AN END TO END PIPELINE TO LOCALIZE NUCLEI IN MICROSCOPIC ZEBRAFISH EMBRYO IMAGES

by

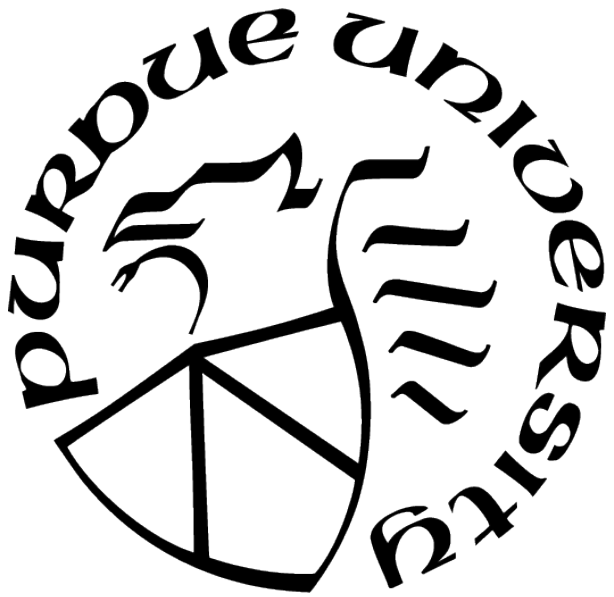
Juan Andres Carvajal Ayala

A thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Electrical and Computer Engineering



Electrical and Computer Engineering

West Lafayette, Indiana

December 2020

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. David Umulis, Co-Chair

Department of Biomedical Engineering

Dr. Edward Delp, Co-Chair

Department of Electrical and Computer Engineering

Dr. Aly El Gamal

Department of Electrical and Computer Engineering

Approved by:

Dr. Dimitrios Peroulis

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Determining the locations of nuclei in Zebrafish embryos is crucial for the study of the spatio-temporal behavior of these cells during the development process. With image segmentations, not only the location of the cell can be known, but also determine if each pixels is background or part of a nucleus. Traditional image processing techniques have been thoroughly applied to this problem. These techniques suffer from bad generalization, many times relying on heuristic that apply to a specific type of image to reach a high accuracy when doing pixel by pixel segmentation. In previous work from our research lab, wavelet image segmentation was applied, but heuristics relied on expected nuclei size .

Machine learning techniques, and more specifically convolutional neural networks, have recently revolutionized image processing and computer vision in general. By relying on vast amounts of data and deep networks, problems in computer vision such as classification or semantic segmentation have reached new state of the art performance, and these techniques are continuously improving and pushing the boundaries of state of the art.

The lack of labeled data to as input to a machine learning model was the main bottleneck. To overcome this, this work utilized Amazon Turk platform. This platform allows users to create a task and give instructions to 'Workers' , which agree to a price to complete each task. The data was preprocessed before being presented to the workers, and revised to make sure it was properly labeled.

Once labeled data was ready, the images and its corresponding segmented labels were used to train a U-Net model. In a nutshell, this models takes the input image, and at different scales, maps the image to a smaller vector. From this smaller vector, the model , again at different scales, constructs an image from this vector. During model training, the weights of the model are updated so that the image that is reconstructed minimizes the difference between the label image and the pixel segmentation.

We show that this method not only fits better the labeled ground truth image by the workers, but also generalizes well to other images of Zebrafish embryos. Once the model is trained, inference to obtain the segmented image is also orders of magnitude faster than previous techniques, including our previous wavelet segmentation method.

1. INTRODUCTION

1.1 Background

Zebrafish (Danio rerio) is a popular model organism to study in developmental biology. Its physiological and genetic similarities with humans make its study relevant for areas such as drug discovery, molecular genetics, and different diseases such as cancer. Moreover, in developmental biology, some of the characteristics displayed by Zebrafish make it an ideal model organism. The transparent embryo's development outside the mother and rapid development allow for convenient study and observation[1]. Computational imaging methods are applied to analyze and quantify the embryo's nuclei over space and time during the crucial periods of early development. Segmentation and localization techniques, widely employed in microscopic imaging, allow for this analytical characterization.

Several nuclei segmentation methods are available and utilized widely by the imaging community. Traditional techniques are commonly dependant on the contrast between foreground and background, or distinctive pixel features [2]–[4]. Preprocessing and calibration steps are also commonly applied to account for imaging environments. The variability and characteristics of the nuclei from real-life microscopic zebrafish images can prove challenging to the previously mentioned methods. Furthermore, images have a high density of overlapping nuclei. Techniques such as the watershed algorithm attempt to segment in these conditions, but over-cutting problems may occur [5], [6]. Other methods striving for accuracy are computationally expensive and may require previous knowledge [7]–[9]. Tzu-Ching Wu [10], [11] developed a wavelet-based segmentation method addressing some of these obstacles. Moreover, Wu applied it specifically on zebrafish embryo images. Our work builds upon Wu's work to improve both inaccuracies on segmentation and computational efficiency. A key focus of Wu's and our work is on usability. The intent is for other researchers to use the developed methods for further study of zebrafish and other organisms during their development process.

1.2 Wavelet Segmentation Method

The Wavelet-based segmentation method developed by Wu [10] addresses challenges associated with the pre-processing and calibrations steps of threshold-based segmentation. With only a nuclei scale factor as an input parameter, its output is robust to noise and changes in intensity. Dependence on only one input parameter allows for a decreased computation complexity compared to other segmentation methods.

The wavelet transform offers an analysis of both time and frequency on its input signals. Wavelets are zero-mean signals that process data at different scales. The convolution operation can be applied between input signals and a specific wavelet to obtain information from the input signal. These outputs, named wavelet coefficients, measure the correlation in frequency and time between the wavelet and each section of the signal [12]. Some applications of these techniques are image compression and denoising.

Wu applies the 2D Continous Wavelet Transform (2D CWT) to embryo images at different scales. Across the different scales, consistent regions from the input image corresponding to nuclei have similar value in the output coefficient matrix. Similarly, background regions in the input image, which vary significantly, display similar values of their own in the coefficient matrixes. Local minimums that appear at the same location across the different scales in the coefficient matrixes are identified as potential nuclei center locations. The area between these local minimums and the zero-cross section is identified as the nuclei mask. If across scales, these masks show a difference lower than a specific threshold, they are identified as nuclei.

On both public datasets and zebrafish embryo images, Wu's wavelet method achieves a higher True Positive (TP) rate and Precision than Derivative Sum (DS), point-wise, and Otsu's methods. With a larger set of test images, however, the performance of these metrics decrease. Overlapping nuclei and areas with a high density of nuclei proved especially difficult for the wavelet method.

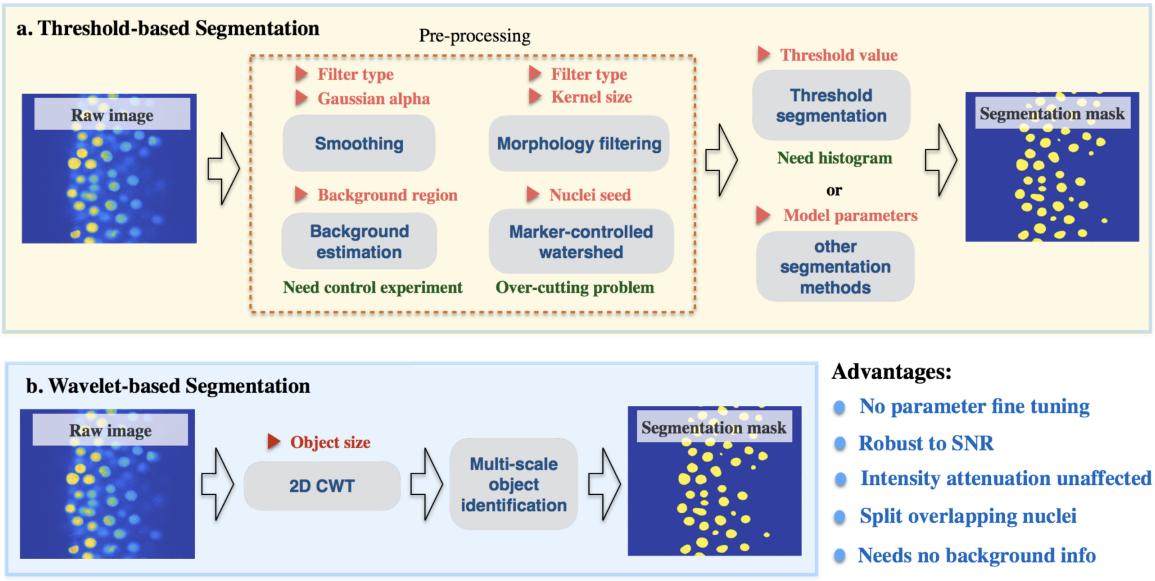


Figure 1.1. Segmentation methods comparison. Figure from [11]. (a) Describes a pipeline with traditional nuclei segmentation methods. (b) Wu's proposed wavelet based segmentation provides good segmentation accuracy without pre-processing steps and only one input parameter.

1.3 U-Net Segmentation Method

Designing a segmentation algorithm that can be computationally efficient, can generalize to a large number of real embryo images, and does not depend on input parameters or heuristics to reach high accuracy is a challenging task. In recent years, deep learning techniques have become ubiquitous in different image processing and computer vision problems, including microscopic image analysis [13]. Several deep learning models have been developed specifically for image segmentation.

These models leverage labeled data to update the model's parameters toward the goal of minimizing a loss function. The loss function compares the output of the model with the desired ground truth data. Given enough training data, this training process will result in a model that can capture the variability of the data when giving the output. For segmentation problems, given an image as an input, the model outputs masks of the desired objects present in the input image.

To generate the ground truth masks of the nuclei in embryo images, this work utilized Amazon Mechanical Turk. This tool allows us to set up a GUI where 'workers' label images following the requester's instructions. Because hundred of users can work on these labeling tasks at the same time, the process of generating data becomes much more efficient and streamlined.

Different segmentation model architectures have been developed. This works applies the U-Net model architecture [14]. This model contains an encoder block first, which is trained to captures the context of the image. This context is passed to a decoder block, which outputs the output mask from this context.

A trained model with the generated ground truth data is able to capture the features form the input images and generate the corresponding output segmentation masks. When running the model on inference mode, the used tools allow for the models to run batches of images in parallel, increasing time efficiency. Furthermore, when combining our masks with some traditional segmentation methods, we obtain a segmentation pipeline robust to changes in intensity, high-density areas, and overlapping nuclei.

2. LABELING NUCLEI IN ZEBRAFISH EMBRYOS USING AMAZON TURK

2.1 CZI File Pre-processing

The obtained zebrafish embryo microscopic image data is in the CZI file format. For each embryo, the file contains Z-stacks of 2-dimensional images. Each slice in the stack is the image representation of the embryo at a specific depth in the z-axis. While separate data channels return image representation of varied chemical reactions with the zebrafish embryo, we focus on the nuclei channel. Given the oval shape of the developing embryos, nuclei at the different z-axis depths will concentrate differently relative to the fixed-size 2-D slice image. The intent is to generate ground truth nuclei labels from these embryo slices. Therefore, to assure time and cost-efficiency in labeling, and to create relevant ground truth data, crucial preprocessing steps are applied. The final labeling pipeline needs to be both scalable to large-scale embryo data and reproducible to new data. These preprocessing steps, therefore, need to be automated.

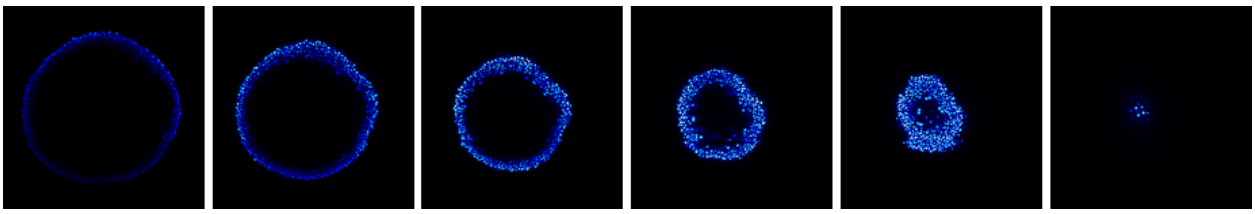


Figure 2.1. Z-slices at different depths of the embryo image. It can be appreciated how the area where nuclei are present changes at different depths of the z-axis.

