SELF-TRAINING AND GENERATIVE DATA AUGMENTATION FOR SEMANTIC SEGMENTATION OF CELLULAR MICROSCOPY IMAGES

Spencer Hann

Oregon Health & Science University

ABSTRACT

It is often the case in machine learning generally, and even more so in deep learning specifically, that limitations are imposed the quality and quantity of data available for a given task. Generative adversarial networks provide promising methods for increasing the amount of data available. Additionally, self-training and other semi-supervised learning methods provide ways of using unlabeled data to boost performance. With this project, I explore the possibility of using generative modeling to create new training examples which can be used for pseudo-labeling in a self-training pipeline.

Index Terms— Semantic segmentation, self-training, generative adversarial networks, u-net, semi-supervised learning

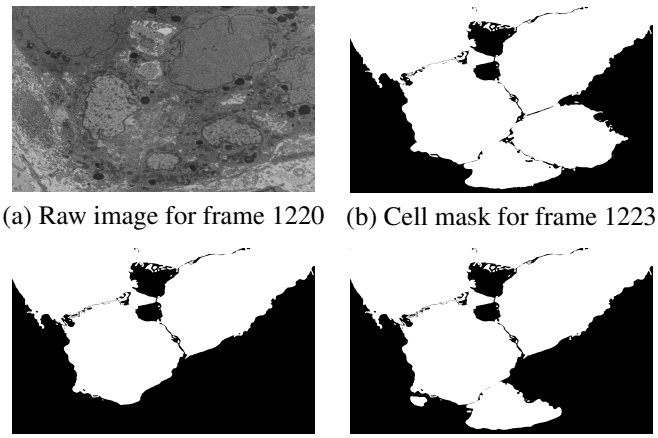
1. INTRODUCTION

A major challenge in machine learning for biomedical imaging is acquiring high quality annotated data. Data like segmentation masks are time and labor intensive for humans to produce, and can only be produced by experts. In this project, I attempt to overcome the difficulties of having a small amount of inconsistently and sparsely labeled data using self-training and generative data set augmentation.

2. CELLULAR MICROSCOPY DATA

This task is made more difficult by the fact that this data set is not only partially labeled, but that all individual training examples have incomplete annotations. Not only do individual images tend to have at least one unlabeled cell, some also have incorrect annotations, where a cell mask persists without change across frames, even as the underlying images change frame-by-frame.

Typically, when using self-training [1] [2] to train with partially labeled data, one has a set of labeled data (complete or near-complete labels) and a set of unlabeled data. The data pipeline can therefore be simplified by keeping these two subsets separate. Due to the inability to partition this data set in that manner, as well as the the size of the data set, for this project, I had to create a pipeline that involved dynamically loading and partitioning image crops.



(a) Raw image for frame 1220 (b) Cell mask for frame 1223

(c) Cell mask for frame 1224 (d) Cell mask for frame 1225

Fig. 1. From frame 1223 (b) to 1224 (c), the two lower cell masks disappear from the segmentation masks. In frame 1225 (d), the lower-center cell mask returns, but the lower-right does not. Many cells are missing annotations, sometimes for hundreds of consecutive frames, or sometimes even throughout the entire data set. There is an additional cell along the right side of the image that is not present in any of the cell masks (though its nucleus is present in the nuclei masks). Many segmentation masks in the training set are considerably sparser than these (see Figure 13).

Figure 1 shows some examples of incomplete segmentation masks. Some cells are not labeled in any frame in the entire data set, while some are only labeled in a subset of the total frames. An additional difficulty is that some cells are actually improperly labeled, with segmentation masks for certain cells placed over the wrong frames. This error is not easy to correct as these error are on a cell-by-cell basis, and do not necessarily invalidate the whole image or segmentation mask.

An additional difficulty in working with this data is the high degree of self similarity between frames. I do not have separate data to test to verify this, but I am not confident in the ability of my model to generalize well to other data sets, because of the low variance in the data with which it was trained.

3. SELF-TRAINING

Self-Training is an iterative semi-supervised learning method that makes use of unlabeled data to create pseudo-labels with which successive generations of models can be trained. For the first generation of models, only existing labeled data is used. These models are then combined to create and ensemble model of parent models. By training multiple models separately, and combining them into and ensemble, intergenerational error accumulation can hopefully be avoided.

For ensuing rounds, new models will be trained using a combination of preexisting labeled data and new data with pseudo-labels create by the parent ensemble from the previous generation. New models are always trained from scratch, without any from of transfer learning using weights from the parents. After a model has been used as a parent once, it is discarded. This helps to further prevent error accumulations. The results of parents are combined using an average voting method, to ensure that only pixels the majority of parents label positively are marked in the output mask.

In addition to being useful when a data set is only partially labeled, self-training can also be used to improve performance on fully labeled data set, but removing some labels or expanding the existing data with additional data from another data set [1].

4. DATA PIPELINE

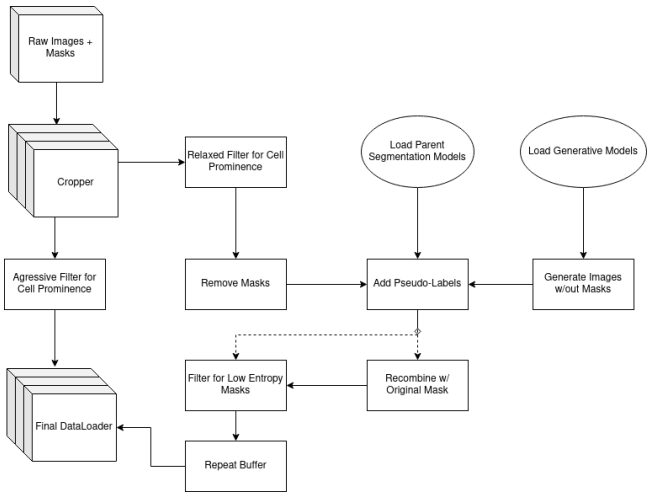


Fig. 2. Pipeline diagram showing all 3 data sources: Original labels, pseudo-labels, and generative pseudo-labels

Figure 2 shows the general strategy for dynamically creating training data, which includes plain labeled data, pseudo-labeled data created from the original data, and pseudo-labeled data created from generated cell microscopy images.

