Part 1: Network Construction

In general, I used Adam optimizer, BCEWithLogitsLoss() for loss function, because it is a binary segmentation, train model for 10 epochs, and used a learning rate schedular for each 6 epochs with gamma value of 0.1.

For the training part of the UNet, I select Adam as optimizer with learning rate 0.001. I choose Adam because Adam is one of the best optimizer options that can handle the problems the other optimizers have like saddle points, get stuck in local minimas, and high curvature regions. I choose 0.001 as learning rate, first I tried with 0.01 but the results are not good enough, model seemed learning too fast, so I decided to use 0.001 as learning rate.

For the batch size selection, I used 4 as batch size. A smaller batch size will reduce the computational complexity of the model, so I tried 4 as batch size to see the results, I managed to get greater than 0.75 F-score, so I thought batch size as 4 is good enough.

I did not used a stopping condition to save the model, because after some point validation accuracy did not change much but training accuracy kept increasing so I took the model after the last epoch.

First, I calculated the total accuracies of both training, validation, and test. But this is a segmentation task, so the total accuracy means nothing. For example, our model can claim 0 for all pixels, and can get 90% accuracy, because already 90% of the original image is background, but we couldn't predict any of the foreground pixels. So, I introduced precision, recall and F-score to get more meaningful performance scores out of the model. To get meaningful precision, recall and F-score, I calculated precision, recall and F-score for each image and take the average of them as the result. In below, you can see a table for precision, recall, F-score, and loss value of training after each epoch and for validation.

Epoch number	Loss Value	Precision	Recall	F-Score
1	0.51	0.39	0.59	0.41
2	0.38	0.67	0.75	0.66
3	0.31	0.73	0.80	0.73
4	0.26	0.75	0.82	0.75
5	0.22	0.76	0.82	0.76
6	0.18	0.77	0.84	0.78
7	0.16	0.78	0.85	0.79
8	0.15	0.79	0.85	0.80
9	0.15	0.78	0.87	0.80
10	0.15	0.78	0.86	0.80

Table 1. Performance metric results for training after each epoch.

Epoch number	Loss Value	Precision	Recall	F-Score
1	0.51	0.14	0.58	0.21
2	0.33	0.004	0.02	0.006
3	0.31	0.01	0.001	0.002
4	0.27	0.13	0.009	0.01
5	0.21	0.65	0.25	0.33
6	0.20	0.004	0.008	0.004
7	0.16	0.67	0.75	0.68

8	0.16	0.64	0.80	0.69
9	0.17	0.53	0.93	0.65
10	0.16	0.57	0.90	0.67

Table 2. Performance metric results for validation after each epoch.

And in the end, the model got 0.69 test precision, 0.92 test recall, and 0.77 test F-score.

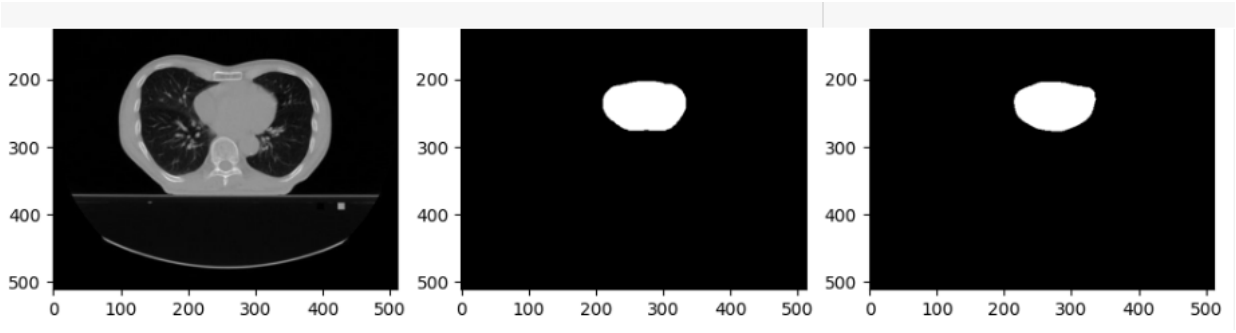

Figure 1. Best prediction (0.96 F-score)

