Vision transformers for segmenting organs and tissues in CT scans of arbitrary imaging ranges

Petter Melander, Morris Thånell

Master's thesis
2024:E46

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Computer Vision and Machine Learning

Centrum Scientiarum Mathematicarum

Vision transformers for segmenting organs and tissues in CT scans of arbitrary imaging ranges

Petter Melander
melanderpetter@gmail.com

Morris Thånell
morris.thanell@gmail.com

June 13, 2024

Master's thesis work carried out at Exini Diagnostics AB.

Supervisors: Ida Arvidsson, ida.arvidsson@math.lth.se
Filip Winzell, filip.winzell@math.lth.se
Johan Brynolfsson, johan.brynolfsson@exini.com
Hannicka Sahlstedt, hannicka.sahlstedt@exini.com

Examiner: Kalle Åström, kalle@maths.lth.se

Abstract

Automatic segmentation of organs and tissues in computed tomography (CT) images can aid clinicians in anatomical contextualization for planning surgery or dosimetry. CT scans can cover varying axial ranges of the body. This thesis aims to develop a neural network based on vision transformers for segmenting organs and tissues in CT images of arbitrary axial ranges. Two models are presented, one based on Swin UNETR, and a new model that uses axial slices for its patch embedding.

It is difficult to segment each rib and vertebra semantically in limited imaging ranges, and therefore, instance segmentation was implemented for those classes. Both models were trained to perform semantic and instance segmentation simultaneously. Sliding window inference was used to segment arbitrary axial ranges, and methods for ensuring continuity of the instances were developed. The instance segmentation was implemented in two ways, one using a discriminative loss function and one using connected component labeling.

The models presented can perform both semantic and instance segmentation simultaneously with a simple approach. Both models performed well on semantic segmentation of all organs except the ribs and vertebrae, with Dice scores above 0.8 for most organs, and our best model achieved a score of 0.847 on test data, averaged across all organs. Instance segmentation of ribs and vertebrae through discriminative loss worked well, with accurate segmentations and few false positives and false negatives. Separating ribs into instances through the use of connected component labeling gave even better results. Overall, the Swin-based model performed better than the slice model.

Keywords: Semantic segmentation, Instance segmentation, CT, Vision transformer

Acknowledgements

We would like to thank our supervisors at LTH, Ida Arvidsson and Filip Winzell, for their guidance and for always taking the time to answer our questions and discuss things, large or small. We would also like to thank our supervisors at Exini, Johan Brynolfsson and Hannicka Sahlstedt, for always being available to discuss problems, offer guidance and advice, and always being there when we needed help. Furthermore, we would like to thank Jens Richter at Exini for taking an interest in our work and offering advice regarding overarching strategy and goals. Finally, we would like to thank all the people at Exini for making us feel welcome at the office, taking an interest in our work, and making these past months even more fun.