4.1. Main Pipeline

In the simplest case, training images and accompanying masks are loaded, scaled, augmented, and cropped. Then each crop is filtered for cell prominence, and discarded if it is below some threshold. Cell prominence is described in Section 4.4. Each image is high resolution (3511 x 5728), and the crops are square 256 x 256 images, so each loaded image produces numerous training crops. For the first round of self-training, this is the only active pipeline.

4.2. Pseudo-label pipeline

In the case of the main pseudo-label pipeline, image-mask pairs are similarly loaded, scaled, augmented, and cropped. Before having pseudo-labels applied, crops are first filtered for cell prominence. This is to prevent the parent ensemble from having too difficult a task; they were trained on cell-prominence-filtered segmentation masks. However, the cell prominence threshold is lower in the pseudo-label pipeline than in the main pipeline. This is to allow the parent ensemble a chance hopefully to segment cells that are left blank in the original data.

Pseudo-labels are then created for each crop, using the parent ensemble. After new segmentation masks are created, they may or may not be re-combined with the original mask.

Finally, images with high-entropy masks are discarded. Image entropy is defined as

$$\text{entropy}(X) = -\sum_{i=0}^{w-1}\sum_{j=0}^{h-1} X(i,j)\ln(X(i,j))$$

where w and h are width and height, respectively, and X is the segmentation mask. This serves as a measure of overall confidence for the masks produced by the parent ensemble. Note that an exact (binary) mask, has zero entropy.

The entropy threshold is calculated dynamically using an inertia-based running average,

$$T_t = \gamma \cdot T_{t-1} + (1-\gamma)\cdot \text{entropy}(X_{t-1})$$
$$T_0 = 0$$

, where T_t is the threshold at time step t, and γ is the inertia constant (by default, $\gamma = .99$).

Because generating pseudo-labels is extremely expensive, approved labels are stored in a repeat buffer. During training, with some small probability a new pseudo-label segmentation mask will be produced, and the oldest will be dropped from the buffer. Otherwise, a prepared pseudo-label will be sampled from the repeat buffer with uniform probability. This is important because, even without filtering (cell prominence or entropy), generating pseudo-labels dynamically would be time intensive. With filtering, it would be prohibitively expensive, because, for every passed pseudo-label, numerous

others are generated and discarded. This way the data that has already been accepted can be recycled, greatly improving the performance of this pipeline.

4.3. Generative pseudo-label pipeline

The generative pseudo-label pipeline feeds into the main pseudo-label pipeline by producing images from the generator (Section 6), and passing those generated images to the parent ensemble for mask pseudo-labeling.

The generator outputs images at 128 x 128 resolution, which are then scaled to 256 x 256 using bilinear upsampling. By only being required to produce lower-resolution images, the generator is ultimately able to create higher-quality fakes.

4.4. Cell prominence

In an attempt the limit the amount of pixel-wise false negatives in the ground truth data, I employ the use of a heuristic I call "cell prominence." A crop's cell prominence is simply the mean of it's cell mask (where cell masks are simply binary images). This provides the percentage of a crop that has been positively labeled. The cell prominence filter in the pseudo label pipeline is relaxed in later generations. In retrospect, the pseudo-label cell prominence filter should have been relaxed sooner. This would have allowed the parent ensembles to begin masking regions of images with missing masks sooner.

4.5. Additional data augmentation

Additional augmentations applied to loaded images include scaling, affine transformations, horizontal and vertical flipping, Gaussian blurring, and pixel-wise Gaussian Noise.

All images are scaled by a factor of 0.25, which drastically reduces the amount of information to train on but makes the task substantially easier by allowing more cell structures in each crop. For the generator/discriminator (Sections 6 & 7), input images are scaled by 0.125. Generator outputs are then upsampled by a scale factor of 2.

The random affine transformation applies rotation, slight scaling using either up or down sampling, and shearing, all applied with bilinear resampling. Affine transformations are not applied to the training data for the generator.

Flipping operations are applied randomly and independently with a uniform probability.

The blurring and noise augmentations are applied randomly and jointly, and only with a small probability, if at all.

5. U-NET

U-Net deep learning architectures are common-place for image segmentation tasks. For this cell segmentation task, I employ a somewhat standard U-Net architecture. Similar to an Autoencoder architecture, which progressively downsamples images while increasing feature depth, the U-Net architecture

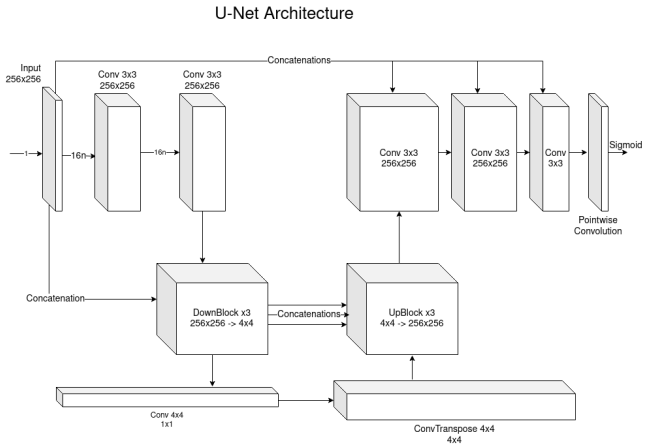


Fig. 3. Custom U-Net architecture. Note the additional convolutional layers at the input and output stages of the network, before the first down block and after the last up block. Also, not the additional concatenations sourced from the input image that are included in the first down block, and every output convolutional block.

forces information through a bottleneck in the middle of the network, where the image is represented as a one dimensional latent vector, before progressively upsampling to an output image. Unlike Autoencoders, however, the U-Net uses additional connections to bridge the gaps between corresponding downsampling and upsampling stages.

