# Unet_based Model for Lung Segment

Jingjing Gao[1]

[1]Department of Physics, University of Cambridge, Cambridge, United Kingdom

**Abstract**

Lung segmentation is a popular problem in medical imaging. In this study, I employ a U-Net-based deep learning model to segment lungs in CT images. The model is trained using losses including binary cross-entropy and the soft Dice loss, and the binary accuracy metric is used to evaluate the model's performance. Subsequently, the model is evaluated, and the Dice similarity coefficient (DSC) along with 2D slice examples from different patients are presented. Finally, the results of the model are analyzed, and potential future developments in lung segmentation are discussed.

**Keywords**: Deep Learning, Medical Image, Lung Segmentation, U-Net.

## 1. Introduction

Lung segmentation is a crucial step in lung cancer diagnosis using computer-aided diagnosis systems. However, achieving accurate lung segmentation poses significant challenges due to factors such as image noise, various lung diseases, the presence of lung nodules, and unique morphological variations. In this study, I propose a novel algorithm for lung segmentation in thoracic Computed Tomography (CT) images based on the U-Net model.

## 2. Methods

### 2.1 Introduction to the U-Net Model

The U-Net architecture was originally developed for medical image understanding and segmentation, finding widespread application in the medical imaging domain and becoming a cornerstone in medical imaging automation. In this section, I provide an in-depth overview of the key technical aspects of this network and their contributions to achieving robust segmentation results.

The architecture of the U-Net model comprises two main components: the contracting path and the expansive path. The contracting path consists of multiple convolutional layers with filters of size $n \times n$ and unity strides in both directions, followed by Rectified Linear Unit (ReLU) activation functions. This path extracts essential features from the input and yields a feature vector of a specific length.

The expansive path retrieves information from the contracting path through a combination of copying and cropping, along with information from the feature vector via up-convolutions. Through successive operations, this path generates an output segmentation map. The pivotal aspect of this architecture lies in the connection between the contracting and expansive paths, facilitating the network's ability to capture highly accurate information from the contracting path and produce a segmentation mask that closely aligns with the desired output.

## 2.2 Task Description

The UNet-based model is designed to fulfill the primary objective of segmenting lungs in CT images using a supervised learning approach. Segmentation involves training the model with provided data, where it learns to map slice images to their corresponding segmented areas. This task is essentially a specialized form of classification, necessitating a pre-labeled mask image to guide the segmentation process.

The segmentation process encompasses several submodules, each dedicated to specific tasks such as data preprocessing, model training, and result analysis. Initially, the data undergoes preprocessing and conversion from its original format to a format suitable for input into the model. PyTorch, a widely-used deep learning framework, is leveraged for training the model on the segmented data, enabling efficient optimization and parameter tuning.

Following training, the model's performance is assessed using various metrics, including the Dice Similarity Coefficient, to evaluate its accuracy and generalization capability. Result analysis involves visualizing the segmented images, comparing them with ground truth masks, and quantitatively measuring the model's performance.

## 2.3 Data Preprocessing

### 2.3.1 MODULE 1: Handling DICOM data

The dataset comprises 1100 2D slice images obtained from CT scans of 12 patients. Each slice has a resolution of 512 x 512 pixels and is stored in DICOM format. Additionally, the dataset includes corresponding mask images delineating the lung area, facilitating supervised segmentation tasks.

I choose to use methods seen in "Practical2" to assist me in handling DICOM data. I design a `dicom_to_numpy` function to facilitate the conversion of DICOM datasets to NumPy arrays with a systematic approach. It begins by iteratively reading each file

within the specified DICOM directory using the `dcmread` function from `pydicom`, ensuring accurate loading of DICOM files. Subsequently, the function organizes the DICOM slices based on their z-coordinate, crucial for maintaining the spatial integrity of the 3D volume, achieved through sorting the slices according to `ImagePositionPatient[2]`. Following this, pixel arrays are extracted from each DICOM slice, containing the essential image data necessary for volume reconstruction. Finally, these pixel arrays are stacked along the last axis using `np.stack`, effectively constructing a 3D NumPy array representing the volume, with each slice represented as a 2D array along the third axis.

In parallel, a list of case directories is generated by concatenating the root directory where all cases are stored (`Dataset/Images`) with the respective case directory names (e.g., `"Case_001"`, `"Case_002"`, ..., `"Case_011"`). Subsequently, an empty list `case_volumes` is initialized to accumulate the NumPy arrays corresponding to each case. Through iteration over each case directory, the `dicom_to_numpy` function is invoked, converting DICOM datasets into NumPy arrays representing the volumes for each case. The resulting NumPy arrays are then appended to the `case_volumes` list, providing comprehensive storage for subsequent analysis or processing.

### 2.3.2 Prepare Datasets

I have designed two key functions aimed at facilitating data preparation for lung segmentation tasks in CT images. Both functions are focused on converting images and masks to PyTorch tensors, ensuring compatibility with deep learning models.

