# Medical Imaging Coursework with option 2B

Cailley Factor

March 2024

## Introduction

The coursework seeks to use a deep learning model to segment the lungs from computational tomography (CT) images of twelve cases from the Lung CT Segmentation Challenge from the Cancer Imaging Archive [1]. Computational tomography of the lungs is an important imaging modality as it can be performed on the timescale of minutes to allow physicians to rapidly diagnose disease [2]. The CT scanner rotates the X-ray emission tube around the patient's body and digital X-ray detectors are used to detect the X-rays opposite the source [3]. Following a full rotation, algorithms are used to reconstruct an axial image slice of the patient [3]. The patient is moved up and down to generate different cross-sectional slices [3]. This method is advantageous over traditional X-rays which contain information about a superposition of images [3]. Examples of diseases for which lung CT images are used for diagnosis, include lung cancer, which is a leading cause of human death [4]. CT images are the most effective and feasible technique to detect lung cancer, detecting four to ten times more cases than traditional X-ray imaging [4]. Segmenting the lungs is a crucial first step to enabling further segmentation of disease-associated lung patterns [2]. Although a variety of methods exist for medical imaging segmentation, deep learning methods have been recognised as some of the most successful in automatic segmentation and this coursework seeks to use a UNet for the task of lung segmentation [5].

## Module 1: Handling DICOM data

The first task of the coursework was to convert the DICOM dataset into a NumPy array per case. The dataset directory is subdivided into two directories, one titled 'Images' and another titled 'Segmentations'. The 'Images' directory contains sub-directories titled with the patient case numbers. Within the patient case number directories, there are DICOM files, each containing metadata and image data for an axial CT slice. Within the 'Segmentations' directory, there are .npz files labelled by patient identification number containing a 3D NumPy array accessed with a key called 'Masks'. Three of the case IDs were found to contain personal information: Case_003, Case_006, and Case_007, including names, birth dates and birth times. In accordance with medical imaging best practice, this information was removed.

To convert the data into a NumPy array per case, the .npz files from the 'Segmentations' directory were sorted by file name, corresponding to case number. The masks were extracted and flipped vertically. The case number was extracted from the name of the .npz file, and used to select the corresponding directory from the 'Images' folder with the same case number. Each DICOM file within the directory was read using the pydicom module's 'dcmread' function and the DICOM attributes 'SliceLocation', 'SliceThickness', and 'PixelSpacing' were obtained using the 'getattr' method. It was found that the numbering of the DICOM file did not always correspond to their 'SliceLocation', and thus the DICOM files for each case were sorted based on their 'SliceLocation' attribute, rather than their file name. For each slice, the image data was extracted as a NumPy array using the 'pixel_array' attribute. For better viewing, the NumPy arrays were converted to Hounsfield units using the 'RescaleSlope' and 'RescaleIntercept' attributes. The sorted NumPy arrays were concatenated into a 3D array and flipped vertically for improved viewing. The 3D NumPy data was stored in a dictionary with the case information.

To ensure that the NumPy arrays were concatenated correctly, a middle coronal slice was visualised for each patient with its corresponding mask. Prior to visualisation, the NumPy arrays were scaled by the true

dimensions calculated based on the 'SliceThickness' and 'PixelSpacing' pydicom attributes. To calculate the coronal slice height, the number of slices was multiplied by the slice thickness, and to calculate the dimensions of the axial slices, the pixel spacing values were multiplied by the size of the axial slice arrays. A middle coronal slice for case 000 is visualised as an example in figure 1, alongside its corresponding segmentation mask.

