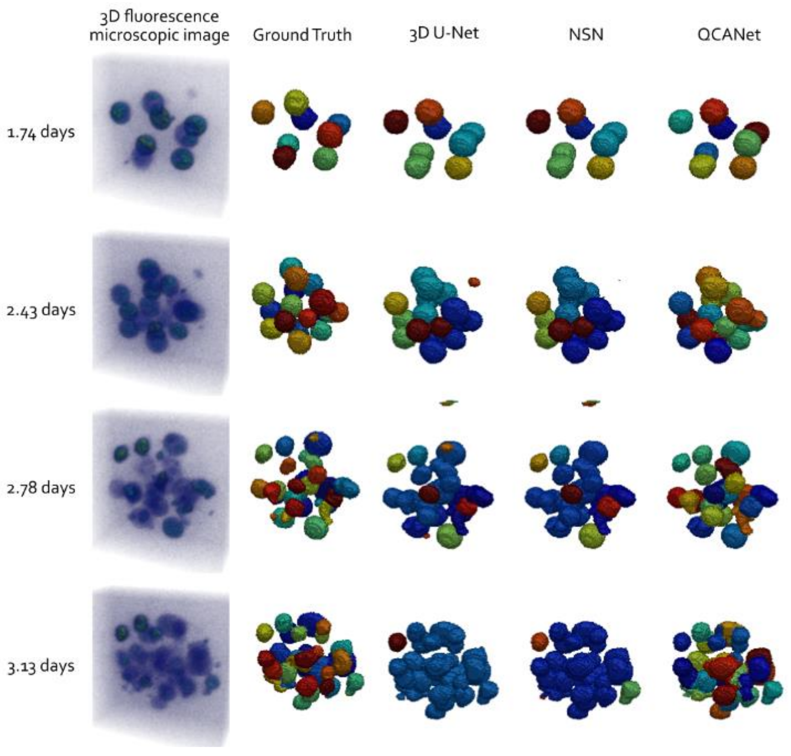
Reproduction of:

3D convolutional neural networks-based segmentation to acquire quantitative criteria of the nucleus during mouse embryogenesis (Figure 3)



By:

Hil Steketee (5072166) → Watershed algorithm

Max Koch (4725190) → 3D U-Net

Hanna den Hertog (5146461) → NDN network

Rozemarijn Ammerlaan (4830571) → NSN network

3D convolutional neural networks-based segmentation to acquire quantitative criteria of the nucleus during mouse embryogenesis

Replicated by Hanna den Hertog (5146461), Hil Steketee (5072166), Max Koch (4725190) and Rozemarijn Ammerlaan (4830571)

Code available at: https://github.com/momkoch3/QCA-net-Reproduction- (https://github.com/momkoch3/QCA-net-Reproduction-)

Introduction

Embryogenesis is the phase with rapid cell divisions in the weeks immediately after fertilisation. It is a highly dynamic process during which cells grow and move in 3D space. The researchers of this paper used a confocal microscope to obtain a time-series of 3D fluorescence images of mouse embryos.

The authors propose their CNN-based Quantitative Criteria Acquisition Network (QCANet) to perform instance segmentation of the individual cells. With instance segmentation, a different label is added to each cell to facilitate quantitative analysis. This contrasts with semantic segmentation, which gives the same label to all cells. This is illustrated in the figure below. In the first row, both methods succeed in segmenting the cells. In the second row, when the cells are no longer sufficiently separated, semantic segmentation fuses overlapping cells into a single object.

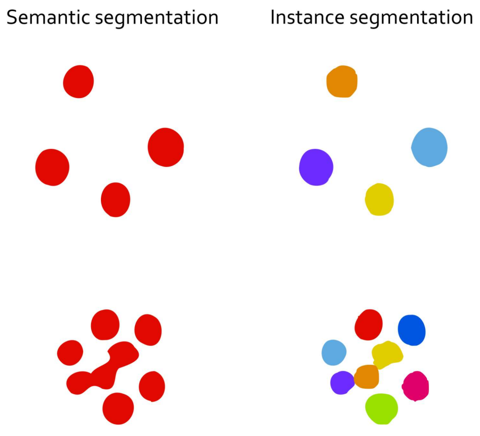


Fig 1: Schematic that shows the concepts of semantic and instance segmentation.

QCANet uses two subnetworks, Nuclear Segmentation Network (NSN) and Nuclear Detection Network (NDN), and combines the results of the two networks during post-processing (see the figure below). NSN segments the "nuclear regions" (e.g. cells).

NDN identifies the nuclei, which lie within each cell. During the post-processing step, the different regions identified by NSN are divided through a watershed algorithm. This watershed uses the nuclei that were identified by NDN as the marker positions.

