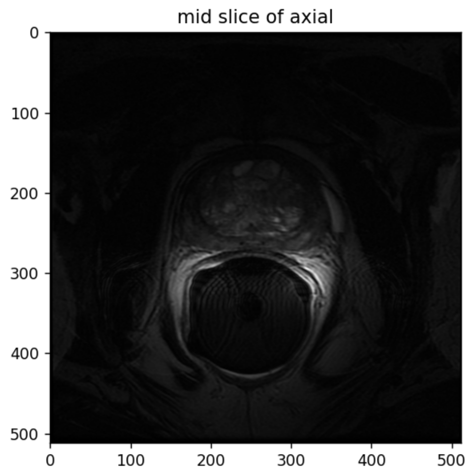
Yuqi Yang, 20150516

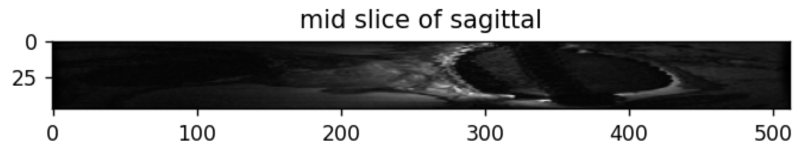
Q1:

Image of each of the 3 center slices of Case00.npy:

axial



sagittal



coronal

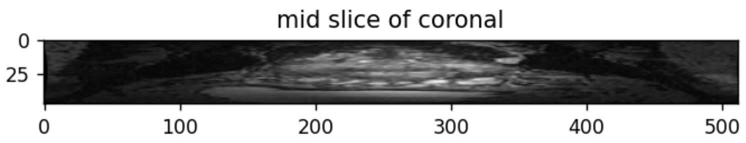
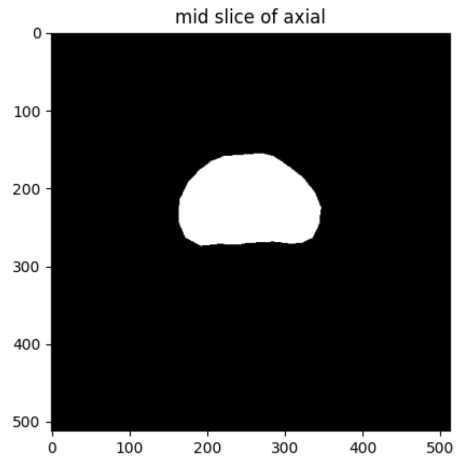
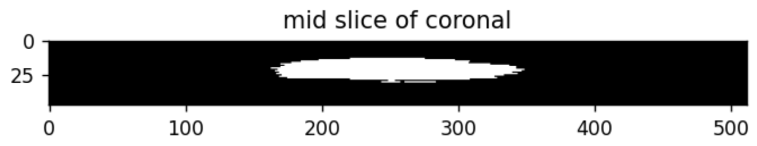


Image of each of the 3 center slices of Case00_segmentation.npy:

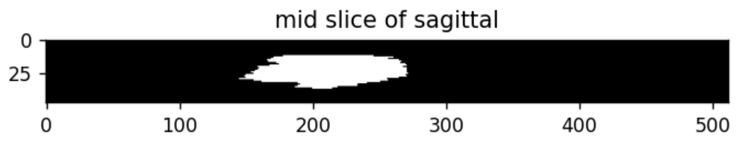
Axial ground truth



Sagittal ground truth



Coronal ground truth



The ground truth image clearly indicates that range and the size of the prostate as it should, however it is a bit hard to distinguish the outer range of the prostate in the regular MRI scan by human eyes, the scan is blurry perhaps due to the way the image is enhanced. The location of the prostate doesn't get clearer as I adjust the scalar opacity mapping using the 3D slicer app's volume rendering, but the structure of the other body parts (maybe bones) are significantly enhanced. Then I played around with the volume section in the 3d slicer, where the "CT-Bone" setting worked the best at enhancing the outline of the prostate.

Image of each of the 3 center slices of Case00.npy in 3D slicer:

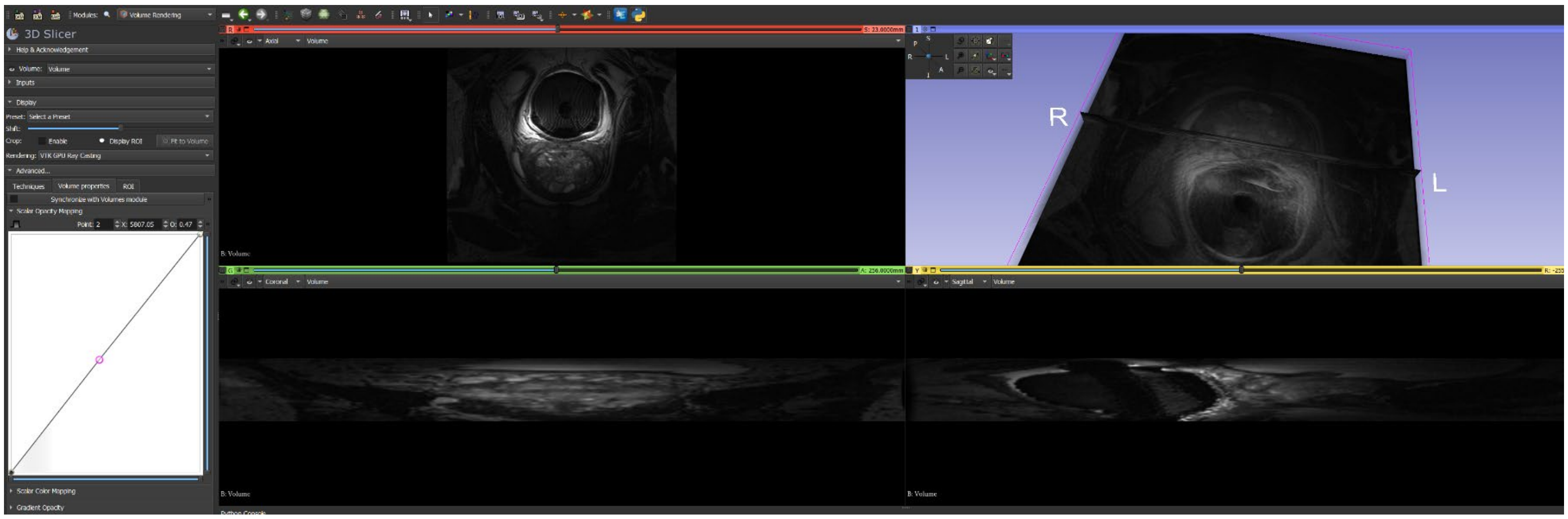
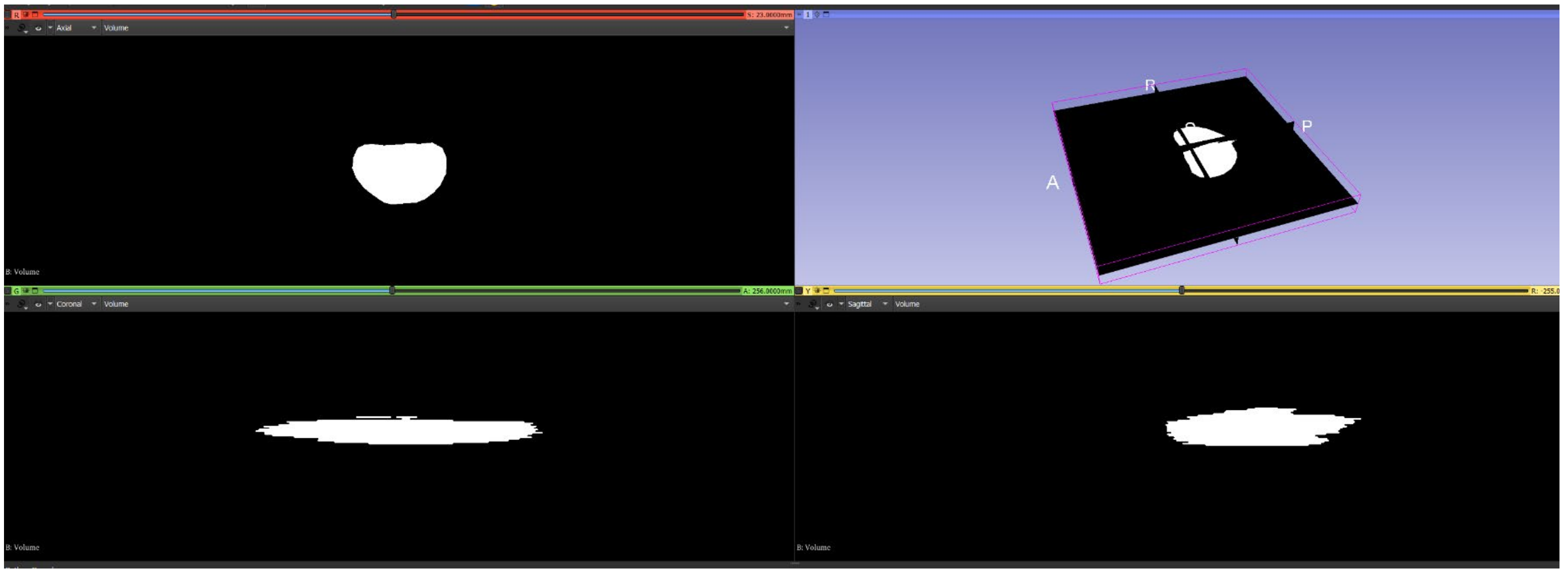
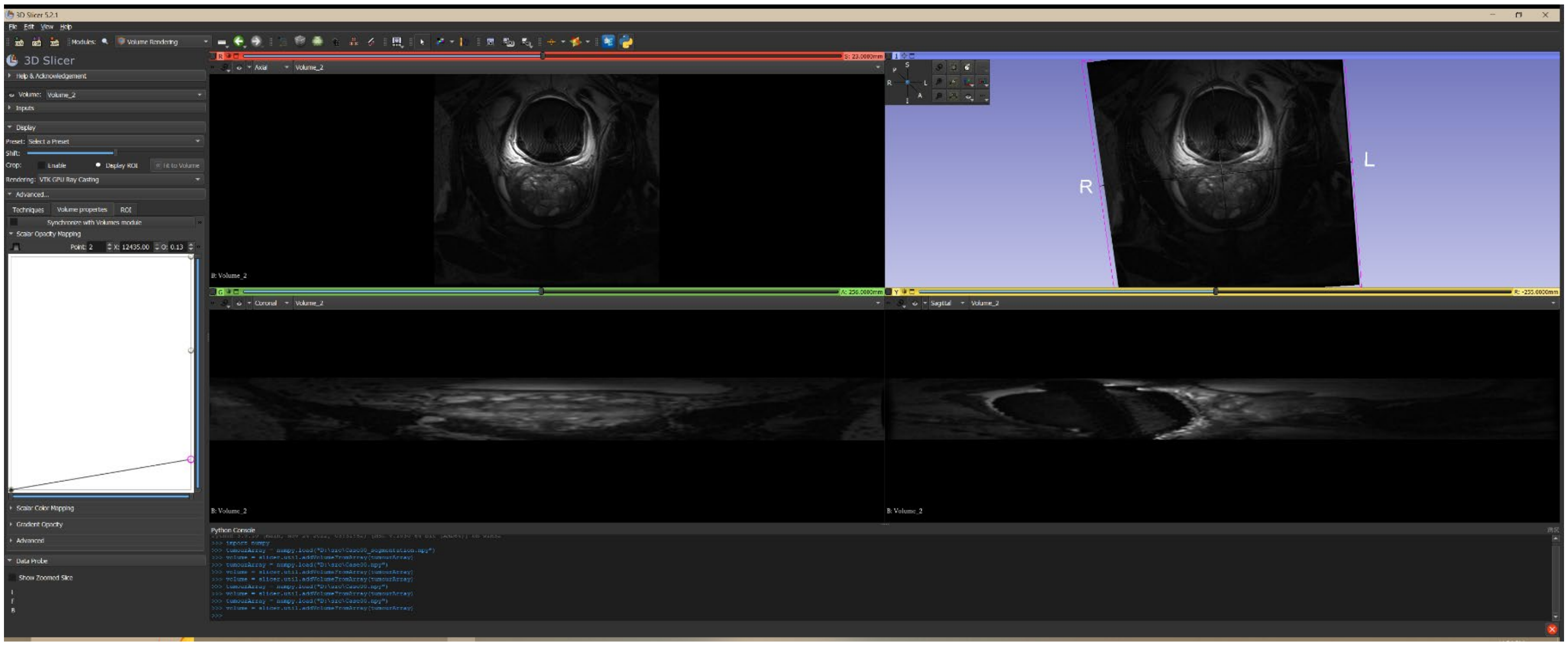