Contents

Abbreviations and Notation

Adam - Adaptive moment estimation

AdamW - Adam with decoupled weight decay

CNN - Convolutional Neural Network

CT - Computed Tomography

FN - False Negative

FP - False Positive

GELU - Gaussian Error Linear Unit

HU - Hounsfield Unit

IoU - Intersection over Union

LN - Layer Norm

LReLU - Leaky Rectified Linear Unit

MLP - Multilayer Perceptron

NLP - Natural Language Processing

PCA - Principal Component Analysis

PET - Positron Emission Tomography

PQ - Panoptic Quality

ReLU - Recitified Linear Unit

ROI - Region Of Interest

RQ - Recognition Quality

SD - Soft Dice

mSD - Mean Soft Dice

SGD - Stochastic Gradient Descent

SQ - Segmentation Quality

Swin - Shifted Window

Swin UNETR - Swin UNEt TRansformers

TN - True Negative

TP - True Positive

UNETR - UNEt TRansformers

ViT - Vision Transformer

1 Introduction

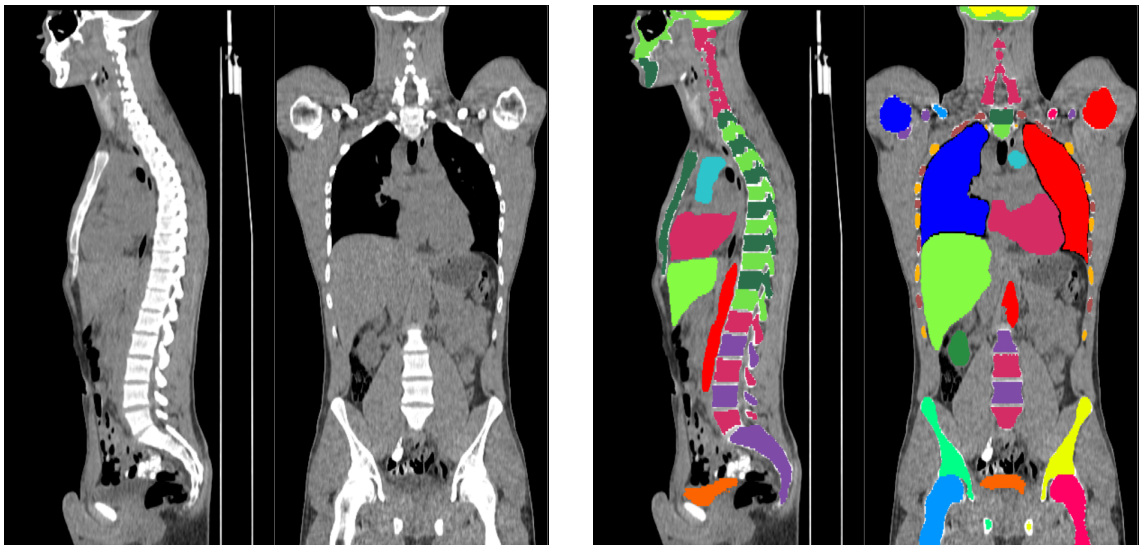
Automatic segmentation of organs and tissues in computed tomography (CT) images can aid clinicians in anatomical contextualization for planning surgery or dosimetry. Deep neural networks can be trained to perform segmentation of organs and tissues in CT images [1]. An example of a CT image and a manual segmentation of certain organs and tissues are shown in Figure 1.

CT examinations can cover various anatomical imaging ranges along the body's vertical axis. These ranges often cover the base of the skull to mid-thigh or the whole body. However, in certain applications, smaller ranges can give sufficient information without exposing the patient to an unnecessarily high dose of radiation [2]. For example, a patient with lung cancer might get a CT scan covering only the thorax and upper abdomen [2, 3, 4]. It is therefore of interest to develop a deep learning model that can accurately segment organs and tissues in CT images of arbitrary axial ranges of the human body.

1.1 Aim and approach

The aim of this thesis was to develop a model for automatic segmentation of organs and tissues in arbitrary fields of view of CT images. To do this, we used vision transformers, a class of neural networks for image analysis that incorporate attention mechanisms, allowing the networks to use global context across the entire input image when assigning each voxel of the image to a class.

We have used Swin UNEt TRansformers (Swin UNETR) [5] as well as a new model of our own design that calculates attention between all axial slices of the input image. To allow the model to learn to segment arbitrary imaging ranges, we used a training protocol where we randomly sample thin axial sections of the CT image (64 slices, or 19.2 cm with the resolution of our data) and feed them into the network during training, and use sliding window inference to segment larger imaging ranges. Finally, we implemented instance segmentation for the ribs and vertebrae, as many of these are almost identical and, therefore, difficult to correctly number when only a few are visible in the input image. This allows our models to segment CT scans of arbitrary imaging ranges accurately.



(a) CT image. **(b)** Segmentation.

Figure 1: An example of a sagittal and a coronal view of CT image and corresponding manual semantic segmentation.

2 Background

In Section 2.1, we begin by describing what segmentation of images is. Section 2.2 describes CT images and their characteristics. Because the anatomy of the spine is featured heavily throughout this thesis we describe the spine and its structure in Section 2.3. Section 2.4 gives a brief overview of deep neural networks and their use in image segmentation. Finally, Section 2.5 is dedicated to describing a few neural network architectures that the models developed in this thesis are inspired by or built upon.

