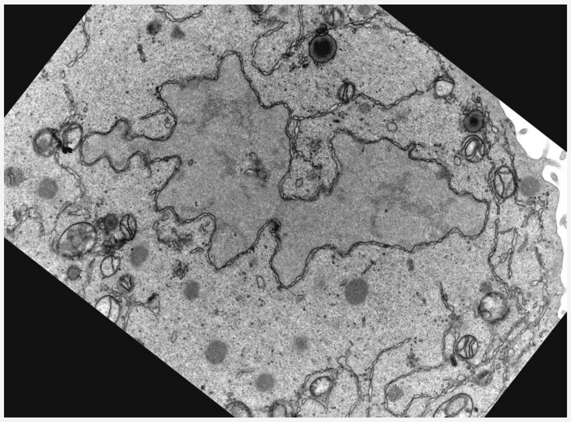
(a) Original slice **(b)** Manual segmentation

(c) Ilastik segmentation **(d)** Algorithm result segmentation

Figure 11.1: Segmentation comparison of slice 0066

Instantly we can see that there are many more erroneously segmented structures from the cytoplasm area than in the results of this algorithm. I believe this is because of the use of general feature descriptors; these are greyscale images with complex objects and these features cannot provide enough descriptive power for the classifier to obtain an accurate segmentation. Another disadvantage to the Ilastik segmentations is they don't manage to obtain all of the nuclear envelope structure, and I believe this is for the same reason.

A further six segmentations obtained from the classifier trained with Ilastik are shown in Appendix C.2, and having these we can move towards a quantitative analysis.


(a) Training image

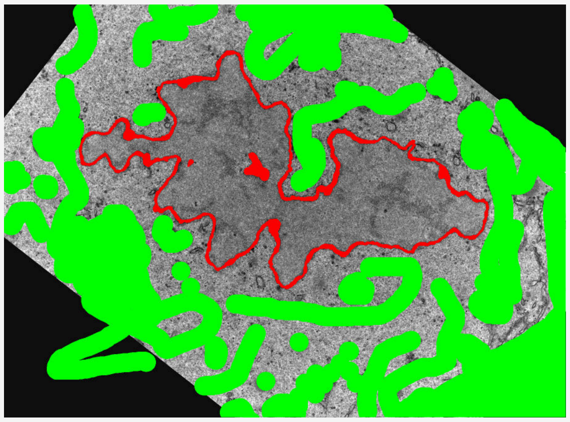
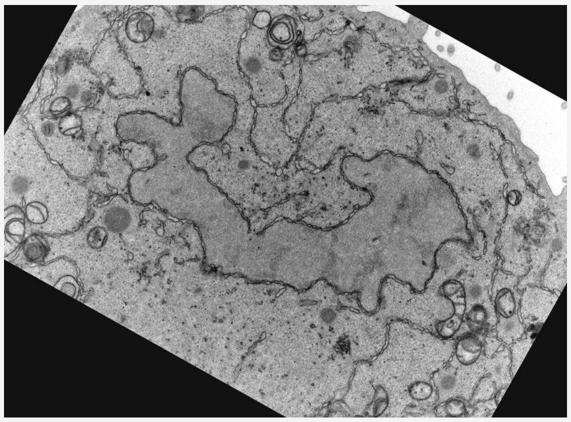
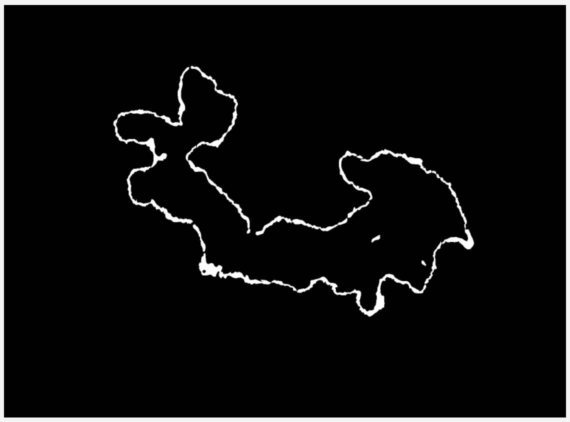
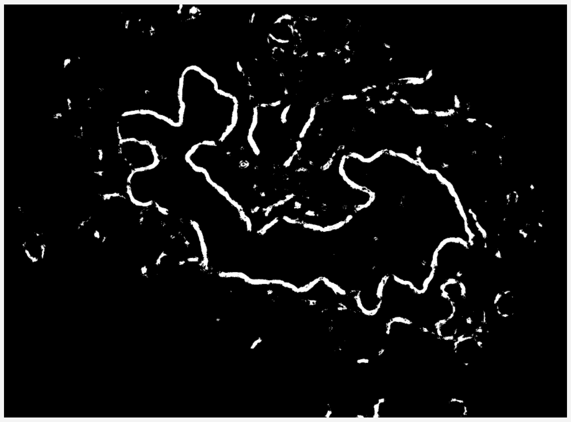

(b) Painted labels

Figure 11.2: Training data for Ilastik


(a) Original slice


(b) Manual segmentation


(c) Ilastik segmentation

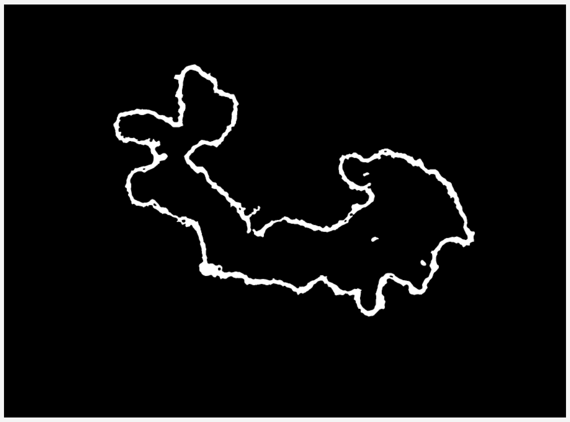

(d) Algorithm result segmentation

Figure 11.3: Segmentation comparison of slice 0022

11.2 Segmentation Quality Metrics

The full table of metrics and user scores for the evaluation dataset is included in Appendix C.1.

11.2.1 Pixel Error

The average pixel error across the evaluation dataset is 0.0195, giving a segmentation quality of around 98% according to this metric. We could just stop there and say this shows a good result, but I think this value shows that while the pixel error can be a good measure it doesn't always give meaningful output.

The reason that this figure is so high is that the area of the nuclear envelope compared to the overall image area is quite low. The majority of pixels in the manual and automated segmentations are background, so there is little contribution from these regions. Take for example an image that contains only background pixels and is completely black; by the pixel error metric this will still achieve a reasonable result when compared to the manual segmentation. Therefore, pixel errors in the nuclear envelope contribute little to the error simply because there are many pixels in the image that do not contain the nuclear envelope.

11.2.2 Jaccard Index

The average Jaccard index for the evaluation dataset is 0.479, which gives a quality of 48%. This would indicate that the automated segmentation has achieved an average result, but I believe it still falls below the expected value.

The reason why this value is lower than expected is that we are in effect segmenting the edge of a region by finding the nuclear envelope that surrounds the nucleus. The Jaccard index is based upon comparing the overlapping regions of two segmentations against the total area of the two segmentations; if this total area is small then individual pixel errors are going to become more significant than for larger areas. If we were to find and compare estimates of the nucleus area from both the automated and manual segmentations of the nuclear envelope it is likely that this metric would give much higher quality scores than it does when comparing the nuclear envelope regions.

Given this, the problems discussed in Chapter 10 have a large negative impact upon this metric with the main contributors being:

- Nuclear pores should not be included in the segmentation of the nuclear envelope but are done so by the algorithm. This is however not as important as capturing the overall shape of the nuclear envelope.
- The perinuclear space is difficult to find with the automated segmentation but is always present in the manual segmentation

11.2.3 Rand Index and Warping Error

The average Rand index for the evaluation dataset is 0.279, giving a segmentation accuracy of 28%.

The purpose of these two error metrics is to penalise segmentations that have differences in topology more than individual pixel differences, and this is why the value for the Rand index is lower than the Jaccard index. Chapter 10 highlights problem cases that the algorithm has difficultly segmenting which create large topological differences between the manual and automated segmentations.

Given that the Rand Index and Warping Error both penalise topological errors and that we know about a number of problems in the algorithm, I assume that the results from the Warping Error would give no more information than those for the Rand Index and omit its calculation from this report.

11.2.4 Comparison to Ilastik

In addition to the comparison provided in Section 11.1 the segmentation error metrics can be run on the results from the classifier trained with Ilastik. The same metrics as above were calculated between the classification results of 14 images chosen from across the whole dataset and their manual segmentations. The average results and their respective values for the nuclear envelope algorithm are shown in the table below.

	Ilastik	**NE Algorithm**
Pixel Error	97.4%	98%
Jaccard Index	26%	48%
Rand Index	11%	28%

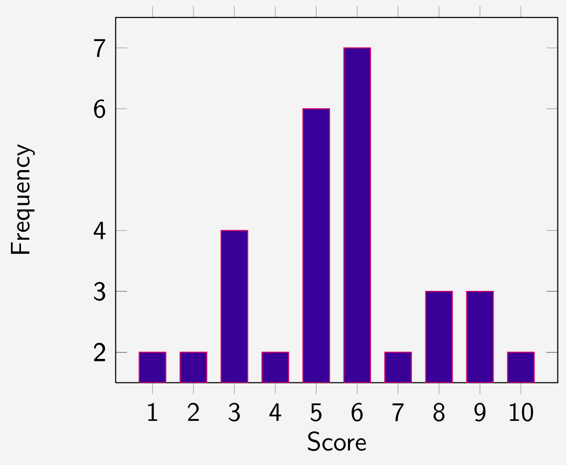
All the values for Ilastik are lower than the quality values for the nuclear envelope algorithm, but this was to be expected after the qualitative evaluation of Ilastik's results. This goes some way to providing assurance that the results are good, even though no meaningful error metric has achieved a particularly high quality value.

11.3 End-user Scoring

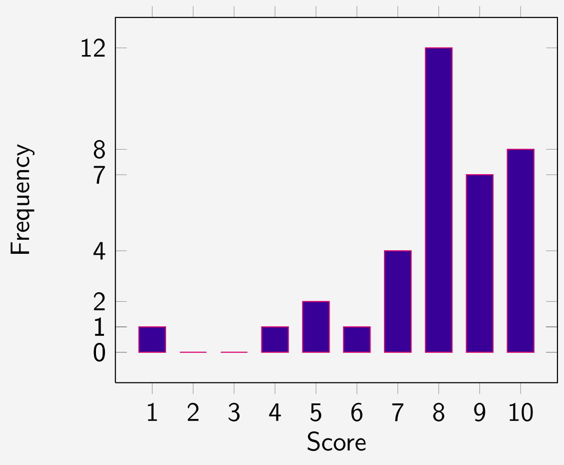
The full table of metrics and user scores for the evaluation dataset is included in Appendix C.1.

The metrics presented in the previous section don't give much evidence to support the segmentations being quite accurate in a global sense. This is primarily due to problems encountered by the algorithm, but there is also some subjectivity to take into account when blurred patches of nuclear envelope are encountered by the manual segmentor. Depending on the person carrying out the segmentation, the blurred patch can either be completely covered or quite restrictively segmented to only the darkest and most certain patches to be nuclear envelope. This and other areas where subjective decisions have to made when creating the manual segmentation account for a further small amount of the lack of quality given by the error metrics.

