U-Net Model Implementation for Pancreatic Stem Cells and GFP-GOWT1 mouse stem cells

Lexi Henrion (ehenrion) Valeria Quero (vquero)
Serena Agarwal (sagarw56)

1 Introduction

For our Deep Learning final project, we chose to implement a cell segmentation model based on the paper "U-Net: Convolutional Networks for Biomedical Image Segmentation" by Ronnenberger et al. The authors of this paper trained and tested their U-Net model on two different Cell Tracking Challenge 2D datasets: "HeLa cells on a flat glass", and "Glioblastoma-astrocytoma U373 cells on a polyacrylamide substrate". We implemented a similar U-Net, but training and testing instead on the 2D datasets "Pancreatic stem cells on a polystyrene substrate" and "GFP-GOWT1 mouse stem cells", which we also obtained from the Cell Tracking Challenge.

Cell segmentation is a technique for defining and distinguishing individual cells' boundaries in microscopic images. The segmentation of cells allows for the analysis of individual cell characteristics, such as morphology and motility. Cell segmentation has a wide range of useful applications, including aiding in diagnosing disease by identifying certain cell characteristics, analyzing cells' response to drugs by measuring changes in cell shape and function, and the study of how diseased cells may behave differently from healthy, or wild-type cells. There are, however, various challenges to successfully segmenting cells, such as overlapping cells, low image quality, or varying morphology between cells. Our model seeks to segment cells with the best quality and accuracy possible given these limitations.

The paper we studied for our project introduced an innovative model to address these challenges. The U-Net is a "fully convolutional network", meaning that it does not have any fully-connected layers. Avoiding dense layers reduces the number of parameters, making the U-Net model more efficient; the architecture from the paper we chose was able to segment a 512×512 image in less than one second on a GPU. A U-Net architecture consists of a contracting path and an expansive path. The contracting path captures contextual information while reducing the spatial dimensions of the image and increasing its depth. The expansive path, on the other hand, takes the feature map from the end of the contracting path and uses upsampling layers to increase the spatial resolution while reducing the number of channels. At each upsampling layer, there is a

skip connection, which is the concatenation of a previous feature map from the contracting path to the current one in the expansive path. This process is to restore the features that may have been lost during the downscaling process of the contracting path. The output map of the U-Net is a binary segmentation map where each pixel represents either foreground or background.

2 Methodology

2.1 Datasets

We used the following datasets from the Cell Tracking Challenge to train and test our data: Pancreatic stem cells on a polystyrene substrate (`https://celltrackingchallenge.net/2d-datasets/`) and GFP-GOWT1 mouse stem cells (`https://celltrackingchallenge.net/2d-datasets/`).

2.2 Preprocessing

An important but easily overlooked aspect of preprocessing was upping the contrast on all of the input cell images to blow out the cells' whites and clarify the boundaries. Our `preprocess_data` function processes each TIFF frame in the dataset with `rasterio` (so we keep the full 16-bit range, and this is a good alternative for georeferencing that worked for our data), reads its matching manual mask that has been man-labeled. Finally, it applies a per-frame stretch by clipping the darkest 2% and brightest 2% of pixel values, rescale the rest to [0, 1], and convert each mask to a binary map. This produced two NumPy arrays of shape (N, H, W, 1). During training, our `get_batch` generator shuffles and slices those arrays into mini-batches then applies two classic U-Net augmentations on the fly:

- An elastic deformation that is intended to make the model learn the cell's shapes and textures rather than just their positioning (3×3 control grid, Gaussian blur)
- Random 180° rotations plus horizontal/vertical flips, always using the same random transform on both image and mask to preserve alignment

As in the paper, we get an infinite variety of training samples with zero disk I/O.

3 Architecture

We followed the same U-Net architecture from the paper we chose, but we implemented it using TensorFlow and we trained and tested our model on different datasets. We also consulted the medium blog post "The U-Net for cell segmentation in PyTorch" which provided a helpful framework and PyTorch implementation.

