COMP 448/548 – MEDICAL IMAGE ANALYSIS

HOMEWORK #3

Part 1: Network Construction

How did you select an optimizer in the training?

I experimented with three different optimizers: SGD, Adam, and RMSprop. For each optimizer, I tested various parameters. For Adam optimizer, I experimented with three different learning rates, three weight decay values, and three different beta values. For SGD optimizer, I experimented with three learning rates, three momentum values, and three weight decay values. For RMSprop optimizer, I experimented with three learning rates, three alpha values, and three decay values. The Adam optimizer with a learning rate of 1e-4, betas set to 0.9 and 0.999, and an epsilon value of 1e-8 performed the best among the three optimizers over 10 epochs.

What was the batch size in the training? How did you select this value?

Since we have a limited dataset, I experimented with batch sizes of 1, 2, 4, and 8. I ran the model for 10 epochs for each batch size. The table below shows the average training, validation, and test metric values, along with the time spent on training. Average precision in the table is basically pixel-level precision. When we examined the test average F scores, batch sizes of 2 and 4 performed the best compared to batch sizes of 1 and 8, with 8 being the worst. I also compared the scores considering the time spent on training, as it is an important factor for large datasets. Additionally, I calculated the standard deviation for each average metric to observe the variability during training. Even though a batch size of 4 requires less training time compared to a batch size of 2, the standard deviation for all metrics is lower with a batch size of 2. Since lower variability indicates more stable performance during training, I chose a batch size of 2 despite the slightly longer training time.

	Batch Size = 1	**Batch Size = 2**	**Batch Size = 4**	**Batch Size = 8**
Train Avg. Loss	0.2362	0.3931	0.4401	0.3376
Train Avg. Precision	0.9013	0.8359	0.6759	0.2235
Train Avg. Recall	0.8336	0.7938	0.6427	0.0803
Train Avg. F Score	0.8567	0.8045	0.6518	0.1042
Valid. Avg. Loss	0.2481	0.3847	0.4190	0.3419
Valid. Avg. Precision	0.6422	0.6493	0.5233	0.1735
Valid. Avg. Recall	0.5677	0.6821	0.5318	0.1182
Valid. Avg. F Score	0.5380	0.6439	0.5078	0.1257
Best Valid. F Score	0.7475	0.8152	0.7999	0.7174
Test Avg. F Score	0.8148	0.8858	0.8587	0.7979
Time Spent (sec)	255	204	199	130

Which stopping condition was used in the training, and which model was saved to predict the outputs for the images in the test?

I experimented with early stopping condition values of 3, 5, and 7. A value of 3 was too low as it caused the model to stop early, while a value of 7 was too high as it pushed the model to train for too long without improvement. A value of 5 proved to be optimal.

The model saved from epoch 6 with validation F score of 0.7962.

What are the training and validation performance metrics (loss value, precision, recall, F-score) throughout the training epochs?

In the original UNet architecture, batch normalization was not implemented. However, I used this method in my UNet architecture to achieve more stable training.