Figure 3 shows the U-Net architecture, while abstracting away the details of the down and up blocks. Detailed diagrams for those blocks can be found at the end of this paper in Section 13.

Each down block consists of two 3 x 3 convolutions, followed by a strided convolution, which reduces the height and width of the feature map by a factor of 2. The output of the strided convolution is cached and re-used in the concatenation layers of the corresponding up block. The strided convolution is followed by a max pooling layer and a dropout layer. Taking into account the strided convolution and the max pooling layer, each down block scales the feature maps height and width by $1/4$.

The up block starts by upsampling the input feature map, which is concatenated with the feature map produced by the corresponding up block. This concatenated feature map is then fed through two plain convolutional layers, which leads into a second concatenation layer, re-using the same cached feature map from the corresponding up block. Finally, the output layer is a transpose convolutional layer, which performs additional up-scaling. With the non-learned upsampling at the beginning of the up block and the transpose convolution at the end, each up block scales the input feature map by a factor of 4.

All convolutional layers in the up and down blocks, use batch norm and a Gaussian Error Linear Unit (GELU) acti-

vation function. The convolutional layers in the input and output stages use GELU activations, but no batch norm. The final output layer is a point-wise convolution followed by a Sigmoid activation.

6. GENERATOR

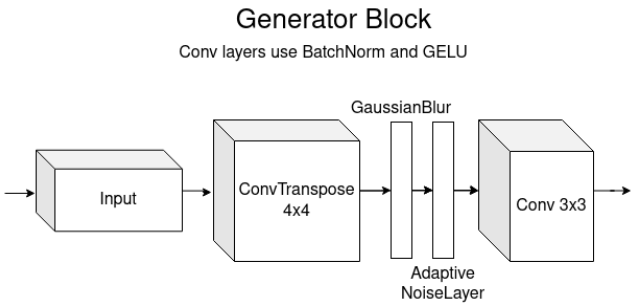


Fig. 4. Custom Generator architecture. Note the Gaussian blur and adaptive noise layers [3] [4] following the upsampling layer.

I experimented with many different generator architectures through out the course of this project, from simple vanilla GANs [5] [6], increasing in complexity to StyleGAN models [3] [4]. I ultimately used a model more similar to those used in simple GANs, but with elements borrowed from the StyleGAN model. Specifically, I employ Gaussian blurring after the upsampling operation, as well as an adaptive noise layer. Unlike the StyleGAN model, I use a strided transpose convolution in lieu of bilinear upsample. This special convolutional layer can be thought of as a method of adaptive upsampling. The Gaussian blur helps to smooth any artifacts that upsampling may leave behind, and the adaptive noise layer helps to continue introducing stochasticity into the network at every stage, which allows for more fine grained detail variation. The input to the network is simply a 1-dimensional random normal vector.

Every convolutional layer, including the transpose convolution, is followed by an instance norm layer and a GELU activation function.

7. DISCRIMINATOR(S)

Through out training I struggle to train generators to produce output images with large-scale structures, i.e. cell walls, nuclei, mitochondria. While generators were generally able to create texturally passable images, many were clearly faked as they lacked these larger identifiable features. To deal with this issue I made adjustments to the discriminator architectures rather than the generator itself.

My first attempt was to add dilated convolutions to the discriminators, increasing each discriminator's receptive field

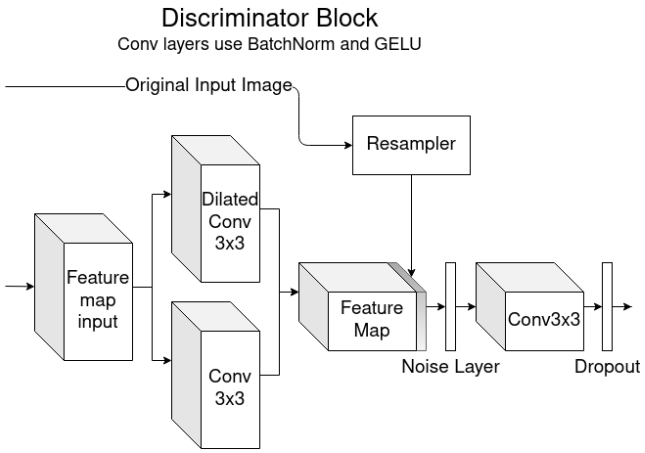


Fig. 5. Custom Discriminator architecture. Note the resampling layer, which allows the discriminator to re-evaluate the original input at various scales and network depths.

earlier in the network. This enables the discriminators to better discern large-scale structures, which in turn incentivizes the generator to produce those structures. While this had the desired effect to some degree, it also would lead the generator to create strange pixelated adversarial patterns that did not look natural on a small scale. To solve this problem I created a discriminator architecture that used dilated and non-dilated convolutional layers in parallel, allowing the discriminator to check for small scale textural qualities while still maintain its increased receptive field.

An additional strategy was to add resampling concatenation layers to every down block in the discriminator. The "resampler" layer would cache the input image and concatenate a downsampled copy at every layer than involved reducing the height and width of the feature map. This way the discriminator was able to view the input image at multiples scales, and at multiple network depths. This approach had a strong effect in improving the discriminators' ability to discern large-scale structures.

8. GANS TRAINING METHODOLOGIES

I experimented with several GAN loss approaches during my experiments: Wasserstein loss [7], Wasserstein loss with gradient penalty [8], non-saturating loss (traditional GANs loss) [5], and non-saturating loss with gradient penalty [9].

My final generator was trained using the final training method, non-saturating loss with gradient penalty. I found this approach led to the most reliable training stability as well as helped to prevent issues I had been having with mode collapse. In an additional effort to prevent mode collapse, or the convergence to specific adversarial patterns, I also train each generator against an ensemble of discriminators.