After some discussion [4, 5] it was decided that a further evaluation method of having the end-users score the evaluation dataset would be useful to have. Dr. Christopher Peddie chose to score the set of evaluation segmentations on a scale of 1-10, where images scored with 1 require the most manual corrections and those with a 10 need the least manual corrections. Dr. Lucy Collinson chose to score the evaluation segmentations based upon the number of over or under segmented regions present. The total number of these errors then gives a score in the same 1-10 region; 10 represents 1-4 errors, 9 represents 5-8 errors and so on until any number of errors greater than 36 is given a score of 1.



(a) Dr. Christopher Peddie



(b) Dr. Lucy Collinson

Figure 11.4: Histogram of user scores

The histograms of scores are shown in Figure 11.4. The histogram for Dr. Christopher Peddie does not give any information other than what we would expect from a relative scoring system, a bell shape curve. I am however pleased by the profile of the scores from Dr. Lucy Collinson, seeing that the majority of slices have somewhere between 0 and 12 corrections to be made. This still may seem like a large number, but the errors that arise from the problems discussed in Chapter 10 tend to lead to large well recognisable errors, so the time for a manual segmentor to fix these errors would be quite small.

11.3.1 High Scoring Slices

Two segmentations that achieved a combined score of one less than the maximum are shown in Figure 11.5. These are very good generally because they don't have any of the cases that cause problems for the algorithm; there are no disconnected sections of nuclear envelope or loops accessible only through thin necks. The blue circles highlight areas in the segmentation where pixels are incorrectly classified as nuclear envelope, typically due to small gaps being missed, and red circles highlight areas where some nuclear envelope was not captured by the algorithm.

11.3.2 Low Scoring Slices

Two segmentations that received low scores are shown in Figure 11.6. The errors highlighted are only a subset of those found that show the most important differences between the manual and automated segmentation.

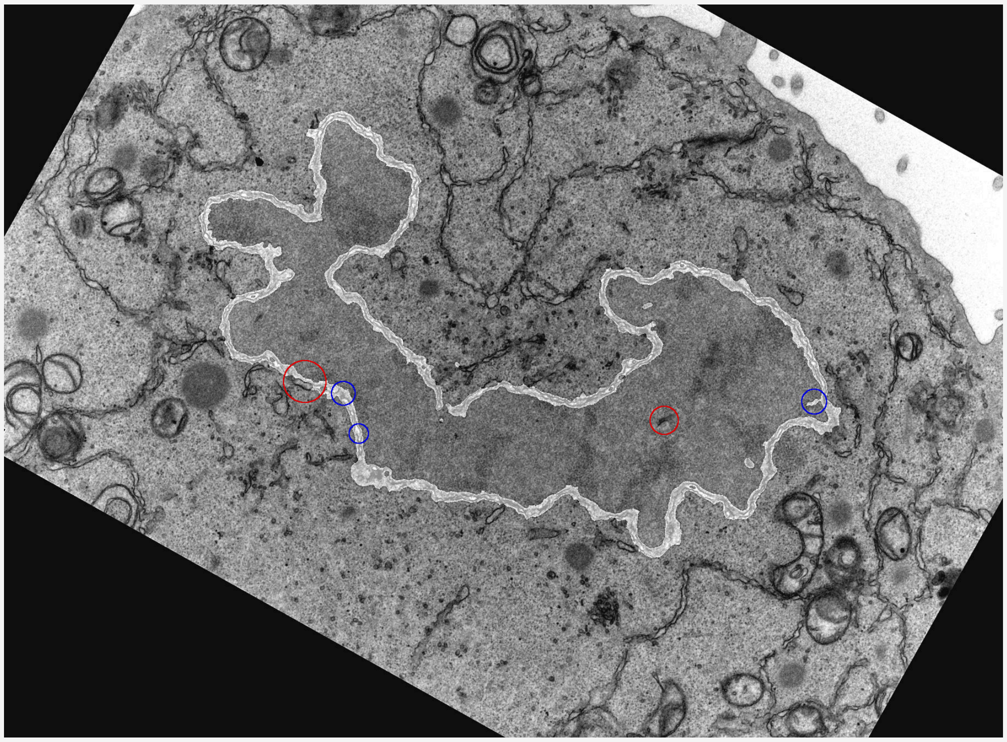
We can see in Figure 11.6a that there are a couple of large envelope sections missed, both due to the problem of disconnected nucleus sections. Slice 0066 (Figure 11.6b) has a bit of difficulty identifying a well defined section of nuclear envelope, which causes gaps in the nuclear envelope hence some additional structures are not pruned from the segmentation.

11.3.3 General Trend

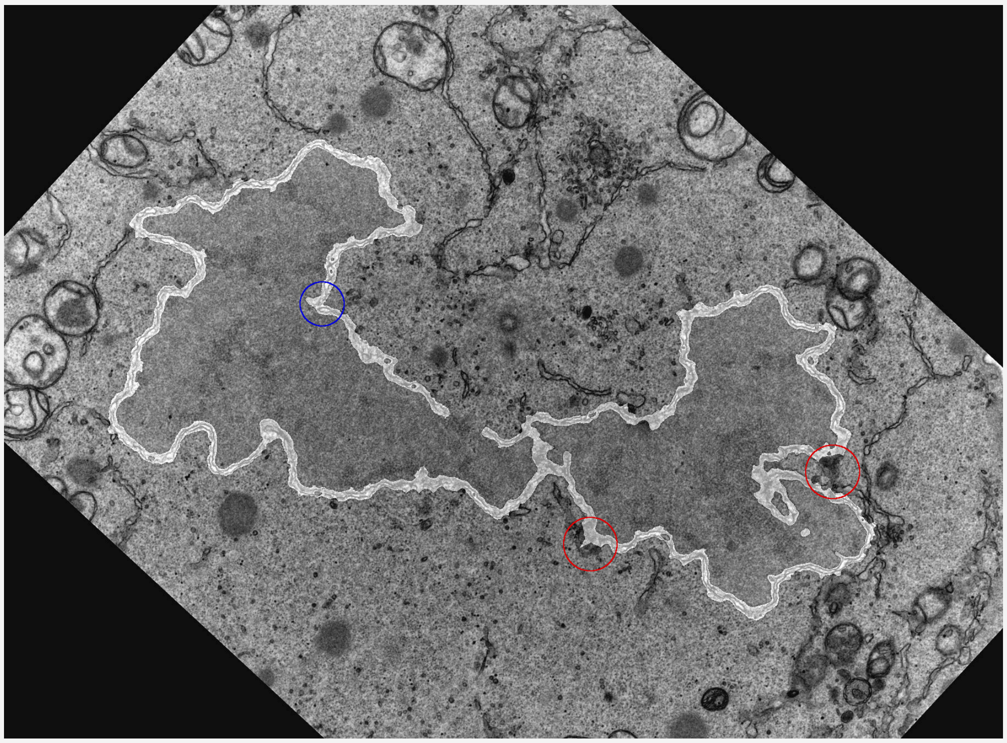
In general the problems noticed by the end-users performing a scoring of the evaluation dataset are those mentioned in Chapter 10. Both Dr. Peddie and Dr. Collinson noticed a general trend that the scores drop off towards the end of the dataset; indeed the average score for the first half of the dataset is above 7 compared to 5 for the second half of the dataset. For a comparison we can use Figures 11.5 and 11.6.

I think that there are a couple of reasons which together go some way to explaining this trend:

- Towards the end of the dataset there tend to be more disconnected parts of the nuclear envelope or loops which are discounted due to the failure to obtain a really accurate nucleus approximation. These parts that are missed by the algorithm also tend to be larger, as we can see in Figure 11.6a.
- The nuclear envelope and the cutting blade tend to be close to perpendicular more often in the later slices, giving a well defined nuclear envelope [1]. This is particularly apparent towards the bottom of Figure 11.6b where we can see many quite bright gaps between the double-lipid bi-layer structure that are not captured by the segmentation.
- Some of the later slices have small changes in the overall focus and brightness. This is quite difficult to see in the examples; a higher resolution makes this more apparent.

