ISSN 2464-4617 (print)
SSN 2464-4625 (DVD)
Computer Science Research Notes
CSRN 3101
WSCG 2021 Proceedings

# Learning Cell Nuclei Segmentation Using Labels Generated with Classical Image Analysis Methods

Damian J. Matuszewski
Department of Information Technology, Uppsala University
Sweden, 75105 Uppsala
damian.matuszewski@it.uu.se

Petter Ranefall
Department of Information Technology, Uppsala University, and SciLifeLab BioImage Informatics Facility
Sweden, 75105 Uppsala
petter.ranefall@it.uu.se

## ABSTRACT

Creating manual annotations in a large number of images is a tedious bottleneck that limits deep learning use in many applications. Here, we present a study in which we used the output of a classical image analysis pipeline as labels when training a convolutional neural network (CNN). This may not only reduce the time experts spend annotating images but it may also lead to an improvement of results when compared to the output from the classical pipeline used in training. In our application, i.e., cell nuclei segmentation, we generated the annotations using CellProfiler (a tool for developing classical image analysis pipelines for biomedical applications) and trained on them a U-Net-based CNN model. The best model achieved a 0.96 dice-coefficient of the segmented Nuclei and a 0.84 object-wise Jaccard index which was better than the classical method used for generating the annotations by 0.02 and 0.34, respectively. Our experimental results show that in this application, not only such training is feasible but also that the deep learning segmentations are a clear improvement compared to the output from the classical pipeline used for generating the annotations.

**Keywords**
Deep learning, U-Net, CellProfiler, Data annotation, Microscopy

## 1. INTRODUCTION

Neural network-based image segmentation and classification have made huge progress in the last decade. However, the ground truth creation is still a major bottleneck for training deep learning models in many applications [Lec15a, Xin17a, Gup19a]. Here, we propose a simple and efficient way to automatically generate training labels for the segmentation of cell nuclei – with minimal manual interaction. Of course, some data still has to be manually annotated to evaluate the automatic label generation and the network's performance. However, our approach limits this only to the images in the test set as the network is trained entirely on the automatically generated labels. Moreover, our deep learning (DL) model has shown to substantially improve the results compared to the classical image analysis pipeline that was used for generating its training labels.

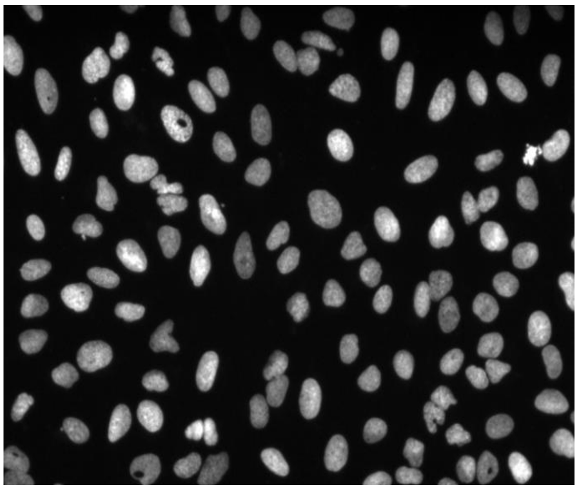
**Fig. 1. Raw input image.**

In Sect. 2, we describe the proposed method and neural network design, as well as the dataset that has been used for evaluation. In Sect. 3, we describe the metrics used for evaluation and present the results of the experiment.

## 2. METHOD

### 2.1 Dataset

As the application for demonstrating the proposed method, we have selected the task of cell nuclei segmentation. We used the BBBC039v1 dataset, available from the Broad Bioimage Benchmark Collection [Cai19a, Ljo12a]. This dataset has a manually created ground truth (GT), here only used for evaluating the results and not for training. The images are 690x520 pixels, and the nuclei typically have a diameter between 10 and 50 pixels. An example of an image from this dataset is shown in Fig. 1. For practical reasons, each image was tiled into four overlapping tiles of size 512x512 pixels. The tiles from the same image were always kept in the same