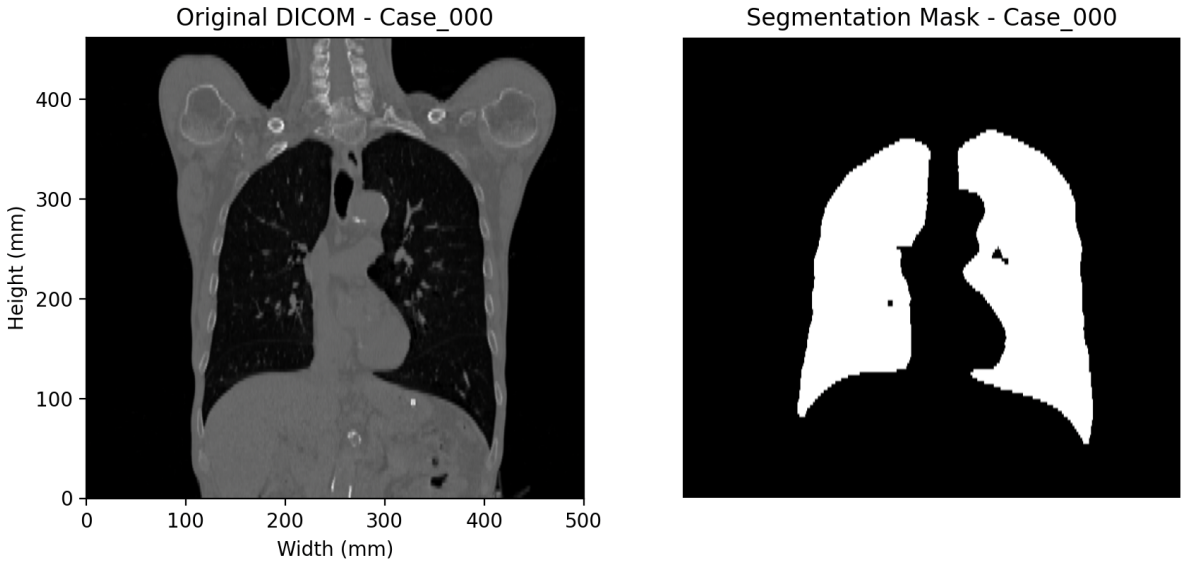


Figure 1: Middle coronal slice of image (left) and true mask (right) for case 000

# Module 2B: UNet-based segmentation

## UNet Architecture

For training a UNet model to segment lungs, the UNet architecture created in Practical 3, shown in figure 2, was implemented in the 'UNet.py' file. A UNet is an architecture consisting of an encoder, bottleneck, and decoder, as well as skip connections [6]. The encoder consists of three convolutional blocks with each block performing two sets of convolutions with a kernel size of 3x3, padding of 1, and stride of 1, followed by batch normalisation on the resulting feature maps, and a ReLU activation function. After each convolutional block, there is 2x2 max pooling with no padding and a stride of two, and two-dimensional dropout with a probability of 0.5. The bottleneck includes a convolutional block identical to the encoder blocks, but expands the channel dimensions to 128. The decoder consists of three stages of upsampling, followed by a concatenation with the corresponding encoder feature map, comprising the skip connection, and a convolutional block. Lastly, the final layer of the network maps the 16 channels to the desired output channels, i.e., one channel in the case of the black and white CT data.

## Training method

A custom loss function combining binary cross entropy and the soft dice loss was utilised as the loss for training. Binary cross entropy loss is defined in equation 1 and soft dice loss is defined in equation 2, where $y_i$ is the true mask, $p_i$ is the soft predicted segmentation (probabilities between 0 and 1) after passing the model outputs through a sigmoid function, and i represents the ith pixel of N pixels. $\epsilon$ is a small constant (e.g., $1e-6$) added for numerical stability. The soft dice loss was implemented using a custom 'SoftDiceLoss' class, and calculated for each batch, then averaged over the batch size. The binary cross entropy loss was

