# Segmentation Techniques for Tracking Moving Cells in Time-Lapse Video Sequences

Alexander Goudemond
University of Kwa-Zulu Natal
Pietermaritzburg, South Africa
219030365@stu.ukzn.ac.za

Serestina Viriri
University of Kwa-Zulu Natal
Pietermaritzburg, South Africa
viriris@ukzn.ac.za

## Abstract

Cell segmentation and cell tracking is important in fields involving microscopy images. The ability to detect cell culture characteristics is valuable for those studying cellular or subcellular features, as well as those doing research around disease treatment, drug development, and cell changes and/or interactions over time. Cell segmentation is a challenging task, as each dataset can have distinct features and complications. Recent works have demonstrated that deep learning techniques for cell segmentation and cell tracking show promising results, and may improve upon traditional cell tracking techniques.
This research investigated and compared different cell segmentation techniques, for the use in 2D time-lapse video sequences. After comparing traditional segmentation techniques against a Neural Network segmentation technique, this research concludes that using Neural Network techniques are the best approach for segmenting multiple datasets with distinct features. By cutting an image into smaller pieces, with square dimensions, a flexible Neural Network architecture can be used (U-Net) to segment cells well. This research found that the Neural Network segmentation performs well on all datasets, including raw/unproccessed images. The research concludes, explaining how a patch size of 256 shows the greatest promise at a generic approach to cell segmentation, through U-Net architecture.

***CCS Concepts:*** • **Computer systems organization** → **Neural networks**; • **Computing methodologies** → **Image processing**; **Image segmentation**; *Tracking*; • **Applied computing** → **Imaging**.

***Keywords:*** Neural Networks, Segmentation, Image Processing, Cell Tracking Challenge, U-Net, Patchify

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

*Conference acronym 'XX, June 03–05, 2018, Woodstock, NY*
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

**ACM Reference Format:**
Alexander Goudemond and Serestina Viriri. 2018. Segmentation Techniques for Tracking Moving Cells in Time-Lapse Video Sequences. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 22 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 Introduction and background

Cell segmentation and cell tracking is valuable for several reasons. Examples include: disease research, tracking tissue growth, tracking cell changes over time (mitosis, apoptosis, etc.), cell velocity changes, lineage tracing, distinguishing between cellular and sub-cellular features as well as development towards drugs and vaccines. Software can greatly assist this work being conducted, and help automate repetitive tasks, which are normally done manually.

There are 2 types of microscopy images that can be used in cell segmentation and cell tracking programs: *fluorescence microscopy* and *transmission microscopy*.

**Fluorescent microscopy** encapsulates techniques around **fluorescent light**. A staining compound or gene (referred to as a 'tag' or a 'reporter') may be introduced into the cell. George [9] explains that high intensity light is then shone on the cells, and these tags reflect a lower energy light, with longer wavelengths. This property results in bright cellular features, against a dark background. Some cells/organisms may be naturally fluorescent, and a tag is not needed. However, many cells may require a tag.

**Transmission microscopy** encapsulates techniques around Transmission Electron Microscope **(TEM) images**. CCBER [1] demonstrates how a beam of electrons is shone onto specimens, resulting in a highly magnified and clear image. This microscope can create images that are Phase Contrast (PhC) (revealing refractive contrast over a specimen) or Differential Interference Contours (DIC) (introduces contrast to otherwise faint images)

Nuclear fluorescence microscopy is a reliable way to stain the nucleus of a cell. This is desirable as nuclei, across species, share a common shape and texture. This provides a good starting point for algorithm design, as most datasets will contain easily visible nuclei. Thus, many algorithms explore nuclei detection and use these 'seed points' to identify the

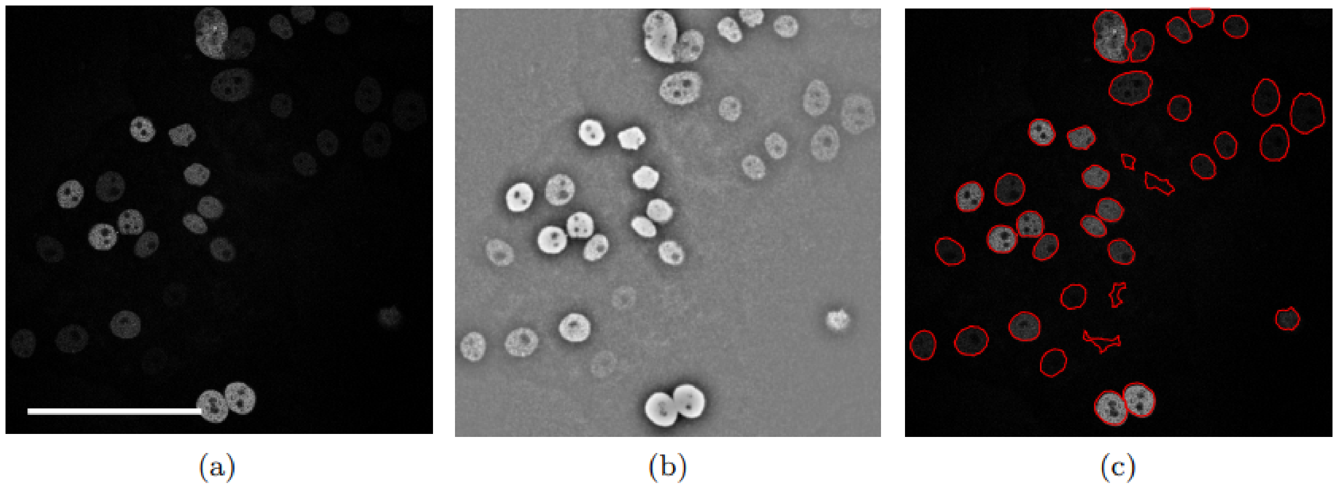


**Figure 1.** Figure showing the application of cell segmentation, sampled from Magnussen [7]

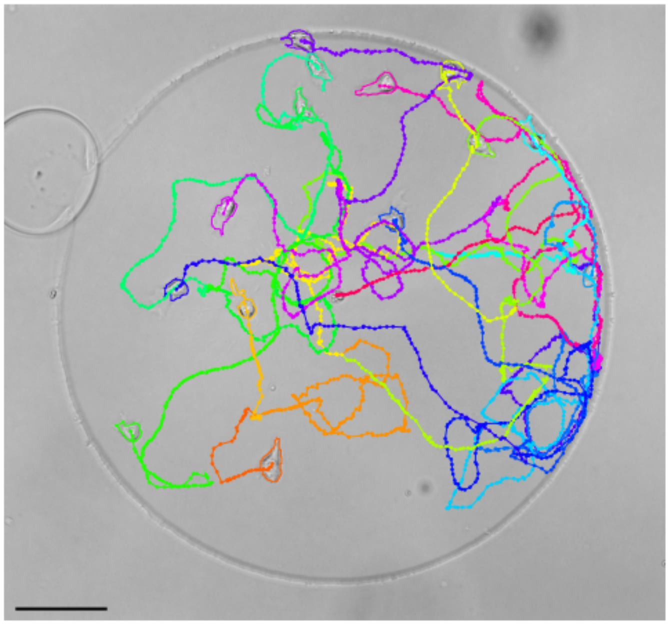


**Figure 2.** Figure showing an example of cell tracking, sampled from Magnussen [7]

whole cell boundary. Some algorithms are only interested in nucleus detection, but whole cell detection is first achieved by nucleus segmentation.

Particle tracking is another application of cell segmentation and tracking techniques, however particles are easier to track. Most cells display *Brownian Motion*, making their movements in a time-lapse sequence erratic and unpredictable. Particles, on the other hand, rarely move and rarely interact with cells. In addition to this, particles often contain smooth boundaries, whereas many cells contain irregular boundaries, that grow and shift over time. Due to these differences, particle tracking is seen as a simpler task than cell tracking, and will not be the focus of this project.

2D cell tracking is much broader, and most of the research conducted already is on 2D datasets, though 3D cell tracking is also possible. For example, Magnusson [7] and Wang et al. [14] demonstrated the ability to track cells in 3D datasets using dynamic iterative programming, and a Convolutional Neural Network (CNN), respectively.

There is also a difference between *global segmentation*, *semantic segmentation* and *instance segmentation*.

**Semantic segmentation** incorporates the **classification** of an item. It is useful in datasets that contain many clearly distinct items, such as the difference between a car, a person and a cloud.

**Instance segmentation** is interested in **identifying individuals** in a dataset. Each boundary of an instance segmented image contains (at least) 1 individual inside of it, and the kind of individual it is, is not important to the model.

**Global segmentation** seeks to separate any objects with contrasting texture - irrespective of foreground or background.

Thus, the cell segmentation and cell tracking algorithms explored in this project focus on instance segmentation techniques. It is worth mentioning, however, that semantic segmentation models may assist with instance segmentation, as will be shown later in this paper with the chosen Neural Network architecture: U-Net.

The U-Net model is outlined in Ronneberger et al.[10], and explains how the U-Net design improves upon the Sliding-Window Convolutional Network (a previously successful method). Specifically, there is a contracting left path, and an expansive right path. The left path uses a normal architecture of a CNN, whereas the right path up-samples the results from the contracting steps. Overall, 23 Convolutional layers are used.

This paper implements a version of the U-net architecture, originally implemented by Bhattiprolu [5]. Although other architectures were considered, this implementation allowed the most flexibility, and a detailed figure showing the design and structure of this architecture can be found in Figure 8. As can be seen, this implementation contains 37 layers in total, and 23 convolutional layers (shown in blue and red).

Figure 1 is an example of the application of cell instance segmentation, by overlaying boundaries on a cell image. Notice how, in this Figure, (a) is the original image, (b) is the processed image and (c) contains the boundaries from segmentation, overlaid on (a).

Wang et al. [15] mentions how cell tracking either takes the form of tracking by model evolution, or the form of tracking by detection. In the former, segmentation and tracking are addressed simultaneously, each frame. This is done by using some kind of hidden data, stored in a feature space. In the latter, the task is often split into cell detection and cell association. The tracking can then be achieved by comparing the 2 objects, and performing some kind of linking algorithm.

Figure 2 is an example of cell tracking, and how the path of a specific cell can be traced over time. Notice in this Figure how every cell is allocated a unique colour - representing an individual cell moving. This application is difficult, as it needs to consider many possibilities for the cells present in the image. A more thorough description of these challenges can be found in Section 2.

It is expected that the dataset provided to the program will contain cells that could be identified by a biologist. This is certainly the case with microscopy time-lapse datasets, as the specimens are grown by the biologists, and thus the kind of cell must be known. However, artificial datasets are being created, and in these situations the kind of cell is expected to be provided along with the dataset.

The Cell Tracking Challenge (CTC) [2], is an initiative to provide datasets and algorithm metrics to the public. There is a need for open source software, and freely available datasets - which the CTCs have the potential to provide. datasets from this site will be used in this project.

At the time of writing, CTC contains 10 publicly available 2D datasets. These datasets contain different kinds of cells, and further datasets may be used in future CTC competitions. The 10 datasets are shown in Table 3.
8 of these datasets contain silver reference annotations (referred to as *silver truth* or ST datasets), all 10 contain gold reference annotations (referred to as *gold truth* or GT datasets) and 1 of the datasets is simulated.

This research extends existing work, and attempts to identify successful techniques for broad level cell segmentation. The motivation is that these segmented images could then be used for tracking applications. The creation of software will abstract the implementation of the model, and enable researchers to focus on the results, without needing to understand or tweak the model used.

