Automated Lung CT Segmentation - DICOM to UNet Analysis: Medical Imaging Coursework Report

Vishal Jain

March 28, 2024

Contents

1 Introduction

This report presents the work done for the Medical Imaging coursework. The aim of the coursework was to explore part of the LCTSC Lung CT dataset and build a 2D image segmentation model using the UNet architecture. This report will provide an overview of the steps taken, the results obtained and present a critical discussion of the results.

2 Module 1: Handling DICOM data

The code and the data for the analysis described in this section can be found in the `src/explore_dicoms.ipynb` notebook and the `Dataset` directory, respectively.

2.1 Dataset Exploration

To investigate the dicoms, the `pydicom` library was used to read the DICOM files from the different cases. The first tag that was investigated was the `PatientIdentityRemoved` tag. This tag is used to indicate whether the patient identity has been removed from the DICOM file. The tag was checked for all the cases and it was found that the value was `YES` for all cases. However, investigating the meta data revealed a few cases still had sensitive information. This was removed by the script `src/de_identify_dataset.py` which is called during the set up instructions shown on the README.md file. The next set of tags investigated were the tags that are relevant to the stratification of the data. These tags were the `PatientSex`, `KVP`, `Manufacturer`, and `ModelName`. The summary of this information is presented in Table 1. Note, patient age would also have been included however it was not available for any of the cases in the dataset. The summary of this information is presented in Table 1.

Case ID	Sex	KVP	Model Name	Manufacturer
000	M	120	Sensation Open	SIEMENS
001	M	120	Sensation Open	SIEMENS
002	M	120	Sensation Open	SIEMENS
003	F	120	Sensation Open	SIEMENS
004	F	120	Sensation Open	SIEMENS
005	M	120	Sensation Open	SIEMENS
006	M	120	Sensation Open	SIEMENS
007	M	120	Sensation Open	SIEMENS
008	F	120	Sensation Open	SIEMENS
009	F	120	Biograph 40	SIEMENS
010	F	120	Sensation Open	SIEMENS
011	F	120	Sensation Open	SIEMENS

Table 1: Summary of patient data

When training a model, it's crucial to ensure that both train and test sets are similarly stratified to avoid introducing bias. In this dataset, all fields except sex are consistent across cases. Consequently, only patient sex has been used for stratification. Given the even distribution between male and female cases, no resampling techniques were required to enforce balance.

The subsequent analysis focused on voxel-related metadata. These fields were checked to ensure consistency across cases. The fields examined included voxel size, slice thickness, image orientation, rescale intercept, rescale slope, patient position, and pixel array shape. It was found that all fields were consistent across all cases except in the rescale intercept values, in which case 9 had a value of -1000 compared to the rest of the cases -1024. However, the rescale intercept is only relevant to convert the pixel values to Hounsfield units, which is not necessary

for the segmentation task. The only thing that needs to be ensured is that the distribution of intensities is consistent across cases. This was checked by plotting the intensity histograms for all cases, which are shown in Figure 1. It is important to note that the voxel intensities were winsorised to the 1st and 99th percentiles to remove outliers before plotting the histograms.