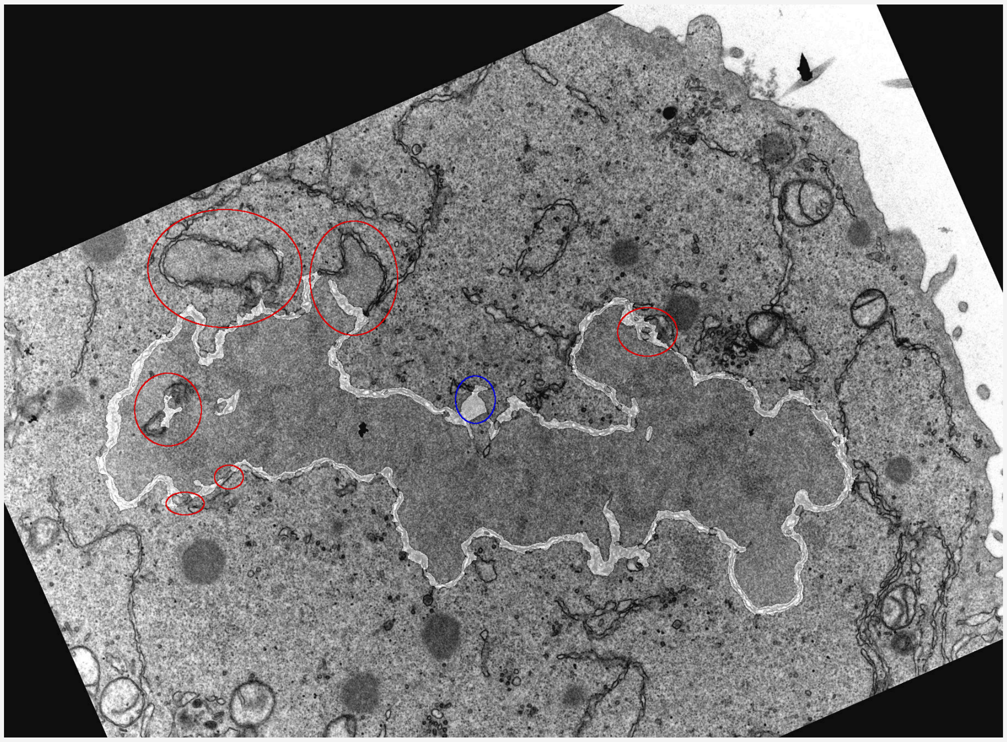


(a) Nuclear envelope segmentation overlaid onto slice 0022

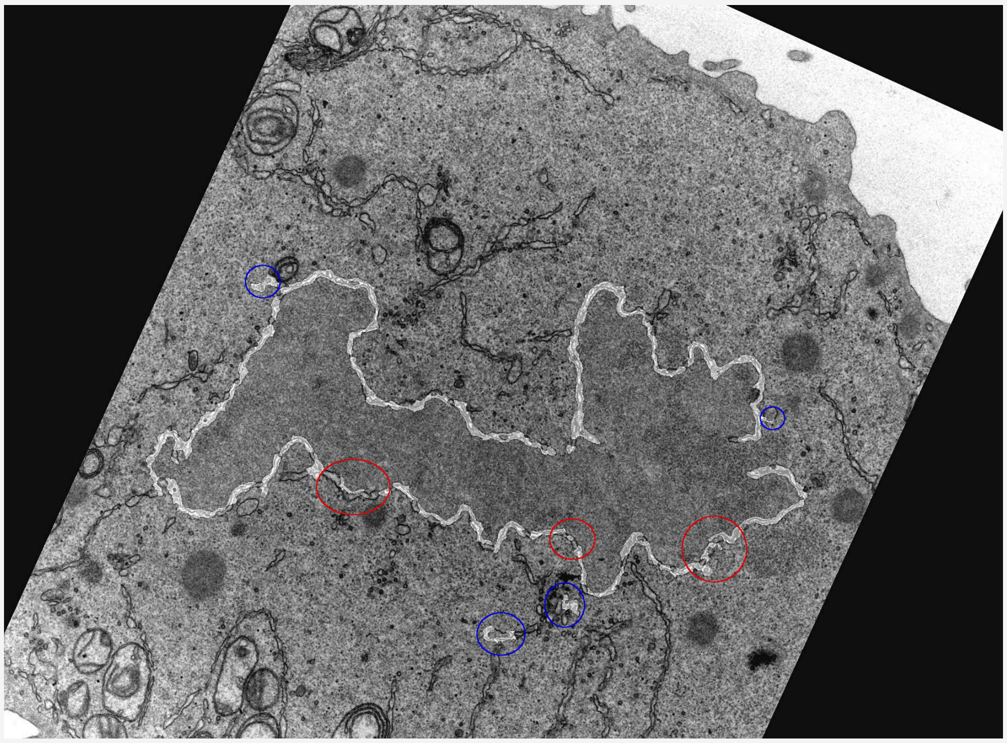


(b) Nuclear envelope segmentation overlaid onto slice 0034

Figure 11.5: High scoring slices from end-user scoring.



(a) Nuclear envelope segmentation overlaid onto slice 0056



(b) Nuclear envelope segmentation overlaid onto slice 0066

Figure 11.6: Low scoring slices from end-user scoring.

11.4 End-user Feedback

The dataset used in this study was chosen as it represents a "control" state and as such is a good starting point for the development of a automatic segmentation algorithm. However, it also contains a higher than usual number of artifacts, including dust and dirt on the sections, folds in the sections, uneven section shrinkage/expansion, and counterstain deposits. Yet, these problems illustrate perfectly the issues that must be overcome in order to successfully automatically segment complex datasets.

Given that the "state-of-the-art" segmentation methods currently available are only really suitable for use on very simplistic datasets and largely fail in an application such as that demonstrated here, the overall result obtained from this automatic segmentation algorithm is remarkable, particularly when comparing 3-dimensional models from manual vs. automatic output. That the algorithm over-segments a number of smaller features such as nuclear pores and small membrane gaps is of relatively little consequence when considering only the overall fit of the output. These small features are most likely best left for the end user to add or subtract manually as their classification can be quite subjective in any case. In evaluating the output, I mostly ignored small over-segmentation errors and concentrated instead on large over-segmentation errors and on under-segmentation, both of which are more likely to influence the 3-dimensional outcome. The images were scored within the series without direct comparison to the manual segmentation. In this case, it is clear that although some images scored poorly, their inclusion in the 3-dimensional reconstruction did not significantly alter the overall appearance of the surface model.

Although fine tuning of the output would still be required by the end user, the most significant impact of this algorithm of course is in the reduction of hours necessary to obtain a sufficiently high quality segmentation of a given dataset. We have estimated that it might be possible to reduce the amount of time spent segmenting a feature such as the nuclear envelope from as much as two to three weeks, to as little as one or two days. It will be extremely interesting to see how much this algorithm can be developed and further adapted to extend application to other datasets and image features, whilst incorporating end user interaction.

Dr. Christopher Peddie

Electron microscopy is capable of imaging cellular structure at nanometre resolution, and with recent advances in serial imaging, can do so in three dimensions to produce information on how proteins, cells and tissues interact in health and disease. However, as 3D data collection becomes simpler and faster, so the quantity of data produced is increasing exponentially, and the ability to analyse the data is becoming the major bottleneck in the workflow. EM is especially challenging, as many features of interest share the same grey values and so cannot be segmented using efficient thresholding algorithms.

Almost every mammalian cell has a nucleus. In many diseased cell types, the nucleus is disrupted or multiple nuclei are present. Therefore, the ability to segment the nucleus through serial EM images will not only give valuable information on the process of nuclear envelope formation but also the nature of mutations in disease-associated genes.

Currently, segmentation of the nuclear envelope from serial images through one cell takes several weeks. This cannot be done by an untrained individual - many years of experience is required for an electron microscopist to recognise and interpret cellular structures. The improvement in segmentation speed given by Stephen's workflow (from 2-3 weeks to one day) will significantly shorten the time spent on image analysis and hence relieve the bottleneck in the process. The expert electron microscopist *will be released from segmentation and able to return to wet-lab and imaging work. The success of this project is also demonstrated by the similarity of the 3D visualisation of the nuclear envelope by automatic and manual segmentation. The algorithms, once tested against nuclei in different cell types and different imaging modalities, will find wide application for cell biologists.*

The ability to work across disciplines, as demonstrated by Stephen in this project, should not be underestimated. Not only does the cell biologist have to learn basic image analysis principles, but the computer vision scientist has to learn the language of cell biology.

Dr. Lucy Collinson

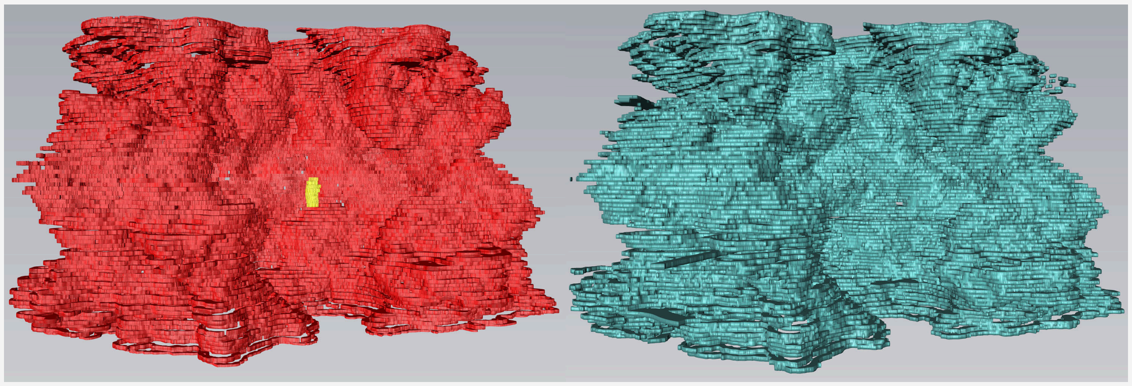


Figure 11.7: Viewing the 3D reconstructions of the manual and automated segmentations from the rear of the cell

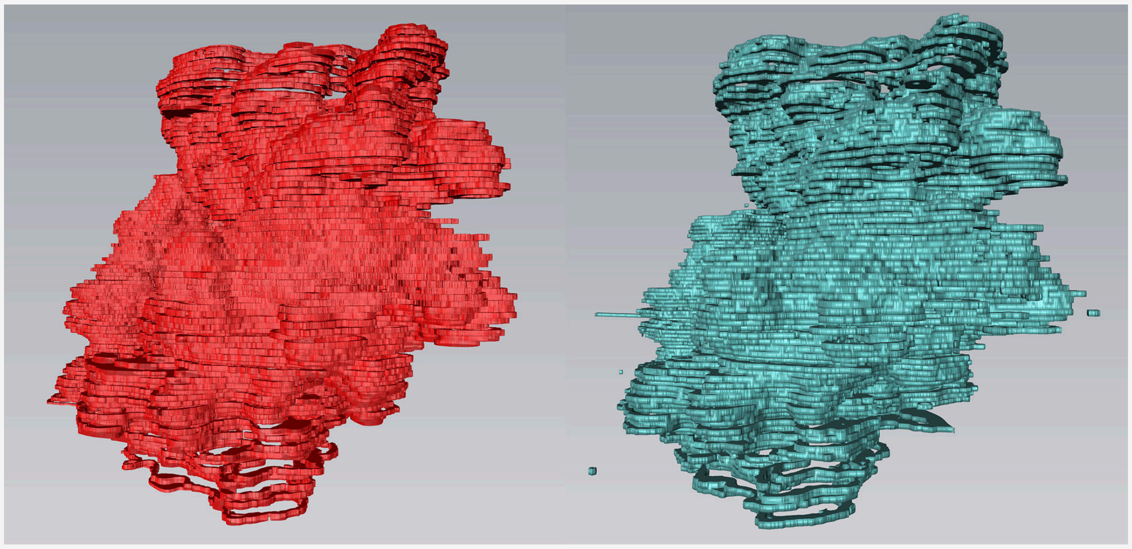


Figure 11.8: Viewing the 3D reconstructions of the manual and automated segmentations from the side of the cell

11.5 3D Reconstruction Similarity

The end goal of the segmentation process is to create a 3D reconstruction of the nuclear envelope which can then be viewed and studied. The overall shape and characteristics are important for the biological research into the formation of the nuclear envelope. Figures 11.7 and 11.8 show comparisons from two different viewpoints of the 3D reconstruction of the manual segmentation, shown in red, and the results of this algorithm, shown in teal, which have not been manually corrected.

The similarity between the two reconstructions is immediately noticeable, and we can see that there are only a few manual corrections that would have a visible effect on the overall shape of the reconstruction. This is one of the particular strengths of the process presented here; whilst the problems in Chapter 10 can cause local segmentation errors, in general the capturing of global shape is very good, as reflected in this comparison.