### 1.1 Research Problem

Datasets of cells are not equivalent. Each dataset contains different kind of cells, each with underlying properties and behaviours. Some datasets contain stem cells, which tend to cluster as they grow. Some contain cells that move frequently, and interact with their surroundings - resulting in an irregular and changing cell boundary. Some contain combinations of cells, or a combination of cells and particles. There are also datasets that contain capturing issues: light saturation, camera shake, video-sequence time-delays, magnification adjustments and cell movement out of frame.

Existing research often uses high-powered, and often unattainable, computers - so there is also a need to conduct this comparison on a more accessible computer, and more accessible resources.

### 1.2 Research Focus

The research implements a U-Net model, compares it against traditional segmentation techniques and shows the pros and cons, trying to show the existing research gaps.

### 1.3 Research Contributions

As can be seen in Table 1, the largest image present is (1036, 1070, 3) and the smallest image present is (512, 512, 3). However, 10 other distinct image dimensions are also present. Attempts to resize these images distorted the visible pixels (negatively impacting this research) , and attempting to reduce all images to the smallest file size, by cropping, would remove approximately half of the data contained in the images with larger file dimensions.

As a result of this, this paper slices the images into smaller pieces, using a Python package called **patchify**, built by Wu [16]. Patchify allows an image to be reduced into smaller square pieces, by iteratively cropping the image in a constant patch size, through a moving square window. In other words, and image of dimension (512, 512, 3) could be broken up into 4 images of (256, 256, 3), using a patch size of 256. A more thorough explanation of this process is detailed in Section 3.6.

Through the use of patchify (and its inverse process, *unpatchify*) , this paper proposes a method to reduce any image into desirable patches, which can then be easily fed into a Neural Network. This approach is beneficial to future research, as it is flexible for any image size, provided the original image is cropped into a dimension that is a multiple of the desirable patch size.

Being able to break a collection of images into smaller patches, also has the potential to increase training data size, for datasets with a small number of images present. In addition to this, images of smaller dimension make training easier, and consume less hardware resources - making it more approachable for researchers. There is also a benefit of reducing the available data contained in an image - allowing a Neural Network to focus on less potential classes in an image. This approach means that a significant amount of training can be accomplished in a few hundreds epochs, with under a thousand images, in environments like Google Colab. This compromise allows good results to come through, with a small training set , as CNNs usually require thousands of images to train.

The conclusion of this research reveals that a patch size of 256 is a good compromise between window size, details of the image, and training needs. In addition to this, some potentially reproducible mistakes are described in detail in Section 4.3, to try assist other researchers with avoiding the difficulties this paper experienced.

### 1.4 Paper Structure

This paper contains 9 sections. The next 3 sections form the remainder of the paper and is structured as follows: Section 2, the literature review, provides details on previous successful approaches. Section 3, the methods and techniques, provides insight into the framework followed in this study. Section 4, the results, provides details of the results that were obtained during the research. Finally, Section 5 elaborates on the insights of this work and concludes the paper. Section 6, 7, 8 and 9 describe the Software Used, Resources Used, Data Availability and Conflict of Interest, respectively. Section 10 contains the references.

## 2 Literature Review

There are several benefits to a program being able to conduct cell segmentation and cell tracking. Some broad level benefits include automatic analysis of datasets (increasing workflow and eliminating bias), the potential for cell feature tracking/detection in real time and even having a program that can interpret the results for the researcher.

Al-Kofahi et al. [3] mentions how cell culture characterization is important with regard to cancer research and drug research. Cancer research benefits in particular because certain changes in cell features (like abnormal cell boundary changes) may be indicative of cancerous growth, mentioned also in Wang, et al. [14]. Magnusson [7] additionally mentions how some researchers need a program that can conduct lineage tracing, velocity tracking and tissue development.

Scherr et al. [11], Magnusson [7] discuss how their software can also incorporate 3D dataset processing.

There are many challenges that are faced with this kind of research. Magnusson [7] explains how software is not always designed to be universal; some aim to address a specific problem only. And some papers only focus on algorithm design, without ever implementing the software for use. This is compounded by the fact that many datasets and ground truths are not publicly available. Scherr et al. [11] and Magnusson [7] also mention a high computational need, with 64 GB of RAM needed to quickly process information. These resources are not always available to researchers, and algorithm comparisons using more accessible equipment is needed. Moen et al. [8] also mentions how most applications of deep learning require supervised learning, with specialized training sets. This illustrates the difficulty with using unseen, live cell video sequences.

Other challenges include cell clustering, joint segmentation, sample diversity and parameter tweaking, as outlined in Magnusson [7]. Fragmented cells, large Signal-To-Noise (SNR) images, annotation/label errors and unpredictable movement are mentioned in Scherr et al. [11]. Wang et al. [14] also mentions how some images require special preprocessing, like bright-field images, DIC images and PhC images.

Attempting to build collaborative software, Fazeli et al. [6], discusses their attempt to build on top of/integrate with existing open-source software. Their program StarDist interacts with existing software TrackMate, which is freely available to the public. They also mention how, although pre-trained models exist, they are likely to under perform on distinct datasets. To address this, they offer the user a way to annotate their datasets, through existing software Fiji. Although this is time-consuming - it produces better results.

There are also biological challenges mentioned. For example, Magnusson [7] explains how staining in fluorescent microscopy images may result in drug reactions at cellular level. This may alter the cell's behaviour, or impact it's interaction with the environment. The staining may also be lost as a result of cell division, so it may not be ideal for lineage tracing.

Scherr et al. [11] goes on to mention how apoptosis and movement out of frame of the video-sequence impacts the algorithms. These challenges are experienced at dataset creation, and need to be considered when the algorithms are designed. Additional pre-processing may be necessary, to handle things like: camera shake, magnification changes, time-sequence capture delays as well as dirty equipment - leading to smudges/ blurred aspects in the dataset.

In terms of segmentation techniques, Al-Kofahi et al. [3], Magnusson [7], Scherr et al. [11], Moen et al. [8] and Wang, et al. [14] mention how staining the nucleus allows for development of the initial segmentation steps. They explain how seed locations can be used as the approximate center point of each nucleus, and the whole cell boundary can be obtained as the next boundary. Different segmentation techniques are successful in this pursuit, though a popular one includes overlaying a mask onto the image, before applying a watershed segmentation technique.

Al-Kofahi et al. [3] mentions how their segmentation includes single channel whole cell segmentation through deep learning, combining techniques from thresholding, the watershed algorithm and a kind of 'blob-detector'. This model needs to be trained offline before use. Other methods considered include active contour models, level-set methods, morphology based methods and snake algorithms. Magnusson [7] discusses how their dynamic iterative algorithm has different segmentation techniques based on whether it is a fluorescent or transmission image. It also incorporates ridge detection for datasets that are clustered. Scherr et al. [11] has a technique involving distance maps and the TWANG algorithm. Wang, et al. [14] mentions region growing as a segmentation technique. Ulman et al. [13] mentions broad-level segmentation methodologies: thresholding, region growing, energy minimization, shape matching, edge detection and machine learning.

With regard to tracking techniques, Magnusson [7] employs a dynamic cell track-linking algorithm, which connects seeds between frames. This kind of detection algorithm is done using a nearest-neighbour approach, in terms of the smallest distance from the change in seed position.

Al-Kofahi et al. [3] and Scherr et al. [11] use a deep learning technique to track the movement of cells in a similar way.

Moen et al. [8] implements a deep learning method using linear programming and a viterbi algorithm. It is also able to solve the cell-out-of-frame problem using 'shadow objects', which track the information about a cell's disappearance in each frame, and then tries to relate the information across frames.

Sun et al. [12] provides an overview of the basic knowledge of deep learning and its applications. In particular, it mentions the popular deep learning techniques used for single-cell optical image studies.

In terms of Deep Learning models, they draw attention to how CNNs and the Generative Adversarial Networks (GANs) have produced good results in image processing tasks. Additionally, popular Deep Learning methodologies for optical images include Transfer Learning, Multimodal Learning, Multitask Learning and End-To-End Learning.

A valuable insight from Sun et al. mentions the popular Fully Convolutional Networks: U-Net, SegNet and DeepLab. These architectures have been modelled after, or improved upon, by many papers already.

The CTC competition, Ulman et al. [13], provides detailed examples of factors that influence this research.

Looking at the quality of cell images and videos, one such comparison is the experience between Signal-To-Noise (SNR) and Computed Radiography (CR). It is found that the best results occur when there is a high SNR and a high CR, with the worst results being a low CR and low SNR (The low SNR used was a seed density of 200, using Gaussian noise). It goes on to mention how, different signal textures or changing cell heterogeneity (different light intensities) can lead to over-segmentation. Also, small pixel distributions and/or irregular cells can impact cell boundary creation. Finally, cell overlap between consecutive frames, and coinciding mitotic events affects tracking.

Ulman et al. [13] shows the results of competitions held in 2013, 2014 and 2015, through the *IEEE International Symposium on Biomedical Imaging* (ISBI), comparing 21 algorithms. These contained a mixture of traditional and deep learning algorithms. Tables 2 and 3 list the methods used for segmentation and tracking in these competitions.

Since 2017, the competition has been available for online submissions. 2019 and 2020 held additional activities through ISBI, and the 2021 ISBI challenge has recently completed. The results from these years are yet to be published, at the time of writing.

Most state-of-the-art deep learning methods are based on, or inspired by, a network called U-Net. This Neural Network (NN) was successful in segmenting a broad range of biomedical images, and is a kind of benchmark to refer to and improve upon. This model is outlined in Ronneberger et al. [10], and explains how the U-Net design improves upon the Sliding-Window Convolutional Network (A previously successful method). Specifically, there is a contracting left path, and an expansive right path. The left path uses a normal architecture of a CNN, whereas the right path up-samples the results from the contracting steps. Overall, 23 Convolutional layers are used. The benefit of this work is to improve the resolution of the output - and segment the cells efficiently.

It is worth mentioning that Ronneberger et al. used a 6GB NVidia Titan GPU, and the training took 10 hours. However, the trained models are saved and can be applied to other tasks as well.

Looking closely at existing deep learning architecture, Moen et al. [8] treats this tracking problem as a linear assignment problem, and uses a supervised deep learning model to optimise a cost function for tracking cells. By using certain cell features, a Hungarian Algorithm can be implemented to select one value - minimizing the cost function and enabling learning.

For segmentation, every cell in a frame is allocated an ID and a JSON file, containing lineage information. These annotations are then used in the tracking step. Nuclear seed segmentation was used, and to validate data, crowdsourcing via a program called *Caliban* enabled users to correct errors with a keyboard and mouse.

To accomplish the tracking: 4 vectors of information (appearance, morphology, motion and neighbourhood) were created and given branches. The branches summarize the information provided to it, and then feed the information into the neural network (A hybrid recurrent convolutional deep learning model) To incorporate temporal information, a LSTM layer may merge any multi-frame information into 4 individual vectors.

The neural network has several layers, the final one applying a softmax for a classification: $P_{same}, P_{different}, P_{parent-child}$ - probabilities that can then be used for tracking.

Al-Kofahi et al. [3], on the other hand, uses a deep learning model to segment images. Tracking is not considered by the paper, but the segmentation is successful for cells that have a variety of stains.

The algorithm requires offline training. Thereafter, there are 4 steps to the algorithm: provide the unseen image, separate nuclei and cytoplasm, detect the nuclei, and detect the cell boundaries. The output of the model is 2 images: an image of nuclear seeds (which could be used for tracking) and an image of whole cell boundaries.

