Breast Cancer Nuclei Segmentation on the SNOW Dataset

Mikey Joyce
Department of Electrical Engineering and Computer Science
University of Missouri
Columbia, MO, USA
mpjyky@umsystem.edu

Filiz Bunyak Ersoy
Department of Electrical Engineering and Computer Science
University of Missouri
Columbia, MO, USA
bunyak@missouri.edu

I. Introduction: Motivation and Background

Computer vision (CV) in recent years has seen many advances in the field, going from a classical computer vision approaches to more new age methods such as various deep learning (DL) algorithms. Many of these methods, classical and deep learning alike, have various applications over a spectrum of different industries and problems. One major research problem that computer vision researchers tackle is biomedical image processing tasks. There are various different domains within this computer vision subset. For example, researchers focus on tasks such as cell segmentation, tumor detection, and organ classification, addressing critical challenges in medical diagnostics and contributing to advancements in healthcare technologies. Particularly, semantic segmentation of cell images is a big challenge for researchers to generalize to a large subsets of images. Overcoming variations in cell morphology, staining techniques, and imaging conditions poses significant hurdles, requiring innovative approaches to enhance the robustness of segmentation algorithms. Breast cancer cells can have many different morphological structures that can make it difficult to segment. The difference between different cancers also makes it hard to generalize as the features of ovarian cancer cells may not be the same features of breast cancer cells, which is a large hurdle for semantic segmentation of cancer cells.

II. Image Processing Data, Formatting, and Pipeline

A. The SNOW Dataset

This paper will utilize the Synthetic Nuclei and annOtation Wizard (SNOW)[6]. The SNOW dataset contains 20,000 synthetic images (Fig. 2.1) of breast cancer nuclei and 20,000 matlab files that contain corresponding masks (Fig. 2.2) of these breast cancer nuclei. The breast cancer nuclei images were generated from a StyleGAN2 image generator [6] that was trained on the BreCaHAD [1] dataset, which contains many images of real breast cancer cells. The masks for the generated synthetic images were obtained using an automatic annotator called HoVer-Net [1]. All-in-all the SNOW dataset was nearly 150 GB when unzipped, as the matlab files contained within were very large.