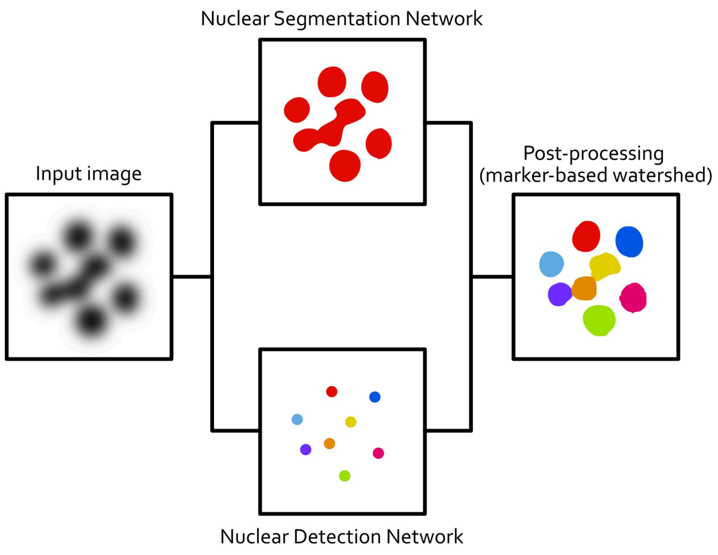


Fig 2: Flow diagram that shows the QCANet algorithm. The NSN and NDN algorithms are executed in parallel, their outputs are combined in post-processing to give the QCANet output.

This blogpost describes our reimplementation of the algorithms in this paper to reproduce figure 3 of the original paper. The next section gives an overview of the models. After that, we will show and compare our results. Finally, the discussion will comment on the reproducibility of the paper.

Models

3D U-Net

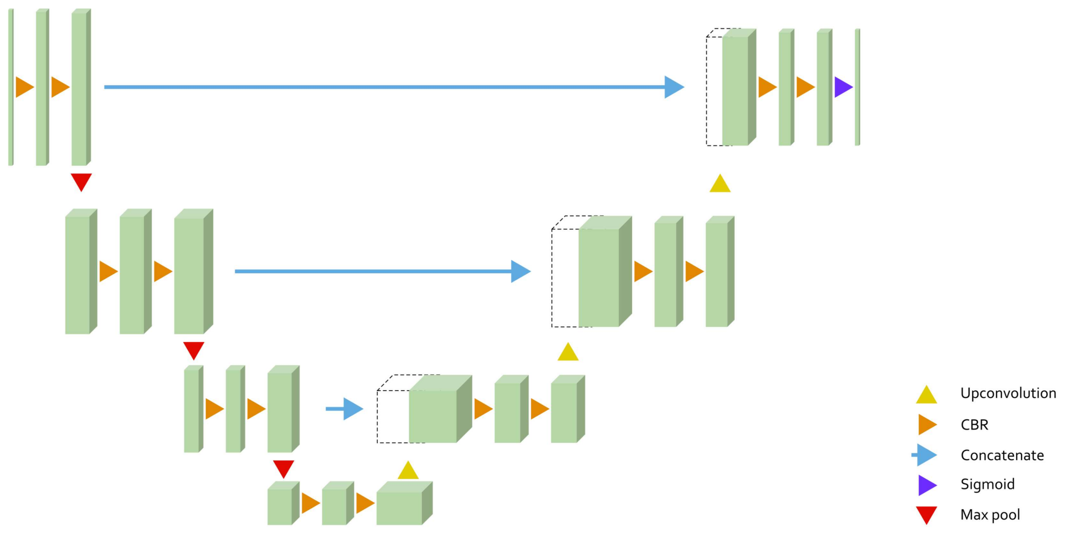


Fig 3: Flow diagram that shows the architecture of the 3D U-Net.

The authors used the architecture of the 3D U-Net from the paper (https://arxiv.org/abs/1606.06650) "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation" by Ö. Çiçek et al. The image above illustrates the architecture. CBR layers consist of a convolution, a Batch Normalisation and a ReLU activation. This network has previously been applied for the segmentation of other bioimages with satisfactory accuracy. A major short-coming of the 3D U-Net is that neighbouring regions are fused, which limits its application.

The 3D U-Net has 19,073,665 trainable parameters.

Nuclear Segmentation Network

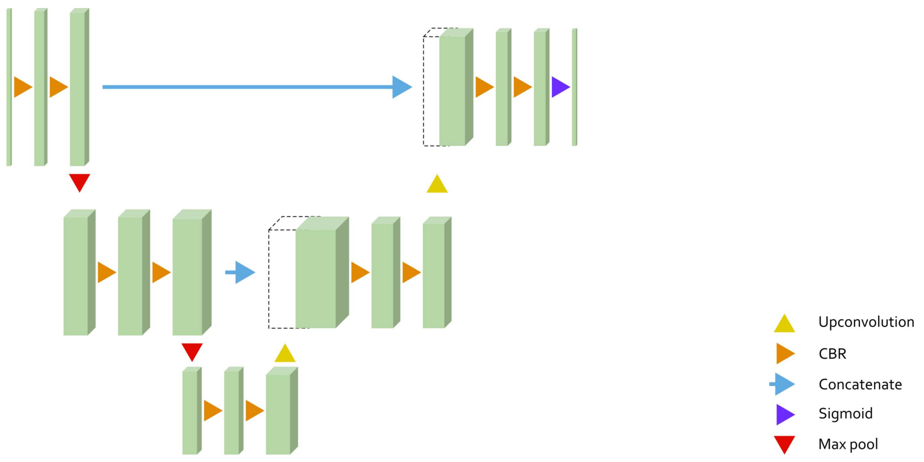


Fig 4: Flow diagram that shows the architecture of the Nuclear Segmentation Network.

The NSN is based on the 3D U-Net. The architecture and corresponding hyperparameters were provided in table 6 of the supplemental material (https://static-content.springer.com/esm/art%3A10.1038%2Fs41540-020-00152-8/MediaObjects/41540_2020_152_MOESM1_ESM.pdf) of the paper. As can be seen in the figure, NSN is more shallow than 3D U-Net. The convolutions are performed using a $3 \times 3 \times 3$ kernel, paired with a padding of size 1 along all borders.