Generated examples can be viewed in Section 13.

9. METRICS

To evaluate my final model I employ four metrics: Dice coefficient, Jaccard index, precision, and recall. The Dice coefficient and Jaccard index are both take into account the intersection of positively labeled pixels. Note that this is the set of all true positives.

The Dice coefficient is defined as

$$D(\hat{X}, X) = \frac{2|\hat{X} \cup X|}{|\hat{X}| + |X|}.$$

The Jaccard index is defined as

$$J(\hat{X}, X) = \frac{|\hat{X} \cup X|}{|\hat{X} \cap X|}.$$

In the above formulations X and $\hat{X}$ are sets of positively labeled pixels. However, it is possible to reformulated these metrics in terms of true positives, false positives, and false negatives:

$$D = \frac{2TP}{2TP + FP + FN}$$
$$J = \frac{TP}{TP + FP + FN}$$

Here it is plain to see the similarity of the two metrics, with the distinction that the Dice coefficient weights true positives more heavily than other classification outcomes. Because the test and validation sets have more complete cell masks than missing, I expect the Dice coefficient to be much higher relative to the Jaccard index on these subsets than on the training set which has many missing masks, which will cause greater false positive rates (where false positives are the results of correctly segmenting cells with missing masks).

In addition to these intersection-style metrics, recall is particularly useful to measuring success in this task. This is because high recall scores demonstrate that a model correctly determines which pixels actually are included in the ground truth masks. However, the recall score is not affected in the cases where a model incorrectly (or correctly in cases of missing annotations) marks a pixels as part of a mask.

For these reasons, I look for results that show recall and Dice coefficient scores which are high relative to their corresponding precision and Jaccard index scores. This shows the model capably learning to reproduce masks that are present in the training data, while hopefully also beginning to learn to account for missing masks as well.

10. RESULTS

Note that in Figure 10, recall scores are much higher than precision scores. This is because precision scores take false

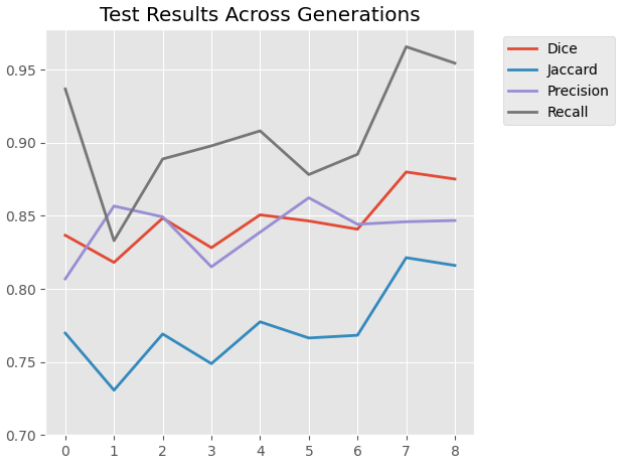


Fig. 6. Results on test set

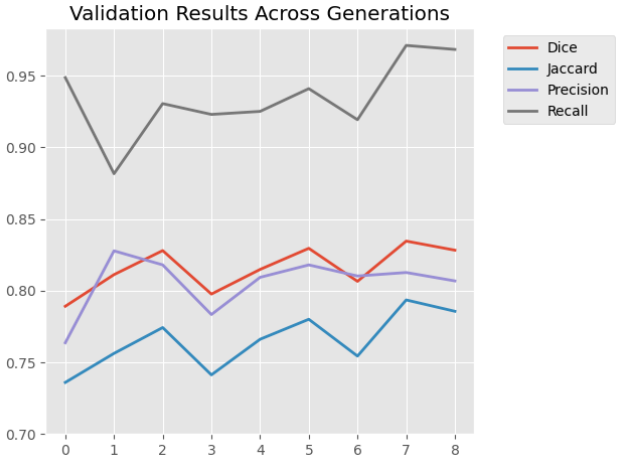


Fig. 7. Results on validation set

Fig. 8. Results on validation set

positives into account but the actual ground truth data contains numerous false negative (missing cell masks). A high recall score shows that a model rarely misses a masked pixel, whereas a comparatively low precision score could indicate that the model is correctly masking pixels that are in fact missing from the original segmentation masks.

Actual model outputs can be viewed in Section 13. These images show that the model is segmenting existing cell masks as well as missing cell masks.

Not that the models perform markedly better on the test set, than on the validation set, and better still than on the training set. This is because the test set has the fewest false negatives (missing masks). However, even the test set was not perfectly complete. I believe that with fully labeled data the final model would test better still.

11. MY CONTRIBUTION

To my knowledge, self-training is typically employed to learn from data sets that are evenly divided between a fully labeled subset and an unlabeled subset. My approach introduces the idea of using a heuristic to dynamically sub-divide the data set during each round of training on a case-by-case basis. Additionally, my approach is novel in its supplementation of real training data with generated training examples, which allow parent ensembles to better pass knowledge to future networks. Together these approaches deal with the problems of having too little data and too few annotations for those data. Perhaps most importantly, this project has the potential to take a poorly labeled data set and improve the quality of the annotations for future users.

12. REFERENCES

[1] Barret Zoph, Golnaz Ghiasi, Tsung-Yi Lin, Yin Cui, Hanxiao Liu, Ekin D. Cubuk, and Quoc V. Le, "Rethinking pre-training and self-training," 2020.

[2] Yauhen Babakhin, Artsiom Sanakoyeu, and Hirotoshi Kitamura, "Semi-supervised segmentation of salt bodies in seismic images using an ensemble of convolutional neural networks," 2019.

[3] Tero Karras, Samuli Laine, and Timo Aila, "A style-based generator architecture for generative adversarial networks," 2019.

[4] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila, "Analyzing and improving the image quality of stylegan," 2020.

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative adversarial networks," 2014.