The nuclei detection uses a multi-scale blob detector, multi-level thresholding and a shape-based watershed segmentation. The cell segmentation uses pixel-level weighting, multi-level thresholding and a seed watershed algorithm.

5 convolution and pooling paths are used, followed by 2 dropout and de-convolution paths. The paper also provides a list of parameters used, which were proposed as desirable for their architecture.

Scherr et al. [11] uses deep learning for segmentation, and a graph-based matching strategy for tracking. The paper considers using Euclidean distance transform for cell boundaries and the inverse normalized distance for neighbour distances to conduct segmentation. It further mentions how combining cell distances with neighbour distances prevents cellular merging.

For segmentation, 1 encoder path connects to 2 parallel decoder paths, and backpropagation from both decoder branches is allowed. Each branch focusses on 1 task: cell distance or neighbour distance.

For tracking, a graph based cell linking algorithm is used. A rectangular region of interest is used as a search space in each frame, and a phase correlation calculation performed for most likely movement. To connect cells, an adapted version of the coupled minimum-cost flow algorithm is used.

Post-processing for both segmentation and tracking is employed, using a watershed method and mask placement method, respectively.

Wang, et al. [14] proposes a deep learning based pipeline, focusing on segmenting densely packed 3D cells. Their system uses a 2 stage pipeline and 1 hyperparameter for a light-weight deep CNN model. The model outputs voxel masks. Additional tools are also discussed: a specialized loss function to detect clustered cells, and details on their touching area-based clustering algorithm to partition foreground and background. For post-processing, they propose an algorithm Touching Area-based Spatial Clustering of Applications with Noise (TASCAN) - which is similar to the Density-based Spatial Clustering of Applications with Noise (DBSCAN) algorithm.

The hyperparameter is the minimum touching area between 2 voxels in the foreground.

Ben-Haim and Riklin-Raviv [4] use 2 deep learning tools to tackle tracking: a Multi-Layer Perceptron (MLP) to generate instances and features of each cell, and a Graph Neural Network (GNN) to connect and track cells. Their MLP uses cell segmentation maps or marker annotations to crop each frame into several sub-images - each containing 1 cell. A hard mining strategy and multi-similarity loss function is the used to train a ResNet network.

The paper provides a detailed breakdown of their Deep Learning architecture - and mentions a new GNN block that allows the system to update both the node and edge feature vectors, at the same time. This dual-update system results in a message passing process between nodes. By designing Message Passing Neural Network (MPNN) blocks, the cell instances and associated feature vectors are encoded as nodes, whereas the movement of cells is encoded as edges. This approach also enables global temporal knowledge, instead of just neighbouring frames.

The paper approaches to solve an edge classification problem in their GNN, producing active edges. These active edges are then used to construct the tracks and lineage trees, and can be used for Mitosis detection.

Wang, et al. [15] approaches tracking using a Deep Reinforcement Learning (DRL) method to link cells between frames. A cost matrix is created using the cell target features, which is then provided to the network as input. The Reinforcement Learning Neural Network (RLNN) predicts the distribution over a solution. In addition to this, a Residual Convolutional Neural Network (RCNN) is mentioned, to improve the learning rate.

The paper uses a U-Net segmentation method to detect cells in a frame, then a single hypothesis tracking method using a Kalman filter and frame-by-frame association produces the cell movement. The specific architecture is a Residual CNN (ResCNN), made up of 5 blocks. The output of these blocks is a noise value, which is used to calculate a probability distribution.

Fazeli et al. [6] takes a different approach, and builds a plugin for existing software TrackMate. A pipeline called StarDist is used to ... StarDist is compatible with both fluorescent and widefield images, and has dependencies ZeroCostDL4Mic and Fiji - open-source software. It is designed to be used by biologists, whom have no programming experience, and produce a file containing all the nuclei's geometric center coordinates. These coordinates are then fed into TrackMate.

## 3 Methods and Techniques

### 3.1 Dataset Acquisition

10 2D time lapse datasets were used in this research, and were accessed from the Cell Tracking Challenge (CTC) website [2] . Each of these datasets contain 2 links: a training set and a challenge set. This project explores the training set images, and uses a partition of the set to test the model.

### 3.2 Dataset Understanding

Table 3 shows a detailed summary of these datasets, as well as some challenges present in each dataset.

Each dataset has 2 folders (training and testing), resulting in 20 folders initially. However, most of the training datasets contain 6 internal directories as well, labelled: 01, 01_GT, 01_ST, 02, 02_ST, 02_GT. A few datasets do not contain the ST folders, meaning that the cumulative amount of directories present in the training folders are 96. The test datasets only contain the 01 and 02 folders, so there are 20 folders present in the test folders. The training dataset has a combined initial size of 4GB, and the test dataset has a combined initial size of 3.56GB.

Each folder has a specific image dimension, which is shown in Table 1. In addition to these dimensions, the folders labelled 01 and 02 have images where the pixel values are in the integer range [0,..., 255]. However, the GT and ST folders contain masks in the float range [0, ..., 1].

In addition to this, the ST and GT folders contain different quantities of training images, per folder. A detailed description of the quantities of images can be found in Table 2. It

**Table 1.** Table showing the image dimensions present in the datasets

| Dataset Name | Folder Dimensions |
|---|---|
| BF-C2DL-HSC | (1010,1010,3) |
| BF-C2DL-MuSC | (1036,1070,3) |
| DIC-C2DH-HeLa | (512,512,3) |
| Fluo-C2DL-Huh7 | (1024,1024,3) |
| Fluo-C2DL-MSC/01 | (832,992,3) |
| Fluo-C2DL-MSC/02 | (782,1200,3) |
| Fluo-N2DH-SIM+/01 | (690,628,3) |
| Fluo-N2DH-SIM+/02 | (773,739,3) |
| Fluo-N2DH-GOWT1 | (1024,1024,3) |
| Fluo-N2DL-HeLa | (700,1100,3) |
| PhC-C2DH-U373 | (520,696,3) |
| PhC-C2DL-PSC | (576,720,3) |

can also be seen at the end of this table the overall quantity of images in both the ST and GT datasets. Studying the table reveals that the GT training data contains inconsistent masks per folder. For example, BF-C2DL-HSC has 49 masks in folder 01, but only 8 in folder 02. The ST folders, in contrast, are consistent throughout. Another noticeable difference is the quantity of images present in the ST folders for the datasets BF-C2DL-HSC and BF-C2DL-MuSC. This paper believes this is because the images contain a recurring stationary object (an object that resembles a petri-dish) and more data has been provided to try to handle that. Finally, the ST datasets do not contain any training data on Fluo-C2DL-Huh7 or Fluo-N2DH-SIM+.

### 3.3 Dataset Pre-processing

In order to try and assist with the segmentation methods, this paper considered the following pre-processing steps: Histogram Equalization, Smoothing, Thresholding and Morphological Operations.
Figure 4 contains a collection of images, which display some of these processing steps. Namely: Morphological Operations, Adaptive Thresholding, Histogram Equalization and Thresholding are described by the figures as images {a1, b1, ..., i1, j1}, {a2, b2, ..., i2, j2}, {a3, b3, ..., i3, j3} and {a4, b4, ..., i4, j4} resepctively. Notice how each positional image contains the same alphabetical tag - this is designed to communicate how specific datasets vary per processing step.

From the experiments done on the dataset, the only processing step that benefited all 10 datasets was a morphological operation, visible in Figure 4 as {a1, b1, ..., i1, j1}. This paper found that 2 morphological operations were particularly useful: increasing the brightness by 15 pixels, then dilation followed by opening as well as increasing the brightness by 15 pixels, then opening following by erosion. Of those 2, the former resulted in the images being brighter and

**Table 2.** Table showing the quantity of training images present the datasets

| Dataset Name | GT Mask Quantity | ST Mask Quantity |
|---|---|---|
| BF-C2DL-HSC/01 | 49 | 1764 |
| BF-C2DL-HSC/02 | 8 | 1764 |
| BF-C2DL-MuSC/01 | 50 | 1376 |
| BF-C2DL-MuSC/02 | 50 | 1376 |
| DIC-C2DH-HeLa/01 | 9 | 84 |
| DIC-C2DH-HeLa/02 | 9 | 84 |
| Fluo-C2DL-Huh7/01 | 8 | N/A |
| Fluo-C2DL-Huh7/02 | 5 | N/A |
| Fluo-C2DL-MSC/01 | 18 | 48 |
| Fluo-C2DL-MSC/02 | 33 | 48 |
| Fluo-N2DH-SIM+/01 | 30 | N/A |
| Fluo-N2DH-SIM+/02 | 20 | N/A |
| Fluo-N2DH-GOWT1/01 | 65 | 92 |
| Fluo-N2DH-GOWT1/02 | 150 | 92 |
| Fluo-N2DL-HeLa/01 | 28 | 92 |
| Fluo-N2DL-HeLa/02 | 8 | 92 |
| PhC-C2DH-U373/01 | 15 | 115 |
| PhC-C2DH-U373/02 | 19 | 115 |
| PhC-C2DL-PSC/01 | 2 | 300 |
| PhC-C2DL-PSC/02 | 2 | 300 |
| **Total** | **578** | **7742** |

were used for neural network training, referred to as the *processed images dataset* by this paper.

### 3.4 Dataset Augmentations

This project encountered 2 significant problems with the datasets used, which led to some augmentations being made.

The first augmentation was to manipulate the type of pixels for the masks, to improve mask visibility. The problem discovered was that the pixels were created in a float range of [0, ..., 1],, and needed to be mapped into an integer range of [0, ..., 255]. The features of the masks were not present until this augmentation was complete and this paper was able to address the problem by first reading the images in using Matplotlib, immediately saving the image under a temporary name, and then reading the temporary image back in using OpenCV.

Through this process the pixel values were successfully updated from a float value into an integer value, and only then the information contained within was visible by the default photo viewer software.

The other augmentation needed was for Neural Network training. The masks provided by the CTC [2] contained no consistent classes. Each individual cell inside a mask was assigned a unique grayscale value, leading to an inconsistent multiclass classification problem.

To address this, this paper read in the masks and forced all of the non-zero class values to take on a value of 1. It was only from this point on that the Neural Network models trained started to produce visible results.

The reader may realize that the 2 augmentations mentioned here appear to counteract one another. The reason this paper initially augmented the masks into integer pixels was to allow the features present to become visible to the user. It was only with the visible information that knowledge of the datasets could be learnt and experimented with.

However, the Neural Network model requires the pixels to be in a range of [0, ..., 1], which is why this paper ensures that the valid classes for training and predictions are binary values of 0 and 1. This paper also ensures that predictions are collapsed into those binary values of 0 or 1 using a threshold of 0.5.

### 3.5 Dataset Traditional Segmentation Techniques

This paper attempted to segment the images using the following methods: Watershed Method, Thresholding, Canny Edge Detection, Simple Contours, Complete Contours, and K-Means Clustering. This paper did find that OpenCV had a specific thresholding value (a value of 17 passed as one of the parameters) that resulted in very good segmentation of all 10 datasets - however this thresholding technique also included undesirable features, like the petri dishes and particles.

Unfortunately, attempts to use this thresholding value in Watershed did not improve the results. Figure 4, images {a5, b5, ..., i5, j5} describe the watershed method and images {a6, b6, ..., i6, j6} describe Canny Edge detection. None of the traditional segmentation techniques was useful for this research.

### 3.6 Dataset Neural Network Segmentation Techniques

This paper investigated 1 Neural Network architecture for segmentation - the U-Net architecture.
This model is outlined in Ronneberger et al. [10], and this paper's implementation of the U-Net architecture can be seen in Figure 8. The original author of the U-Net model used here is Bhattiprolu [5].