11.6 Segmentation at an Earlier Stage of Mitosis

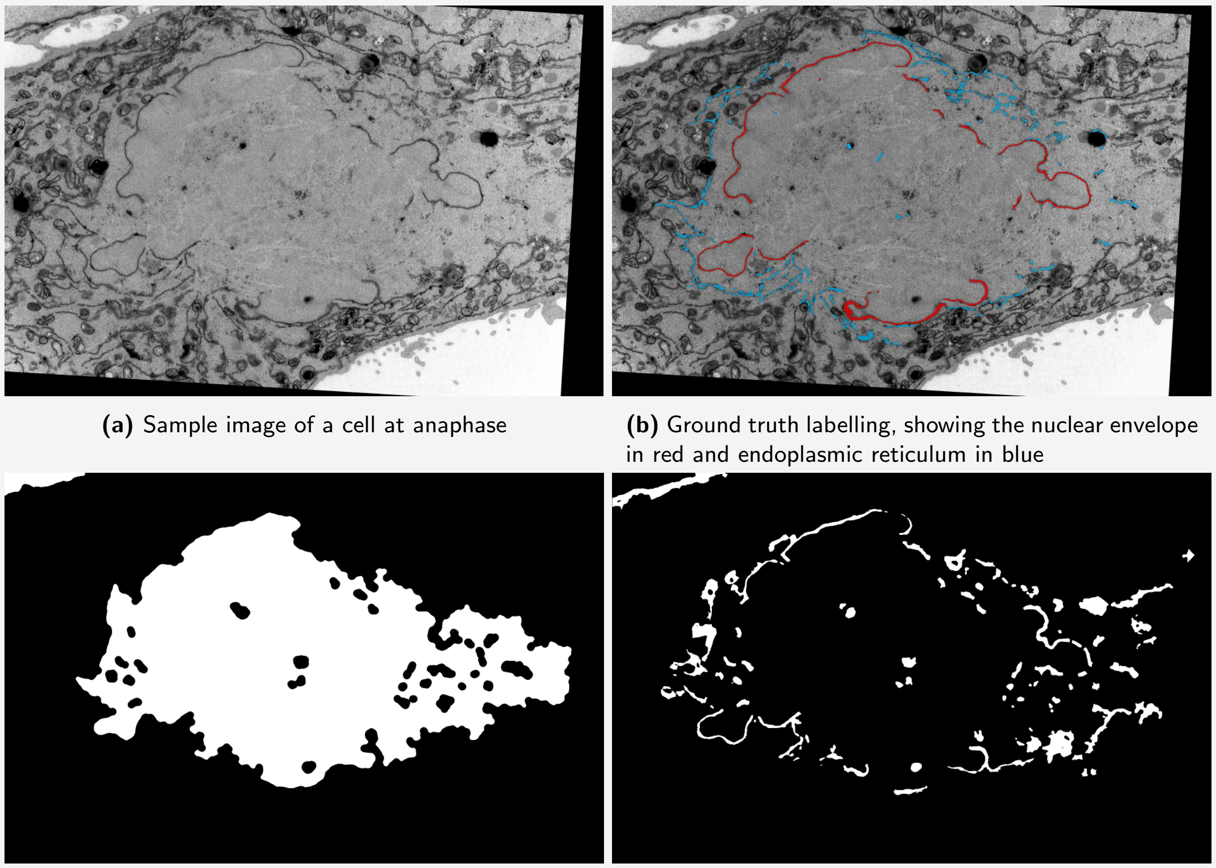
Whilst the focus of this project is upon the dataset of images of the cell at a late stage of mitosis ,it is interesting to see how well the algorithm generalises to the earlier stages. At these stages some of the assumptions for cells at telophase, such as minimal movement of the nuclear envelope between slices, break down, so filtering steps that involve 3D consistency may no longer be safe and would have to be adapted to incorporate the 3D slice information. An example of a nucleus at anaphase is shown in Figure 2.2b and the same slice is used in this evaluation starting with Figure 11.9.

One of the first key things that we notice in Figure 11.9a is that the texture difference between the nucleus and cytoplasm is less easy to spot, most likely due to the lack of a complete nucleus structure. It is however still present, as the texture towards the centre of the image is quite different to that towards the right edge. Other visual differences between the cell at anaphase and telophase (Figure 2.2c for example) include:

- The nuclear envelope is much more fragmented with many large gaps between its sections, which will limit the applicability of pruning connected structures.
- Outside of the nucleus area the cell structures are a lot more cluttered and thus provide more help for the algorithm in terms of those areas having high variance.
- Considerably more endoplasmic reticulum is present and is closer to the nuclear envelope than it is at other stages of mitosis.
- The nuclear envelope is much thinner than at telophase and the double lipid bi-layer structure is less significant.
- More dark areas are present within the nuclear envelope boundary that are no classified as nuclear envelope.

The other images in Figure 11.9 show the manual segmentation, nucleus approximation and automated segmentation after running the sample image through the algorithm. The performance is not as clear cut as it is for the cells at the late stage of mitosis used throughout this report, however I am primarily looking for how well the areas of nuclear envelope are captured; the many large gaps reduce the performance of pruning connected structures considerably so non-nuclear envelope structures were expected in the segmentation.

We can see that some nuclear envelope sections are only partially captured because they are initially found to be inside the nucleus approximation in Figure 11.9, and I make the assumption that any pixels in the nucleus approximation cannot be part of the nuclear envelope and prevent them from being segmented as such. The reason for these pixels being erroneously segmented as nucleus is because of the reduced thickness of the nuclear envelope. One step in the algorithm uses a morphological closing on the nucleus approximation to remove insignificant holes and to generally smooth the outline, but the radius of the circular structuring element is too large for the cell at anaphase as it sometimes spans the nuclear envelope. Figure 11.10 shows the nucleus approximation and resulting segmentation of the the same cell using a small radius closing operator, with a marked improvement in proportion of nuclear envelope captured. However, once this change has been made it introduces another problem of capturing many other dark structures in the cell that are not classified as nuclear envelope.


(a) Sample image of a cell at anaphase

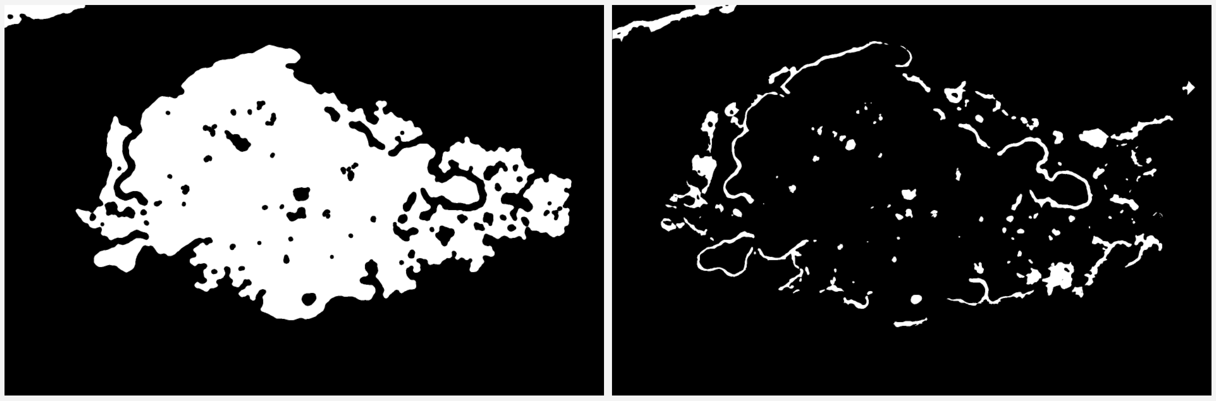
(b) Ground truth labelling, showing the nuclear envelope in red and endoplasmic reticulum in blue

(c) Nucleus approximation

(d) Segmentation

Figure 11.9: Evaluation of the algorithm on a cell at anaphase.

Further problems exist with defining the nucleus approximation, as the smaller difference in texture causes the flood fill to capture area outside the nucleus. Overall the segmentation quality of the cell at anaphase is considerably lower than of those at telophase, but that is to be expected given the differences in features and the assumptions that can no longer be made. The type of image exemplified here pose a different set of challenges to those tackled within this project, however I do believe that many of the concepts and ideas can be carried forward to a next iteration of the algorithm.



(a) Nucleus approximation **(b)** Segmentation

Figure 11.10: Showing the effect of reducing the size of the morphological closing operator to prevent closing holes that contain nuclear envelope.

Conclusion 12

Results

With a minimal amount of manual corrections to the automated segmentation the overall time to segment the sample dataset has been reduced from 2-3 weeks to on average a day. This a significant reduction in time and one that should make an impact to the throughput of the datasets from the manual segmentation experts, and particularly for those datasets of cells at the same stage in mitosis. This is the first investigative work into segmentation of the nuclear envelope so these are excellent results. Furthermore, a comparison to the best state of the art software also shows that the results achieved are a considerable improvement upon those currently possible from other existing methods.

What makes this significant reduction possible is the use of many simple image processing techniques to create a pipeline that works together from the inside of the nucleus out towards the nuclear envelope. The significant difference in texture between the cytoplasm and the inside of the nucleus is the key factor to the success of this work, and further improvements in the ability to identify it will only further the accuracy of the segmentation process. As we have seen, more complex segmentation techniques based around machine-learning classification have been successful in recent research, but these are so far unnecessary for this application. The use of these simple image processing techniques provide many advantages, not least of which is a significant decrease in algorithm development and execution time.

We have also seen in Chapter 10 that there are many complexities that cause difficulties in the segmentation process. These problems have difficult solutions, if any at all, that require much more research and time to find. Many of them stem from the difficulties in segmentation of electron microscopy images as a whole. Again, however, it is fortunate that they do not significantly impact the overall quality of the 3D reconstruction as they only affect small sections of nuclear envelope, where segmentation errors can be fixed manually. We can see this from the 3D reconstruction similarity, without considering manual corrections, in Figures 11.7 and 11.8, which I think is the best proof of the success of this project.

A general framework for further segmentation algorithms for structures in EM images was presented in Chapter 8. The advantages of such an implementation are to decouple the algorithm from a GUI, something that is in the pipeline to develop soon, and to enable easy testing and isolation of particular parts of the segmentation algorithm. It is always beneficial to employ good practises from the start of a project or piece of software.

Development

Having a good intuitive understanding of what different transformations provide really helped to see how they can be applied in other ways. For instance, the use of a skeleton transformation for reducing the number of seed points in local regions for finding the nucleus approximation. This gives a significant decrease in the number of seed points found, with a knock-on effect in terms of computation time. A good intuitive understanding of the underlying algorithms and techniques always helps to think of ways in which they can be applied outside their normal uses and applications that you do not find in a textbook.

The dataset upon which the project was based was perhaps underestimated in terms of its complexity. The temptation when approaching this type of segmentation application is to study the images and segment them as a human, rather than considering how a computational solution might approach the problem. There is no doubt that the typical image of a cell obtained through electron microscopy contains a lot of complexities and intricacies, but I think what makes this application particularly difficult is the contrast in complexities at different levels of computational vision. The images tend to be most complex in the high-level vision range; structures have complex shapes and interactions with each other, making analysis harder for both human and computer. However for lower-level vision there is little information as the image is greyscale and furthermore the majority of the pixels reside in a smaller subset of the 0 to 255 greyscale range for 8-bit images. It is possible that colour images could immediately give much more descriptive features at a lower level, but that is a restriction of the electron microscopy technique.

The evaluation of the algorithm's segmentation of a cell at a different stage of mitosis shows that the nuclear envelope algorithm developed for the cell at telophase provides a base for further development, but still has some way to go before being generally applicable to cells at all stages of mitosis. These different stages provide many more complexities that will definitely require extensive research before they can be solved.

Further Research 13

13.1 Segmentation of other Sub-cellular Structures

The vision of the end-users at CRUK is to have a general purpose toolkit for the segmentation of all interesting sub-cellular structures, most of which are introduced in this report. This is important considering their end goal of defining how the formation of the nuclear envelope is regulated by these other structures, so it is important to examine and perhaps quantify their interactions with the nuclear envelope.

It is also possible that obtaining segmentations for other structures could increase the accuracy of the segmentation for the nuclear envelope. Take for example the vesicles, of which there are often many within the cytoplasm and never any inside the nucleus. Given their close spatial clustering, we can imagine using them to artificially increase the variance within the cytoplasm. A simple way to do this is to draw minimum or maximum intensity lines joining all vesicles within a certain neighbourhood size. The variance of regions within the image is the most important feature when carrying out the flood fill algorithm to find an approximation to the nucleus, and if we could artificially increase the variance outside of the nucleus, whilst retaining the original low variance inside, it should be possible to obtain a more accurate approximation.