Image of each of the 3 center slices of Case00_segmentation.npy in 3D slicer:



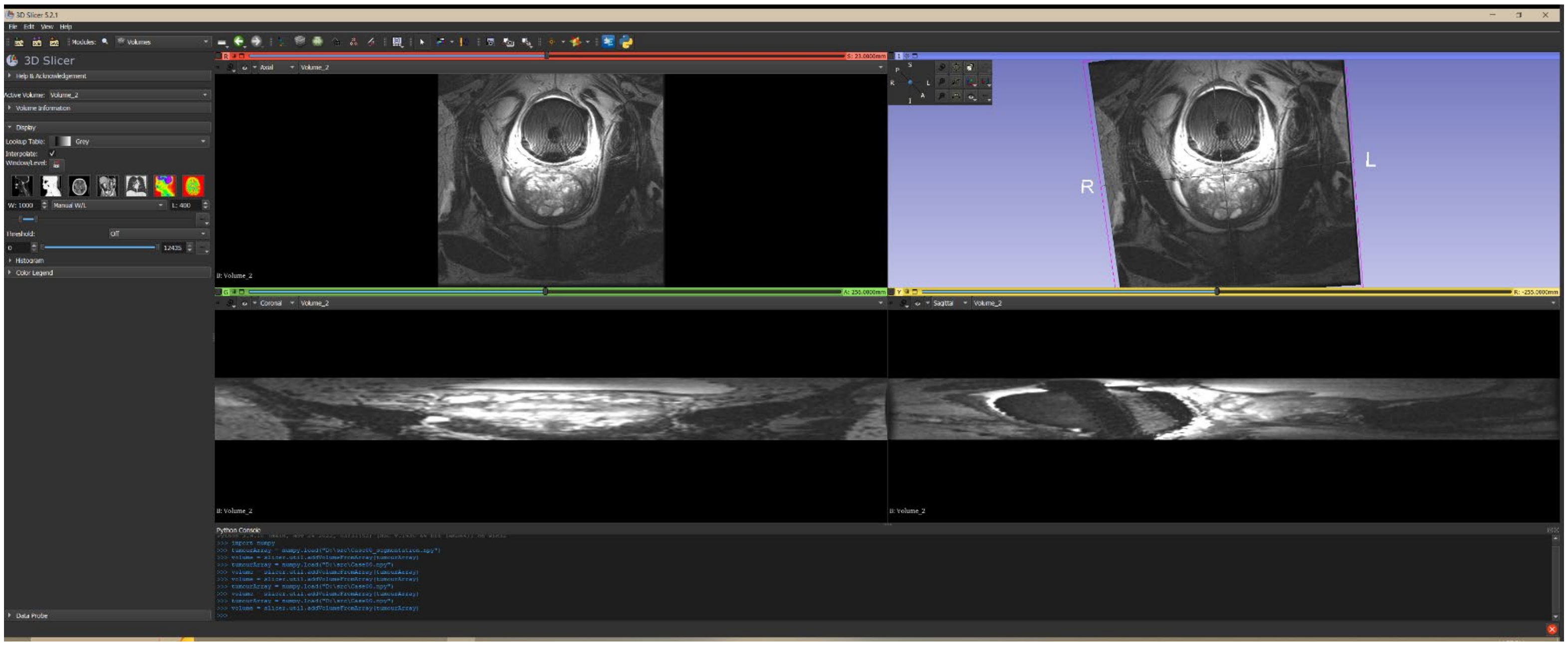
Yuqi Yang, 20150516

Image of each of the 3 center slices of Case00.npy in 3D slicer Volume Rendering:



```
>>> import numpy
>>> tumourArray = numpy.load("D:\src\Case00_segmentation.npy")
>>> volume = slicer.util.addVolumeFromArray(tumourArray)
>>> tumourArray = numpy.load("D:\src\Case00.npy")
>>> volume = slicer.util.addVolumeFromArray(tumourArray)
>>> volume = slicer.util.addVolumeFromArray(tumourArray)
>>> tumourArray = numpy.load("D:\src\Case00.npy")
>>> volume = slicer.util.addVolumeFromArray(tumourArray)
>>> tumourArray = numpy.load("D:\src\Case00.npy")
>>> volume = slicer.util.addVolumeFromArray(tumourArray)
>>>
```

Image of each of the 3 center slices of Case00_segmentation.npy in 3D slicer Volume:



```
>>> import numpy
>>> tumourArray = numpy.load("D:\src\Case00_segmentation.npy")
>>> volume = slicer.util.addVolumeFromArray(tumourArray)
>>> tumourArray = numpy.load("D:\src\Case00.npy")
>>> volume = slicer.util.addVolumeFromArray(tumourArray)
>>> volume = slicer.util.addVolumeFromArray(tumourArray)
>>> tumourArray = numpy.load("D:\src\Case00.npy")
>>> volume = slicer.util.addVolumeFromArray(tumourArray)
>>> tumourArray = numpy.load("D:\src\Case00.npy")
>>> volume = slicer.util.addVolumeFromArray(tumourArray)
>>>
```

After examining the entire 50 datasets, the maximum depth of the volumes is 512 shades, 9 bits (or 54 layers). MRI scans contain different number of slices, some has 47, some has 29 and so on. The dimensions of the volumes are 512 by 512, 320 by 320, 256 by 256 and 384 by 384. The values are not the same foe all volumes in the dataset.