Figure 3 is used to describe the Neural Network Segmentation, visually.
In order to segment images via this technique - the images first need to be prepared for the architecture. This architecture expects the images to be square, often falling in one of the following dimensions: 64, 128, 256, 224 or 256. These sizes are desirable as it allows for the Down-Sampling and Up-Sampling to occur neatly, following the multiples of 2.

**Table 3.** Table showing the datasets available on CTC website [2]

| dataset Description | dataset Name | Cell Classification | Dataset Challenges |
|---|---|---|---|
| Mouse hematopoietic stem cells in hydrogel microwells | BF-C2DL-HSC | Mouse Stem Cells | Recurring 'petri dish' is visible in all images, which must be ignored |
| Mouse muscle stem cells in hydrogel microwells | BF-C2DL-MuSC | Mouse Stem Cells | Recurring 'petri dish' is visible in all images, which must be ignored |
| HeLa cells on a flat glass | DIC-C2DH-HeLa | HeLa Cells | DIC microscopy; tightly packed; complex intensity patterns; boundaries difficult to find |
| Human hepatocarcinoma-derived cells expressing the fusion protein YFP-TIA-1 | Fluo-C2DL-Huh7 | Derived Liver Cells | Combination of Nucleur Fluorescent Microscopy lit cells, and non lil cells |
| Rat mesenchymal stem cells on a flat polyacrylamide substrate | Fluo-C2DL-MSC | Rat Derived Bone-Marrow Stem Cells | Fluorescence microscopy; irregular shape and size |
| GFP-GOWT1 mouse stem cells | Fluo-N2DH-GOWT1 | Mouse Embryonic Stem Cells | Fluorescence microscopy; mixture bright and dim objects |
| HeLa cells stably expressing H2b-GFP | Fluo-N2DL-HeLa | HeLa Cells | Nucleur fluorescence microscopy; size and shape variability |
| Glioblastoma-astrocytoma U373 cells on a polyacrylamide substrate | PhC-C2DH-U373 | Brain Tumour Cells | Inconsistent and abnormal cell shapes; many particulates present |
| Pancreatic stem cells on a polystyrene substrate | PhC-C2DL-PSC | Pancreatic Stem Cells | PhC Microscopy; sparse cells; complex intensity patterns |
| Simulated nuclei of HL60 cells stained with Hoescht | Fluo-N2DH-SIM+ | (Simulated) Derived Blood Cells | Nuclear fluorescence microscopy; many cells; bright and dim regions |

There are 2 ways to prepare the datasets for this model - *resizing* or *cropping*. This paper realized, however, that resizing the images was negatively impacting the results. Instead, the paper found that '**patchifying**' the images was more successful.

**Patchifying** an image will be defined in this paper as using the Python API *patchify*, created by Wu [16], to neatly extract smaller pieces (or patches) of an image. This system works by traversing the image left to right, top to bottom, and 'cutting' patched pieces out of the image. The final result resembles 'cake slices' being formed, from the original image.

For example, an image with dimensions
(832, 992, 3) and a patch size of 128 would result in the following:

**1)** The image is reduced in size to (768, 896, 3) because $\lfloor \frac{832}{128} \rfloor = 6$ and $\lfloor \frac{992}{128} \rfloor = 7$.
Thus $128 * 6 = 768$ and $128 * 7 = 896$.
**2)** The reduced image can be broken into $7 * 8 = 56$ individual images
**3)** Each of those individual images will have a dimension of (128, 128, 3)

To train the model, both the images and masks are **patchified**, and fed into the model, being used for training. Once the model is ready, the predictions and score of the models may be calculated using the Mean IoU metric between the patched images and patched masks. A more detailed explanation of this metric appears in Section 4.1. This metric can either be calculated at patch level, immediately after training, or can be calculated from the rebuilt, *unpatchified* image. The results in Table 4 reveal that either option is indicative of the model's performance.

To describe the methodology in detail , the Key present in Figure 3 will be referenced below.

Figure 3 contains 3 sections: a green section, a blue section, and a purple section. The green section represents the steps needed to interact with the masks for both training and predictind for all datasets. Similarly, the blue section describes the process for the images to be used to train and predict for all the datasets. The purple section represents the metric used - the Mean IoU score.

Starting with the green section, the masks need to be cropped and updated to from a mutli-class system to a binary class system (shown with Figures 3b and 3c respectively. For training, the binary cropped mask can then be cut into patches, represented by Figure 3d. However, depending on the Mean IoU score calculated, either the patches or the cropped binary mask is sufficient.

Moving to the blue section, the images used also need to be cropped, which can be seen in Figure 3f. However, for both training and predicting, the image must be cut into patches. the models have been designed to interact with the patchified images, and so the only way to predict is by patchifying an image, as shown by Figure 3g.

Depending on the type of prediction necessary, the patchified predictions shown by Figure 3h can either be compared immediately with the patchified masks, Figure 3d, (present from training) or they can be used to rebuild the mask, shown by Figure 3i.

If rebuilt, Figure 3i can then be compared with Figure 3c, as another option to compute the Mean IoU Score. The dotted lines present in the figure are intended to show this comparison, either after model training, with a dedicated test set of patchified images and masks, or with the original cropped binary mask and the rebuilt predicted mask.
Either way, the Mean IoU score is indicative of the model's success.

This concludes the methodology used in this project, for Neural Network Segmentation. Section 4.4 goes into more details about the success of this methodology.

### 3.7 Experimental Setup

This paper approached the task in several steps, using many Jupyter Notebooks. Jupyter Notebooks were chosen as a large amount of data needed to be processed, and the notebooks allow for execution to occur in more than 1 order - which is useful if a programming environment has memory restrictions, and one can only afford to run a specific number of cells each runtime.

The notebooks were built as the author progressed with the project, and contain the results from experimentation on the datasets. The experiments were organized into stages:

1. Setup and describe the dependencies used through the project
2. Explore the dataset and its properties in more detail
3. Conduct pre-processing on the dataset
4. Conduct traditional segmentation techniques on the dataset
5. Isolate the training data for Neural Network Segmentation
6. Conduct Neural Network Segmentation on the training data
7. Reflect on the results and challenges experienced

20 notebooks were used in total, and are described in more detail in the **Software** section, as well as Table 5.

In order to conduct the Neural Network Segmentation, this paper needed to randomly distribute the images in order to train a model that had an even exposure to all types of cells. This was achieved in the following way: the desirable datasets were loaded (either ST or GT) and shuffled, using SKlearn. The random state used to shuffle the data was a state of 42, and a test size of 33%. From these shuffled image paths, only the training data is considered (the test data that is generated by SKlearn was discarded). Depending on the dataset, the quantity of images at this point can either be less than 400 images (GT dataset, after train test split) or over 5000 (ST dataset, after train test split). The next step is to patchify the images. The goal was to have close to 3500 patchified images, and this paper found that having 387 GT images, and 300 ST images produced close to 3500 patchified images to work with. The patchified images (and corresponding, patchified masks) are then shuffled again, using the same random state and test size. Finally, the first 800 training images are selected as the training data, and the first 200 test images are selected as the test data. The reason for the small training size of 800 images is due to the resource restrictions outlined in greater detail in Section 4.3.

It is worth noting that the 512 model was not able to be trained with this quantity of images. In fact, around 150 training images and 50 test images was all this paper was able to achieve with the Google Colab Free Tier environment.

## 4 Results and Discussion

### 4.1 Metric Explanation

This paper identified that the conventional metric Accuracy, used to determine model success, was uninformative in the training process. Instead, the more reliable metrics were the loss function, as well as post-training calculation of a **Mean Intersection of Union** (Mean IoU) score, also referred to as a Jaccard Index: $J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A|+|B|-|A \cap B|}$. The larger the value of this ratio, the better the result of the model.

However, one drawback of using the Mean IoU score is *negative scoring*. This paper discovered that some of the models trained perform particularly well, as compared against the true masks. In fact, there were some models that even captured information present in the image that was not present in the mask. In other words, the prediction was better than

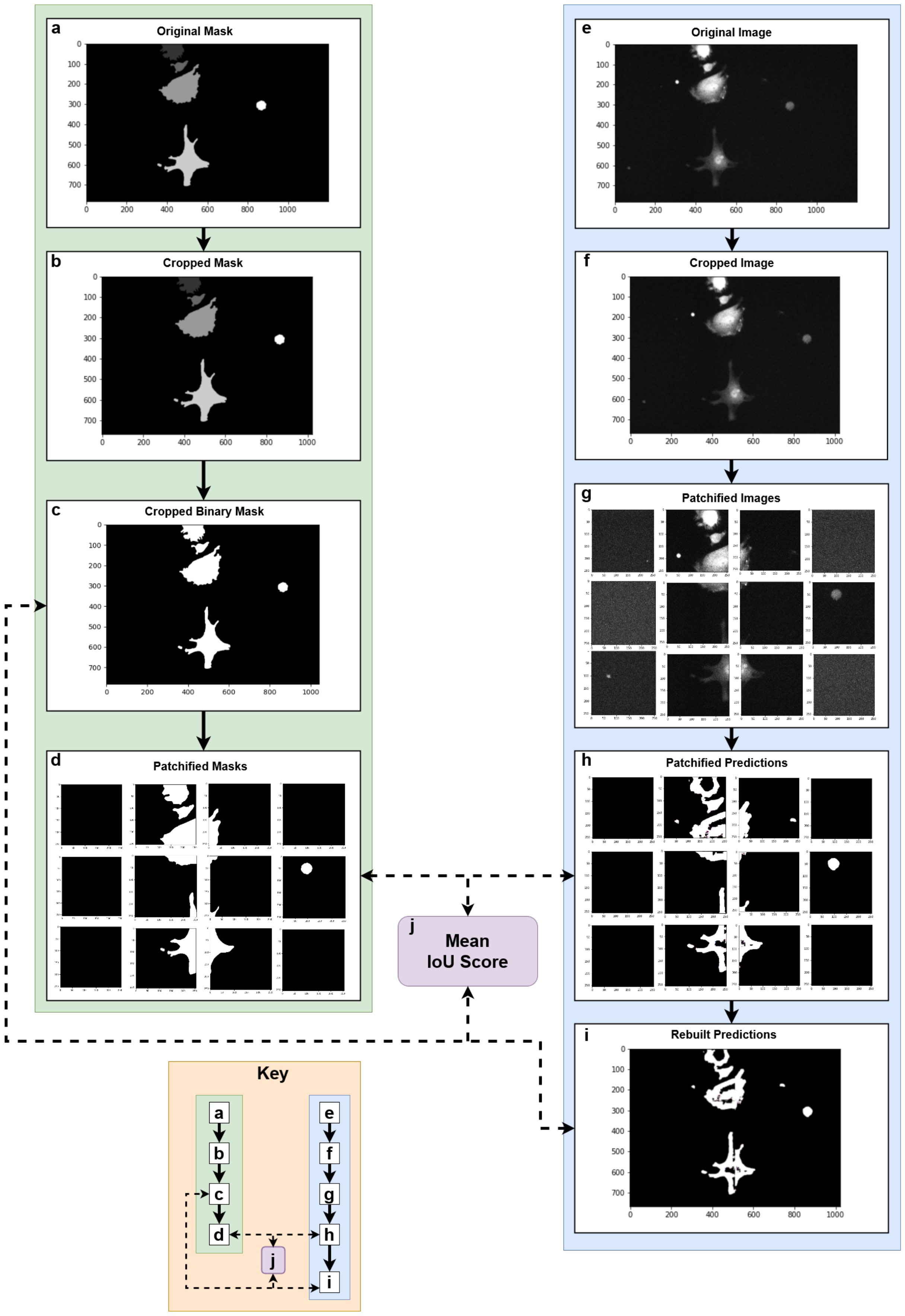


**Figure 3.** Flow Diagram showing the project methodology

the true mask - and yet the prediction received a lower score, due to this metric.
This disparity means that a high performing model may receive a lower score than expected. The attempt this paper has made to circumvent this include plotting the original image, true mask and predicted masks, alongside one another and then comparing the data. By looking at the figures generated, the reader may identify whether a model is performing well or not, and then cross examine this with the Mean IoU scores.