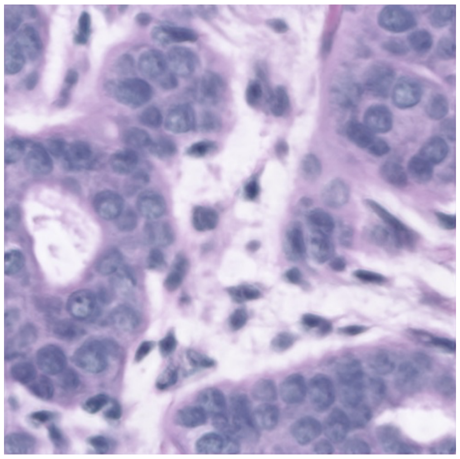


Fig. 2.1: Example image in the SNOW dataset

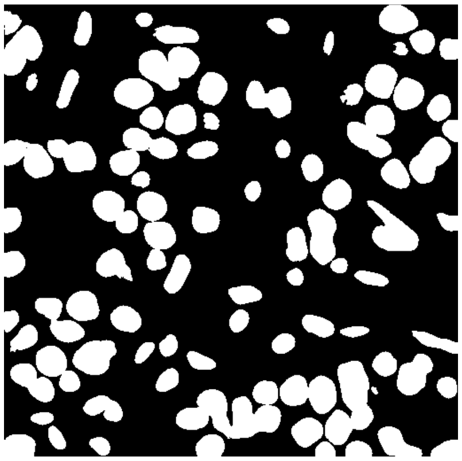


Fig. 2.2: Corresponding mask for Fig. 1

B. Data Formatting Pre-Pipeline

Before the pipeline begins it is important that the data that is being utilized is formatted in a uniform manner to minimize errors. For instance, the masks were broken up into four different directories, due to their massive size, while the images were contained within only one directory. So, the images were also broken up into four different directories containing 5,000 images each; this would prove useful later in the project as Google Drive struggles to even upload 5,000 images at once. The files within the dataset had a strange naming convention that was not consistent. Due to this the files were all renamed with the following format: 000000.png, where each file increases the number by one. The images and the masks have the same names, so if an image is named 000001.png, then its corresponding mask is named 000001.mat as well. Since the masks were very large files, and this would cause overhead when uploading to Google Drive, the PNGs were extracted from the matlab files with Scipy.io [8]. When extracting these images, it is clear that they are not binary masks, but instead seem to have shading (Fig. 2.3). This is probably due to the dataset being inclusive to researchers interested in developing instance segmentation algorithms. Since this project is doing semantic segmentation, the mask is binarized by thresholding all pixels greater than zero to be of value 255. This thresholding technique was performed by using NumPy's array operations [10]. Lastly, the data was uploaded to a Google Drive directory for it to later be accessed utilizing Google Colab [4].

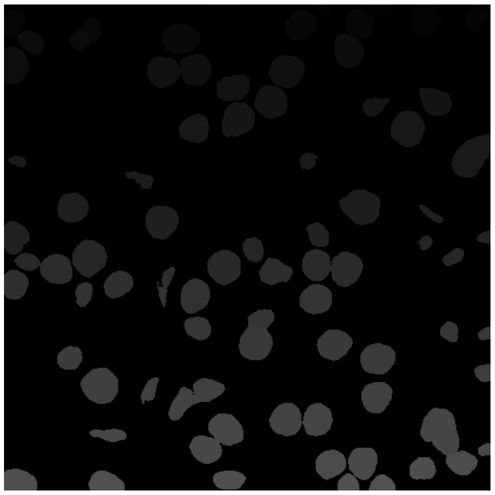
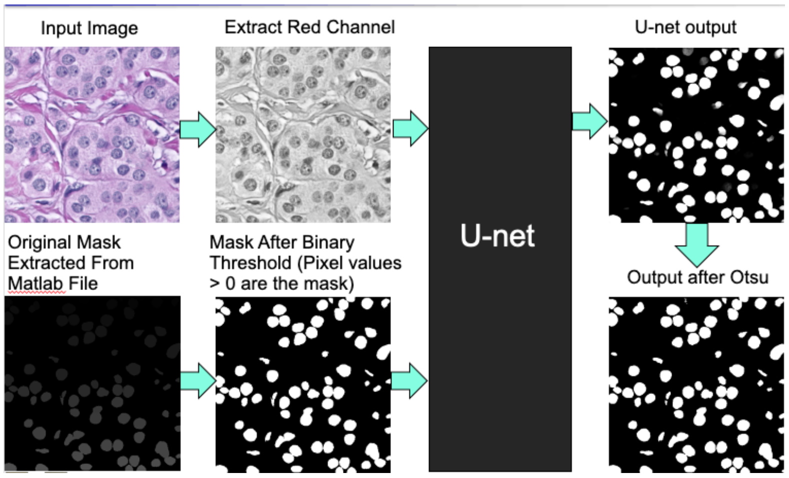


Fig. 2.3: Example mask extracted from matlab files before binary threshold

C. *Image Processing Pipeline*

Now that the data is formatted in a consistent manner and uploaded to the Google Drive, the image processing pipeline is ready to utilize this data. During the pipeline there are essentially two stages. The first stage is the training stage, while the second stage is more of a testing/production stage, where the pipeline could be utilized by a user. In the training stage, we reserve 3,600 images and masks for the training of the segmentation, while 400 are left for cross validation to ensure the model is performing as expected. In the training phase, the red channel is extracted from the original RGB images as it brings out the cells in a clearer format (Fig. 2.4). It also allows the model to use less compute as there won't be as many operations to perform. During training, the masks are also converted to a binary format as pre-processing. The model that this pipeline utilizes is the U-net deep learning architecture for biomedical image segmentation [7]. After the model is done training then a separate notebook can outline the testing/production phase of the pipeline. In this phase, 1,000 images that the U-net model did not see during training are fed to the U-net. This phase has the same pre-processing step as the training phase, where the red channel is extracted from the image and then passed to the U-net model. After the U-net model is run a predicted mask is obtained where the output is a grayscale image. After this, the Otsu threshold from OpenCV [3] is utilized to make the image binary. Fig. 2.4 outlines both stages of the pipeline in one. The only difference in the pipeline between the training and testing stage is that in testing, the expected masks are not provided to the U-net architecture.