Figure 2.3 shows examples of embryo slices at different depths. The first preprocessing step is to find the contours of the embryo's enclosing circle, discarding the black areas of the image that contain no nuclei information. To obtain each contour, we first apply an opening morphology operation with an ellipse as a structuring element. The objective of this is to preserve foreground objects with a similar shape to the structuring element. Next, we add a gaussian blur to remove some of the noise that might still be present in the image.

We finally binarize the image with a specific threshold, obtained using Otsu's method [2]. This technique minimizes the weighted within-class variance, $\sigma_w^2(t)$, to select the threshold. Equation 2.1 describes the equation to minimize, which depends on threshold t. $P(\mathrm{i})$ is the probability for each pixel value, taken from the normalized image histogram. Once the image is binarized, we find the position of the pixels that are foreground. We then find the maximum and minimum values of the x and y axes from those positions. From these values, we determine where to crop the original image to obtain a sub-image with just the relevant nuclei information.

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t) \tag{2.1}$$

$$q_1(t) = \sum_{\mathrm{i}=1}^{t} P(\mathrm{i}) \quad q_2(t) = \sum_{\mathrm{i}=t+1}^{I} P(\mathrm{i}) \tag{2.2}$$

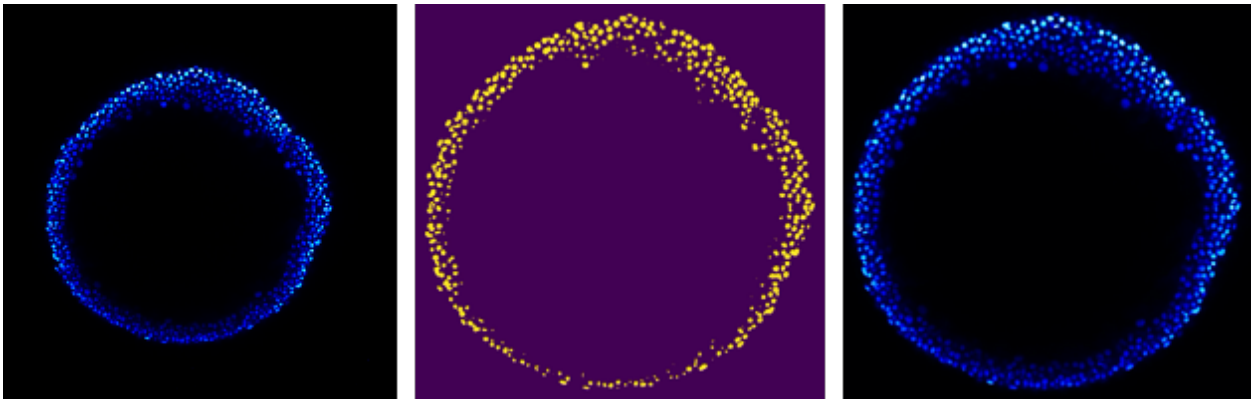
$$\mu_1(t) = \sum_{\mathrm{i}=1}^{t} \frac{\mathrm{i}P(\mathrm{i})}{q_1(t)} \quad \mu_2(t) = \sum_{\mathrm{i}=t+1}^{I} \frac{\mathrm{i}P(\mathrm{i})}{q_2(t)} \tag{2.3}$$

$$\sigma_1^2(t) = \sum_{\mathrm{i}=1}^{t} [\mathrm{i} - \mu_1(t)]^2 \frac{P(\mathrm{i})}{q_1(t)} \quad \sigma_2^2(t) = \sum_{\mathrm{i}=t+1}^{I} [\mathrm{i} - \mu_2(t)]^2 \frac{P(\mathrm{i})}{q_2(t)} \tag{2.4}$$

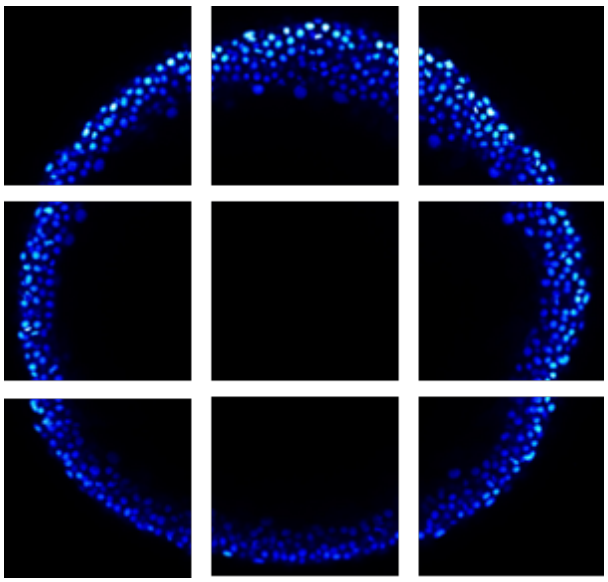
For an Amazon Turk worker to accept the task and provide an accurate segmentation, the given image should not present a task that is too complicated or time-consuming. For this reason, we slice the sub-image into further sub-images. Consequently, some slices will not have any nuclei information. To discard these slices, we check that the maximum and minimum pixel values are not equal. If they are equal, the slice does not hold any meaningful information. Finally, to be able to reconstruct each original embryo image slice, we save metadata pertinent to all the pre-processing.

2.2 Labelling Nuclei with Amazon Mechanical Turk

Amazon Mechanical Turk is a web platform where requesters post manual tasks to be answered or completed by Amazon Turk workers. Each task has a pre-defined price to be paid if the requester accepts the competed task and it is done correctly. The requester



(a) **Result of Otsu thresholding**



(b) **Slices of cropped image**

Figure 2.2. Embryo slices cropping and slicing. (a) Shows the result of thresholding and the respective cropping of the original image given the binary image. (b) Displays the resulting slices of the cropped image to pass on to the labelling task.

provides instructions and a Graphical User Interface with the necessary tools to complete the task. For our problem, we selected the Instance Segmentation Template provided by Amazon Turk. This template provides brush and polygon tools that allow the workers to label pixels of the images as specific instances of the given classes, which for our case it is only the nuclei class. Workers are asked to label each instance with a different color. Further instructions, along with both correct and incorrect segmented examples are given. To upload

batches of images, Turk requires a file pointing to the URL of each image at an AWS S3 bucket.

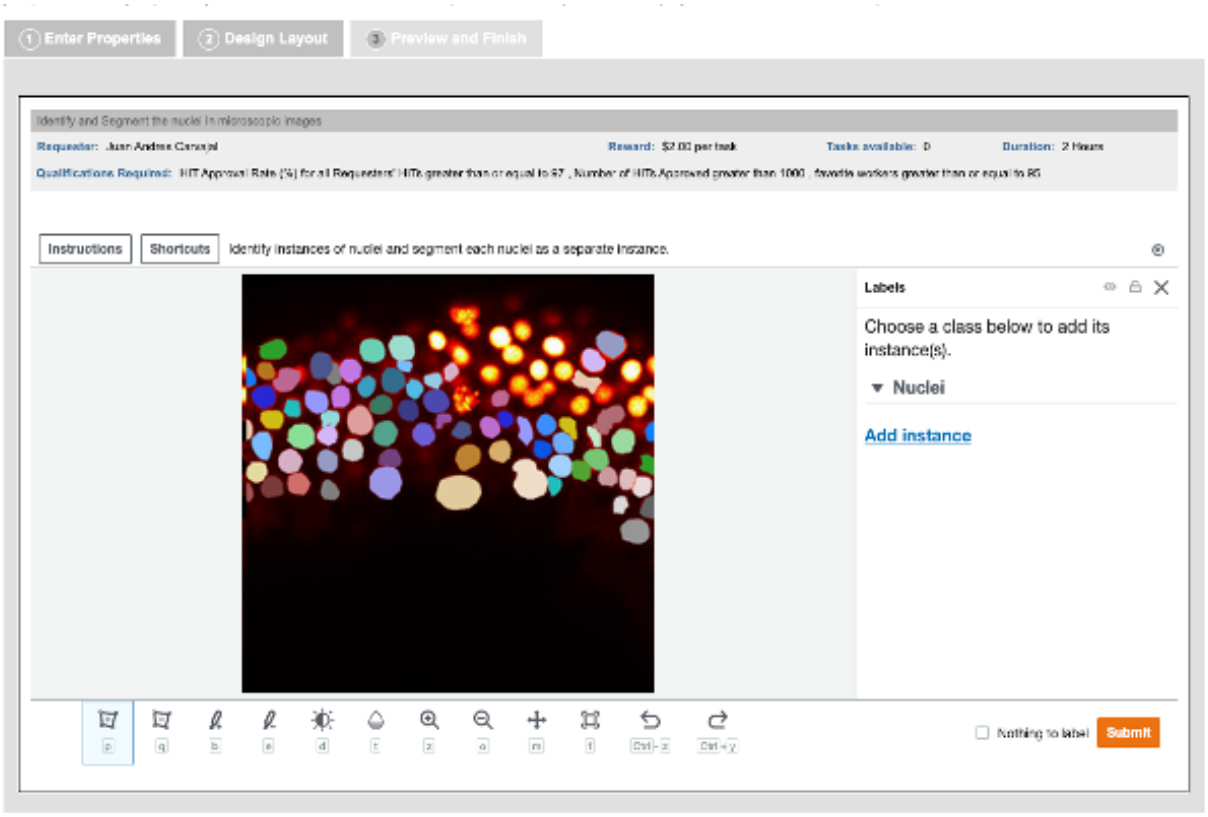


Figure 2.3. View of the Instance Segmentation tool on Amazon Mechanical Turk. The tools on the bottom of the image are used to label pixels corresponding to individual nuclei. The requester can access further instructions and explanation to what a nuclei is on the instructions tab seen on the upper left. It can be appreciated how each instance of a nucleus is labeled with a different color.

Amazon Mechanical Turk outputs the results via a csv file. The final image holding the nuclei mask annotations is encoded in base64. The colors, in HEX format, for each instance are given in a JSON object for each image. These two outputs are used to create the ground-truth data, which is explained in further detail in the next section.

2.3 Approving or Rejecting Labeling Work

Once workers finish labeling a batch of images, the requester must approve or reject each resulting image. We have developed a pipeline, along with a simple Graphical User Interface for a reviewer to accept or reject the worker's labeling in an efficient manner.

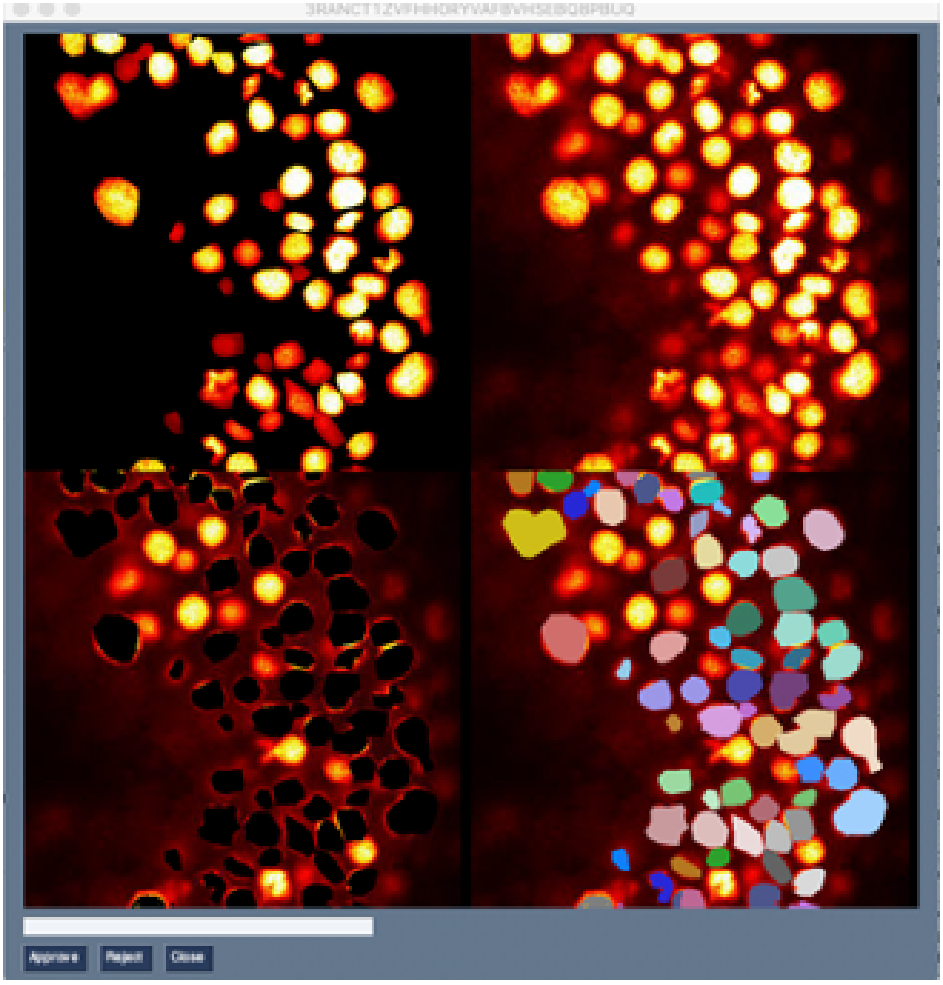


Figure 2.4. Graphical Interface to review image labeling. The original embryo image slice I is shown on the top right section. P_I on the upper left, respectively, provide good visualization of which pixels from the original image I will be labeled as nuclei. Conversely, N_I on the bottom left, show which pixels of the original image I will be left out due to the labelling. Ideally, these will mostly be pixels corresponding to noise. As shown in this example, with the given images, the reviewer can easily see which nuclei have been missed by the worker.

