

MPhil Data Intensive Science
University of Cambridge

Medical Imaging Coursework

UNET Model for Lung Segmentation in CT Scans

Andreas Vrikkis

March 28, 2024

L^AT_EX Word count: 2825 (excluding Contents Page, References and Appendix)

Abstract

In this study, we train a UNET model for lung segmentation on 2-dimensional CT scan images. Our approach utilises a tailored loss function, integrating Binary Cross Entropy with Dice Similarity Scores, and incorporates a DSC Threshold to effectively manage images with predominantly background pixels. While the model achieves Binary Accuracies as high as 0.997% in specific slice locations, it exhibits reduced accuracy in slices with sparse lung tissue. This paper outlines the performance metrics, delves into the model's limitations, and suggests potential avenues for enhancement.

Contents

1	Introduction	3
2	Background	3
2.1	Accuracy & Loss Metrics	3
3	Method	4
3.1	Data Handling	4
3.2	Training	4
3.2.1	Train-Test-split	4
3.2.2	Loss Function	5
3.2.3	Training Loop & Chosen Metrics	5
4	Results & Discussion	5
4.1	Model Performance	5
4.2	Improvements	13
5	Conclusion	13
References		14
Appendix		15

1 Introduction

UNet, developed by Ronneberger et al.[1], is one of the earliest and most popular convolutional neural network architectures for medical image segmentaions. This technique is built on an encoder-decoder structure [2] consisting of a contracting path for downsampling and an expansive path for upsampling. A notable feature is the skip connections that concatenate the feature maps from the contracting path to the corresponding upsampling path. This enhances the detail lost during downsampling. In this project, we train a simple UNET (full architecture in the Appendix) to perform lung segmentation on 2D CT scan images. We start with an overview of the metrics employed for training and evaluating the model, then outline our methodology. Following this, we present the results of the model's performance and conclude with a discussion on the model's limitations and possible enhancements.

2 Background

2.1 Accuracy & Loss Metrics

Binary Cross Entropy (BCE) loss is used to perform pixel-wise binary classification. Each pixel in the image is classified as either part of the target we are segmenting or not. It is defined for a set of pixels x as follows:

$$L_{BCE}(p(x), y) = \frac{1}{N} \sum_{i=1}^N (-y_i \cdot \log(p(x_i)) - (1 - y_i) \cdot \log(1 - p(x_i))) \quad (1)$$

where $p(x_i)$ is the predicted probability of pixel i being part of the target after applying the sigmoid function, y_i is the ground truth label, and N is total number of pixels.

The Dice Similarity Coefficient (DSC) is a measure of the overlap between the areas of the prediction and ground truth. For binary segmentation, it is defined as:

$$DSC = 2 \frac{\sum_{i=1}^N y_i p_i}{\sum_{i=1}^N y_i + \sum_{i=1}^N p_i} \quad (2)$$

where y_i and p_i are the ground truth and predicted pixel values respectively, and N is the total number of pixels. A pixel contributes to the intersection only when both y_i and p_i are 1. DSC ranges from 0 (no overlap) to 1 (perfect overlap). By redefining DSC as a loss function, we get:

$$L_{DSC}(p(x), y) = 1 - 2 \frac{\sum_{i=1}^N y_i p_i}{\sum_{i=1}^N y_i + \sum_{i=1}^N p_i} \quad (3)$$

We can combine the two loss functions using a weight coefficient $\beta \in [0, 1]$ as:

$$L_{total}(p(x), y) = \beta \cdot L_{BCE}(p(x), y) + (1 - \beta) \cdot L_{DSC}(p(x), y) \quad (4)$$

Finally, we consider the Binary Accuracy (BA) metric, which measures the proportion of correctly classified pixels. For a model outputting a probability per pixel for being part of the target, a threshold is set to derive the binary classification. For N pixels, BA is calculated as:

$$BA = \frac{\text{True Positives} + \text{True Negatives}}{N} \quad (5)$$

We discuss the limitations of each loss function and metric in the discussion section.

3 Method

3.1 Data Handling

We briefly cover the data loading process. Each case folder contains a series of DICOM files, which represent a single 2D slice from a 3D scan. We define an image loading function that performs the following:

1. For each DICOM file in a specified directory, read the metadata using the `pydicom` python package [3].
2. Retrieve and store the file name and the ‘SliceLocation’ attribute corresponding to the position of the slice along a reference point.
3. Sort the file names based on ‘SliceLocation’.
4. Use the ordered list to assemble the 3D structure of the scan by stacking the 2D arrays of the slice.

We choose to structure the data as a dictionary keyed by the case numbers (‘Case_000, Case_001,...’) and valued by numpy arrays of image pixels. For example, `case_arrays[‘Case_002’]` [50] would load the 49th slice of case 2. This is done to enable easier access to the data when training the model.

If the ‘SliceLocation’ ordering is omitted, the truth segmentation data does not match the patient images. Since the provided segmentation do not contain metadata, we write a script that plots the image overlayed with the corresponding segmentation data at specified regular intervals. The user can then visually inspect the images and determine whether the segmentation data is aligned with the image data. It was found that the segmentation data was aligned based on the z-axis, from under the lungs up to the head.

We then create a function `create_paired_data` that pairs the input image and segmentation dictionaries based on the case number into a single dictionary. This is useful when dividing the data into training and validation sets. Again this is keyed by the case number, and the values are a tuple of the image 3D numpy array and its corresponding segmentation 3D numpy array. We visually inspect multiple image-segmentation pairs to ensure that the data is loaded correctly. Finally we define a custom PyTorch Dataset class [4] that takes the paired data dictionary as input and returns the image-segmentation pairs as tensors in the correct shape for batching.