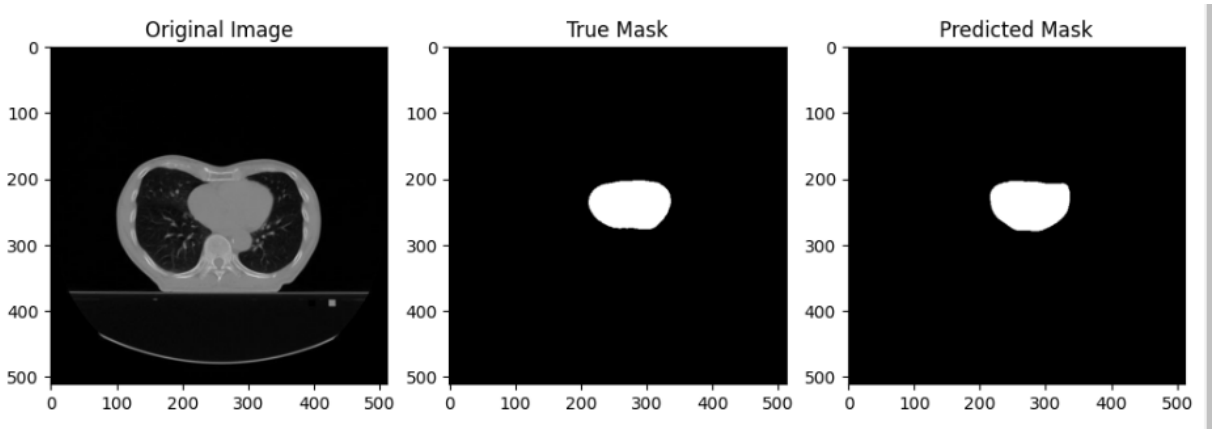

Figure 2. Second best prediction (0.95 F-score)

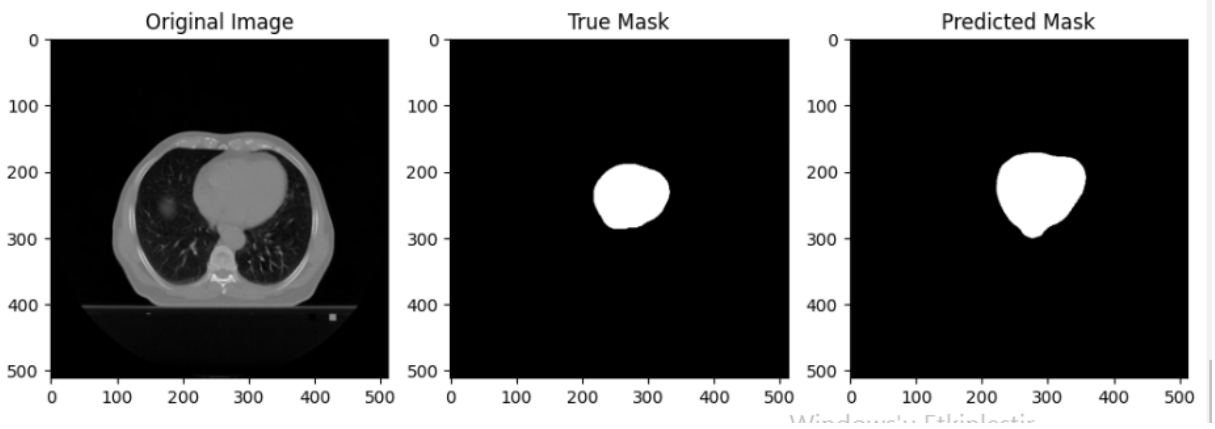

Figure 3. Acceptable prediction (0.77 F-score)