Input Image
Extract Red Channel
U-net output
U-net
Output after Otsu
Original Mask Extracted From Matlab File
Mask After Binary Threshold (Pixel values > 0 are the mask)
Fig. 2.4: shows the image processing pipeline that was developed

III. Approach: Algorithms and Module Implementation Details

Since the pre and post processing steps were somewhat easy to implement utilizing libraries such as OpenCV and NumPy [3][10], this section will cover the U-net algorithm and its implementation utilizing Keras [2].

A. *U-net Algorithm*

The U-net algorithm is a convolutional neural network (CNN) that was invented in 2015 for semantic segmentation of biomedical images [7]. It essentially has two phases of the algorithm. The first of which is an encoder which pools the image down, and the second step being a decoder with the outputs of each layer in the encoder appended alongside the convolutional transpose that is required to pool up. Fig. 3.1 shows the U-net architecture that was invented in 2015.

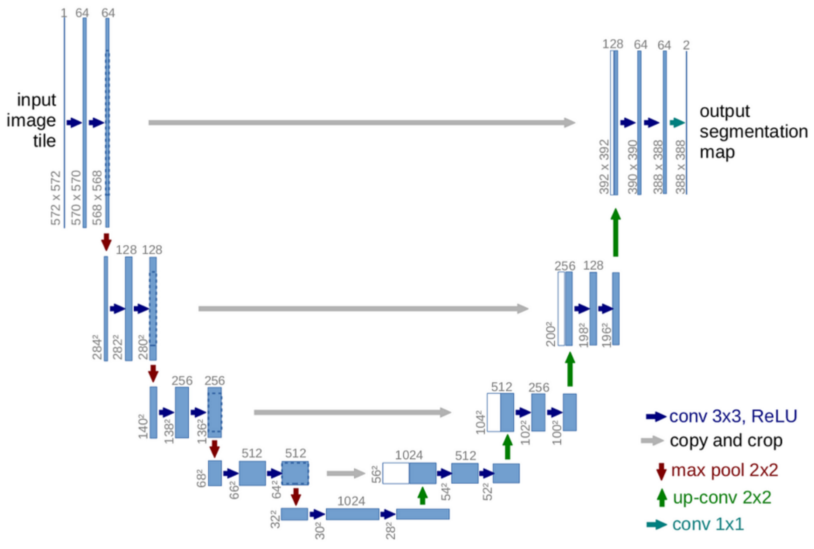


Fig. 3.1: Shows the U-net architecture from the 2015 paper. This image was taken from that paper as well [7].

As you can see from the U-net architecture that was proposed in Fig. 3.1, there are many layers. The original architecture had layers like so: 64 →128 → 256 → 512 → 1024 → 512 → 256 → 128 → 64. This filter setup all in all gave the model more than 7 million trainable parameters. This would require an extensive amount of compute which would take time. The limitations of the Colab environment that was being utilized were: 50 GB RAM and NVIDIA Tesla T4 GPU with 15 GB of GPU RAM. Because the computational infrastructure was not infinite and time was also constrained, it made more sense to make the model have less filters which would allow it to use less compute. The model setup that was implemented has a filter structure like so: 16 → 32 → 64 → 128 → 256 → 128 → 64 → 32 → 16. This setup has roughly 2 million trainable parameters which essentially means the train time would take drastically less time than the initial model. Something else that was added in the model that was implemented were dropout layers. These layers are implemented to essentially reduce the chance that the model overfits. It does this by randomly killing, or dropping out, neurons so that they would not affect the final decision. This essentially acts as a noise inducer during the training which makes the model more robust at the end of the day. The dropout function doesn't get utilized when the model is ready for testing/production.