$$M_p[\mathrm{i,j}] = \begin{cases} 1 & \text{if } L[\mathrm{i,j}] > 0 \\ 0 & \text{otherwise} \end{cases} \qquad M_n[\mathrm{i,j}] = \begin{cases} 1 & \text{if } L[\mathrm{i,j}] = 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

$$P_I = I \circ M_p \tag{2.6}$$

$$N_I = I \circ M_n \tag{2.7}$$

As mentioned in the previous section, Amazon Turk will return a csv file with an image that contains only the instance labelling for each nucleus, defined here as L . For the review pipeline, we first generate two binary masks from L, M_p and M_n, as described in equation 2.5. With equations 2.6 and 2.7 we generate and P_I and N_I from the original embryo image I . These images hold only the pixel only values of the embryo image I where there is a labeled nucleus and where there is not, respectively. We then super-impose each nucleus on top of the original embryo image, to generate a superimposed image. Finally, we display I, P_I, N_I, and the superimposed image to the reviewer, as seen on Figure 2.4.

The pipeline creates a client service with the Amazon Mechanical Turk API. This allows the reviewer to only review the images that have not been approved or rejected, and skip the rest. Furthermore, it allows the reviewer to accept or reject the labeling directly via the GUI.

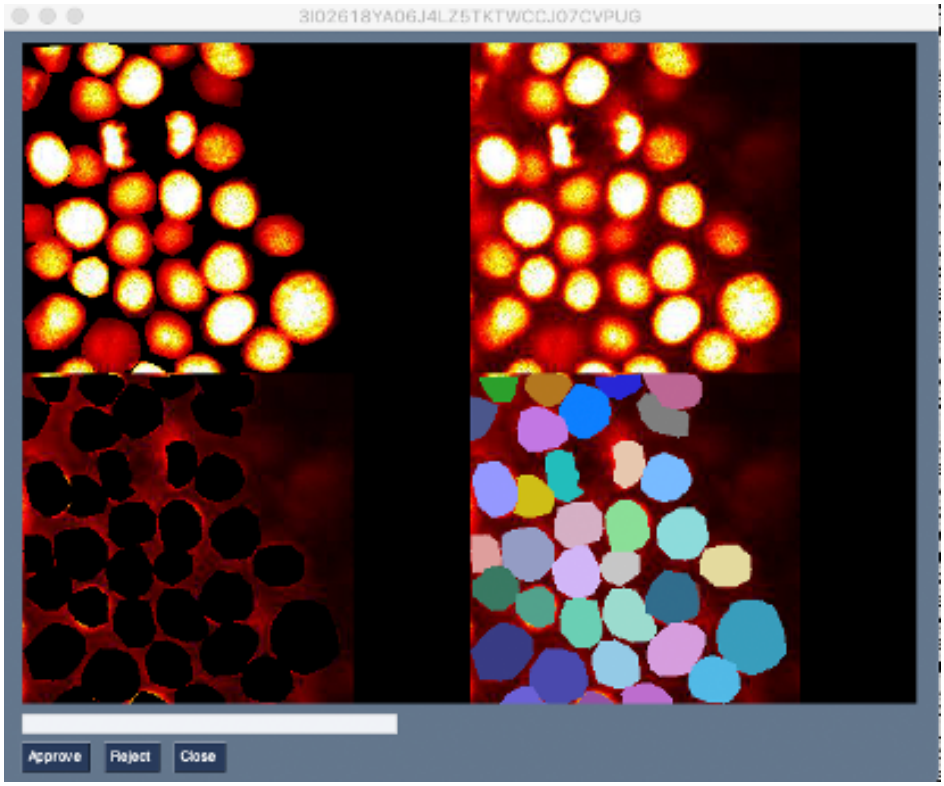


Figure 2.5. Graphical interface to reject or accept labeling. Once the reviewer has decided whether the image has been properly labeled or not, the reviewer can accept or reject using the bottoms in the lower bar. This will automatically, via the Amazon Turk API, tell the worker that his work has been approved or rejected. Additionally, the reviewer can optionally send a message to the worker to explain why his labelling has been approved or rejected.

3. TRAINING A U-NET MODEL TO PREDICT NUCLEI MASKS

3.1 Generating Training Data

A vast amount of labeled data is needed to train a Deep Learning model. In the previous chapter, we explained how this work used Amazon Mechanical Turk to generate ground truth data and the format of the labeled data. Additional pre-processing is needed to prepare the data for the model.

We want the model to predict not only a mask of each nucleus but also the border for each. For each image that went through the labeling pipeline, we have an image with all the labeled nuclei annotations and the JSON object with the color for each annotation instance. To obtain the borders as accurately as possible for each nuclei instance, we isolate each instance in individual binary images. We loop over the colors in the JSON object and find the matching pixel locations for each color in the full image. Basically, having N colors in the JSON object, represented as their respective pixel values $c_1, c2, ...c_N$, each representing an individual labeled nucleus annotation:

$$F[\mathrm{i,j},k] = \begin{cases} 1 & \text{if } L[\mathrm{i,j},k] = c_k \\ 0 & \text{otherwise} \end{cases} \qquad with \quad k \,\epsilon\, 1,2,,N \tag{3.1}$$

F then holds at each channel $k \,\epsilon\, 1,2,,N$ a binary mask of a single nuclei annotation. Using OpenCV's implementation of [15] we find the contours of the nuclei annotations from each binary mask. Since each binary mask only holds an individual nucleus annotation, this is a trivial task. Additional steps previous to finding the contour were taken to account for human error during the labeling, which are explained in a later chapter. From the contours obtained from F we generate matrix C. This holds at each channel k a binary mask of the respective contour from the nucleus found at each of F's channels.

Finally, applying equations 3.2 and 3.3, we obtain T_F and T_C. These are the complete binary images of all the nuclei instance masks and nuclei contours, respectively. We go

through the procedure explained in this section for each embryo image I and labeled image L pair. For the training process of our model, I will serve as the input, while T_F and T_C will serve as the ground truth output data.

$$T_F[\mathrm{i},\mathrm{j}] = \sum_{k=1}^{N} F[\mathrm{i},\mathrm{j},k] \tag{3.2}$$

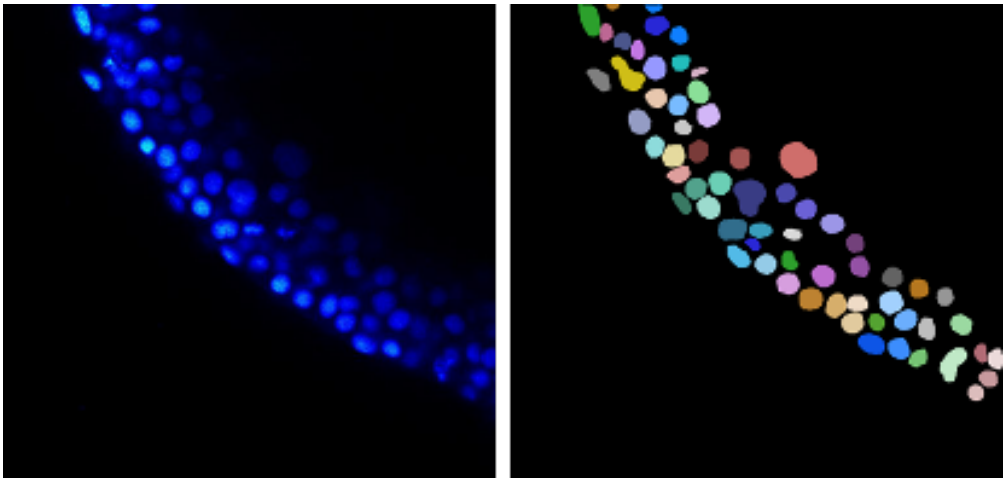
$$T_C[\mathrm{i},\mathrm{j}] = \sum_{k=1}^{N} C[\mathrm{i},\mathrm{j},k] \tag{3.3}$$

3.2 U-Net Model Architecture

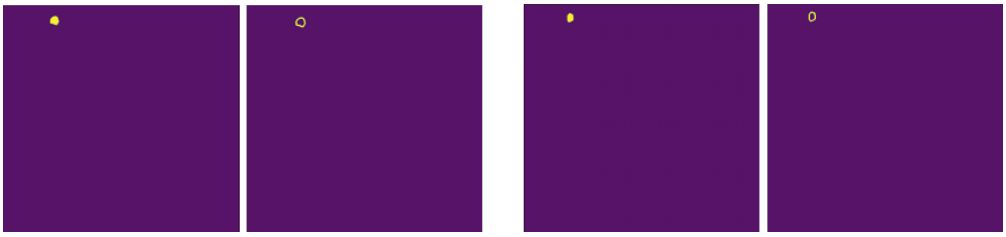
We use the U-Net Model architecture from the paper. After training, this model allows us to obtain a fine-grained segmentation mask output given an image of arbitrary size. The following subsections describe the components of this architecture and the functioning of the model as a whole.

3.2.1 Convolutional Layers

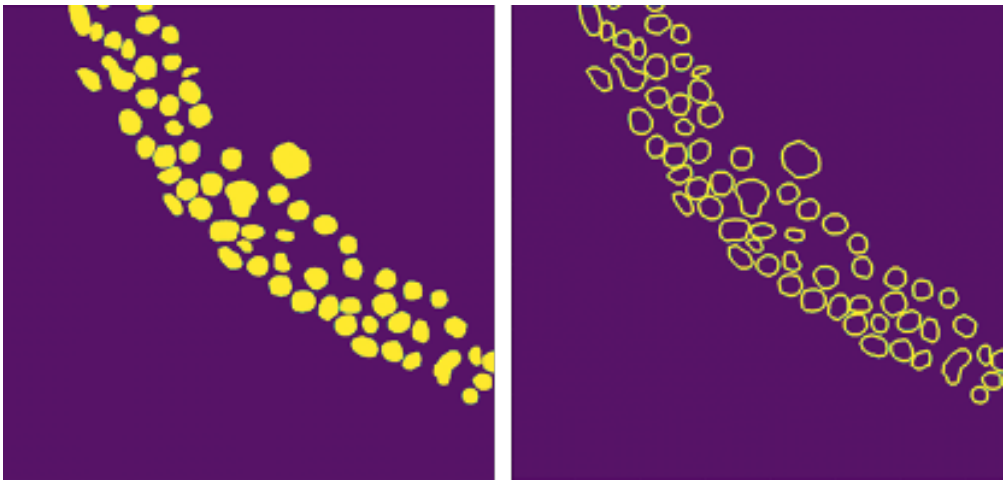
Convolution operations are the underlying mechanism of deep convolutional neural networks, which have propelled the success of deep learning. A discrete convolution is a linear transformation in which its output preserves ordering information from its input. The fundamental component of the convolution operation is the kernel. In the case of 2-dimensional images, represented as matrices, the kernel is a 2-dimensional matrix. The operation consists of sliding the kernel across the entire input. The values of the kernel overlap with corresponding values of the input at each location. First, the product between values that are overlapping is computed. Then these products are summed up to obtain the corresponding output for that specific location. This process is repeated over all locations as the kernel slides trough the input. The output of this operation will also be a matrix, named a feature map. Regularly, in convolutional neural networks, several kernels with distinct values are



(a) **Original Image I and Labeled Image L**



(b) **Individual nuclei annotations and respective contours**



(c) **Pair Binary images T_F and T_C used as Ground Truth.**

Figure 3.1. Generating training images from Amazon Turk labeling. Looping over the colors of the labels in (a) to isolate the nuclei will return images as in (b). After obtaining all the individual contours, applying equations 3.2 and 3.3 we obtain the training images as seen in (c)

utilized, resulting in the same number of feature maps as the number of kernels applied. If we have a 3-dimensional matrix as input, we denote this third dimension as channels. The

kernel would also need to be 3-dimensional and have the same amount of channels as the input to produce the 2-dimensional output feature map. The convolution operation also utilizes a stride parameter. The stride specifies how many locations steps to take when sliding the kernel trough the input. The result is equivalent to that of a subsampling [16]. The input can be padded with other values, commonly zero, to control the size of the output.Figure 3.2 describes this operation on a 2-dimensional input with a single channel.