2.1 Segmentation of images

Segmentation of images is a technique where objects or classes of objects in an image are masked, each pixel or voxel being assigned to a certain class or instance. In CT images, classes could for example be liver, heart, or background. The task of semantic segmentation is to assign a class to each pixel in an image. If multiple objects of the same class are present in an image, they are not distinguished as different instances of that class. For example, if segmenting people in an image, all people would be assigned the same class, and those pixels would all share the same mask [6]. Instance segmentation is the task of separating the different instances of a class, giving them separate masks. In the example, each person would be segmented as separate instances in the segmentation. Instance segmentation can segment multiple classes and separates each class into instances [7].

2.2 Computed tomography

Computed tomography (CT) is an imaging technique often used in medical imaging. An X-ray source sends X-rays through all points along an axial slice of the body from a certain angle, and the attenuation is measured on the opposite side. Because X-ray attenuation is roughly proportional to density, this generates a density map of that slice of the body from that angle. Many of these density maps are measured from different angles, and a tomographic reconstruction algorithm is used to generate a cross-sectional axial slice of the body from the density maps. Multiple slices can then be combined to make a 3D view of the body. Because X-rays can be harmful to patients, it is desirable to minimize the radiation dose given to the patient [8, 9]. However, a lower dose generally results in poor image quality, low sharpness, and high noise [9].

CT images are measured in Hounsfield units (HU), which are measurements of X-ray attenuation. The Hounsfield scale is defined such that, at standard temperature and pressure, air has a value of -1000 HU, and distilled water has 0 HU. Soft tissues in the human body

typically have values between -100 and 100 HU, while in the upper limit, bones can reach 1000 HU or even 2000 HU for very dense bones [10, 11]. Metals, which are used in many implants, can take values of over 3000 HU. CT systems often save HU values as 12-bit integers, giving 4096 values that are then mapped to the range -1024 to 3071 HU, a range that covers most tissues in the human body. Some newer CT systems store HU values using 16 bits, giving a much wider range of values. This can be advantageous as scans of lungs can give values below -1024 HU, and metal implants can have values higher than 3071 [11].

2.3 The spine

The vertebral column, or spine, is the central axis of the human skeleton. In humans, the vertebral column usually consists of a total of 33 vertebrae, divided into five groups: cervical, thoracic, lumbar, sacral, and coccygeal, as seen in Figure 2. The vertebrae are connected by ligaments and intervertebral discs. Each vertebra consists of two main parts, a ventral body and a dorsal vertebral arch, the vertebral arch being posterior to the ventral body. The vertebral arch has three protruding processes. An illustration of the shape of vertebrae can be seen in Figure 3. Thoracic vertebrae all have a pair of ribs connected to them, while the other vertebrae do not [12]. In this work, a greater focus was put on the thoracic and lumbar vertebrae. Normally, a spine contains 12 thoracic vertebrae and five lumbar vertebrae. However, some people can have fewer or more than 12 thoracic vertebrae, and some people can have a sixth lumbar vertebra [13, 14, 15]. This sixth lumbar vertebra is usually a result of the lumbarization of the first sacral vertebra, where the topmost vertebra of the sacrum has partially or completely separated from the sacrum, which normally consists of five fused vertebrae [16].

2.4 Deep neural networks for image segmentation

A deep neural network is a parametric machine learning model comprising multiple layers, processing inputs to generate outputs. Each layer is a mathematical function that transforms its input data, and the output from one layer is passed as input to another. The layers do not have to be ordered linearly. For example, the output of a layer can serve as input to several layers, or a layer could get inputs from many layers. Layers can even take their previous outputs as their next inputs, creating a recursive neural network. A deep neural network is called deep because it has many layers that produce its final output. The layers are made up of adjustable parameters, also called weights, that affect the output of that layer. The ability to adjust the parameters allows a network to be trained to perform certain tasks.