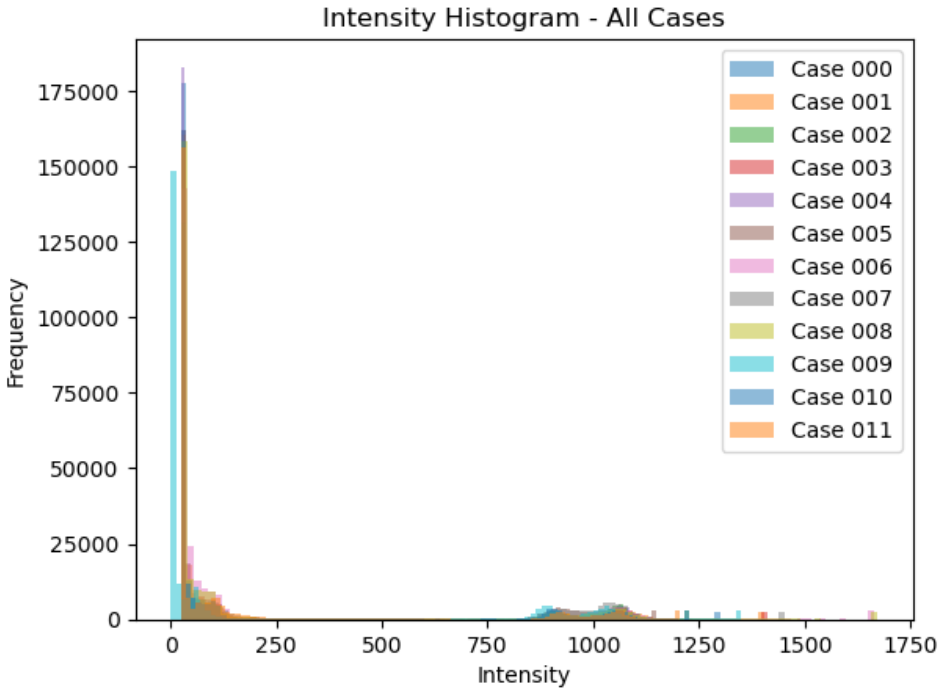


Figure 1: Intensity histograms for all cases

These histograms demonstrate that the intensity distributions are consistent across the different cases in dataset. However, case 9 is identifiable by a histogram peak slightly shifted to the left relative to the peaks of other cases, illustrating a minor deviation in its intensity distribution. To correct for this, a pre processing step was introduced to winsorise and normalise the intensity distributions across all cases before training the model.

2.2 Converting DICOMs to 3D Numpy Arrays

The next step was to convert the DICOM files to 3D numpy arrays. The relevant code for this can be found in the `src/utils.py` script under the functions `load_npz` and `dicom_dir_to_3D_arr`. To load the dicoms related to a single case into a 3D array the following steps were taken:

1. Read all the dicom files in the directory using the `pydicom.dcmread`.
2. Sort the files by their z coordinate using the 3rd element of the `ImagePositionPatient` attribute.
3. Extract the pixel array from each dicom file using the `pixel_array` attribute.
4. Use `numpy.stack` function to stack the 2d pixel arrays sorted by their z coordinate along a new axis to create a 3D numpy array.

To load the masks which were provided in the form of a `.npz` file, `numpy.load` function was used to load the file and extract the mask array. To verify proper alignment between the 3D image and mask arrays, they were reviewed in ITKSNAP, a standard medical image viewer. Due to

ITKSNAP's format limitations, the .npz mask files could not be loaded natively and were first converted to NIfTI. Creating a NIfTI file from a numpy array necessitates specifying an affine matrix—a 4x4 matrix mapping voxel to world coordinates. Rather than deriving this matrix from DICOM metadata, both the DICOM images (which are natively viewable in ITKSNAP) and masks were converted to NIfTI using an identity affine matrix, simplifying the process. The code for which can be found in the `src/utils.py` script under the function `make_niftis`. The image and lung mask volumes were then reviewed for each case in ITKSNAP and were confirmed found to be correctly aligned by the loading process defined previosly. An example of the image and mask for case 000 is shown in Figure 2. It is worth noting that because the affine matrix was not derived from DICOM metadata, the images and masks were not displayed in their correct physical dimensions in ITKSNAP. However, this does not affect the segmentation process as the model is trained on the 3D numpy arrays directly, all that matters is the shape and alignment of the image and mask arrays.