The locating of holes in the nuclear envelope is something that requires a good segmentation of the nuclear envelope, but also a way of closing the boundary. As previously discussed in Section 10.6 this is not something that is easy due to complexities of the shape of the nucleus. Being able to reliably close the boundary contours is also something that becomes a lot more difficult when the nuclear envelope is much less defined, as at earlier stages in the mitotic process.

A suitable method for segmentation of the endoplasmic reticulum could be a classifier based approach. One of the difficulties that limits the applicability of the classification approach to segmenting the nuclear envelope is that of distinguishing the endoplasmic reticulum and the nuclear envelope, as the only information to do so is their spatial proximity to the nucleus. The segmentation of the nuclear envelope as presented herein provides two advantages.

- As the nuclear envelope and endoplasmic reticulum have similar features, it can be seen as an automated process for generating training data for a classifier based approach.
- We already have the segmentation for the nuclear envelope so the classifier does not need to distinguish between the two structures.

This should allow further research to get close to a fully automated approach to segmenting the endoplasmic reticulum.

13.2 GUI and Manual Corrections

It was always one of the aims of this project to provide the end-users with a interface such that they can begin to use the algorithm. Time constraints, and the view that a GUI is not the most important aspect of a research project mean that this task has been left out of my work. This being said, I do not believe that creating a suitable GUI would be too time consuming; the algorithm framework operates independently of any front-end application.

Using manual corrections of segmentations from the expert users is critical to the quality of the achieved segmentations and this would become an essential part of the GUI. There are many ways of interacting with the user for obtaining manual corrections; the most simple would be to use a black or white paint brush of varying dimensions, painting onto the output of the automated segmentation as it is shown overlaid on the original image. This could be put in place to modify non-corrupted slices where the segmentation may only need minor tweaks.

For the case where 3D interpolation gives an approximation to a corrupted slice a more intuitive method of manipulating the segmentation would be to use a skeleton representation. Junctions could be introduced to the skeleton and vertices moved around in order to make large movements of the segmentation easier, similar to manipulating a curve in an interactive drawing package.

13.3 Manual Seed Point Input

Another method through which we can be assured to obtain good seed points for both the nucleus and nuclear envelope flood fill is enabling the end-user to check, correct and add seed points to the sets used. Most of the hard work of the flood filling to obtain regions would still be automated, and if the seed points found are good then this step would be short anyway. The trade-off here is increasing the amount of work for the manual user and hence the segmentation time, but an increase in the quality of the segmentation and the alleviation of some of the problems encountered in Chapter 10 may offset this.

13.4 Investigation of Nuclear Envelope Characteristics during Mitosis

Throughout the process of mitosis the nuclear envelope breaks down in order for the nucleus to split and the daughters to form. Given this process, at any given stage of mitosis the nuclear envelope could not be present, partially reformed or be a complete boundary around the nucleus. Looking at the example of a cell at a different stage of mitosis in Section 11.6, it is clear that the quality of the segmentation produced by this algorithm is highly dependent on the completeness of the nuclear envelope.

For the different stages of mitosis some adaptations will need to be made to the algorithm. An increase in the number of gaps in the nuclear envelope will increase the risk that the texture feature will not be quite descriptive enough to prevent leakages of the flood fill algorithm into the cytoplasm. This is one of the problems we do see in Figure 11.9.

It may be possible to offset the downsides of the algorithm when run on images of cells at a different stage of mitosis by investigating whether cells at these stages have any other features that could aid a phase-specific segmentation. Manual input from the user of approximately which stage of mitosis the cell is at may therefore improve further segmentations and versions of this algorithm.

13.5 Quantifying Nuclear Envelope Characteristics

After the segmentations produced by the algorithm have been corrected by the end-user, statistics could be developed to quantify certain characteristics of the nuclear envelope. Simple characteristics such as thickness and area come to mind, but others such as curvature and elongatedness may be useful but slightly more difficult to compute. Characteristics tailored towards the investigation of the Cell Biophysics group at CRUK are also possible, perhaps including the number of gaps in the nuclear envelope, or their average size.

If a good segmentation of other structures within the cell is also developed then further characteristics of both those structures and how all the structures interact could prove useful for their research. An example of such a statistic could be the distribution of distance of vesicles from the nucleus. If the locations of the vesicles and the nuclear envelope are known, then a distance map technique could be used to find the distance from vesicles to their nearest point on the nuclear envelope. How these distances are distributed may provide further insight into the regulation of the nuclear envelope formation.

13.6 GPU Computation

Nearly all the different steps in this algorithm can be parallelised to the point that the overheads of such parallelisation and marshalling are negligible with respect to runtime; there are enough slices such that there would always be work to be done. An extension of the normal multi-threaded Java code shown in Section 7.3 is to utilise the massive capabilities of GPUs, where image filtering and processing methods are very natural and fast. The computers available to the end-users have powerful graphics cards, so the speed up may be quite considerable given that no use is currently made of the GPU processing abilities.

Interfacing with GPUs from Java is not something that is naturally within its capabilities, but it is possible and some tutorials and guidelines exist online to aid development in this paradigm. I do think this would be a very worthwhile investment of time purely for the potential runtime savings, regardless of any changes in the overall algorithm.

13.7 An Uncertainty Measure from 3D Consistency

The manual correction process could be made more efficient by drawing the user's eye to areas in the results where we are not sure of the segmentation quality. A simple way of defining this would be to build upon the idea of support from neighbouring slices as used in the 3D consistency steps in the algorithm.

Again taking the minimal movement assumption, we could build an evidence map for a particular slice based upon the locations of the segmentations in its neighbouring slices. Comparing this to the particular slice's segmentation would highlight areas where:

- Sections of nuclear envelope are missing, possibly because they have been pruned erroneously (Section 10.5) or chunks are missing due to well defined perinuclear gaps (Section 10.3).
- Structures that are not the nuclear envelope are segmented as such, for example when structures fail to be pruned because of gaps in the nuclear envelope (Section 10.6).

If the aim of the manual corrections is to produce a globally correct shape, and small errors in the nuclear envelope segmentation are negligible, then identifying these areas through uncertainty maps in a user interface would be a useful feature. Furthermore, an uncertainty metric could be produced for each slice as a whole by summing values in the uncertainty map. This could guide the user to particularly poorly segmented slices to focus their attention.

Bibliography

[1] C. Peddie, in discussion.

[2] L. Collinson, in discussion.

[3] B. Larijani, in discussion.

[4] D. Rueckert, in discussion.

[5] L. Pizarro, in discussion.

[6] Fast Marching Methods: A boundary value formulation. `http://math.berkeley.edu/~sethian/2006/Explanations/fast_marching_explain.html`.

[7] ImageJ Documentation Wiki. `http://imagejdocu.tudor.lu/`.

[8] Spatial Filters - Gaussian Smoothing. `http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm`.

[9] Superpixel: Empirical Studies and Applications. `http://ttic.uchicago.edu/~xren/research/superpixel/`.

[10] The Cell Cycle & Mitosis Tutorial. `http://www.biology.arizona.edu/cell_bio/tutorials/cell_cycle/cells3.html`.

[11] M. D. Abramoff, P. J. Magelhaes, and S. J. Ram. Image processing with ImageJ. *Biophotonics Int*, 11(7):36–42, 2004.

[12] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. SLIC Superpixels. Technical Report 149300, EPFL, June 2010.

[13] B. Andres, U. Köthe, M. Helmstaedter, W. Denk, and F. A. Hamprecht. Segmentation of SBFSEM Volume Data of Neural Tissue by Hierarchical Classification. In *Proceedings of the 30th DAGM symposium on Pattern Recognition*, pages 142–152, Berlin, Heidelberg, 2008. Springer-Verlag.

[14] I. Arganda-Carreras. AnalyzeSkeleton [ImageJ Documentation Wiki]. `http://imagejdocu.tudor.lu/doku.php?id=plugin:analysis:analyzeskeleton:start`.

[15] I. Arganda-Carreras. Skeletonize3D [ImageJ Documentation Wiki]. `http://imagejdocu.tudor.lu/doku.php?id=plugin:morphology:skeletonize3d:start`.

[16] J. E. Cabrera. Gray Level Correlation Matrix Texture Analyzer. `http://rsbweb.nih.gov/ij/plugins/download/GLCM_Texture.java`.

[17] A. Criminisi, J. Shotton, and E. Konukoglu. Decision Forests for Classifcation, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning. Technical Report TR-2011-114, Microsoft Research, October 2011.

[18] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *CVPR*, pages 886–893, 2005.

[19] P. Dollár, Z. Tu, and S. Belongie. Supervised Learning of Edges and Object Boundaries. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 1964–1971, Washington, DC, USA, 2006. IEEE Computer Society.

[20] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient Graph-Based Image Segmentation. *Int. J. Comput. Vision*, 59:167–181, September 2004.

[21] E. Frise. Level Sets - Fiji. `http://fiji.sc/wiki/index.php/Level_Sets`.

[22] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[23] R. Jain, R. Kasturi, and B. G. Schunck. *Machine Vision*. McGraw-Hill, 1995.

[24] V. Jain, B. Bollmann, M. Richardson, D. R. Berger, M. N. Helmstaedter, K. L. Briggman, W. Denk, J. B. Bowden, J. M. Mendenhall, W. C. Abraham, K. M. Harris, N. Kasthuri, K. J. Hayworth, R. Schalek, J. C. Tapia, J. W. Lichtman, and H. S. Seung. Boundary Learning by Optimization with Topological Constraints.

[25] V. Jain, S. Seung, and S. C. Turaga. Machines That Learn to Segment Images: a Crucial Technology for Connectomics. *Current Opinion In Neurobiology.*, 10:1–14, 2010.

[26] V. Jain, S. C. Turaga, K. Briggman, M. N. Helmstaedter, W. Denk, and H. S. Seung. Learning to Agglomerate Superpixel Hierarchies. In J. Shawe-Taylor, R. S. Zemel, P. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 648–656. 2011.

[27] I. Kononenko and M. Kukar. *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Horwood Publishing, 2007.

[28] A. Kreshuk, C. N. Straehle, C. Sommer, U. Köthe, G. Knott, and F. A. Hamprecht. Automated segmentation of synapses in 3D EM data. In *ISBI*, pages 220–223. IEEE, 2011.

[29] B. Larijani and D. L. Poccia. Nuclear envelope formation: mind the gaps. *Annu. Rev. Biophys.*, 38:107–24, 2009.

[30] M. Longair. "Find Connected Regions" ImageJ Plugin. `http://www.longair.net/edinburgh/imagej/find-connected-regions/`.

[31] A. Lucchi, Y. Li, X. Boix, K. Smith, and P. Fua. Are Spatial and Global Constraints Really Necessary for Segmentation?

[32] A. Lucchi, K. Smith, R. Achanta, G. Knott, and P. Fua. Supervoxel-Based Segmentation of Mitochondria in EM Image Stacks with Learned Shape Features. *IEEE Transactions on Medical Imaging*, 30(11), 2011.

[33] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

[34] B. R. Masters. History of the Electron Microscope in Cell Biology. In *Encyclopedia of Life Sciences (ELS)*. John Wiley & Sons, Ltd: Chichester, March 2009.

