Course: CMPT 743 Spring 2019

Assignment 01

Author: Lei Pan

**Active Contour and U-net Report**

The purpose of this assignment is for one part to implement active contour; and for another part to implement U-net structure and to train the U-net to segment medical images.

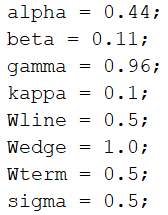
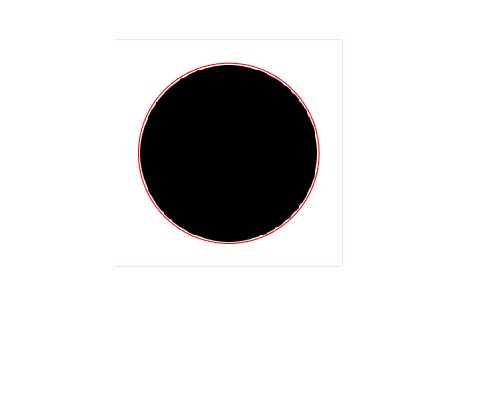
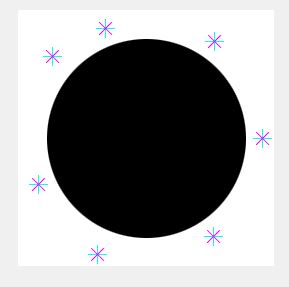
**I. Active Contour**

The active contour parameters are tuned according to different shapes. For each shape, I randomly generated 30 different groups of parameters, compute and save the results. The user initiate points are set to the same for the same image in order to compare the feature of each parameter. The results are based on 200 iterations.

**I.a Circle Image**

The circle image is relatively easy to have a good final contour. The original image with the initialization,

the resulting image with the segmentation and the used parameters are below:

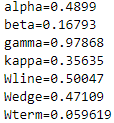
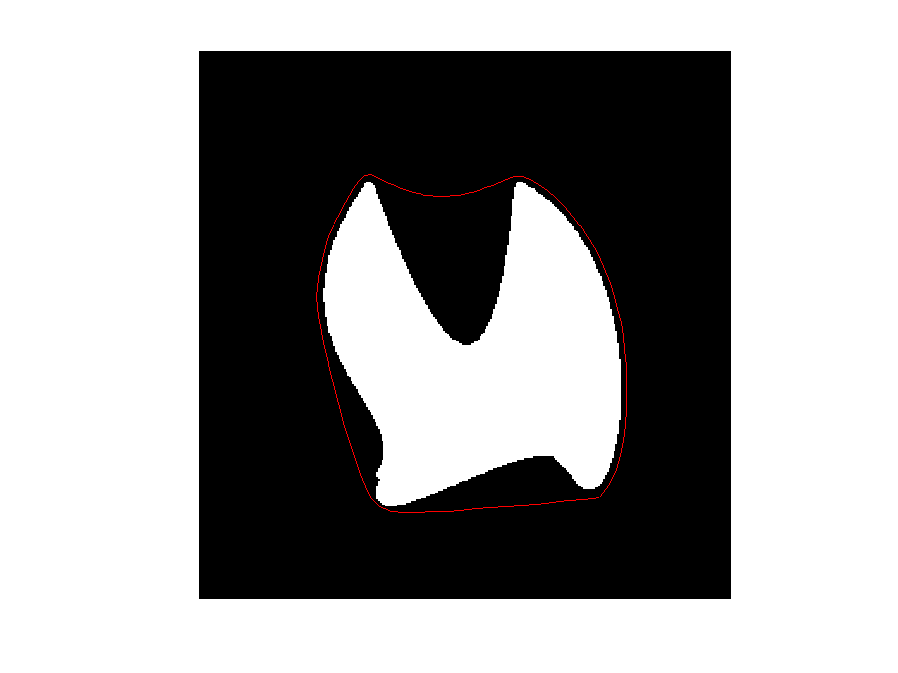
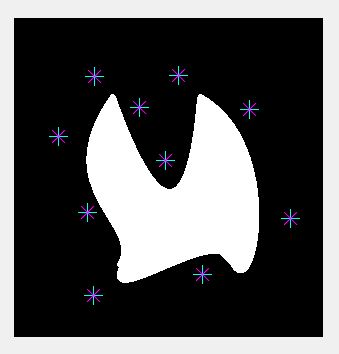


*Figure 1 Active contour and related parameters on image “circle.jpg”*

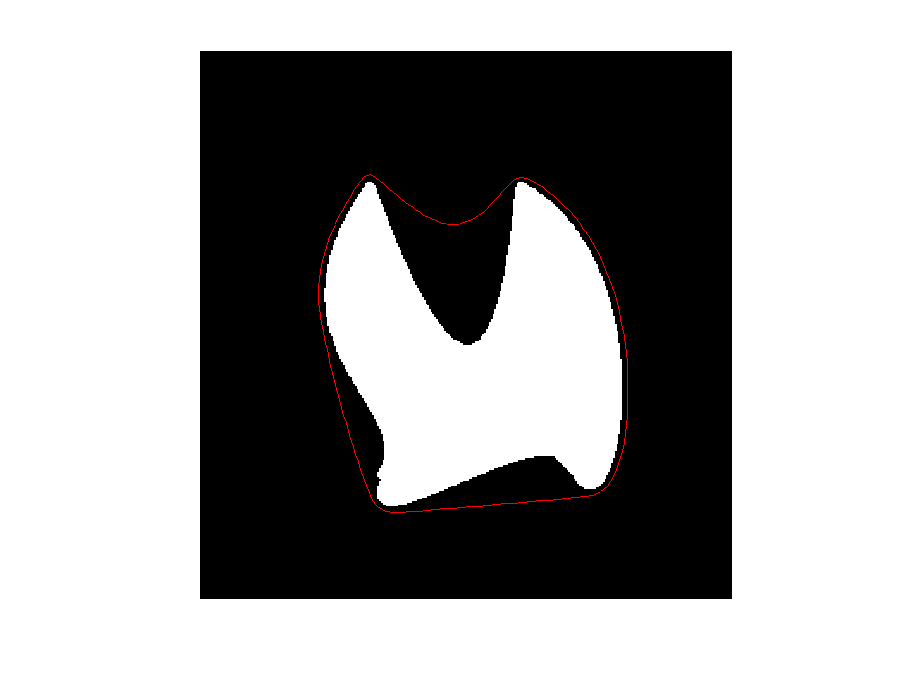
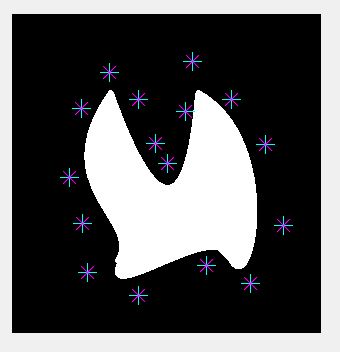
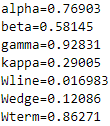
**I.b Shape Image**