[6] Alec Radford, Luke Metz, and Soumith Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2016.

[7] Martin Arjovsky, Soumith Chintala, and Léon Bottou, "Wasserstein gan," 2017.

[8] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville, "Improved training of wasserstein gans," 2017.

[9] William Fedus, Mihaela Rosca, Balaji Lakshminarayanan, Andrew M. Dai, Shakir Mohamed, and Ian Goodfellow, "Many paths to equilibrium: Gans do not need to decrease a divergence at every step," 2018.

13. APPENDIX - MISCELLANEOUS FIGURES

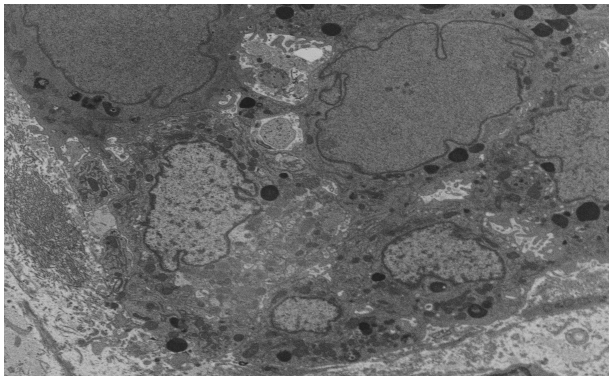
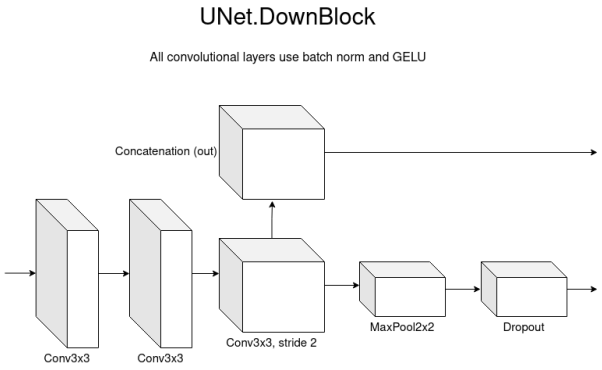
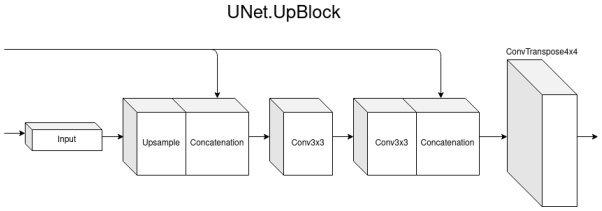


Fig. 9. Frame 1224, from Fig 1, resized for detail.



(a) Down block, repeated in U-Net architecture 3 times.



(c) Up block, repeated in U-Net architecture 3 times.

Fig. 10. Core building blocks for the down and up-sampling stages in the U-Net architecture. Note, the down block has a strided convolutional layer as well as a max pooling layer, meaning that each down block scales the image/feature map by a factor of $1/4$. Inversely, the up block has both and up-sampling layer (bilinear), and a transpose convolution, meaning that each up block scales the image/feature map by a factor of 4. Also, note the dual concatenation layers in the up block, immediately after the upsample and again after the internal convolutions.