B. *U-net Implementation*

Utilizing Keras [2], I implemented the U-net architecture. The nice part about utilizing a library such as Keras is that it doesn't require the user to implement the complex convolution operations, pooling operations, and optimizer that is required to build a deep convolutional neural network. Instead, all of these operations are easily implemented in Keras like legos so a user can just put together the modules like lego bricks, to create a new neural network. The Keras modules that were utilized to implement the U-net architecture were: Conv2D, Conv2DTranspose, Dropout, MaxPooling2D, concatenate, ModelCheckpointZ, Model, and load_model. The activation function that was utilized in the output layer of the U-net architecture was the sigmoid activation function. It was possible to use a different activation function such as a binary step activation function to skip the Otsu step in the pipeline altogether, but it is not certain if it would provide a higher accuracy at the end. The optimizer that was utilized for the U-net was the Adam optimizer which minimized the binary cross entropy loss function. Lastly, 50 epochs will be utilized to train the model.

C. *U-net Evaluation Metrics*

To evaluate the U-net there are various metrics that were utilized during and after training of the U-net. The first evaluation metric was the accuracy score of the U-net outputs, which essentially calculates the percentage of pixels that it had a correct prediction for. Fig. 3.2 shows the equation for accuracy.

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

Fig. 3.2: shows the accuracy equation. TP stands for true positive; TN stands for true negative; FP stands for false positive; and FN stands for false negative. Image was taken from [11].

The next evaluation metric that was utilized was the AUC score essentially scores the models ability to distinguish positive and negative instances, or in our situation pixel values of 0 or 1. Fig. 3.3 shows the equation for the AUC score.

$$AUC = \frac{\sum Rank(+) - |+| \times (|+|+1)/2}{|+|+|-|}$$

where:
$\sum Rank(+)$ is the sum the ranks of all positively classified examples
$|+|$ is the number of positive examples in the dataset
$|-|$ is the number of negative examples in the dataset

Fig. 3.3: shows the calculation for the AUC score. Image was taken from [12].

The last two scores that were used to evaluate the effectiveness of the model were the precision and recall scores. The precision metric is evaluating the accuracy of the positive predictions that the model made. Essentially, meaning if the U-net predicted the pixel value to be 1, how many of those predictions were actually right. The recall score is similar to precision, except it takes into account the false negatives. Fig. 3.4 shows the recall and precision equations.

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

Fig. 3.4: shows the equations for precision and recall. This image was taken from [13].

IV. Experimental Results and Discussion

In this section, the results of the training and testing of the U-net model will be reported. After that, a discussion of possible improvements of the model will occur.

A. *Training and Testing Results*

For the U-net implementation, four different metrics were utilized for testing the performance of the model: accuracy, AUC score, precision, and recall. The following are some figures that show these metrics during the training of the model.