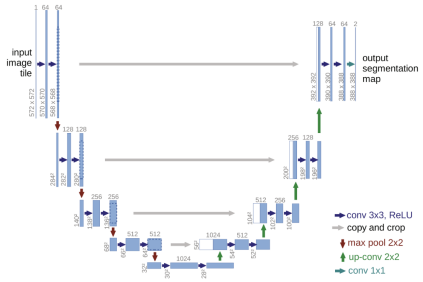

Figure 1: The U-Net architecture from Ronnenberger et al. (2015).

The above image is the architecture from the paper we chose. We followed this framework exactly, although the code itself is different. We started by feeding our image input into a DoubleConv block, which applies two sequences of the following operations: a 2D convolutional, batch normalization, ReLU activation, and a dropout layer.

After the initial DoubleConv, we passed the feature map through four Downscale blocks. Each Downscale block consists of a max-pooling operation to reduce the spatial dimensions of the feature map by 50%, followed by another DoubleConv. Each Downscale step reduces the spatial dimensions of the feature map, while increasing its depth, or number of feature channels.

We then fed our feature map through the expansive path, consisting of four Upscale blocks. Each Upscale block starts by performing bilinear upsampling to increase spatial resolution, then concatenates it with the corresponding feature map from the contracting path. This process is called a skip connection. It finally applies another DoubleConv. Each Upscale step increases the spatial dimensions of the map, while reducing the depth of the feature map, and decreasing the number of channels. We chose to use bilinear interpolation for our upsampling to restore spatial resolution to our spatially increased images without adding additional trainable parameters.

After four Upscale steps, we finally applied a final 1×1 convolutional layer to map the feature map to the desired number of output classes, producing the final predicted segmentation mask.

4 Training

The model is compiled using the Adam optimizer with a learning rate of 1×10^{-4}, and a custom loss function consisting of a combination of weighted binary cross entropy and Tversky loss. The weighting of the loss is vital in order to "punish" the model more heavily for false negatives; otherwise, it will learn to output a fully black mask, since that easily gives us 93% accuracy.

As we train (using `model.fit`), we simultaneously evaluate our model using

the Dice coefficient, a measure of similarity between the ground truth masks and our output masks, in order to save the best seen model at each epoch using `tf.keras.callbacks.ModelCheckpoint`. Other callbacks we include are early stopping (what it sounds like – stop if the model doesn't improve for a significant number of epochs), and reducing the learning rate upon a plateau (if the model plateaus for a sufficient number of epochs, we reduce the learning rate in order to gently encourage it to stabilize around the minima). These callbacks serve to keep the model improving, even if gradually, since improvements of just 1% can have a huge impact on the quality of the output masks in such a delicate task as cell segmentation.

After 20 epochs, we consistently achieve about 97% validation accuracy, and after 50, over 98%.

5 Results

We evaluated our U-Net model's performance on two independent datasets: pancreatic stem cells and GFP-GOWT1 mouse stem cells. Qualitatively, the model accurately delineated individual cell boundaries even in regions with overlapping cells or low contrast.

5.1 Pancreatic Stem Cells

After 50 epochs of training on the pancreatic stem cell dataset, the U-net achieved:

- Training loss: 0.4659
- Training Dice coefficient: 0.9850
- Validation loss: 0.4196
- Validation Dice coefficient: 0.9865
- Highest validation binary accuracy: 0.9865

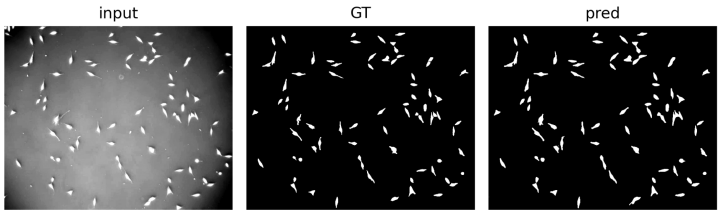

Figure 2: Segmentation result for pancreatic stem cells after 50 epochs.

