Final Report

(a) Report and Justification of Choices

In the previous modules, I made the following choices:

1. Model Architecture: SimpleUNet was chosen as the model architecture due to its simplicity and efficiency for image segmentation tasks.
2. Pretrained Weights: The model can load pretrained weights to benefit from transfer learning. If no weights are provided, it loads the weights from "./weights/SimpleUNet_v3.pt".
3. Data Loading: The pydicom library was used to read DICOM images, and the pathlib library was used to handle file paths. The DicomDataset class was used to load the image and mask data.
4. Data Split: The data was split into train and test sets using the train_test_split function from sklearn. A test size of 1/3 and a random state of 42 were used for reproducibility.
5. DataLoader: Data loaders were created for the train and test datasets using torch.utils.data.DataLoader. The batch size was set to 3 for training and 1 for testing.
6. Loss Function: A custom loss function combining binary cross entropy and the soft Dice loss was used. This loss function is commonly used for image segmentation tasks.
7. Optimization: The Adam optimizer with a learning rate of 1e-1 was used to optimize the model parameters.
8. Evaluation Metric: The BinaryAccuracy metric from the torchmetrics library was used to compute the accuracy of the model predictions.
9. Training Loop: The model was trained for 10 epochs. In each epoch, the training data was iterated over, and the loss was computed and used for backpropagation to update the model parameters.
0. Testing Loop: After each epoch, the model was evaluated on the test data. The loss and accuracy were recorded for each epoch.
1. Model Saving: The model weights were saved to "./weights/saved_weights.pth" after training.
2. Result Visualization: The loss and accuracy plots were saved as "loss.png" and "acc.png" respectively. The predicted masks for selected test images were also saved.

(b) Analysis Report and Results

In the previous modules, a SimpleUNet model was trained and evaluated for image segmentation. The key findings and observations are as follows:

1. Training Loss: The training loss decreased gradually over the epochs, indicating that the model was learning to fit the training data.

2. Testing Loss: The testing loss also decreased initially but started to plateau after a few epochs. This suggests that the model may have reached its optimal performance on the test data.
3. Accuracy: The accuracy of the model on the test data increased with each epoch, indicating that the model was improving its predictions.
4. Dice Similarity Coefficient (DSC): The DSC measures the overlap between the predicted masks and the ground truth masks. The DSC values were calculated for each test image and averaged. Higher values indicate better segmentation performance.
5. Result Visualization: The predicted masks were saved as PNG images for selected test images. The images showed the original image, the ground truth mask, and the predicted mask side by side. The accuracy and DSC values were also displayed on the images.
6. Analysis: The visual inspection of the predicted masks showed that the model was able to capture the general shape and location of the target objects. However, there were instances of false positives and false negatives, indicating room for improvement.
7. Learning Experience: Through this coursework, I gained hands-on experience in training and evaluating a deep learning model for image segmentation. I learned about different components of the pipeline, such as data loading, model architecture, loss function, optimization, and evaluation metrics. I also learned about the challenges and limitations of image segmentation tasks.

(c) Results Acceptability and Potential Improvements

The results obtained from the model were acceptable to some extent. The model was able to segment the target objects with reasonable accuracy, but there is room for improvement.

Potential causes for imperfect results could include:

1. Limited Training Data: The model may not have been exposed to a diverse range of images and variations in the target objects. Increasing the size and diversity of the training dataset could help improve the model's performance.
2. Model Complexity: The SimpleUNet model used in this coursework is a relatively simple architecture. Using a more complex model or exploring other state-of-the-art architectures could potentially improve the segmentation performance.
3. Hyperparameter Tuning: The choice of hyperparameters, such as learning rate, batch size, and number of epochs, can significantly impact the model's performance. Conducting a systematic hyperparameter search could help find optimal values.
4. Data Augmentation: Applying data augmentation techniques, such as random rotations, flips, and scaling, can help increase the variability of the training data and improve the model's generalization ability.
5. Post-processing Techniques: Applying post-processing techniques, such as morphological operations or conditional random fields, can help refine the predicted masks and reduce false positives or false negatives.

In the future, to improve the results, I would consider implementing the above suggestions and also explore other advanced techniques and architectures specific to the task of medical image segmentation.