2.4.1 Training neural networks

A network is trained by iteratively changing the parameters to bring its output closer to the desired output. In this work, the networks are trained with supervised learning, meaning that the data used as input has been labeled with its corresponding desired output in advance. The

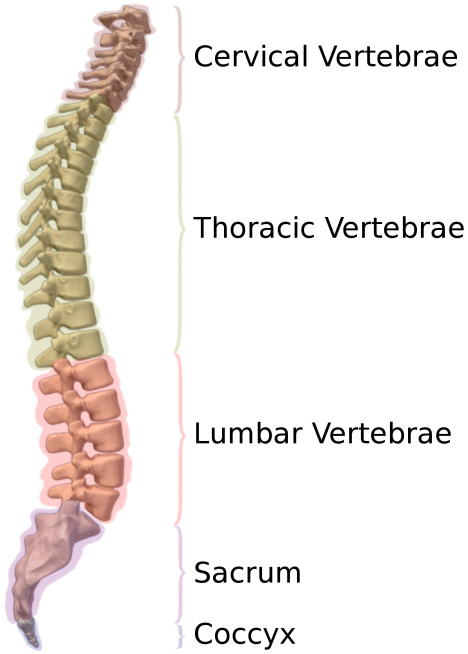


Figure 2: The human spine, showing the division of vertebrae into five sections. *Image credit: Wikipedia user DrJanaOfficial. Used under CC BY-SA 4.0 [17].*

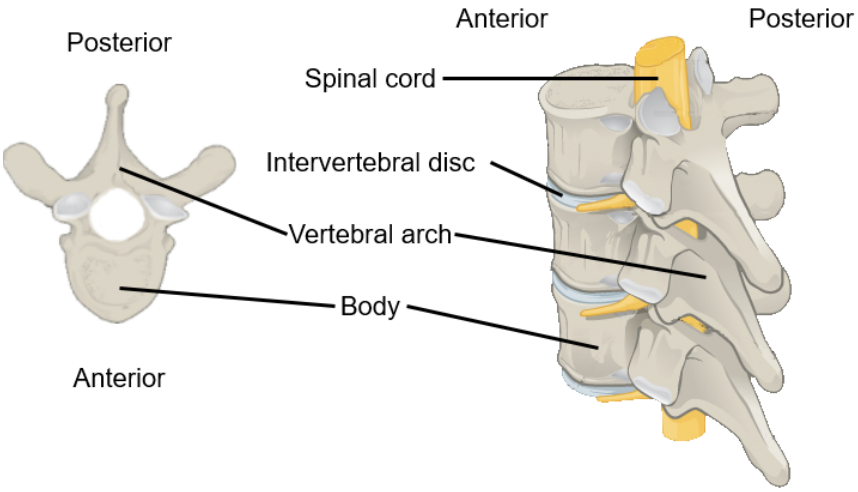


Figure 3: Illustration showing the shape of vertebrae. *Image credit: Wikipedia user Jmarchn. Used under CC BY-SA 3.0 [18].*

labels are also called ground truth. A loss function is used to determine how close an output is to the ground truth, and the network is trained by iteratively adjusting its parameters to minimize the loss.

To adjust the parameters such that the loss is reduced, the gradient of the loss function with respect to the parameters is calculated. The gradient is calculated using backpropagation, which utilizes the chain rule to reuse the partial derivatives. A function called an optimizer decides how much the parameter values will be changed in the direction of the negative gradients for each iteration. To reduce memory usage and the computational cost of calculating the gradients, the data can be randomly split into batches, and the optimizer takes steps based on the gradient calculated only with the data points of one batch at a time. Each step is thus done with a new batch containing new data points. Using batches affects the optimization since the optimization steps are based on a subset instead of the entire training data set. This means that each step will not be optimal for the entire dataset, however more optimization steps will be taken for the same amount of datapoints. This optimization method is called stochastic gradient descent (SGD). When all training data points have been used to update the parameters, one epoch has passed. By training the network for many epochs, the network's performance is expected to improve and produce outputs closer to the ground truth [19].