#### Prepare Images Tensor

The `prepare_images_tensor` function is responsible for converting DICOM datasets into a PyTorch tensor containing image data. It starts by generating a list of case directories corresponding to the DICOM datasets within the specified dataset directory. For each case directory, the function utilizes a separate function, `dicom_to_numpy`, to convert DICOM datasets into NumPy arrays. These arrays undergo reshaping to ensure a consistent dimensionality of (512, 512) across all slices. Subsequently, the reshaped slices are consolidated into a single NumPy array, which is then converted into a PyTorch tensor.

#### Prepare Masks Tensor

The `prepare_masks_tensor` function focuses on preparing a PyTorch tensor containing mask data from segmentation files. It iterates over segmentation files within the dataset directory, loading segmentation masks from NPZ files using NumPy. Similar to the image preparation process, the segmentation masks are reshaped to maintain uniformity in dimensions. Once reshaped, the masks are aggregated into a single NumPy array and subsequently converted into a PyTorch tensor.

**Dataset Division**

Additionally, I have divided the provided dataset into training and test datasets with a split ratio of 2:1, respectively. This division ensures that the deep learning models can be trained and evaluated effectively for lung segmentation tasks.

## 2.4 Model Training

### 2.4.1 UNet Architecture

The UNet class is a neural network architecture commonly used for semantic segmentation tasks, particularly in medical image analysis. It consists of an encoder-decoder architecture with skip connections, allowing for precise localization of features at different scales. Figure 1 illustrates the UNet architecture.

1. **Initialization:** The `__init__` method initializes the UNet class. It takes two parameters: `in_channels` and `out_channels`, representing the number of input channels (e.g., for RGB images, `in_channels` would be 3) and the number of output channels (e.g., for binary segmentation, `out_channels` would be 1).

2. **Encoder Blocks:** The UNet architecture begins with a series of convolutional blocks in the encoder part of the network. Each convolutional block consists of a convolutional layer followed by an activation function (commonly ReLU) and optionally other layers such as batch normalization and dropout. These blocks progressively reduce the spatial dimensions of the input while increasing the number of feature channels.

3. **Maxpooling:** After each convolutional block, maxpooling layers are applied to reduce the spatial dimensions of the feature maps, thus capturing increasingly abstract representations of the input image.

4. **Middle Layers:** At the middle of the network, there is typically a series of convolutional blocks with larger receptive fields to capture more global features of the input.

5. **Decoder Blocks:** The decoder part of the network consists of a series of upsampling and concatenation operations followed by convolutional blocks. The upsampling operations increase the spatial dimensions of the feature maps, while the concatenation operations combine feature maps from the encoder with those from the decoder via skip connections.

6. **Final Layer:** The final layer of the network consists of a convolutional layer with a single output channel, which produces the segmentation mask for the input image.

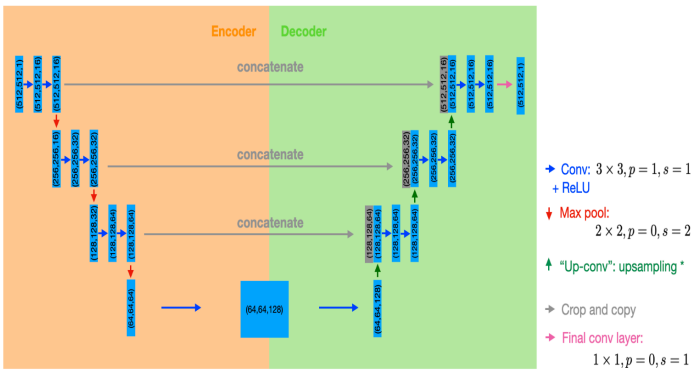


Figure 1: UNet Architecture

### 2.4.2 Loss Function

The choice of a custom loss function combining binary cross-entropy (BCE) loss and soft Dice loss is motivated by the need to address the limitations of using binary cross-entropy alone in image segmentation tasks. While binary cross-entropy loss is effective for comparing individual pixels, it may not accurately capture the similarity between predicted and ground truth segmentation masks, especially when dealing with small segmented areas.

In contrast, the soft Dice loss offers a more nuanced measure of similarity by considering the overlap between predicted and ground truth masks. It calculates the Dice coefficient, which reflects the ratio of the overlapping area to the total area. Incorporating soft Dice loss alongside binary cross-entropy provides a complementary metric that accounts for segmentation similarity beyond pixel-wise accuracy.

The custom loss function is defined as a combination of binary cross-entropy and soft Dice loss, aiming to leverage the strengths of both metrics. It can be represented as:

$$L_{\text{custom}} = \alpha \times L_{\text{BCE}} + (1 - \alpha) \times L_{\text{Dice}}$$

where:

- $L_{\text{custom}}$ is the custom loss function,
- $L_{\text{BCE}}$ is the binary cross-entropy loss given by:

$$L_{\text{BCE}} = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

  where $N$ is the total number of samples, $y_i$ is the ground truth label for sample $i$, and $\hat{y}_i$ is the predicted probability for sample $i$.

- $L_{\text{Dice}}$ is the soft Dice loss given by:

$$L_{\text{Dice}} = 1 - \frac{2 \sum_{i=1}^{N} y_i \hat{y}_i + \epsilon}{\sum_{i=1}^{N} y_i + \sum_{i=1}^{N} \hat{y}_i + \epsilon}$$

where $\epsilon$ is a small constant added to avoid division by zero, $y_i$ is the ground truth binary mask for sample $i$, $\hat{y}_i$ is the predicted binary mask for sample $i$, and $N$ is the total number of samples.