3.2 Training

3.2.1 Train-Test-split

To split the data into training and test sets, we randomly shuffle the order of the case keys (case numbers) and take the first $\frac{2}{3}$ (first 8) as training data and the remaining $\frac{1}{3}$ (last 4) as test data. This split is based on case numbers rather than randomly selecting two-thirds of all image slices to prevent data leakage between training and test sets. It’s crucial to keep slices from the same case together as they are likely correlated, ensuring that a patient’s data is not split across both sets. The test set must remain entirely unseen by the model. We fix the seed before choosing the train-test cases to ensure reproducibility.

3.2.2 Loss Function

We implement the combined loss function L_{total} from eq.(4) as a PyTorch class. This loss combination allows the model to learn both pixel-wise classification accuracy (via BCE) and spatial overlap accuracy between predicted and ground truth segmentations (via Dice Loss). For BCE loss, we use `BCEWithLogitsLoss` from `PyTorch.nn`, which combines the sigmoid activation function and the BCE loss and is more numerically stable than applying each separately. For the DSC coefficient, we define a function that applies eq.(2) to post-sigmoid predictions and the ground truth masks. We note that this formulation of DSC loss has an undesirable property: when the true mask is empty and the model predicts even a few pixels as target, $DSC = 0$ as there is no overlapping area. This leads to $L_{DSC} = 1$, which overpenalises the model. Also, this can occur often as a large proportion of truth masks are empty. To address this, we introduce a DSC Threshold. If the sum of predicted pixels is below this threshold, and the ground truth contains no positive pixels, the function considers the prediction as effectively empty and returns a perfect score of 1. The effects of this threshold are discussed in the Results section.

3.2.3 Training Loop & Chosen Metrics

Our custom training loop accepts several inputs: the model, train and test data loaders, learning rate, number of epochs, DSC metric threshold, and the weight β for L_{total} . We opt for the Adam optimizer [5] due to its adaptive learning rate features. Each epoch involves iterating over training data in batches of size 3, with the model set to training mode to activate dropout and batch normalisation. These help in preventing overfitting and enhancing the model’s generalisation to new datasets. We then pass the input image data x through the model to obtain the predicted segmentation masks $p(x)$. We calculate L_{total} and use the optimiser to update the model parameters. We log the Binary Accuracy (BA) from eq(5) using PyTorch’s `torchmetrics.BinaryAccuracy` class. This is done after applying the sigmoid function and setting a threshold of 0.5 for the predicted masks. BA is accumulated across each batch and then averaged at the end of each epoch. Additionally, the average loss for each epoch, denoted as \bar{L}_{total} , is also recorded.

After the training phase in each epoch, the model is set to evaluation mode with `model.eval()`. This disables dropout and batch normalisation, and allows the model to make deterministic predictions. We now use the test DataLoader to loop over the test data in batches and calculate the average BA of the test data per epoch.

The model’s training is performed on a single NVIDIA RTX 3060 GPU with 16GB memory. We conduct the training loop over 10 epochs, setting a learning rate of 0.01 and a β value of 0.7. Initially, $\beta = 0.5$ was used, which resulted in sensible but often fragmented and discontinuous lung tissue masks. We believe this is because the model was biased towards the L_{DSC} which has high penalties, especially when the ground truth mask is mostly empty. Given the dataset’s imbalance with a majority of background pixels, the inconsistency of DSC loss for nearly empty masks led us to increase the BCE loss weight to 0.7. This led to more continuous masks with good spatial overlap between the predicted and ground truth masks. The DSC Threshold is set at $512 \times 512 \times 0.005$ (approximately 0.05% of total pixels), ensuring the model is still penalised for substantial positive pixel predictions against an empty ground truth mask.

4 Results & Discussion

4.1 Model Performance

Fig.1 illustrates the average loss \bar{L}_{total} and accuracy metrics over the 10 training epochs. L_{total} decreases over the epochs, indicating that the model makes better predictions as the training

progresses. At the last 3 epochs there is an increase in the downwards trend compared to the previous 3 steps. This suggest that the model could be trained for more epochs to further improve its performance. However, the average binary accuracy \overline{BA} per epoch plotted on the right of Fig.1 reveals that the model’s predictions on the test data starts to deteriorate after the 7th epoch. This indicates that the model is starting to overfit the training data, since on the same region the mean train BA is still improving. This could be further investigated by training the model for more epochs, and repeating the training to confirm this effect. Strategies such as early stopping, increased dropout, and enhanced batch normalisation could also be employed to mitigate overfitting.

While the \overline{BA} values approach the ideal score of 1, this metric has limitations due to the segmentation pixels of interest being a minority in the total pixel count. The accuracy score dominated by the background pixel classifications. Furthermore, BA, as a measure of correctly classified pixel proportion, aligns more with the BCE loss function than the DSC loss, which focuses on area overlap. Logging the mean DSC per epoch would be beneficial, providing insights into model performance relative to the DSC loss. This additional metric would reveal whether the model is improving in terms of spatial overlap between the predicted and ground truth masks.