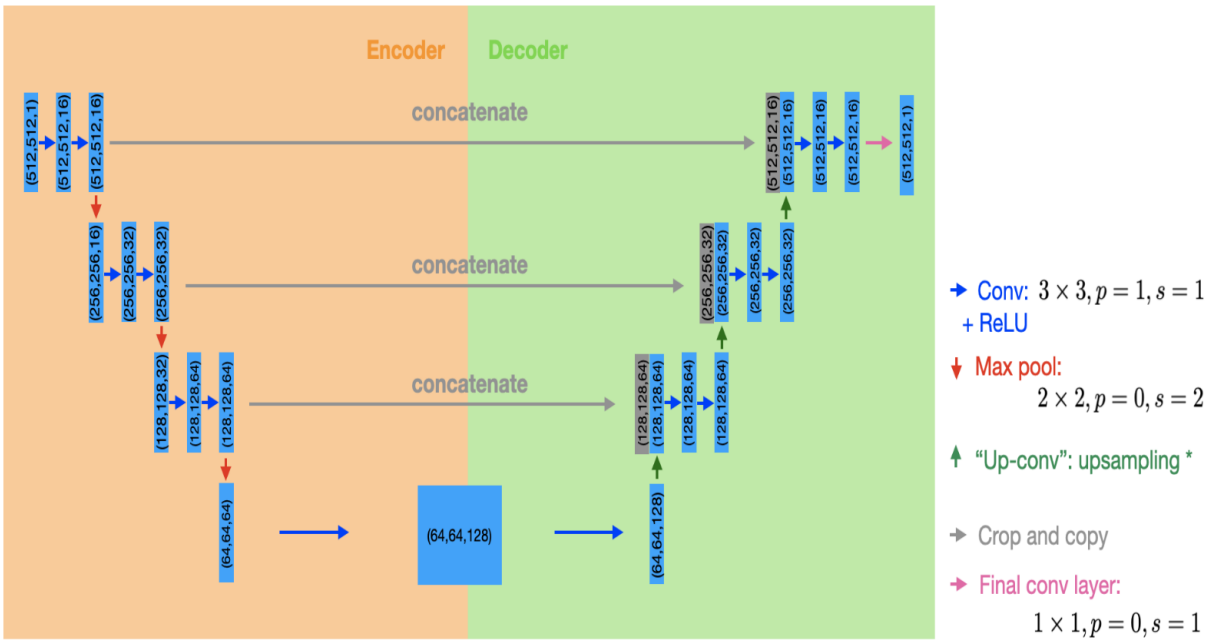


Figure 2: Diagram of the UNet architecture

implemented using the 'torch.nn' 'BCELoss' method, which by default calculates the average loss per batch. An addition of the two loss functions, each weighted by 0.5, was utilised as the custom loss function for training the model to avoid training an extra weight parameter (3).

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p_i) + (1-y_i)\log(1-p_i)\right] \tag{1}$$

$$L_{DSC}(p(x), y) = 1 - \frac{2\sum_{i=1}^{N} y_i p_i}{\sum_{i=1}^{N} y_i + \sum_{i=1}^{N} p_i + \epsilon} \tag{2}$$

$$L_{custom}(p(x), y) = 0.5 L_{BCE}(p(x), y) + 0.5 L_{DSC}(p(x), y) \tag{3}$$

A custom dataset class inheriting from the PyTorch 'Dataset' class was created. The constructor takes in the DICOM directories and corresponding mask paths and iterates over them, sorting the DICOM data by the 'slice_location' attribute. Each sorted slice and corresponding mask slice is stored in a list, as well as the case number and slice location. The 'len' method ensures that the number of DICOM slices matches the number of segmentation masks and returns the total number of slices in the dataset. The 'getitem' method is used to retrieve a specific item from the dataset using an index and converts the DICOM slices and segmentation masks into PyTorch tensors, adding a channel dimension at the beginning of these tensors, as required by the UNet input format. This method returns these tensors, alongside the case number and slice location.

For training, a random seed of 42 is set for PyTorch function reproducibility. The custom dataset class is instantiated and the list of all sub-directories within the 'Images' and the .npz files within the 'Segmentations' directory is fed into the custom dataset class to yield the slice and mask tensor pairs, as well as case number and slice location. The dataset is divided into train (2/3) and test (1/3) datasets, using PyTorch's 'random_split' function. From PyTorch 'DataLoader', data loaders are used to wrap around the datasets to enable the batching of the data and also the shuffling of the data, so the model doesn't simply learn the order of the data.

The 'train.py' module contains the training loop for the model. It initialises the UNet model from 'UNet.py' described in the previous section, as well as the custom loss function and an Adam optimiser.

Lists are initialised to store the training and validation results from the model. At each epoch, the model is set to training mode and the validation and training losses and accuracies are set to 0. For each batch in the training data loader, the old gradients are cleared. For the forward pass, the images from the training data loader are passed through a model and then through a sigmoid function to yield a prediction. The loss function is calculated based on the predictions and true masks. Gradients are calculated for the backward pass and then the model parameters are updated based on the gradients. The accuracy is calculated based on binary accuracy with a threshold of 0.5. The average batch loss and accuracy are accumulated.

The model is then set to evaluation mode and PyTorch's 'no_grad' method is implemented in order to prevent gradient calculations during the evaluation loop. The evaluation loop in the training method calculates the average batch loss and accuracy for the test set for each epoch. The average loss and accuracy per batch for the epoch is calculated by dividing by the number of batches and returned for each epoch.

## Analysis & Discussion

The average batch loss and accuracy per epoch from training 10 epochs with a batch size of 3 and a learning rate of 0.1 is displayed in figure 3.