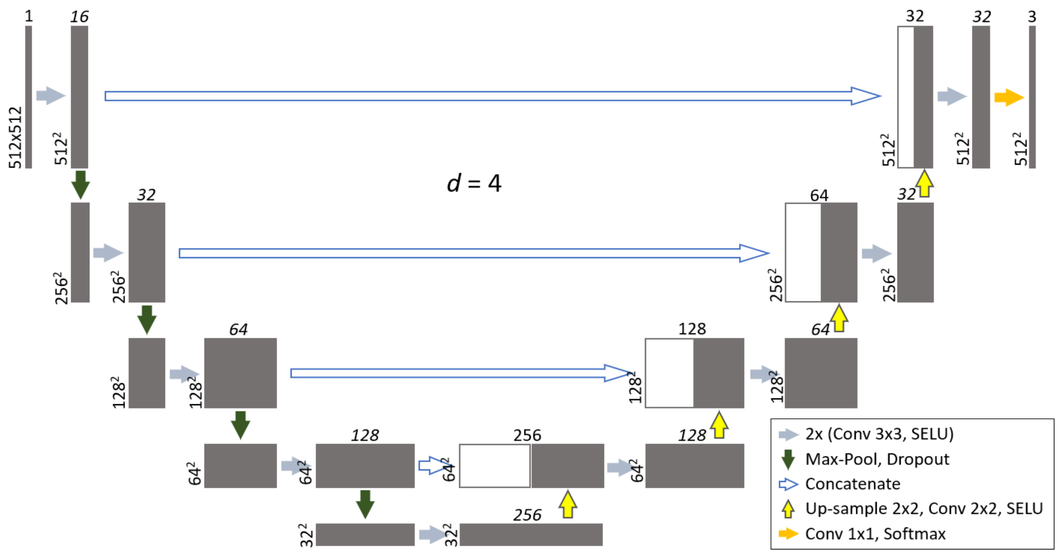

**Fig. 2. The U-Net-based deep learning architecture.**

data partition: training, validation, or test data subset, respectively. The initial dataset had 200 images with a recommended fixed split of 100-50-50, which resulted in 400 image tiles for training, 200 for validation, and 200 for the test. Before feeding the image tiles to the network, we first augmented them with all combinations of flipping and multiple 90 degrees rotations (this resulted in 8 image tiles from 1; it was done only with the training and validation sets and not the test set) and then normalized them by subtracting the corresponding mean intensity value from each augmented image tile and dividing it by the standard deviation, which is a standard procedure in deep learning.

## 2.2 Automatic generation of training labels

We used segmentation results of classical image analysis methods as the labels for training our neural network model. The segmentations were created using the CellProfiler [Mcq18a] module IdentifyPrimary-Objects with the threshold method ‘Minimum cross-entropy’ and the default Threshold smoothing scale=1.3488 to detect the objects [Sez04a], ‘Shape’ as the Method to distinguish clumped objects, and ‘Intensity’ as the Method to draw dividing lines. The object separation in CellProfiler is done using the Watershed algorithm [Vin91a]. The difference between ‘Shape’ and ‘Intensity’ when distinguishing clumped objects is that ‘Shape’ uses the local maxima of the distance transform [Bor86a] of the binary objects as seeds, whereas ‘Intensity’ uses the local maxima of the input image intensities as seeds. The separation lines are drawn by the Watershed algorithm for the previously defined seeds, based on the distance transform for ‘Shape’ or the input image intensities for ‘Intensity’.

We also tried the three other segmentation configurations: Shape/Shape, Intensity/Shape, and Intensity/Intensity. However, the selected Shape/Intensity configuration gave the best results by a great margin. In all configurations, we used the same 10 to 50 pixels interval for the allowed object size (diameter). Finally, we tried the consensus between different segmentations: a combination of the three best segmentation configurations obtained by masking out (so that they are ignored during the training of the neural network) those cell clusters that did not get the same number of objects after the separation. The main idea behind it was to purify the training data by excluding difficult cases that were most likely erroneously labeled. However, this did not improve the results with respect to using only the best of the segmentation settings (Shape/Intensity), as shown in Section 3.

## 2.3 Neural network design

We used a modified U-Net [Ron15a], a fully convolutional neural network presented in Fig. 2. Following [Mat19a], we reduced the number of feature maps in each layer with respect to the original U-Net and increased it in the last convolutional block. We kept the same depth of the architecture as the original U-Net, i.e., 4. However, we added dropout layers [Sri14a] after each max-pooling with increasing rates: 0.2, 0.25, 0.3, and 0.35, respectively. We used zero padding and selu [Kla17a] as the activation function in all convolutional layers except the last one in which we used softmax. We also used L2 weights regularization with l=0.0001 in all convolutional layers except the output layer. We optimized the models using Adam [Kin14a] with the default hyperparameters except for the learning rate which was set to 0.00001. To make the training reproducible and to reduce the stochastic factors in our comparisons of training with different labels, we used the same

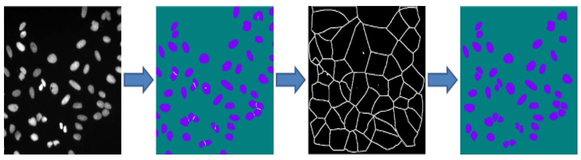
**Fig. 3. Post-processing. From left to right: 1) raw input image, 2) segmentation results, 3) the cell separation lines are dilated and merged with the background; the resulting image is then skeletonized and dilated, 4) new separation lines are subtracted from the nuclei segmentation results; the objects touching the image edges are removed.**

network architecture and hyper-parameters in all models. Moreover, we fixed the random initialization seeds of the network weights (we used Glorot normal initialization [Glo10a]) in all models as well. As a consequence, all models started with the same values in the corresponding network weights.