The Nuclear Segmentation Network has 1,147,585 trainable parameters.

Nuclear Detection Network

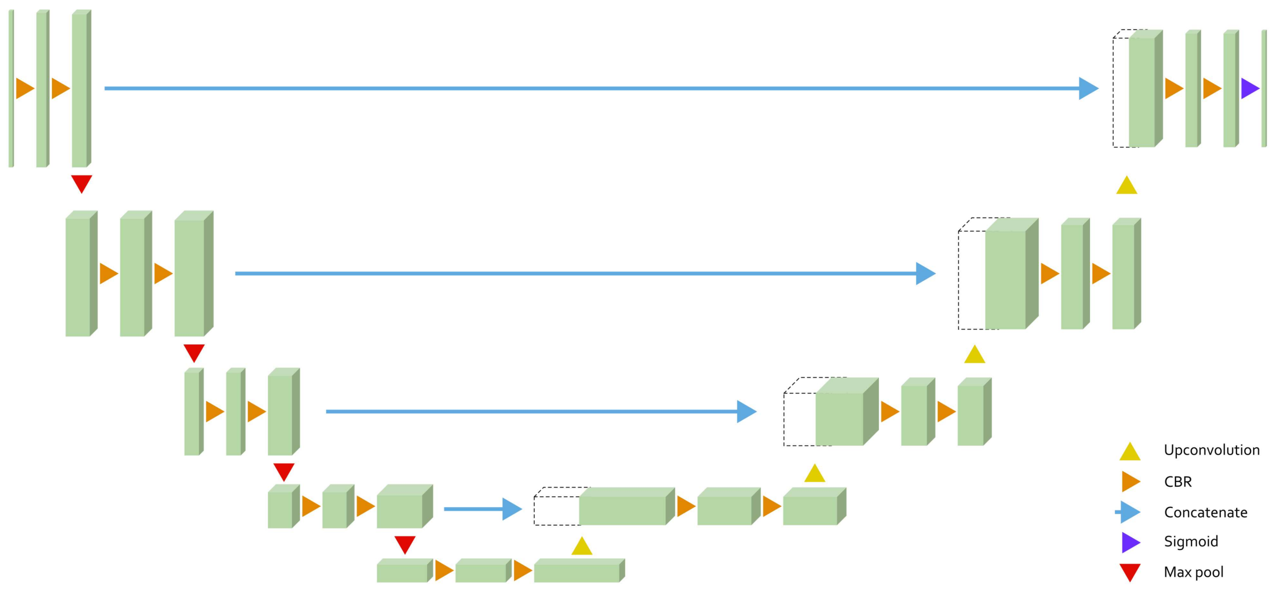


Fig 5: Flow diagram that shows the architecture of the Nuclear Detection Network.

The NDN is also derived from 3D U-Net, but this time more layers are added. Moreover, the size of the convolution window is increased to $5 \times 5 \times 5$, with a padding of 2. The precise architecture of NDN can be found in table 7 in the supplement (https://static-content.springer.com/esm/art%3A10.1038%2Fs41540-020-00152-8/MediaObjects/41540_2020_152_MOESM1_ESM.pdf). Please note that the authors made a mistake in the table, $p = 1$ should be replaced with $p = 2$.

The Nuclear Segmentation Network has 44,450,473 trainable parameters.

Data

The images and ground truth data can be downloaded from the webpage of the Broad Bioimage Benchmark Collection (https://bbbc.broadinstitute.org/BBBC050).

Images

The dataset comprises fluorescently labelled microscopic 3D images of eleven early-stage mouse embryos. For each embryo, a time-series of 502 images was taken, where the temporal resolution (i.e. time between each frame) is 10 minutes. Eleven time-points were selected by the authors, resulting in eleven 3D images per embryo.

Ground Truth

The authors of this paper manually created ground truth instance segmentations for eleven time-points for each of the eleven embryos. In this manner, a training set was created with the segmented nuclei (for training NDN) and with segmented cells (for training NSN).

Test data

In order to objectively evaluate the general performance of the different networks, a separate test set and ground truth was created. The settings of the microscope differed for this measured test data, allowing better judgment of the generalization of the networks. The figure below gives an example of an input-ground truth pair of the test set for a given slice in the z-dimension