## 4.2 Literature Comparison

Magnusson [7] mentions how software is not always designed to be universal - rather it tends to focus on solving a narrow, and specific problem. This research may assist with the production of generic software, for the use of cell segmentation through the U-Net Neural Network.

The dynamic iterative algorithm, described by Magnusson [7] , uses different segmentation techniques depending on whether the image is a fluorescent or transmission image, and may also assist with detection of clustered cells. Although the implementation is unknown, Magnusson's implementation may include the researcher specifying, at the beginning of the segmentation process, the properties of the dataset being used. This paper's approach, to investigate a global technique to segment any cellular images, would help abstract the process from the user. The user would not need to specify any properties, and would only need to ensure that training has been conducted previously. In the event that Magnusson's algorithm is implemented automatically - this paper can still benefit the user, as only 1 technique needs to be applied to all images. In addition to this, a researcher could upload images containing a combination of cells, from multiple datasets - a limitation that may be present in software that attempts to detect cell properties before execution.

Wang et al. [14] proposes a method to segment a specific kind of cellular images: densely packed 3D cells. Although this technique produces voxel masks, it is limited by the kind of data that can be used. This paper did not explore 3D cell images, but it proposes that the U-Net architecture could be extended into a 3D form, as the tightly packed cells in some of the datasets used by this paper appear to be segmented well.

Moen et al. [8] conducts segmentation through Nucleur seed segmentation, as well as through the program *Caliban*, to try address errors using the public. This paper may yield fruitful results, by generating masks of each image, which could be overlaid on the original. These overlaid masks have the potential to assist with the Nucleur seed segmentation, as well as other segmentation techniques, like watershed.

In addition to this, Al-Kofahi et al. [3] implements segmentation using a Neural Network with 4 steps: provide the image, seperate nuclei and cytoplasm, detect the nuclei and finally detect cell boundaries. This paper's mask predictions could be applied in such a way to mimic the technique outlined by Al-Kofahi et al.

Scherr et al. [11] conducts segmentation using Euclidean distance transform for cell boundaries and the inverse normalized distance for neighbour distances. This paper could be improved by attempting to implement a distance-based method to detect cell boundaries in the masks.

## 4.3 Challenges Experienced

Many challenges were experienced in conducting this research. Some challenges were outside the control of the author, such as poor internet connection and load shedding. However other challenges presented themselves over the course of the research. The challenges experienced can be clustered into sections: **Dataset Preparations**, **Dataset Properties** and **Resource Restrictions**.

**Dataset Preparations**

The raw training datasets are saved as TIFF images. These images contain integer pixels in the range [0, ..., 255], whereas the masks contain float pixels in the range [0, ..., 1]. This means that the masks are not visible using existing photo software, as each mask appears completely black. This paper had to investigate which image processing packages could extract these values, and map them into an integer range that corresponded with the main images. OpenCV, Pillow and Matplotlib where used - and it was discovered that Matplotlib was the only package that would show the contents of the masks.

The paper then discovered that reading in the images via matplotlib and then immediately saving them using a grayscale colourmap resulted in the pixels being mapped onto an integer domain appropriately. It was only then that the masks were visible using inbuilt photo viewing software (such as the inbuilt Windows App)

However, this paper learnt that although integer pixels for the masks allowed the reader to view the data - the Neural Network models investigated actually require the images and the masks to be in float pixel form. Attempts to have the model use the integer pixels almost always led to the loss function receiving a NaN (*Not a Number*) value in python. Thankfully, this can be addressed through SKLearn or Tensorflow, as these packages have functions that can map integer pixels into float pixels. One example used here was the *MinMaxScaler* object from SKLearn.

In addition to the inconsistent pixels - the masks provided contain a unique colour for each cell identified in the image.

As an example, this means that once a raw mask has been mapped onto integer pixels values, then masks containing 15 cells have 16 colours - 1 black, 1 white and 14 shades of grey. Initial attempts to train Neural Networks on these images failed, as it was discovered that the model was attempting to predict anywhere from 1 to 256 classes, per image. Results improved significantly once all non-zero values in the mask were set to 1 (producing a binary mask system).

Unfortunately, this reduction from a multi-class classification to a binary-class classification meant that cells that were packed close together ended up losing their boundaries. This paper intended to explore techniques to replace these boundaries - but ran out of time.

As shown in Table 1, the training data provided by the CTC [2] does not contain consistent image dimensions. Although this does not impact the traditional image processing segmentation - it does impact the Neural Network segmentation. The architecture used in this research is a CNN, which requires the images to be a consistent square dimension, often falling in one of the following dimensions: 64, 128, 224, 256 or 512. It is worth noting that the input dimension can be any even number, however powers of 2 are popular with U-Net models due to the Down-sampling and Up-sampling process.

## Dataset Properties

Some of the datasets present in this research contain additional problems, that impacted the processing steps.

The BF-C2DL-HSC and BF-C2DL-MuSC datasets have a recurring circular object, which this paper refers to as a 'petri dish'. This object is omitted in the masks, however both traditional segmentation techniques and Neural Network segmentation techniques sometimes preserve these petri-dishes. This paper was not able to investigate techniques to remove this petri dish from traditional segmentation techniques, however the well performing Neural Network models learn to ignore it.

BF-C2DL-MuSC also contains recurring contours inside the frames, which this paper refers to as 'snail trails'. These snail trails emerge in the traditional segmentation techniques, but appear to be ignored in the raw and processed datasets.

PhC-C2DH-U373 contains recurring static objects, which this paper refers to as 'water shadows'. These objects resemble poorly lit cells with refracted light on them and this paper's attempts to segment the images often include these objects. A post processing step to detect stationary pixels over the entire time-frame may assist with this, however the low quantity of images present in this particular dataset may make this approach infeasible.

DIC-C2DH-HeLa contains large, tightly packed cells that are almost transparent. Due to this, traditional segmentation techniques fail to find the boundary of the cells, often segmenting the cellular components inside the cell only. This paper was unable to find a traditional segmentation technique to segment such cells.

Fluo-N2DH-GOWT1 and Fluo-N2DL-HeLa contain inconsistent masks - whereby some masks have a few cells present, and others have many. This is problematic because the cells in this dataset display Brownian Motion, and so are rather consistent between frames. The inconsistent masks also lead to issues with patchifying the dataset, as sections of these images will correspond to a black mask. Although this is not immediately a problem, the training and prediction steps of this research may yield a 'negative' score from this scenario, meaning that the model is scored poorly, when in reality it may be segmenting more cells then those present in the mask. For this reason, the author also had to investigate how to visually decide if a model is performing well or not.

This visual check is done in 2 ways. The first approach is whereby a single image is cut into patches, used to generate predictions, and rebuilt - allowing the reader to compare the orginal mask against the predicted mask. Another approach is to do this for all the images in the test set - and then generate videos - allowing the reader to see the movement of cells over time. Unfortunately, this paper was unable to generate videos of the predicted datasets, so a plot showing cells side-by-side is used instead.

## Resource Restrictions

This paper was unable to conduct the Neural Network segmentation offline. A loan laptop from UKZN was used (Section 7 contains the hardware descriptions) and unfortunately was not able to be used for Neural Network segmentation, possibly as a result of: not enough VRAM on the dedicated GPU, incorrectly setting up CUDA, attempting to train with too many images, or images with too large a set of dimensions.
A workaround was to use the free tier of *Google Colab*, which allowed the Neural Network segmentation to be conducted online. However, the resources made available in the free tier (~12GB RAM, ~80GB storage) meant that the paper frequently had the runtime disconnect due to using all available RAM. This meant that work had to be designed in sections, to allow multiple runtime sessions to occur in 1 notebook, to avoid crashing. In addition to this, the limited RAM made training difficult, with only 800 images being allowed for patch sizes of 128 and 256, but only 150 images for patch size of 512.
If using Google Colab, the user may request a GPU, as a replacement to the default CPU runtime. However, resource restrictions are put in place, such that using the GPU too

frequently, or for too long, automatically disconnects and disables use of the GPU's for a short time. Hence, training a model needs to be done cleverly, and only once - to allow the most amount of work to be accomplished.
Lastly, if reading in images into Google Colab, hosted on Google Drive, it is import to ensure the image paths are sorted. This paper discovered that sometimes images are stored in a (seemingly) random order, possibly as a result of a Hash Map. It is important to sort the image paths before reading in the images, to ensure compatibility between multiple folders.

## 4.4 Results

Unfortunately, no traditional segmentation techniques proved successful in segmenting all 10 datasets. Some traditional segmentation techniques proved helpful in specific datasets - but no 1 technique was identified for all 10. Techniques considered included: Watershed Method, Thresholding, Canny Edge Detection, Simple Contours, Complete Contours, and K-Means Clustering. This paper proposes that the reason the traditional segmentation techniques failed was because of the combined challenges present in the datasets. Table 3 contains a more detailed description of these challenges experienced.

However, the Neural Network Segmentation explored in this paper proved beneficial, and the technique to **patchify** an image may be beneficial for future research. Table 4 contains the Mean IoU score collected on all 24 models, exploring different patch sizes for training. The highlighted cells represent the best scored models in the relevant dataset, and it can clearly be seen that the best performing patch size is 256, across both the GT and ST sets. With reference to Table 2, it is worth noting that the GT dataset has less training data available, as compared with the ST dataset. In addition to this, the ST dataset has more stable collections of images, and omits Fluo-C2DL-Huh7 and Fluo-N2DH-SIM+. It is for these reasons that the the ST models perform better than the GT models.

The first 18 results from Table 4 were used to inform the generation of 6 further models, trained on the processed datasets. These results are contained at the bottom of Table 4.
Comparing the results in Table 4, the best performing models are the models labelled 256_3. The difference between models 1, 2 and 3 in every patch size has to do with the training parameters: Model 1 trains for 30 epochs, with a batch size of 4, Model 2 trains for 100 epochs, with a batch size of 4 and Model 3 trains for 100 epochs with a batch size of 8. In addition to this, due to the challenges described in Section 4.3, this paper was only able to train the Neural Networks with up to 800 patchified input images and up to 250 patchified training images.

However, the models trained with a patch size of 512 consumed so much RAM in Google Colab (and consequently crashed), that this paper was not able to train more than 125 input images and 50 test images at a time... CNN models normally require large amounts of training data in order to become useful, and it is for this reason that the 512 patch size models are not explored in greater detail.

On a similar note, the 128 patch size images are poorly predicted, and due the quantity of images produced by selecting a smaller patch size, the model continues to perform poorly, even after receiving 800 input images and 250 test images. This paper believes that one reason contributing to poor performance of 128 patch size images is due to the fact that the majority of training data will be empty space - containing masks where the only class is 0.

Thus, the 256 patch size images are a good compromise between image reduction, image complexity and file storage.