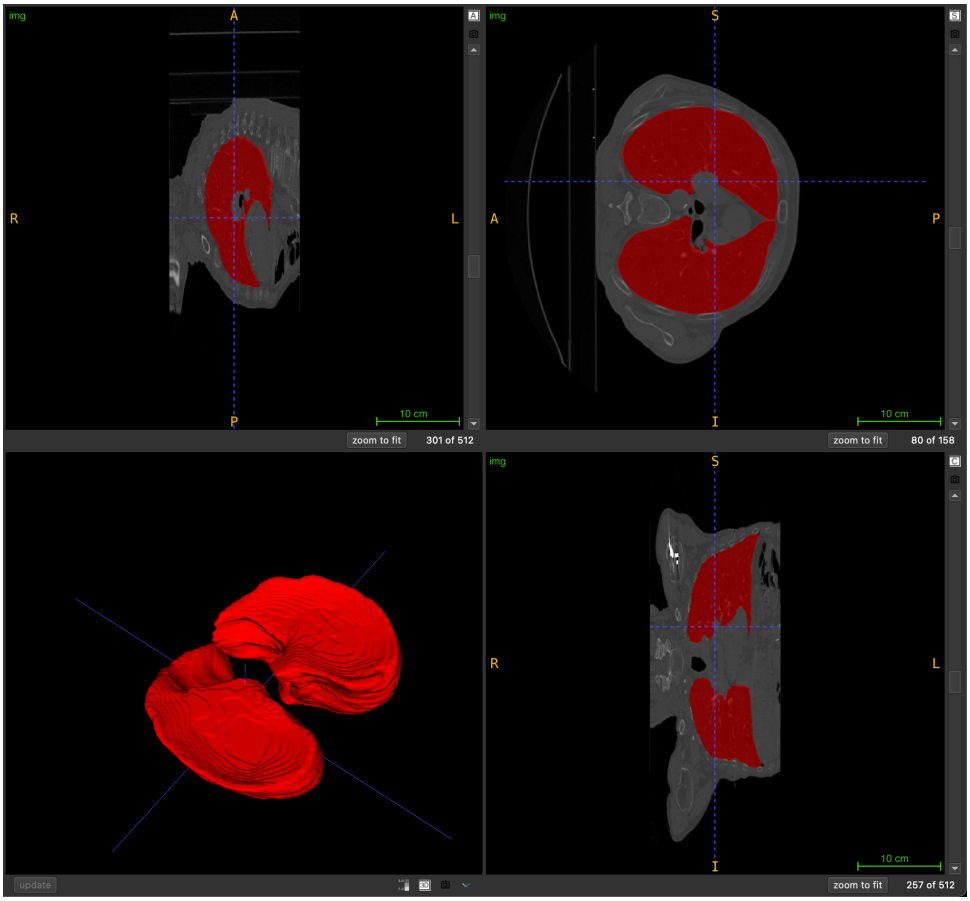


Figure 2: ITKSNAP view of the image and mask for case 001

3 Module 2: UNet-based segmentation

This section describes the training and evaluation of a 2D UNet model for lung segmentation. The code for this analysis can be found in the `src/train.py` and `src/test.py` scripts.

3.1 Model Architecture

The model architecture utilises a 2D UNet, comprising an encoder and a decoder. The encoder features a sequence of double convolutional layers, each separated by max pooling layers. A double convolutional layer consists of two consecutive blocks, with each block performing the

operations of convolution, batch normalisation, and ReLU activation in sequence. Following this, downsampling is achieved through max pooling, complemented by a dropout layer for regularisation. Conversely, the decoder is structured around a series of upconvolutional layers, each succeeded by a double convolutional layer. During the decoding process, feature maps from the encoder are concatenated with the upsampled feature maps. The model architecture is shown in Figure 3.

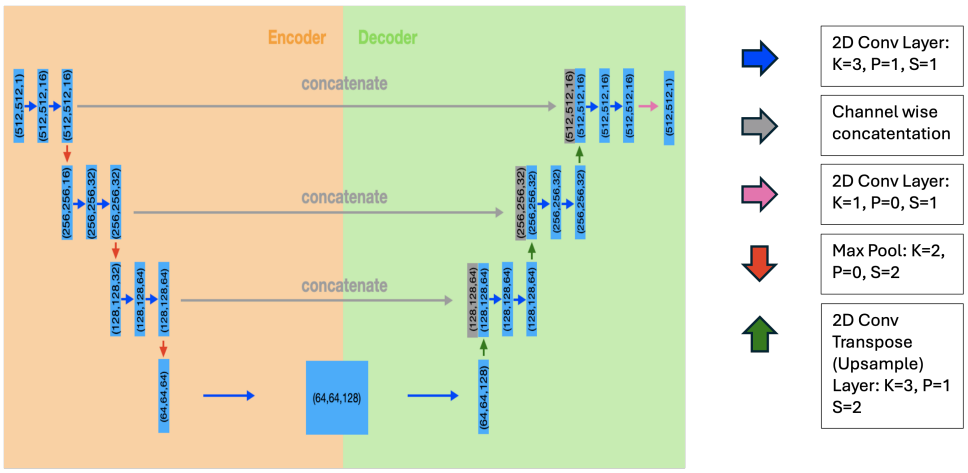


Figure 3: Diagram of the UNet architecture used, illustrating the dimensions of the input at each layer in the format HxWxC, where H is height, W is width, and C represents the number of channels. The accompanying legend (right) employs the following key: K for kernel size, S for stride, and P for padding.

3.2 Data Splits and Preprocessing

The dataset is divided into training and testing sets using a 2:1 ratio. To avoid data leakage, all slices from a specific patient are allocated to the same set, ensuring the model isn't trained and tested on slices from the same case. Additionally, cases are distributed between the splits to maintain a 50-50 male-female ratio, respecting the dataset's diversity and preventing the model from developing biases towards features associated with a single gender. Further, since the original data exploration revealed a slight discrepancy in the intensity distributions of case 9, a preprocessing step was introduced to winsorise and normalise the intensity distributions across all cases. This was achieved by winsorising the intensity values to the 1st and 99th percentiles and subsequently normalising the pixel values to the range [0, 1]. This process was applied to each slice independently.

3.3 Training

The model was trained for 10 epochs, using a batch size of 3 and the adam optimiser with a learning rate of 0.1. The loss function used was the binary cross entropy loss via the `BCEWithLogitsLoss` class from the PyTorch library in combination with a custom implementation of soft dice loss. The custom implementation can be found in the `src/losses.py` script. The soft dice loss is a metric that measures the overlap between the predicted and ground truth masks. It is particularly relevant for imbalanced datasets, as is the case here where the lung region is significantly smaller than the background. Both loss terms are equally weighted in the final loss function. The average loss and binary accuracy for each epoch are plotted in Figure 4. Note how the binary accuracy is already 95% after the first epoch and only shows a rough increase of 4% over the next 9 epochs. Compared with the loss which has a more consistent decrease over the 10 epochs, changing by about 30% in total. This shows how the soft dice loss

is a more sensitive metric than the binary accuracy, which is expected given the imbalanced nature of the dataset.