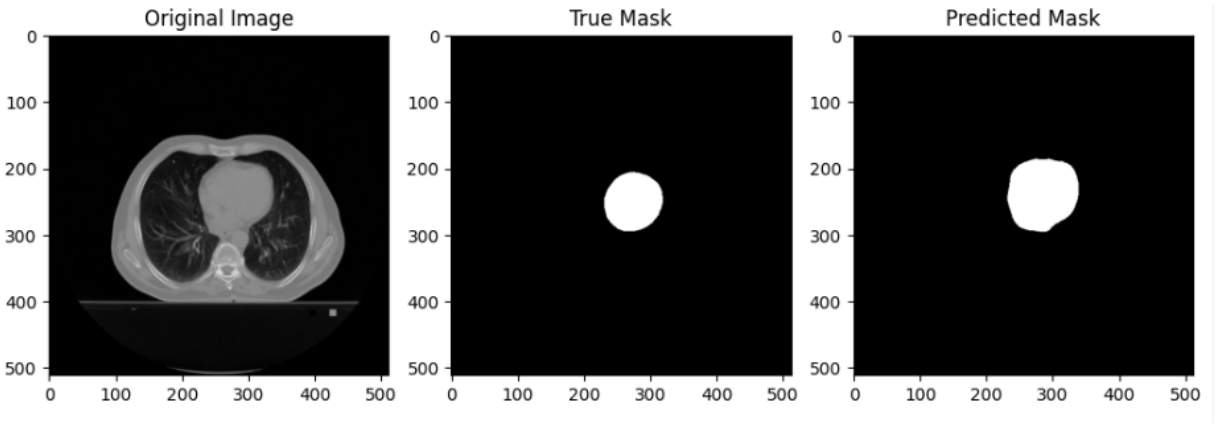

Figure 4. Acceptable prediction (0.77 F-score)

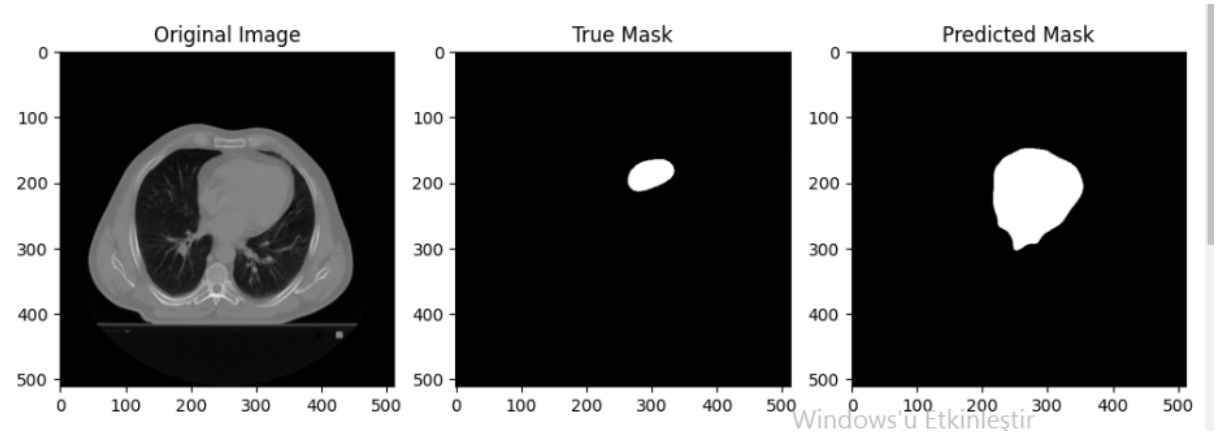

Figure 5. Bad prediction (0.29 F-score)

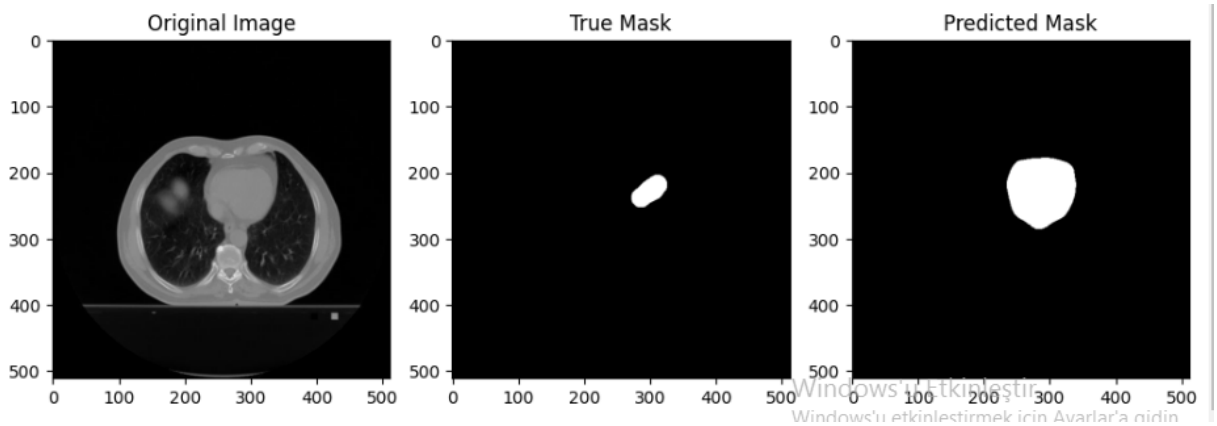

Figure 6. Bad prediction (0.32 F-score)