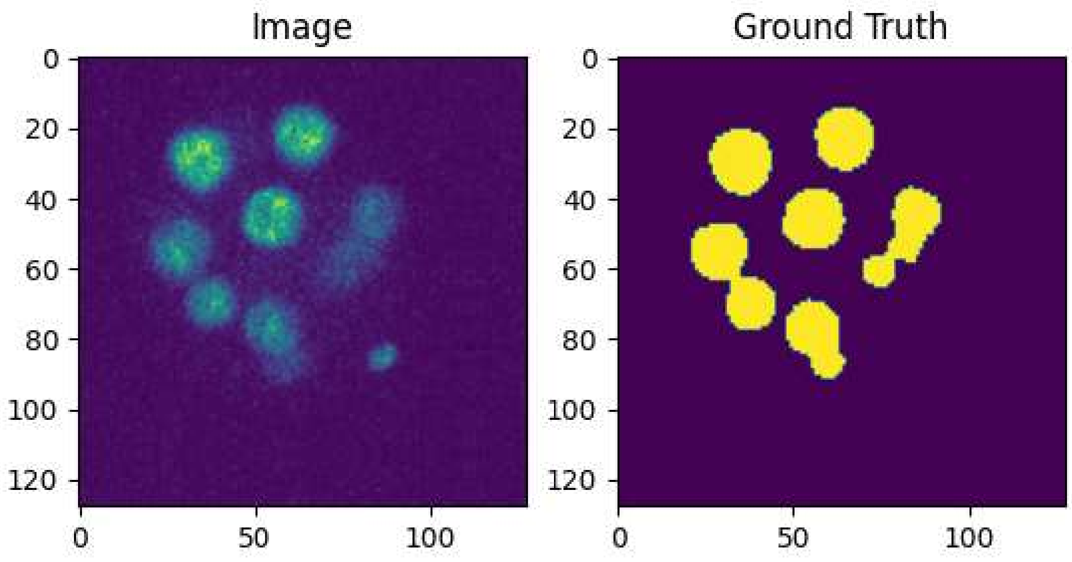


Fig 6: Example of the original image from the dataset (left) and the ground truth (right).

Data Augmentation

Because the training set was limited, the authors decided to increase its size by flipping the training data on the x-, y- and xy-axis (or not flip it), to increase the training data to four times its original size. This resulted in a total training data size of 484 (11 embryos $\times$ 11 time point images $\times$ 4 flipping options). This training data was subsequently split into a training set and validation set, where one embryo was used as validation. Therefore the actual training size was 440.

Methods

For this reproduction, we followed the methods described in the paper (https://www.nature.com/articles/s41540-020-00152-8). The authors of the paper did not make every step of their implementation explicit. This section of the blog covers the steps we took during our reproduction that were not addressed in the paper.

Pre-processing

Before the aforementioned networks could be trained, the data had to be pre-processed to a size of 128 $\times$ 128 $\times$ 128. A part of this resizing was an interpolation of the z-axis, which had to be performed due to the resolution difference between the different axis of the microsope, see 'Spatial resolution' entries in Supplementary

Tables 1 and 2 of the supplemental material (https://static-content.springer.com/esm/art%3A10.1038%2Fs41540-020-00152-8/MediaObjects/41540_2020_152_MOESM1_ESM.pdf). For example, the spatial resolutions of the train data were 0.8:0.8:1.75 for the x- y- and z-axis respectively. The z-axis was then interpolated with a factor 1.75/0.8 = 2,1875.

After interpolating, the resulting shape was brought to $128 \times 128 \times 128$ by applying mirror padding. Since the width and height of some of the images of the training data could already exceed 128 (this differed per embryo), every image dimension was padded at one extreme (for example at the beginning or the end) with the maximum 'to-be-padded' value of any of the dimensions and then the first 128 values of every dimension were selected. Note that this method implicates that part of the images can be cut off. This can be harmless, but if a cell is at the border of the image, this could mean that this cell and its information is lost. However, since flipping on the axes was applied, this same cell could still be retained in these other flipped images. This is why we went ahead with this simple strategy.

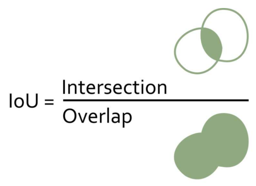
Finally, the images were normalized using the following formula, ensuring the values of the images were between 0 and 1:

$$I' = \frac{I - I_{min}}{I_{max} - I_{min}}$$

A custom Torch Dataset class was defined that performed the pre-processing steps mentioned above. Using this Dataset class, three datasets were created: the training set (containing the data of 10 embryos), the validation set (data of embryo 11) and the test set (containing data of 4 new embryos). These datasets were subsequently fed into three separate Torch Dataloader objects, which were used during training to supply our network with training data and paired ground truth data. The networks were trained on the training dataset and validated every epoch on the validation dataset. Ultimately, the network at the epoch with the highest Intersection over Union (IoU) was selected and saved and tested on the test dataset post-training (just like the authors did).

The IoU-score is a metric often used in semantic segmentation tasks and compares the amount of true positives (is a voxel/pixel correctly classified)(=Intersection) to the total amount of true positives, false positives and false negatives (=Overlap):

$$IoU = \frac{TP}{TP + FP + FN}$$



Note that the output/prediction of the networks give probabilities for every voxel of belonging to either the background or the cell (or cell nucleus in case of the Nuclear Detection Network). To calculate the IoU between the output/prediction and the ground truth (which consists of zeros and ones), the probability values had to be rounded off or set to either zero or one.

Training the networks

All the network architectures were defined and trained using the PyTorch library. The training itself was performed using Kaggle (https://www.kaggle.com/) notebooks, which is an online platform that offers online computing with 30 hours of free GPU-usage per week. The GPU we selected from the list of options was the Nvidia Tesla P100 (16GB). Since notebooks had a running limit of 12 hours, we made sure to create checkpoints every epoch using the Save function from Torch. The network state dictionary, optimizer state dictionary, losses and (IoU) accuracy scores of the train and validation data were saved every epoch.