Q3:

Ideally training data should be around 70%, where validation and test split the rest of the 30%, now since the testing data only consists of 10% of the entire dataset, we can split the data into 70% training, 20% validation, and 10% testing.

training	validation	testing
[21, 50, 60, 43, 16, 69, 97, 67, 94, 29, 86, 89, 74, 25, 40, 7, 70, 63, 13, 47, 6, 55, 39, 24, 3, 68, 72, 11, 18, 57, 44, 80, 76, 64, 82, 95, 85, 52, 93, 14, 10, 32, 51, 96, 84, 87, 65, 27, 28, 90, 42, 88, 92, 49, 23, 34, 19, 12, 8, 15, 0, 37, 2, 75, 56, 30, 71, 78, 81, 38]	[22, 58, 33, 36, 79, 48, 45, 77, 17, 53, 5, 46, 31, 9, 41, 91, 66, 98, 20, 83]	[35, 62, 59, 26, 1, 99, 61, 54, 4, 73]

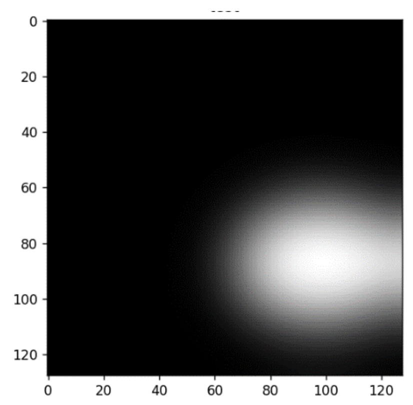
Q4:

Since the image datasets is blurred out to different extent, I wanted to try to remove the blur by sharpening the picture and then reduce the noise level. I played around with the sharpening kernels “[-1,-1,-1], [-1,9,-1], [-1,-1,-1]”, “[[0,-1,0], [-1,5,-1], [0,-1,0]]”, “[[0,-1,0], [-1,6,-1], [0,-1,0]]” using the filter2D() function. Out of the 3, “[[0,-1,0], [-1,6,-1], [0,-1,0]]” did the best and distinguishing this smoothing case.

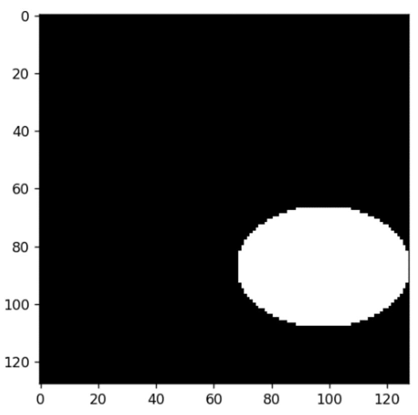
I then tried to threshold the picture using contrast enhancements, after comparing many images that are smoothed to different level with its original segmentation, I conclude that 100-120 is a good threshold value for preserving most of the original details without losing too much information. However, through observing the sharpened image, 230 appears to be the best threshold to use.

Surprisingly, thresholding after sharpening does not perform better than just thresholding. Therefore, I wanted to see if it’s just this data perform like so. I then test on another data, where the thresholding after sharpening procedure performs the best among all. Therefore, I decide to use the thresholding after sharpening method for my data pre-processing part.

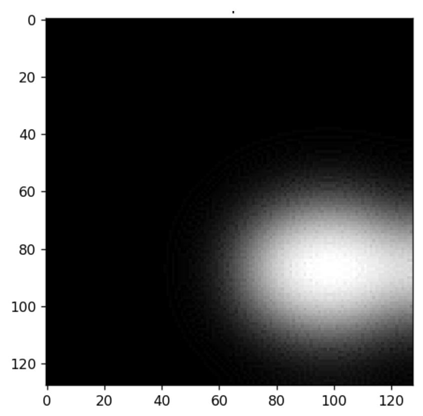
Original data 1



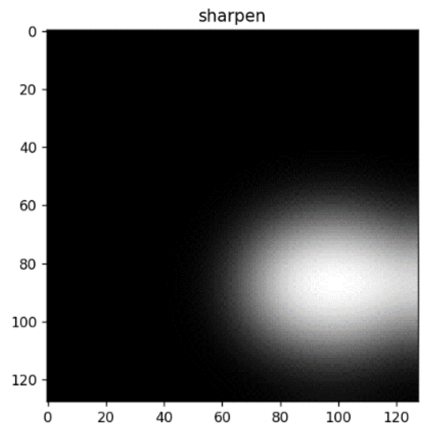
Segmentation data 1



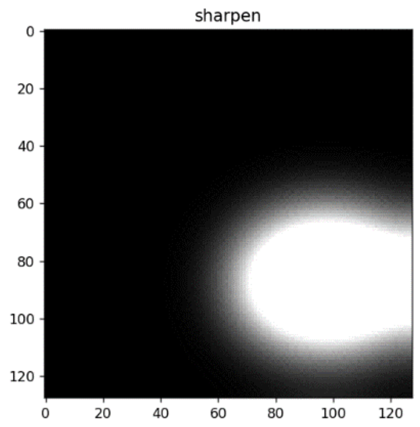
“[-1,-1,-1], [-1,9,-1], [-1,-1,-1]”



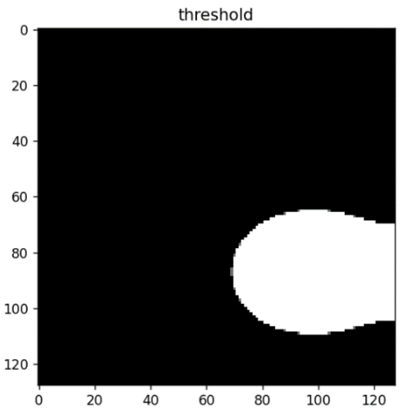
“[[0,-1,0], [-1,5,-1], [0,-1,0]]”



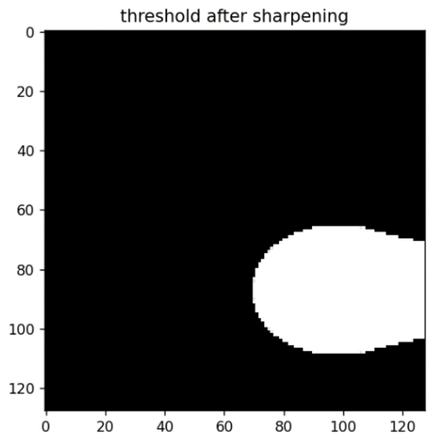
“[[0,-1,0], [-1,6,-1], [0,-1,0]]”



