Master’s Project Report

Roman Nett

Spring 2023

Abstract:

Using cell graphs for an image classification problem can greatly improve performance, and allow for more realistic modeling of cell interactions. However, generating these cell graphs requires accurate image segmentation. Recently, deep learning techniques have been developed for image and instance segmentation, but they often lack wide testing. Stardist is one of those models, using start-convex polygons to detect cells from a top-down approach. Applying Stardist for the first time to the CRC dataset after training on separate data, I generate results that back up the claim that these deep learning methods can be successful. Additionally, Stardist itself can label a dataset it was not trained on and produce cell graphs that successfully indicate the true class of the original image.

Intro:

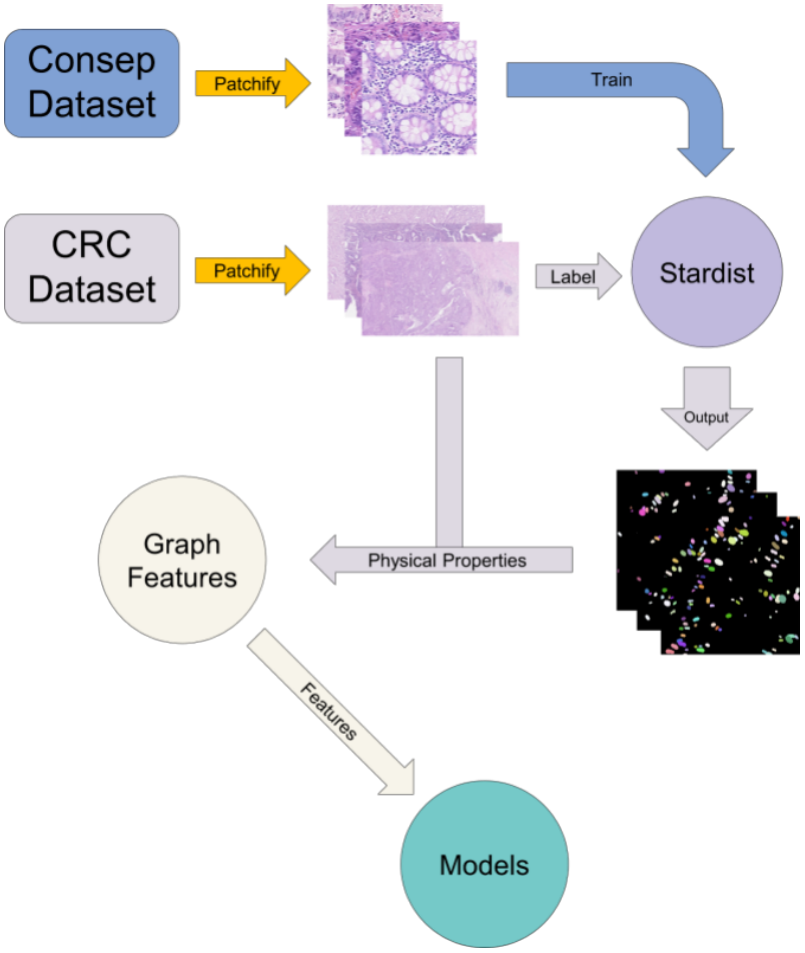
Coming into the semester, I had never actually implemented any image segmentation methods, or worked very much with computer vision. As the Stardist model was built on top of the U-Net architecture, I first set out to create a proof of concept. Using just the base U-Net structure, I trained a model on the Consep dataset. I also used the watershed method to clean my results. Watershed is a computer vision technique that uses boundaries between background and foreground classifications to create areas of uncertainty. These areas of uncertainty can be tweaked depending on the dataset to separate cells more or less, depending on how close the true cells are. This simple model reached a mean intersection-over-union score of 70.7%, which was a good sign for the model. After this “proof of consep” I moved on to the Stardist model. The end goal was to train this model and then apply it to another dataset. After creating the cell graphs generated from the cell labels, I could analyze them for their graph features. If the cell graphs obtained from this dataset could then generate good results, we know that the Stardist model has been validated on out-of-bag data.

Methodology:

1. Overview

   All models and scripts were run on my Dell XPS laptop, which is >4 years old and has 16GB of RAM. This is not an overwhelming amount and is not the newest computer, which shows that the methods I used are very efficient and cost-effective.