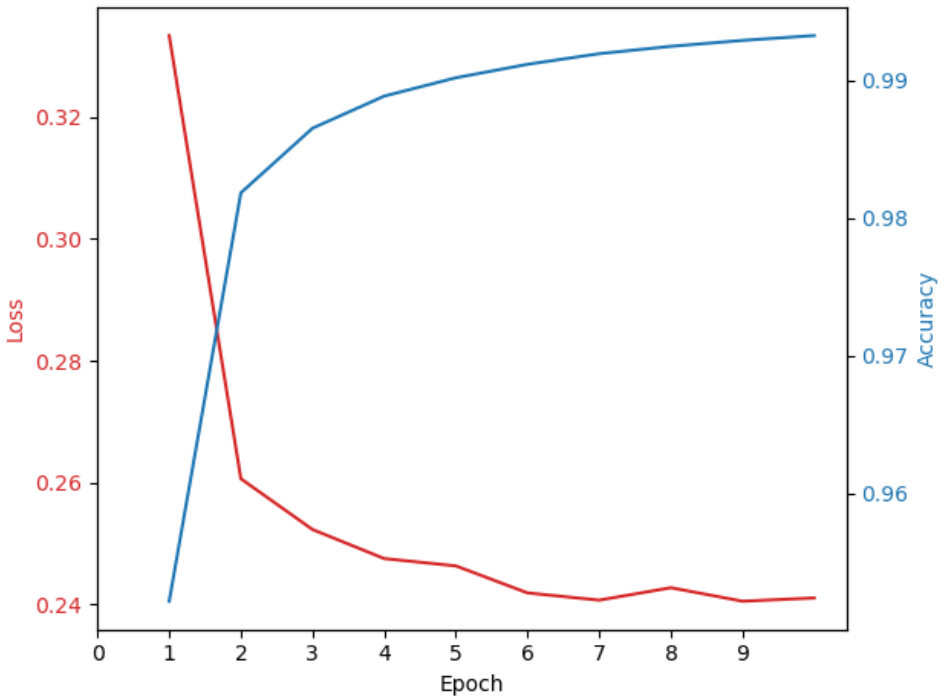


Figure 4: Plot of the average loss and binary accuracy for each epoch during training. The blue line represents the loss, while the red line denotes the binary accuracy.

3.4 Evaluation

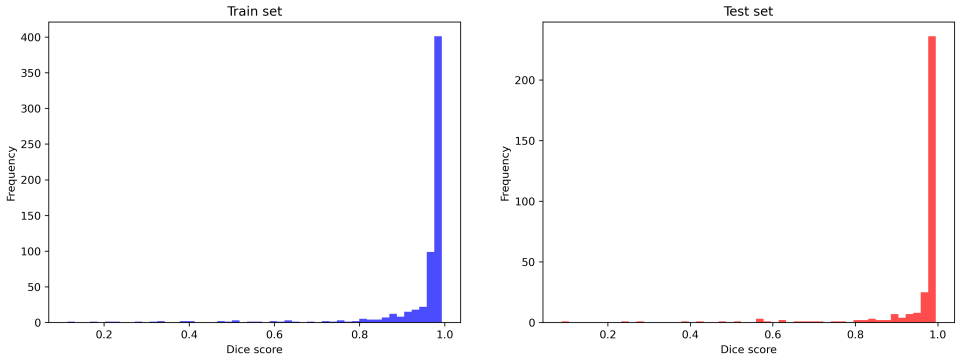
To evaluate the model the distribution of dice scores between the train and test set are visualised as histograms in 5 and boxplots in 6. The same can be done for the distribution of the binary accuracies. The distributions between the train and test are shown in histogram form in Figure 7 and boxplot form in Figure 8. The summary statistics are presented in table 3. The model performance can also be visually assessed by comparing the predicted masks with the ground truth masks for cases with the highest, lowest and typical dice scores (typical here means dice scores that fall in the IQR). This is done in figs 12, 10, 11 for the train set and in figs 12, 13, 14 for the test set.