The difficulty of tuning parameters on shape image is the concave part. The original image with initialized with 2 cases – small amount and big amount. The original image with the initialization,

the resulting image with the segmentation and the used parameters are below:



*Figure 2 Active contour and related parameters on image “shape.png”(small amount initialization)*

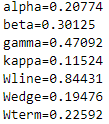
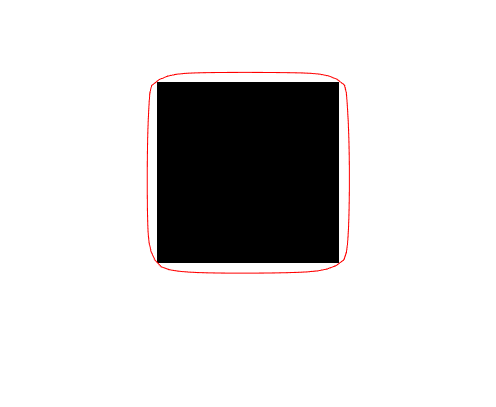
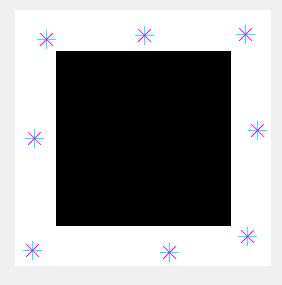
** 

*Figure 3 Active contour and related parameters on image “shape.png”(big amount initialization)*

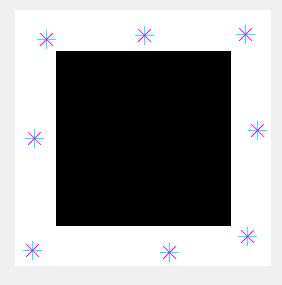
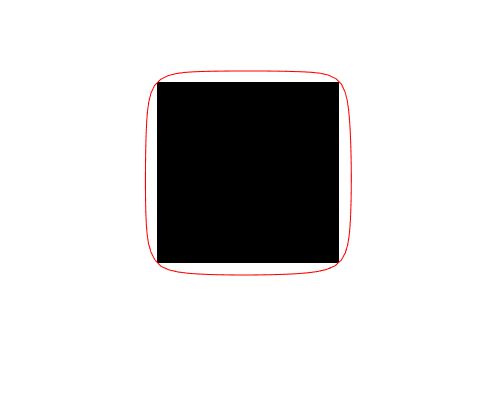
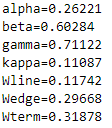
We can see that with big amount initialization points, the concave part on top is slightly better.

**I.c Square Image**

I found two groups of parameters that are both relatively good.The original image with the initialization, the resulting image with the segmentation and the used parameters are below:



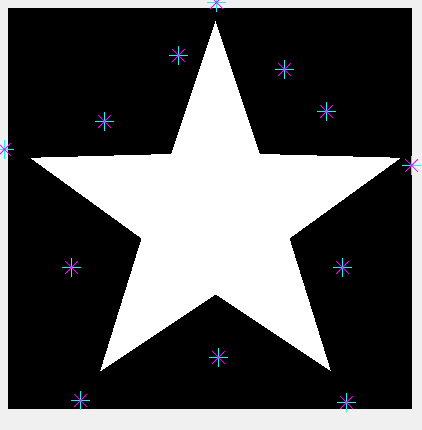
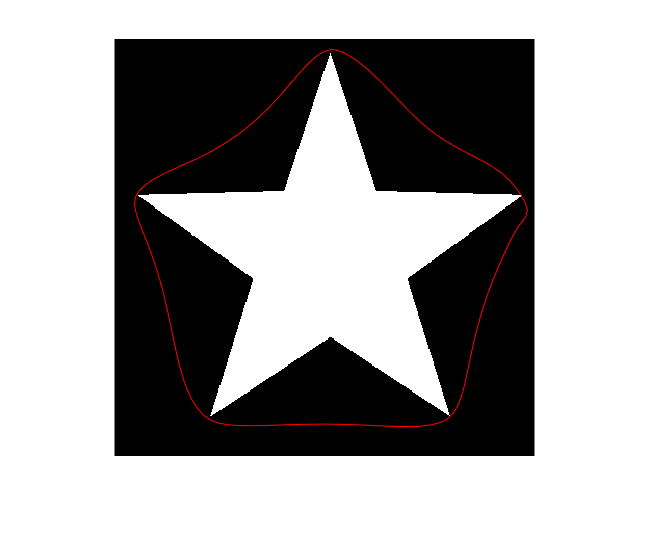
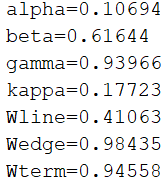
*Figure 4 Active contour and related parameters on image “square.jpg”(one example)*

**

*Figure 5 Active contour and related parameters on image “square.jpg”(another example)*