The output feature maps will hold information on the response of the input with a specific kernel. For images, examples of kernel values can be edge detectors in different directions or changes in illumination for image inputs. In that case, the output feature maps will be a response with the filter represented by the kernel [17]. For deep convolutional neural networks, these kernels are not explicitly specified but learned. Furthermore, several operations or layers, including convolutions, are stacked and run sequentially. These operations are more commonly referred to as layers. The output from one operation can serve as input for a subsequent layer. The intuition behind this is that kernels at the initial layers will learn more basic filters, such as the examples given previously. As the layers are closer to the network's output, the kernels hold more abstract information. Each layer's output values are propagated through the network and the learned kernels map its inputs into relevant information for the subsequent layer. The goal is to allow the final layers to output pertinent information for the defined problem.

3.2.2 Rectified Linear Units

Convolution is a linear operation, which implies that when stacking only these operations in a network, the mapping between network input and output will be linear. Neural networks need to be able to learn non-linear mappings for nontrivial problems. For this reason, activations that compute a non-linear function were introduced. These activations are usually applied after a convolution. In our work, we use the rectified linear activation, commonly know as ReLUs, described in equation 3.4. This function is very close to linear, which helps during the optimization of gradient-based methods.

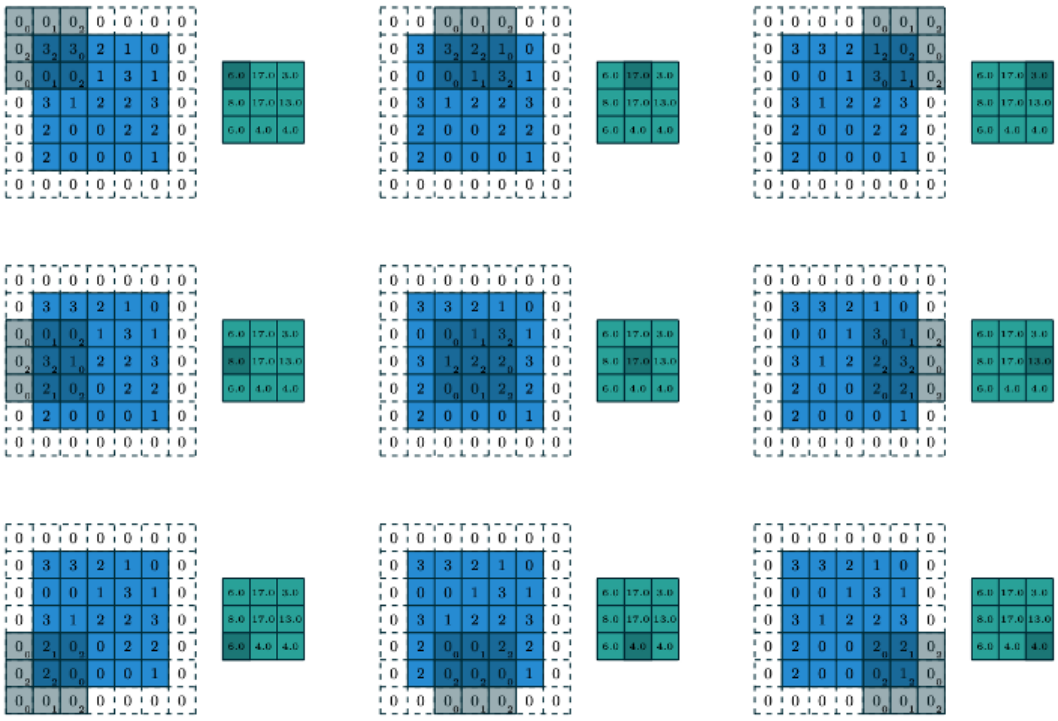


Figure 3.2. Convolution operation. Figure from [16] describing a convolution operation for a single kernel. The shaded area represents the kernel. Sliding the kernel trough the input, in blue, with a stride of 1, outputs each element of the output feature map, in green. Zero-padding is can also be seen at the edges of the input.

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} = \max\{0, x\} \tag{3.4}$$

3.2.3 Max-Pooling Layers

When applying convolution, the position of features at the input is maintained in the output feature map, describing the kernel response at each position. Consequently, if the input's features or objects or features have small variations in their position (e.g rotations, shifts), the output feature map would be completely different. Ideally, if the same features are present, albeit translated slightly, we want the kernel response at the output feature map to be the same. In other words, the feature maps should be translation invariant.

In convolutional neural networks, pooling layers provide this translation invariance. These layers slide a fixed-sized window over the feature map. At each location, one value obtained from those inside the window is selected as the output. Most commonly, as is the case in this work, the max-pooling layer is utilized. In this case, out of the values inside the window, the maximum value is selected. Usually, the window slides in a non-overlapping manner. The final output will be a down-sampled version of the input feature map, with its values chosen according to the described procedure. Figure 3.3 provides a simple example of this operation.

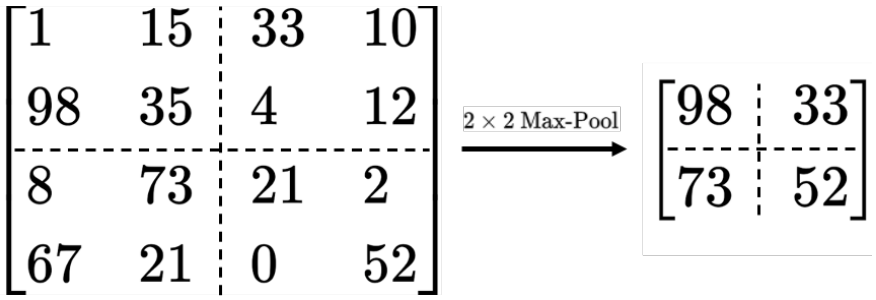


Figure 3.3. Max-Pooling Operation. The left matrix represents the input to the max-pool layer.In this case, 2×2 max-pooling is applied. The rightmost matrix shows the output of the operation.

3.2.4 Transposed Convolutional Layers

As seen on Figure 3.2, when applying a convolution operation, the output feature map has a lower resolution than the input. For tasks such as classification, this is acceptable since the intent is to learn what features are important in the image. However, convolutional neural networks can also up-sample input feature maps to generate outputs with higher resolution. The technique employed in these cases is the transposed convolution. The advantage of this operation over common interpolation is that it uses learned filters to produce the output. For segmentation tasks, this allows the output masks to present fine-grained details compared to the coarse ones obtained with interpolation.

In essence, the transposed convolution maps each element of the input to many elements at its output. We can represent the regular convolution operation as a matrix multiplication

between the kernel. Equation 3.5 shows an example of a 3×3 kernel K, and 3.6 an input I with size 4×4.

$$K = \begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix} \tag{3.5}$$

$$I = \begin{pmatrix} \mathrm{i}_{0,0} & \mathrm{i}_{0,1} & \mathrm{i}_{0,2} & \mathrm{i}_{0,3} \\ \mathrm{i}_{1,0} & \mathrm{i}_{1,1} & \mathrm{i}_{1,2} & \mathrm{i}_{1,3} \\ \mathrm{i}_{2,0} & \mathrm{i}_{2,1} & \mathrm{i}_{2,2} & \mathrm{i}_{2,3} \\ \mathrm{i}_{3,0} & \mathrm{i}_{3,1} & \mathrm{i}_{3,2} & \mathrm{i}_{3,3} \end{pmatrix} \tag{3.6}$$

From 3.5, we construct the kernel matrix C in 3.7. And by flattening 3.6 we obtain I_f. The output of the matrix multiplication between these two matrices will be of size 4×1. This matrix can be resized to obtain the output feature map of this convolution operation, F.

$$C = \begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix} \tag{3.7}$$

$$I_f = \begin{pmatrix} \mathrm{i}_{0,0} & \mathrm{i}_{0,1} & \mathrm{i}_{0,2} & \mathrm{i}_{0,3} & \mathrm{i}_{1,0} & \mathrm{i}_{1,1} & \mathrm{i}_{1,2} & \mathrm{i}_{1,3} & \mathrm{i}_{2,0} & \mathrm{i}_{2,1} & \mathrm{i}_{2,2} & \mathrm{i}_{2,3} & \mathrm{i}_{3,0} & \mathrm{i}_{3,1} & \mathrm{i}_{3,2} & \mathrm{i}_{3,3} \end{pmatrix}^T \tag{3.8}$$

$$CI_f = \begin{pmatrix} f_{0,0} \\ f_{1,0} \\ f_{2,0} \\ f_{3,0} \end{pmatrix} \quad F = \begin{pmatrix} f_{0,0} & f_{1,0} \\ f_{2,0} & f_{3,0} \end{pmatrix} \tag{3.9}$$

Having defined the convolution operation as matrix multiplication, we show how by transposing C and applying matrix multiplication between C^T and a vector L with size 4×1, resized from an input of size 2×2, we obtain a 16-dimensional vector as the output. This output is resized to obtain H, with size 4×4.

$$L = \begin{pmatrix} l_{0,0} & l_{0,1} & l_{0,2} & l_{0,3} \end{pmatrix}^T \tag{3.10}$$

$$C^T L = \begin{pmatrix} h_{0,0} & h_{0,1} & h_{0,2} & h_{0,3} & h_{0,4} & h_{0,5} & h_{0,6} & h_{0,7} & h_{0,8} & h_{0,9} & h_{0,10} & h_{0,11} & h_{0,12} & h_{0,13} & h_{014} & h_{0,15} \end{pmatrix}^T \tag{3.11}$$

$$H = \begin{pmatrix} h_{0,0} & h_{0,1} & h_{0,2} & h_{0,3} \\ h_{0,4} & h_{0,5} & h_{0,6} & h_{0,7} \\ h_{0,8} & h_{0,9} & h_{0,10} & h_{0,11} \\ h_{0,12} & h_{0,13} & h_{014} & h_{0,15} \end{pmatrix} \tag{3.12}$$

We have showed that by utilizing C^T, we can move from a lower resolution input L to a higher resolution H. Hence, by use of the transposed convolution matrix we have obtained a one-to-many mapping from the input to the output. Furthermore, just as the value of the convolutional kernels are learned, the optimal transposed convolution kernel values are also learned during training.

3.2.5 1×1 Convolution

As inputs are propagated through a neural network, each layer's output feature map will have as many channels as the number of kernels or filters present at that specific layer. Many filters are preferred, so that the the networks gains the ability to map different possible features at an input to some representation in the output. This contributes to mapping the initial input to a specific desired non-trivial output. For classification tasks, feature maps at some point in the layer are flattened and fed into a dense layer. We can combine and define

the output of dense layers to be the same number of classes in the classification task. Hence, this output is the network's mapping of inputs to the respective classes.

Segmentation networks, however, do not have a defined number of classes to output. Instead, the final output of a segmentation network is another image representing a mask from the input image. How to generate up-scaled feature maps has been described in the previous section. Similar to convolution, transposed convolution also uses several kernels for an optimal up-scaling, generating feature maps with the same number of channels. There needs to be a proper transition from the several feature map channels to the number of channels in the final output mask. As an example, if the output of the network is a gray-scale mask, then there needs to be a transition to a single channel. This network architecture makes use of 1×1 convolutions to learn the optimal way to combine these channels to achieve the desired output mask [18].