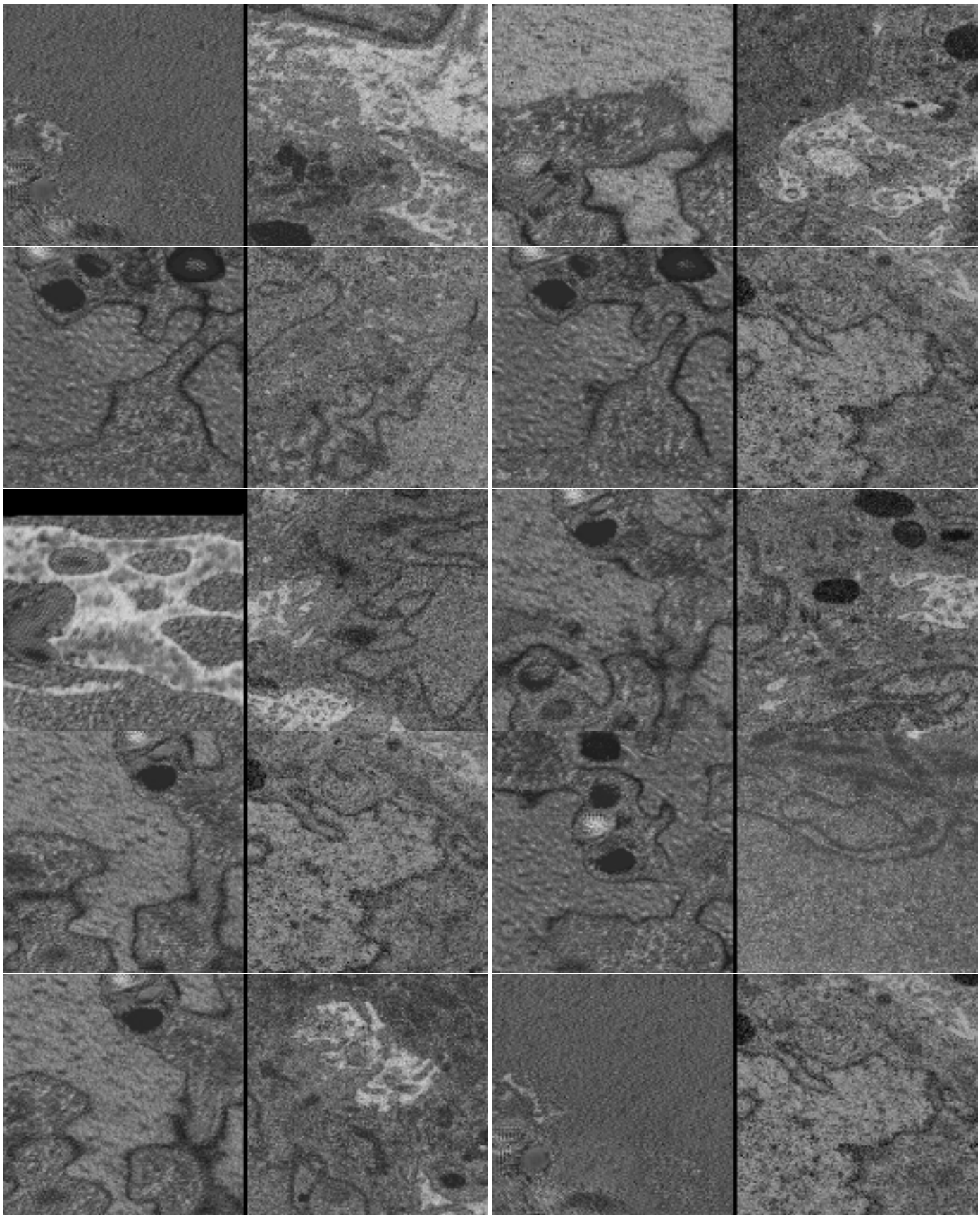


Fig. 11. Images on the left are generated. Images on the right are cropped from the original dataset. Some instances of mode collapse are still apparent. (Randomly selected)

Fig. 12. Additional generated and ground truth images. (Not cherry-picked – randomly selected)

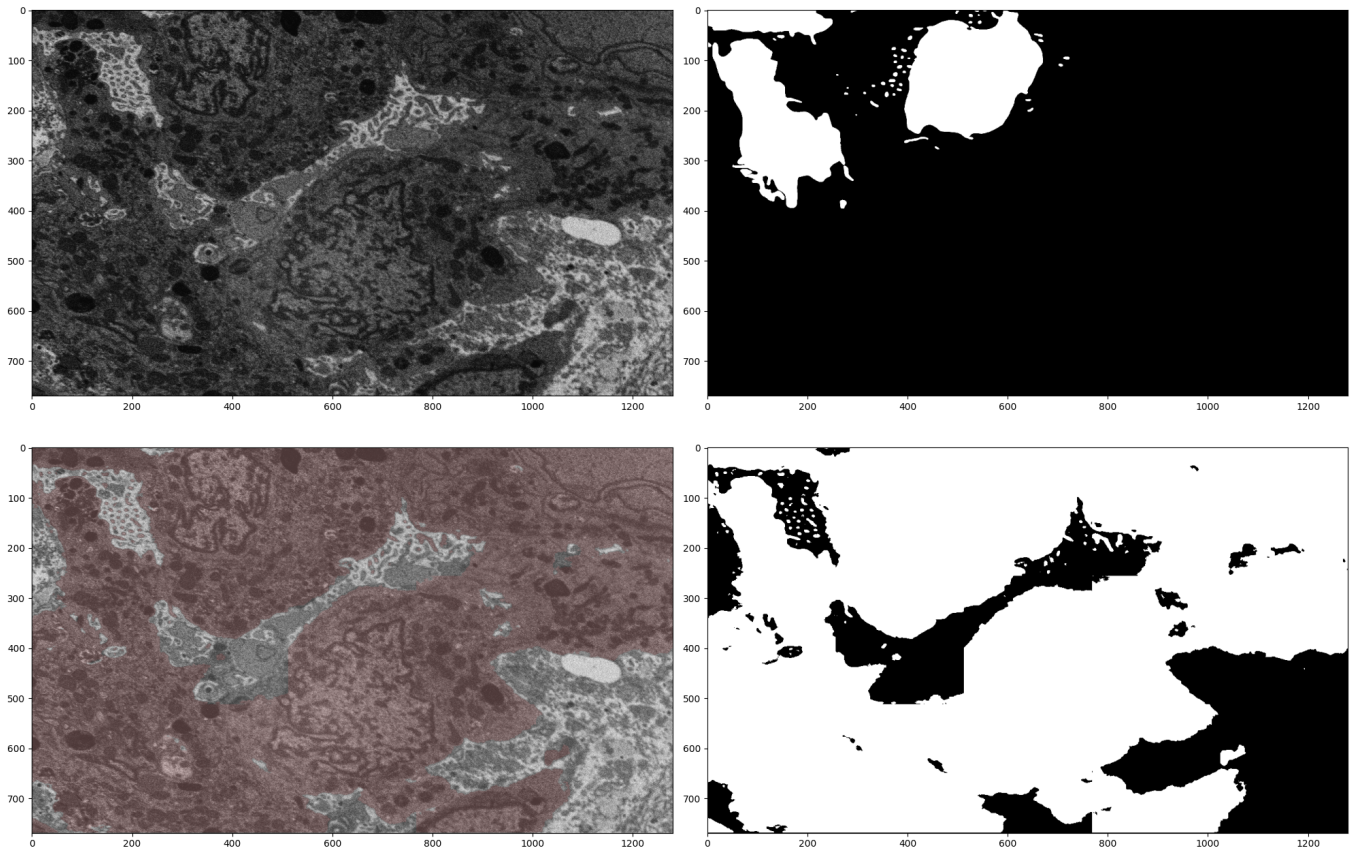


Fig. 13. This image shows the models capability to fill in missing masks, that it was not able to train on. It was taken from the training set. It was randomly selected from a subset of the training data I knew to have very sparse training masks. (upper-left) Raw cellular microscopy image; (upper-right) Original cell masks; (lower-right) Output cell masks; (lower-left) Output cell masks super-imposed over original image;