This paper also discovered that the ST models predict far better than the GT models, and there is also potential for ST models for predicting GT datasets, with fair results. More research will need to be conducted on this, but the structure of the data present in the ST dataset, as outlined by Table 2, make the ST dataset an ideal starting point for future research, as well as models to predict multicellular images.

Figures 5 and 6 show the combined predictions of the best performing ST model - 256_3. Figure 5 contains a grid format that shows the first 4 datasets contained in the ST set. Entries *a, b, c* and *e* correspond to the cells in Table 2. Notice how entry *d* is missing - this is intentional as the ST dataset does not contain images for that set. Similarly, in Figure 6, *f, h, i* and *j* corresponds to the remaining cells in Table 2, and entry *g* is missing for the same reason as entry *c.*
The cells in those tables are enumerated from 1 to 6, and correspond to: *raw dataset, dataset mask, raw dataset prediction, processed dataset, dataset mask, processed dataset prediction.*

Paying special attention to Figures 5 and 6, we can identify entries *c, e and i* are poorly predicted, whereas the other predictions are very good! This may warrant future research focusing on those datasets, or datasets with similar properties, instead of all 10. In particular, Figure 5 cell *c3* contains a subtle checkered pattern, which appears to partition the image into 4 pieces. This result is due to the the process of **patchifying** the images - and post processing processing techniques could attempt to resolve it.

Finally, Figure 7 contains an example of the best GT model predicting the entries missing from Figures 5 and 6: *d* and *g*.

It can be seen in this figure that the GT model performs well on the processed datasets, and has mixed results for entries *d3* and *g3*.

This paper randomly selected images from the 10 datasets for GT and the 8 datasets for ST, and generated a cumulative plot in the form of Figures 5, 6 and 7. From these repeated experiments, the results suggest that the ST model performs better on the raw dataset - as apposed to the processed dataset. This is reinforced by Table 4 as well. For the 2 entries not present in ST - the best raw GT model seems to predict entry *d3* better, whereas the processed GT model seems to predict entry *g6* better.

Considering only the ST dataset and the 8 datasets present in the ST Figures 5 and 6, some very interesting properties emerge. First, the 'petri dish' that is present in the Figure 5, entries *a* and **b** is ignored by the model! Similarly, the 'snail trails' present in Figure 5 entry *b* and the 'water shadows' present in Figure 6 entry *j* are ignored as well! Figure 5 entry *a* is segmented well, as are the entries Figure 6 entries *f, h* and *j*.

Finally, the results from Table 4 reveal that there might be better performance conducted on the Raw datasets (as apposed to the processed datasets). Thankfully, the variance between the Mean IoU score between patches and the entire image suggest that either metric is a reliable way of identifying the model's success.

## 4.5 Possible Extensions

From the work conducted, this paper was not able to investigate post processing techniques for the predicted masks. An extension of this research could be to recreate the successful models, and then conduct post-processing to try and recover valuable information. Possible techniques could include *Region Growing or Watershed Method* (to recover cell boundaries) as well as *Region Filling or Morphological Operations* (to smooth out partial segmentations, appearing in a 'donut' shape).

It may be beneficial for others to investigate whether patch sizes of 128 or 512 improves the model, when using more advanced hardware. Other possible contributions may involve a relationship between the quantity of training data necessary to the try bring about high levels of correct predictions.

Others may wish to use this research on individual datastes - and see if that proves useful. This may be desirable for datasets that possess challenging features, such as tightly packed cells, clustered cells, having multiple kinds of cells in the image, having varied cells boundaries, having intense light present, having camera shake, or containing lots of noise.

It may be beneficial for this research to be extended into a comparison between only Neural Networks - and not attempt to use any traditional segmentation techniques, to save time. Other papers may also wish to do a comparison between simpler and more complicated U-Net architectures.

Other papers may benefit from using this research as a starting point to conduct cell tracking on the 10 datasets provided by the CTC [2]

Other research may benefit by cropping an equal portion of every edge - not just the bottom and far right edges.

**Table 4.** Table showing the Mean IoU Scores on 24 U-Net models, trained on both the raw datasets and processed datasets

| Dataset | Model Name | Patch Size | Mean IoU Score - Patches | Mean IoU Score - Images |
|---|---|---|---|---|
| Raw GT | 128_1 | 128 | 0.000202 | 0.000299 |
| | 128_2 | 128 | 0.232335 | 0.138728 |
| | 128_3 | 128 | 0.230308 | 0.162782 |
| | 256_1 | 256 | 0.097374 | 0.026934 |
| | 256_2 | 256 | 0.449333 | 0.284303 |
| | 256_3 | 256 | 0.425110 | 0.280346 |
| | 512_1 | 512 | 0.067855 | 0.068410 |
| | 512_2 | 512 | 0.262273 | 0.139076 |
| | 512_3 | 512 | 0.299222 | 0.183152 |
| Raw ST | 128_1 | 128 | 0.428539 | 0.325181 |
| | 128_2 | 128 | 0.539064 | 0.558336 |
| | 128_3 | 128 | 0.536814 | 0.571650 |
| | 256_1 | 256 | 0.548595 | 0.472323 |
| | 256_2 | 256 | 0.729635 | 0.718074 |
| | 256_3 | 256 | 0.773320 | 0.786547 |
| | 512_1 | 512 | 0 | 0 |
| | 512_2 | 512 | 0 | 0 |
| | 512_3 | 512 | 0.5005644 | 0.186807 |
| Processed GT | 256_1 | 256 | 0.306474 | 0.416345 |
| | 256_2 | 256 | 0.285340 | 0.338467 |
| | 256_3 | 256 | 0.343469 | 0.338467 |
| Processed ST | 256_1 | 256 | 0.641061 | 0.546153 |
| | 256_2 | 256 | 0.706629 | 0.667407 |
| | 256_3 | 256 | 0.709928 | 0.667407 |

**Table 5.** Table showing the contents of the 20 Jupyter Notebooks used

| **Notebook Number** | **Programming Environment** | **Notebook Size (MB)** | **Notebook Summary** |
|---|---|---|---|
| 001 | VS Code | 0.004 | *Welcome and Setup* - Explains how to recreate environment to run all notebooks, and dependencies |
| 002 | VS Code | 5.94 | *Dataset Exploration* - Investigates how to represent images together, colourmaps, and hidden information |
| 003 | VS Code | 6.67 | *Dataset Colour Exploration* - colourmap comparison and side-by-side video comparison |
| 004 | VS Code | 19 | *Dataset Initial Pre-Processing* - pre-processing techniques |
| 005 | VS Code | 5.59 | *Dataset Secondary Pre-Processing* - pre-processing techniques |
| 006 | VS Code | 1.13 | *Processing Images* - bulk morphological processing |
| 007 | VS Code | 16.9 | *Initial Segmentation* - traditional segmentation techniques |
| 008 | VS Code | 12 | *Secondary Segmentation* - traditional segmentation techniques |
| 009 | VS Code | 0.0666 | *Bulk Segmentation* - apply bulk thresholding with OpenCV and generate side-by-side video comparison |
| 010 | VS Code | 1.11 | *Isolate Training Data* - this notebook organises images and masks for segmentation |
| 011 | Google Colab | 5.1 | *Neural Network Benchmark Preparation* - prepares training data for raw dataset models, as a benchmark |
| 012 | Google Colab | 17.9 | *Neural Network Benchmarks* - trains 18 models on Raw datasets |
| 013 | Google Colab | 27.6 | *Neural Network Benchmark Reflection* - reflect on model success and compare information |
| 014 | VS Code | 0.0111 | *Dataset Tertiary Pre-Processing* - reviews that processed datasets are best option for Neural Network Segmentation |
| 015 | Google Colab | 0.348 | *Neural Network Processed Preparation* - prepares training data for processed dataset models |
| 016 | Google Colab | 3.3 | *Neural Network Processed Images* - trains 6 models on processed datasets |
| 017 | Google Colab | 8 | *Neural Network Processed Reflection* - reflect on model success and compare information |
| 018 | Google Colab | 10.8 | *Neural Network Predictions altogether* - load and predict models side-by-side, as well as mix-and-match models and datasets |
| 019 | Google Colab | 8.4 | *Neural Network Image Mean IoU Scores* - load models and datasets and conduct image Mean IoU Scores |
| 020 | VS Code | 0.060 | *Conclusion* - condenses all prior work and insights into 1 notebook |
| **Total (MB)** | | **141.93** | |

## 5 Conclusion

This paper considered segmentation techniques on the 10 datasets described in this paper. It began by considering traditional segmentation techniques, and then moved on to Neural Network segmentation using the U-Net architecture.

The traditional segmentation techniques yielded no usable results, and this paper concluded that traditional segmentation techniques, to segment all 10 of the datasets used, may not be possible. However, the use of Neural Networks hint at the possibility of a generic model being built and used to classify any cell datasets. This generic model can be designed and implemented to use **patchified** images - that is, a collection of smaller pieces of the images, with a more consistent square dimension.
By taking a set of images and producing a larger set of patchified images, with smaller square dimensions, it is possible to feed any images into the Neural Network model for training. This paper found that the best patch size to use may be a patch size of 256 - yielding a good compromise between reduced file dimension, visible features present in the patch and resource management.

Specifically, the ST datasets used in this research, with a patch size of 256, achieve the best results. These results are a Mean IoU score between 70% and 80%, depending on the kind of dataset used, as well as whether the patches are used to calculate the score, or the entire image.
There is a need to extend this work, in order to use these insights to conduct broad level cell tracking.

## 6 Software

20 Jupyter notebooks were created as the software for this research. More information about each notebook can be found in Table 5.

## 7 Resources Used

The author used a combination of resources to achieve this project.

VS Code was used as the IDE for the majority of the Jupyter Notebooks, with resources being saved offline onto a laptop. For the Neural Network experiments, Google Colaboratory was used in conjunction with Google Drive.

The laptop used has the following specifications: HP i7 9th gen CPU, 16GB RAM, Intel UHD Graphics 630 inbuilt graphics card, NVIDIA GeForce GTX 1660 Ti dedicated graphics card, 250 GB SSD and 1TB HDD.

Windows 10 home was used as the operating system. Anaconda was used as a virtual environment.

With the exception of Figures 1 and 2, all the images uploaded in this paper were generated by the author, and the original images came from the CTC [2] website. Figures included were generated by the author online, through a website called App Diagrams

All notebooks have been uploaded onto Github under the following link: UKZN COMP700 Project. The zipped source files of all 10 datasets is ~7GB, and will not be uploaded onto Github to prevent pull issues with other people. Instead, the user may download the datasets from the source destination, at the following link: CTC datasets. The author will attempt to upload the source files onto *SharePoint* and share those resources with the necessary UKZN staff as well.

## 8 Data Availability

The data used to support the findings of this study are publicly available.

## 9 Conflict of Interest

The authors declare that they have no conflicts of interest.

## References

[1] The transmission electron microscope | ccber. URL: https://www.ccber.ucsb.edu/ucsb-natural-history-collections-botanical-plant-anatomy/transmission-electron-microscope.

[2] Cell tracking challenge – where your software moves cells…, 2013. URL: http://celltrackingchallenge.net/.

[3] Yousef Al-Kofahi, Alla Zaltsman, Robert Graves, Will Marshall, and Mirabela Rusu. A deep learning-based algorithm for 2-d cell segmentation in microscopy images. *BMC Bioinformatics*, 19, 10 2018. doi:10.1186/s12859-018-2375-z.

[4] Tal Ben-Haim and Tammy Riklin-Raviv. Graph neural network for cell tracking in microscopy videos. *arXiv*, 02 2022. URL: https://arxiv.org/abs/2202.04731v1, doi:10.48550/arXiv.2202.04731.

