A convolutional neural network segmenting fission yeast microscopy images

Ajkuna Seipi, Hongyi Shi, Louis Perrier
CS-433 Machine Learning, project 2, fall 2022

***Abstract*—The identification of cell borders in microscopy images constitutes a bottleneck for large-scale experiments. The laboratory of Biophysicy of the EPFL Institute of Physics (IPHYS) has been working on the $Saccaromyces\ pombe$ organism. They have presented a convolutional neural network (CNN) named YeaZ. The latter is highly accurate and it transfers well to other conditions[2]. However, cell segmentation remains still a challenge. Some of the smallest cells cannot be predicted without a frequent human interventions. As cells progress in their division cycle, they also grow. The goal is to use temporal data to improve the precision of the cell segmentation, with the hope that later time points, in which cells are bigger, are helpful for the segmentation of the earlier time points in which cells were just born and are therefore small.**

I. Introduction

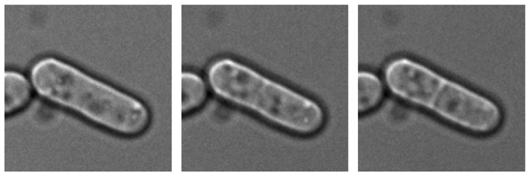
The fission yeast $Schizosccharomyces\ pombe$ has become a prominent model in molecular biology, both in yeast genetics and to investigate the molecular mechanism of the cell cycle[3]. The current segmentation methods for yeast images rely on classical image processing techniques. However, the segmentation produced by those techniques is not perfect and most of the time, it requires human correction. The challenges for yeast image segmentation are the irregular shapes, cell crowding, and light conditions. The main challenge in our case was to find a way to detect a border between two newly born cells, as soon as it appears. This border is called septum and is often too thin in its early stage to be detected by the YeaZ CNN[2]. A way to deal with that is to take into account the temporal variable. In fact, by training on several time frames, we aim to predict the newborn yeast cells, which are hard to identify with exclusively the present frame, since we cannot be sure if it is indeed two newborn cells or a single established cell as the septum separating two newborn cells may be very thin and hard to distinguish.

Therefore, we had to choose an adequate model which can handle time series data.

II. Data set

We were provided with a data set containing four videos. Each of those videos is a data set of the shape Time x Height x Width (TxHxW). 10 minutes separate each frames and through several frames, we can see the growth of the fission yeast cells. As their name suggests, the cells divide themselves into two cells by a fission mechanism, i.e. one cells, when grows big enough, splits into two equal cells.

The four data sets are sequence frames of size *(180x1024x1024)*, i.e. there are 180 images of 1024 pixel high and wide, which are saved in multi-layer TIF files. Each of those data sets contains both the **raw greyscale images** (i.e. microscopic images) and their corresponding **ground truth masks**, both of the same size. In a segmentation task, we consider the masks as labels. Each cells has its own unique ID in the mask and septa are labeled as background.


(a) Raw images

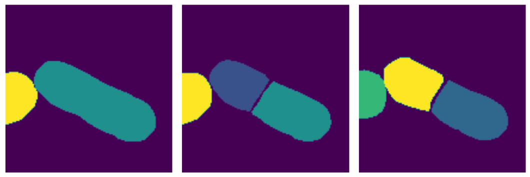

(b) Ground truth masks

Fig. 1: 3 sequential patch frames. At frame 4 (left), there is no septum, so it is a single big cell At frame 5 (middle), a thin septum is visible, so the ground truth mask identifies two newborn cells but it may be difficult for the YeaZ CNN to do so. The septum is more visible on the raw image at frame 6 (right).

In the following parts, we define X the set of raw greyscale images, Y_{true} the set of ground truth masks, and Y_{pred} the set of prediction masks made using our model.

III. The Model

We decided to use the **3D U-Net** model. As its name suggests, it is the well known U-Net model adapted to volumetric images. U-Net is a convolutional neural network frequently used for the segmentation of biomedical images and developed in the university of Freiburg, Germany.[1] We expect the temporal variable of our data set to act like a 3rd dimension in volumetric images.