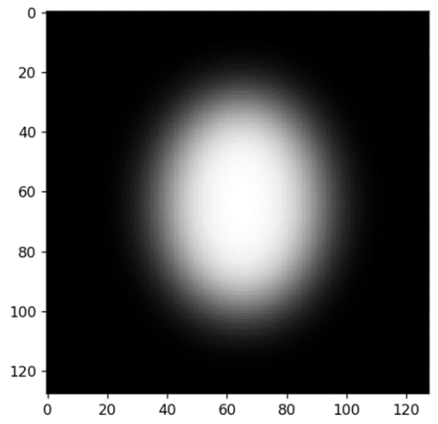
Threshold



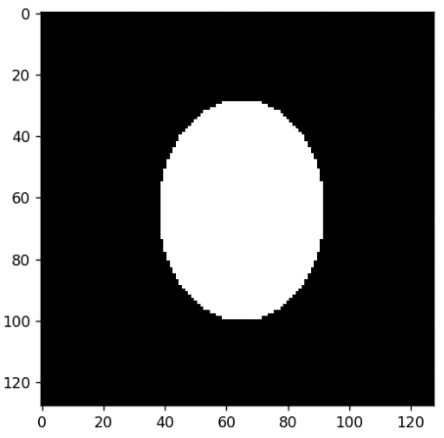
Threshold after sharpening



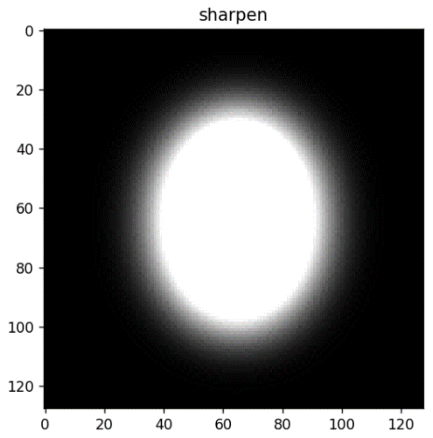
Original data 2



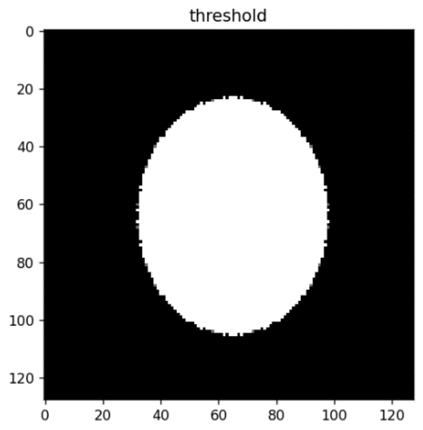
Segmentation 2



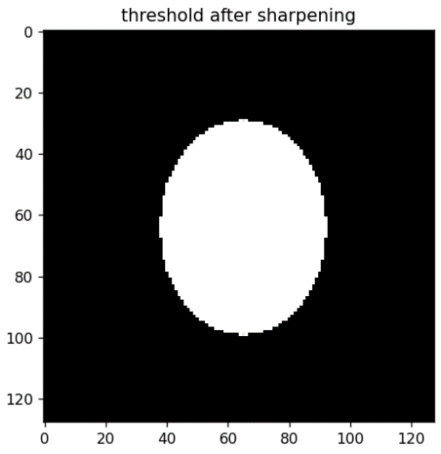
“[[0,-1,0], [-1,6,-1], [0,-1,0]]”



Threshold



Threshold after sharpening



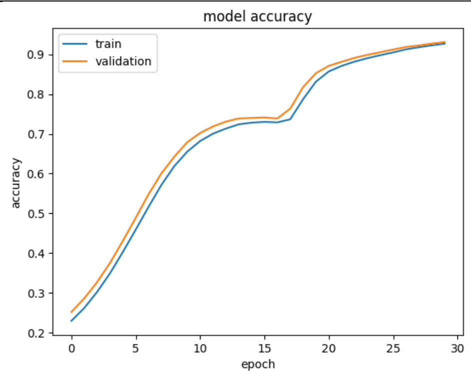
Q5:

Experiment with the learning rate and make note of what this does to the performance of your model. you are responsible for implementing the Dice Coefficient and Hausdorff distance metrics.

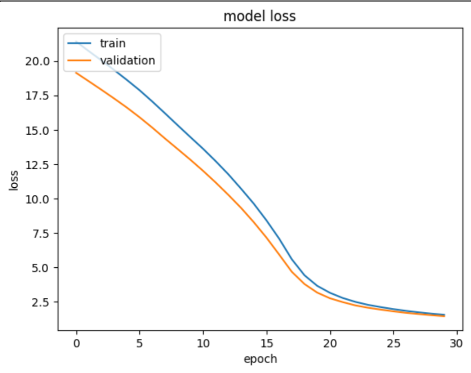
I first start running the test under a learning rate of 0.00001, which can be too small of a learning rate where can come with cost of longer learning time, and may become permanently stuck with a high training error. As we can tell from the table, under an epoch size of 30, the curve of both accuracy and loss isn't smooth, indicating that the model needs more time to train. I then tried to implement 50 epochs, the result is pleasant (as seen in the table). However I think for the complexity and size of this data set, this setup will be too time consuming (9 minutes), therefore I wanted to see if reducing the learning rate will make a difference, I tried a learning rate of 0.001 with EarlyStopping engaging too (monitor val_loss), I also added additional metrics such as precision and recall. The 0.001 learning rate is relatively large , it may cause the model to converge too quickly to a suboptimal solution. The model performed relatively well with epoch of 30, I then was curious to see if it will do the same on their a learning rate of 0.0001. The progression graphs for 0.0001 is relatively similar to 0.001 with epoch equals 30, meaning they performed equally. therefore to save some time I wanted to see if reducing the epoch size will make a difference to its performance, as we can see the model performance as well as it is with previous two hyperparameter setups.

I understand that for categorical classification or semantic segmentation problems, it is hard to get above 99% of accuracy, therefore, I think I'm happy with a learning rate of 0.0001 or 0.001 and epoch size of 20.

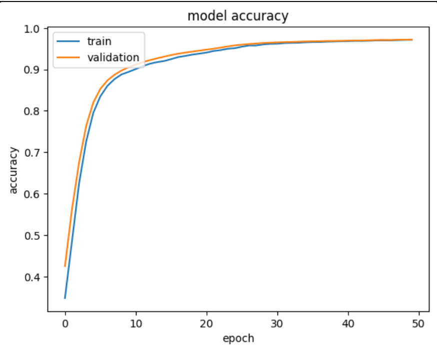
Now moving on to explaining the graphing part, as mentioned above, I choose to add precision and recall besides the regular accuracy as my metrics. The precision metric focuses on Type-I errors(FP), but it cannot measure the existence of Type-II error, which is false negatives. A low precision score (<0.5) means your classifier has a high number of false positives which can be an outcome of imbalanced class or untuned model hyperparameters. In the other hand, the recall metric focuses on type-II errors(FN). Recall towards 1 will signify that your model didn't miss any true positives.