[5] Dr Sreenivas Bhattiprolu. Python for microscopists and other image processing enthusiasts, 11 2022. URL: https://github.com/bnsreenu/python_for_microscopists/blob/master/204-207simple_unet_model.py.

[6] Elnaz Fazeli, Nathan H. Roy, Gautier Follain, Romain F. Laine, Lucas von Chamier, Pekka E. Hänninen, John E. Eriksson, Jean-Yves Tinevez, and Guillaume Jacquemet. Automated cell tracking using stardist and trackmate. *F1000Research*, 9:1279, 10 2020. doi:10.12688/f1000research.27019.1.

[7] Klas EG Magnusson. *Segmentation and Tracking of Cells and Particles in Time-lapse Microscopy.* PhD thesis, 11 2016.

[8] Erick Moen, Enrico Borba, Geneva Miller, Morgan Schwartz, Dylan Bannon, Nora Koe, Isabella Camplisson, Daniel Kyme, Cole Pavelchek, Tyler Price, Takamasa Kudo, Edward Pao, William Graf, and David Van Valen. Accurate cell tracking and lineage construction in live-cell imaging experiments with deep learning. 10 2019. doi:10.1101/803205.

[9] George Rice. Fluorescent microscopy, 01 2019. URL: https://serc.carleton.edu/microbelife/research_methods/microscopy/fluromic.html.

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *Lecture Notes in Computer Science*, pages 234–241, 2015. doi:10.1007/978-3-319-24574-4_28.

[11] Tim Scherr, Katharina Löffler, Moritz Böhland, and Ralf Mikut. Cell segmentation and tracking using cnn-based distance predictions and a graph-based matching strategy. *PLOS ONE*, 15:e0243219, 12 2020. doi:10.1371/journal.pone.0243219.

[12] Jing Sun, Attila Tárnok, and Xuantao Su. Deep learning-based single-cell optical image studies. *Cytometry Part A*, 97:226–240, 01 2020. doi:10.1002/cyto.a.23973.

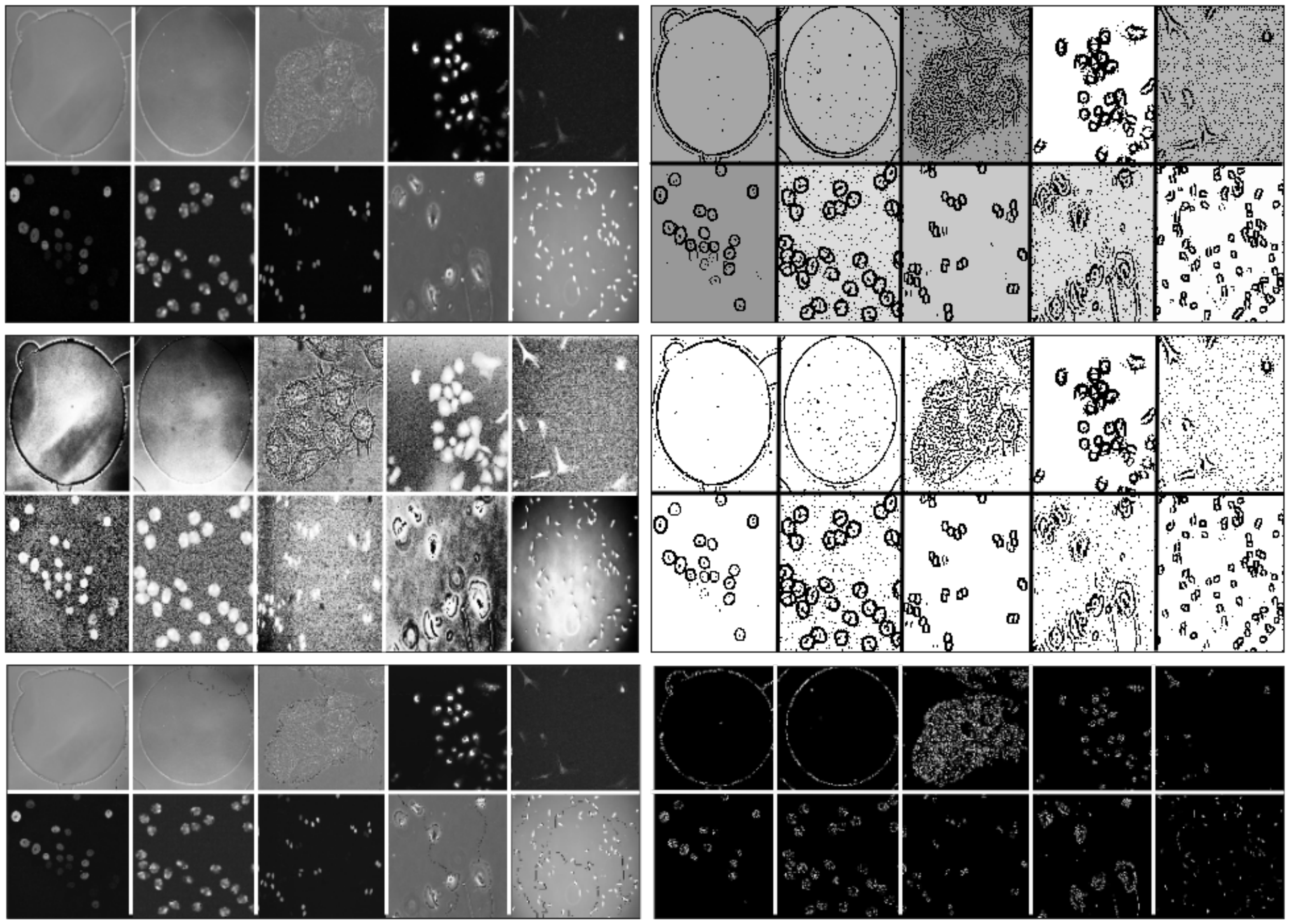
[13] Vladimír Ulman, Martin Maška, Klas E G Magnusson, Olaf Ronneberger, Carsten Haubold, Nathalie Harder, Pavel Matula, Petr Matula, David Svoboda, Miroslav Radojevic, Ihor Smal, Karl Rohr, Joakim Jaldén, Helen M Blau, Oleh Dzyubachyk, Boudewijn Lelieveldt, Pengdong Xiao, Yuexiang Li, Siu-Yeung Cho, Alexandre C Dufour, Jean-Christophe Olivo-Marin, Constantino C Reyes-Aldasoro, Jose A Solis-Lemus, Robert Bensch, Thomas Brox, Johannes Stegmaier, Ralf Mikut, Steffen Wolf, Fred A Hamprecht, Tiago Esteves, Pedro Quelhas, Ömer Demirel, Lars Malmström, Florian Jug, Pavel Tomancak, Erik Meijering, Arrate Muñoz-Barrutia, Michal Kozubek, and Carlos Ortiz-de Solorzano. An objective comparison of cell-tracking algorithms. *Nature Methods*, 14:1141–1152, 10 2017. URL: https://www.nature.com/articles/nmeth.4473, doi:10.1038/nmeth.4473.

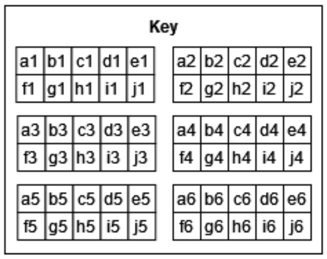
[14] Andong Wang, Qi Zhang, Yang Han, Sean Megason, Sahand Hormoz, Kishore R. Mosaliganti, Jacqueline C. K. Lam, and Victor O. K. Li. A novel deep learning-based 3d cell segmentation framework for future image-based disease detection. *Scientific Reports*, 12, 01 2022. doi:10.1038/s41598-021-04048-3.

[15] Junjie Wang, Xiaohong Su, Lingling Zhao, and Jun Zhang. Deep reinforcement learning for data association in cell tracking. *Frontiers in Bioengineering and Biotechnology*, 8, 04 2020. doi:10.3389/fbioe.2020.00298.

[16] Weiyuan Wu. Patchify source code, 11 2022. URL: https://github.com/dovahcrow/patchify.py.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009





Key

| | | | | |
|---|---|---|---|---|
| a1 | b1 | c1 | d1 | e1 |
| f1 | g1 | h1 | i1 | j1 |

| | | | | |
|---|---|---|---|---|
| a2 | b2 | c2 | d2 | e2 |
| f2 | g2 | h2 | i2 | j2 |

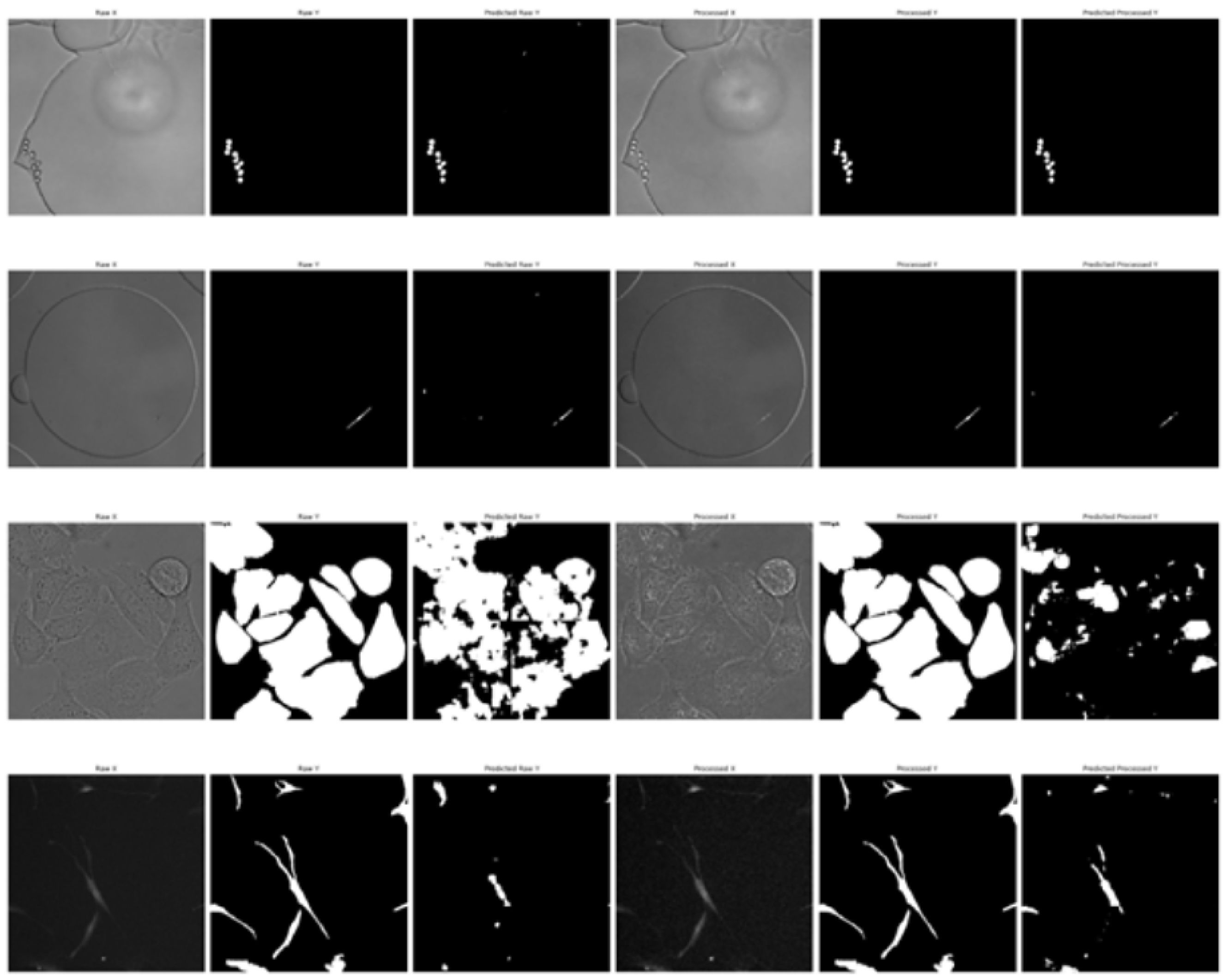
| | | | | |
|---|---|---|---|---|
| a3 | b3 | c3 | d3 | e3 |
| f3 | g3 | h3 | i3 | j3 |

| | | | | |
|---|---|---|---|---|
| a4 | b4 | c4 | d4 | e4 |
| f4 | g4 | h4 | i4 | j4 |

| | | | | |
|---|---|---|---|---|
| a5 | b5 | c5 | d5 | e5 |
| f5 | g5 | h5 | i5 | j5 |

| | | | | |
|---|---|---|---|---|
| a6 | b6 | c6 | d6 | e6 |
| f6 | g6 | h6 | i6 | j6 |