By adjusting the value of $\alpha$, the relative importance of binary cross-entropy and soft Dice loss can be fine-tuned based on the specific requirements of the segmentation task. This combined loss function offers a comprehensive approach to evaluating model performance, considering both pixel-wise accuracy and segmentation similarity.

Without any prior knowledge, I choose $\alpha = 0.5$, so the binary cross-entropy loss ($L_{\text{BCE}}$) and soft Dice loss ($L_{\text{Dice}}$) have the same weight.

### 2.4.3 Training Process

The training process involves iteratively optimizing the parameters of the UNet model to minimize the costum loss function in part 2.4.2 on the training dataset.

1. **Model Initialization**: The process begins by initializing the UNet model. This involves defining the architecture of the model, including the number of layers, types of layers, and activation functions.

2. **Data Loading**: Training images and their corresponding masks are loaded into the function using PyTorch's DataLoader. This enables efficient loading and processing of the dataset, particularly beneficial when handling large amounts of data.

3. **Optimization**: The Adam optimizer is chosen for parameter optimization. Adam's adaptive learning rate and momentum properties accelerate convergence and prevent the model from getting stuck in local minima.

4. **Training Loop**: The function implements a training loop that iterates over the entire training dataset for a specified number of epochs. Within each epoch, mini-batches of data are further iterated over, allowing the model to learn from multiple examples simultaneously.

5. **Forward Pass and Loss Calculation**: For each mini-batch, the model performs a forward pass to compute predicted segmentation masks. The loss function (`custom_loss`) is then calculated based on the predicted masks and the ground truth masks. Here, a combination of binary cross-entropy and soft Dice loss is used to provide a comprehensive measure of accuracy and segmentation similarity.

6. **Backward Pass and Parameter Update**: After computing the loss, a backward pass is performed to compute gradients of the loss function with respect to the model parameters. These gradients are then used by the optimizer to update the model's parameters, minimizing the loss.

7. **Metric Calculation**: During training, binary accuracy is computed as an additional metric to monitor the model's performance. Binary accuracy measures the proportion of correctly classified pixels in the segmentation masks.

8. **Model Saving**: Upon completion of training, the trained model's parameters are saved to a file named `'trained_model.pt'`. This allows the model to be loaded and used for inference or further fine-tuning in subsequent sessions.

## 2.5 Model Evaluation

I implement the `evaluate_model` function to predict and assess the performance of the trained UNet model on the test dataset. This function executes the following steps:

1. **Setting the Model to Evaluation Mode**: Initially, the model is set to evaluation mode using `model.eval()`. This ensures that the model behaves deterministically during inference, disabling features like dropout or batch normalization, which are typically active during training.

2. **Preparing the Test Data Loader**: The function creates a data loader for the test dataset using PyTorch's `DataLoader` class. Each batch contains a single image along with its corresponding ground truth mask.

3. **Iterating Over Test Samples**: The function iterates over the test data loader, processing one sample (image and mask pair) at a time.

4. **Performing Inference**: For each sample, the input image is forwarded through the model to obtain the predicted segmentation mask. The model's output is passed through a sigmoid function to ensure the values are in the range [0, 1], representing probabilities.

5. **Computing Dice Similarity Coefficient (DSC)**: The Dice Similarity Coefficient is calculated for each sample, quantifying the similarity between the predicted segmentation mask and the ground truth mask. I set the threshold=0.5, so the predicted segmentation mask changes to binary values 0 or 1. Then, the following formula is applied to calculate the DSC:

$$DSC = \frac{2 \times TP}{2 \times TP + FP + FN}$$

Where:

- $TP$ (True Positives) represents the number of pixels that are correctly predicted as belonging to the segmented region in both the predicted mask and the ground truth mask.

- $FP$ (False Positives) represents the number of pixels that are incorrectly predicted as belonging to the segmented region in the predicted mask but actually do not belong to it in the ground truth mask.
- $FN$ (False Negatives) represents the number of pixels that are incorrectly predicted as not belonging to the segmented region in the predicted mask but actually belong to it in the ground truth mask.

The DSC ranges from 0 to 1, where a value of 1 indicates perfect overlap between the predicted and ground truth masks, while lower values indicate less overlap and thus poorer segmentation performance.

6. **Computing Binary Accuracy**: Additionally, binary accuracy is computed for each sample, measuring the proportion of correctly classified pixels in the predicted mask compared to the ground truth mask. The formula for binary accuracy is:

$$\text{Accuracy} = \frac{\text{Total number of correctly classified pixels}}{\text{Total number of pixels}}$$

This metric measures the proportion of correctly classified pixels in the predicted mask compared to the ground truth mask.

7. **Aggregating Results**: The computed DSC and binary accuracy values are collected for each sample, along with the real (input) images, predicted masks, and ground truth masks. These results are stored in lists for further analysis or visualization.