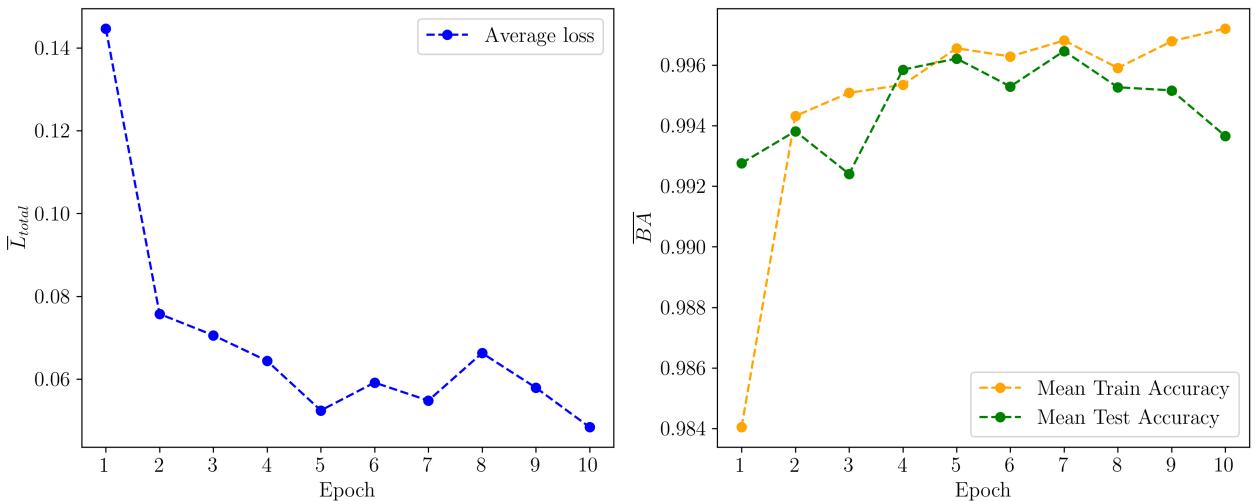


Figure 1: Training and test metrics over 10 epochs. The left plots shows the average combined loss \overline{L}_{total} over each epoch. A downwards trend indicates that the model is making better predictions as the training progresses, but is not necessarily indicative of the model’s performance on the test data. The right plot shows the average binary accuracy \overline{BA} over each epoch, the green line represents the BA on the training data and the blue line represents the BA on the test data. While the \overline{BA} on the training data continues to improve, the \overline{BA} on the test data starts to decrease after the 7th epoch. This suggests that the model is starting to overfit the training data.

Fig.2 depicts the frequency histograms of the DSC scores between the predicted masks and the ground truth. For both the train and test slices, the majority have a DSC in the range of 0.95 to 1. The train slices have higher proportion of DSC scores in the range of 0.95-1.00 compared to the test slices, as expected. This might also suggest that the model is overfitting the training data. However, the substantial peak at 1.00 is partly an artefact of the implemented DSC Threshold, which elevates many predictions to 1. To assess this threshold’s impact, we also present histograms with a reduced threshold of 20 pixels (approximately 0.007% of the image pixels) in Fig.3. Here, several slices initially scoring 1.00 drop to 0.00 – these are instances where the predicted mask exceeds 20 pixels, yet the ground truth mask is empty. This illustrates the limitations of the DSC metric, especially for nearly-empty ground truth slices.

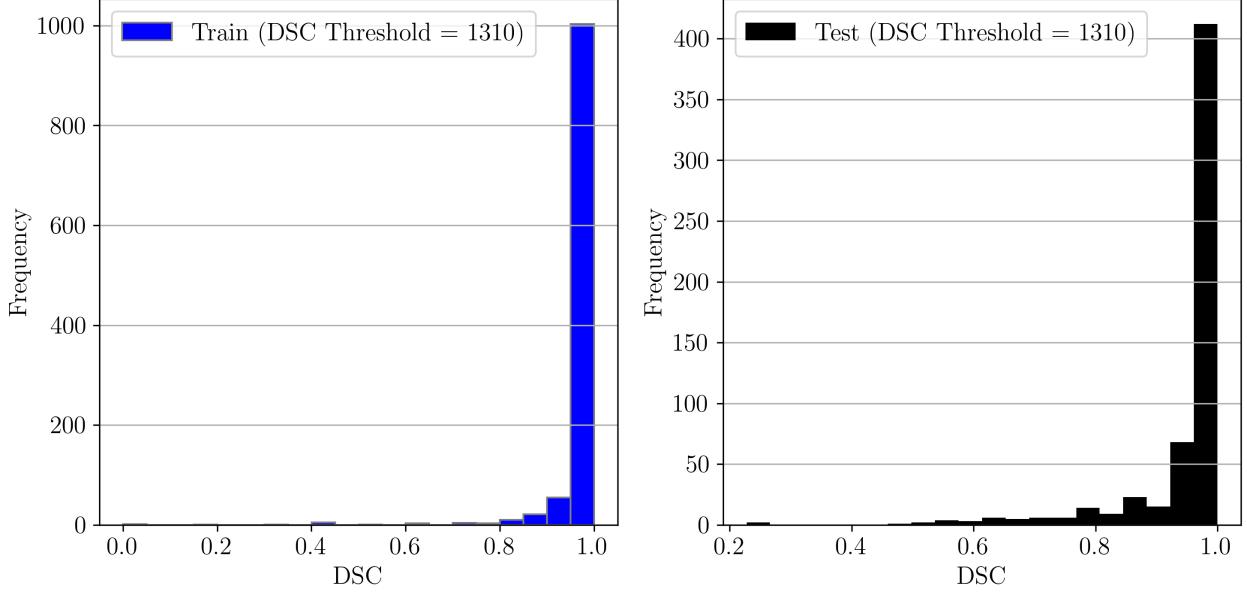


Figure 2: Histogram of the DSC scores between the predicted masks and the provided ground truth masks for a DSC Threshold of 1310 pixels. The left plot shows the DSC histogram scores for the training set, while the plot on the right shows the DSC scores for the test set. As expected, more predictions occupy the 0.8-0.95 range for the training set than the test set, as DSC loss was included in the model's loss function L_{total} . A proportion of the peak is caused by the DSC Threshold itself.