Dice Score	**Train Set**	**Test Set**
Mean	0.943	0.949
Max	0.993	0.995
Min	0.114	0.0877
Upper Quartile	0.986	0.990
Lower Quartile	0.965	0.977
Interquartile Range (IQR)	0.0210	0.0129
Standard Deviation	0.118	0.117

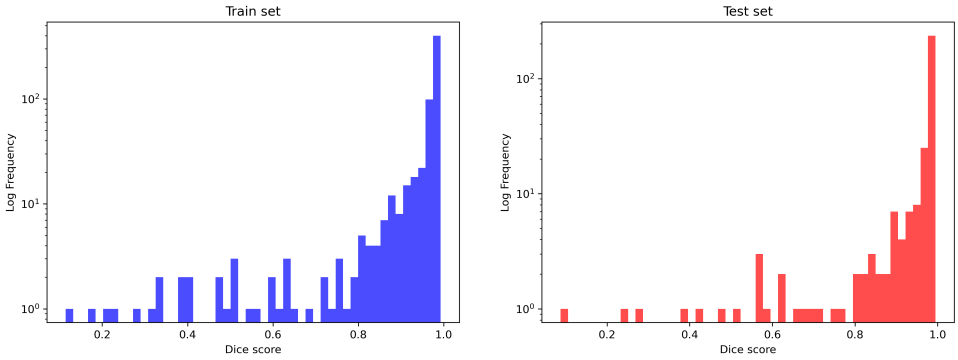
Table 2: Train and Test Split Dice Scores

Accuracy	**Train Set**	**Test Set**
Mean	0.996	0.997
Max	0.999	0.998
Min	0.992	0.997
Upper Quartile	0.996	0.997
Lower Quartile	0.996	0.997
IQR	0.00014	0.00023
Standard Deviation	0.00031	0.00015

Table 3: Train and Test Split Binary Accuracies

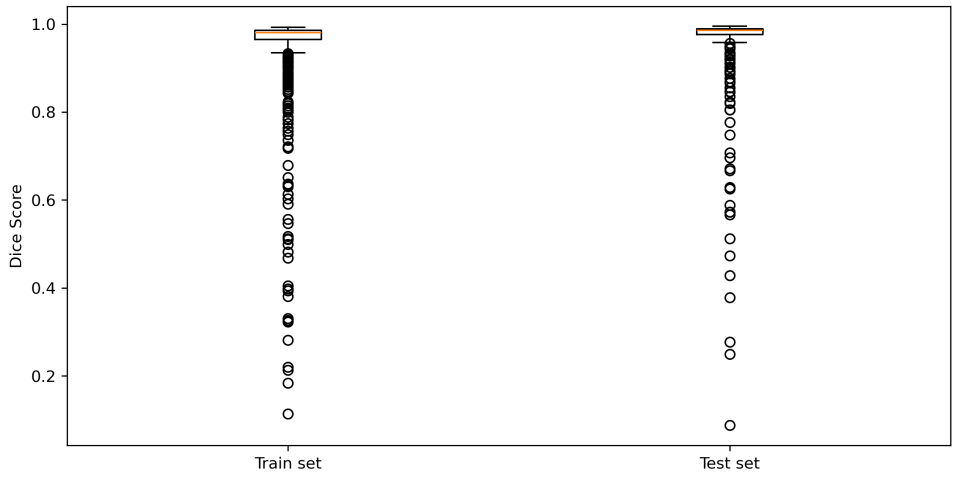


(a) Histogram of dice scores for the train and test sets.

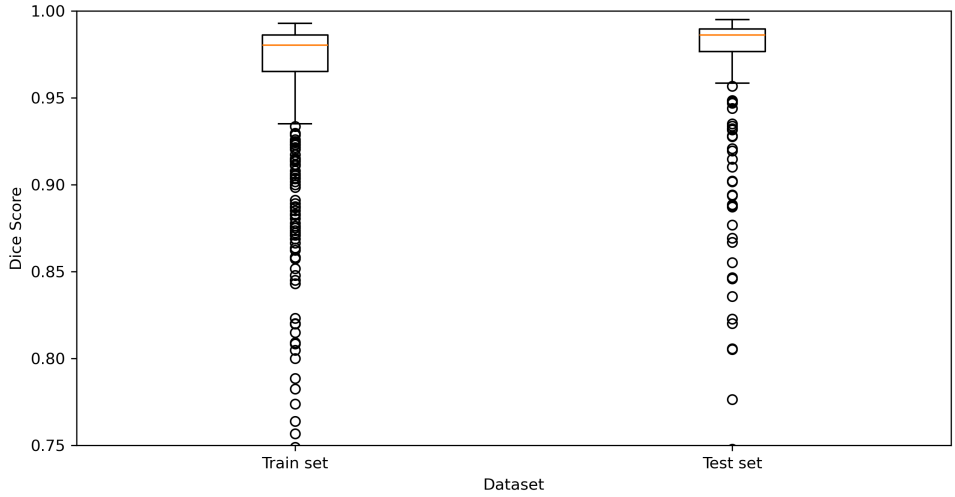


(b) Histogram of dice scores for the train and test sets using a log scale for the y-axis.