A 3D U-Net model has the same structure as the regular U-Net model, the only difference being the use of 3D layers instead of 2D layers i.e. it uses 3D convolution layer instead of regular 2D convolution layer as example. Thus, we have an analysis path and a synthesis path, each made of 4 levels. There is a **shortcut connection** from each analysis level to

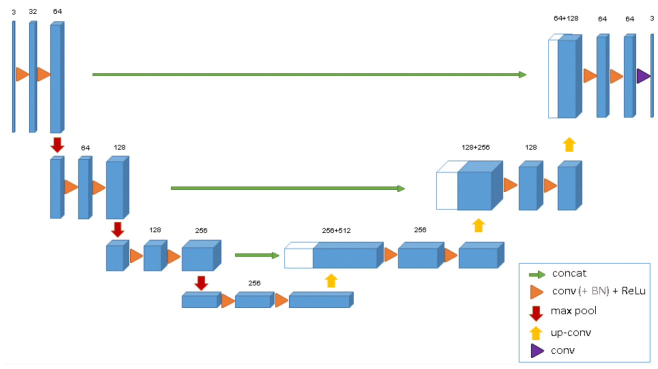


Fig. 2: architecture of a 3D U-Net model [1]

its corresponding synthesis level i.e. synthesis level of same resolution. In the analysis path, at each levels there are two **3D convolutions**, with a kernel of size *3x3x3*, and one **3D max-pooling**, with a kernel of size *2x2x2*. In the synthesis path, we use a **transposed 3D convolution**, with a kernel of size *2x2x2* and a stride of size *2* in each dimension, and two **3D convolution**, with a kernel of size *3x3x3*. Each 3D convolution in both the analysis and synthesis paths is followed by a **batch normalization** and a **regularized linear unit (ReLU)** layer.

As our dataset Y is made of greyscale images, we have a single input channel.

We used **Binary Cross Entropy loss** L_{BCE} to train our model

$$L_{BCE} = -\frac{1}{2}\sum \log p_i$$

where p_i is the probability of a pixel $y_{pred} \in Y_{pred}$ belonging to class i, $i \in \{0, 1\}$.

With a such architecture and an input size of *16x64x64*, we get a total of *90,316,290* parameters.

To implement the model, we used the following code https://github.com/bnsreenu/python_for_image_processing_APEER/blob/master/tutorial122_3D_Unet.ipynb. The code uses the Keras library, a popular library for Deep Learning.

We exclusively ran our model using free Google Colab, the machines used were thus rather limited in term of performance and capacity which caused us a few trouble.

IV. Data processing

A. Images patching

180 frames of 1024x1024 px images are too big to be processed all at once on regular machines which is why we first split them into smaller patches that we use for training. We split them into patches of size 16x64x64 with a step of 5x64x64. Those patches might overlap but only in the temporal dimension and the size is big enough to display a few cells on the same patch. This means that, the fist sample is made of the 16 first time frames (from time 0 to time 16), the next sample is made of the time frames going from 5 to 21, and so on.

We also split them into patches of size 16x128x128 with step of 5x128x128, this size is more fitted to our need as

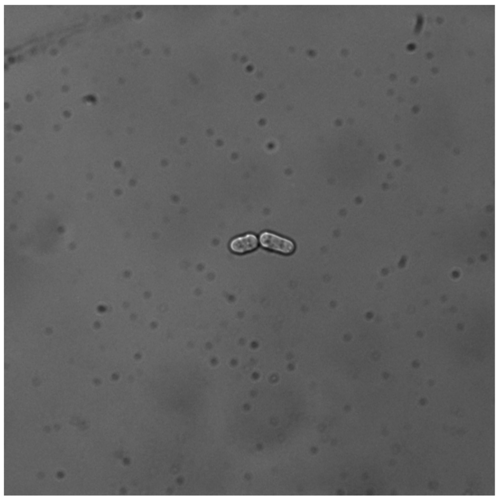


Fig. 3: a raw image frame. There are two cells, the image is mostly empty and will thus translate to a mostly empty masks.

it can display an entire cell in a single patch, which was not possible with patches of size 64x64, however its size make the training and the prediction much harder to run. We compare the **mean IoU (Intersection Over Union)** between the two size of patches.