Epoch: 20

Train Average Loss: 0.3350

Train Average Pixel-level Precision: 0.7864

Train Average Recall: 0.7138

Train Average F Score: 0.7396

Validation Average Loss: 0.3335

Validation Average Pixel-level Precision: 0.7064

Validation Average Recall: 0.6087

Validation Average F Score: 0.6207

Test Average Precision: 0.8976

Test Average Recall: 0.9082

Test Average F1 Score: 0.8954

F score > 0.85: Good

0.75 <= F score <= 0.85: Acceptable

F score < 0.75: Problematic

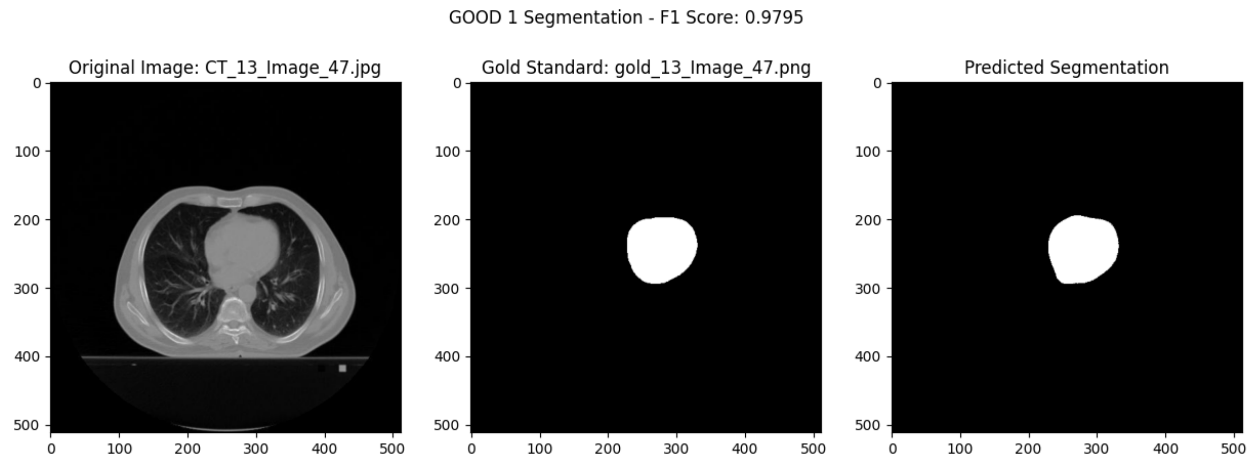

GOOD 1 Segmentation - F1 Score: 0.9795
Original Image: CT_13_Image_47.jpg
Gold Standard: gold_13_Image_47.png
Predicted Segmentation

Figure 1.1 Good Segmentation - 1

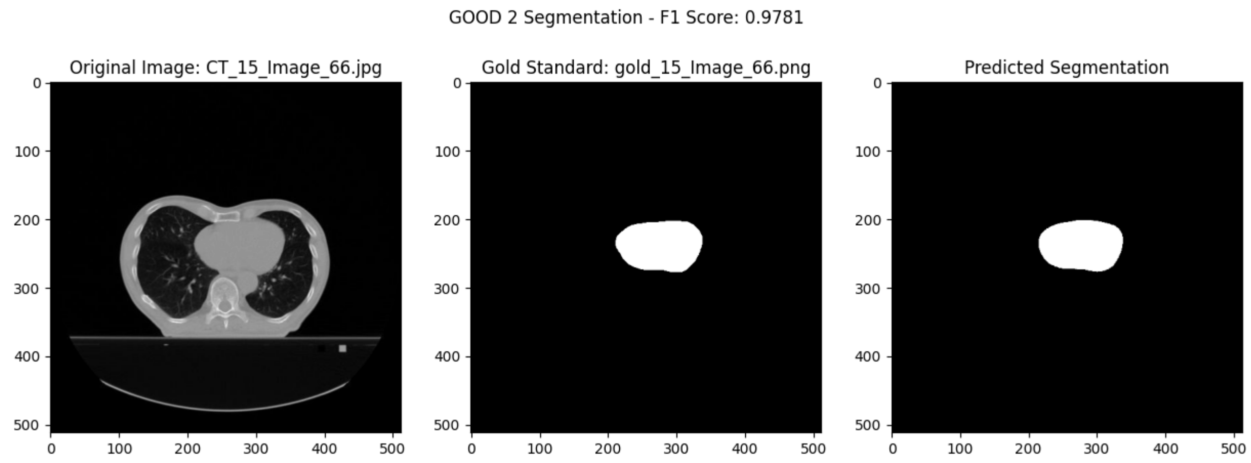

GOOD 2 Segmentation - F1 Score: 0.9781
Original Image: CT_15_Image_66.jpg
Gold Standard: gold_15_Image_66.png
Predicted Segmentation

