Razi Khawaja
Segmentation Report
5/10/2024

1. Introduction
    1.1. The aim of this project is to create a CNN based on the U-Net architecture to properly segment ultrasound images produced by ultrasound ablation studies. The model is intended to take in an image and output a best guess of the affected tissue. While the exact use of the model is beyond the scope of the project, it's easy to imagine that the model could be part of a workflow that allows a medical provider to 'see' what tissue they're affecting in a realtime procedure.
    1.2. The ultrasound ablation procedure involves pulsing ultrasound waves in to a targeted area with the intent of damaging tissue in a target but non-invasive way. The benefit of this approach is that targeted cell structures, perhaps cancer cells, can be physically destroyed such that the patient would not need to undergo more invasive procedures to eradicate such malignant tissue.
2. Background
    2.1. We were given a training set of 880 input images and their corresponding labeled images and a test set of 400 input images and their corresponding labels. The input images are RGB, though they appear greyscale, and the labels are greyscale. All images are 256x1024. All images come from test cases where there are 100 frames per test case, except for one case in the training set that only had 80 frames.
    2.2. Upon visual inspection, it's very difficult to make sense of the images as the images are fuzzy and non-distinct; the data does not make sense to human eyes so a computer based approach to rating the images is a natural solution to the problem of finding the affected tissue.
    2.3. Image processing techniques, statistical approaches to data preprocessing and postprocessing, and machine learning models were considered to create images of the affected tissues. I was only able to use machine learning models and a logistic model to tune the pixel results through I believe there is more to explore in image processing techniques and data postprocessing.
    2.4. An attempt was made, in particular, to create a domain transformation to feed in to a logistic regression model though it did not come together in the project. The basic idea is that the probability of a pixel being affected is in some part affected by the probability of its neighbors being affected. This makes intuitive sense as a cell that is directly next to a decaying cell would have a higher probability of being

affected by whatever destroyed its neighbor. This attempt did not make it in to the final system due to time constraints though it is a direction I would like to explore.

2.5. The metric that was used to grade performance is the Dice score. DiceScore = 2TP / (2TP + FP + FN).

3. Procedure

3.1. Data Preprocessing

3.1.1. For training and testing, it was decided that a 16x downscale of both sets would be used. This is due purely to computational time considerations. While it would be interesting to explore performance metrics across downscale sizes (does the model perform better at 4x as opposed to 16x?), this was not done in any meaningful way.

3.1.2. An interesting quirk in the labeled images is that they are not binarized but true greyscale. I am not sure if this is an artifact of downscaling, but some labeled image pixels had values that were not either 0 or 255. I created a threshold function within my dataset loader class to address this though I am not sure if that had any meaningful effect as of yet.

3.2. U-Net Training

3.2.1. The final U-Net model was trained on 16x images for 50 epochs, a batch size of 16, and a learning rate of 1e-3. These training parameters were found through a grid search and a model with a minimal validation loss was chosen.

3.2.2. The training tracked validation and training losses. The model with the best validation loss was chosen.

3.2.3. Worthy of note is that the model currently outputs non-thresholded values, that is real number approximations of what a pixel value ought to be.