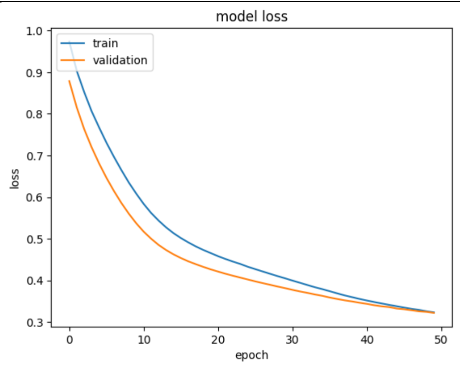
Epoch = 30 LR = 0.00001



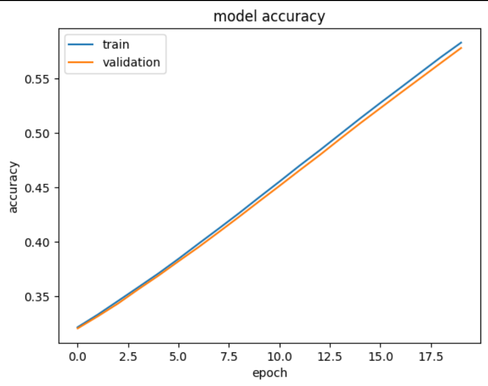
Epoch = 30 LR = 0.00001

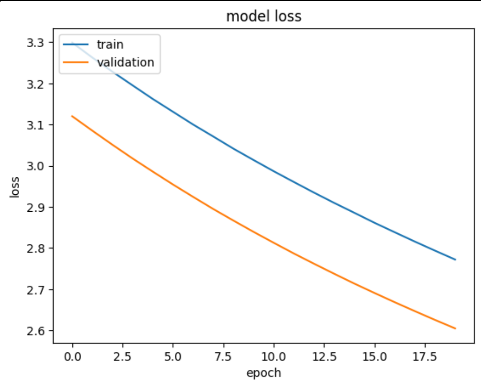


Epoch = 50 LR = 0.00001

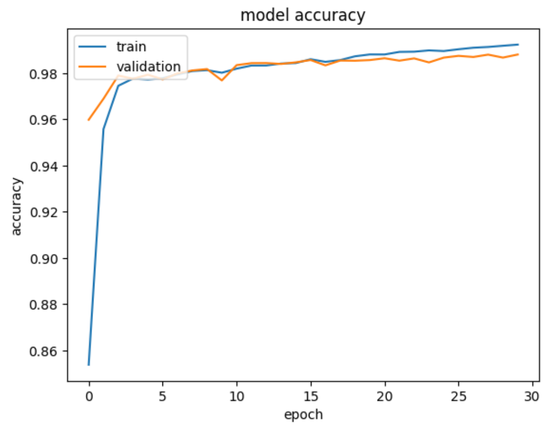
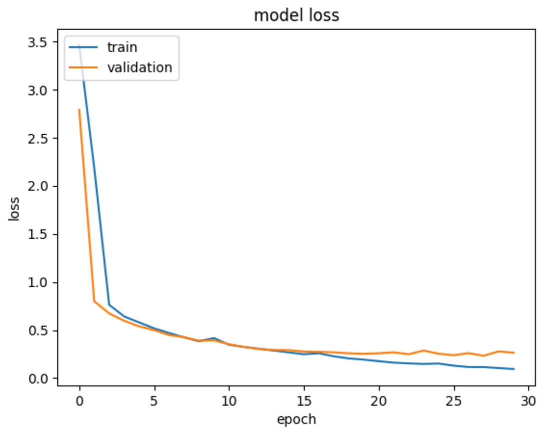
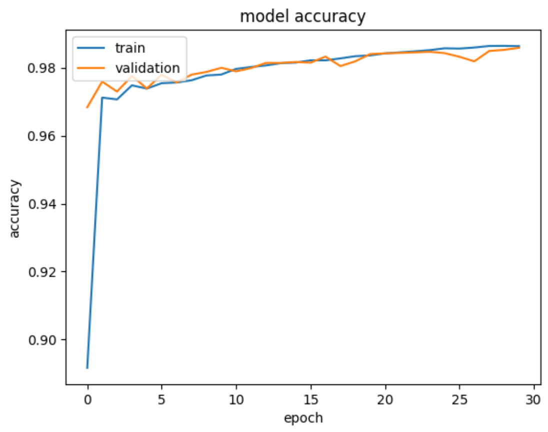
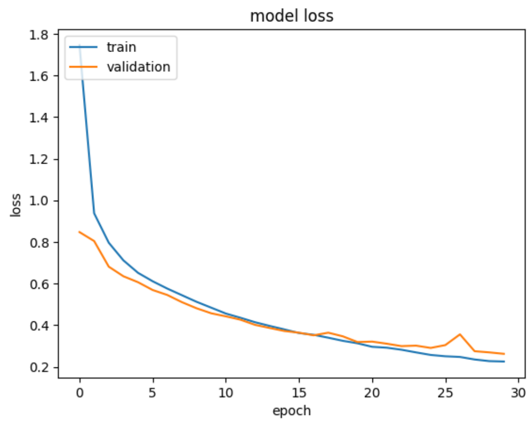
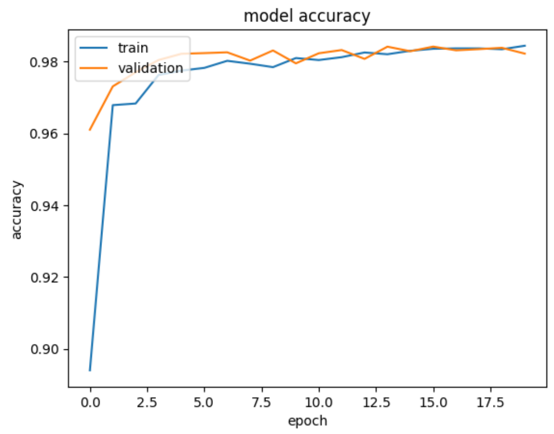
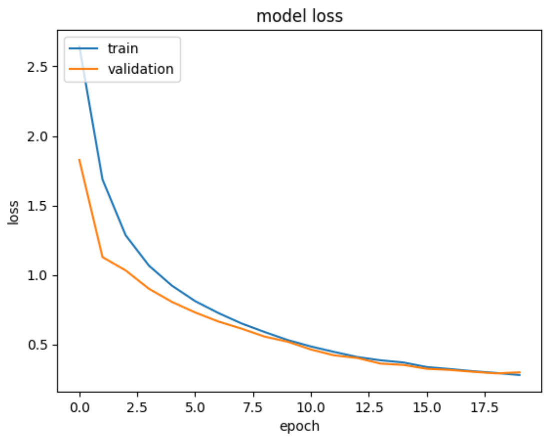


Epoch = 50 LR = 0.00001





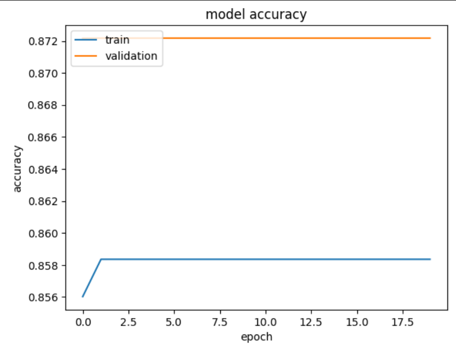
Yuqi Yang, 20150516

Epoch = 20 LR = 0.000001	Epoch = 20 LR = 0.000001
 Epoch = 30 LR = 0.001	 Epoch = 30 LR = 0.001
 Epoch = 30 LR = 0.0001	 Epoch = 30 LR = 0.0001
 Epoch = 20 LR = 0.0001	 Epoch = 20 LR = 0.0001

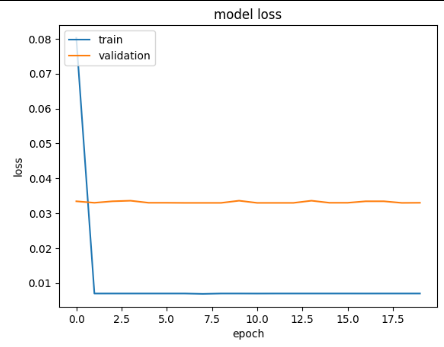
Q6:

Yuqi Yang, 20150516

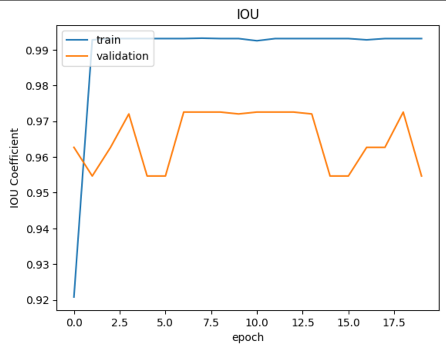
I first tried the same hyperparameter I used for the standard model on the IoU loss model, the result is not ideal. The flat lines indicate the accuracy or loss didn't improve at all or only slightly over the period of time, this could be due to the model converge to a optimal solution too quickly. I try to decrease the learning rate to 0.00001 to see if it makes a difference, and I also engaged early stopping again. As we can see the model performed significantly better in terms of loss rate (lr=0.0001, epoch=15), however it didn't all converge to a significant optimal solution, therefore I wanted to increase the epoch size too allow more time for the model to learn (epoch=40). However the model doesn't perform as I expected it, the accuracy dropped significantly after 15 epoch, an early stopping was called at epoch 37, although the model loss was minimized to extreme, I would not say this tuning was successful even I know I'm not overfitting here, since the val_loss is getting better and better (lr=0.00001 epoch=40). I wanted to see if decrease the learning rate slightly will help , I tried 0.000001 and epoch equals 50. the result images show that the learning rate is too large that the model would require more epochs to compromise to the learning rate.