We trained the network to assign each pixel in the input image to one of the three classes: background (BG), nucleus (N), or nuclei separation lines (SL). We used a custom loss function $\lambda_D$: the class-weighted sum of the log Dice coefficient losses [Mat19a] defined by:

$$\lambda_D(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -(0.2 \log \delta_{BG} + 0.4 \log \delta_N + 0.4 \log \delta_{SL}), \quad (1)$$

$$\delta_C = \frac{2\sum y_i \widehat{y_i} + 1}{\sum y_i + \sum \widehat{y_i} + 1}, \quad (2)$$

where $y$ is the ground truth, $\hat{y}$ is the segmentation of class $C$, and $\delta_C$ is the pixel-wise Dice coefficient for that class. We trained each model for 50 epochs and selected the version with the smallest loss on the validation set annotated in the same way as the corresponding training set.

## 2.4 Post-processing

To ensure that the resulting division lines separate the clumped nuclei, we dilated them with a 3x3 pixel square structuring element, merged them with the background, and then skeletonized, followed by dilating the skeleton by 1 pixel. Finally, we subtracted the new divisions from the nuclei mask. Nuclei touching the edges of the images were also excluded, both in the ground truth and in the segmentations. See Fig. 3.

## 3. RESULTS

### 3.1 Metrics

As the metrics for comparing our results, we used two types of accuracy: 1) the pixel-wise Dice coefficient for the nuclei and separation lines before the post-processing; and 2) the object-wise Jaccard index $J$ of the segmented nuclei after the post-processing:

$$J = \frac{|M|}{|G| + |R| - |M|}, \quad (3)$$

where $|G|$ is the number of nuclei in the ground truth (GT), $|R|$ is the number of nuclei in the result, and $|M|$ is the number of 1-to-1-matched nuclei between the results and the GT. There is a results-GT nucleus match if a segmented nucleus has a spatial overlap with exactly one nucleus in the GT, and that corresponding GT nucleus overlaps with exactly one nucleus in the segmentation results. The overlaps were computed between eroded (with a 3x3 pixels square structuring element) masks to avoid confusion of objects near the separation lines. The advantage of this metric is that it penalizes the wrong number of objects, but tolerates minor pixel differences. On the other hand, the Dice score coefficient penalizes wrong object boundaries but does not evaluate the number of segmented objects. Some applications require accurate nuclei counting while others need their precise boundary segmentation. As the proposed approach is not bound to any particular application, we report and compare our results with both metrics.

### 3.2 Results

Tables 1 and 2 present the results of the proposed methods evaluated on the test set with the GT (the manual annotations of the dataset). We can clearly see that adding the DL improved the results compared to the classical segmentations that had been used as input for its training. The DL results when trained on the best of the classical segmentation configurations (Shape/Intensity) are not far behind the results when training the same network (and with the same hyper-parameters) on the manual GT. Therefore, we can conclude that training on the output of a classical segmentation method is a good way to avoid the tedious work of creating manual labels for training and validation sets. However, the test set still has to be fully annotated to allow a proper assessment of the DL performance.

We can also observe that, contrary to our initial expectations, the segmentation consensus gave worse results in training DL than the best segmentation configuration alone. One possible explanation is that a noticeable part of the training data was masked out, and thus, the network had to learn how to perform segmentation on a smaller number of cells. A smaller, less representative training dataset usually leads to poorer generalization; and that is what we believe happened here. This may lead to the conclusion that it is better to include noisy or imprecise annotations rather than removing them from the training dataset. However, this hypothesis has to be tested in a much larger study with multiple datasets, various types, and levels of data corruption, and hence, it is outside the scope of this paper.