2.4.2 Overfitting

To be useful, a neural network trained with supervised learning should not only learn to solve its task on the training data but also generalize, meaning it should be able to perform well on new data which it was not trained on. If the model is too complex and flexible, it could lead to the model learning details or noise specific to the training data, thus generalizing poorly. The same could be true if there is too little data to train on. Several regularization methods exist that prevent overfitting while maintaining model complexity [20]. In this work, we used dropout and data augmentation to improve our models' generalization.

Dropout is a method used during training where some parameters in the model have a certain probability of not being used when calculating the output [19]. The parameters that are not used are not updated when adjusting the network's parameters. The parameters that are dropped are random and change for every point of training data. Dropout forces the network to be able to make its predictions with only a subset of its parameters. Dropout is only used during training, and when the model is used for inference, the entire network is used. When used for inference on new data, all parameters are multiplied by their probability of being kept, to compensate that dropout was used. Using all parameters is similar to an average of all networks with dropout used during training, which decreases the risk of overfitting the model to the training data.

Data augmentation is a way of increasing the amount of training data by applying transforms that preserve key features and properties of the data while still altering it in some way [19]. For images, such transformations could, for example, include flipping, cropping, or rotating the image. With more training data, the network is trained to perform well on a more diverse set of images and should thus generalize better.

2.4.3 Convolutional neural network

A convolutional neural network (CNN), described in [20], is a type of neural network frequently used on image data, that produces its output using convolutions. The values in the convolutional filters, or kernels, are learnable model parameters. In this work, 3D images were used as input, and thus the kernels used were also 3D. Using the same kernel and parameters for all pixels is called parameter sharing, meaning the network looks for certain features all over the image. The output of one convolution is often called a feature map.

A CNN is structured in layers where each convolutional layer finds features in the output of the previous layer, meaning that the CNN finds more complex features in deeper layers. The kernels can be multidimensional, having many channels, allowing the kernels to find multiple different features in each layer. Each layer often consists of three parts: the convolutions, an activation function, and lastly, downsampling. The downsampling can be performed by having a stride length of the convolutions greater than one. This means that not all pixels will have a corresponding pixel in the output, making the output smaller. Another common method of downsampling the output is pooling layers, common ones being max- and average pooling. They replace the output of the layer with the maximum or average value of a part of the image, thereby downsampling it. Because the image is downsampled in deeper layers, a kernel of the same size in a deep layer covers a larger region of the image than one in a shallow layer, allowing it to find larger and more complex features. Transposed convolutions can be used to produce larger outputs than the input, allowing for upsampling. It works by first padding the input image with zeros around the image and/or between all rows and columns, and then by using a stride of one, the output feature size will be larger than the input size.

2.4.4 Activation functions

Activation functions are essential parts of neural networks, providing necessary non-linear properties to enable the network to learn complex representations [21]. For example, two matrix multiplications in a row without a non-linear activation function in between can be written as a single matrix multiplication, meaning there is no advantage in using several layers of matrix multiplication. A non-linear activation function between the two matrix multiplications enables the network to utilize both parameter matrices to create a richer feature representation [20].

Different activation functions can impact a network's capacity to learn in general and the speed at which it learns in particular. Many different activation functions have been proposed, often based on the properties of their gradients [21]. In this project, three different activation functions have been used in many places: rectified linear unit (ReLU) [22], leaky ReLU (LReLU) [22], and Gaussian error linear unit (GELU) [23]. These functions are defined as

$$\begin{aligned} \mathrm{ReLU}(x) &= \max(0, x), \\ \mathrm{LReLU}(x) &= \begin{cases} x & \text{if } x > 0, \\ \alpha x & \text{otherwise.} \end{cases}, \\ \mathrm{GELU}(x) &= x\Phi(x), \end{aligned}$$