**I.d Star Image**

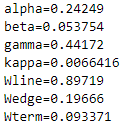
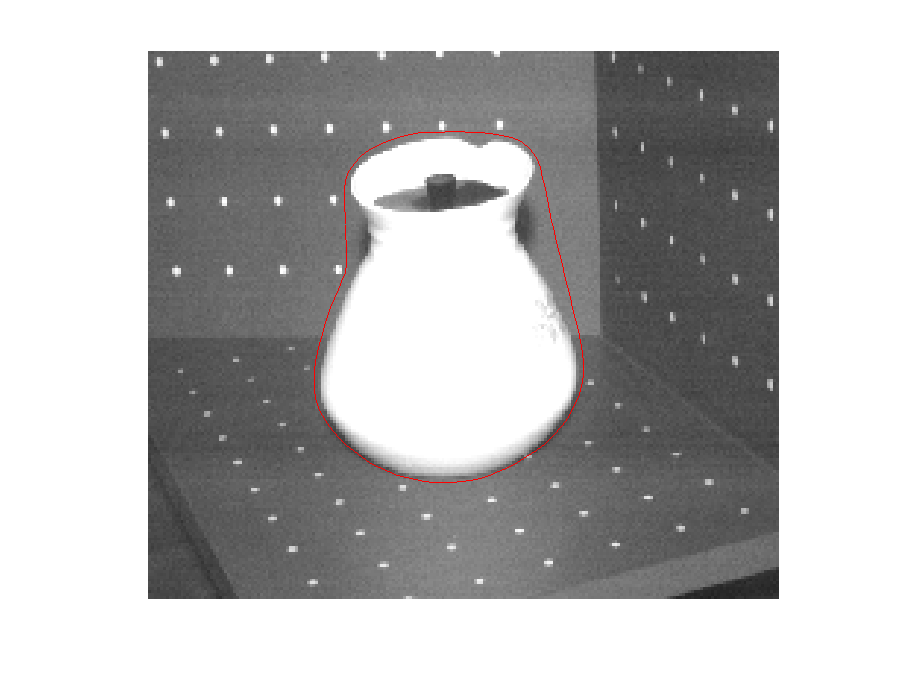
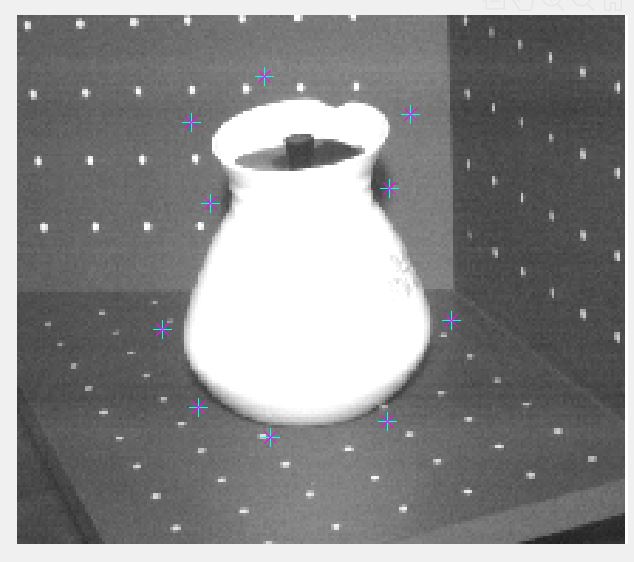
The star image is super hard to get a satisfied result. The original image with the initialization, the resulting image with the segmentation and the used parameters are below:

*Figure 6 Active contour and related parameters on image “star.png”*

**I.e Vase Image**

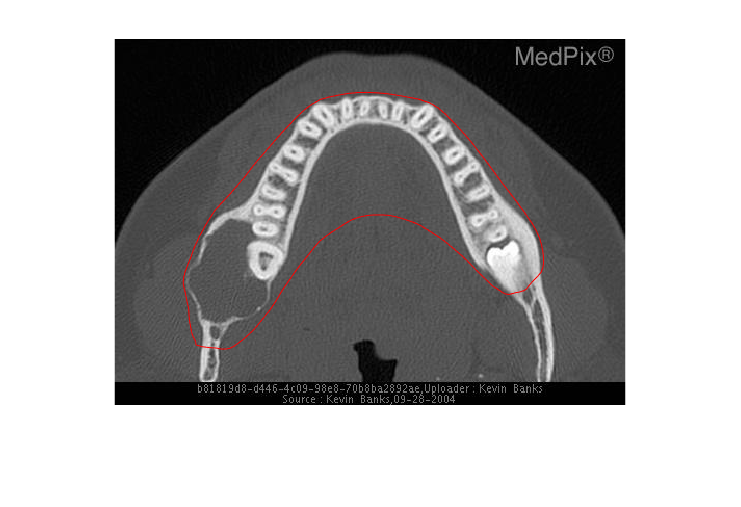
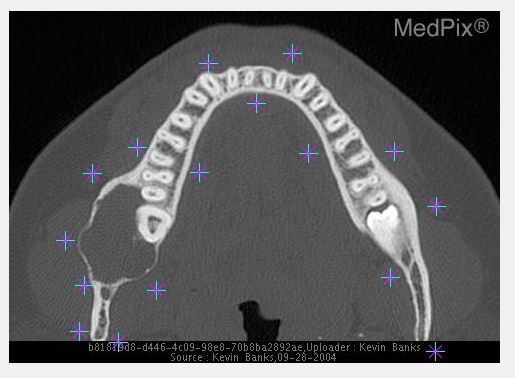
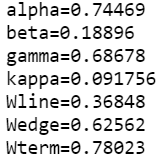
The original image with the initialization, the resulting image with the segmentation and the used parameters are below:



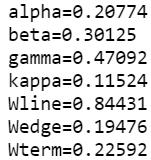
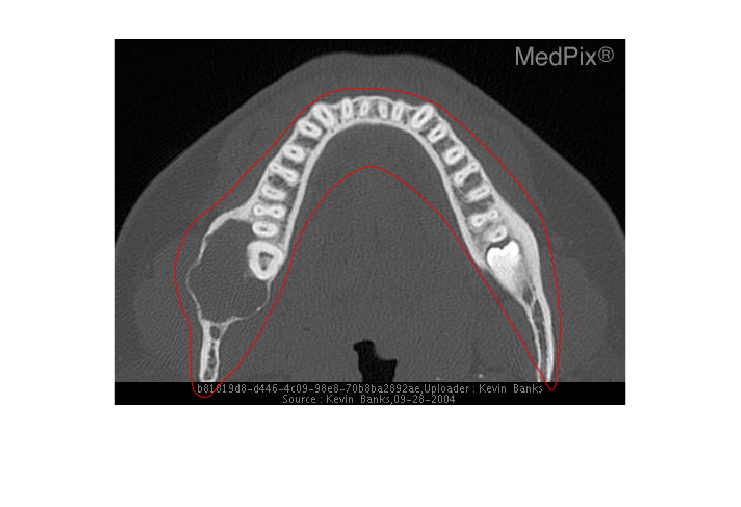
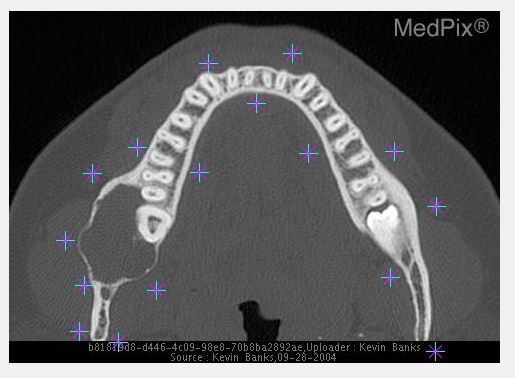
*Figure 7 Active contour and related parameters on image “vase.tif”*

**I.f Dent Image**

The original image with the initialization, the resulting image with the segmentation and the used parameters are below:

*Figure 8 Active contour and related parameters on image “dental.png”(one example)*

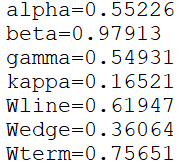
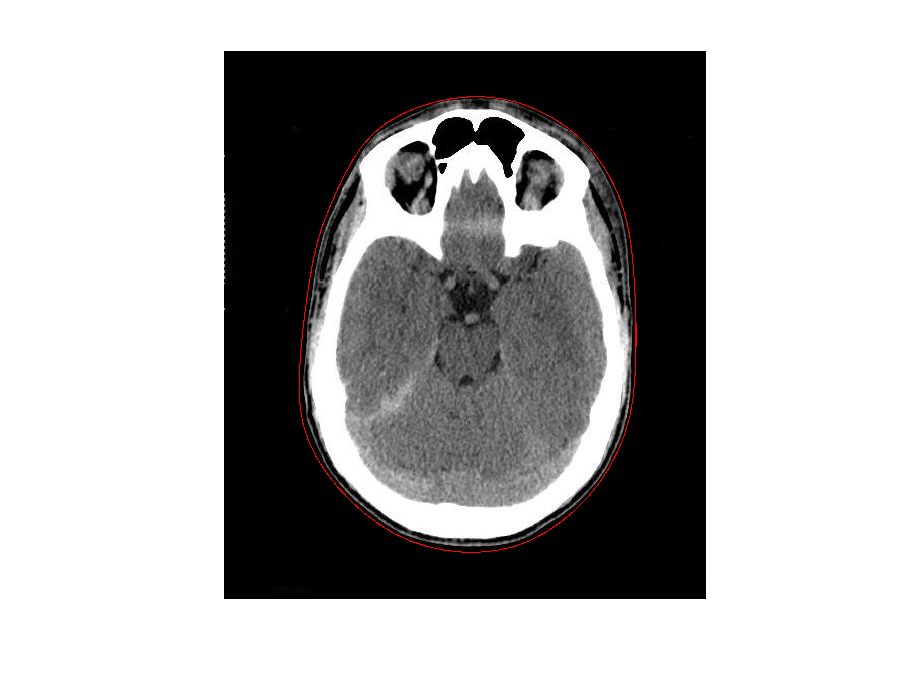


*Figure 9 Active contour and related parameters on image “dental.png”(another example)*

**I.e Brain Image**

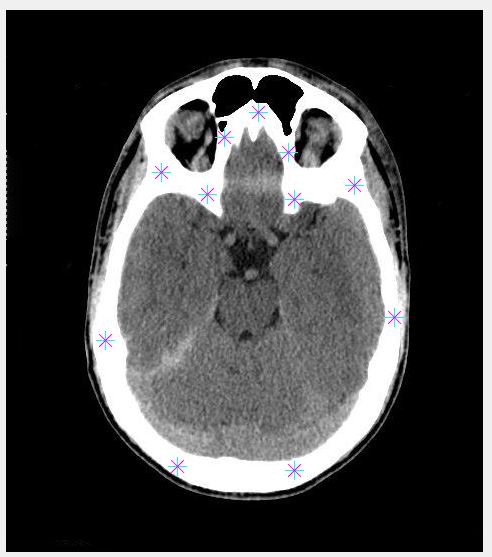
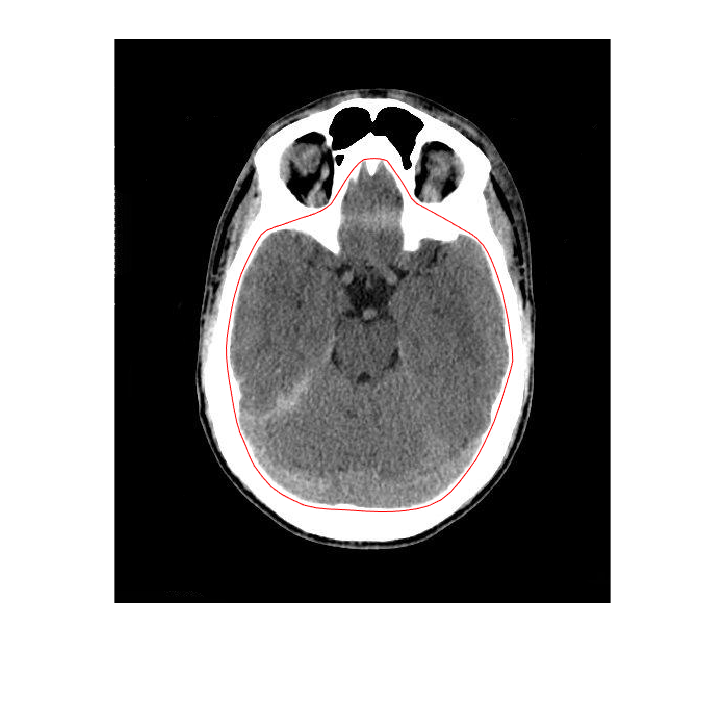
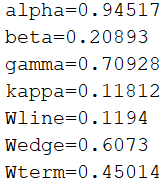
The original image with the initialization, the resulting image with the segmentation and the used parameters are below:

**–** The outer shell of the skull.



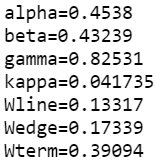
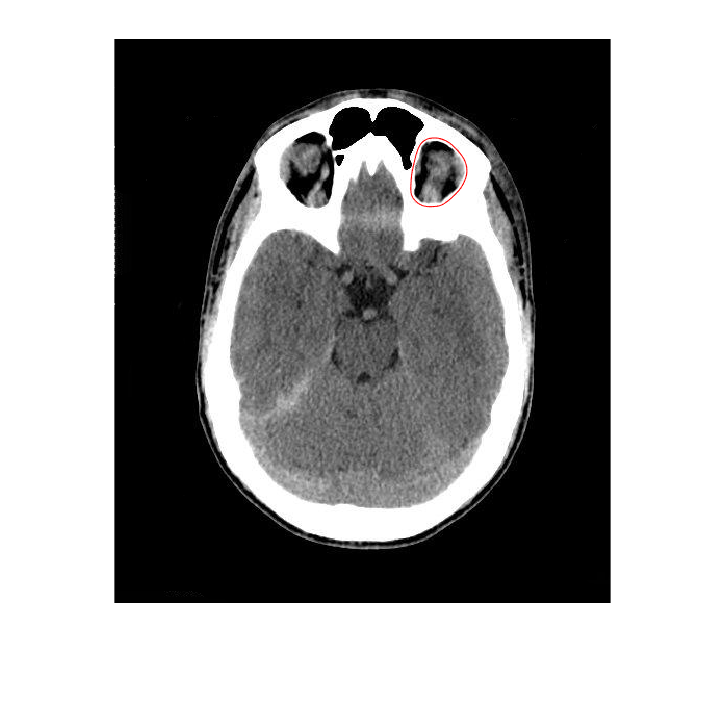
*Figure 10 Active contour and related parameters on image “brain.png”(outer shell)*

**–** The inner contour of the brain matter.

** **

*Figure 11 Active contour and related parameters on image “brain.png”(inner contour)*

**–** The right eye hole.



*Figure 12 Active contour and related parameters on image “brain.png”(right eye hole)*

**I.f Analysis of parameters**

From the tuning of parameters, we can have a better understanding of each parameters’ feature: alpha makes the spline act like a flexible membrane, beta lets it behave more like a thin plate, gamma is the step size, kappa regulates the influence of the external forces fx and fy, and wline, wedge, wterm influences the external energie of the snake.

**II. U-net**

I trained my network on GPU on images of size (572 \* 572), the same input size as the original net.

I used all the data augmentation strategies given in the assignment: flip, zoom, rotate, gamma correction, and elastic deformations. Furthermore, for each strategy, I provided different options such as flipping horizontally or vertically, rotate 90 or 180 degree more details can be seen in code. I also compared the tradeoff of different augmentation strategies (in chapter II.d).

I deviated from the original network structure (with the ReLU in the last step) and compared both cases - with or without relu (in chapter II.b). I trained up to 30 epoches and found out that the results didn’t change much after 12 epoches. In epoch 12, the result is pretty good. For 12 epoches, the training time is about 15mins on sfucloud gpu. The comparison on different epoches can be seen in chapter II.e.

Besides, I compared different learning rates (0.1 and 0.001), and the result is in chapter II.f.

The test data ratio is changed to 0.3 in order to have more test results. The test image accuracy is calculated according to different training conditions. The figures shown in the following chapters are Input Image - Ground-truth Label - Predicted Results (from left to right).

**II.a With vs Without Batch Normalization**

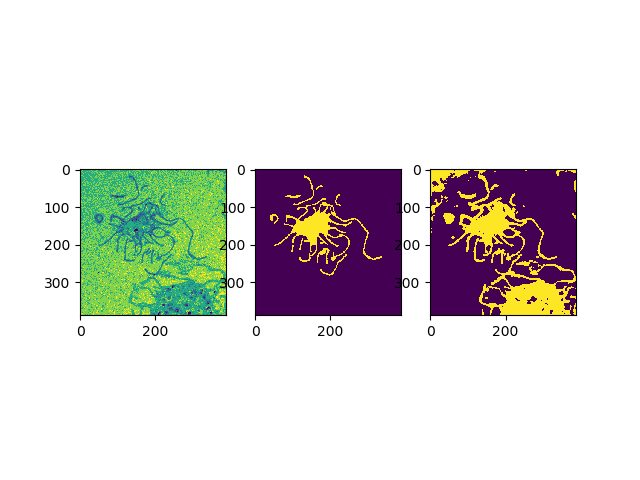
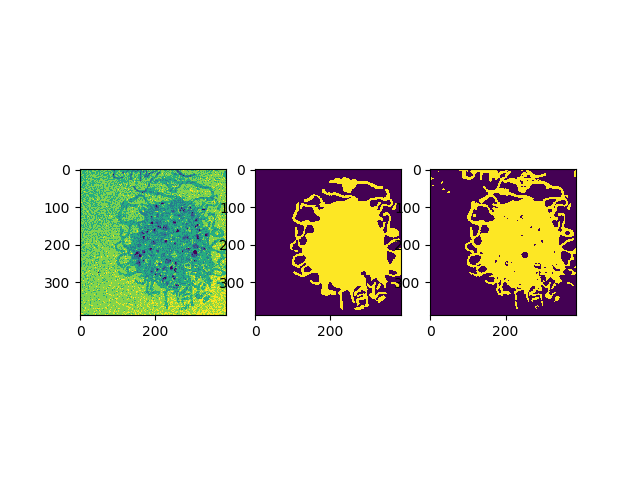
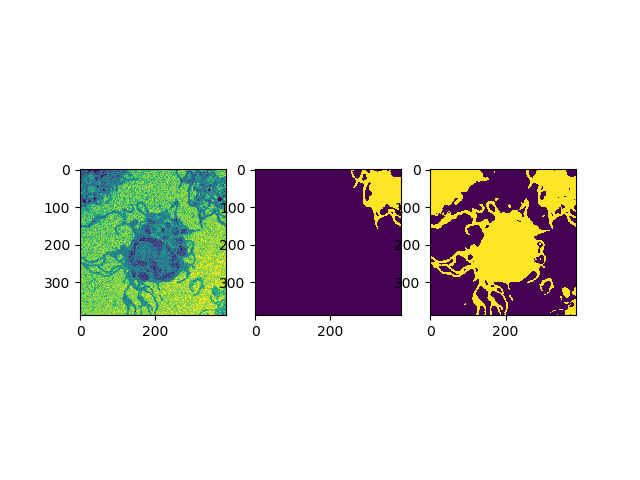
I compared the results under different batch normalization conditions(without Batch Normalization, with Batch Normalization placed before ReLU layer, or placed after ReLU layer). The neural networks are trained with same input images, and same training epoch 12. The learning rate is 0.001. No data augmentation is used for this comparison. The pixel-wise accuracy is calculated for each test image.