The IoU evaluates the overlap between **Ground Truth** Y_{true} and **Prediction region** Y_{pred}. It is computed as follows:

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

with TP the true positive rate, FP the false positive rate and FN the false negative rate. A pixel is a true positive if its value $y_{pred} = 1 = y_{true}, y_{pred} \in Y_{pred}, y_{true} \in Y_{true}$.

B. Removing empty patches

We also removed patches which have mask containing only 0, meaning they show only background pixels and no cells. Indeed, most frames in our data set show a lot of empty space. Once split into patches, there were a majority of those empty patches. As they are easy to predict, it was impacting highly our validation metric results making it less meaningful. However, when we tried to apply our model to predict large images and not only patches, we encountered a significant lack of precision that might be related to the lack of empty patches which conducted to overfitting. Namely, on empty patches, the model tends to predict cells on the borders of the patch. We tried to correct this by training the already trained model on a different dataset which kept those empty patches. The results after 10 to 30 epochs were arguably better, prediction on large images showed less noise.

C. Binarize the masks

The initial Ground Truth set Y_{true} is made of different masks with several values such that $Y_{true} \in \{0, 1, ...N\}^3$ where N is the number of cell in the image. Thus, for $y \in Y_{true}$, if $y = 0$, the pixel lies in the background, and if $y > 0$, y is the ID of the cell the pixel belongs to. Since we decide to do a binary segmentation, we have to binarize the masks; turn them into masks containing only two classes, i.e. value 0 for background and 1 for cell.

D. Standardize

A crucial part in the preprocessing of our data is to standardize the images X. In fact, we need to have a maximum value of pixel equal to 1, $\max(X) = 1$. This ensures that each input has a similar data distribution. To do this, we divided the input images $x \in X$ (which are arrays of pixels) by the maximum value, $\frac{x}{\max(X)} \forall x \in X$.

E. One-hot encoding

In image segmentation, we deal with categorical data which are variables that contain label values rather than numeric values. In our model, we have two categories, the background and the cells. Thus, we use one hot encoding to convert data to prepare it for the model. We convert each categorical value into a new categorical column and assign a binary value of 1 or 0 to those columns. So, each integer value is represented as a binary vector.

F. Split data set into training and validation set

As said in section II, we have four sequences of frames of the same size. To avoid overfitting as much as possible, we include the four sequences of frames into a unique data set of patches. We then simply split randomly the data set into training and validation sets in an 0.8:0.2 ratio (respectively).

V. Results

We trained our model with different parameters, and we compared the Mean IoU for each of them.

First of all, we have two output channels in our model, therefore we apply the **Softmax** function as activation function in the final layer. By using one hot encoding, instead of having one category to predict (i.e. only cell), we have two categories including the background category.

As an optimizer, we use the **Adam optimization algorithm** with an initial learning rate of *0.0001*. As Adam is an adaptive optimizer, we assume the initial learning rate is not that important and did not compare with other initial learning rates.

A. One hot encoding

Using one hot encoded labels to train our model gave us much better results. We compared the results by training a same model on patches of size 16x64x64. While using one hot encoded labels, we achieved a mean IoU of 0.9601 while without, the best result we could get is 0.8702.

B. Patch size

One of our main focus was to determine the optimal choice of patch sizes. As stated in the section II, taking the whole sequence frames took too much memory and therefore the training process was too long.

Thus, we decided to train our model with different smaller patch sizes. The results are displayed in the table below.

Patch size	**Mean IoU**	**Epochs**
16,32,32	0.8812	100
16,64,64	0.9601	100
16,128,128	0.9600	100

TABLE I: The effect of patch size

As seen in the table above, having patches of very small size produces bad results. Since the patches are small, only a few images show non-trivial display of the fission yeast cells; most patches either show only background or fully lie inside a cell. Therefore, there is an overfitting of the segmentation task.