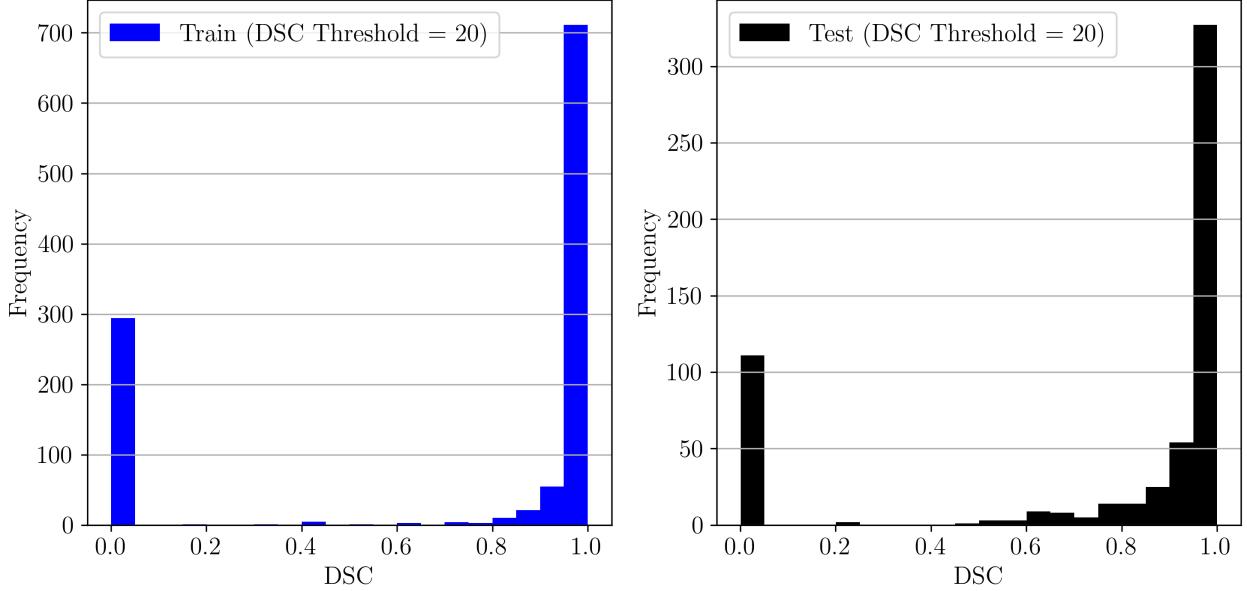


Figure 3: Histogram of the DSC scores between the predicted masks and the provided ground truth masks for a DSC Threshold of 20 pixels. The artificial peak at 1 is now diminished, with a proportion moving to the 0. The test set has a higher variability in DSC scores than the train set which has most them concentrated around the 0.8-1.0 range.

Fig.4 shows the DSC score and BA score for each slice of Case_001. This plot highlights the limitations of both metrics. The range of BA scores is very narrow, limited between 0.97-1.00. This is because the majority of the pixels are part of the background and hence the background class dominates the BA score. It captures a high amount of agreement in the beginning and the end of the slices, which are the slices that contain little to no lung tissue. Conversely, the DSC

score is problematic in that same region. It fluctuates rapidly from 0.00 to 1.00, particularly in the slices towards the end of the series (progressing from above the lungs to the head). This is due to the presence of empty turb masks in these slices, rendering the DSC score less informative. Nevertheless, it effectively indicates high agreement in the middle slices where lung tissue is present.