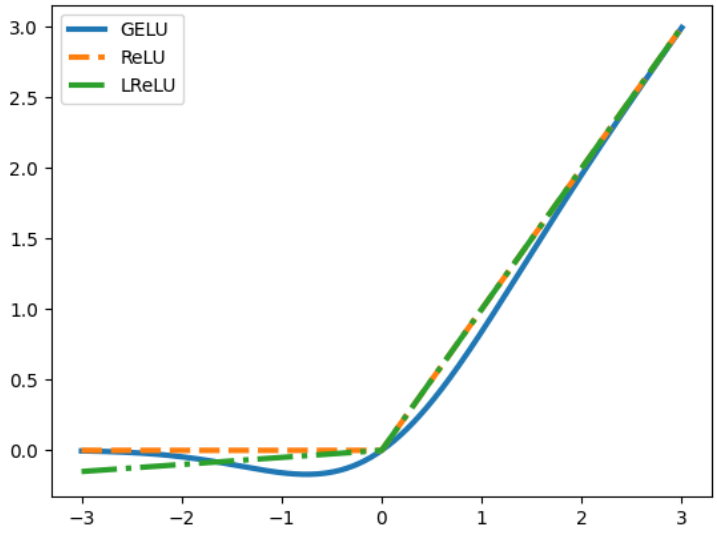


Figure 4: GELU, ReLU, and LReLU with $\alpha = 0.05$.

where α is some constant with $\alpha << 1$ and $\Phi(x)$ is the cumulative distribution function of the standard normal distribution. These functions are plotted in the range $[-3, 3]$ in Figure 4.

2.4.5 Optimization methods

Adaptive moment estimation (Adam) is an optimization method similar to SGD. However, it assigns a learning rate to each parameter and adjusts the learning rates according to the first and second-order moments of the gradient. In Adam with decoupled weight decay (AdamW), the weight decay is not linked with the learning rate, which means the weight decay and learning rate are optimized separately [24].

2.4.6 Sliding window inference

Sliding window inference is a method for performing inference on input images that are larger than what the network can take as input. The main principle behind sliding window inference is to do inference on a smaller window of the input image and then shift/slide the window along the image. This is done over the entire image, and the inference outputs are then stitched together to produce the final output. The windows used can be made to overlap with each other, in which case a function is used to decide what values each pixel within the overlapping region should get. One way is to sum the predicted probabilities for each class in the overlapping voxels, possibly with different weights for different windows, to get the final output.

2.4.7 Dice score

The Dice score is a metric used to determine the similarity in shape of two objects [25]. It is commonly used to evaluate and train models for the segmentation of medical images. The Dice score of two objects is defined as two times the magnitude of their intersection divided by the sum of their magnitudes, see Equation 1.

$$\text{Dice score} = 2 \times \frac{|X \cap Y|}{|X| + |Y|} \tag{1}$$

When used to evaluate models for semantic segmentation, X is the output of the model, and Y is the ground truth of the respective class. A Dice score of 1.0 is a perfect overlap of the output and the ground truth, while 0 means they do not overlap at all. Since the intersection and the sum of the two objects are discrete, the Dice score is not differentiable and can not be used as a loss function as it is. Soft Dice loss was introduced to use the Dice score as a loss function during training [26, 27]. It uses the predicted probabilities for each class instead of the predicted class and also takes one minus the score so that minimizing the loss translates to better segmentation. Soft Dice loss for binary segmentation is defined as

$$\text{SD} = 1 - \frac{2\sum_{i=1}^{N} p_i g_i + \alpha}{\sum_{i=1}^{N} p_i^2 + \sum_{i=1}^{N} g_i^2 + \beta} \tag{2}$$

and the mean soft Dice loss (mSD) over all classes, used for multiclass segmentation, is defined as

$$\text{mSD} = \frac{1}{C}\sum_{j=1}^{C} \text{SD}_j, \tag{3}$$

where the number of voxels in the image is denoted as N, p_i is the predicted probability for the given class for the i:th voxel and g_i is the ground truth for the same voxel. C is the number of classes, and SD_j is the soft Dice loss of the j:th class. α and β are small smoothing terms for avoiding zero in the numerator and denominator, respectively. Avoiding zero in the numerator can be important since if the numerator is zero, the gradient is zero, and the network cannot learn.