Obviously, the bigger are the patches and the better is the prediction of the images. But, by taking bigger images, it requires more resources and more time for the model to train. We need to find a threshold between the time constraint and the prediction accuracy. That is why we focused on working on patches of size 16x64x64.

C. Loss function

We trained our model with two loss functions, and we compared the results.

First, we used the **Binary Cross Entropy loss** L_{BCE}. This loss function is the one usually used for binary segmentation tasks. The reason is that its gradient can be easily computed.

Then, we tried to train our model with the **Dice Coefficient loss** $L_{DCE} = 1 - DCE$. The main reason to use this loss function is that the our actual goal is to maximize the Dice Coefficient (i.e. the IoU). The Dice Coefficient is also known to perform better at class imbalanced problems.

Loss	**Mean IoU**	**Epochs**
L_{BCE}	0.9601	100
L_{DCE}	0.8322	100

TABLE II: The effect of L_{BCE} and L_{DCE} loss functions

We can see in the table above that our model does not train well with the Dice Coefficient Loss. The dice coefficient diverges at each epoch during the training process. Therefore, we get a low mean IoU. Thus, we decided to train our model with the L_{BCE} loss for better results.

D. Batch size

We trained our model with different batch sizes on patches of size *16x64x64*. We could not train with a batch size bigger than 8 because of the lack of memory. As it can be seen above, the size of batches can clearly improve the mean IoU. By taking a large number of samples at the same time, we

Batch size	**Mean IoU**	**Epochs**
1	0.9231	100
4	0.9533	100
8	0.9601	100

TABLE III: The effect of batch size

reduce overfitting at each training epoch and avoid diverging results which lower the mean IoU.

E. Activation layer function

As a matter of change, we vary slightly the initial model by using *leaky* ReLU with a factor of $\alpha = 0.3$, instead of the basic ReLU. We compare the model of batch size 8 with patches of size 16x64x64 with both activation function layers.

As seen below, The *leaky* ReLU function improves slightly the mean IoU.

Activation function	Mean IoU	Epochs
ReLU	0.9601	100
Leaky ReLU	0.9625	100

TABLE IV: The effect of the *leaky* ReLU and ReLU

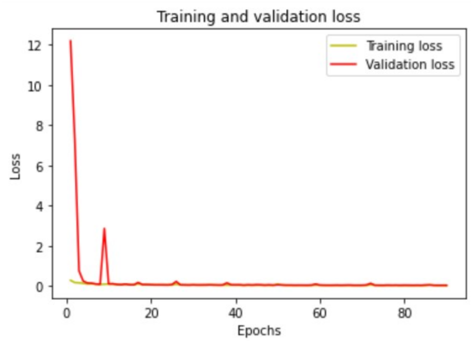


Fig. 4: Training and validation loss curve for 16x64x64 patches, batch size of 8 and *leaky* ReLU function

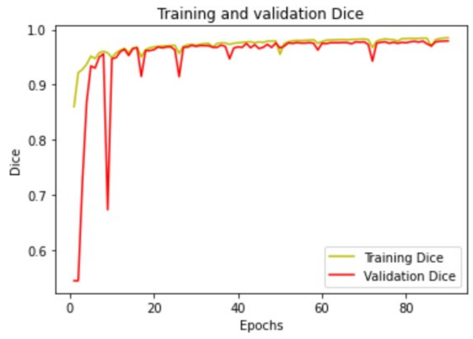


Fig. 5: Training and validation dice coefficient curve for 16x64x64 patches, batch size of 8 and *leaky* ReLU function

VI. Comparison to other models

The YeaZ model achieved a mean IoU between 0.91 Other researchers have conducted the same type of experiment on volumetric data set and have compared different model. The advantage of the 3D U-Net is its simple implementation and the fact that as a model, it is not as resource-heavy compared to others due to its fewer components. A further step would be to compare the results on our data set with other type of models, such as CLSTM 2D U-Net[4] or Temporal Attention Networks such as U-TAE.[5]

VII. Summary

Through the experience, we have shown that 3D U-Net can be efficiently used on videos frames in order to segment fission yeast microscopic images. Indeed, despite 3D U-Net being optimized for volumetric images, it can be easily adapted to 2D video frames by using the 3rd dimension as the temporal dimension.