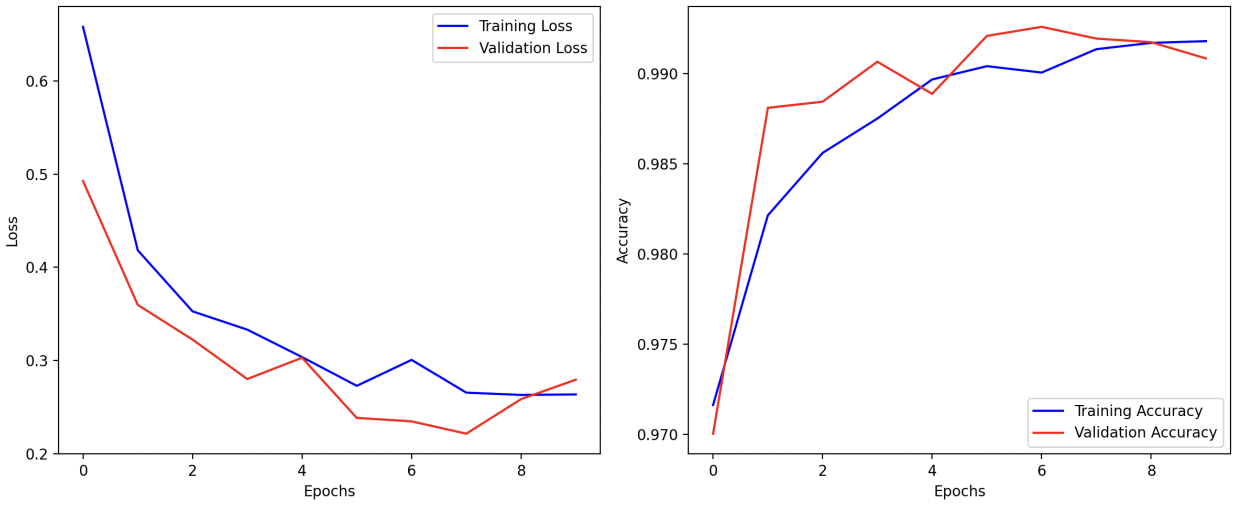


Figure 3: Plot of the average batch loss (left) and average batch accuracy (right) per epoch for the training and testing data

A method in 'eval.py' was created to compare the predictions from the trained model to the ground truth segmentation masks. The predictions are created by passing the outputs from the model through a sigmoid function. For the dice calculation, a threshold of 0.5 was also applied; this can be directly incorporated into the accuracy calculation with the 'BinaryAccuracy' method from 'torchmetrics'. The dice score coefficient is calculated as in equation 4, where $\epsilon$ is a small constant (e.g., $1e-6$) added for numerical stability.

$$\text{Dice} = \frac{2 \times \sum(y_{true} \cdot y_{pred})}{\sum(y_{pred}) + \sum(y_{true}) + \epsilon} \tag{4}$$

For each index in the training and testing datasets, the accuracy and dice score was calculated, and appended to a dictionary, alongside the mask and prediction array tensors converted to NumPy arrays, and key DICOM data attributes such as the slice location and case number. Histograms were plotted of the accuracy and dice score coefficient (DSC) (4), as shown in figure 4 for the training dataset and figure 5 for the testing dataset.

A great deal of similarity between the histograms plotting accuracy and DSC for the training and testing datasets is present. A notable feature of the DSC histograms is the high frequency of DSC values of 0. This is due to the fact that the DSC will be 0 for slices where the true mask is all 0s, i.e., the lungs are not present

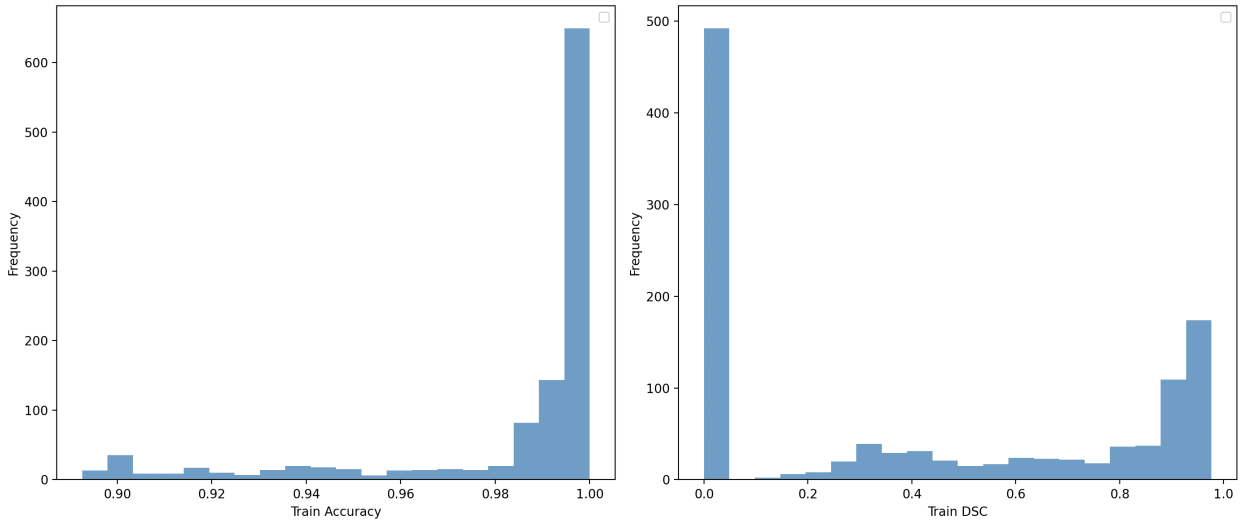


Figure 4: Histograms of accuracy (left) and DSC (right) per 2D slice for the training dataset