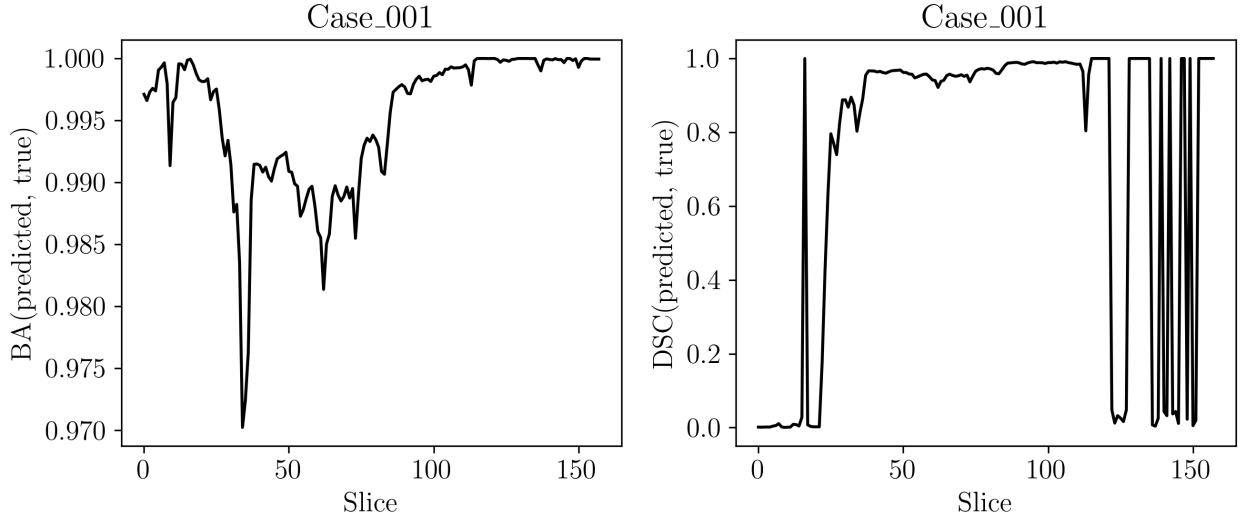


Figure 4: Scatter plots of DSC scores (left) and BA scores (right) for each slice of Case_001. These are in order of the ‘SliceLocation’ metadata, physically starting from below the lungs up to the head.

The final plots depict examples of 2D slices in patients of cases [001,003,004,011]. Cases[001,003] were part of the training set and cases[004,011] were part of the test set. Fig.5, Fig.6, and Fig.7 display slices exhibiting the best, median, and worst DSC scores, respectively. Slices with a perfect DSC score of 1.00 were excluded from the ‘best’ category due to their empty ground truth masks and predictions below the DSC Threshold. The selected best slices all have DSC above 0.990, with predicted masks very close to true masks, and with continuous lung segmentations. These slices are typically located around the centre of the 3D scans, where the lung tissue takes its largest proportion of the image. In all these cases, the model is able to differentiate between the lung and the trachea, and the segmentations are mostly continuous in both train and test cases. We can already observe that the model is better at identifying the lung tissue when they are symmetrically present in the image and bordered by clear edges.

In the median-performance slices, we start to see a differentiation between the train and test cases. The train cases have a DSC score of 0.95-0.98, while the test cases have a DSC score of around 0.870. These slices are all located below the midpoint of the 3D scans. Case_011 in particular has a fragmented prediction, and in both Case_011 and Case_004 the model struggles to accurately identify narrower regions of lung tissue.

In it unsurprising that the lowest DSC scores are in mostly empty truth masks. These slices are located well before the lungs appear. The fact that in Case_003 the model predicts lung tissue in the air between the arms and the head might be a sign of overfitting. Moreover, the worst slices share a common feature: all have ground truth masks that predict a single lung tissue, while the model always predicts two or more. This suggests that the model might be overemphasising the presence of paired lung tissues, which is typical in most slices. It appears that the model is sensitive to the double change in pixel value gradient that occurs within the bounds of the body cross section. This limitation becomes apparent in scenarios where only one lung tissue is present or completely absent. The model tends to inaccurately predict a dual change in the pixel value gradient in such instances, as observed in regions like the arms and

head. This pattern suggests that the Convolutional Network might be performing an operation similar to a Sobel filter on the images, followed by thresholding. This method of edge detection is a common characteristic in such networks.

The results of this model are generally successful in a rather narrow, middle range of the 3D scan. The model is able to identify the lung tissue when it is mostly symmetrically present in the image and have clear edges. It is not able to identify the lung tissue when it is only present in one side, and often predicts lung tissue where the truth is empty. It also shows signs of overfitting. In an application where the model's accuracy is crucial such as early tumour detection, this could be a major limitation, and the model would not be acceptable.