[35] E. Meijering. FeatureJ: A Java Package for Image Feature Extraction. `http://imagescience.org/meijering/software/featurej/`.

[36] M. Nixon and A. Aguado. *Feature Extraction and Image Processing*. Newnes, 2002.

[37] M. Petrou and P. G. Sevilla. *Image Processing: Dealing with Texture*. Wiley, 2006.

[38] W. M. Rand. Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

[39] J. Schindelin. Fiji is just imagej (batteries included). ImageJ User and Developer Conference, 2008.

[40] B. Schmid. ImageJ 3D Viewer. `http://rsbweb.nih.gov/ij/plugins/3d-viewer/`.

[41] J. Shi and J. Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 1997.

[42] J. Shi, D. Martin, C. Fowlkes, and E. Sharon. Tutorial: Graph Based Image Segmentation. `http://www.cis.upenn.edu/~jshi/GraphTutorial/Tutorial-ImageSegmentationGraph-cut1-Shi.pdf`.

[43] K. Smith, A. Carleton, and V. Lepetit. Fast Ray Features for Learning Irregular Shapes. In *2009 IEEE 12th International Conference on Computer Vision (ICCV)*, IEEE International Conference on Computer Vision, pages 397–404, 2009.

[44] C. Sommer, C. Straehle, U. Koethe, and F. A. Hamprecht. ilastik: Interactive Learning and Segmentation Toolkit. In *8th IEEE International Symposium on Biomedical Imaging (ISBI 2011)*, 2011.

[45] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis and Machine Vision*. Thomson Learning, 3rd edition, 2008.

[46] P. Thvenaz. Image Differentials. `http://bigwww.epfl.ch/thevenaz/differentials/`.

[47] S. Turaga, K. Briggman, M. Helmstaedter, W. Denk, and S. Seung. Maximin affinity learning of image segmentation. In *Advances in Neural Information Processing Systems 22*, pages 1865–1873. 2009.

[48] L. Vincent and P. Soille. Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13:583–598, June 1991.

[49] J. Walter. FFT Filter. `http://rsbweb.nih.gov/ij/plugins/fft-filter.html`.

[50] A. Weston, H. Armer, and L. Collinson. Towards native-state imaging in biological context in the electron microscope. *J Chem Biol*, 2009.

[51] Guang-Zhong Yang. 13. Image Registration. Computer Vision Course Lecture Notes, *Department of Computing, Imperial College London*.

[52] Guang-Zhong Yang. 4. Region Based Segmentation. Computer Vision Course Lecture Notes, *Department of Computing, Imperial College London*.

[53] Guang-Zhong Yang. 5. Colour and Texture. Computer Vision Course Lecture Notes, *Department of Computing, Imperial College London*.

A Implementation

A.1 AlgorithmInstance Class Definition

```java
package segmentation.algorithms;

import ...;

public abstract class AlgorithmInstance
  implements Comparable<AlgorithmInstance>
{
  public ImagePlus originalImage;
  public ImagePlus scaledImage;

  public double scaleFactor;
  public int sliceNumber;
  public String imageID;
  public String srcLocation;
  public String saveLocation;
  public String tempDir;
  protected String tempLocation;

  public boolean noisy = false;

  public AlgorithmInstance(int sliceNumber, String imageID, String
      tempDir, double scaleFactor, String srcLocation)
  { ... }

  public abstract void serialise();
  public abstract void unserialise(int flags);

  public int compareTo(AlgorithmInstance instance)
  {
    if (this.sliceNumber == instance.sliceNumber)
    {
      return 0;
    }
    else if (this.sliceNumber < instance.sliceNumber)
    {
      return -1;
    }
    return 1;
  }
}
```

Listing A.1: Class definition of AlgorithmInstance

A.2 Algorithm Class Definition

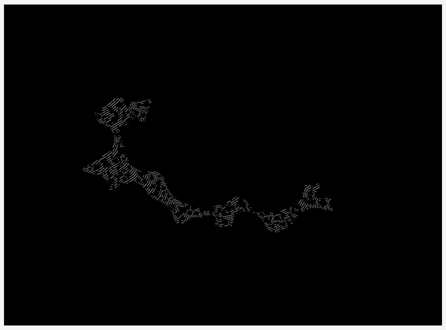
```
package segmentation.algorithms;

import ...;

public abstract class Algorithm<I extends AlgorithmInstance>
{
  private ArrayList<AlgorithmProcess> processes;
  private List<I> instances;

  private ArrayList<ProgressListener> progressListeners;
  private ArrayList<MessageListener> messageListeners;
  private ArrayList<StatusListener> statusListeners;

  public Algorithm() {...}
  protected abstract void setup();

  protected void addInstance(I instance)
  {
    instances.add(instance);
  }

  protected void addProcessingStep(AlgorithmProcess process)
  {
    processes.add(process);
  }

  public void run() throws AlgorithmProcessException
  {
    // Setup
    setup();

    for (final AlgorithmProcess process: processes)
    {
      // Unserialise just the images we need at this stage
      for (I instance: instances)
      {
        instance.unserialise(process.requiredImages());
      }

      if (process instanceof AlgorithmStep<?>)
      {
        ...
      }
      else if (process instanceof AlgorithmFilter<?>)
      {
        ...
      }

      // And serialise all the instances
      for (I instance: instances)
      {
        instance.serialise();
      }
    }
  }

  public void registerProgressListener(ProgressListener listener){...
  public void registerMessageListener(MessageListener listener){...
  public void registerStatusListener(StatusListener listener){...
}
```

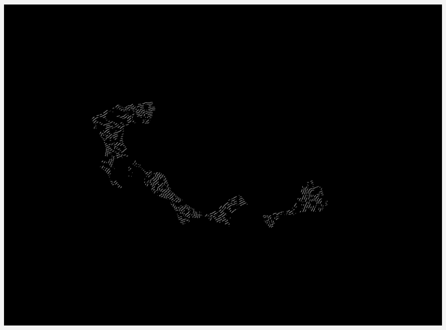
Listing A.2: Class definition of `AlgorithmInstance`

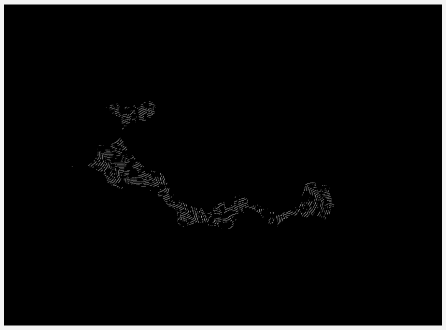
B

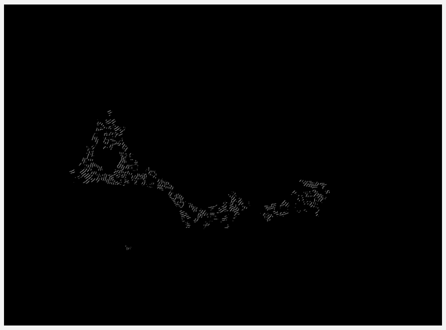
Algorithm

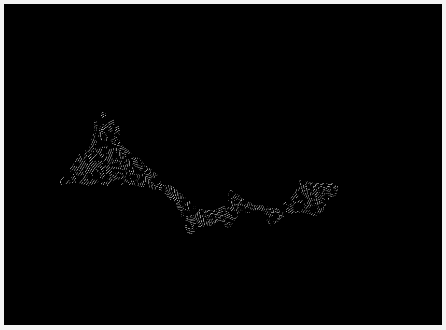
B.1 Seed Points for Neighbours of 0049


(a) Seed points for slice 0046


(b) Seed points for slice 0047


(c) Seed points for slice 0048


(d) Seed points for slice 0050


(e) Seed points for slice 0051

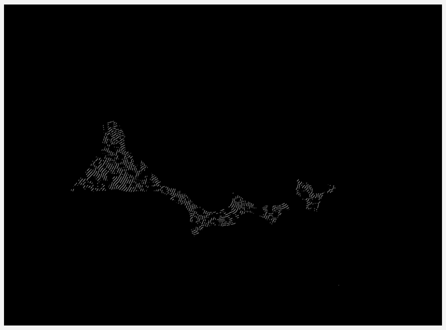
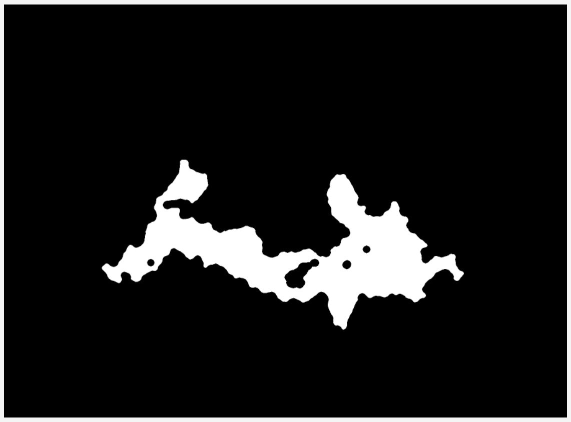
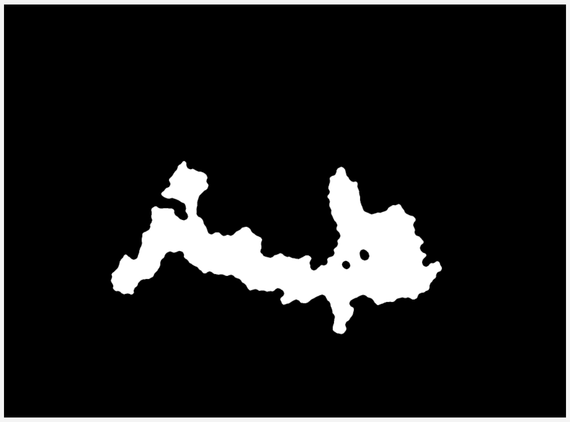
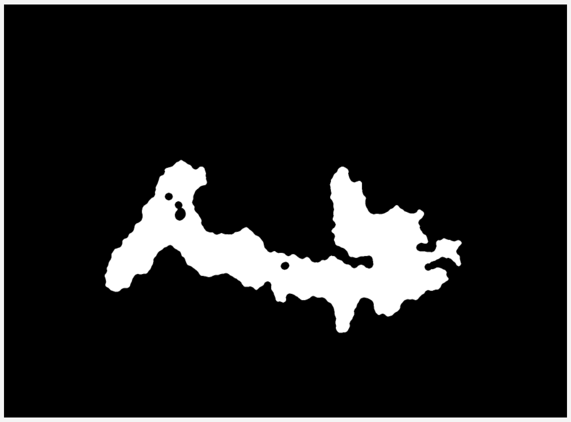

(f) Seed points for slice 0052

Figure B.1: Seed point support from slices surrounding number 0049.