To select the above 6 examples, I used following logic. To get 2 good examples, I chose 2 predictions which got the best F-score. To get 2 bad examples, I chose 2 predictions which got the worst F-score. To get acceptable 2 examples, I choose 2 predictions which got slightly better than average performance of the model, which is 0.77 F-score. When we checked the bad examples, we can say model has problems segmenting small objects. Prediction masks seems much larger than the ground truth masks. Model got acceptable performance with more

circular shaped objects, which are not too small, and get good results for more cylindrical images which not too small.

Part 2: Network Architecture Modification

	Training Set			Validation Set			Test Set		
	Precison	Recall	F-score	Precison	Recall	F-score	Precison	Recall	F-score
Default network design	**0.78**	**0.86**	**0.80**	**0.57**	**0.90**	**0.67**	**0.69**	**0.92**	**0.77**
Modification in the number of down and up sampling steps	**0.74**	**0.84**	**0.77**	**0.51**	**0.45**	**0.43**	**0.81**	**0.31**	**0.40**
Modification in the number of feature channels	**0.79**	**0.85**	**0.80**	**0.60**	**0.86**	**0.68**	**0.72**	**0.89**	**0.78**

When we compare the results of methods, we can say that default network design and the design that have modification in the number of feature channels (multiplied with 2 all features) got almost same F-score, but with less downsampling model got significantly bad F-score compared to others. This significant difference shows us downsampling is an important part of the UNet, which is quite reasonable because downsampling helps the model to learn the semantics of the images.

When we compare default model with 3 downsample model, even though they got similar training F-scores, there is a huge gap between test F-scores, which indicates a huge difference between segmentation performance. Removing 1 downsample layer reduced the training time, but also decreased the model performance significantly. When we compare default model with doubled feature size model, there is no significant difference in training results, validation results, and test results. But I should say the training time of doubled feature size is more than default.

Between these 3 models, I prefer default model, because default model did better than 3 downsampling model in result, and faster to train than doubled feature sized model.

Part 3: Dropout in Network Architecture

	Training Set			Validation Set			Test Set		
	Precison	Recall	F-score	Precison	Recall	F-score	Precison	Recall	F-score
Default network design	**0.78**	**0.86**	**0.80**	**0.57**	**0.90**	**0.67**	**0.69**	**0.92**	**0.77**
Network w/dropout (p= 0.5)	**0.77**	**0.75**	**0.72**	**0.52**	**0.56**	**0.49**	**0.80**	**0.50**	**0.58**
Network w/dropout (p=0.3)	**0.78**	**0.86**	**0.80**	**0.39**	**0.96**	**0.53**	**0.54**	**0.95**	**0.67**
Network w/dropout (p=0.1)	**0.80**	**0.87**	**0.81**	**0.59**	**0.87**	**0.68**	**0.72**	**0.90**	**0.78**

After integrating the dropout layers to the model, I observe the regularization effect in the results. I tried dropout with p=0.5, p=0.3, and p=0.1. For training part, there is no significant effect of dropout on precision, recall, and F-score. But in validation, for dropout with p=0.5, I see a huge decrease in recall, for p=0.3, I see a huge decrease in precision, and for p=0.1, there is no significant difference. In test, for p=0.5, I see an increase in precision, but a huge decrease in recall, which caused a huge performance loss comparing to the default model. For p=0.3, I observe a huge decrease in precision, and a slight improvement in recall, which in total caused a decrease in performance of the model. For p=0.1, in test, again there is no significant difference, but p=0.1 got a slightly better F-score in the end.

P-value of 0.1 performed best segmentation over other two p-values. This results shows us we don't need to too harsh on the models predictions, we don't need to dropout too many nodes, but still with a small dropout like 0.1, we can still improve models' overall performance on the test set. This shows us we do not need to regularize more, which indicates that overfitting is not a problem for our model.

According to my results, increasing the p-value in dropout layers not decreased the difference between training and test set performance results, conversely increased the gap.

Berk Coşar 69557