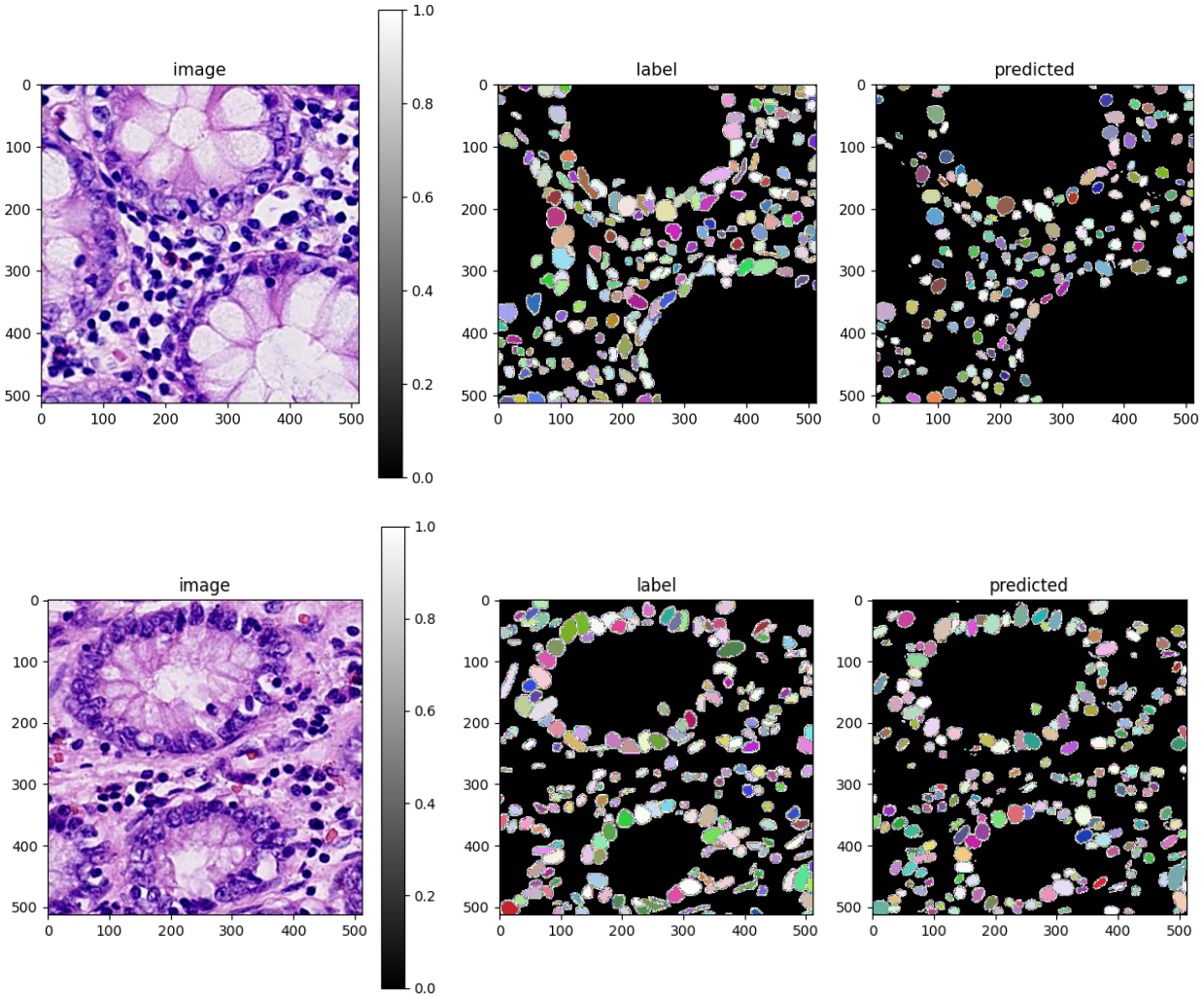
   The flowchart below shows the process I went through.



2. Cell Segmentation

The Stardist model is based on U-Net, which utilizes layers of downsampling over convolutional neural network layers followed by the same amount of upsampling. A key feature is that each layer of downsampling is copied over to its corresponding layer of upsampling, meaning the model has information from every stage of analysis. The Stardist model also uses star-convex polygons to further refine the cell classifications. Stardist approaches the problem from a “top-down” perspective, first classifying areas it thinks there is a cell, and then further defines where the borders are - rather than a “bottom-up” approach that starts by classifying foreground and background pixels and then working to separate them. To prevent the size of the input from being overwhelming, I broke my input images into patches of 512x512 pixels. This not only allowed the model to focus on more resolution but also allowed me to easily label images of different sizes. Consep, the training dataset, was patched into 2x2 patches, whereas CRC, the testing dataset was patched into 9x15 patches - for a total of 135 patches per image. I trained the Stardist model for 400 iterations of 100 epochs each, ending

with more than 90% accuracy on the validation set. This training took around 36 hours. Shown in the figure below are sample predicted labels from both the training and validation set respectively.