*Table 1 Comparison of results under different Batch Normalization conditions*

|  |  |  |  |
| --- | --- | --- | --- |
| Image Name | Accuracy (pixel-wise) | | |
| Without BN | With BN before ReLU | With BN after ReLU |
| BMMC\_43.tif | 0.883 | 0.642 | 0.616 |
| BMMC\_44.tif | 0.626 | 0.928 | 0.927 |
| BMMC\_45.tif | 0.918 | 0.814 | 0.794 |
| BMMC\_46.tif | 0.564 | 0.566 | 0.565 |
| BMMC\_48.tif | 0.676 | 0.848 | 0.834 |
| BMMC\_49.tif | 0.981 | 0.981 | 0.981 |
| BMMC\_50.tif | 0.725 | 0.781 | 0.749 |
| BMMC\_51.tif | 0.504 | 0.830 | 0.819 |
| BMMC\_52.tif | 0.119 | 0.851 | 0.888 |
| BMMC\_53.tif | 0.326 | 0.353 | 0.337 |
| Average | 0.632 | 0.759 | 0.751 |

We can see form the table above that, although our batch size is 1, with batch normalization, the accuracy is still higher as it can better center and normalize the image. I also noticed that batch normalization placed before ReLU is slightly better than placed after ReLU. Below are some examples of predicted results in the condition batch normalization placed before ReLU.

It can be seen that some ground truth labels are not very accurate, as it missed segmentation of some cells. The predict results can be even better than the ground truth(e.g BMMC\_43.tif)

\

*Figure 13 Test results under condition batch normalization placed before ReLU*

*(from top to bottom BMMC\_43.tif BMMC\_44.tif BMMC\_45.tif)*

**II.b With vs Without ReLU in the last step**

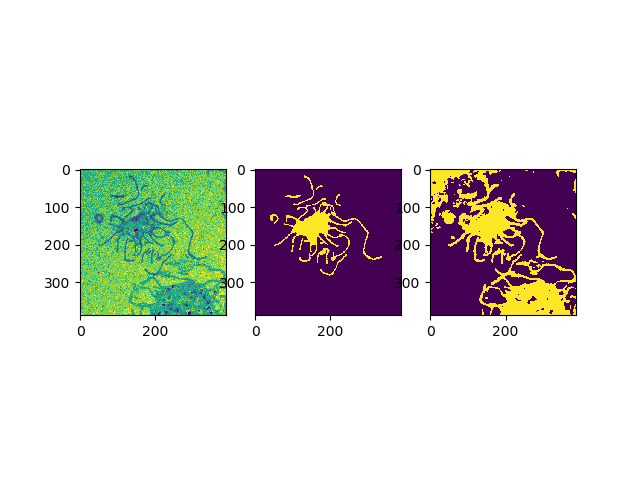
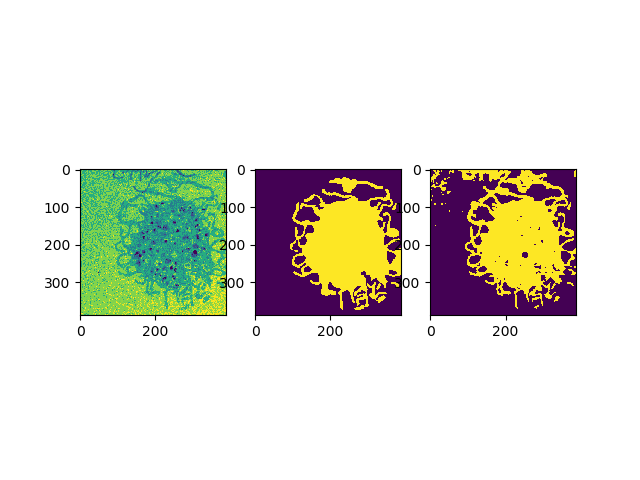
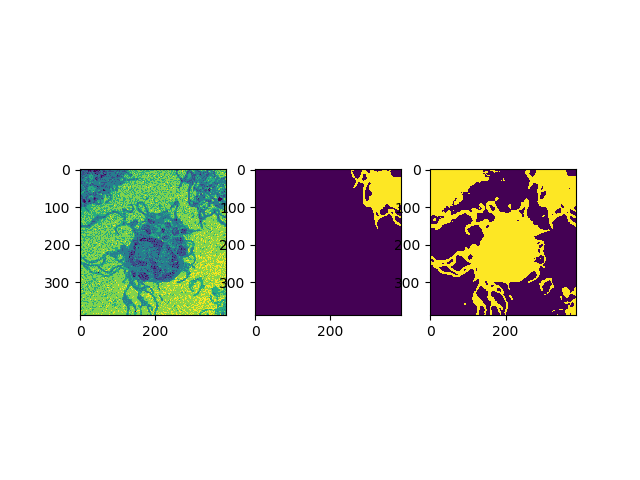
As mentioned before, I deviated from the original network structure and compared both cases - with or without the ReLU in the last step. The neural networks are trained with same training epoch 12. . The learning rate is 0.001. All data augmentation is used for this comparison.