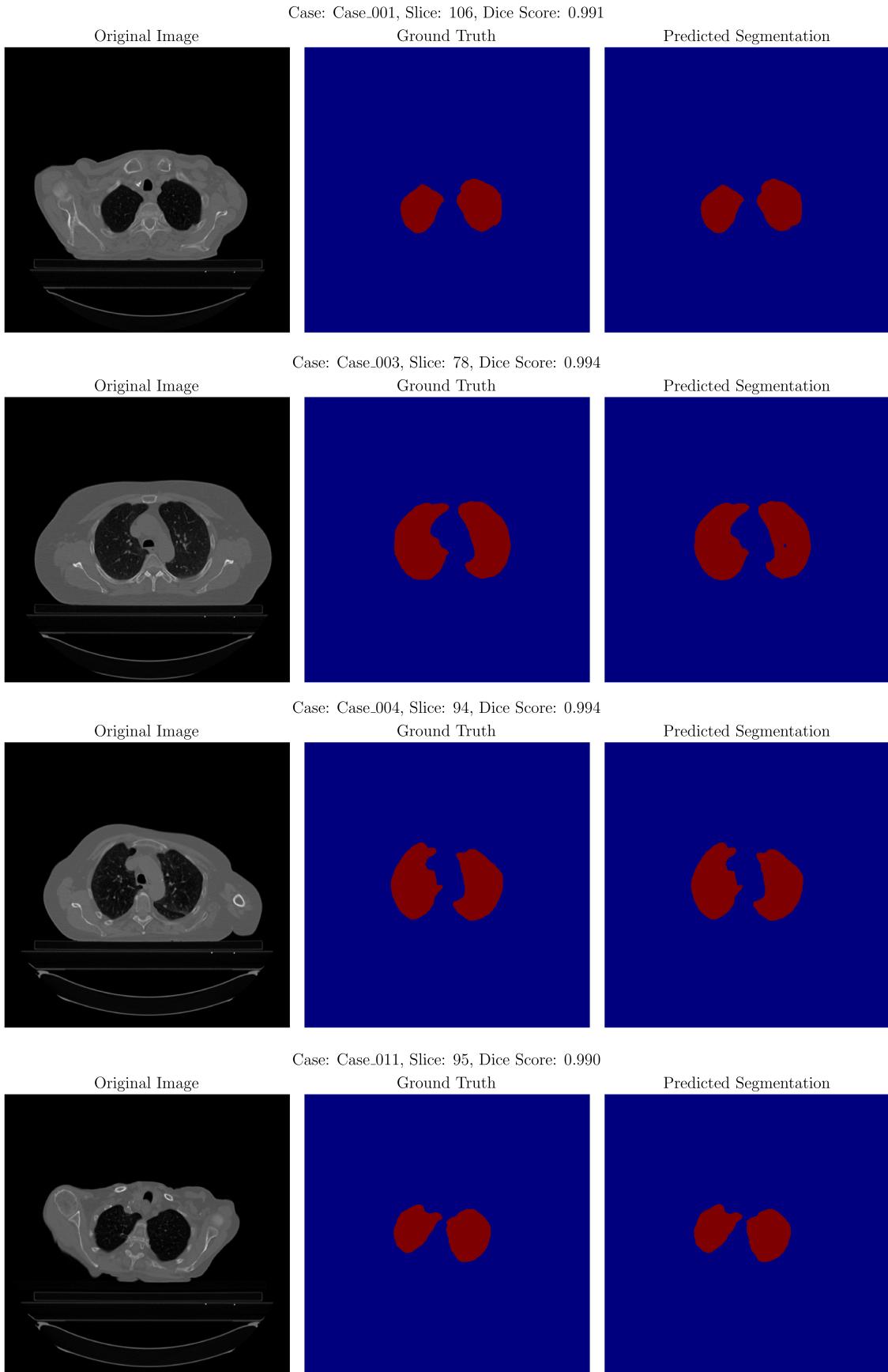


Figure 5: Best slices for cases [001,003,004,011]. These achieved the highest DSC scores, ommiting the slices with $DSC=1$ as they represent the slices with empty ground truth masks with predictions below the DSC Threshold. For each case we plot the original image, ground truth mask, and the predicted segmentation from our model.