Figure 5: Comparison of dice score distributions for the train and test sets with different y-axis scales.



(a) Boxplot of dice scores for the train and test sets.



(b) Zoomed in.

Figure 6: Comparison of the dice score distributions between the train and test splits using box plots. Top: original box plot, bottom: zoomed in box plot.

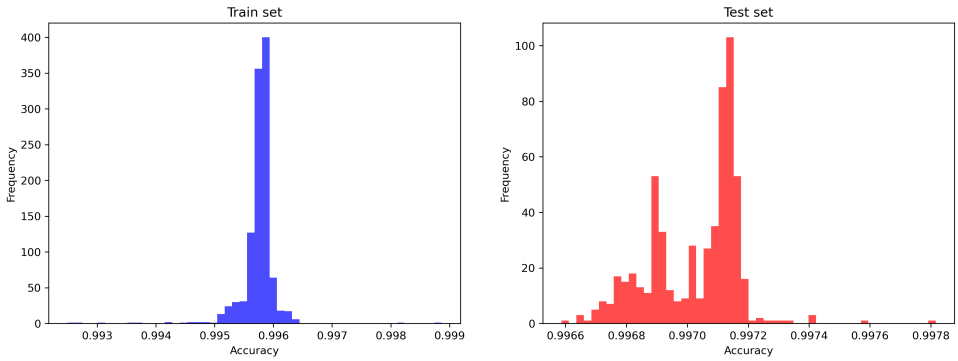


Figure 7: Histogram of binary accuracies for the train and test sets.

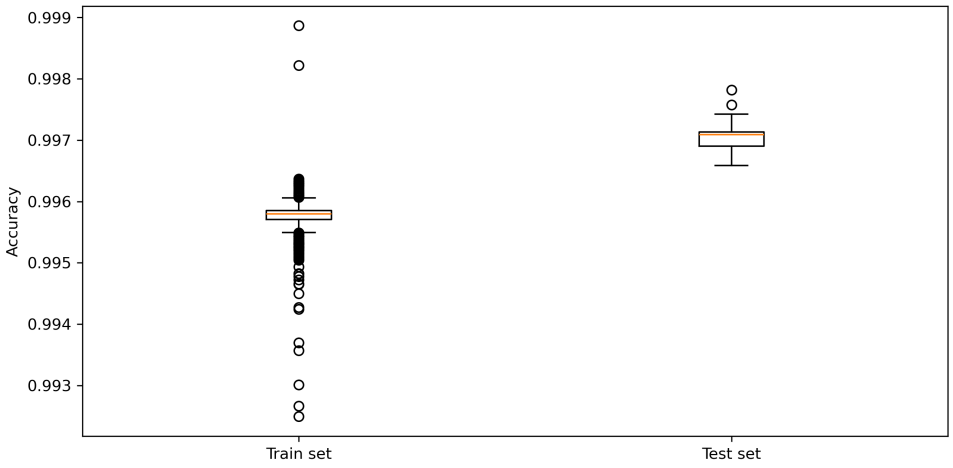


Figure 8: Boxplot of binary accuracies for the train and test sets.

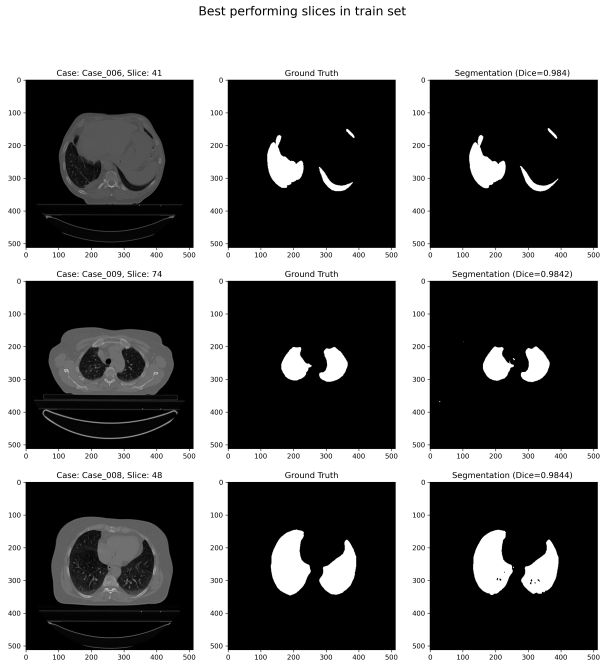


Figure 9: Comparison of the predicted masks with the ground truth masks for the train set. Cases from the train set with the top 3 dice scores as shown.