*Table 2 Comparison of results on with or without ReLU in the last step*

|  |  |  |
| --- | --- | --- |
| Image Name | Accuracy (pixel-wise) | |
| Without ReLU | With  ReLU |
| BMMC\_43.tif | 0.642 | 0.622 |
| BMMC\_44.tif | 0.935 | 0.912 |
| BMMC\_45.tif | 0.810 | 0.771 |
| BMMC\_46.tif | 0.565 | 0.569 |
| BMMC\_48.tif | 0.849 | 0.836 |
| BMMC\_49.tif | 0.981 | 0.981 |
| BMMC\_50.tif | 0.775 | 0.761 |
| BMMC\_51.tif | 0.831 | 0.818 |
| BMMC\_52.tif | 0.871 | 0.865 |
| BMMC\_53.tif | 0.347 | 0.370 |
| Average | 0.761 | 0.751 |

We can see form the table that, not using ReLU in the last step is slightly better.

Below are some test results of using ReLU in the last step. Same test images are choosen for comparison.



*Figure 14 Test results under condition using ReLU in the last step*

*(from top to bottom BMMC\_43.tif BMMC\_44.tif BMMC\_45.tif)*

**II.c With vs Without Data Augmentation Strategies**

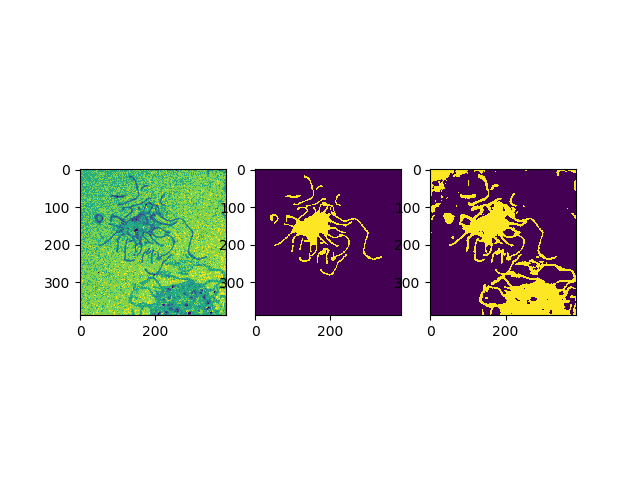
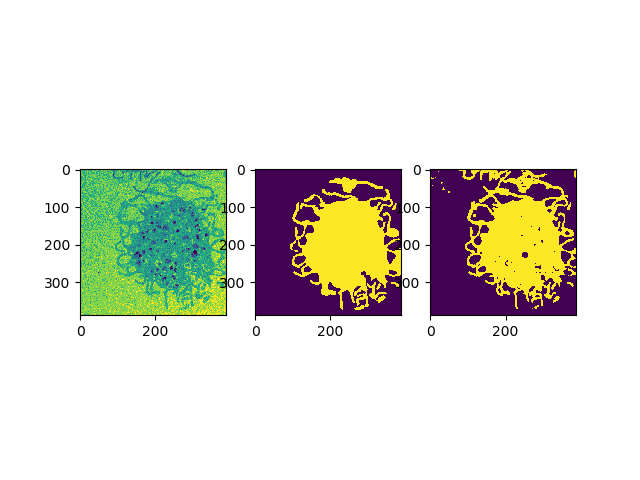
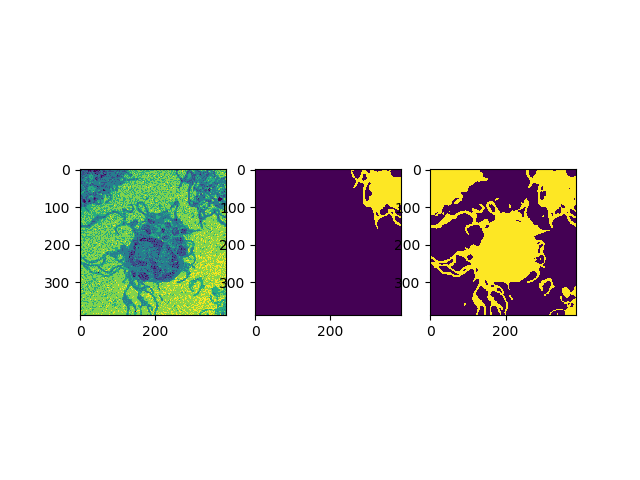
In this part, I compare with all data augmentation strategies case with without any data augmentation. In the next part, I compare the results of using different data augmentation strategies. The neural networks are trained with same training epoch 12 and also epoch 18, as I want to train more epochs to see changes in this case. The learning rate is 0.001.

*Table 3 Comparison of results on with or without data augmentation strategies*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Image Name | Accuracy (pixel-wise) | | | |
| With Data Aug(12 epochs) | Without Data Aug(12 epochs) | With Data Aug  (18 epochs) | Without Data Aug(18 epochs) |
| BMMC\_43.tif | 0.642 | 0.642 | 0.667 | 0.643 |
| BMMC\_44.tif | 0.935 | 0.928 | 0.942 | 0.922 |
| BMMC\_45.tif | 0.810 | 0.814 | 0.843 | 0.768 |
| BMMC\_46.tif | 0.565 | 0.566 | 0.564 | 0.568 |
| BMMC\_48.tif | 0.849 | 0.848 | 0.855 | 0.839 |
| BMMC\_49.tif | 0.981 | 0.981 | 0.981 | 0.981 |
| BMMC\_50.tif | 0.775 | 0.781 | 0.788 | 0.759 |
| BMMC\_51.tif | 0.831 | 0.830 | 0.835 | 0.811 |
| BMMC\_52.tif | 0.871 | 0.851 | 0.931 | 0.908 |
| BMMC\_53.tif | 0.347 | 0.353 | 0.327 | 0.363 |
| Average | 0.761 | 0.759 | 0.773 | 0.756 |

We can see form the table that, using data augmention can increase the accuracy. The benefit is more obvious when training with more epochs as it fakes more training data.