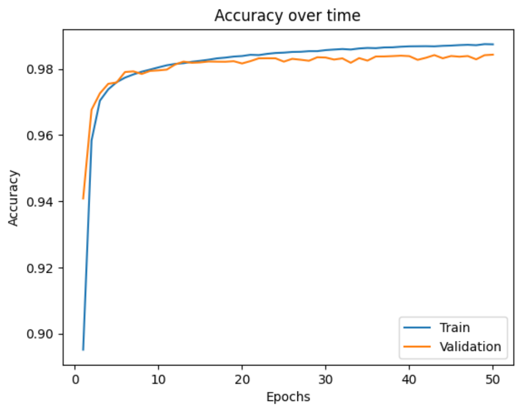


Fig. 4.1: Accuracy over time during training

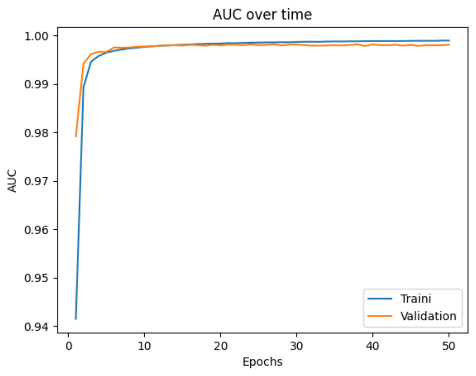


Fig. 4.2: AUC score over time during training

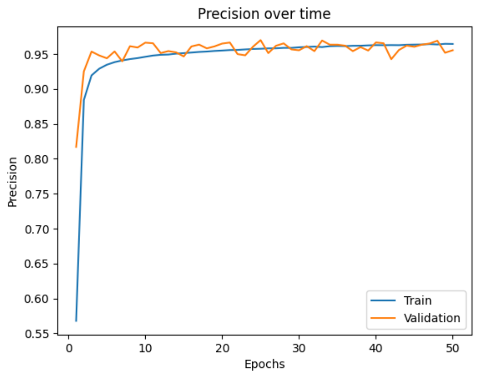


Fig. 4.3: Precision over time during training

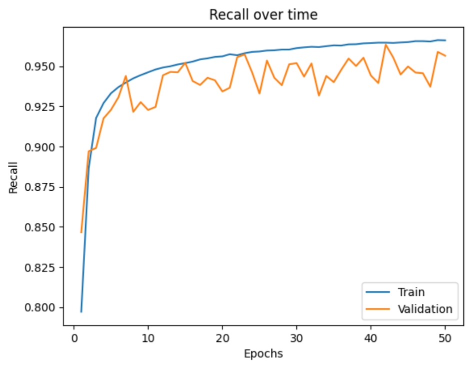


Fig. 4.4: Recall over time during training

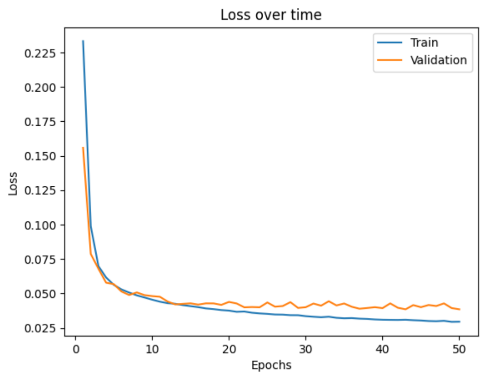


Fig. 4.5: BCE loss function over time during training

Based on these scores it demonstrates that the model learned extremely quickly and seemed to learn what was wanting to be learned, given the high accuracy scores. It may even be worth it to note, was 50 epochs overkill? As you can see it was still learning at 50 epochs, but was it necessary at the end of the day? The difference between the models at 20 epochs vs 50 epochs are very minimal, although the 50 epoch model definitely performs better. Now, we will conduct a test with the 50 epoch model on one of the images to see if it did what was expected visually.

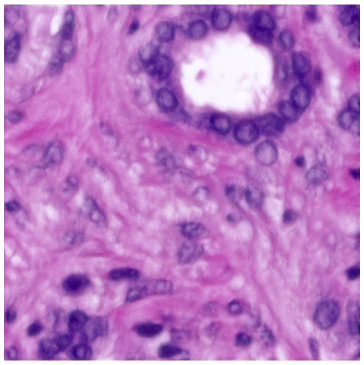


Fig. 4.6: Example test image

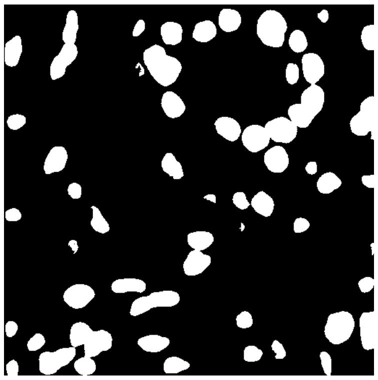


Fig. 4.7: Expected mask corresponding to Fig. 4.6

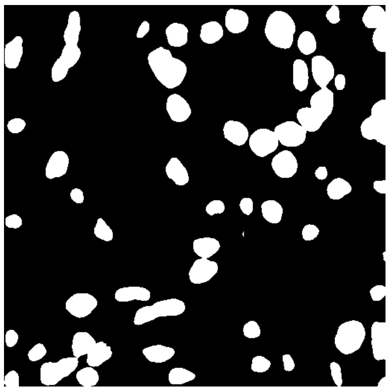
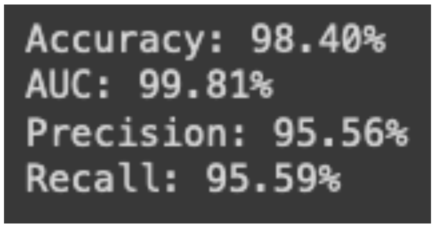


Fig. 4.8: Predicted mask from the image processing pipeline

As we can see from these images, it appears the the image processing pipeline that was developed segments test images with high accuracy. There are only minimal differences between the expected mask and the predicted mask. Lastly, the testing metrics still need to be reported.


```
Accuracy: 98.40%
AUC: 99.81%
Precision: 95.56%
Recall: 95.59%
```
Fig. 4.9: Testing results; Contains accuracy, AUC, precision, and recall

From the test results that utilized 1,000 test images that the U-net model had not seen, it appears that the model performs at a strong rate.