Figure 1.2 Good Segmentation – 2

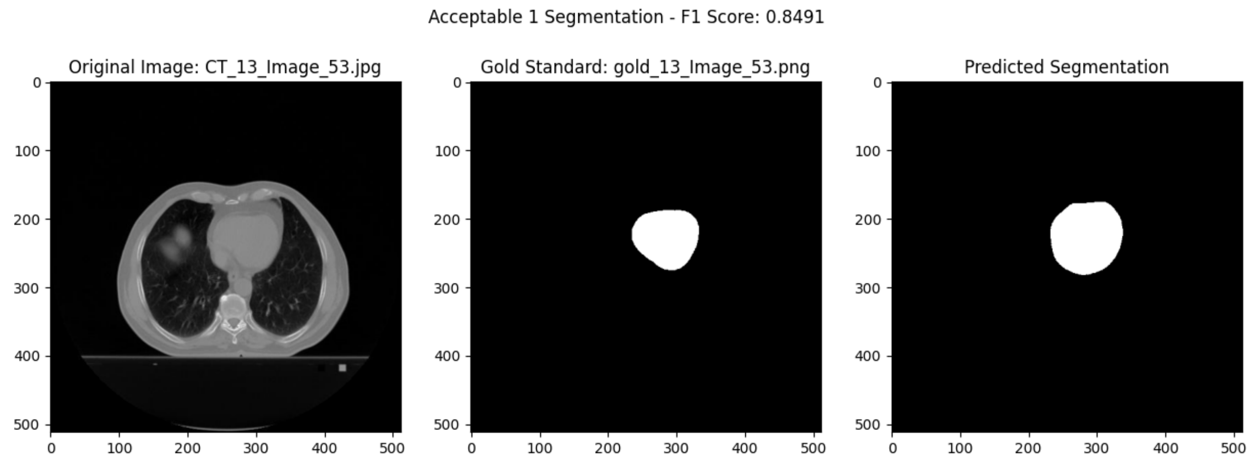
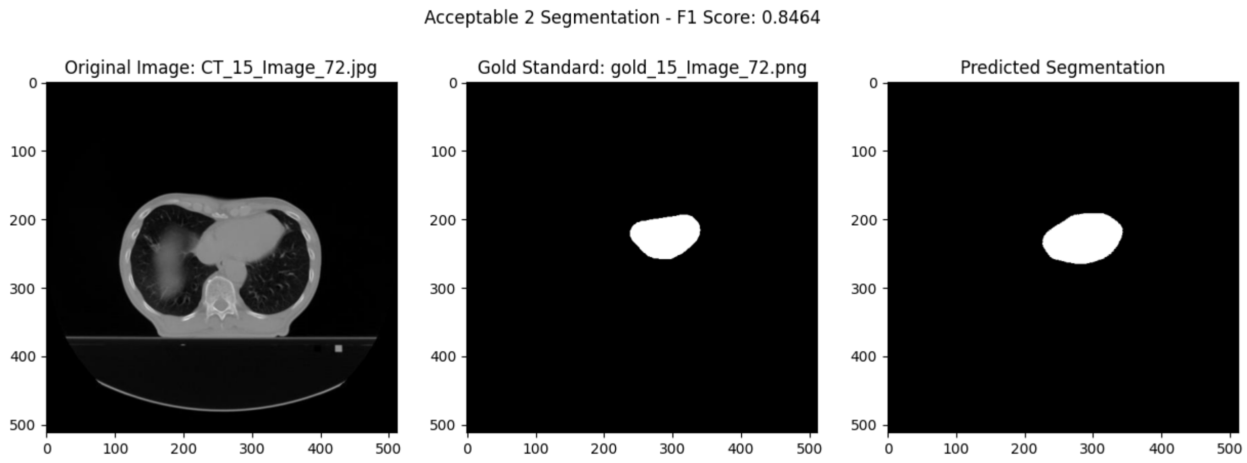
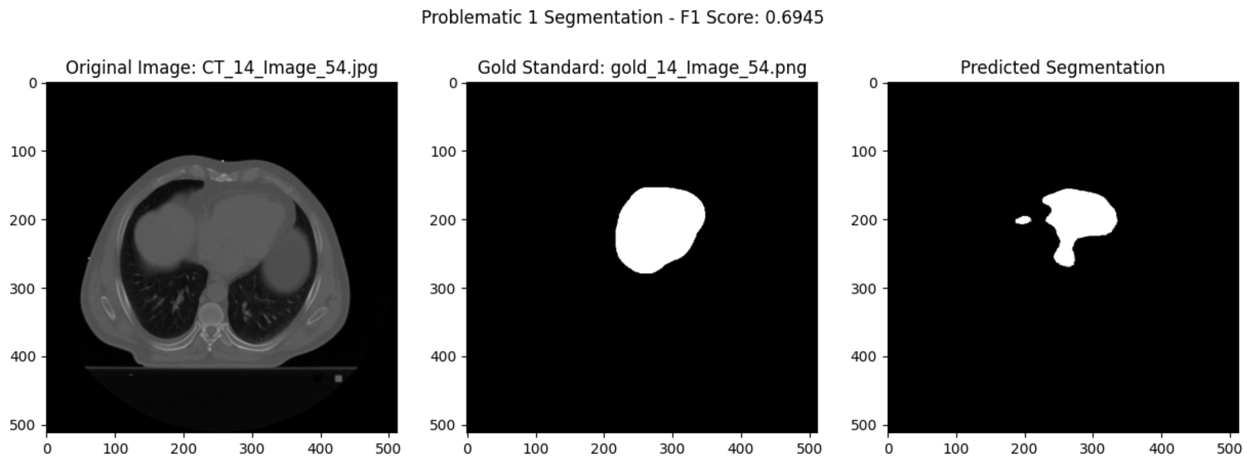