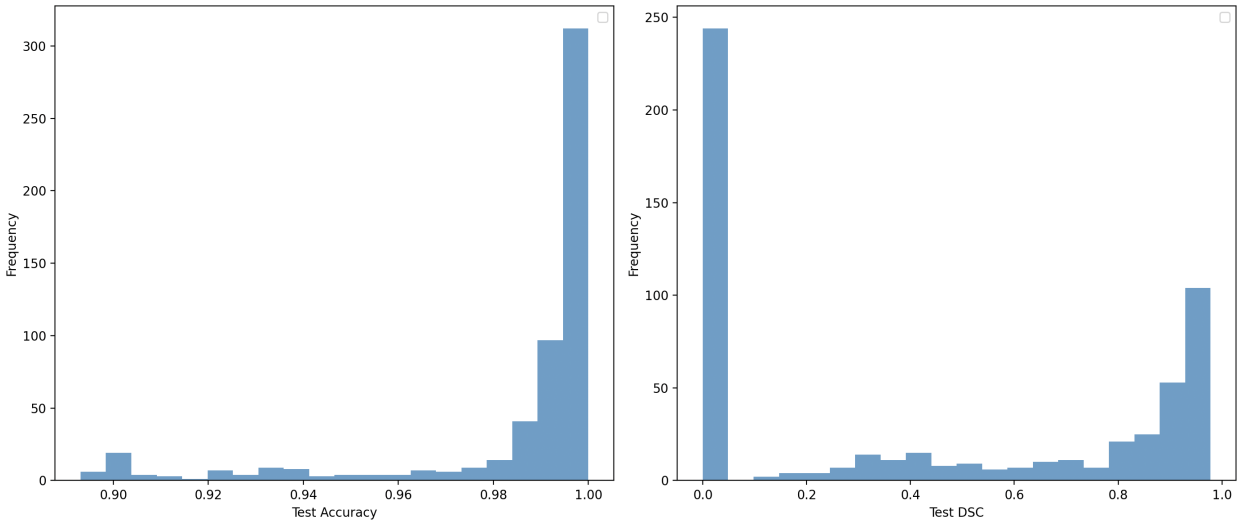


Figure 5: Histograms of accuracy (left) and DSC (right) per 2D slice for the testing dataset

in the slice. This is because the numerator of the DSC will be 0, even if the denominator is very small or epsilon. Examples of of DSC values of 0 can be seen in figure 6, which plots the real image, ground truth segmentation, and predicted mask for axial slices from three different patients with the lowest DSC values, i.e., DSC of 0. Notably, these figures are accompanied by high accuracy values, which reflects the fact that the model is predicting mostly or all 0s correctly. Thus, the DSC score is a poor reflection of the accuracy of the model when there is no lung segmentation present in the image, and likewise the dice loss component of the custom loss function is also sub-optimal for training on these slices.

Figures 7 and 8 plot the real image, ground truth segmentation, and predicted mask for axial slices with median DSC values, and high DSC values, respectively. The high DSC slice values are accompanied by higher accuracies than the slice values with median DSC values, but lower accuracy values than those for the slices with 0 DSC for the reasons discussed above.

The results show that the UNet model has clearly learned to segment lungs, exemplified by the images in figure 8. Given the challenges of the dice score for slices lacking lungs, a useful improvement would be to weight the two loss components using a learned parameter at the cost of increased computational complexity. A threshold of 0.5 is used to make the predictions for the evaluation method for accuracy and DSC calculation. The patchy appearance of the segmentation of figure 7 suggests that a potential improvement could be decreasing the threshold for the evaluation method, although this may lead to slightly reduced accuracy in the slices with no lungs with more pixels wrongly being classified as part of lungs. Other potential improvements in the results include training on more data than 12 cases, which was a significant limitation. Different data augmentation strategies could be implemented such as rotation and flipping. Architectural enhancements could also be implemented such as increasing the depth, i.e., the number of layers, in the model, as well as the width, i.e., the channel dimensions of the hidden layers of the model, though, also at the expense of greater computational complexity. Overall, however, the model performed remarkably well at lung segmentation given the small training set, especially measured by the accuracy metric.