Given a feature map of size $m \times n \times k$ with k representing the number of channels, we pass this feature map as input to a layer with j convolutional kernels of size $1 \times 1 \times k$. Consequently, the output of this layer is a feature map of size $m \times n \times$ j. Note that the $1 \times 1 \times k$ kernels are learned during training. Therefore, this layer has learned an optimal way to combine k channels into j channels. Following the previous example of the desired output mask as a gray-scale image, j $= 1$ and the input feature maps are combined into a single channel.

3.2.6 Model Architecture

The U-Net model architecture is a fully convolutional network. This implies that all of its learned parameters are from convolutional layers and its derivations (e.g transposed convolutions). In [19] Long et al introduced the use of fully convolutional networks to produce a same size mask output from an input image. The mask output classifies each pixel in the input image as one of the possible classes. The authors utilize transposed convolution to upscale feature maps generated by a chain of convolutional layers. Additionally, [19] combines the outputs of convolutional operations with outputs of subsequent transposed convolutions. Further convolutional layers input this combination in order to properly assemble the given

information. Compared to only applying the up-scaling, this technique results in more fine-grained details on the output segmentation.

As detailed previously, the outputs of convolutional layers is a many to one mapping. Therefore, the convolution is in a sense summarizing the information, outputting a feature map with what is relevant to each learned filter. In contrast, the transposed convolutions, via their one to many mapping, attempt to recover localization information. Combining the outputs of these two operations generates a better sense of the important summarized features as well as the localization information.

In [14], Ronneberg et al extend this idea by generating a model architecture that increases the number of output channels during the up-scaling of the intermediate feature maps. This aids the summarized information from the convolutions to propagate better during the up-sampling stages to higher resolutions.

Figure 3.5 describes the architecture. An input image is passed trough two subsequent convolutional layers with 3×3 kernels. A rectified linear unit (ReLU) activation follows each convolutional layer. The output feature maps are then down-scaled with max-pooling layers. Next, the process of passing the input feature maps trough the convolutional operations and then reducing the scale with max-pooling is repeated. This continues until the feature map reaches a reduced size. This sequence is defined as the contracting path. Intuitively, this path obtains the important information from what is present in the input image by applying convolution at different scales. The vector at the end of this path is commonly referred to as the context.

The context vector is then used as input for a 2×2 transposed convolution, doubling its resolution. At this point, the up-scaled feature maps are stacked together with the closest resolution output feature maps of the contracting path, cropped to fit the up-scaled result. As previously explained, this information, with details as to what features are present as well as where they are located, is optimally combined by applying further convolution layers followed by ReLU activations. Repetition of this up-scaling and combinations steps generates a symmetric path to the contracting process. By continuously up-scaling and combining information, the network yields feature maps with a resolution similar to that of the original

image. By applying 1×1 convolution, the higher resolution feature maps are synthesized into the desired segmentation channels.

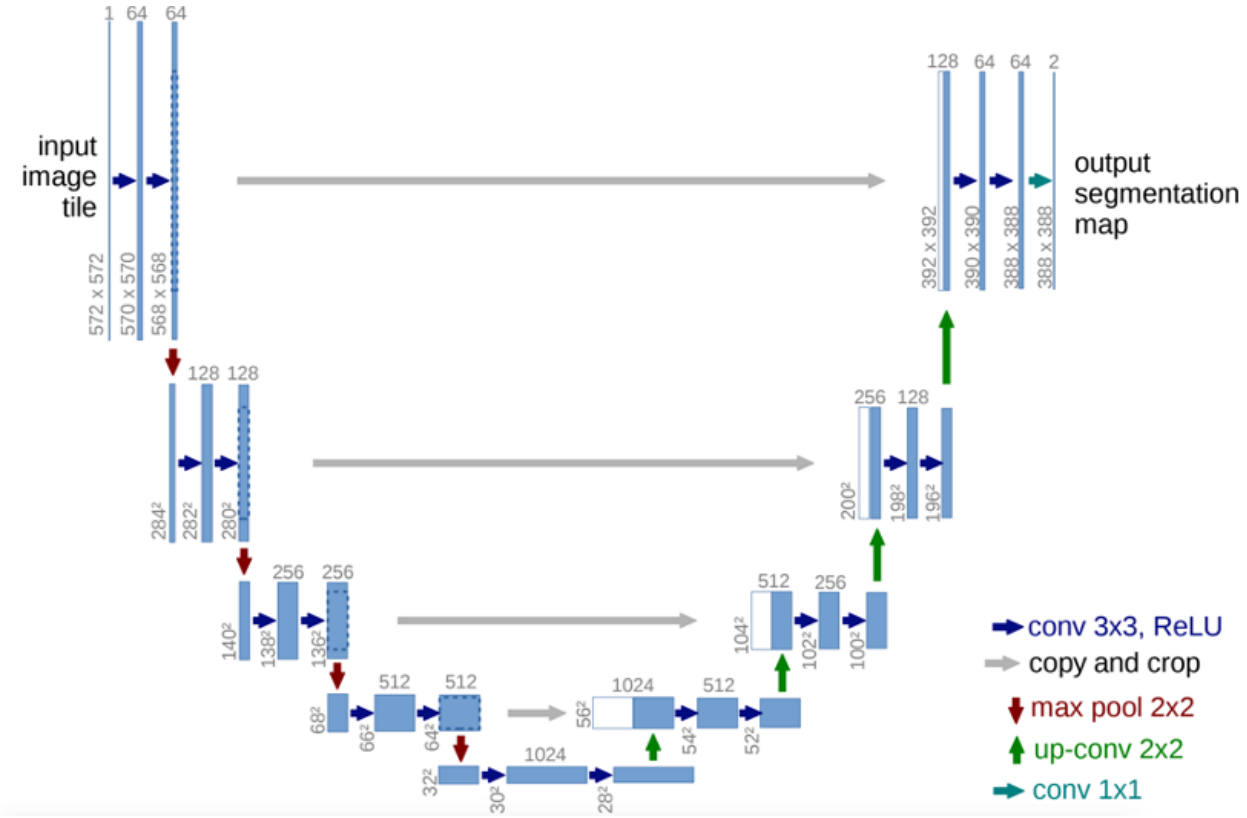


Figure 3.4. U-Net model architecture. Figure from [14]. Blue boxes represent feature maps. Top number describes the number of feature map channels, while the bottom left numbers describe the resolution. Colored arrows show the paths and operations applied at the different stages of the network. Note that each convolution layer is followed by an activation function.

3.2.7 Loss Function

The training process modifies the networks' learnable parameters (i.e convolution kernels) trough a technique named backprogration. Parameters are modified to minimize a defined loss function which compares the network output and the respective ground truth image.

After the last 1×1 convolutional layer, a pixel-wise softmax is applied to the output segmentation mask. Softmax for each pixel x, at channel k of the feature map, after the final activation is defined as

$$p_k(\mathbf{x}) = \exp\left(a_k(\mathbf{x})\right) / \left(\sum_{k'=1}^{K} \exp\left(a_{k'}(\mathbf{x})\right)\right) \tag{3.13}$$

where K is the total number of output channels. For this calculation, the number of output channels correspond to the number of classes. Therefore, at each location x, the prediction after 3.13 should ideally output a value close to 1 at channel k to predict the class corresponding to that channel, and a value close to zero at the other channels. The pixel-wise loss function is obtained by computing the cross entropy between the predicted softmax value and the ground truth value, as in 3.14. The term y_k represents the ground truth masks for each class at each channel k. The loss is then averaged between all pixels to obtain the final value.

$$L(\mathbf{x}) = -\sum_{k \in K} y_k(\mathbf{x}) \log\left(p_k(\mathbf{x})\right) \tag{3.14}$$

The original U-Net paper incorporates an additional weight term w. This term gives greater importance in the loss function to the pixels that are at the borders of the objects.

Our works build upon the implementation of [20] and the 2018 Kaggle Data Science Bowl. Likewise, the goal was to segment nuclei. The method by the best performing submission combines networks with a couple of loss implementations. Besides applying softmax activation on the output model layer combined with cross entropy loss, some experiments used as activation the logistic function defined in 3.15 for each element x of the output feature map. In contrast to softmax, the logistic function is applied to each element x individually. Thus, for each prediction at x the loss is calculated with respect to two classes that correspond to the binary values of the ground truth masks. Defined as the binary cross entropy loss, this can be seen as a specific case of 3.14 when number of classes is equal to 2. Furthermore, [20] adds a new term to the loss for both of these calculations. In addition to the cross entropy, or binary cross entropy, a soft dice loss term is added to the calculation.

$$f\left(x\right) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{3.15}$$

$$D_l = 1 - c_D \tag{3.16}$$

Where c_D is the Dice Coefficient. The coefficient measures how similar two samples are. Having to samples A and B, it is formally defined as

$$c_D = \frac{2|A \cap B|}{|A| + |B|} \tag{3.17}$$

For sets, $A \cap B$ in equation 3.17 measures the intersection between the two sets. The two sets compared in our case are the network prediction and the mask ground truth. A dice coefficient equal to one indicates full possible similarity or overlap. For matrix operations, we can represent this intersection with the element wise multiplication between two matrices. Since the ground truth mask is a binary mask, this implies that the prediction values being multiplied by the zero values in the mask will stay as zero on the output. Similarly,for values corresponding to 1 in the ground truth, the intersection operation will output the prediction values, with higher values representing higher score confidence. For the matrix equivalent of the set cardinality, we can do a sum over the values of respective matrix. Similarly to the cross entropy loss, this is computed over the output feature map channels and then averaged.

3.2.8 Using a Pre-Trained Feature Extractor

When training deep neural networks, it is common practice to use learned parameters from similar networks that have already been trained. The intuition is that several filters learned for a task in the convolutional layers, can be useful for a completely different task. Some of the most widely available pre-trained networks are the ones trained for classification. These have the advantage of being trained with publicly large datasets to predict a large number of classes. For the U-Net architecture, this loading of pre-trained parameters applies only to the contracting path. These can reduce the amount of training time needed, as well as assist in achieving optimal performance for the contracting path.

Following [20], we load pre-trained parameters from ResNet architectures trained on the Imagenet dataset to classify images into 1000 classes. The Resnet model architecture [21] was developed to allow models to stack more convolutional layers. Previous to this, model performance would decrease as the network grew deeper. The main contribution of this work was to add skip connections at residual blocks, depicted in figure. In the worst case scenario, when adding more layers, this blocks will allow a direct mapping from its input to its output, hence performance should not be affected. With this architecture deeper networks were successfully trained on Imagenet, reaching higher classifications accuracy. We utilize pre-trained parameters from Resnet architectures with different depths. This gives the contracting path in our architecture knowledge from learned filters or kernels.

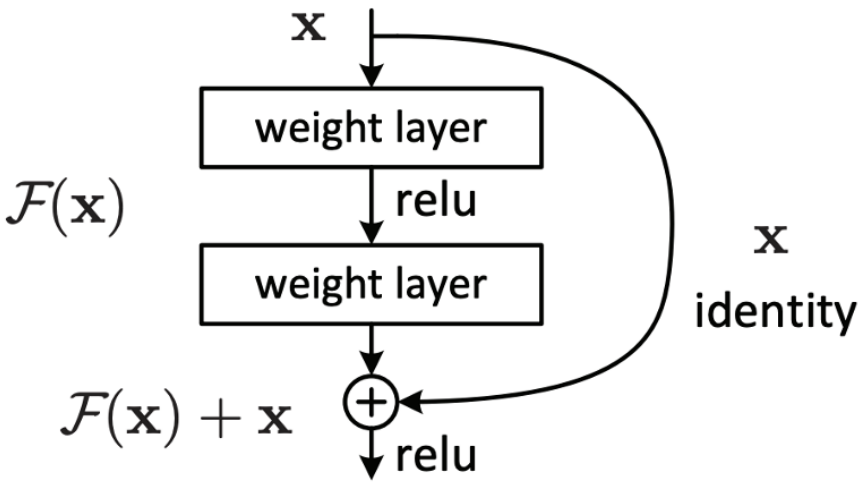


Figure 3.5. Residual block. Figure from [21]. The block input is connected to the block output directly, bypassing the intermediate convolutional layers. Stacking blocks of this type allowed networks to grow deeper without compromising performance.