3D U-Net

The original 3D U-net paper (https://arxiv.org/abs/1606.06650) of Çiçek et al. was applied to a 5-fold classification problem and therefore had 5 output channels, an output Softmax function and used a Cross entropy loss function. Since in our case the images only had to be classified to be a cell or background, our output channels were equal to two in the case of Softmax. Since this was a binary classification problem we mainly made use of the Sigmoid activation function (with number of output channels equal to one), which is equal to making use of a Softmax output function in the binary case, but is easier to implement. To check whether this made a difference in results, we trained the 3D U-Net using either a Sigmoid or SoftMax output-function and compared the results. In the case of a Sigmoid output function, we rounded of the predictions using a threshold of 0.5. In the case of the Softmax output function, we used the Argmax function of Torch between the two output channels, where one channel stood for the background class probability and the other channel for the cell class probability. Additionally, since the authors use the DICE-loss function for NSN, we also tested using a DICE-based objective/loss function instead of a BCE-loss, since this allows for a more equal comparison. Although the authors of the paper noted that they trained their 3D U-Net for 150 epochs before it converged, we observed a much quicker convergence around approximately 10 epochs and therefore set the total amount of epochs to 15 for 3D U-Net.

The hyperparameters used:

Loss function: Binary Cross Entropy (BCE) OR DICE-loss
Optimizer: Adam
Learning rate: 0.0001
Weight decay: None

Momentum Decay rate estimates (beta1,beta2): (0.9, 0.999) (=standard Torch ADAM values)
Total amount of training epochs: 15

Nuclear Segmentation Network (NSN)

The training of the NSN was very similar to that of 3D U-Net. We used the Sigmoid Output function because of its simplicity and used a DICE-loss function. We observed a similar convergence rate. The optimizer used was Stochastic Gradient Descent (SGD), since the authors also used this.

The hyperparameters used:

Loss function: DICE-loss
Optimizer: SGD
Learning rate: 0.0001
Weight decay: None
Momentum : 0.9 (=Torch default)
Total amount of training epochs: 15

Nuclear Detection Network (NDN)

For training the NDN network we used the same settings as the NSN. Convergence was much slower.

The hyperparameters used:

Loss function: DICE-loss
Optimizer: Adam
Learning rate: 0.0001
Weight decay: None
Momentum Decay rate estimates (beta1,beta2): (0.9, 0.999) (=standard Torch ADAM values)
Total amount of training epochs: 50

Post-processing

Post-processing contained two parts, de-interpolation of the segmented images and creating instance segmentation from the sementic segmentation provided by the networks.

The de-interpolation method was not mentioned by the authors. Addionally, nothing was mentioned on what their procedure was to handle padding or cut data from the resizing in the pre-processing of the images. We decided on using the resize transfrom function from the skimage python library to resize the images to 51 × 128 × 128 (since the original images contained 51 slices).

After resizing the sementic segmentations, the authors mention having used a watershed algorithm on the combined output of the NSN and NDN to create the full QCAnet. They do not provide the methods for instance segmentation used for the post-processing of the 3D U-Net and NSN outputs.

For the 3D U-Net and NSN networks instance segmentation was achieved by labeling continuous voxel volumes as a single instance. This method seemed to reproduce the results of the authors closest. This was achieved with the skimage library, using the skimage `measure.label()` function.

The watershed algorithm made use of finding the middle of each segmented nuclei of the NDN via the local maximum of a distance function. These centres were used markers for the watershed algorithm performed on the output of the NSN. Together this made the QCANet output. For this we made use of the `distance_transform_edt` from scipy and segmentation package of skimage.

The final image was created with pyvista.

Results

The 3D U-net networks and the Nuclear Segmentation Network seemed to converge pretty quickly; between 5 and 10 epochs. The Nuclear Detection Network took longer to converge; about 40 epochs were needed. This discrepancy in convergence can be explained by the Neural Detection task being more diffuclt to learn. An indication of the performance of the networks can be obtained from the figure 6. It should be noted that the input data is training data for this figure, as the dataset did not include groundtruth data for the NDN network.