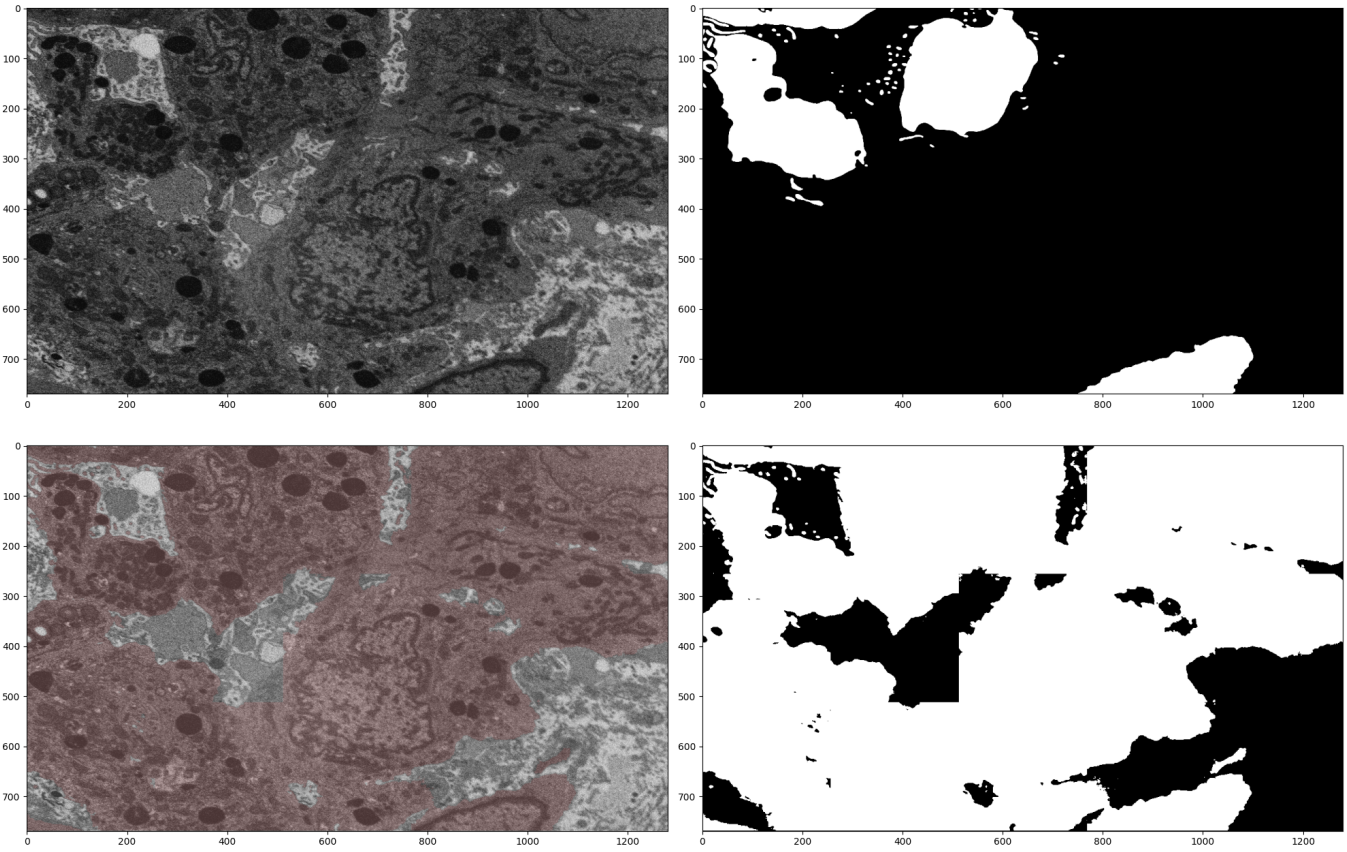


Fig. 14. This image was taken from the training set. It was randomly selected from a subset of the training data I knew to have very sparse training masks. (upper-left) Raw cellular microscopy image; (upper-right) Original cell masks; (lower-right) Output cell masks; (lower-left) Output cell masks super-imposed over original image;