2.5 Network architectures

We will begin this section by describing the popular U-net architecture for image segmentation before moving on to describing attention mechanisms and how they are used in Transformer architectures. We will then give an overview of Vision Transformer (ViT), a network architecture for image analysis using Transformer blocks. We will then describe UNETR (UNEt TRansformers), a U-Net-like extension of ViT for image segmentation. Finally, we will describe Swin Transformer, an architecture inspired by ViT but designed to fix some of its drawbacks, and Swin UNETR, a U-Net-like extension of Swin Transformer for image segmentation.

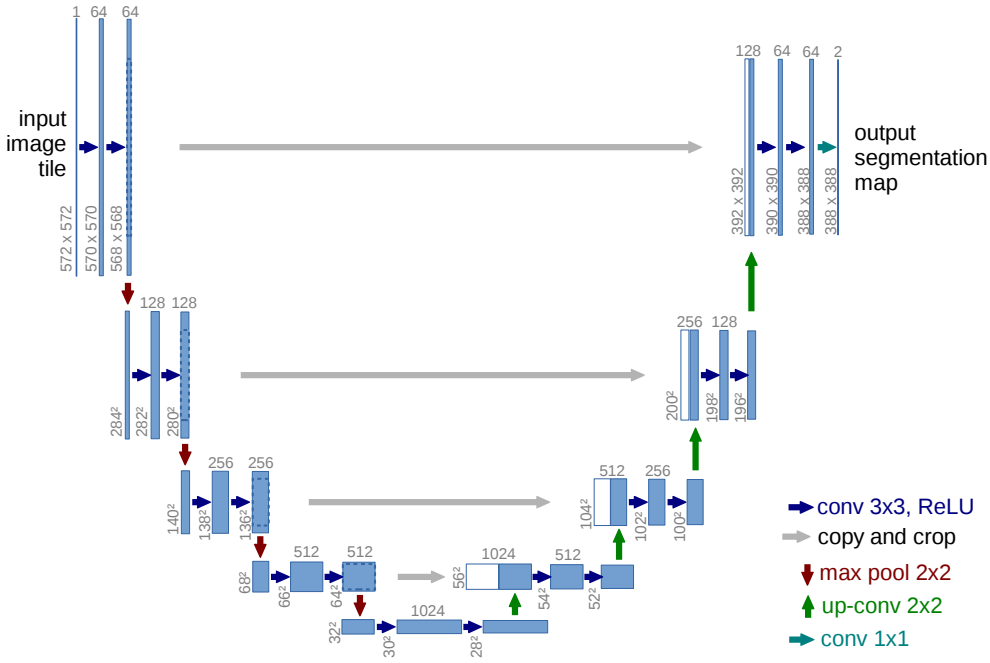


Figure 5: The architecture of the original U-Net model. *Image taken from [28]. Reproduced with permission from Springer Nature.*

2.5.1 U-Net

The U-Net architecture introduced by Ronneberger et al. [28] is a fully convolutional neural network. It is divided into two parts: a contracting path that downsamples the input image and a symmetric expansive path that upsamples the image to its original size. The upsampling is performed using transposed convolutions. For each upsampling in the expansive path, the corresponding feature map from the contracting path is concatenated to its input, which is called a skip connection. Its titular U-shape allows the network first to find small local features and then deeper into the network to find larger, more complex features. By utilizing skip connections during the upsampling, the network can combine the information from the deeper parts with the earlier parts to get details about large complex features and finer details. The architecture of the U-net can be seen in Figure 5, where each blue box represents a feature map, and the white boxes represent copied feature maps after each skip connection.

2.5.2 Scaled multi-head dot product attention

An attention mechanism is a part of a deep learning model meant to enable the model to pay attention to relevant parts of the input data when interpreting that data. This enables the model to focus on the parts of the data that are most relevant for its task and disregard the less important parts [29]. In recent years, an attention mechanism called scaled dot-product has gained popularity. Originally designed for natural language processing (NLP), this form of attention is designed for sequences. It allows the network to focus on different parts of the sequence when interpreting each element.