B. Discussion

The results from the image processing pipeline that was implemented are quite strong. But this does not mean that breast cancer is now solved, it merely means that the U-net model is strong at modeling the synthetic data that was fed to it during training and testing. There are still improvements that could be made. For instance, the pipeline that was implemented utilized Otsu at the end to binarize the output of the U-net. This operation could have been performed with similar or better accuracy by changing the activation function of the output layer of the U-net to be a binary step activation function rather than a sigmoid activation function. This improvement also would make it much easier for the user to utilize the U-net because they wouldn't have to worry about binarizing the result of the semantic segmentation at the end. Some disadvantages of using deep learning to solve this problem are the compute cost of deep learning. Classical computer vision approaches require very minimal compute, where the training of the U-net took around 3 hours even with a very strong computer.

V. Conclusion and Future Work

All in all, the goal of semantic segmentation of synthetic images of breast cancer nuclei was accomplished with strong results. In the future there are many things that need to be tested to see if this U-net solution could hold up in production. For instance, in the future the U-net needs to be tested with real breast cancer nuclei images as the synthetic images may not account for things that happen in nature that cannot be predicted. If that test proves to be resourceful, this image processing pipeline needs to be tested on other cancer nuclei to see if what it learned from the breast cancer nuclei is translatable amongst cancers.

Note to the grader, files contained within the project directory:

- explore.ipynb: mainly just to visualize the images when exploring the dataset
- main.py: mainly just to save the matlab masks as PNGs
- unet_segmentation.ipynb: this is a google colab notebook that was utilized to train the U-net
- test.ipynb: this is a google colab notebook utilized to test the U-net

REFERENCES

[1] Aksac, A., Demetrick, D. J., Ozyer, T. & Alhajj, R. Brecahad: a dataset for breast cancer histopathological annotation and diagnosis. *BMC research notes* **12**, 1–3 (2019).

[2] F. Chollet and Others, "Keras Documentation," Keras Team, 2023. [Online]. Available: https://keras.io/.

[3] G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000.

[4] Google Colaboratory, "Google Colab," [Online]. Available: https://colab.research.google.com/.

[5] J. D. Hunter and Contributors, "Matplotlib: Visualization with Python," Matplotlib Development Team, 2023. [Online]. Available: https://matplotlib.org/.

[6] K. Ding, M. Zhou, H. Wang, O. Gevaert, D. Metaxas, and S. Zhang, "A Large-scale Synthetic Pathological Dataset for Deep Learning-enabled Segmentation of Breast Cancer," Scientific Data, vol. 10, no. 1, p. 231, 2023. [Online]. Available: https://doi.org/10.1038/s41597-023-02125-y

[7] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," arXiv preprint arXiv:1505.04597, 2015. [Online]. Available: https://arxiv.org/abs/1505.04597

[8] P. Virtanen and Others, "SciPy: Open Source Scientific Tools for Python," SciPy Developers, 2023. [Online]. Available: https://www.scipy.org/.

[9] Python Software Foundation, "Python Documentation," Python Software Foundation, 2023. [Online]. Available: https://docs.python.org/3/.

[10] T. E. Oliphant et al., "NumPy: Array computation in Python," NumPy Community, 2020. [Online]. Available: https://numpy.org/

[11] Nomidl, "What is Precision, Recall, Accuracy, and F1 Score?" [Online]. Available: https://www.nomidl.com/machine-learning/what-is-precision-recall-accuracy-and-f1-score/, 2023.

[12] Turing.com, "AUC-ROC Curves and Their Usage for Classification in Python," [Online]. Available: https://www.turing.com/kb/auc-roc-curves-and-their-usage-for-classification-in-python, 2023.

[13] V7 Labs, "F1 Score: A Complete Guide," [Online]. Available: https://www.v7labs.com/blog/f1-score-guide, 2023.