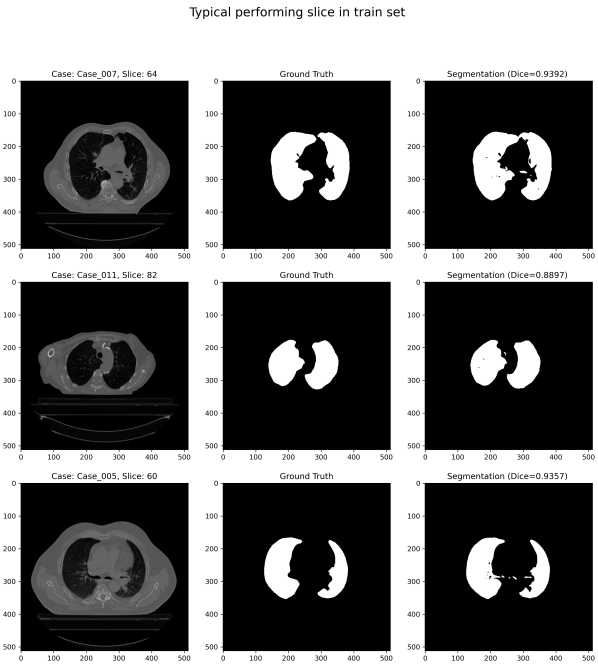


Figure 10: Comparison of the predicted masks with the ground truth masks for the train set. 3 random cases from the train set with dice scores in the IQR are shown.

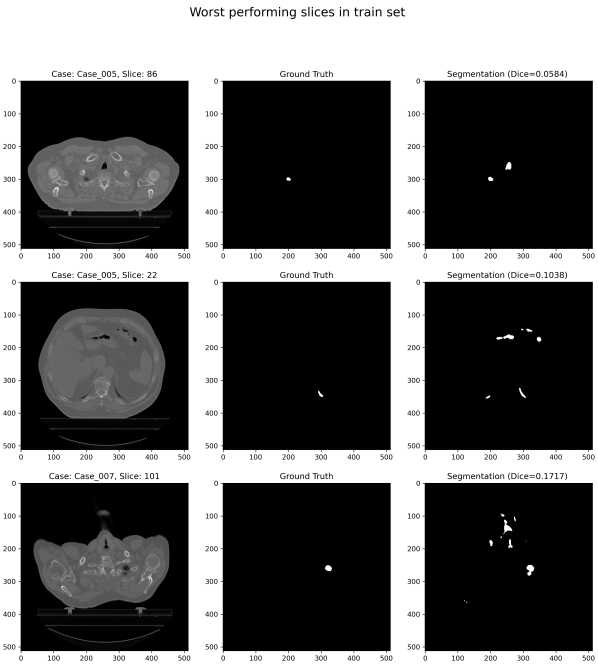


Figure 11: Comparison of the predicted masks with the ground truth masks for the train set. Cases from the test set with the worst 3 dice scores as shown.

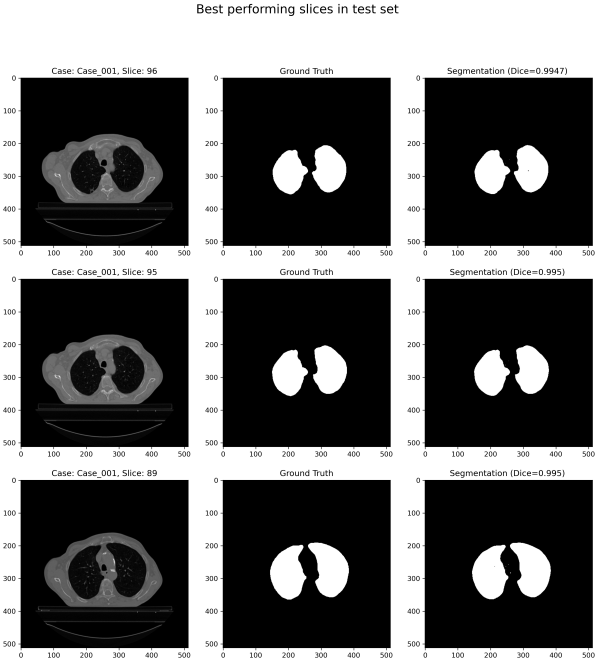


Figure 12: Comparison of the predicted masks with the ground truth masks for the test set. Cases from the test set with the best 3 dice scores as shown.