# 3. Results and Discussion

## 3.1 Experiment Settings

The UNet model was initially trained for 10 epochs with a batch size of 3 and a learning rate of 0.1. While a learning rate of 0.1 is typically considered large and may not yield optimal results, the Adaptive Moment Estimation (ADAM) optimizer was chosen to dynamically adjust the learning rate during training, mitigating potential issues.

For training, the provided Dockerfile and environment file are configured for execution on a local CPU, which might take up to 2 hours to complete the training process. However, to expedite training, the model was trained on Google Colab using a GPU (Nvidia Tesla T4 GPU), resulting in a significantly reduced runtime of approximately 7 minutes with the default parameters.

Throughout the training process, losses and binary accuracies were computed for each epoch and recorded in lists. These metrics serve as valuable insights for monitoring the model's performance and visualizing the segmentation results.

Subsequently, predictions were made and the trained UNet model was evaluated on the test dataset. The Dice Similarity Coefficient (DSC) was computed between the pre-

dicted segmentations and the ground truth segmentations to quantify the level of agreement between them. This evaluation process provides a comprehensive assessment of the model's segmentation accuracy and effectiveness.

## 3.2 Resluts Analysis

### 3.2.1 Losses and Accuracies for Training Process

Figure 2 illustrates the curves of loss and accuracy per epoch for 10 epochs while training with a batch size of 3. As expected, the loss curve shows a downward trend, and the accuracy curve trends upward overall. To demonstrate the convergence of the loss curve, I also trained the model for 20 epochs and plotted the loss curves and accuracies in Figure 3. It is evident that the loss converges as the number of epochs increases.

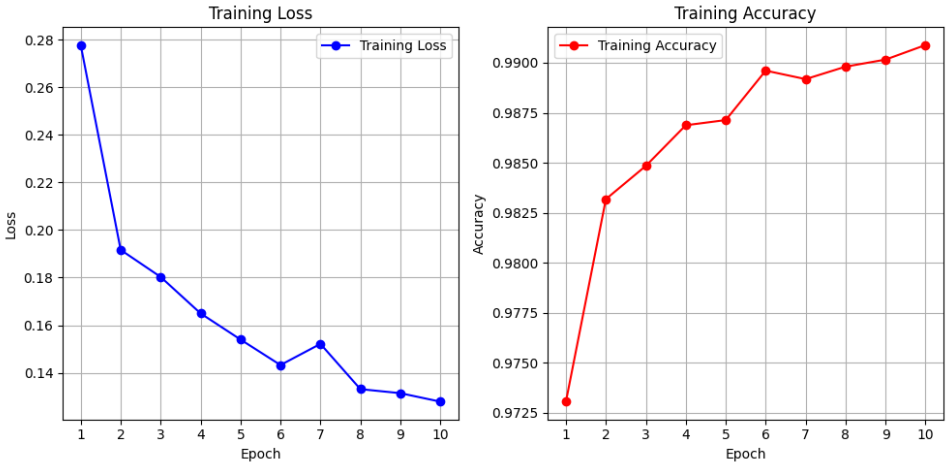


Figure 2: Losses and Accuracies for 10 Epoches

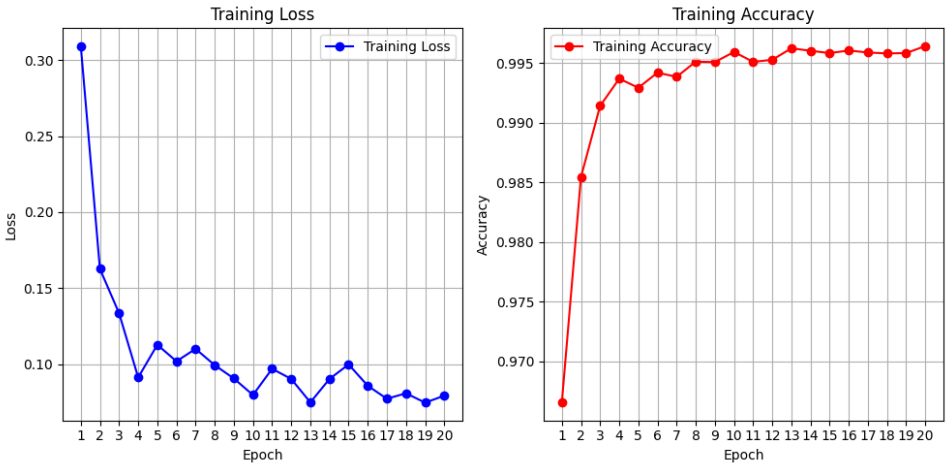


Figure 3: Losses and Accuracies for 20 Epoches

After training is completed, I save the trained model and use it to predict the 2D slices of images from both the training and testing sets.

I then plot the accuracy and Dice Similarity Coefficient (DSC) for the predicted masks compared to the ground truth masks. The accuracy curve and DSC curve for the training dataset are presented in Figure 4, while those for the test dataset are shown in Figure 5.