I then labeled and calculated the centroids for my testing dataset. Calculating the physical features for the full testing images required too much memory to be allocated, so I instead did this while the images were still in patches, and then stitched them together. In order to do this, I had to ensure the id of the cell was updated appropriately, and the pixel location of the centroid was offset correctly based on which patch it was in.

3. Graph Formation

I generated the cell graphs in a standard distance-based method. First, I found the distance between each pair of centroids and then filtered down the pairs that were within 64 pixels of each other. From these pairs, I created an edge and added it to the list. I then used this list of edges to create a graph in NetworkX.

4. Features

After finding the features of a few samples, I removed some features that were the same for all of them. I used the methods provided to me by Sudipta to generate 20 global features: average degree, average clustering, the ratio of the largest connected component, component count, average eccentricity, diameter, 90th percentile eccentricity, 90th percentile diameter, number of central points, percent of central points, vertex count, edge count, the 2 eigenvalues of the adjacency matrix, the energy, the lower and upper slope, the number of ones, and the trace and energy of the laplacian matrix. I found these features for each of the 139 images, except for one graph that contained over 50,000 cells, as there was not enough space to allocate the matrix for the linear algebra features.

5. ML Algorithms

I conducted my final analysis in R and tried 3 different types of classical machine learning algorithms: support vector machine, random forest, and k nearest neighbor. I used 3 different SVM kernels: radial, linear, and polynomial. For k nearest neighbor I tested k's from 1 to 10. The CRC dataset has samples with normal tissue, low-grade cancer, and high-grade cancer. I create models for both two-class classifications (normal vs cancerous) and three-class classifications (normal vs low-grade vs high-grade). I split my features into random samples of 80% training data, and 20% testing data, and averaged my results on the testing set over 20 different samples.

Results:

2-class Classification

Model	SVM (radial kernel)	SVM (linear kernel)	SVM (poly kernel)	Random Forest	KNN (k=1)
Average Accuracy	73.8%	79.5%	67.0%	67.0%	88.4%
Aggregated Confusion Matrix	N C N 232 86 C 69 173	N C N 245 68 C 56 191	N C N 196 124 C 105 135	N C N 196 124 C 105 135	N C N 269 31 C 34 226

For the confusion matrices: N = normal, C = cancerous

The row is the predicted class, the column is the actual class

They are summed over the 20 samples

3-class Classification

Model	SVM (radial kernel)	SVM (linear kernel)	SVM (poly kernel)	Random Forest	KNN (k=1)
Average Accuracy	63.9%	67.5%	60.5%	60.5%	81.6%
Aggregated Confusion Matrix	N L H N 246 62 73 L 34 57 17 H 6 10 55	N L H N 230 39 47 L 40 70 20 H 16 20 78	N L H N 270 108 91 L 10 15 0 H 6 6 54	N L H N 270 108 91 L 10 15 0 H 6 6 54	N L H N 261 30 1 L 25 91 39 H 0 8 105

For the confusion matrices: N = normal, L = Low Grade, H = High Grade
The row is the predicted class, the column is the actual class
They are summed over the 20 samples

For two-class classification support vector machines, the linear kernel performed the best, having an average testing accuracy of 79.5% over the 20 samples. The random forest models performed a little worse, with an average accuracy of 67.0%. The nearest neighbor models performed better than both the other models, with a k of 1 having the highest average accuracy of 88.4%. For three-class classification support vector machines, the linear kernel again outperformed the other kernels with an average accuracy of 67.5%. Random forests again did slightly worse averaging 60.5% accuracy, and yet again the k of 1 nearest neighbor took the top spot with a mean of 81.6% accuracy.

Conclusion:

Even with relatively noncomplex classical machine learning models, I was able to achieve admirable performance for two and three-class classification. With even more complicated and higher dimension models, it is expected that you could reach even higher accuracy, for potential state-of-the-art results. I have shown that using deep learning techniques, even when trained on a different dataset, produces cell segmentations that can provide good results. By using the standard cell-graph creation pipeline, I ensure that the only variable I changed was cell segmentation. Overall, I am very happy with what I was able to complete over the semester. I have learned a lot - especially in the realm of image processing and segmenting, and other computer vision techniques. I have shown that the deep learning methods that I used to segment the images can allow for good cell graph performance, and generalize across datasets. Once the Stardist model was trained, it did not need to be modified to label new images, and the segmented cells were clear and separated. This greatly speeds up the image segmentation portion of the cell graph creation methodology.