4. DETECTING AND LOCALIZING NUCLEI

4.1 Predicting Nuclei Masks and Border

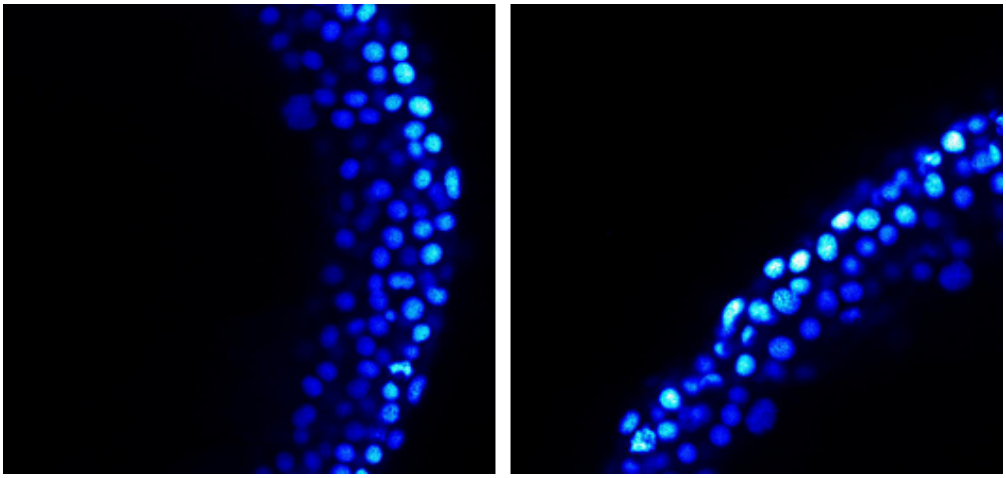
A U-Net model architecture, with a sigmoid activation after the output, is trained to predict two channels. Each channel represents an image containing the full nuclei masks and the border for each nuclei, respectively. Following a similar technique to [20], the reason for these two channels is to do a pixel-wise subs-traction between the nuclei border image and the full nuclei image. This facilitates the detection and localization of individual nuclei by creating separation of nearby and overlapping nuclei.

At each channel, the model predicts outputs between 0 and 1. At a given location x, the value will represent how confident the model is at predicting that x should be classified as part of a nuclei mask or border, respectively. We convert both output channels to gray-scale images by multiplying each value by 255. To remove noise and wrongly classified pixels, which would not have similar values in their surroundings, we apply Gaussian blur. Finally, we threshold the blurred images to obtain binary images. Figure 4.1 shows an example of these binary images from a model prediction. Finally, pixel-wise subtraction is done on the binary images. The output of this operation holds an binary image S where each nucleus mask is clearly separated from nearby nuclei, as seen in Figure 4.2.

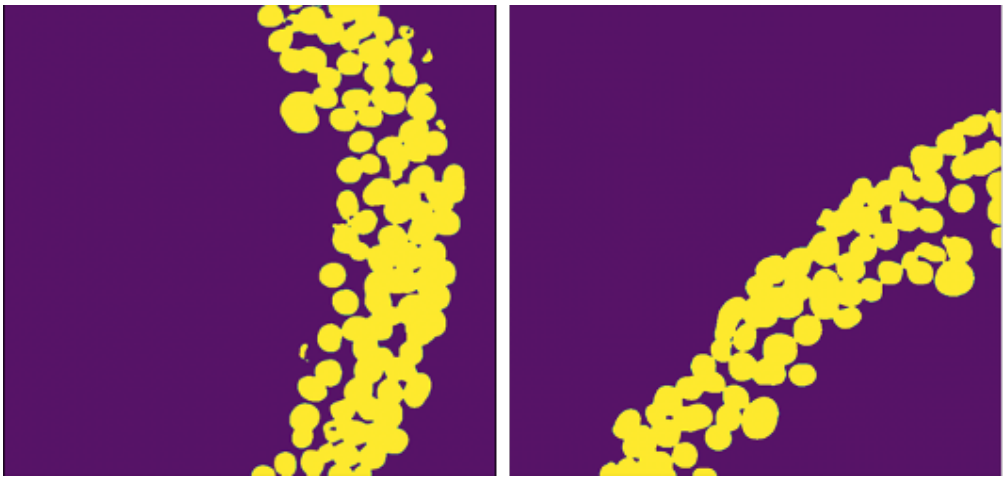
4.2 Localizing Nuclei Centers

Identifying nuclei centers and area is simplified once we obtain the separated nuclei masks in S. The most straightforward method is to use OpenCV's implementation of [15] to find each nucleus' contours. For each of the nucleus' contours we calculate its moments M_{mn}. Applying equation 4.1 we obtain the nuclei area and centroid.

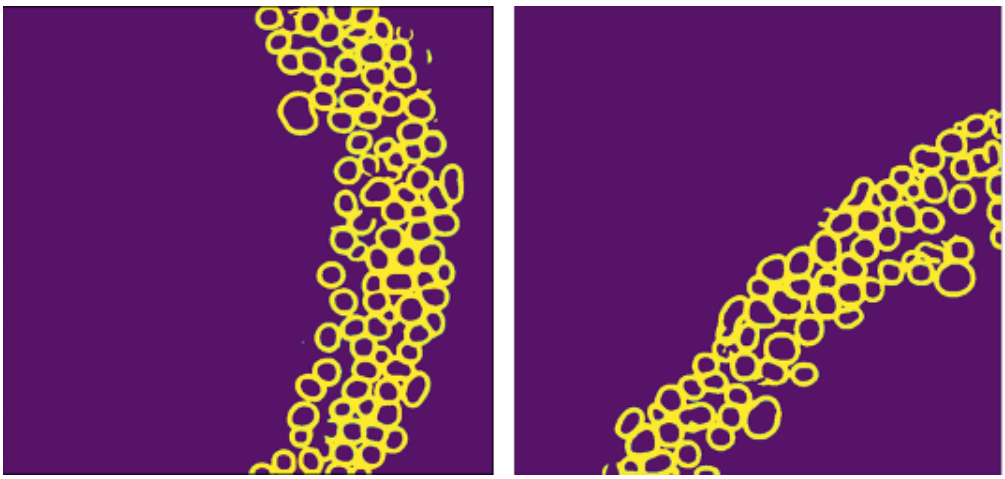
$$C_x = \frac{M_{10}}{M_{00}} \qquad C_y = \frac{M_{01}}{M_{00}} \qquad \text{Area} = M_{00} \tag{4.1}$$



(a) **Input Embryo Image Slices**



(b) **Full Nuclei Masks Channel Predictions**



(c) **Nuclei Border Masks Channel Predictions**

Figure 4.1. Network prediction. Given a couple of sample images (a), the network predicts two output channels, as seen in (b) and (c). Note that this images have been pre-processed to remove noise.

We propose a second method to obtain the nuclei area and centers. This arises from some of the specific characteristics of the embryo images. Given that the embryonic data is obtained at a time of development, some of the nuclei are in process of dividing. Moreover, some nuclei might be so close to one another that to the Amazon Turk worker they appear as the same nuclei. Hence, more than one nuclei are labeled as one instance. Similarly, during our prediction pipeline some nuclei may fail to be separated. For this reason, the proposed method attempts to identify when the nuclei shape might correspond to two nuclei, identifying centers for both.

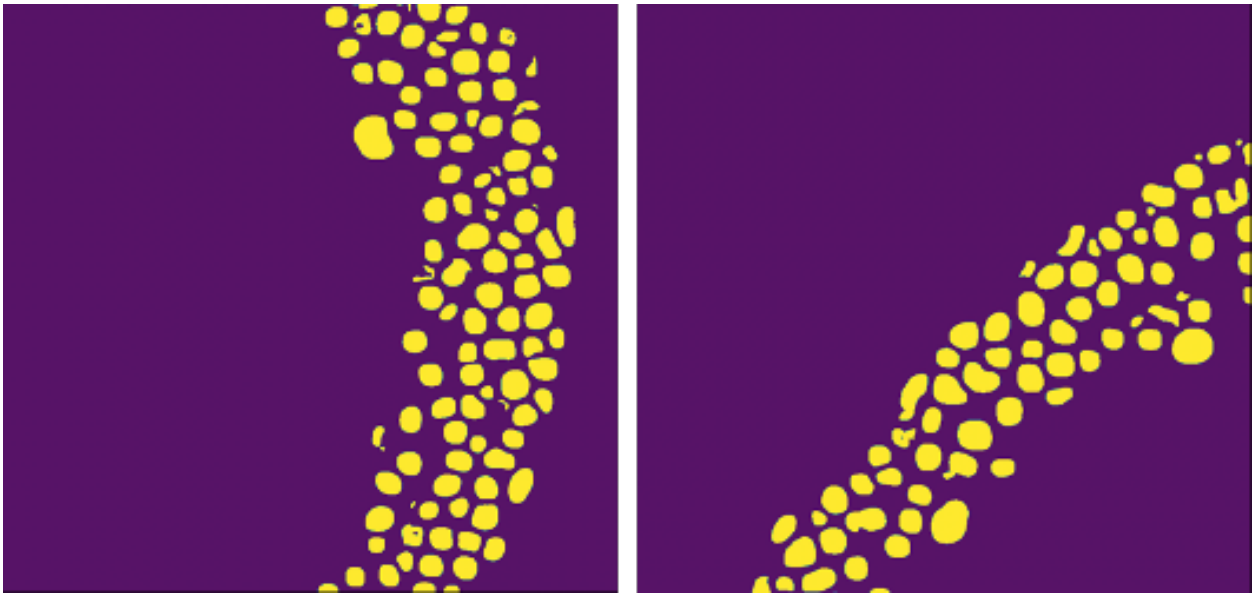


Figure 4.2. Pixel-wise subtracted result S. Compared to 4.1, nuclei are clearly separated from one another.

Given binary image S, we first calculate the euclidean distance transform $\mathrm{D_T}$ for each pixel x in this image.

$$\mathrm{D_T(x)} = \min_{\mathrm{y \in P}} d\left(x, y\right) \qquad \text{with} \quad d\left(x, y\right) = \sqrt{\sum_{\mathrm{i=1}}^{n} \left(x_{\mathrm{i}} - y_{\mathrm{i}}\right)^2} \tag{4.2}$$

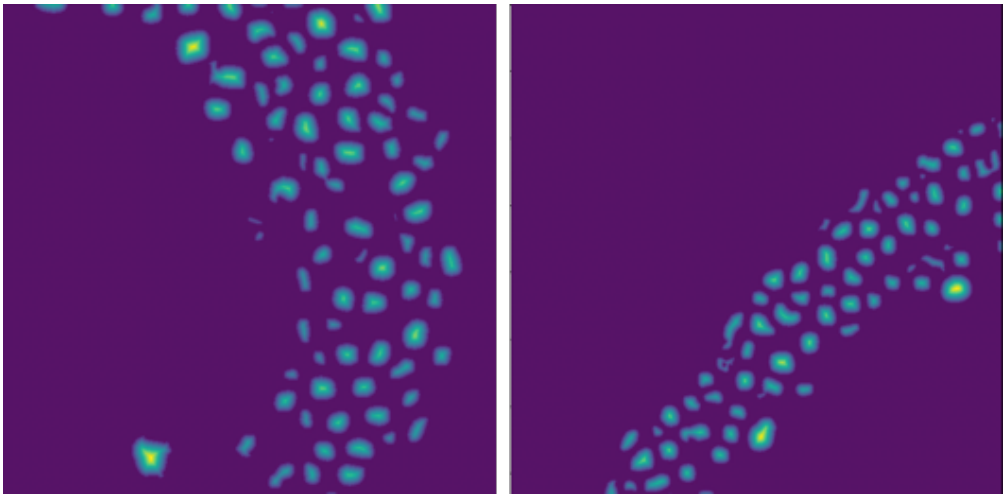
The set of points P are the points where the input pixel value is 0. Next, we obtain the local maxima in the full distance transform image D_T. To ensure that local maxima

are not too close to one another, we define a minimum distance parameter between local maxima, m_d. The local maxima locations are then each used as seeds when applying the watershed-by-flooding algorithm to a negative version of the distance transform, $-D_T$.