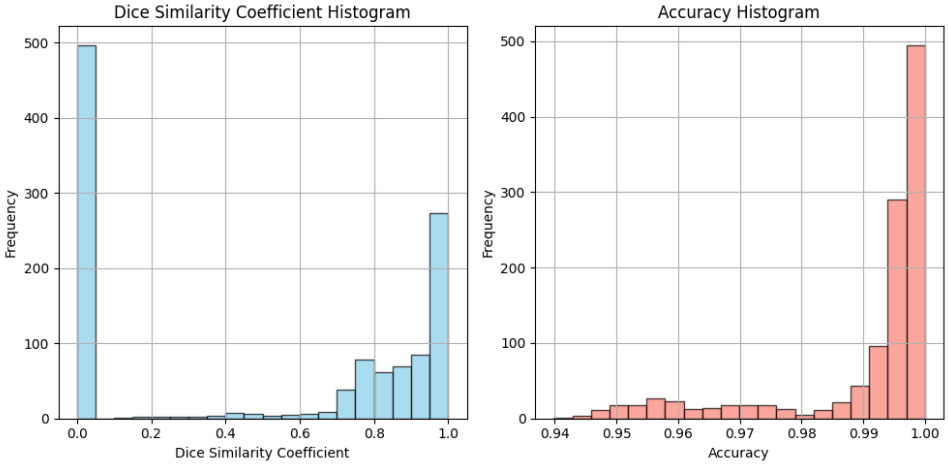


Figure 4: Accuracy and DSC Curves for Training Dataset

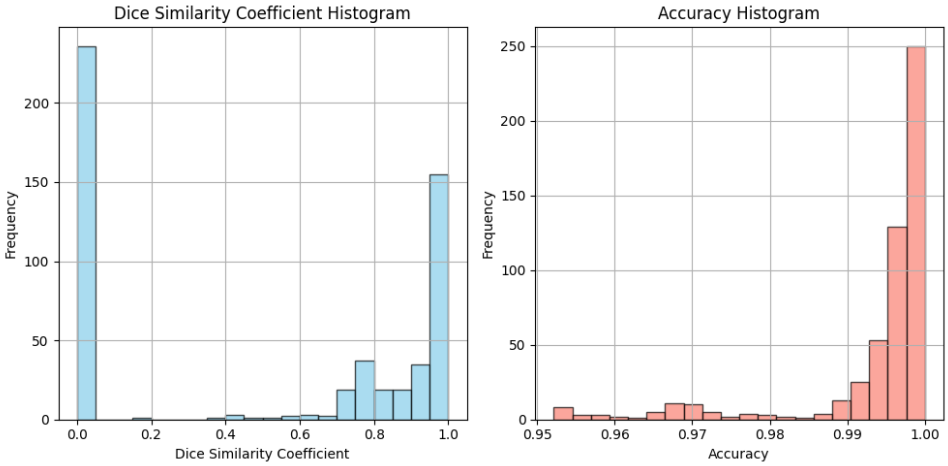


Figure 5: Accuracy and DSC Curves for Test Dataset

I examined the frequency distribution of the Dice Similarity Coefficient (DSC) and accuracy for both the training and test datasets. Surprisingly, nearly half of the samples in the datasets exhibited a DSC value of 0, indicating a lack of correspondence between the predicted and ground truth masks.

Further investigation revealed that approximately half of the ground truth masks were entirely black, rendering them uninformative for evaluation purposes.

Interestingly, despite these challenges, the DSC distributions were comparable between the training and test datasets. However, it was notable that the training dataset contained a higher proportion of samples with elevated DSC values compared to the test dataset.

This discrepancy in DSC values suggests a potential issue with overfitting. During training, the model may have become overly accustomed to the specific patterns present in the training data, leading to reduced generalization performance on unseen test data.

To address this, strategies such as regularization techniques, data augmentation, or adjustments to the model architecture could be implemented to enhance the model's ability to generalize beyond the training dataset.

### 3.2.2 Examples of 2D Slices

To illustrate examples of 2D slices in the train dataset with varying DSC values, I selected samples representing the best, worst, and in-between DSC values. These examples are visualized in Figure 6, Figure 7, and Figure 8, respectively. Each figure showcases three examples from the respective category, with each row containing the real image (left), the ground truth segmentation (middle), and the corresponding prediction (right).