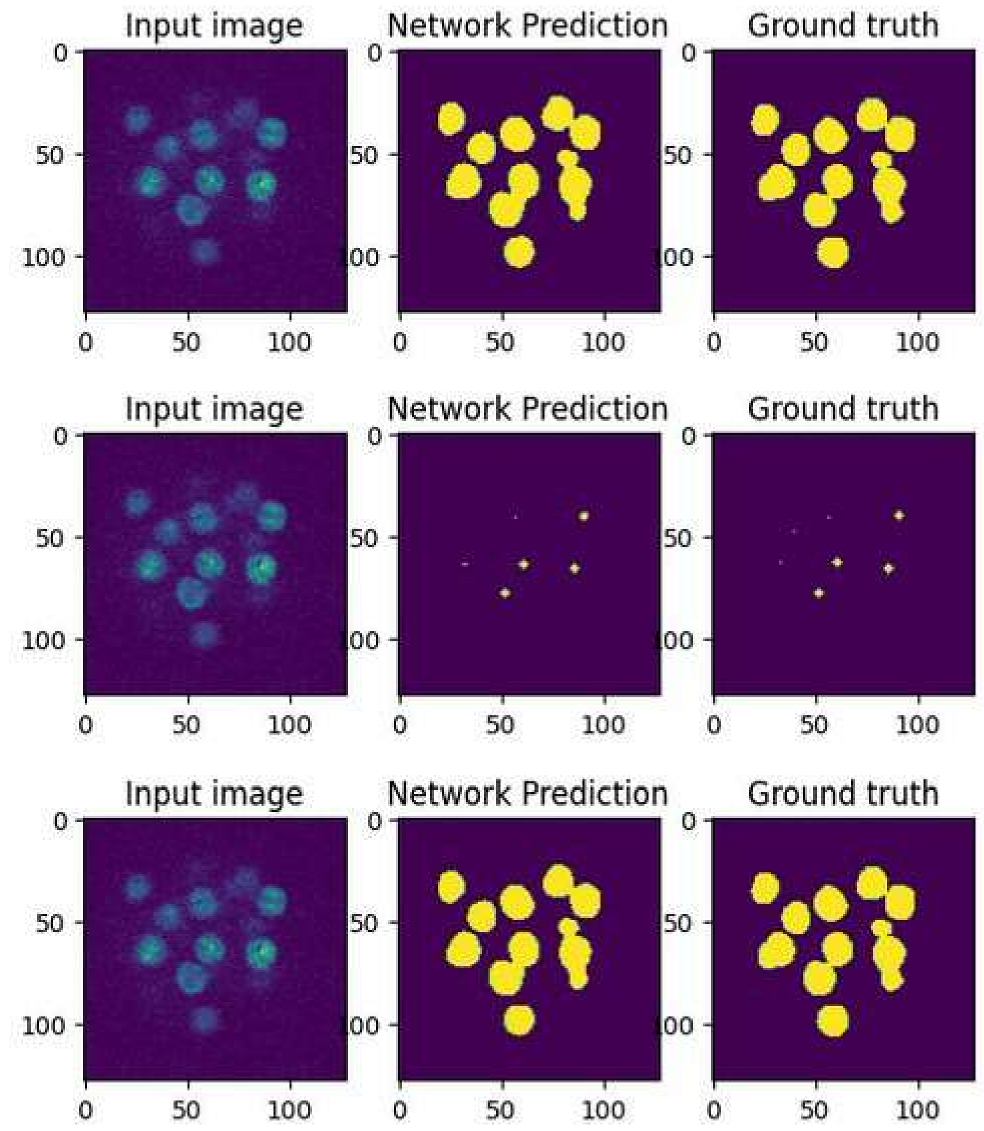


Fig 7: Single 2D slice semetic segmentation output of training data. Networks used from top to bottom: Unet, NDN, NSN

Table 1 below summarizes the found average validation and test scores for the networks with the highest validation IoU scores, for all the different networks that were trained. As can be seen from this table; changing either the Loss function our output function of the 3D U-Net does not constitute a significant change in IoU scores. We can also observe that the more shallow NSN network performs as well as the more deep 3D U-Net, indicating that this level of deepness might not be needed for this simple binary segmentation task. The much lower IoU score for the NDN can be explained by the fact that in this task much smaller areas have to be predicted, making it inherently harder to reach high IoU scores. Also the average test IoU score is missing for the NDN since the authors did not supply any test data for the NDN.

When comparing the values we found to the values of the authors, it becomes immediatly clear that our values are significantly higher, see discussion for more on this.

IoU-Scores

Architecture	**Validation IoU**	**Test IoU**	**Test IoU Authors**
3D U-Net Softmax BCE	0.916	0.824	0.702
3D U-Net Sigmoid BCE	0.916	0.817	-
3D U-Net Sigmoid DICE	0.922	0.823	-
NSN	0.920	0.831	0.742
NDN	0.612	-	-

Table 1: This table shows the average IoU scores for the validation and test set for the different architectures and compares these to the values found by the authors of the reproduction paper.

Recreation of figure 3

The main deliverable of this project was a reproduction of figure 3 from the original paper. As we can see from figure 8, the 3D U-Net and NSN alone are able to perform instance segmentation, however when cells are quite close to each other they fail. This becomes more apparent for later cell embryo stages, where there are more cells and where the cells are more close together. Even in these later stages, the QCANet seems to perform instance segmentation quite well. This is in alignment with the findings of the authors of the paper we have replicated.