Acceptable 1 Segmentation - F1 Score: 0.8491
Original Image: CT_13_Image_53.jpg
Gold Standard: gold_13_Image_53.png
Predicted Segmentation

Figure 1.3 Acceptable Segmentation – 1



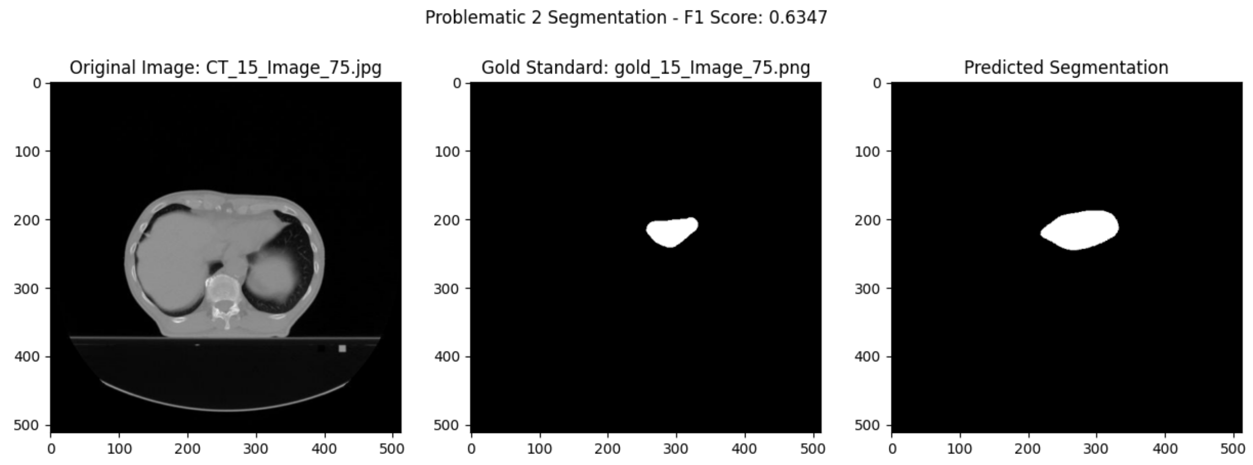
Acceptable 2 Segmentation - F1 Score: 0.8464
Original Image: CT_15_Image_72.jpg
Gold Standard: gold_15_Image_72.png
Predicted Segmentation
0
100
200
300
400
500

Figure 1.4 Acceptable Segmentation – 2



Problematic 1 Segmentation - F1 Score: 0.6945
Original Image: CT_14_Image_54.jpg
Gold Standard: gold_14_Image_54.png
Predicted Segmentation
0
100
200
300
400
500

Figure 1.5 Problematic Segmentation – 1



Problematic 2 Segmentation - F1 Score: 0.6347
Original Image: CT_15_Image_75.jpg
Gold Standard: gold_15_Image_75.png
Predicted Segmentation

Figure 1.6 Problematic Segmentation – 2

The model performs well when dealing with simple, continuous boundaries and consistent shapes. However, its performance decreases when faced with complex shapes. We can infer that while the model is sufficient for obtaining general features, it needs increased complexity and additional operations for feature extraction to effectively handle more complex boundaries.

Part 2: Network Architecture Modification

	Training Set			Validation Set			Test Set		
	Precision	**Recall**	**F Score**	**Precision**	**Recall**	**F Score**	**Precision**	**Recall**	**F Score**
Default Network	0.7864	0.7138	0.7396	0.7064	0.6087	0.6207	0.8976	0.9082	0.8954
Modification in the number of down and up-sampling step	0.8508	0.7649	0.7923	0.5482	0.7663	0.5919	0.8584	0.7649	0.8090

Modification in the number of feature channels	0.9289	0.8731	0.8948	0.7885	0.6843	0.7107	0.9242	0.9084	0.9099

I experimented with both using 8 and 32 channels in the model. When the number of down-sampling and up-sampling was decreased, it led to overfitting, as observed in the table. The training metrics values are higher than those of the default network. However, both validation and test F scores decreased, with a more significant drop in the test scores, indicating that the model cannot generalize well enough.