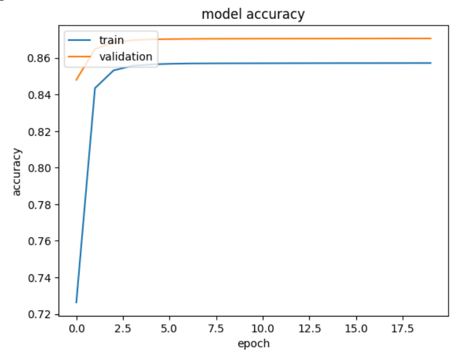
Epoch = 20 LR = 0.001



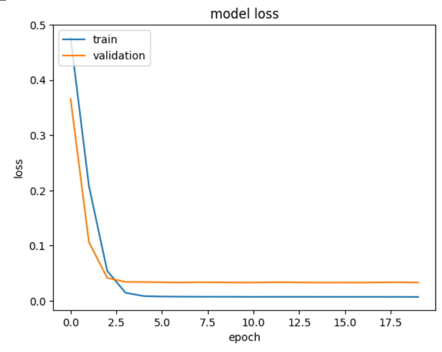
Epoch = 20 LR = 0.001



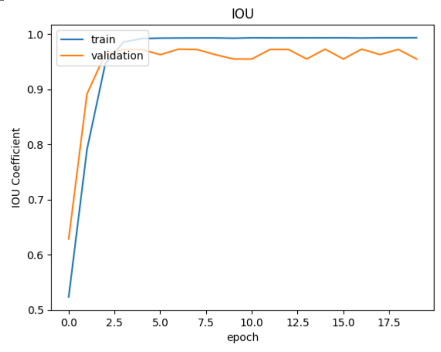
Epoch = 20 LR = 0.001



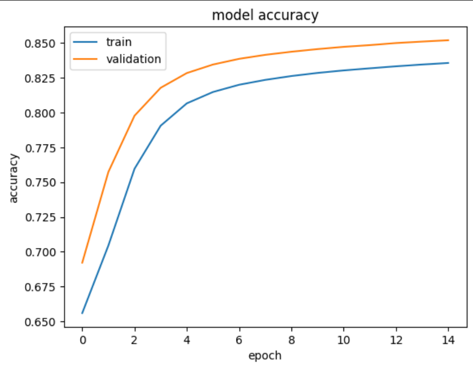
Epoch = 20 LR = 0.0001



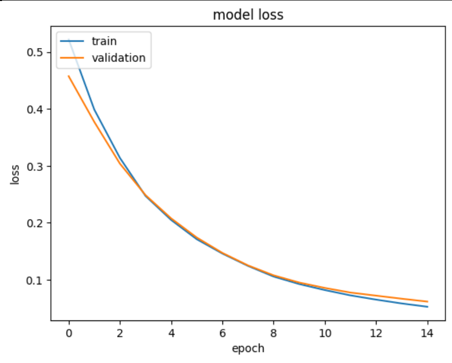
Epoch = 20 LR = 0.0001



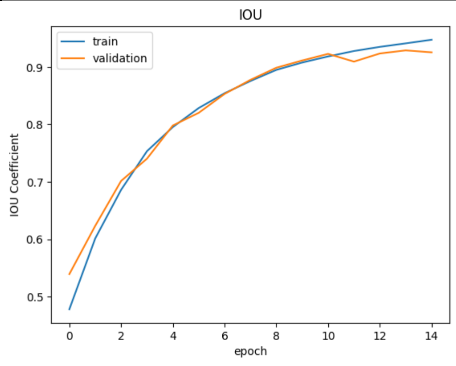
Epoch = 20 LR = 0.0001



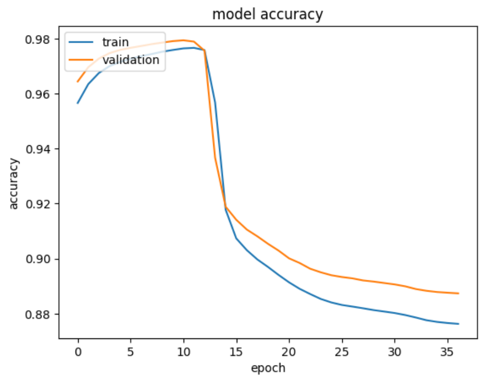
Epoch = 15 LR = 0.00001



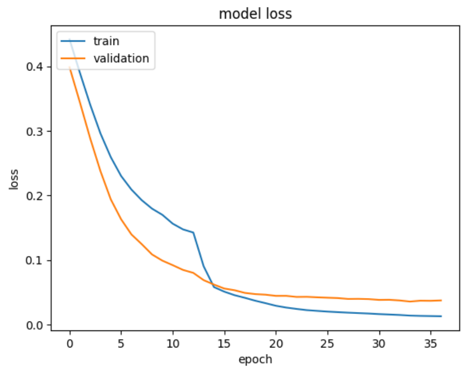
Epoch = 15 LR = 0.00001



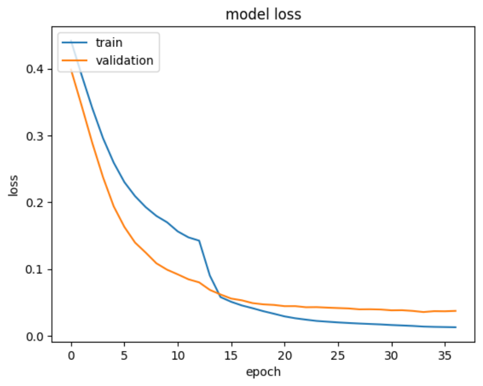
Epoch = 15 LR = 0.0001



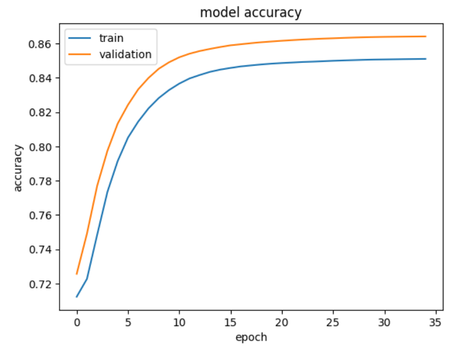
Epoch = 40 LR = 0.00001



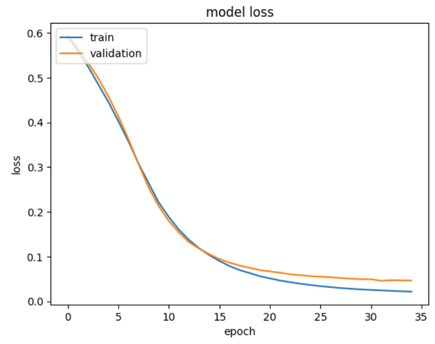
Epoch = 40 LR = 0.00001



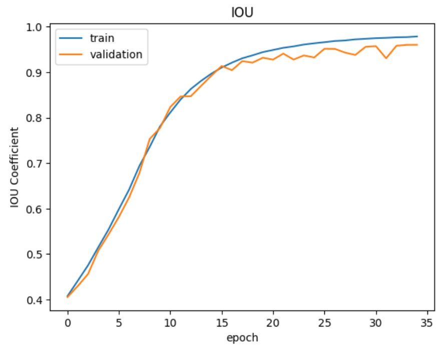
Epoch = 40 LR = 0.00001



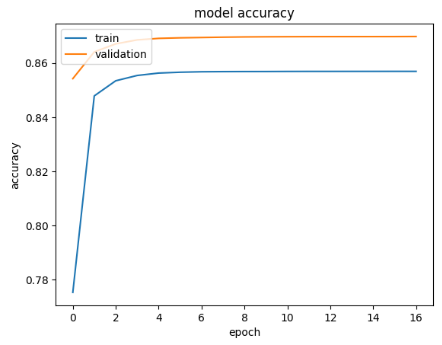
Epoch = 50 LR = 0.00001



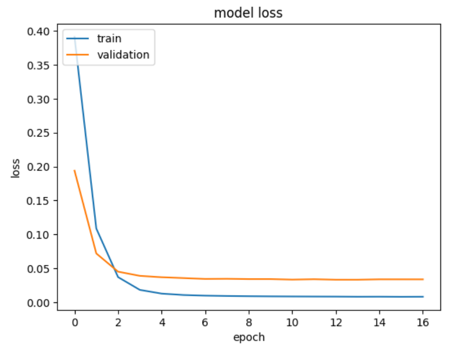
Epoch = 50 LR = 0.00001



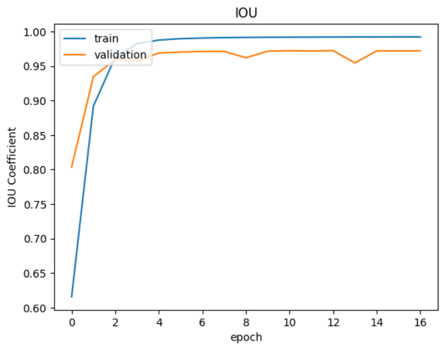
Epoch = 50 LR = 0.00001



Epoch = 30 LR = 0.0001



Epoch = 50 LR = 0.0001



Epoch = 50 LR = 0.0001

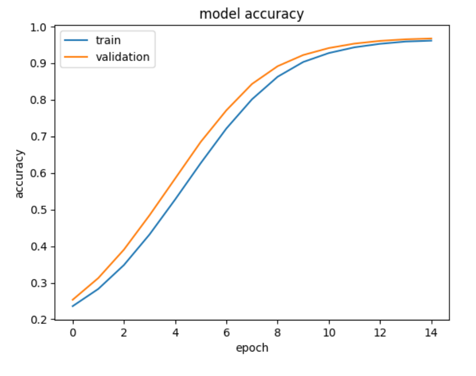
Overall, I'm most satisfied with the hyperparameter of learning rate equals 0.00001 and epoch of 50, but comparing the rate that the model converge to an optimal solution, I think I would rather sacrifice the efficiency for a better performance.

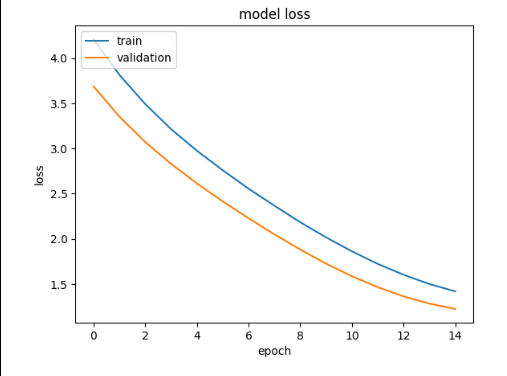
Q7

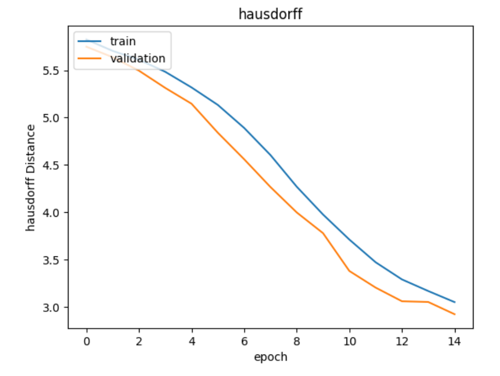
Since the hausdorff distance is the longest distance giving a point in one of the two set you have to travel, it is the greatest of all the distance from a point in one set to the closest point and the other side. therefore we want to minimize the hausdorff distance in between the ground truth image (checkpoint is set to monitor min), and the predicted image array. The below is a good example of a good hausdorff distance, as the training progress, the hausdorff distance decreases.