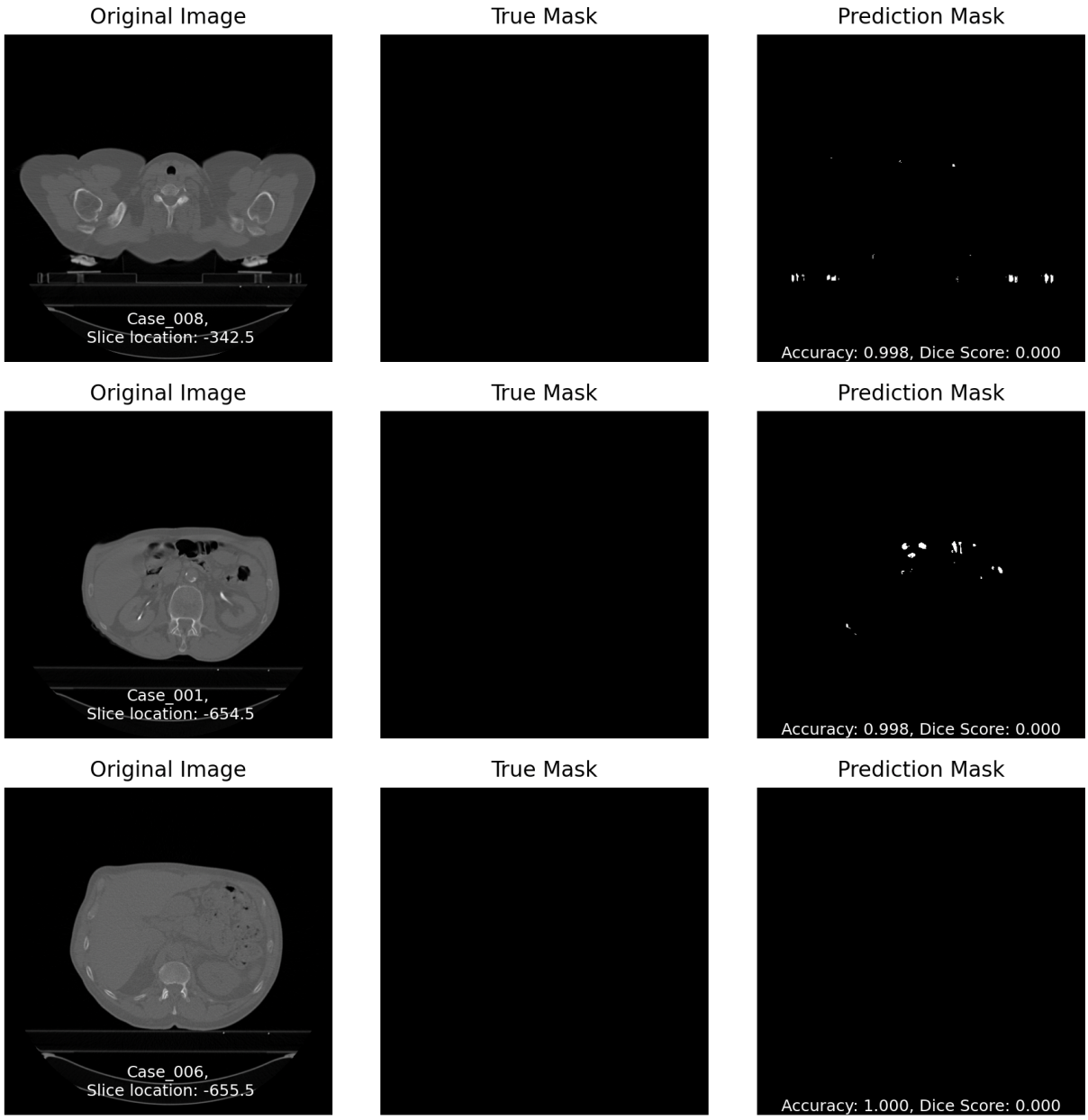


Figure 6: Real image (left), ground truth segmentation (middle) and prediction (right) with low DSC values from three different patients