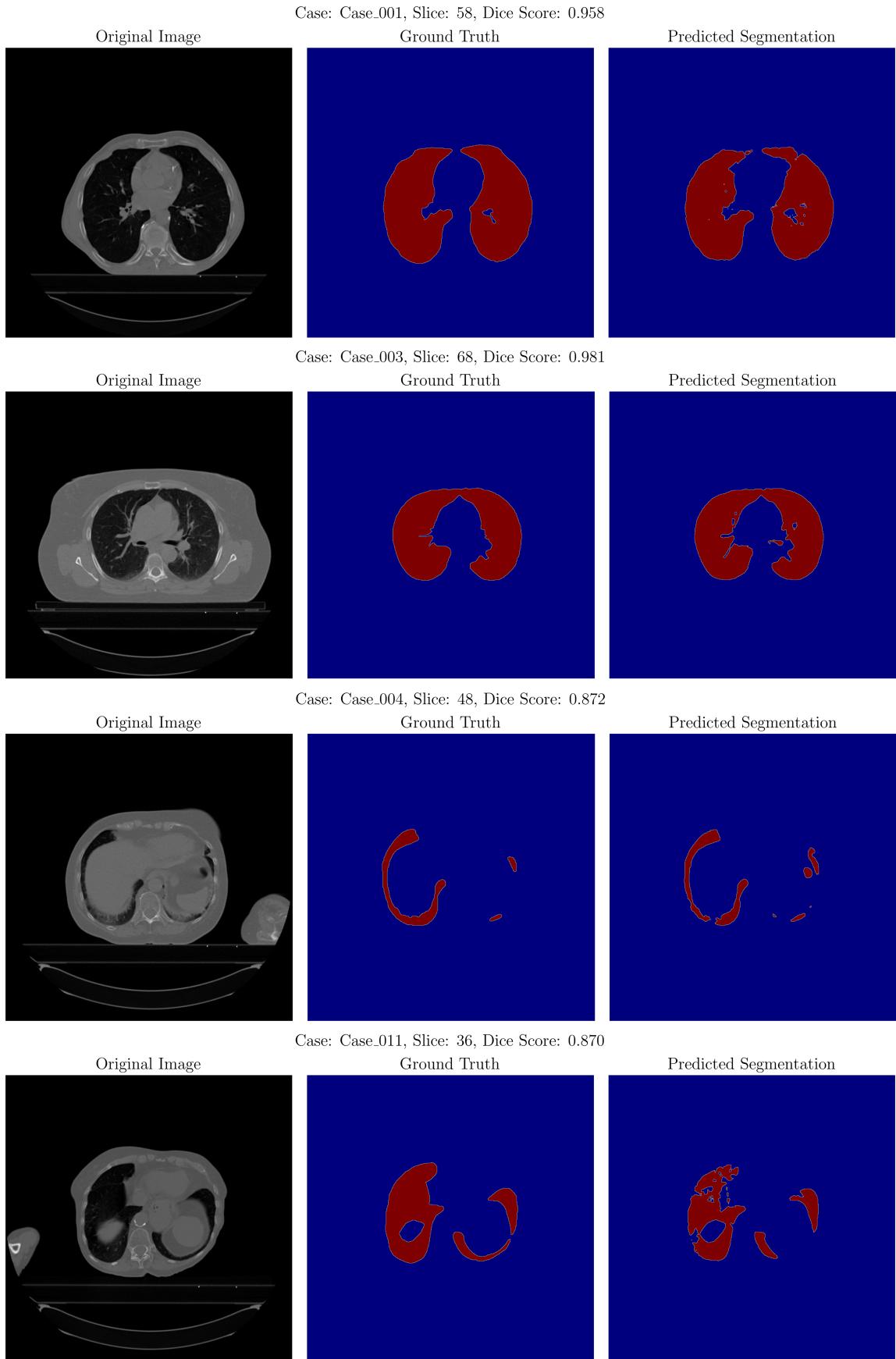


Figure 6: Slices with in-between DSC scores for cases [001,003,004,011]. For each case we plot the original image, ground truth mask, and the predicted segmentation from our model.

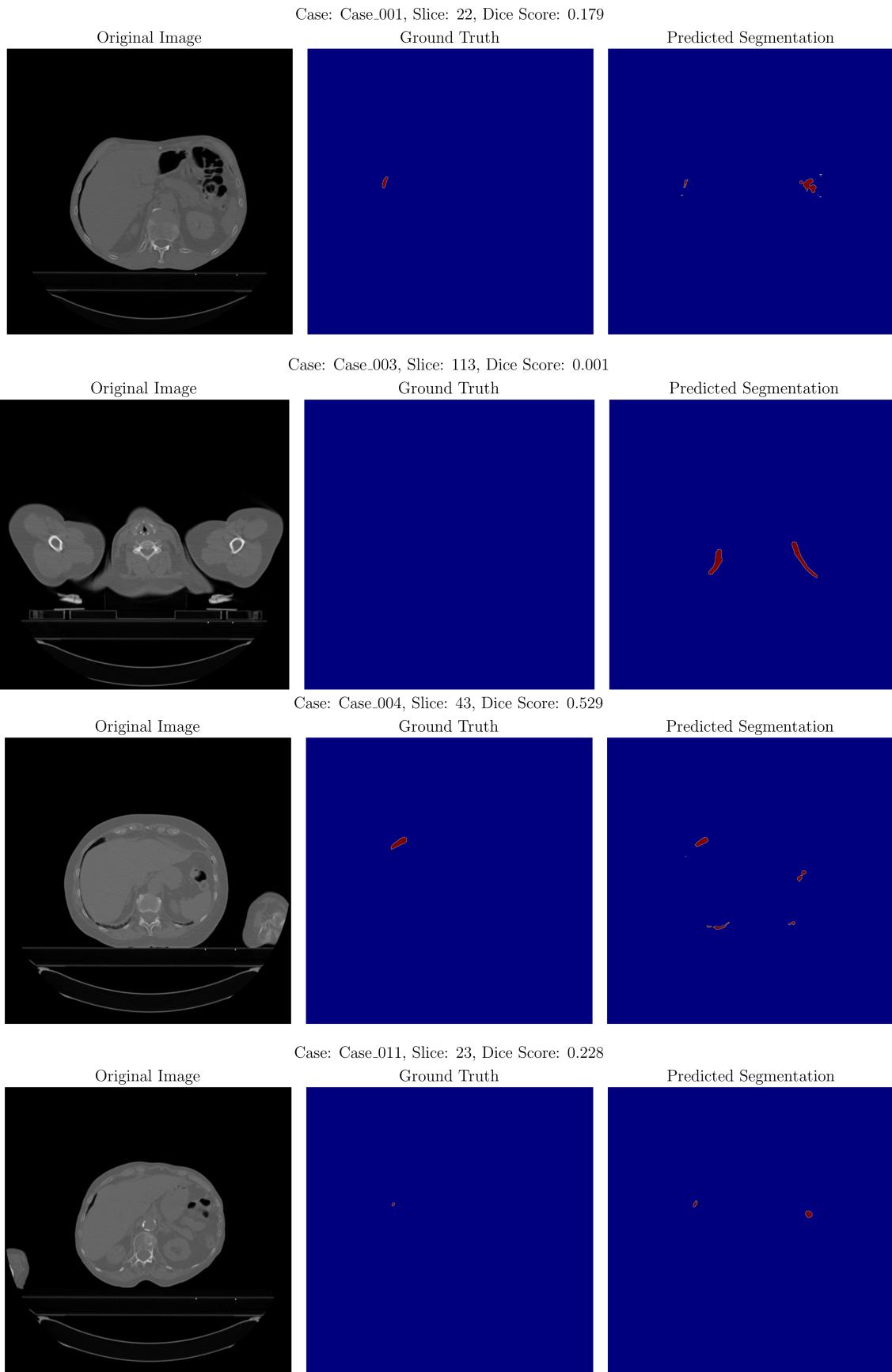


Figure 7: Slices with the worst DSC score slices for cases [001,003,004,011].For each case we plot the original image, ground truth mask, and the predicted segmentation from our model.

4.2 Improvements

To improve this model, we propose the following strategies:

- Modify the Loss function to give greater weight to the BCE score in cases with mostly empty truth masks, and increase the emphasis on the DSC score for fuller masks. This adjustment could mitigate the individual limitations of these two metrics.
- The model could be trained on a larger dataset or by applying data augmentation techniques. This could include random rotations, translations, and scaling of the image data. This could help the model to learn more general features of the lung tissue, and avoid the horizontal symmetry bias.
- If access to additional data is limited, fine-tuning the model’s hyperparameters could potentially enhance its performance. Modifications could be made to the learning rate, epoch count, β value, DSC Threshold, and even the choice of optimiser. Implementing early stopping, along with increased dropout and batch normalisation, might address issues of overfitting.
- During training, a validation set could be used to optimise the model’s hyperparameters. This can also allow a learning rate scheduler that reduces the learning rate when the validation loss stops improving.