Yuqi Yang, 20150516

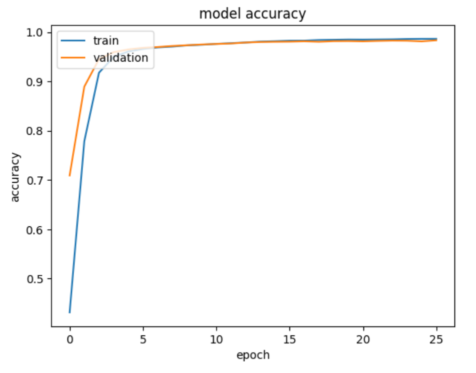
Hausdorff Distance: Epoch = 15 LR = 0.00001, metrics = [hausdorffDistance, "accuracy"]

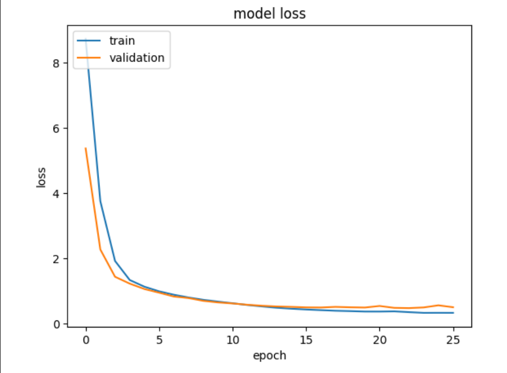


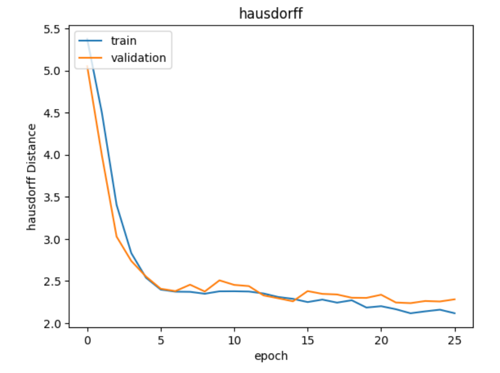




Hausdorff Distance: Epoch = 30 LR = 0.0001, metrics = [hausdorffDistance, "accuracy"]

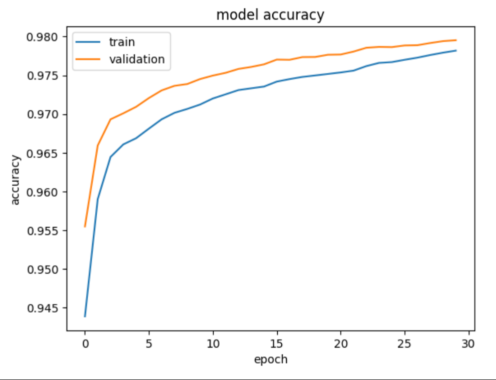


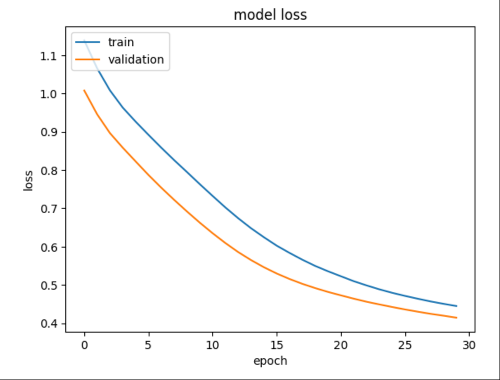


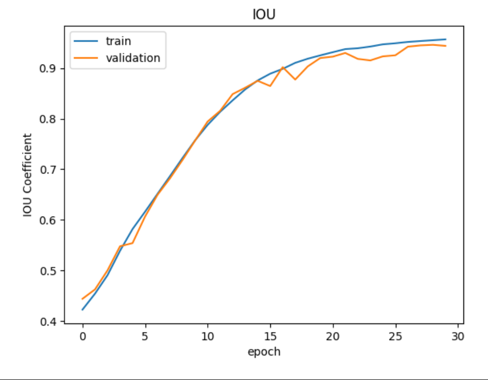


The intersection over union IoU is a good matrix to measure overlaps between two arrays, if the prediction is completely correct, IoU =1, the lower the IOU, the worse the prediction result. Therefore, we want to maximize the IOU index (checkpoint is set to monitor max). I added the early stopping here different from the above example, also with a greater epoch size of 30.

IOU: Epoch = 30 LR = 0.00001, metrics = [IOU, "accuracy"], EarlyStopping(monitor='val_loss', patience=3)

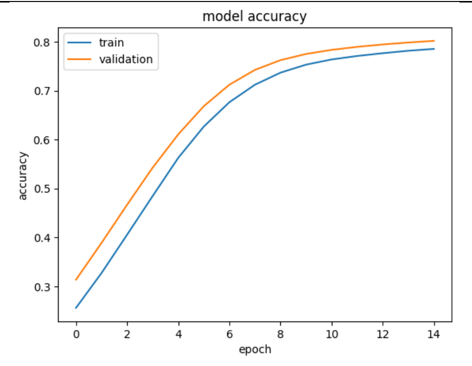
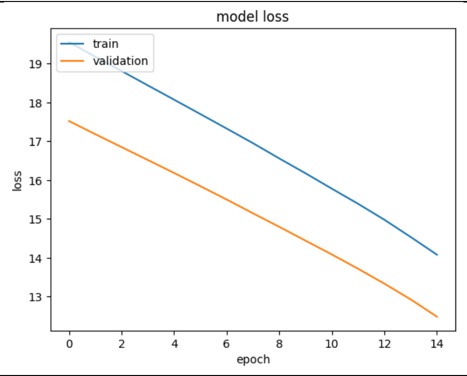
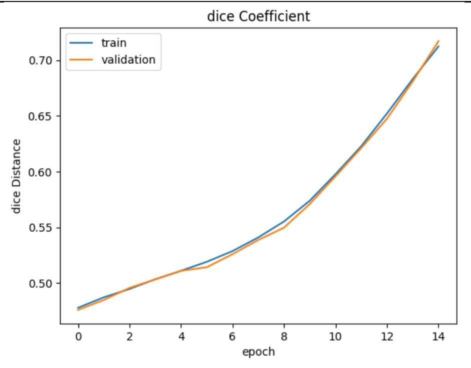
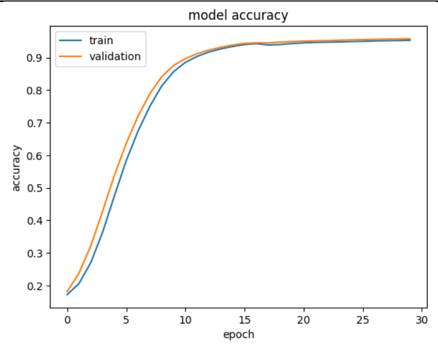
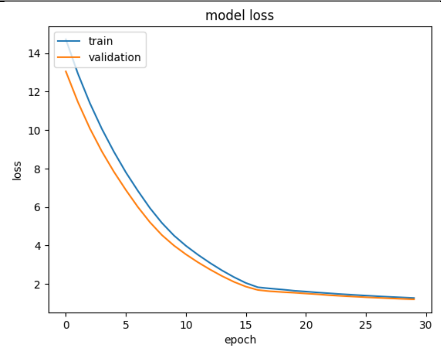
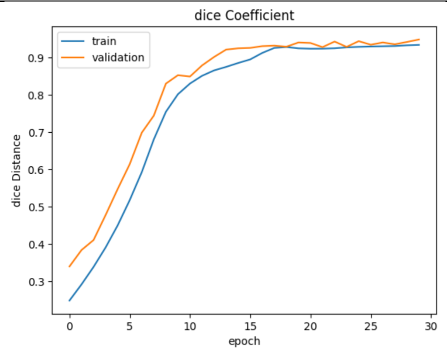






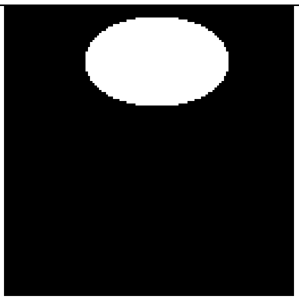
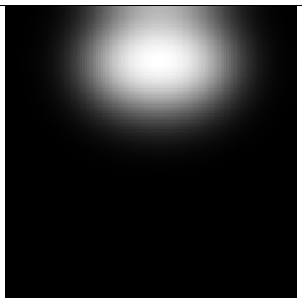
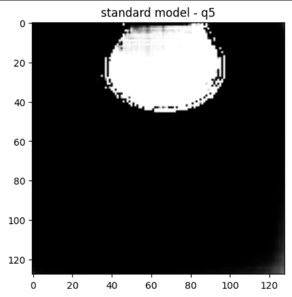
The dice coefficient is very similar to the IOU index they are positively correlated meaning if one says the model is performing better then the other will agree. Like the IOU they both went from zero to 1 where one indicating the greatest similarity between predicted and ground truth. Therefore a similar preset can be define here, however I did take off early stopping here and decreases the epoch to 15 just to see if this makes a difference. I will repeat the same setup for the IOU as above too.