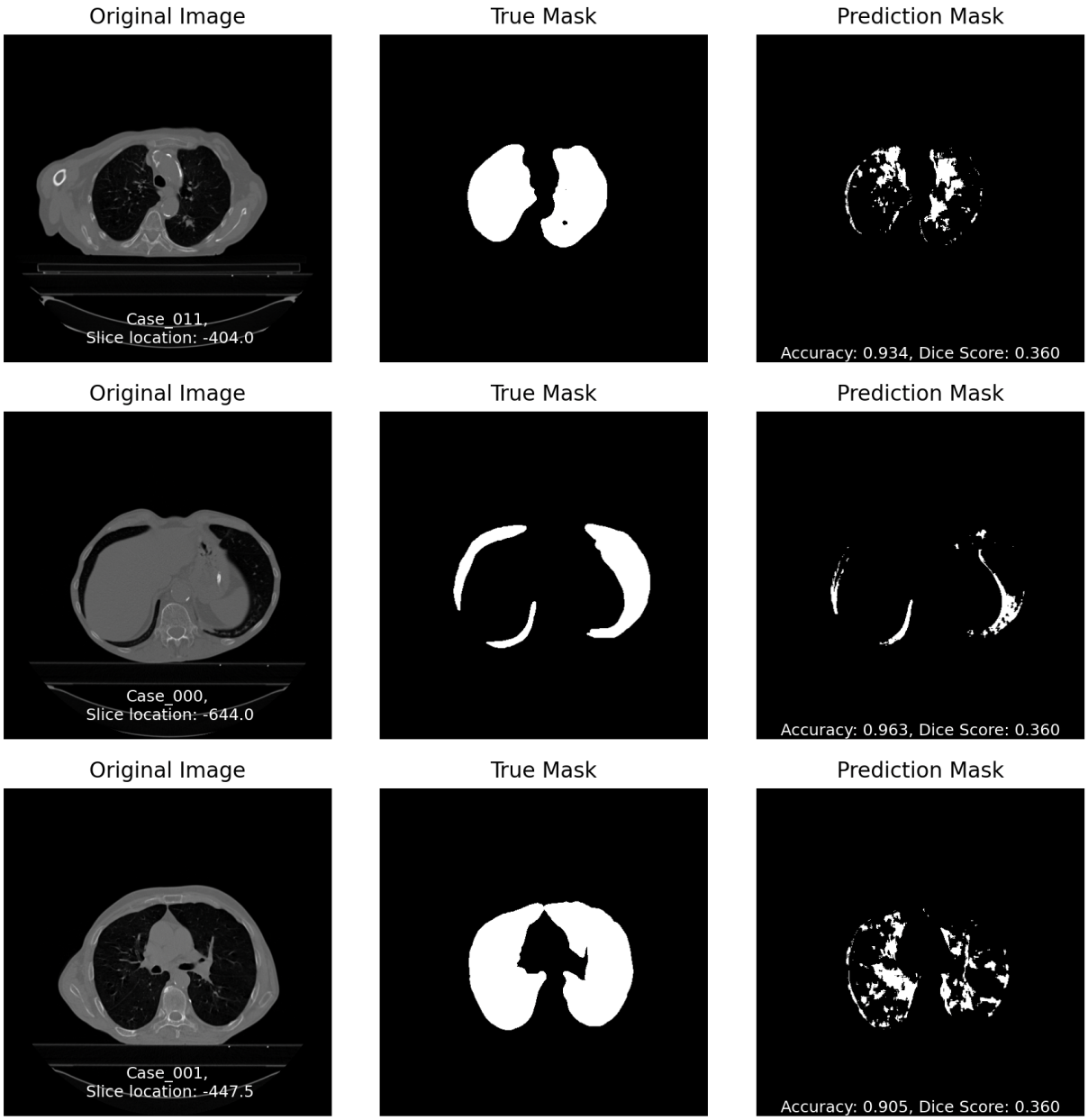


Figure 7: Real image (left), ground truth segmentation (middle) and prediction (right) with near median DSC values from three different patients