B.2 Nucleus Approximations for Neighbours of 0064


(a) Nucleus approximation of slice 0062


(b) Nucleus approximation of slice 0063


(c) Nucleus approximation of slice 0065

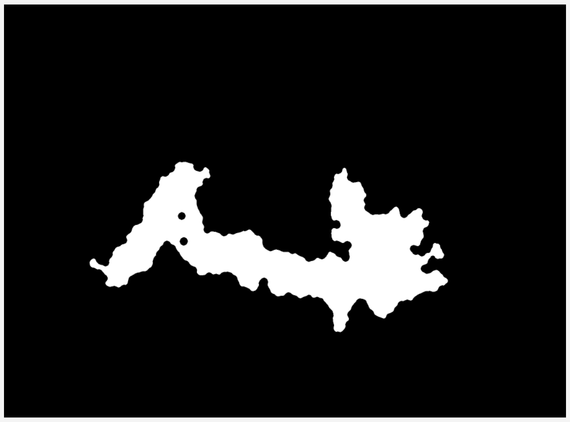

(d) Nucleus approximation of slice 0066

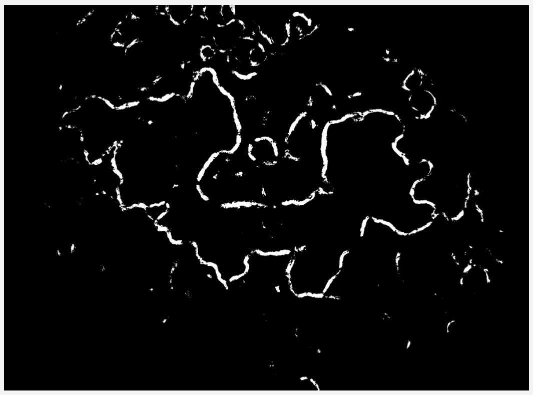
Figure B.2: Support from neighbours of 0064 for 3D consistency of the nucleus approximation.

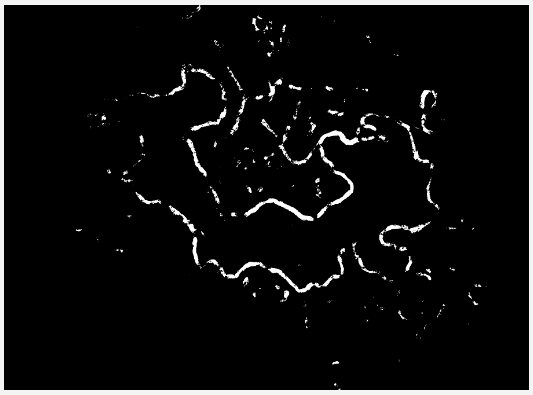
C Evaluation

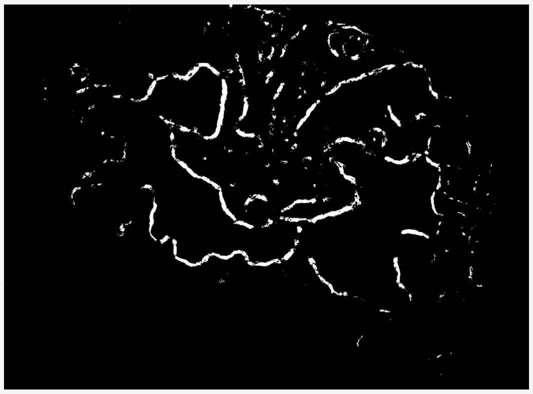
C.1 Error Metrics and Scoring Results

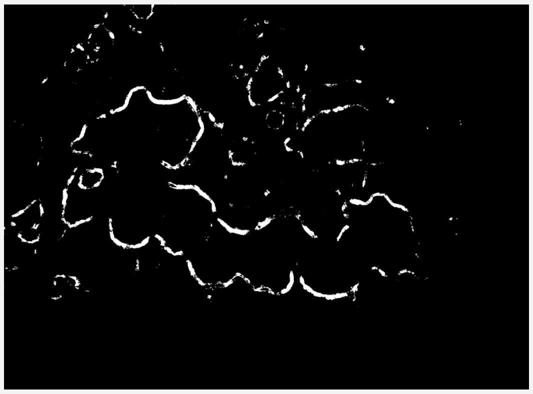
Image	Pixel Error	Jaccard Index	Rand Index	Dr. Peddie	Dr. Collinson
0000	0.0195	0.506	0.1519	7	9
0002	0.01705	0.5043	0.2607	8	9
0004	0.02168	0.5121	0.2145	9	9
0006	0.02408	0.4575	0.1563	6	9
0008	0.0441	0.2602	0.2373	2	2
0010	0.01762	0.5369	0.1255	10	8
0012	0.02131	0.4114	0.2297	7	7
0014	0.01994	0.4917	0.1247	8	9
0016	0.01772	0.5174	0.3319	6	8
0018	0.02108	0.5080	0.1741	7	8
0020	0.02041	0.5245	0.3904	6	7
0022	0.01352	0.5645	0.1725	10	9
0024	0.01879	0.5446	0.1628	5	7
0028	0.0231	0.4692	0.2176	5	8
0030	0.02623	0.4774	0.2667	6	7
0032	0.02698	0.4165	0.194	6	7
0034	0.02365	0.3976	0.2398	9	9
0036	0.02175	0.4423	0.2729	9	9
0040	0.02273	0.371	0.3043	6	7
0042	0.02289	0.3771	0.2834	8	6
0044	0.01812	0.4277	0.3617	8	8
0046	0.01965	0.4345	0.4683	6	7
0048	0.01593	0.481	0.367	5	6
0050	0.01647	0.5446	0.3453	6	7
0052	0.01667	0.508	0.4289	5	8
0054	0.01475	0.566	0.356	4	6
0056	0.02065	0.4536	0.2793	1	3
0058	0.02165	0.4371	0.2663	2	5
0060	0.01872	0.4466	0.3871	3	4
0062	0.01524	0.5095	0.229	3	7
0064	0.01332	0.4954	0.4016	3	6
0066	0.01181	0.4829	0.4065	1	0
0068	0.01539	0.5025	0.1699	5	8
0072	0.01443	0.5373	0.3308	4	7
0074	0.01229	0.5544	0.4806	2	4
0076	0.01654	0.5319	0.21	3	7
0078	0.01587	0.5258	0.3197	5	7