In our case, the **Binary Cross Entropy loss** worked the best as we were aiming for binary segmentation and the model using *leaky* ReLU layer performed faster and slightly better. We achieved a Mean IoU of *0.9625* after *100 epochs* on patches of size *16x64x64* with a batch size *of 8* and using the **Dice Coefficient** as validation metrics and the **Adam optimizer** with an initial learning rate of *0.0001*. Each epoch took *4* minutes on average.

Our implementation is far from being perfect due to the machine limitation, for example, training on slightly bigger patches might have given better results. Indeed, even though our model achieved a 0.96 Mean IoU, it was still struggling with some specific patches. We noted quite a few cases where the prediction almost integrally failed: when the ground truth mask should have been almost covered of cells, our model would predict it has integrally empty. This is caused by the patch size that we had to limit due to our machine capacity. The patch does not catch any cell's edge as it is too small and thus struggles to predict correctly.

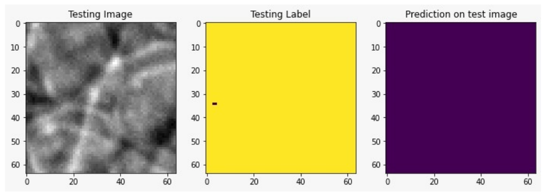


Fig. 6: Failed patch prediction

Furthermore, another data pre-processing that we could have applied is data augmentation by applying different type of **rotations** with different angles. **Zoom** and **Flipping** the data set has shown to prevent overfitting as well. Data augmentation could have been a good factor for accuracy and mean IoU improvement[2]. Once again, due to our machine limitation, it was quite hard to further augment our data set.

Despite 3D U-Net being a model much lighter than the other available models, it still remains a challenge to run on large data sets with high resolution images. In fact, it is common practice to split input data set into smaller patches in order to train it more efficiently. In our case, the pictures were of size 1024x1024 px which is rather a regular image resolution nowadays as it is hardly the size of a laptop screen or the resolution of a 1800p video which is considered as standard on popular social media platforms such as Youtube. Pictures and videos taken by modern camera usually have an even higher resolution, running such a deep learning model on a data set made of images taken on daily life basis translates into a challenge without the right tools.

References

[1] Özgün Çiçek et al. (2016). *3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation*. DOI: https://doi.org/10.48550/arXiv.1606.06650. URL: https://arxiv.org/abs/1606.06650. (accessed: 18.12.2022).

[2] Dietler et al. (2020). *A convolutional neural network segments yeast microscopy images with high accuracy*. DOI: https://doi.org/10.1038/s41467-020-19557-4. URL: https://www.nature.com/articles/s41467-020-19557-4. (accessed: 18.12.2022).

[3] Wikipedia (2022). *Schizosaccharomyces pombe*. URL: https://en.wikipedia.org/wiki/Schizosaccharomyces_pombe. (accessed: 18.12.2022).

[4] Rose Rustowicz et al. *Semantic Segmentation of Crop Type in Africa: A Novel Dataset and Analysis of Deep Learning Methods*. URL: https://openaccess.thecvf.com/content_CVPRW_2019/papers/cv4gc/Rustowicz_Semantic_Segmentation_of_Crop_Type_in_Africa_A_Novel_Dataset_CVPRW_2019_paper.pdf. (accessed: 18.12.2022).

[5] Vivien Sainte Fare Garnot and Loic Landrieu. *Panoptic Segmentation of Satellite Image Time Series with Convolutional Temporal Attention Networks*. URL: https://arxiv.org/pdf/2107.07933.pdf. (accessed: 18.12.2022).