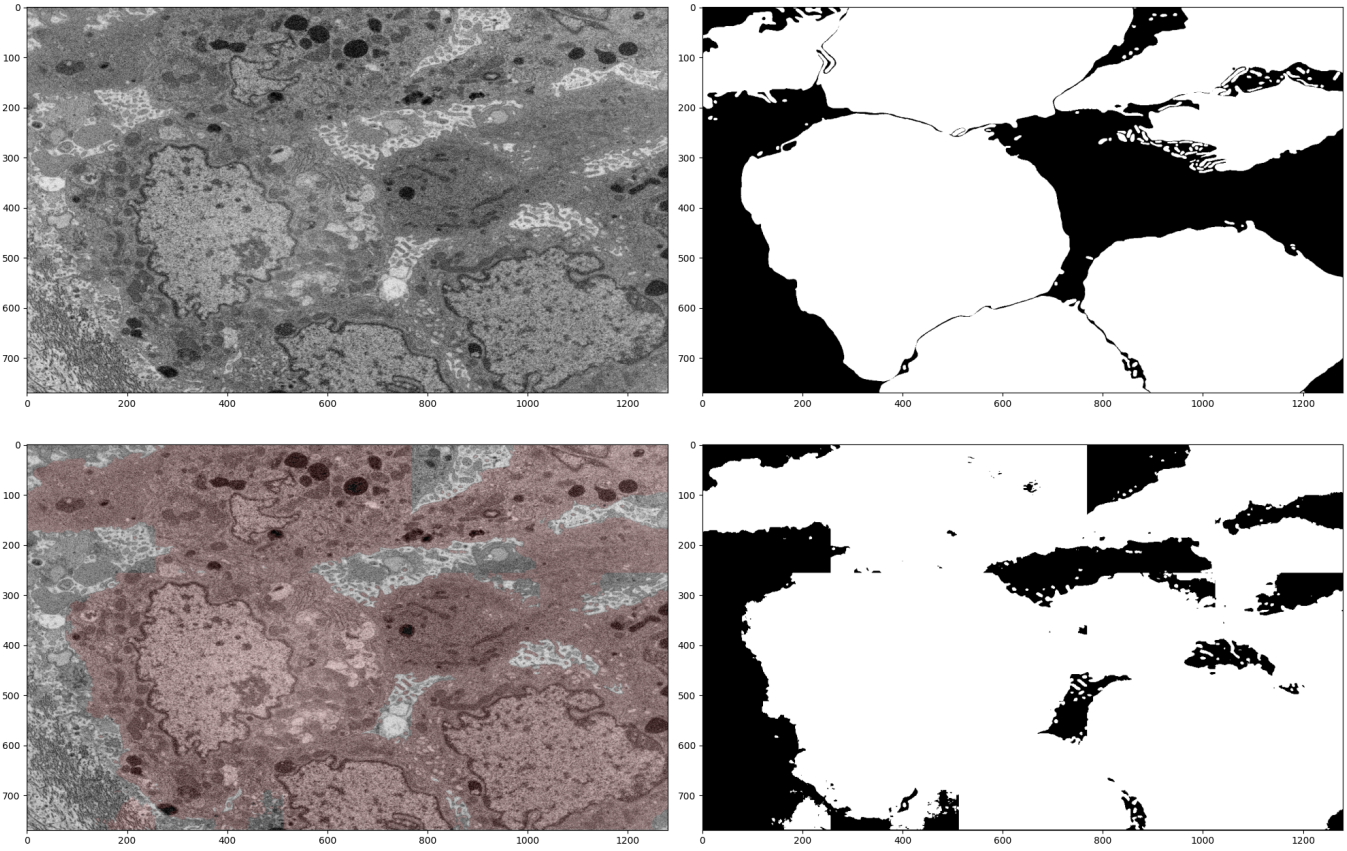


Fig. 15. This image was randomly selected from the test set. (upper-left) Raw cellular microscopy image; (upper-right) Original cell masks; (lower-right) Output cell masks; (lower-left) Output cell masks super-imposed over original image;

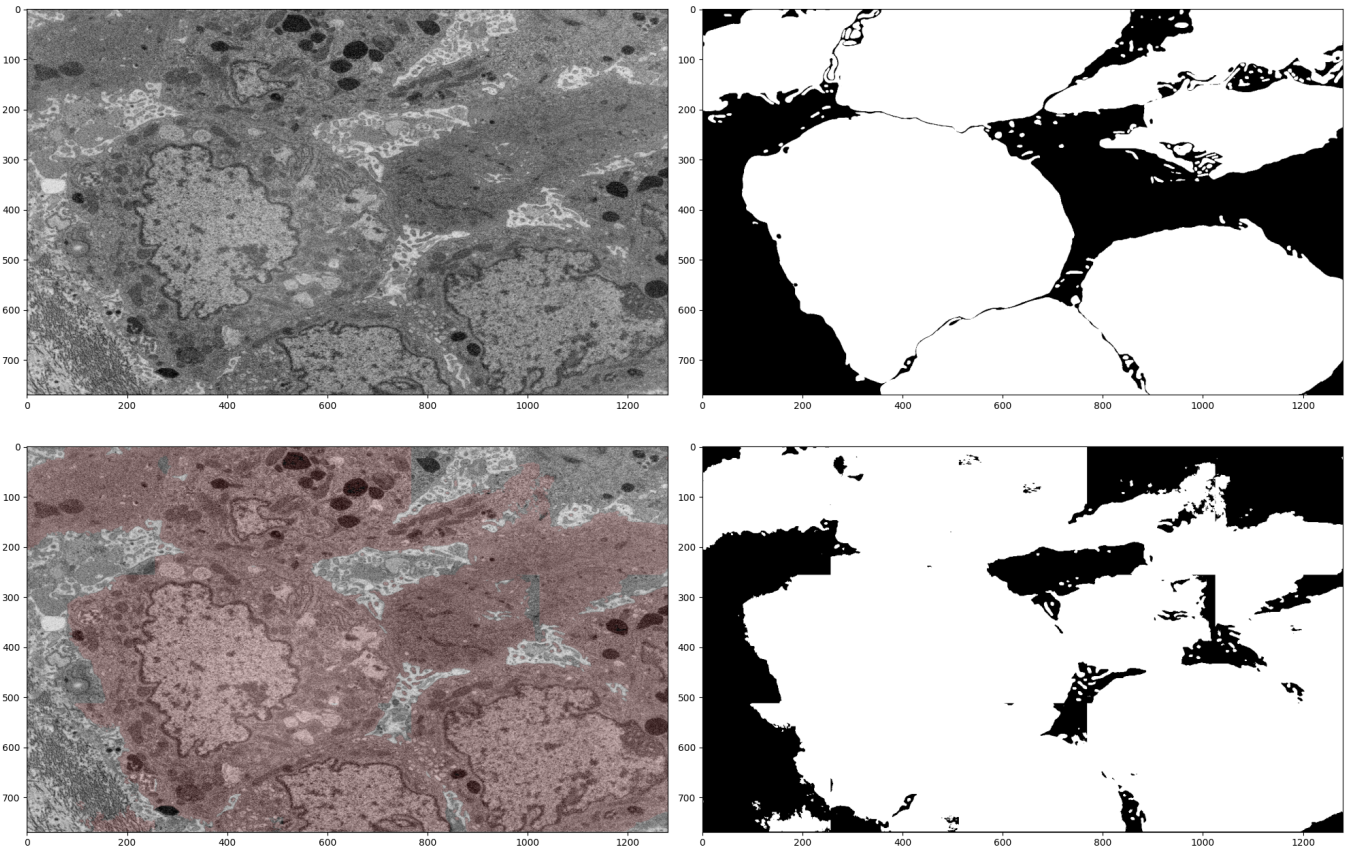


Fig. 16. This image was randomly selected from the test set. (upper-left) Raw cellular microscopy image; (upper-right) Original cell masks; (lower-right) Output cell masks; (lower-left) Output cell masks super-imposed over original image;