3.3. Logistic Model Tuning

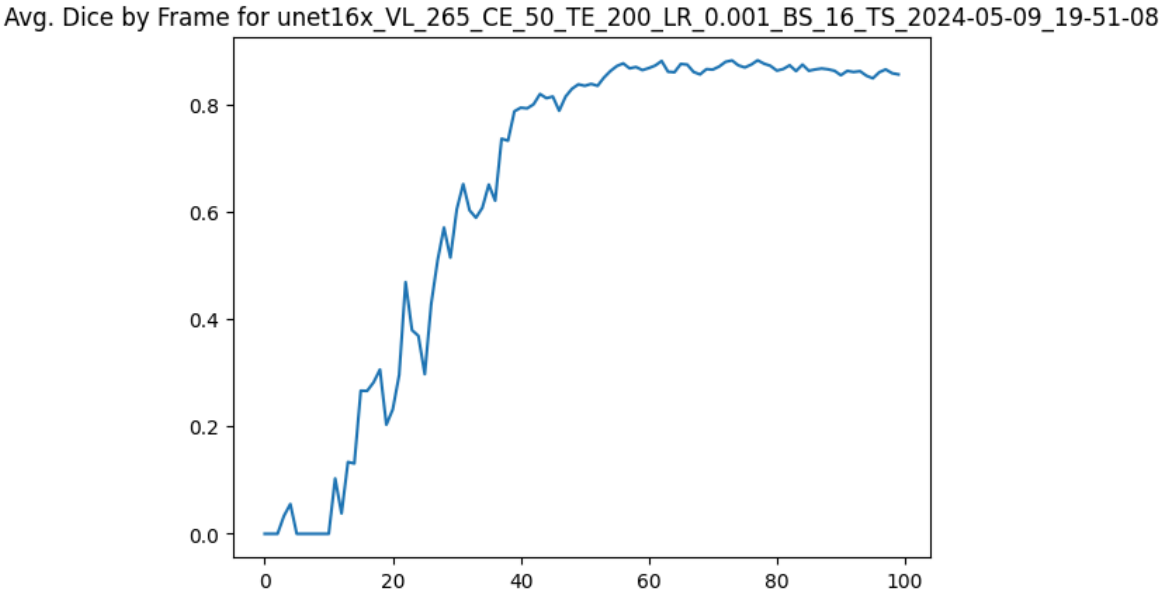
3.3.1. A logistic model was then trained on the model output of real number pixel values of the model and the labeled data. An optimal threshold was chosen that was the highest possible threshold that did not impede the dice score average of the model. The logistic model predicts a value that is either 0 or 255 based on the input pixel value. The predicted vector of pixel values is then reshaped in to whatever the input image shape was and sent off to the scoring pipeline.

3.4. Performance Assessment

3.4.1. The images are then scored on dice metrics and results are calculated and graphed. Worthy of note is that the dice score of any image that does not have any true positives will be zero since there are no true positives in the numerator.

4. Results

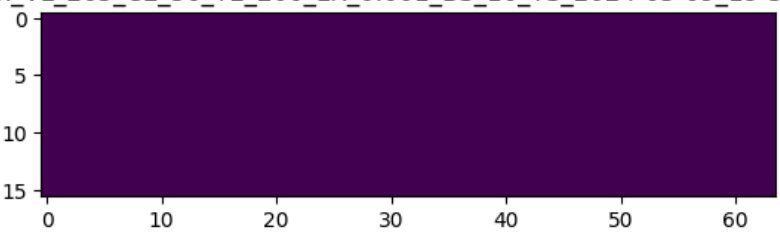
4.1. Results were encouraging but not remarkable. As shown below, the dice score per frame is very low before the ~40 frame mark and levels off at ~.85 afterwards.



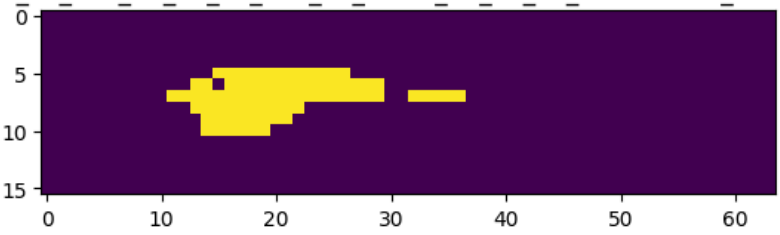
4.2. How this is happening is apparent if we take a look at segmentation predictions for two test cases. For test case 3 on frame 21 and 22, the prediction went from 0 predicted affected pixels to a more usable half correct prediction.