5 Conclusion

In this project, we trained a simple UNET architecture to perform lung segmentation on 2D CT images. We used a custom loss function of weighted Binary Cross Entropy loss and Dice Similarity Score to ensure the model emphasizes both on individual pixel accuracy and spatial overlap of the prediction and ground truth mask provided. We introduced a DSC Threshold metric to address scenarios with predominantly empty ground truth masks, which significantly influences segmentation accuracy. A discussion into the limitations of both accuracy metrics is presented, along with an evaluation on the performance of our trained UNET. Overall, the UNET performs well in slices that have a significant proportion of lung tissue. The predictions deteriorate in quality and accuracy for slices with little to no lung tissue. The model also fails in asymmetric slides, and displays signs of overtraining. Future improvements could involve defining an adaptable loss function that adjusts based on the proportion of lung tissue in truth masks, minimising the limitations of both metrics.

References

- [1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, May 2015. arXiv:1505.04597 [cs].
- [2] Huimin Huang, Lanfen Lin, Ruofeng Tong, Hongjie Hu, Qiaowei Zhang, Yutaro Iwamoto, Xianhua Han, Yen-Wei Chen, and Jian Wu. UNet 3+: A Full-Scale Connected UNet for Medical Image Segmentation. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1055–1059, May 2020. ISSN: 2379-190X.
- [3] D. L. Mason et al. pydicom: An open source DICOM library. <https://github.com/pydicom/pydicom>. [Online; accessed 10/03/2024].
- [4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
- [5] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. arXiv:1412.6980 [cs].

Appendix

UNet Model Architecture Used

Fig.8 shows the architecture of the UNet model used in this project. This is provided by the ‘Data Science in Medical Imaging’ course, slides by Dr Lorena Escudero Sanchez, at the University of Cambridge.

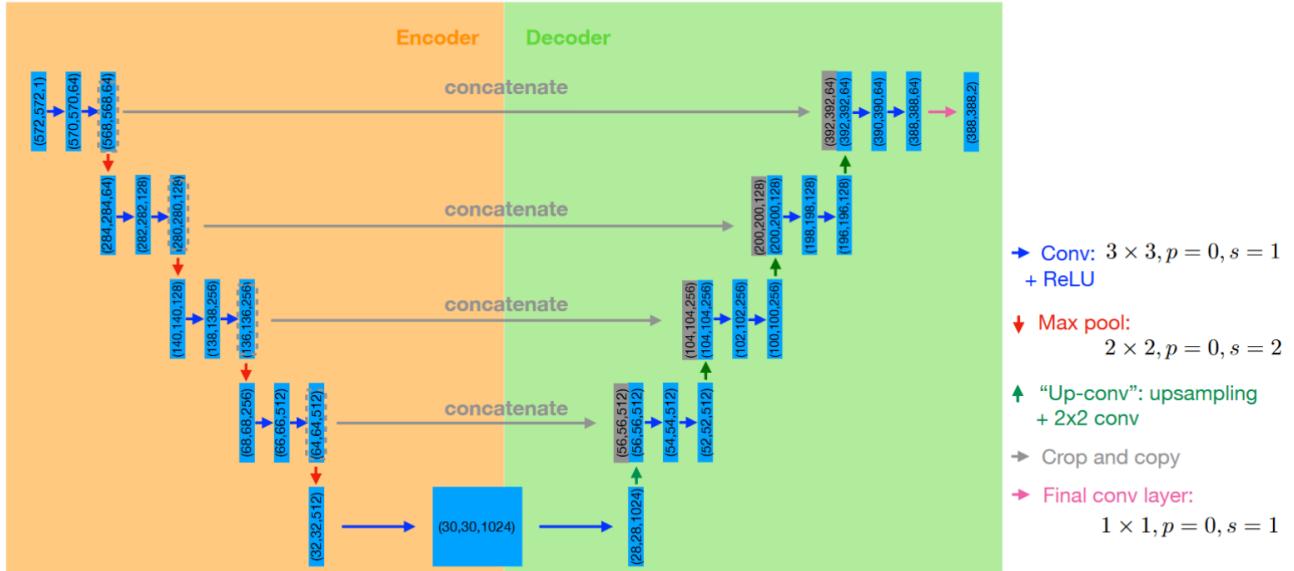


Figure 8: The architecture of the UNet model used in this project.

Autogeneration Tools

Use of Co-pilot

Co-pilot was utilised in the following ways:

- Autocompleting documentation for functions and for scripts.
- Understanding error messages with the ‘Explain this error’ button next to the error message.
- Responding to general queries, for example:

Prompt: How can I change a function name and have it change throughout my workspace?

Answer: In Visual Studio Code, you can use the “Rename Symbol” feature to change a function name throughout your workspace. Here are the steps:

1. Place your cursor on the function name you want to change.
2. Right-click the function name, and select “Rename Symbol” from the context menu. Alternatively, you can use the shortcut F2.
3. Enter the new function name and press Enter.

VS Code will automatically find and update all occurrences of the function name in your workspace. This includes not only the function definition but also all places where the function is called.