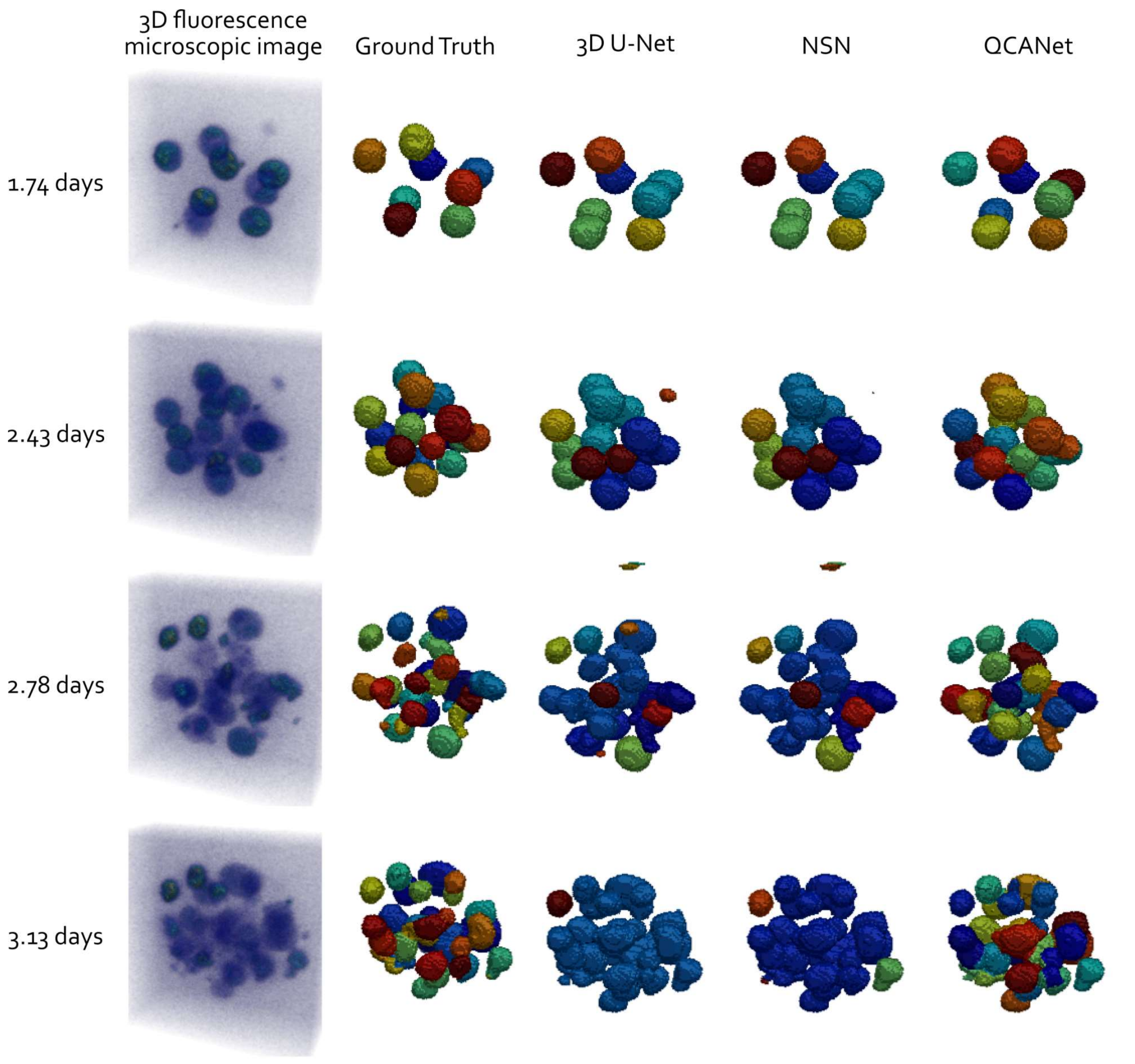


Fig 8: Reproduction of figure 3 from the original paper.

Discussion

Altough the IoU values in the Results sections are different from the ones the authors found, the most important part of the independent reproduction (reproducing figure 3) was succesful. However, while implementing the methods mentioned in the paper, it became clear that not all details are mentioned or supported. These will de discussed further in this section.

Amount of epochs and comparison of IoU scores

As became clear in the results section, the IoU scores that we found are significantly higher than those of the researchers, even though our (3D U-Net and NSN) networks were only trained for about 5-10 epochs instead of the at least 100 epochs the authors mention. The exact reason for this discrepancy is unclear. Regarding the difference in amount of epochs; to us it seems unlikely that such a simple task (binary segmentation of cells) with only ~400 training images will take more than 100 epochs to converge and give high IoU scores. It is possible that the >100 training epochs the authors mention are for the full 11-fold cross validation they have performed, training for about 10 epochs for every different validation, however this remains unclear.

Regarding the difference in IoU scores; it is also unclear why the IoU scores of the authors are relatively low for such a simple task as binary segmentation. It might be that the way they calculated their IoU is different. We have checked our IoU scores with the built in Torch function called JaccardIndex (which is equivalent to IoU) and found exactly the same values. Another thing that is strange when looking at their scores, is that the score of their more deep 3D U-net is actually significantly lower than the score of their NSN. For a deeper network we would expect at least a IoU score that is similar after convergence, which is what we found.

Cross Validation

The researchers use 11-fold cross validation to test the performances of the different algorithms. They used the data of 10 embryos (110 samples) as training data and 1 embryo (11 samples) as validation data. This process was repeated 11 times to complete the 11-fold cross-validation, which would take up a lot of time. Since the scope of this reproducibility project is to only reproduce figure 3, we chose not to do 11-fold cross-validation as it was infeasible to run our algorithms for long amounts of time on an online GPU. Instead, we chose to do a validation using the samples of the first 10 embryos as training data and the samples of the 11th embryo as validation data.

Sigmoid vs. Softmax

For the activation functions of the NSN and NDN, the researchers chose to use a softmax function, which is used to compress the inputs of multiclass problems into values between 0 and 1. However, the same result can be achieved using a sigmoid function, since there are only 2 classes in our case (background and non-background). We tested our NSN, NDN and 3D U-Net networks with both activation functions and observed virtually no difference in the train and validation accuracy for the IoU as well as for the train and validation loss, see figure 9. Something to note in figure 9 is that after 5 epochs, the validation accuracies of both the sigmoid and softmax activation functions get worse, which probably means that they are overfitting. We would recommend using the sigmoid function in this 2 class problem as it is a less complex function and should work faster than the softmax function.