Razi Khawaja
Segmentation Report
5/10/2024

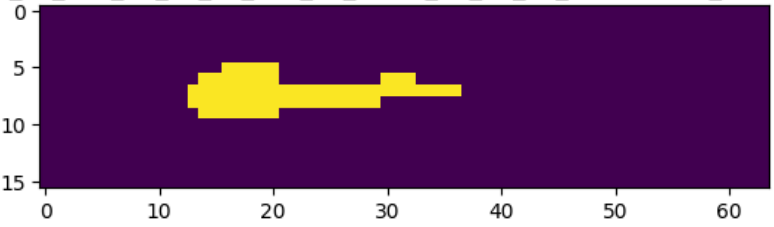
Pred- unet16x_VL_265_CE_50_TE_200_LR_0.001_BS_16_TS_2024-05-09_19-51-08 | C: 3 F: 21



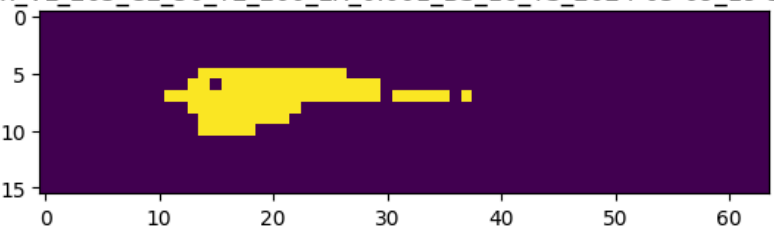
Label- unet16x_VL_265_CE_50_TE_200_LR_0.001_BS_16_TS_2024-05-09_19-51-08 | C: 3 F: 21



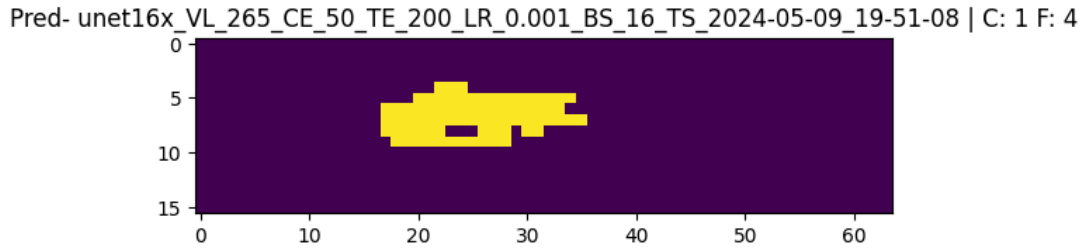
Pred- unet16x_VL_265_CE_50_TE_200_LR_0.001_BS_16_TS_2024-05-09_19-51-08 | C: 3 F: 22

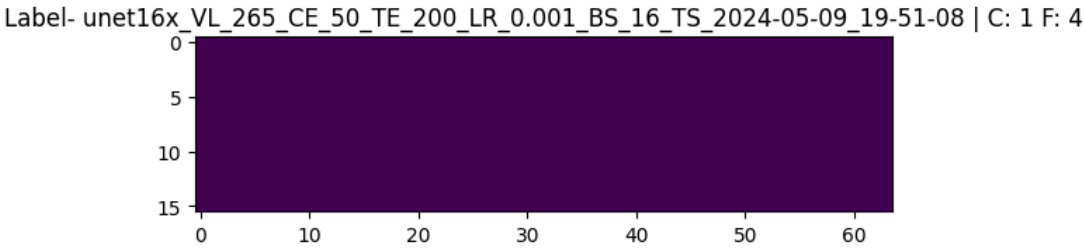


Label- unet16x_VL_265_CE_50_TE_200_LR_0.001_BS_16_TS_2024-05-09_19-51-08 | C: 3 F: 22