It was already observed that the default model with 16 channels was not optimal for capturing complex details in CT images. My initial expectation was that increasing the number of channels to 32 would make the model able to learn more complex features and achieve a better F score compared to the model with 16 channels. Increasing the number of channels also increases the number of parameters. This is the reason why the model was expected to perform better than the model with 16 channels. As expected, the model with 32 channels performed better than the model with 16 channels in terms of average test F score and the accuracy of segmented images. The only advantage of the model with 8 channels was its speed. It was not good at capturing complex details compared to the models with 16 and 32 channels and performed worse in terms of F score.

Which method performs the best segmentation performance over other methods? What are the possible reasons for this difference?

The model with 32 channels performed the best with a test average F score of 0.9099. We can infer that the model has good generalization capabilities compared to other models. The reason for this is that the model with 32 channels has more parameters which makes the model able to capture fine details.

The modification in the number of down/up-sampling steps led the model to perform worse across validation and testing. Fewer down-sampling steps result in a smaller receptive field. With a small receptive field, the model can capture less spatial context and may overlook important features necessary for effective segmentation.

What are the observed differences over the default network design in terms of segmentation performance, training time, and the gap between training and test set results of methods?

The default network provided a balanced performance with reasonable training, validation and test results with less training time compared to the network with 32 channels. Reducing the number of down-sampling and up-sampling steps led to overfitting. This issue was observed by the disparity between training and test performance. In contrast, increasing the number of feature channels to 32 improved all the metrics in evaluation at the cost of longer training times. While these modifications can lead to performance improvements, they also introduce trade-offs between training time, model capacity, and the ability to generalize across training, validation, and test.

Which network architecture do you prefer among these three methods and why?

The choice of model depends on the scenario. If we have limited GPU power and a tight schedule to obtain results, we can choose the model with three down and up-sampling steps since the model has fewer parameters and requires less training time.

If we need mediocre localization and segmentation compared to the model with 32 channels, we can choose the default model since it achieves the second-best F score in the test phase.

If capturing fine details in segmentation, such as the shape of the segmented part, is as important as localization, and there are no constraints on GPU power or schedule, we can choose the model with 32 channels since it provides the most detailed segmentation with good localization.

Part 3: Dropout in Network Architecture

	Training Set			**Validation Set**			**Test Set**		
	Precision	**Recall**	**F Score**	**Precision**	**Recall**	**F Score**	**Precision**	**Recall**	**F Score**
Default Network	0.7864	0.7138	0.7396	0.7064	0.6087	0.6207	0.8976	0.9082	0.8954
Network w/dropout (p = 0.1)	0.8228	0.7345	0.7626	0.7155	0.6235	0.6252	0.9099	0.9158	0.9047
Network w/dropout (p = 0.3)	0.9058	0.8321	0.8634	0.7407	0.6437	0.6890	0.9392	0.9037	0.9208
Network w/dropout (p = 0.5)	0.8936	0.7467	0.8080	0.4273	0.7979	0.5165	0.7724	0.8902	0.8174

I applied dropout in the down-sampling after the max-pooling operation but after the 3x3 convolution operation, and in the up-sampling after the concatenation of feature maps but after the 3x3 convolution operation to regularize the encoding and the decoding processes considering the recommendations from the paper “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. I also applied dropout before the 3x3 convolution operations. It caused the model to perform worse compared to the method mentioned above.

Do you observe the regularization effect in your results after integrating the dropout layers? What performance results demonstrate the regularization effect of dropout layers?

In the training set, the networks with dropout operation showed improved evaluation metric scores compared to the default network. In the validation set, the network with dropout = 0.3 showed the best performance with the improvement in all metrics. This indicates that the model provides the best generalization with a dropout value of 0.3.

Which p-value performs the best segmentation performance over the other two p-values? Comment on the possible reasons for this situation.

A dropout value of 0.3 performs well and provides good generalization, while retaining sufficient model complexity. It prevents overfitting without causing significant underfitting. This dropout value also helps maintain training stability by reducing the variance in performance metrics and leading to consistent improvement across different datasets. A dropout value of 0.1 showed some improvement in the evaluation metrics compared to the default network model, but it did not provide enough regularization as seen in the metrics. A dropout value of 0.5 caused underfitting by being too aggressive on the model.

Could we say that by increasing the p-value in the dropout layers, the difference between training and test set performance results decreases depending on your results?

Increasing the dropout rate can reduce the difference between training and test set performance. This trend indicates that overfitting was reduced during training. However, there is a trade-off at a dropout value of 0.5. While the gap between training and test performance decreases, the validation performance metrics significantly drop. This indicates that while dropout helps in regularizing the model and improving its generalization capacity, too high of a dropout value can drop the model’s learning capacity and this may decrease the performance on unseen data.