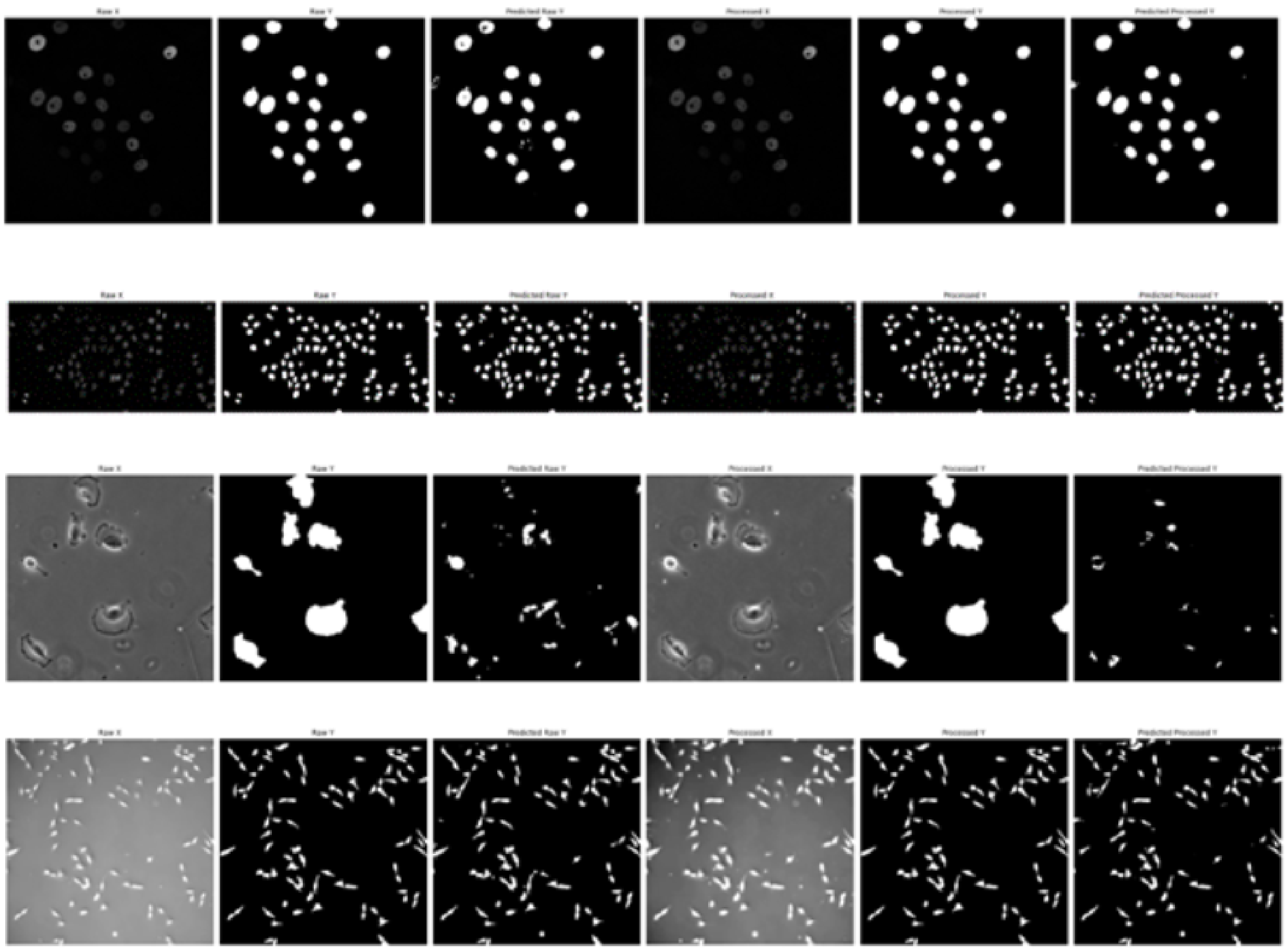
**Figure 4.** Figure showing a sample of the processing and segmentation techniques considered



**Key**

| | | | | | |
|---|---|---|---|---|---|
| a1 | a2 | a3 | a4 | a5 | a6 |
| b1 | b2 | b3 | b4 | b5 | b6 |
| c1 | c2 | c3 | c4 | c5 | c6 |
| e1 | e2 | e3 | e4 | e5 | e6 |

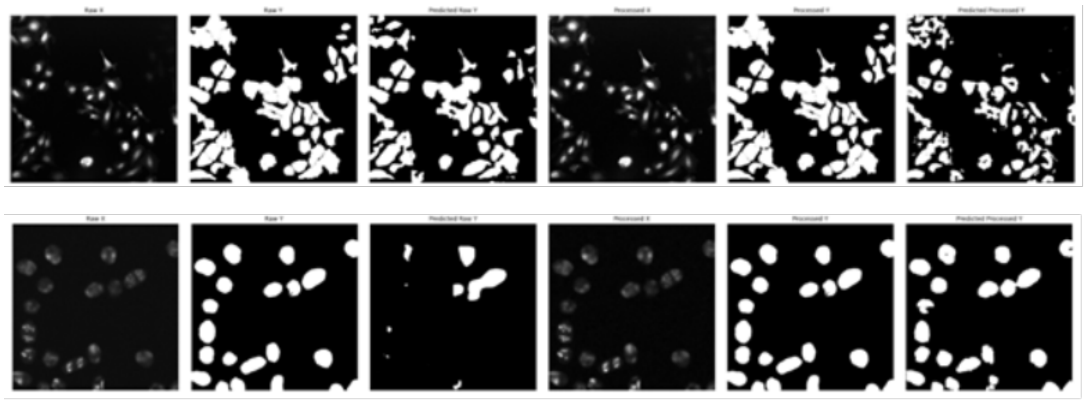
**Figure 5.** Figure showing 4 ST predictions on raw and processed datasets



**Key**

| | | | | | |
|---|---|---|---|---|---|
| f1 | f2 | f3 | f4 | f5 | f6 |
| h1 | h2 | h3 | h4 | h5 | h6 |
| i1 | i2 | i3 | i4 | i5 | i6 |
| j1 | j2 | j3 | j4 | j5 | j6 |

**Figure 6.** Figure showing 4 ST predictions on raw and processed datasets



**Key**

| | | | | | |
|---|---|---|---|---|---|
| d1 | d2 | d3 | d4 | d5 | d6 |
| g1 | g2 | g3 | g4 | g5 | g6 |

**Figure 7.** Figure showing 2 GT predictions on raw and processed datasets

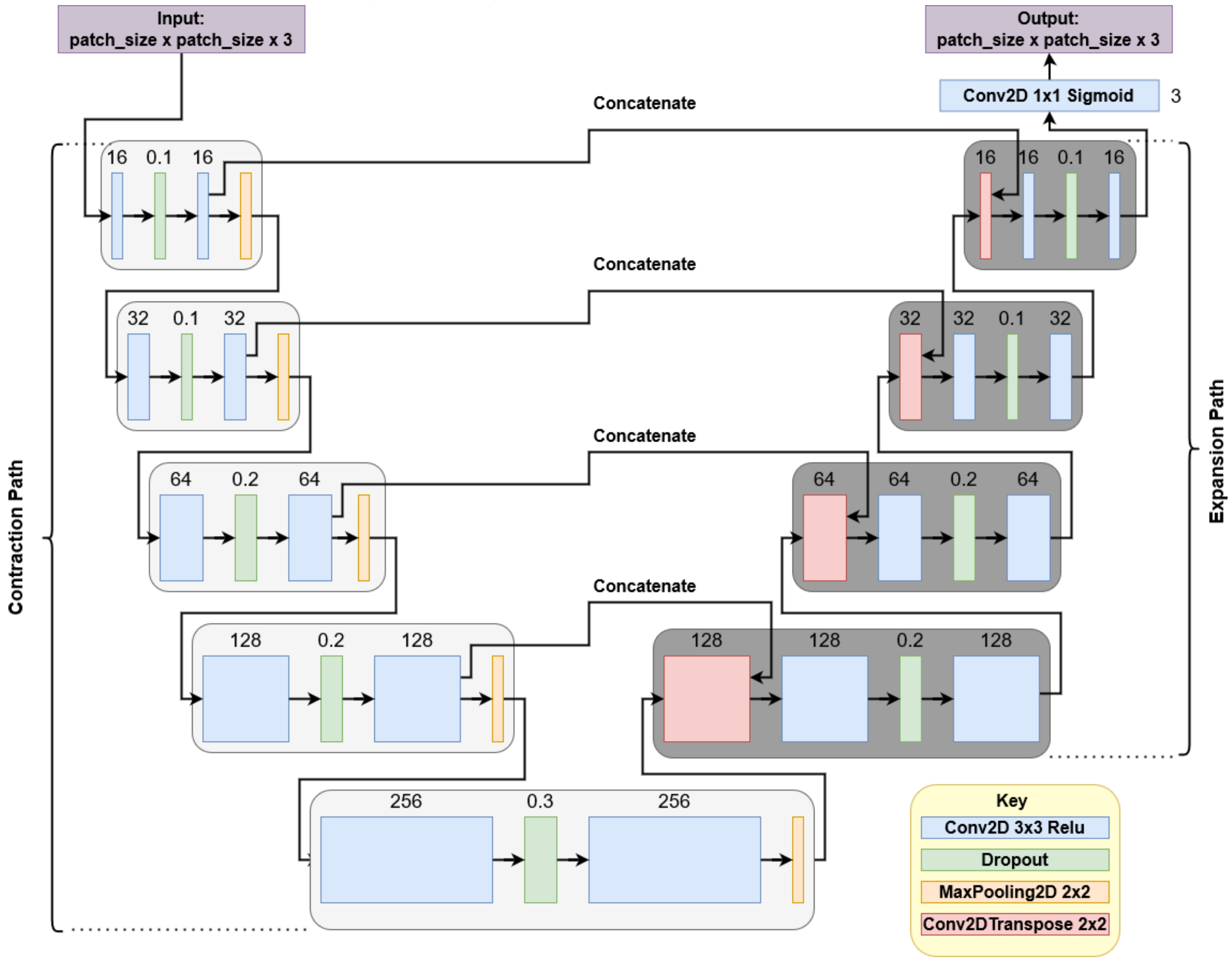


**Figure 8.** Figure showing the U-Net architecture used