Below are some test results of using data augmentation. Same test images are choosen for comparison.



*Figure 15 Test results using Data Augmentation Strategy*

*(from top to bottom BMMC\_43.tif BMMC\_44.tif BMMC\_45.tif)*

**II.d Comparison on Different Data Augmentation Strategies**

I tried different data augmentation strategies with different options.

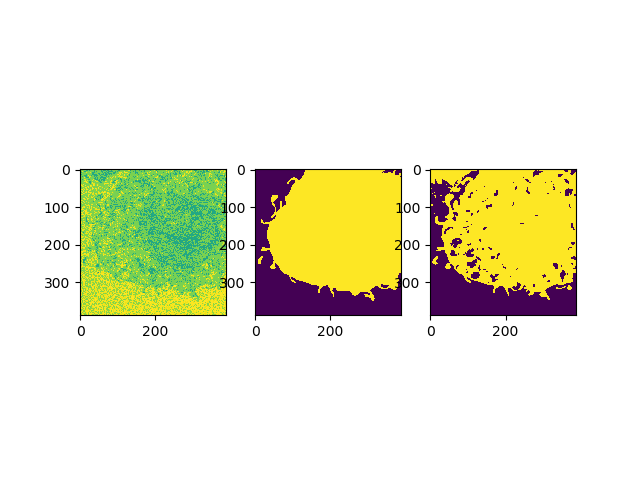
* Flip the image horizontally or vertically
* Zoom the image by 1/0.95 or 1/0.9
* Rotate the image 90 or 180 degree
* Gamma correction 0.8
* Elastic deformations with sigma in range (6, 12)

I applied each data augmentation strategy separately to compare the different characteristics of each augmentation strategy. The neural networks are trained with same training epoch 12. The learning rate is 0.1.

*Table 4 Comparison of results on different data augmentation strategies*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Image Name | Accuracy (pixel-wise) | | | | |
| Flip | Zoom | Rotate | Gamma | Elastic deformation |
| BMMC\_52.tif | 0.930 | 0.884 | 0.863 | 0.930 | 0.935 |
| BMMC\_53.tif | 0.674 | 0.430 | 0.383 | 0.755 | 0.915 |
| BMMC\_7.tif | 0.708 | 0.851 | 0.864 | 0.708 | 0.751 |
| Average | 0.771 | 0.722 | 0.703 | 0.798 | 0.867 |

We can see from the table that we may be able to benefit more from elastic deformation strategy. Especially for the prediction result of BMMC\_53.tif, the benefit is significant. As our dataset is limited and also fake training data is generated randomly, to draw a more precise conclusion, we need more information on that.



*Figure 16 Test results using only elastic deformation strategy (BMMC\_53.tif)*

**II.e Comparison on Different Epochs**

I compared the test result on different training epochs. The data is trained up to 30 epochs. All data augmentation is used in order to generate better weights. The learning rate is 0.001.

*Table 5 Comparison of results on different epochs*

|  |  |  |  |
| --- | --- | --- | --- |
| Image Name | Accuracy (pixel-wise) | | |
| Epochs 12 | Epochs 18 | Epochs 30 |
| BMMC\_43.tif | 0.642 | 0.667 | 0.682 |
| BMMC\_44.tif | 0.935 | 0.942 | 0.953 |
| BMMC\_45.tif | 0.810 | 0.843 | 0.841 |
| BMMC\_46.tif | 0.565 | 0.564 | 0.566 |
| BMMC\_48.tif | 0.849 | 0.855 | 0.858 |
| BMMC\_49.tif | 0.981 | 0.981 | 0.981 |
| BMMC\_50.tif | 0.775 | 0.788 | 0.770 |
| BMMC\_51.tif | 0.831 | 0.835 | 0.827 |
| BMMC\_52.tif | 0.871 | 0.931 | 0.959 |
| BMMC\_53.tif | 0.347 | 0.327 | 0.333 |
| Average | 0.761 | 0.773 | 0.777 |

We can see that the average accuracy increases with more epochs in this case. Due to the limit time and GPU, most test results in other chapters are obtained based on 12 epochs.

**II.f Comparison on Different Learning Rate**

Finally, I compared the test result on different learning rate (0.001 and 0.01) with or without data augmentation. The training epoch is 12.

*Table 6 Comparison of results on different learning rates*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Image Name | Accuracy (pixel-wise) | | | |
| With Data Aug  lr=0.001 | With Data Aug  lr=0.1 | Without Data Aug  lr=0.001 | Without Data Aug  lr=0.1 |
| BMMC\_43.tif | 0.642 | 0.643 | 0.642 | 0.265 |
| BMMC\_44.tif | 0.935 | 0.921 | 0.928 | 0.564 |
| BMMC\_45.tif | 0.810 | 0.793 | 0.814 | 0.271 |
| BMMC\_46.tif | 0.565 | 0.568 | 0.566 | 0.834 |
| BMMC\_48.tif | 0.849 | 0.845 | 0.848 | 0.584 |
| BMMC\_49.tif | 0.981 | 0.981 | 0.981 | 0.955 |
| BMMC\_50.tif | 0.775 | 0.776 | 0.781 | 0.370 |
| BMMC\_51.tif | 0.831 | 0.824 | 0.830 | 0.689 |
| BMMC\_52.tif | 0.871 | 0.864 | 0.851 | 0.930 |
| BMMC\_53.tif | 0.347 | 0.364 | 0.353 | 0.795 |
| Average | 0.761 | 0.758 | 0.759 | 0.626 |

We can see that the average accuracy under learning rate 0.001 is higher than that under learning rate 0.1. With data augmentation, the result is better and the influence of learning rate is smaller.

**II.g Analysis of particular results**

We notice that the predict result for input image BMMC\_46.tif and BMMC\_53.tif are relatively lower compared to other images in our dataset. It’s mainly because the two images are much brighter which lead to the difficulty in segmentation. More training epochs or data augmentation will be needed to generate a good result for them.

**III. Bonus**

**III.a Internal Energy**

I implemented the internal energy *getInternalEnergyMatrixBonus* function. More implementation details can be found in code.