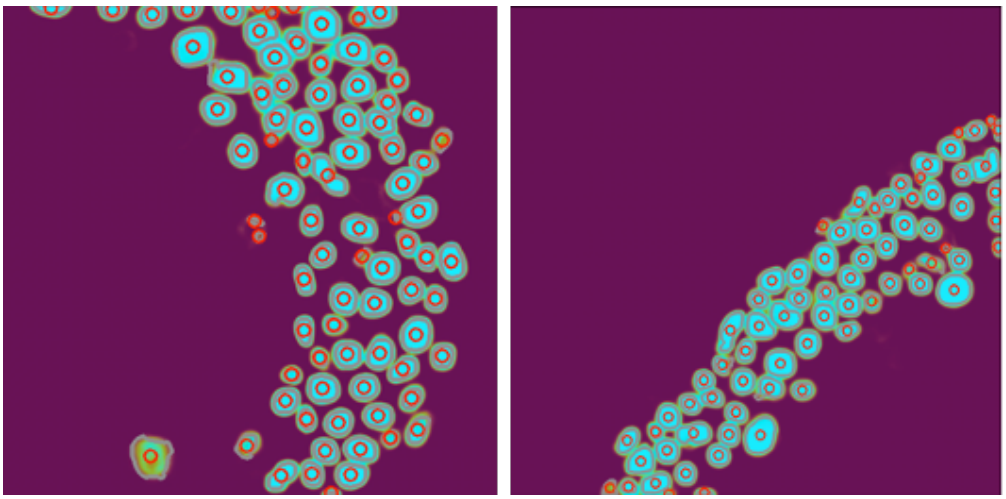
The watershed algorithm is based on the premise that the pixel intensities can be interpreted as heights if the image was a topographic map. Continuing with the topography analogy, the areas with lower pixel intensity are considered basins, while the higher value areas are seen as hills or ridges. The watershed algorithm proposes to flood the basins. Wherever water from different basins would come in contact with each other, a line is drawn to prevent this. These drawn lines provide the segmentation of the homogeneous areas of the original image. To deal with irregularities in the input images, a variant of the algorithm where markers indicate the point from where to start the flooding is used. Specifically, the maxima locations are used as markers to segment $-D_T$.

Having obtained the segmented nuclei, we then proceed to find the contours and centers for each segmented object utilizing [15] and 4.1. Intuitively, by giving markers based on the distance transform's maxima, we are helping segment nuclei that might still be overlapping but where the common circular shape of nuclei is not present. Clearly, even if the segmentation was applied to the distance transform, the obtained nuclei locations apply to image S and therefore to the original input.

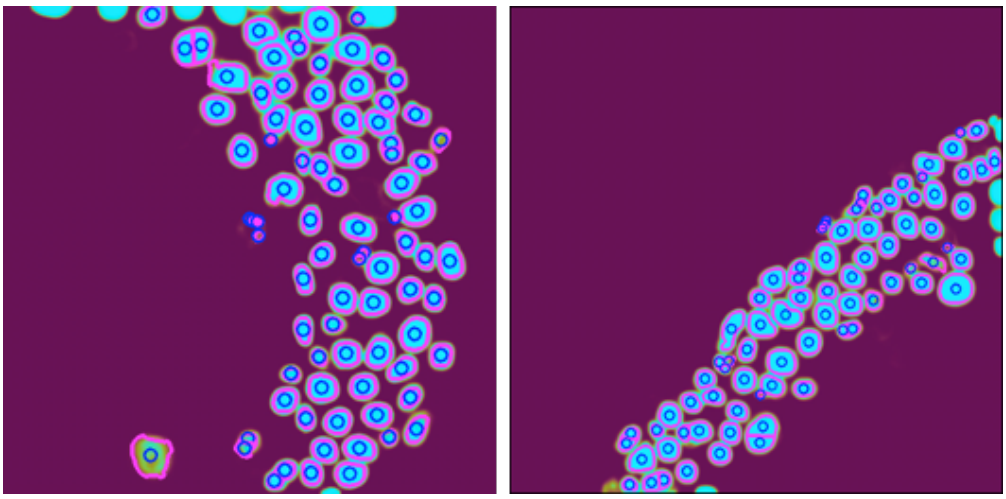
Note how the examples on Figure 4.3 for the first method identify a center even when it appears the nuclei is dividing or appear as overlapping nuclei. The second proposed method does a better job of identifying two centers for these cases. However, the latter suffers from over-identification at certain locations.



(a) **Distance Transform Outputs**



(b) **Nuclei Contours and Centroids with [15]**



(c) **Nuclei Contours and Centroids with D_T and Watershed Segmentation**

Figure 4.3. Nuclei Detection and Localization using the two proposed methods.

5. RESULTS

5.1 Nuclei Segmentation Results

We first present results that compare our proposed U-Net segmentation with the Wavelet method proposed by [11]. For this, we make use of only the full nuclei masks channel from the network output.

We utilize metrics that compare pixel-wise classification. Both the ground truth data, obtained from Amazon Turk workers' labelling, and the predictions are presented as binary images. The performed analysis give an insight over how similar the pixels predicting nuclei locations are to the real nuclei locations. The utilized metrics are :

1. Jaccard Index: Also commonly know as Intersection Over Union (IoU) score. Quantifies the overlap between prediction mask and ground truth mask. In our case we only measure the IoU over the pixels representing nuclei (i.e pixels with value 1 in the binary image).

$$IoU = \frac{\text{target } \cap \text{prediction}}{\text{target } \cup \text{prediction}} \tag{5.1}$$

2. Precision: Measures the performance of the prediction of not classifying as positive a negative sample. In this work, positive samples correspond to nuclei pixels.

$$\text{Precision } = \frac{TP}{TP + FP} \tag{5.2}$$

3. Recall: Measure the ability of the prediction to find all positive samples.

$$\text{Recall } = \frac{TP}{TP + FN} \tag{5.3}$$

4. F1 Score: For binary data it is equivalent to the dice coefficient defined in equation 3.17.

$$\text{F1} = \frac{2 * (Precision * Recall)}{(Precision + Recall)} \tag{5.4}$$

where:

TP = True Positives. A positive prediction corresponding to a positive ground truth value.

FP =False Positives. The predictions gives a positive value but the ground truth does not.

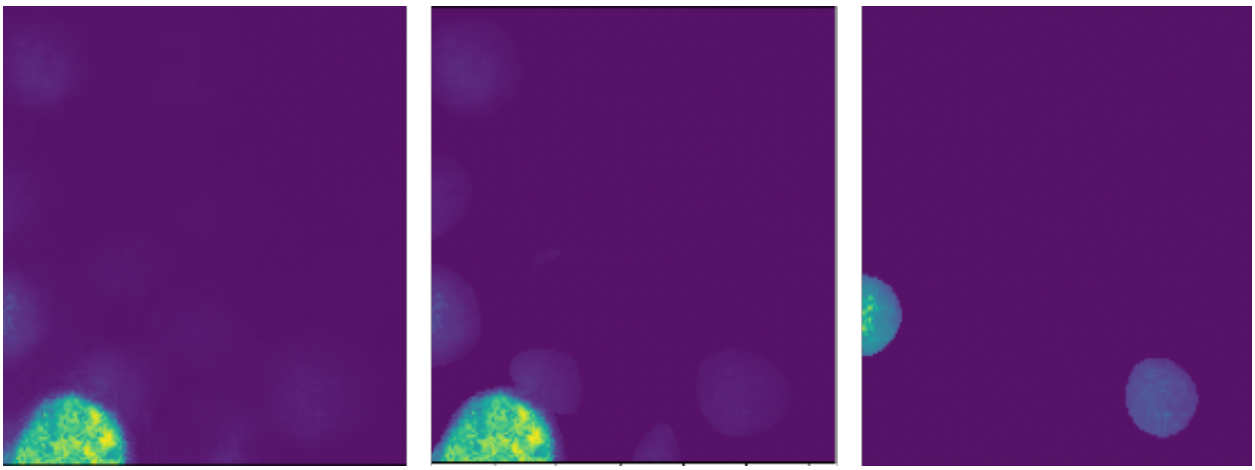
FN = False Negative. The predictions gives a negative value but ground truth shows as positive.

We produce results over a set of 150 sliced embryo images. Metrics are obtained for each image. Results in Table 5.1 correspond to statistics of the metrics over all the images.

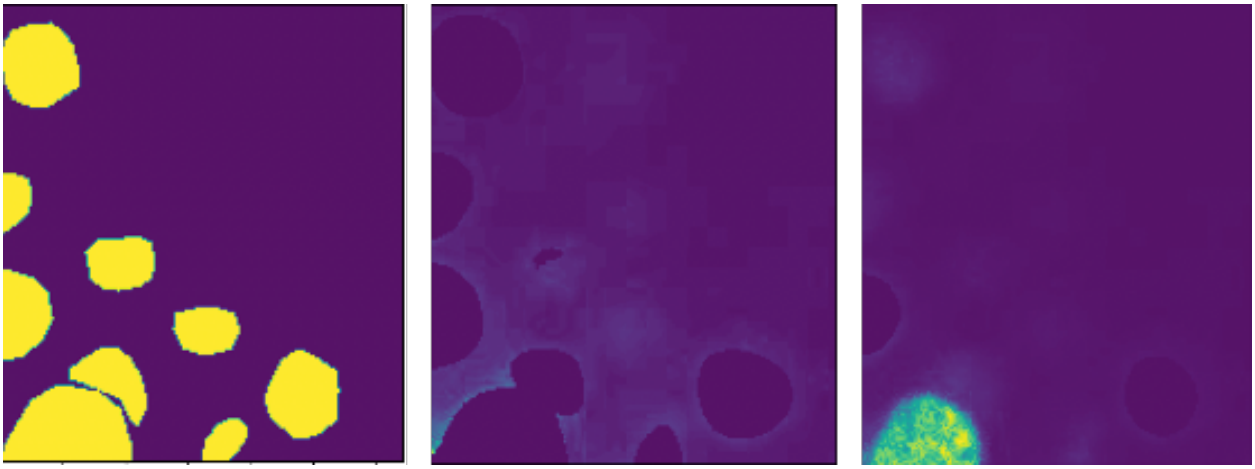
Table 5.1. Segmentation metric comparison between the U-Net model and the Wavelet method.

IoU	**Wavelet Segmentation Method**	**U-Net Segmentation**
Mean	0.516	0.731
Standard Deviation	0.127	0.167
Median	0.535	0.776
Maximum	0.739	0.924
Precision	**Wavelet Segmentation Method**	**U-Net Segmentation**
Mean	0.905	0.841
Standard Deviation	0.136	0.132
Median	0.942	0.876
Maximum	1.00	0.992
Recall	**Wavelet Segmentation Method**	**U-Net Segmentation**
Mean	0.550	0.835
Standard Deviation	0.152	0.166
Median	0.556	0.889
Maximum	0.855	0.974
F1 Score	**Wavelet Segmentation Method**	**U-Net Segmentation**
Mean	0.670	0.830
Standard Deviation	0.143	0.131
Median	0.697	0.874
Maximum	0.850	0.961

From the results we conclude that, by looking at the Jaccard Index and F1 Score, the trained model outputs segmentation masks that tend to be more similar pixel for pixel than the outputs by the wavelet method. An important note is that the wavelet segmentation was not trained on any data. Therefore its output, while similar in the sense of looking to segment nuclei, do have other considerations. The wavelet method tends to be safer in its predictions, explaining its higher Precision. However, judging by its Recall, it misses pixels that should be labeled as nuclei. The F1 score proves that the U-Net prediction provides a better overall balance.



(a) **Original Image and Positive Masks from U-Net and Wavelet, respectively**



(b) **Ground Truth and Negative Masks from U-Net and Wavelet, respectively**

Figure 5.1. Lower Intensity Nuclei Comparison.

We also provide a visual comparison from the outputs from both models. Figure shows samples from this analysis. Using the technique introduce in Chapter 2, in Figure 5.4 we mask the original image with the mask outputs from the compared methods. It can be appreciated how the wavelet method misses some very clear nuclei. Since the numerical analysis is done at the pixel level, even small misses which sometimes are not clearly seen produce low scores. An example of this behavior can be seen on Figure 5.1 . The wavelet method struggles with lower intensity nuclei. However, both the U-Net model prediction and the Turk labels provide masks for this nucleus type.

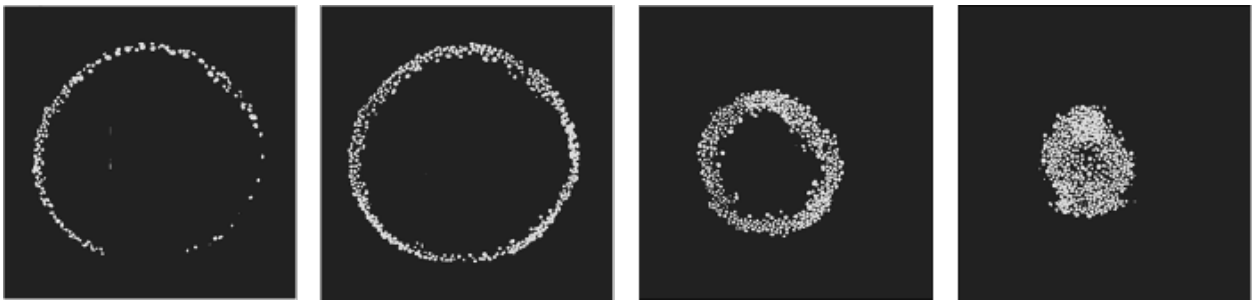


Figure 5.2. Reconstructed 2d embryo slices from model prediction at different depths.