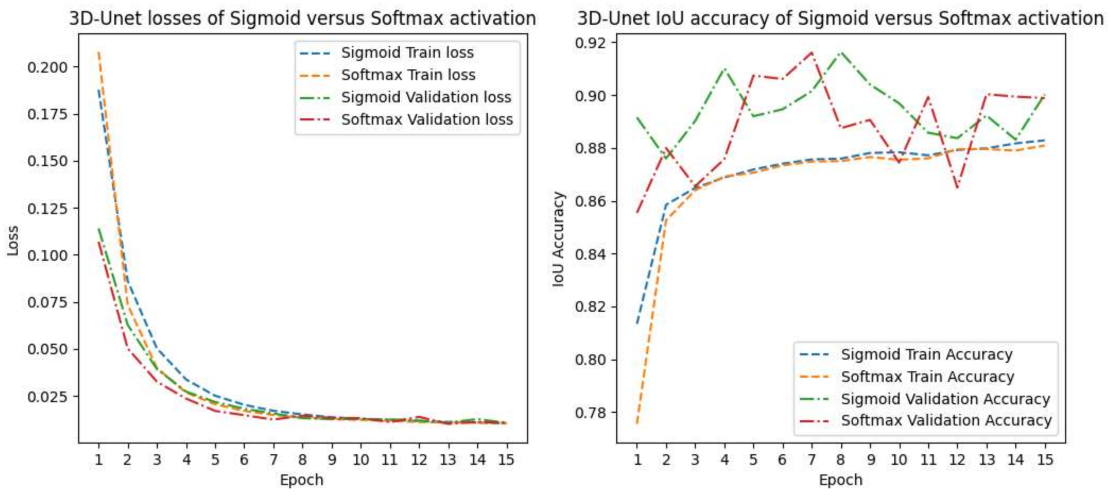


Fig 9: Influence of two different output functions, softmax and sigmoid. Left: Training and validation loss for different epochs and different output functions. Right: IoU accuracy for different epochs and different output functions. (For all algorithms the same optimizer, learning rate and loss function were used)

Sloppiness of writing

Another thing to remark about the paper is the sloppiness of the writing that is used. Some sentences talking about the amount of parameters of the different algorithms lack some commas. This leads to misinterpretation, as "NDN had 44,447,940 parameters more than 3D U-Net." has a different meaning than "NDN had 44,447,940 parameters, more than 3D U-Net." Our NDN had indeed 44,447,940 parameters, not that amount more than the 3D U-net. We think it is not acceptable for something like this to happen in a paper that is published in an acclaimed journal like Nature.

Different optimizers and loss functions

One of the goals of the paper was to show that their own developed algorithm (QCANet), outperformed the other algorithms (3D U-Net and 3D Mask R-CNN). This was done by comparing the measures IoU, SEG and MUCov for the different algorithms. To be able to do this in an equal manner, you would expect that the only difference between the algorithms would be the used networks. However, the paper uses different optimizers and loss functions for each algorithm, which makes a fair comparison quite difficult. We tested the NSN with different optimizers, the Stochastic Gradient Descent (SGD) that was used by the paper and an Adam optimizer which was used in the 3D U-Net. No significant differences were observed, so it remains questionable why the researchers chose to use different optimizers. We also looked at the use of different loss functions for the 3D U-Net, a dice loss function and binary cross entropy (BCE) loss. No real differences were observed in IoU values, however there was more variation in the validation accuracy using the BCE loss. We think for a scientific deep learning paper in which a comparison between different algorithms is made, it would be more precise to use the same optimizers and loss functions or explain why different ones were used.

Additionally we believe that for a more equal comparison of the networks a traditional watershed algorithm should be applied to the output of the NSN and 3D U-Net. As a sequence of erosion, maximum local distance and watershed would already improve the instance segmentation significantly from solely applying continous volume labeling.

Omitted edge-cells

Due to the limited field-of-view of the microscope, occasionally a cell is only partially visible near the edge of the image. In case the nucleus of such an edge-cell is located beyond the border of the image, NDN will not be able to detect its nucleus, while NSN has successfully segmented the rest of the cell (see Fig 10). Therefore, the cell will disappear during the watershed algorithm.

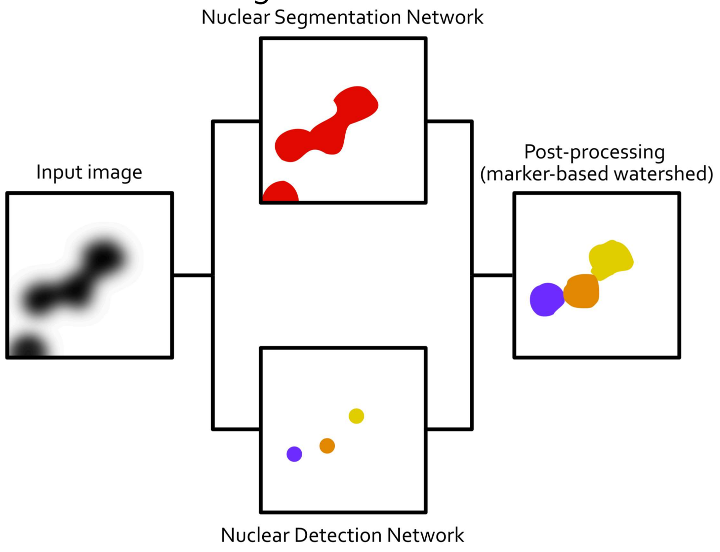


Fig 10: Flow diagram that shows the QCANet algorithm with a disappearing cell at the edge. The NSN segments the edge-cell, but the NDN did not segment a nucleus for the edge-cell. The edge-cell is not recovered during post-processing, due to the missing marker for the watershed algorithm. This results in this cell not appearing on the instance segmented image.

Missing NDN test images

Another point of critisism stems from the absence of test ground truth images for the Nuclear Detection Network. Unlike 3D U-Net and QCANet, this means we were unable to test the performance of NDN by itself.