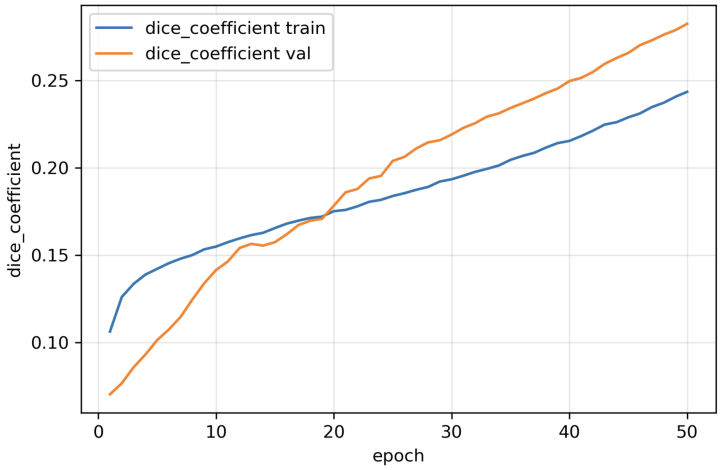

Figure 3: Training and validation Dice coefficient curves over 50 epochs for the pancreatic stem cell dataset.

5.2 GFP-GOWT1 Mouse Stem Cells

After 20 epochs of training on the GFP-GOWT1 mouse stem cell dataset, the u-net achieved:

- Training loss: 0.3620
- Training Dice coefficient: 0.4130
- Validation loss: 0.2503
- Validation Dice coefficient: 0.5643
- Highest validation binary accuracy: 0.9844

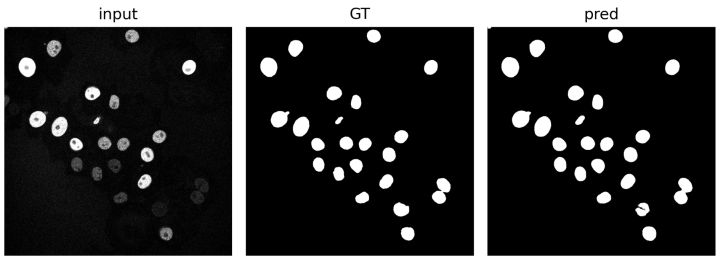

Figure 4: Segmentation result for GFP-GOWT1 mouse stem cells after 20 epochs.

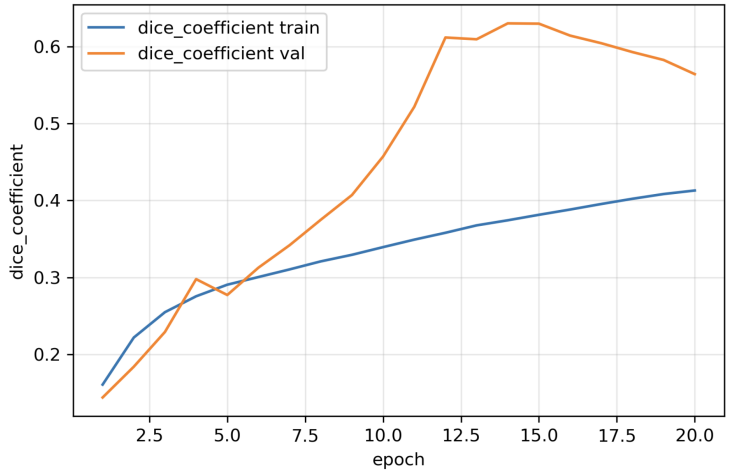

Figure 5: Training and validation Dice coefficient curves over 20 epochs for the GFP-GOWT1 mouse stem cell dataset.

For more relevant figures, please see the results (`https://github.com/serenaagarwal/dl-final-project/tree/main/results`) folder on our repository.

6 Challenges

Our funniest (and most problematic) challenge was that initially, we were getting between 93 and 94% accuracy, which sounds great until we realized that our positive class (the cells) made up between 6 and 7% of the images themselves, and our model was simply learning that there were no cells in any of the images. Even though Dice loss is a commonly used loss function for cell segmentation, and we actually assumed it to be better than the simpler approach outlined in the paper, since it equally weights false positives and false negatives, it is only effective when the classes do not contain a heavy imbalance, as in our data. Tversky loss can be considered a weighted form of Dice loss, allowing us to weigh false negatives more heavily than false positives. We combine this with weighted binary cross entropy for finer pixel-wise classification. On top of that, our augmentations nearly spoiled the training. We initially set the elastic deformation amplitude up too high ($\alpha > 1000$) and used too little Gaussian smoothing ($\sigma < 30\,\text{px}$), which produced some "tears" and folds in the cell images.