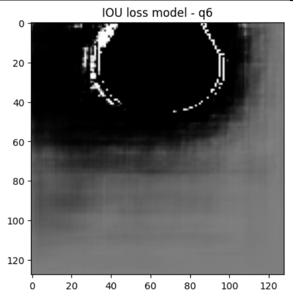
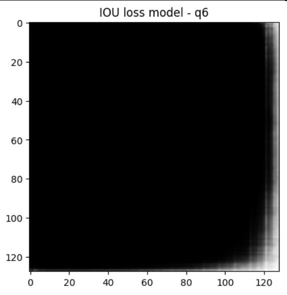
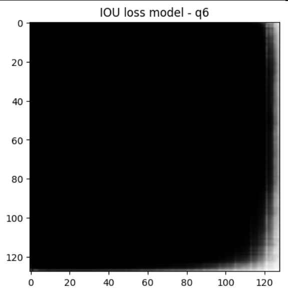
Yuqi Yang, 20150516

Dice Coe: Epoch = 15 LR = 0.00001, metrics = [Dice coe, "accuracy"]		
		
Epoch = 30 LR = 0.00001, metrics = [Dice coe, "accuracy"], EarlyStopping(monitor='val_loss', patience=3)		
		

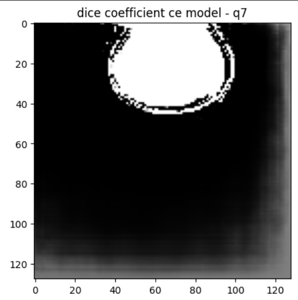
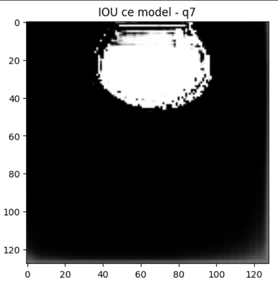
I think perhaps due to their positive correlated relation, a good hyperparameter set up works well on IOU performs great on dice coefficient too.

Q8:

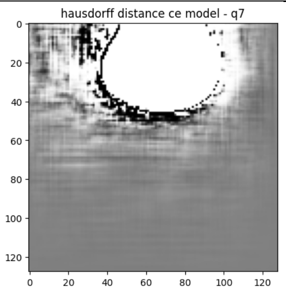
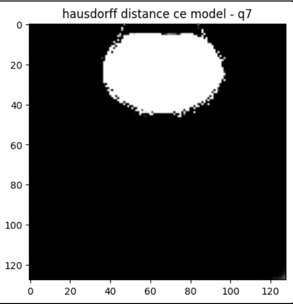
Ground Truth	Blurred Image	CE loss function – q5 metrics = [IOU, "accuracy", precision, recall] lr = 0.0001 epoch=20, earlystopping
		
IOU loss function – q5 metrics = [IOU, "accuracy"] lr = 0.00001 epoch=15, earlystopping	IOU loss function – q5 metrics = [IOU, "accuracy"] lr = 0.0001 epoch=30, earlystopping	IOU loss function – q5 metrics = [IOU, "accuracy"] lr = 0.00001 epoch=30, no earlystopping

dice coefficient ce model - q7
metrics = [dice coeff , "accuracy"]
lr = 0.00001 epoch=30, earlystopping
	IOU ce model - q7
	metrics = [IOU, "accuracy"]
	lr = 0.00001 epoch=30, earlystopping

hausdorff distance ce model - q7
metrics = [hausdorff , "accuracy"]
lr = 0.00001 epoch=15
	hausdorff distance ce model - q7
	metrics = [hausdorff , "accuracy"]
	lr = 0.0001 epoch=30, earlystopping

What do you notice about how the different metrics and loss functions affected the predicted segmentations?

From the above table we can see, the IOU loss function performs the worst when its model accuracy, loss index and IOU value appear to be reasonable and efficient (Epoch = 50 LR = 0.00001) during training and validation process, but the qualitative evaluation performs very poorly. After several attempts of recompile the model itself, I still couldn't figure out the reason, my best estimate is that the IOU loss function is imperfect, since when categorical cross entropy is taken as the loss function, where IOU is monitored as well; the segmentation is clear/acceptable. So far, the tendency I can conclude here is that, given a stricter learning rate ideally smaller than 0.00001, correspondingly a epoch size of around 15 can aid in the process of prediction (epoch size greater than 50 will decrease the accuracy of segmentation prediction). Therefore, to answer the question, I can for sure conclude that the loss functions well make a huge difference in terms of predicting segmentation. My several attempts of IOU_loss testing can be found in the above table.

I think in terms of predicting segmentations, IOU and dice coefficient generally agree with each other, I did not provide all 10 pictures, but the correlation between dice coefficient and IOU is noticeable. In the example I provided above, I notice a ring around the edge of the object happens to be on both metrics involved predictions. However, as seen in the dice coefficient picture above, there appears to be an outline of grayish shade on the edge of the background, but this ring of shade cannot be seen in the IOU model background.

The similar situation of IOU loss model happens to hausdorff distance model as well, given a perfectly tuned hyperparameter set up that performance well under training and validation, but did poorly at the testing part. Since the general segmentation shape can be seen for the "lr = 0.00001 epoch=15" prediction, only the background looks vague. I think it's a sign of not given enough rounds of training, so I raised the epoch size to 30 and also increase the learning rate to 0.0001, then solved the question. it's noticeable that this learning rate of 0.0001 with epoch size of 30 did perform better during the training and validation process can be seen in the table above (q7).

In conclusion, models that use a loss function of categorical crossentropy tend to maintain the high performance from training and validation to testing, however I realized that the IOU_loss function may be imperfect, since I did do the research of how to implement IOU loss function. instead of "1-IOU(y_true,y_pred)", some people use "IOU(y_true,y_pred)", I don't think this is a huge factor that would affect my testing implementation. Metrics plays a certain role in the process of prediction, but not a huge effect.

Q9:

Model	Metric	Avg Precision	Avg Recall	Loss	Accuracy	Featured metric
Loss=ce	Accuracy, loss	0.9904	0.9472	0.4759	0.9807	N/A
Loss=iou_loss	Accuracy, IOU, IOU_loss	0	0	-15099493	0.8667	IOU=1.00
Loss=ce	Accuracy, loss, IOU	0.9356	0.9367	0.4493	0.9780	N/A
Loss=ce	Accuracy, loss, hausdorff distance	0.9805	0.9566	0.4532	0.9836	N/A
Loss=ce	Accuracy, loss, dice coefficient	0.9698	0.8093	1.3298	0.9572	N/A

Note any observations that you make about the results. Were some metrics higher than others? What do the metrics indicate about the similarity between the prediction and the ground truth labels?

As expected, since the image rendering for IOU loss function model is pitch black or poorly performed, I paid a terrible value for average precision an average recall, and the negative 15099493 loss. The 3 values indicates the amount of TP is close to 0 (likely is 0), and the FN and FP value is close to maximum (likely is the maximum). However, to my surprise, the accuracy obtained from the evaluate() function appeared to be normal (0.8667), I anticipate that this value is slightly affected by the training and validation accuracy. And the IOU index appears to be overfitting, however, I do not understand why this scenario will occur because I added early stopping to prevent overfitting already.

Yuqi Yang, 20150516

To my human eyes, I think there is an negative correlation between loss and accuracy, the dice coefficient model contains the most impurity in its object area, I also has the lowest accuracy (other than IOU loss model), and the greatest loss value. Therefore the smaller the loss value, and the greater the accuracy, well indicate a greater similarity between the prediction and the ground truth. And since accuracy is calculated from precision and recall, they share a positive correlation between themselves and accuracy, the evidence can be traced on dice coefficient again, because it has a lower recall value its accuracy is the second lowest, and he also has the most impurity in its prediction.

Due to the nature of the metrics, the closer the loss is to 0, the better the segmentation, and the other 3 metrics has a positive correlation with the quality of the model performance, therefore, the loss will always be smaller than accuracy.

Library use documentation (general library like NumPy is not recorded):

Q2: cv2.GaussianBlur, math.random

Q3: na

Q4: cv2.filter2D, np.stack

Q5: na

Q6: na

Q7: K for keras

Q8: na

Q9: K for keras, np.vstack

P.S.:

1. implement each model with different setup by commenting out different hyperparameter