We can reconstruct the mask for each 2D embryo slice from the saved metadata when preparing the images for Amazon Turk, as seen in Figure 5.2 . This in turn allows us to visualize a 3D representation of all the nuclei present in an embryo. Figure 5.3 displays screenshots of an interactive 3D visualization from the nuclei masks using the open source 3D Slicer software platform [22].

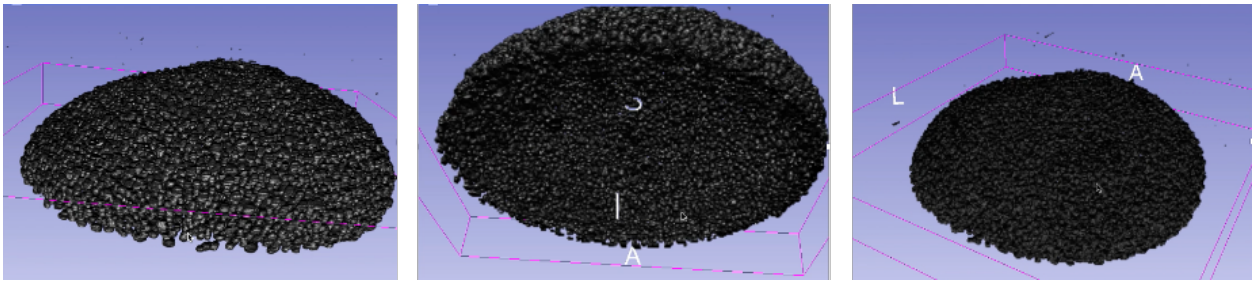
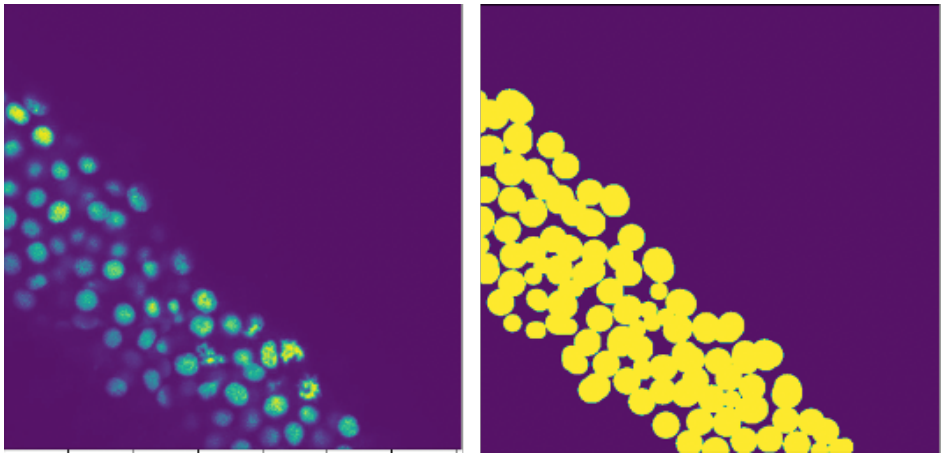
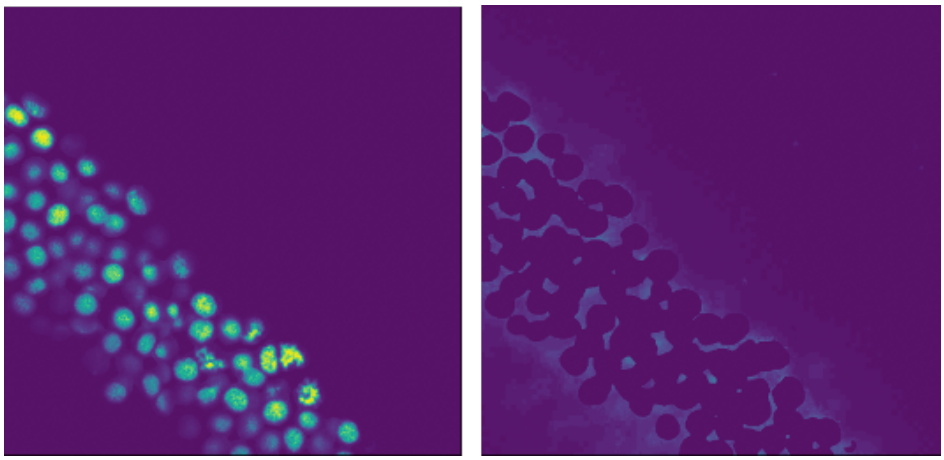


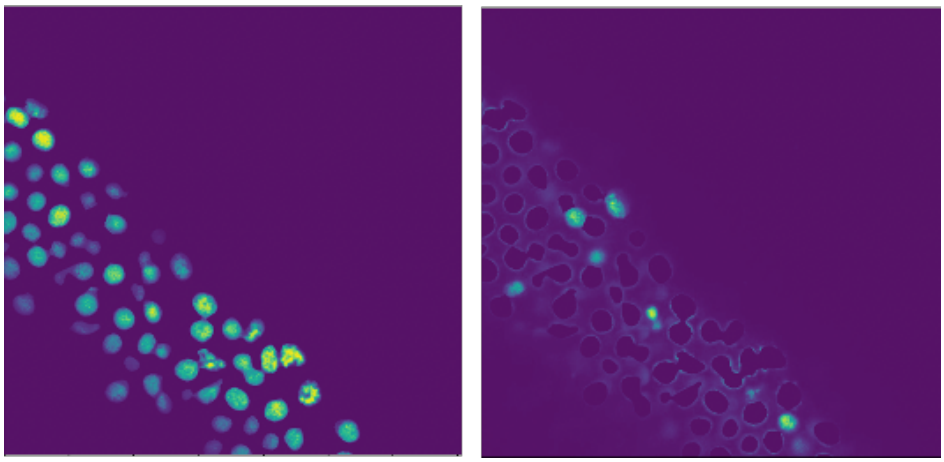
Figure 5.3. 3D visualization screenshots of predicted nuclei present in an embryo.



(a) **Original Embryo Slice and Ground Truth Mask**



(b) **Positive and Negative Mask from to U-Net Prediction**



(c) **Positive and Negative Mask from to Wavelet Prediction**

Figure 5.4. Results Comparing Masks from U-Net and Wavelet Methods.

6. CONCLUSION

6.1 Conclusion

We have developed and end to end pipeline to detect and localize nuclei in microscopic Zebrafish embryo slices. Compared to [11], our method provides more precise detection and segmentation of nuclei. We achieve this by pre-processing the data so that it can be uploaded to Amazon Turk, an online tool where workers label data, therefore reducing the time needed to create trustworthy ground truth data. Users of this pipeline can decide to accept or reject the labelling via a GUI, which completes the data preparation process in a way that can be easily reproduced as new microscopic data becomes available.

From this data we have trained a segmentation convolutional neural network. Specifically we have used the U-Net model architecture. Building upon techniques from [20], our model trains and generalizes well for segmentation of embryo images at various depths, hence containing nuclei of different sizes and intensities. Furthermore, its output allows for nearby and overlapping nuclei to be separated. The next step in the pipeline takes this information and detects individual nucleus's centers and areas. The detection and localization steps are considerably faster than previous methods.

These characteristics, of both precision and efficiency, make our pipeline a useful tool for research on these types of imaging. Additionally, our end to end pipeline facilitates the work to generate data to train new models for the study of other types of organisms.

6.2 Future Work

In the future, efforts will be focused on creating a pipeline to generate labelling data, train and predict other types of cells or proteins present in Microscopy Imaging. Specifically, being able to detect and track mRNA over the course of the embryo's development would provide valuable insights towards developmental biology.

A great challenge of detecting other types of molecules with machine learning is the ability to generate training data. For mRNA, for example, labelling the images with the same procedure as the nuclei presented in this work would be extremely time consuming,

due to the very small pixel sizes and amount of mRNA present in an embryo. Previous works, such as [23], remove the dependency on bounding boxes or pixel annotation for the ground truth data. Instead, they utilize simple points as annotations and modify their loss functions to localize with only this type of label. Combining this type of work with our labeling pipeline would reduce the amount of time needed to produce enough data to train a well performing model. This would in turn open the door to detect and track some more traditionally challenging molecules.

REFERENCES

[1] F. R. Khan and S. Alhewairini, "Zebrafish (danio rerio) as a model organism," in. Nov. 2018.

[2] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, pp. 62–66, 1979.

[3] H. J. Vala and A. Baxi, "A review on otsu image segmentation algorithm," 2013.

[4] H. Cai, Z. Yang, X. Cao, W. Xia, and X. Xu, "A new iterative triclass thresholding technique in image segmentation," *IEEE Transactions on Image Processing*, vol. 23, pp. 1038–1046, 2014.

[5] L. Vincent and P. Soille, "Watersheds in digital spaces: An efficient algorithm based on immersion simulations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 6, pp. 583–598, Jun. 1991, ISSN: 0162-8828.

[6] S. Beucher and C. D. M. Mathmatique, "The watershed transformation applied to image segmentation," in *Scanning Microscopy International*, 1991, pp. 299–314.

[7] M. Ambühl, C. Brepsant, J.-J. Meister, A. Verkhovsky, and I. Sbalzarini, "High-resolution cell outline segmentation and tracking from phase-contrast microscopy images," *Journal of microscopy*, vol. 245, pp. 161–70, Feb. 2012.

[8] P. R. Gudla, K. Nandy, J. Collins, K. Meaburn, T. Misteli, and S. Lockett, "A high-throughput system for segmenting nuclei using multiscale techniques," *Cytometry. Part A : the journal of the International Society for Analytical Cytology*, vol. 73, no. 5, pp. 451–466, May 2008, ISSN: 1552-4922.

[9] H. Irshad, A. Veillard, L. Roux, and D. Racoceanu, "Methods for nuclei detection, segmentation, and classification in digital histopathology: A review—current status and future potential," *IEEE Reviews in Biomedical Engineering*, vol. 7, pp. 97–114, 2014.

[10] T.-C. Wu, X. Wang, L. Li, Y. Bu, and D. M. Umulis, "Waveletseg: Automatic wavelet-based 3d nuclei segmentation and analysis for multicellular embryo quantification," *bioRxiv*, 2020.

[11] T.-C. Wu, "Waveletseg: A novel automatically wavelet-based 3d nuclei segmentation method and analysis platform in embryonic imaging.," Doctor of Philosophy in Agricultural and Biological Engineering [Unpublished Doctor of Philosophy Dissertation]Purdue University. 2019.

[12] W. Hoff, *Introduction to wavelets in image processing*, 2015.

[13] F. Xing, Y. Xie, H. Su, F. Liu, and L. Yang, "Deep learning in microscopy image analysis: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, pp. 1–19, Nov. 2017.

[14] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pp. 234–241, 2015, ISSN: 1611-3349.

[15] S. Suzuki and K. be, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, 1985, ISSN: 0734-189X.

[16] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[17] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *Lecture Notes in Computer Science*, pp. 818–833, 2014, ISSN: 1611-3349.

[18] M. Lin, Q. Chen, and S. Yan, *Network in network*, 2013.

[19] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015.

[20] J. Caicedo, A. Goodman, K. Karhohs, B. Cimini, J. Ackerman, M. Haghighi, C. Heng, T. Becker, M. Doan, C. McQuin, M. H. Rohban, S. Singh, and A. Carpenter, "Nucleus segmentation across imaging experiments: The 2018 data science bowl," *Nature Methods*, vol. 16, Dec. 2019.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[22] R. Kikinis, S. D. Pieper, and K. G. Vosburgh, "3d slicer: A platform for subject-specific image analysis, visualization, and clinical support," in *Intraoperative Imaging and Image-Guided Therapy*, F. A. Jolesz, Ed. New York, NY: Springer New York, 2014, pp. 277–289, ISBN: 978-1-4614-7657-3.

[23] J. Ribera, D. Güera, Y. Chen, and E. J. Delp, *Locating objects without bounding boxes*, 2019.