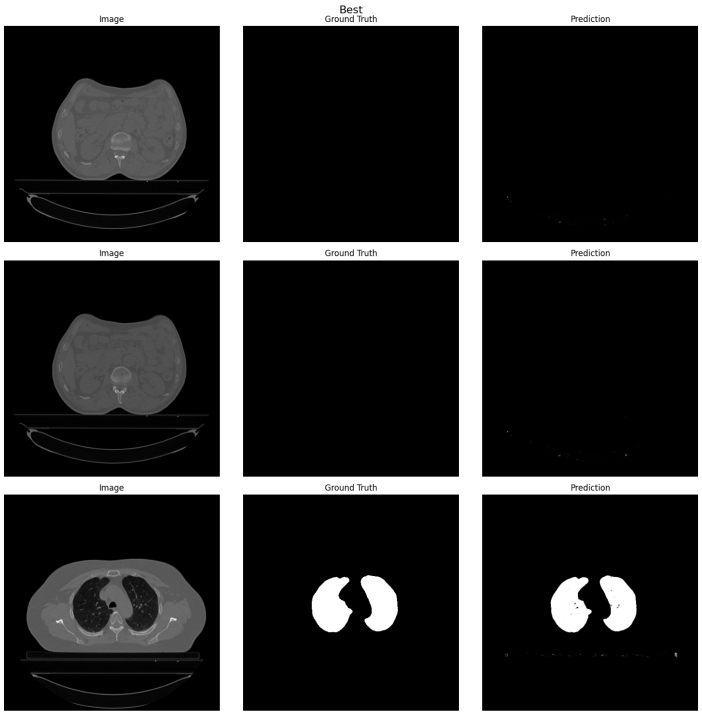


Figure 6: Examples with Best DSC Values in the Train Dataset

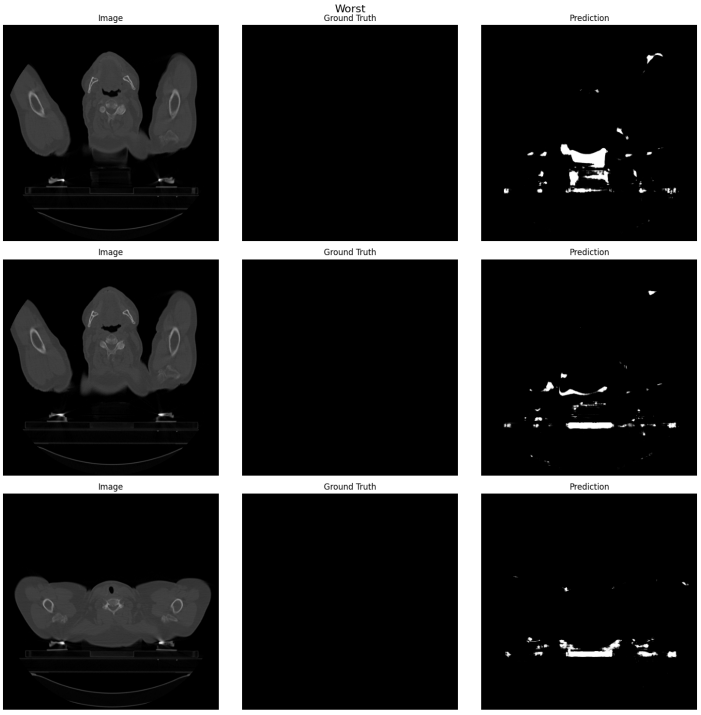


Figure 7: Examples with Worst DSC Values in the Train Dataset

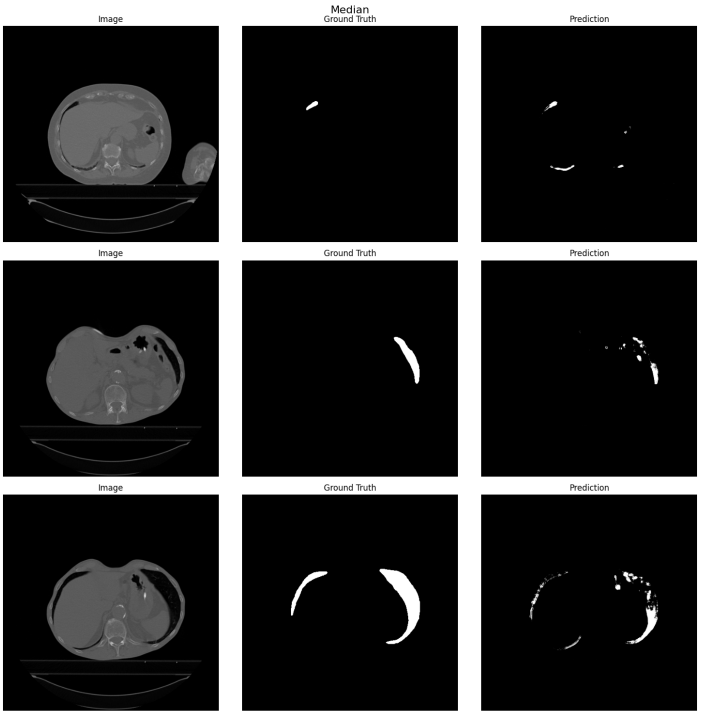


Figure 8: Examples with In-Between DSC Values in the Train Dataset

Similarly, for the test dataset, Figure 9 illustrates examples with the best DSC values, Figure 10 shows examples with the worst DSC values, and Figure 11 exhibits examples with intermediate DSC values.