Our initial training was extremely slow. We used the same numbers of filters/parameters in the U-Net model as the original paper, where their training time was about 10 hours. This was difficult to work with and debug, so we tried cutting down on the number of filters we used, beginning with 16 instead of 64.

This dramatically reduced training time from about 15 minutes per epoch to about 5. It seemed to still effectively train, so we kept it. However, the model still took about two hours to run 50 epochs, so we ran out of time to truly see where the model would plateau. It's possible we would have to run it for 10 hours, but we may be able to break 99%!

7 Reflection

Our project was challenging, but we were satisfied with the results and what we were able to accomplish. We achieved our base and target goals– our base goal was to have a working implementation of a cell segmentation model and our target goal was to reach at least 98% accuracy on our model. We ended up achieving these goals, with a final accuracy of 98.65% for the pancreatic stem cell dataset and 98.43% for the mouse stem cell dataset. Our stretch goal was for our model to be generalizable to different cell datasets, and to include either tracking or quantitative analysis of certain metrics such as cell volume. We were able to successfully reach a ¿98% accuracy on two vastly different cell types: "Pancreatic stem cells on a polystyrene substrate" and "GFP-GOWT1 mouse stem cells". However, we were not able to extract quantitative metrics from our data, although it is a direction we would be interested in taking our project in the future. Our model did not work out exactly the way we expected it to, as we ran into challenges with our model outputting blank segmentation masks. We felt that our U-Net implementation followed the same framework as that of the U-Net from our paper, but we were not successfully outputting masks. This challenge was a significant hurdle for us, and though we were ultimately able to overcome it and make successful changes to our model, it was an unexpected obstacle.

When we were reimplementing the paper, we weren't thinking about how different cell types and different cell images would need to be handled differently. In our case, if we had been working with large cells taking up about 50% of space on our images, our initial approach likely would have worked fine, but that wasn't the case. Likewise, our model might not do as well on samples with cells taking up a majority of the space on the input images, since our weighting is designed for the other way around. We learned a lot about how to tune a model to our specific needs using the convenience of a library like Keras alongside custom implementations of aspects we need control over, like our loss. Our thinking became less "one size fits all." If we started the project over again, I think we would be able to notice earlier on how quirks of our data might impact how we want to create our model and build around that initially, rather than scrambling to adapt. We also noticed that our augmentations did not cause a significant change in performance, perhaps because the original paper's dataset featured one huge cell filling most of the frame, whereas we work with many smaller cells in each image. Thus, with more time, we could have tested this dataset on their same set and draw better conclusions as to what loss functions, preprocessing steps, and augmentation strategies are most effective for different

cell morphologies.

If we had more time, we would work on improving our validation accuracy even more, aiming for ¿99% accuracy. We would also work on extracting different metrics from our segmentation masks, such as cell area, x and y location of the cell nucleus, and other morphological metrics. We would also implement a tracking algorithm to be able to track the movement of the cells. We would extract data such as total distance moved (measured from the nucleus), cell velocity, and other metrics to evaluate the behavior of cells.

Our biggest takeaway from this project was the understanding we gained of how cell segmentation models work, and why they are effective. Before this project, we had incredibly minimal knowledge of the mechanisms behind cell segmentation, and we learned that, while challenging, coding a U-Net from scratch is indeed possible. We were proud to be able to construct a U-Net model with very minimal background knowledge of the framework, and we took away from this project that many more deep learning concepts are within our reach than we initially thought. We are looking forward to applying our newfound knowledge to future deep learning projects.

References

[1] Ronneberger, O., Fischer, P., Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In: Navab, N., Hornegger, J., Wells, W., Frangi, A. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. *Lecture Notes in Computer Science*, vol. 9351. Springer, Cham. https://doi.org/10.1007/978-3-319-24574-4_28

[2] Hansen, Bjorn. The U-Net for Cell Segmentation in PyTorch, Medium, 30 Dec. 2020. https://medium.com/codex/the-u-net-for-cell-segmentation-in-pytorch-d34dddcdaccb. Accessed Apr. 2025.

[3] Hesaraki, Saba. Tversky Loss, Medium, 27 Oct. 2023. https://medium.com/@saba99/tversky-loss-902f5f8cc35f. Accessed Apr. 2025.