### 3.3 Processing Time

The initial segmentation and the post-processing were done in CellProfiler 3.1.9 on a laptop computer (64 GB RAM, 2.90 GHz Intel® Core™ i7-6920HQ CPU, Windows 10 64 bit). The initial segmentation took about 5s/image and the post-processing took about 9s/image. These values are if run non-parallel, and only including the actual image processing – not saving, etc. However, in practice, this is automatically parallelized in CellProfiler. The neural network training and inference were done on a stationary computer (32 GB RAM, 3.30 GHz Intel® Core™ i9-7900X CPU, NVIDIA GeForce RTX 2060 SUPER 8 GB GPU, Windows 10 64 bit). Training time: ~ 43ms/update step, 140s/epoch (3200 image tiles), inference time: ~ 12ms/image tile, 2s/test set (200 image tiles).

## 4. CONCLUSION

We have demonstrated how classical image analysis methods can be combined with deep learning not only to reduce the tedious work of annotating images but also to improve the results offered by the classical pipelines. Our results show that it is possible to train a successful neural network without using any manually annotated ground truth; by using the output of a classical segmentation pipeline as training labels instead. Moreover, this approach gave a clear performance improvement compared to the classical methods that were used in generating the training data.

## 5. ACKNOWLEDGEMENTS

DJM received funding from the Swedish e-science initiative eSSENCE. PR is funded by the Swedish Science for Life Laboratory (SciLifeLab). The authors have no further relevant financial or non-financial interests to disclose.

| Method | Nuclei | Separations |
|---|---|---|
| *DL trained on GT* | *0.964* | *0.492* |
| DL trained on Shape/Intensity | **0.958** | **0.355** |
| DL trained on Consensus | 0.948 | 0.257 |
| Classical Shape/Intensity | 0.939 | 0.29 |

**Table 1.** Pixel-wise Dice coefficient before post-processing.

| Method | Jaccard index |
|---|---|
| *DL trained on GT* | *0.892* |
| DL trained on Shape/Intensity | **0.838** |
| DL trained on Consensus | 0.805 |
| Classical Shape/Intensity | 0.504 |
| Classical Intensity/Shape | 0.481 |
| Classical Shape/Shape | 0.464 |
| Classical Intensity/Intensity | 0.251 |

**Table 2.** Object-wise Jaccard index after post-processing.

## 6. REFERENCES

[Bor86a] Borgefors, G. (1986), Distance transformations in digital images, Comput.vision, Graphics, Image Processing, 34: pp. 334–371.

[Cai19a] Caicedo, J. C., et al. (2019). Evaluation of Deep Learning Strategies for Nucleus Segmentation in Fluorescence Images. Cytometry Part A, 95(9): pp. 952–965.

[Glo10a] Glorot, X., Bengio, Y. (2010) Understanding the difficulty of training deep feedforward neural networks. In Proc. of the 13th International Conference on Artificial Intelligence and Statistics: pp. 249–256.

[Gup19a] Gupta, A., et al. (2019) Deep learning in image cytometry: a review. Cytometry Part A, 95(4): pp. 366–380.

[Kin14a] Kingma, D.P., Ba, J.L. (2014) Adam: A method for stochastic optimization. In Proc. of the International Conference on Learning Representations (ICLR), arXiv:1412.6980

[Kla17a] Klambauer, G., et al. (2017) Self-normalizing neural networks. Advances in Neural Information Processing Systems: pp. 971–980.

[Lec15a] LeCun, Y., et al. (2015) Deep learning. Nature, 521(7553): pp. 436–444.

[Ljo12a] Ljosa, V., et al. (2012). Annotated high-throughput microscopy image sets for validation. Nature Methods, 9, 637.

[Mat19a] Matuszewski, D. J., Sintorn, I.-M. (2019) Reducing the U-Net size for practical scenarios: Virus recognition in electron microscopy images. Computer Methods and Programs in Biomedicine, 178: pp. 21–39.

[Mcq18a] McQuin C., et al. (2018). CellProfiler 3.0: Next-generation image processing for biology. PLoS Biology, 16(7): e2005970.

[Ron15a] Ronneberger, O., et al. (2015) U-net: Convolutional networks for biomedical image segmentation. In Proc. of the International Conference on Medical Image Computing and Computer-Assisted Intervention: pp. 234–241.

[Sez04a] Sezgin, M., Sankur, B. (2004), Survey over image thresholding techniques and quantitative performance evaluation. Journal of Electronic Imaging, 13(1): pp. 146–165.

[Sri14a] Srivastava, N., et al. (2014) Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1): pp.1929–1958.

[Vin91a] Vincent, L., Soile, P (1991)., Watersheds in digital spaces: An efficient algorithm based on immersion simulations. IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(6): 583598.

[Xin17a] Xing, F., et al. (2017) Deep learning in microscopy image analysis: A survey. IEEE Transactions on Neural Networks and Learning Systems, 29(10): pp. 4550–4.