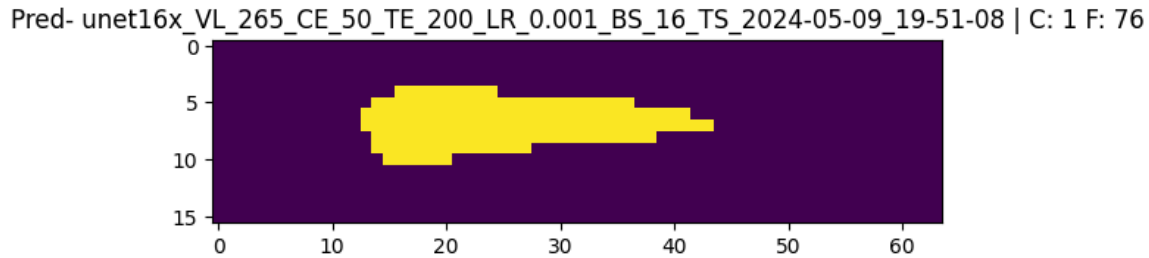


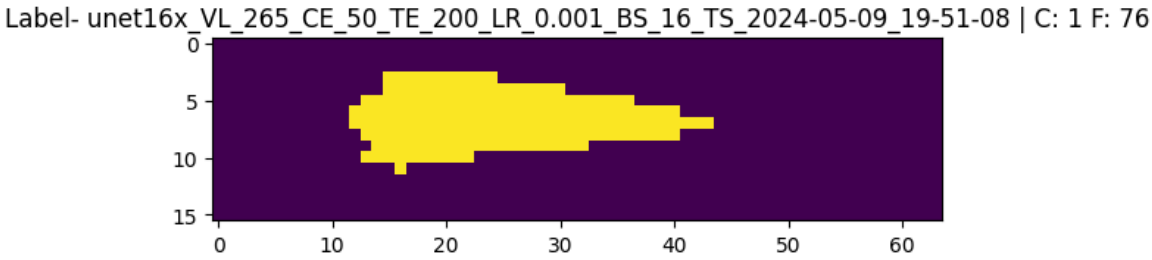
For test case 1 on frame 4, the system predicts way too many false positives.





The system seems to converge nicely on later images, as shown below on test case 1 on frame 76. Very useablely correct!





5. Conclusion

5.1. While the system did not give good results in the frames, the results are encouraging. I will list several areas of improvement that I wish to explore.

5.2. Data Processing

5.2.1. As noted above, image preprocessing techniques may be able to eliminate much of the noise in the input image such that downstream systems could have better prediction. Significant image processing may reduce the need for a machine learning model in general, but that would

be remarkable. For example, smoothing the image based on a localized histogram could reveal some pattern in the data that may be easier to learn from.

5.2.2. I did not implement any considerable preprocessing that would be standard techniques in machine learning e.g. standardization, normalization, etc. This would be an obvious place to look for the future. Notably, the test set performed worse than the training set with interesting FP or FN blocks in the segmentations. Perhaps the test case patients are a bit darker or lighter and are thus throwing off the real number inputs to the logistic function.

5.3. Domain Transform

5.3.1. I intended to transform the data used for the logistic function by creating a set of interaction terms that would be used to augment the available amount of information available to the logistic function. I did not implement this in time for submission.

5.4. In-Line Full CNN

5.4.1. Perhaps the most obvious consideration is that the model itself does not train any internal logistic model and an astute reader might be wondering why I chose to do that in the first place. The reason is ultimately my inexperience. I wanted to be able to create domain transformations from the CNN outputs and take a more classical approach to creating viable predictions since I heard that this has not been tried yet. A ‘professional’ version of this project would have a way to train the model and keep everything ‘in-house’ in the model architecture without the addition of extra bells and whistles to try and coax a correct answer from a trained model.

5.5. Curve Fitting

5.5.1. A naive approach to creating a linear separator would be fitting a curve to the model outputs to determine pixel values for an output mapping. This is simpler and may have been better. An earlier version of this project did implement something like this but I did not have the time to fit it in to the final pipeline.

5.6. Overall, I’m satisfied with the pipeline I was able to construct here. I may explore implementing some of the above areas of exploration in my own time. The project

code can be viewed in the project repo at https://github.com/EdgyPage/SegmentationProject. Access to the full graphical outputs can be requested by contacting me.