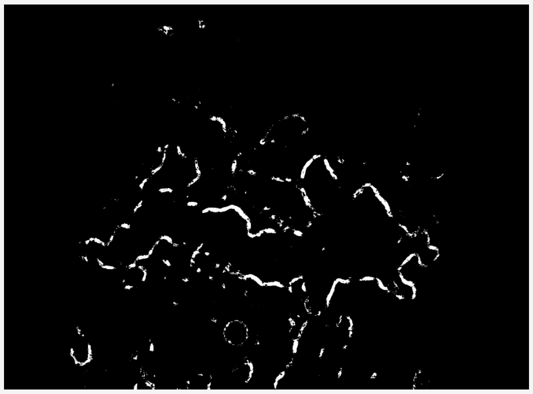
C.2 Ilastik Segmentations


(a) Slice 0010


(b) Slice 0018


(c) Slice 0028


(d) Slice 0046


(e) Slice 0062

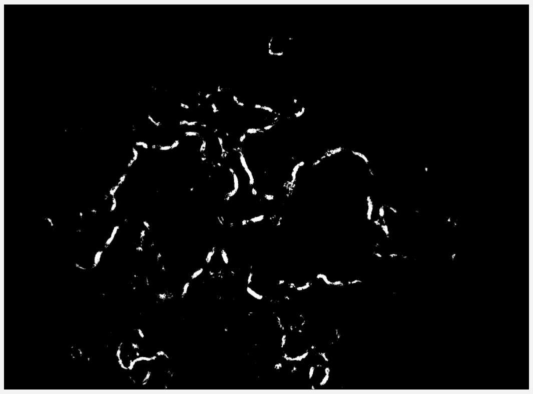

(f) Slice 0074

Figure C.1: Ilastik segmentations

D Results

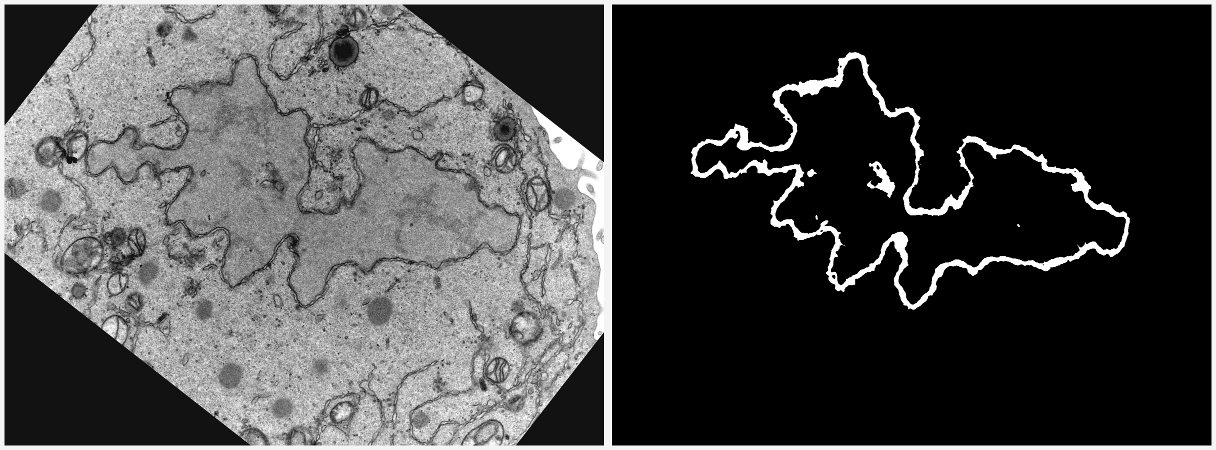


Figure D.1: Slice 0000

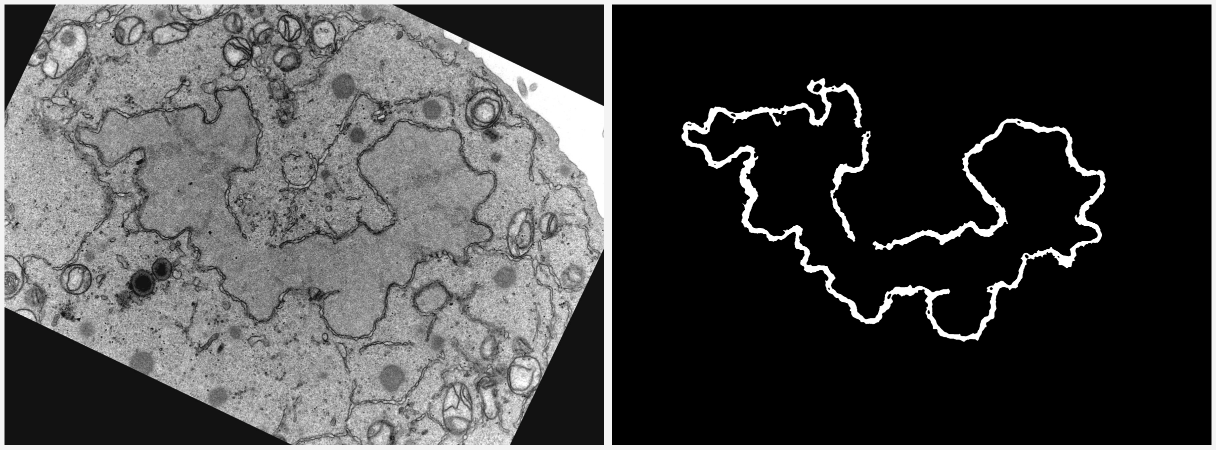


Figure D.2: Slice 0014

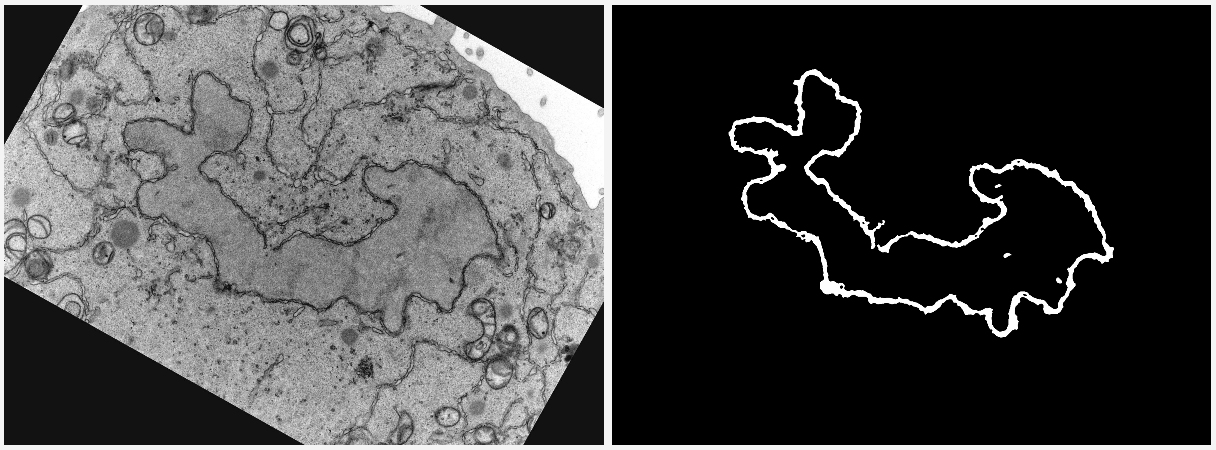


Figure D.3: Slice 0022

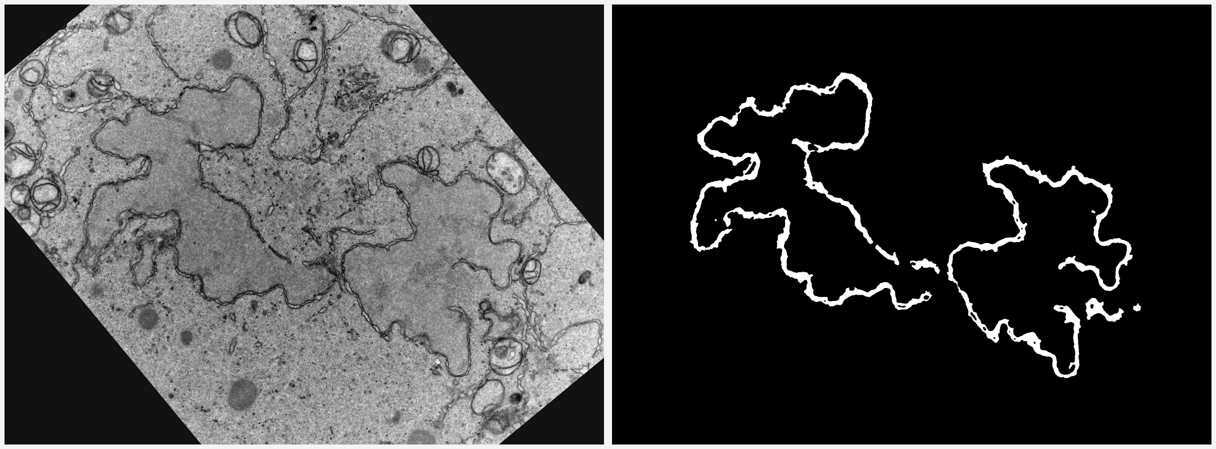


Figure D.4: Slice 0030

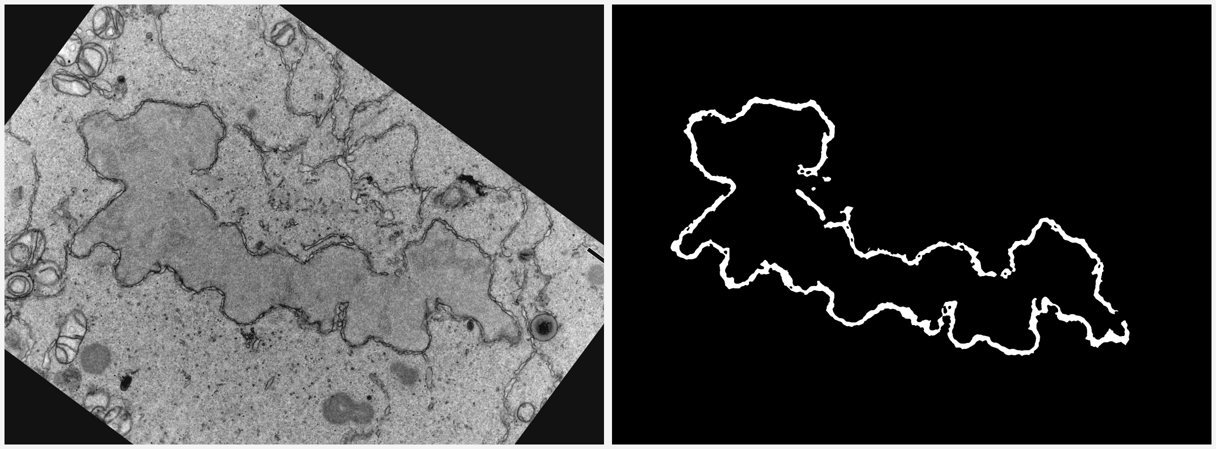


Figure D.5: Slice 0042

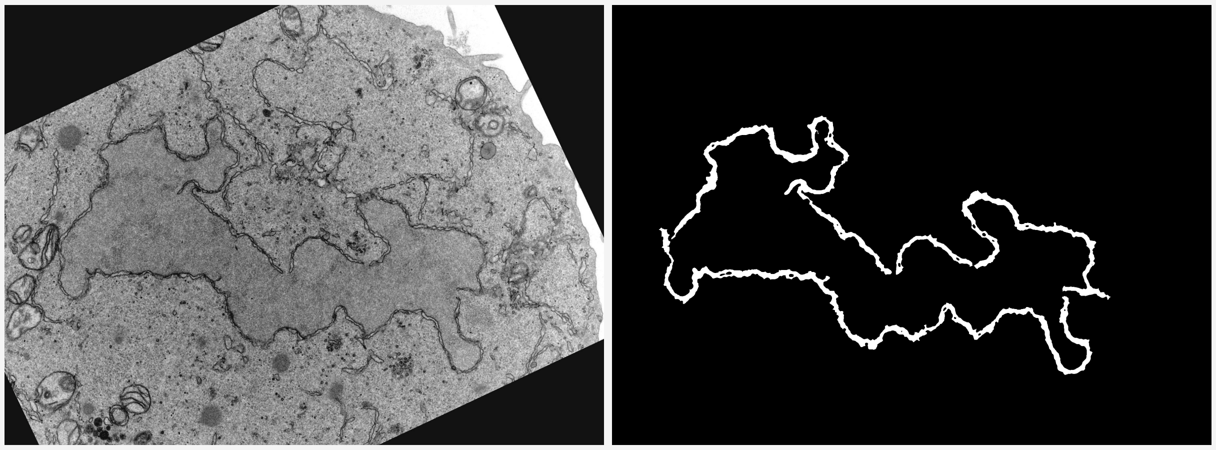


Figure D.6: Slice 0051

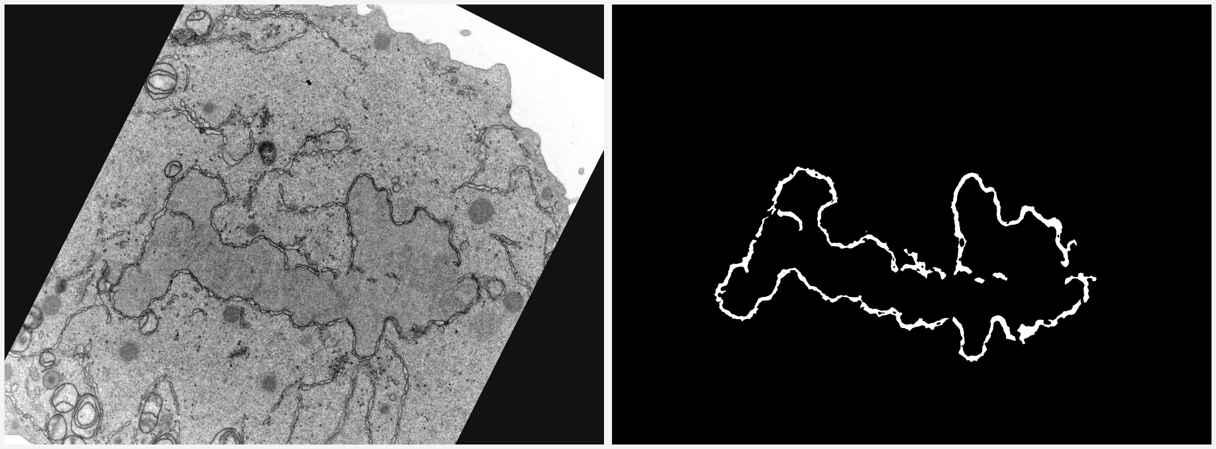


Figure D.7: Slice 0064

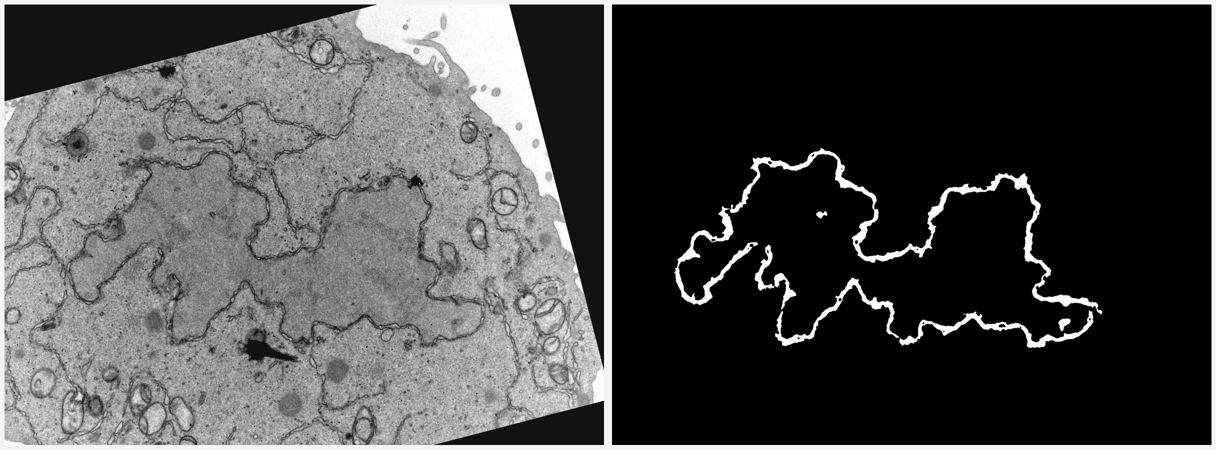


Figure D.8: Slice 0075