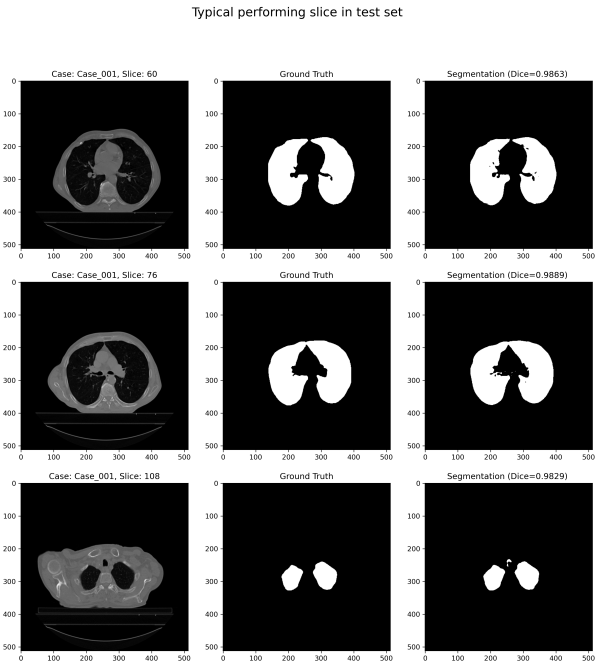


Figure 13: Comparison of the predicted masks with the ground truth masks for the train set. 3 random cases from the test set with dice scores in the IQR are shown.

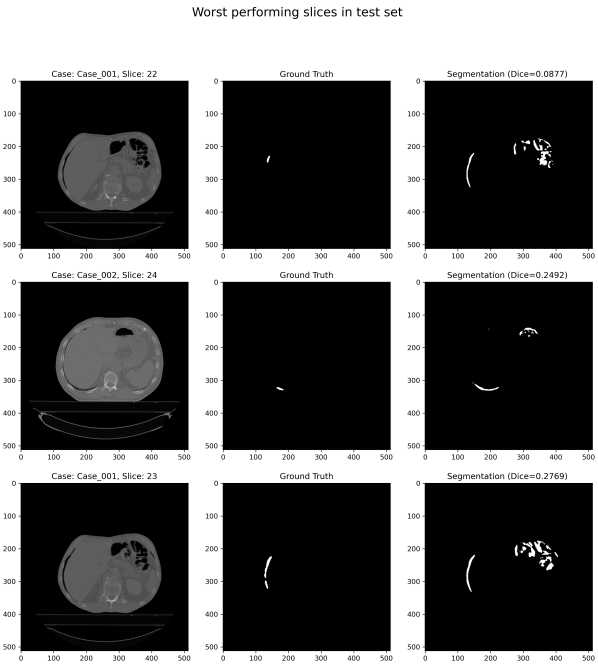


Figure 14: Comparison of the predicted masks with the ground truth masks for the test set. Cases from the test set with the worst 3 dice scores as shown.

3.5 Discussion and Future Work

Clear trends can be established from looking at the best and worst cases from the final model on both the train and test sets. Firstly, the model performs well on cases where the lungs are prominent and well-defined, as seen in the best cases. Conversely, the model struggles with cases where the lungs are less distinct, as seen in the worst cases. This is likely due to data imbalance, there are by definition going to be fewer edge slices which contain small amounts of lung tissue compared to the central slices which contain the majority of the lung tissue. As such, the model has less data to learn how to identify the lungs in the superior and inferior most axial slices which contain the lungs. Further, it is likely that the most variation in the lung shape and position occurs precisely in these slices which there are few samples of. Future work could involve data resampling techniques to address this imbalance, such as oversampling the edge slices or using data augmentation techniques to artificially generate more edge slices. Further, exploring architectures in which information between slices can be shared, such as 3D convolutions or recurrent neural networks, could be beneficial in better segmenting the lungs in the edge slices. One may wonder how it is possible the train loss shown in fig 4 seems to plateau at 0.25, indicating average dice scores of 0.75, while the average dice scores presented in table 2 are around 0.95? The answer lies in the fact during training many slices contained no lung tissue. The masks for such cases were all zeros and the dice loss was 1 regardless of the model prediction. This was not a problem for model optimisation as all the gradients would simply propate through the binary cross entropy loss term. However, it does heavily inflate the train loss as many batches would contain these background slices which would always have a dice loss of 1 regardless of the models agreement. This is why the train loss is so high compared to the test loss. Future work would also address this shortcoming of the training script.

4 Conclusion

In summary, this report presents a thorough exploration of the subset of the LCTSC dataset given as part of this coursework and trains a 2D UNet model for lung segmentation. The model

achieves good performance with an average dice score of 0.949 and an average binary accuracy of 0.997 on the test set. The model struggles with cases where the lungs are less distinct, likely due to data imbalance. Future work could involve data resampling techniques to address this imbalance, such as oversampling the edge slices or using data augmentation techniques to artificially generate more edge slices. Further, exploring architectures in which information between slices can be shared, such as 3D convolutions or recurrent neural networks, could be beneficial in better segmenting the lungs in the edge slices.

5 Appendix

5.1 CoPilot and ChatGPT usage

CoPilot was used heavily when writing the code in `src/figs.ipynb`. It was also used to create the docstrings shown in this repo. It was also used to format and generate the latex for the figures and tables in this report.