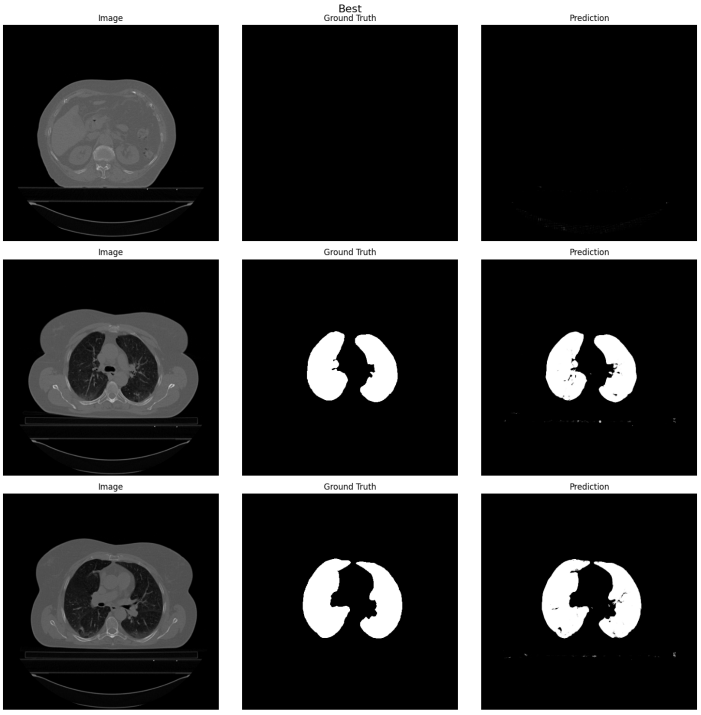


Figure 9: Examples with Best DSC Values in the Test Dataset

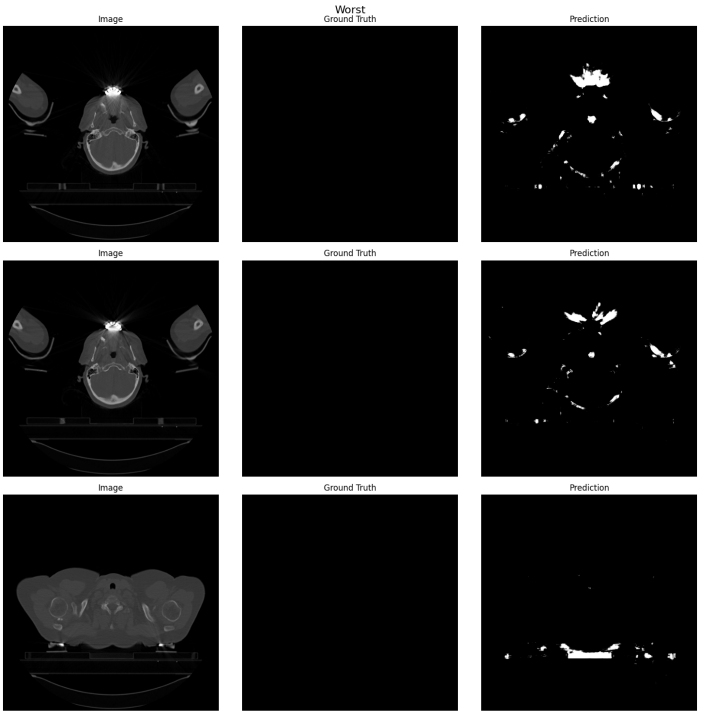


Figure 10: Examples with Worst DSC Values in the Test Dataset

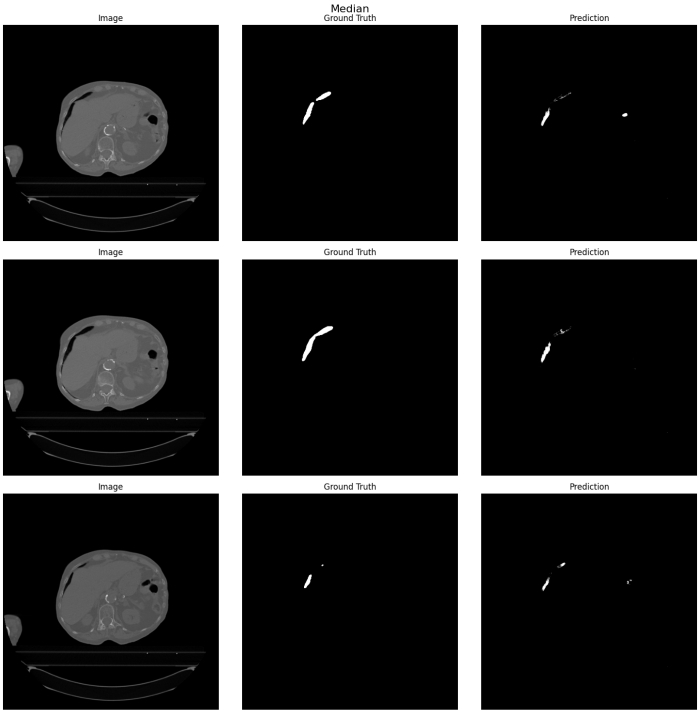


Figure 11: Examples with In-Between DSC Values in the Test Dataset

## 3.3 Summary and Future Development

Carefully checking the Dice Similarity Coefficient (DSC) of our train dataset and test dataset, we find that approximately half of the dataset has a DSC nearly 0, indicating suboptimal performance of our model. Moreover, upon examining examples of 2D slices from different patients, we observe some peculiar patterns:

- Examples with the best DSC scores often exhibit blank true masks and blank predicted masks.
- Conversely, examples with the worst DSC scores tend to have blank true masks but non-blank predicted masks.