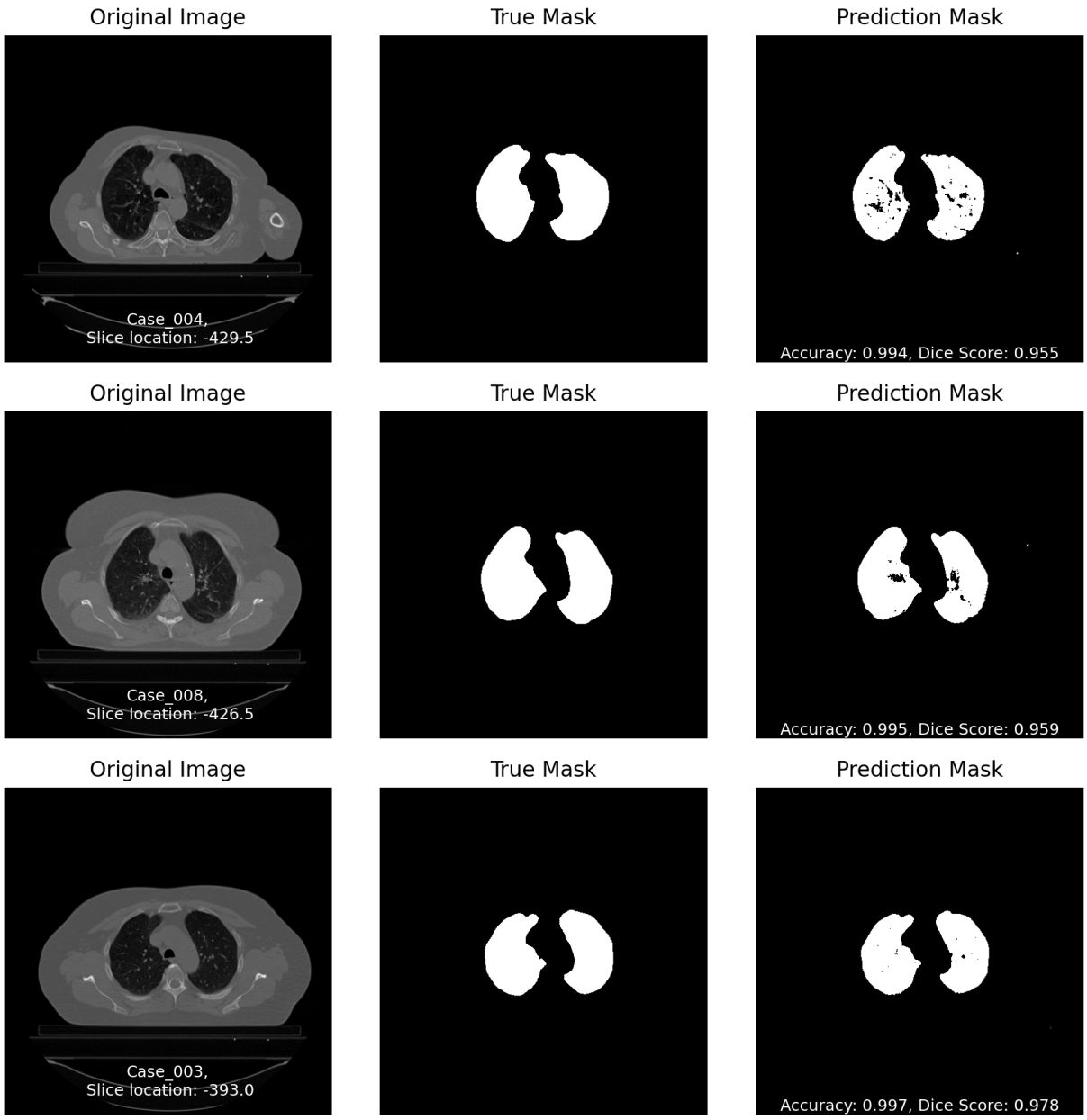


Figure 8: Real image (left), ground truth segmentation (middle) and prediction (right) with high DSC values from three different patients

# References


[1] Tracy Nolan and Michael Rutherford. *Lung CT Segmentation Challenge 2017 (LCTSC)*. Pages Wiki Collections. Aug. 2023.

[2] Diedre Carmo et al. "A Systematic Review of Automated Segmentation Methods and Public Datasets for the Lung and its Lobes and Findings on Computed Tomography Images". In: *Yearb Med Inform* 31.1 (Aug. 2022), pp. 277–295. DOI: 10.1055/s-0042-1742517.

[3] Paula R. Patel and Orlando De Jesus. *CT Scan*. Last Update: January 2, 2023. StatPearls Publishing, 2023.

[4] Hanguang Xiao et al. "The Progress on Lung Computed Tomography Imaging Signs: A Review". In: *Appl. Sci.* 12.18 (2022), p. 9367. DOI: 10.3390/app12189367. URL: https://doi.org/10.3390/app12189367.

[5] G. Almeida and J.M.R.S. Tavares. "Versatile Convolutional Networks Applied to Computed Tomography and Magnetic Resonance Image Segmentation". In: *J Med Syst* 45.79 (2021), pp. 1–2. DOI: 10.1007/s10916-021-01751-6. URL: https://doi.org/10.1007/s10916-021-01751-6.

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)* 9351 (May 2015), pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28. URL: http://lmb.informatik.uni-freiburg.de/.


# Auto-generation tool citations

ChatGPT 4.0 was used for the following tasks: - Generating names for figures based on their index values for producing examples of 2D slices with their corresponding predicted masks and true masks in 'main.py'. Alongside the existing code for plotting the values, the following prompt was used: 'how to name these files based on their unique index'. - Improving the plot_images function in 'main.py'. Code was submitted to ChatGPT alongside the prompt: 'How to fix the code such that the three images produced for each index in the loop make a 3x3 grid'. - Flattening y_pred and y_true in the forward method for calculating the soft dice loss in 'loss.py'. Alongside the image dimensions and the existing code, the following prompt was submitted: 'Best way to flatten these images'.

GitHub Copilot was used to help write documentation for Doxygen software and comments within the code.