These findings are quite unusual and warrant further investigation to understand the underlying reasons behind such discrepancies.

The imperfections in our results stem from several factors, with a significant one being the limited size of our training dataset. With only approximately 1000 images and masks available for training the UNet model, and training for a mere 10 epochs with a batch size of 3, our dataset is notably small. This scarcity of data increases the risk of overfitting, where the model may become too specialized to the training dataset and perform poorly on unseen data, as evidenced by its suboptimal performance on the test dataset.

To address this issue, several solutions can be implemented:

1. **Regularization techniques:** Introducing methods such as L1 or L2 regularization can help prevent overfitting by penalizing complex models that fit the training data too closely.

2. **Data augmentation:** Increasing the diversity of the training samples by applying transformations such as rotation, flipping, scaling, or adding noise can lead to a more robust model.

3. **Acquisition of additional datasets:** Obtaining more data from external sources, such as the Lung CT Segmentation Challenge (LCTSC)[2] dataset, can significantly enrich our training dataset and improve the model's generalization ability.

Moreover, the presence of non-lung regions in the images can contribute to imperfections in the model's predictions, especially when the masks become blank in those regions while the predicted mask does not. Standard chest CT images often contain various non-lung structures such as muscles, the heart, and extraneous blank spaces. Although our UNet-based model is proficient in segmenting lung areas, it may inadvertently overlook inflammatory tissues adhering to the lung wall.

To enhance the model's robustness, we can adopt strategies similar to those described in the paper "A fully automatic deep learning system for COVID-19 diagnostic and prognostic analysis."[3] This involves utilizing the cubic bounding box derived from the segmented lung mask to crop lung areas in CT images, defining this cubic region as the lung-region of interest (ROI). By focusing solely on the lung region, we can ensure the retention of inflammatory tissues while eliminating extraneous areas outside the lungs.

Furthermore, implementing a non-lung area suppression operation, as introduced in the aforementioned paper, can help attenuate the intensities of non-lung areas within the lung-ROI, thereby standardizing the lung-ROI and facilitating subsequent analysis.

In addition to these strategies, exploring alternative deep learning architectures beyond UNet[1], such as those proposed by Khan et al.[4][5] and Chen et al.[6], offers promising avenues for improving lung segmentation accuracy. Each method brings unique strengths and may be suitable for specific applications or datasets, emphasizing the importance of considering diverse methodologies in model development.

1. **Khan et al.[5]**: They developed a deep learning structure for automatic lung nodule identification in CT scans. Their approach involves using the VGG-SegNet implementation for initial segmentation of lung nodules in scaled 2D CT images. Subsequently, they extract crucial features using methods such as Pyramid Histogram of Oriented Gradients, Grey Level Co-Occurrence Matrix, and Local Binary Pattern. These features are then integrated with pre-trained deep learning models' learned characteristics and used for training a classifier.

2. **Chen et al.[6]**: They proposed a dilated convolution-based U-Net framework for precise and effective segmentation of CT images to improve lung cancer treatment effectiveness. Dilated convolution samples the original image at intervals equal to the dilation rate, allowing for larger receptive fields without increasing the number of parameters.

Overall, addressing these challenges and implementing appropriate solutions will lead to more accurate segmentation results, facilitating enhanced diagnostic and prognostic analysis, particularly in the context of medical imaging.

# 4. References


1. Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. Munich: Springer International Publishing. doi: 10.1007/978-3-319-24574-4_28

2. Yang, J., Sharp, G., Veeraraghavan, H., Van Elmpt, W., Dekker, A., Lustberg, T., and Gooding, M. (2017). Data from Lung CT Segmentation Challenge (LCTSC) (Version 3) [Data set]. The Cancer Imaging Archive. https://doi.org/10.7937/K9/TCIA.2017.3R3

3. Wang, S., Zha, Y., Li, W., Wu, Q., Li, X., Niu, M., Wang, M., Qiu, X., Li, H., Yu, H., Gong, W., Bai, Y., Li, L., Zhu, Y., Wang, L., and Tian, J. (2020). A fully automatic deep learning system for COVID-19 diagnostic and prognostic analysis. European Respiratory Journal, 56(2), 2000775. doi: 10.1183/13993003.00775-2020

4. Thanoon, M. A., Zulkifley, M. A., Mohd Zainuri, M. A. A., and Abdani, S. R. (2023). A Review of Deep Learning Techniques for Lung Cancer Screening and Diagnosis Based on CT Images. Diagnostics (Basel), 13(16), 2617. doi: 10.3390/diagnostics13162617

5. Khan, M. A., Rajinikanth, V., Satapathy, S. C., Taniar, D., Mohanty, J. R., Tariq, U., and Damaeviius, R. (2021). VGG19 Network Assisted Joint Segmentation and Classification of Lung Nodules in CT Images. Diagnostics (Basel), 11(12), 2208. doi: 10.3390/diagnostics11122208

6. Chen, K. B., Xuan, Y., Lin, A. J., and Guo, S. H. (2021). Lung computed tomography image segmentation based on U-Net network fused with dilated convolution. Computer Methods and Programs in Biomedicine, 207, 106170. doi: 10.1016/j.cmpb.2021.106170