Scaled dot-product attention requires input data in the form of three sequences of vectors called query, key, and value. In many vision transformer models, such as those used in

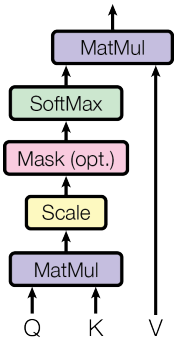


Figure 6: Scaled dot product attention. We do not use masking in this project. *Image taken from [30].*

this project, these three sequences are always identical to each other, being the output of a previous layer. The mechanism uses all queries, keys, and values to compute a new representation for each value as a weighted sum of all previous values. The dot products of a query and all keys are calculated and scaled by $\frac{1}{\sqrt{d}}$, with d being the dimension of the queries, keys, and values. They are then scaled so that they sum to one through the softmax function [19]. Since the dot product of two vectors can be interpreted as a similarity measure, this operation can be thought of as creating a vector of similarities between the query and all keys. These similarities are then used as the weights when summing all values to create a new representation of the value corresponding to the query. This is repeated so that all values get a new representation [30]. For a schematic view of scaled dot product attention, see Figure 6.

If all queries, keys, and values are packed together into matrices Q, K, and V, scaled dot product attention can be expressed as

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V. \tag{4}$$

Multi-head attention is a way to allow the model to look at the data in multiple representations when using the attention mechanism. Before the data is fed into the attention mechanism, it is transformed using h different learned linear projections, projecting the dimension of the sequence elements down to $1/h$ of the original dimensionality. Then, each of these projected sequences is transformed using attention before being concatenated and transformed with a final linear projection. Since the projections are learned, the model can learn to emphasize different aspects of the data with each projection, and the attention mechanism can therefore focus on several different properties in the data [30]. With h attention heads, multi-head scaled dot product attention can be expressed as

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(head_1, ..., head_h)W^O \\ \text{where } head_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V). \end{aligned} \tag{5}$$

Here, W_i^Q, W_i^K, W_i^V, and W^O are the projection matrices of the query, key, value, and output, respectively. See Figure 7 for a schematic view of multi-head attention.

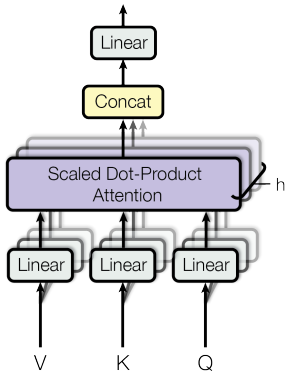


Figure 7: Illustration of multi-head attention, consisting of several attention layers running in parallel. *Image taken from [30].*

2.5.3 Transformer encoders

A transformer is a type of deep neural network initially introduced by Vaswani et al. in [30] for NLP. It uses an embedding layer, where each token in a sequence is embedded into a high-dimensional space using a learned projection. This enables the network to learn meaningful representations of tokens. A positional encoding (learned or fixed) can also be added to the sequence at this stage to let the network incorporate information about the positions of the tokens in the sequence.

The embedded token sequence is then fed into several transformer encoder layers. These layers compute the scaled dot-product multi-head attention between all the tokens in the sequence so that every token can attend to every other token and, therefore, every token can get context from the entire sequence added to its representation. After the attention has been calculated, a feed-forward layer consisting of two linear projections with a non-linear activation function in between is used to transform the output of the attention mechanism. Skip connections and layer normalizations are also utilized, see Figure 8. The original transformer architecture also uses transformer decoder layers, but we did not use these in this project and will not describe them here. This type of encoder structure outputs a sequence of token embeddings with rich feature representations incorporating global context across the entire sequence [30].

2.5.4 Vision Transformer

Vision Transformer (ViT) is a transformer-based model designed for image analysis [31]. Originally designed for image recognition, it divides the image into patches of a set size, e.g., 16x16 pixels for a 2D image. These patches are then embedded to high-dimensional vectors through a linear projection or a convolution with kernel size and stride equal to the patch size. Once embedded, these patches are viewed as a sequence, and a positional encoding is added. The encoding can be one-dimensional or multi-dimensional to fit the dimensionality of the image, but in the original paper presenting ViT, one-dimensional positional encoding was found to perform as well as multi-dimensional positional encoding [31].

Once the sequence of patch embeddings has been created, it has the same structure as a

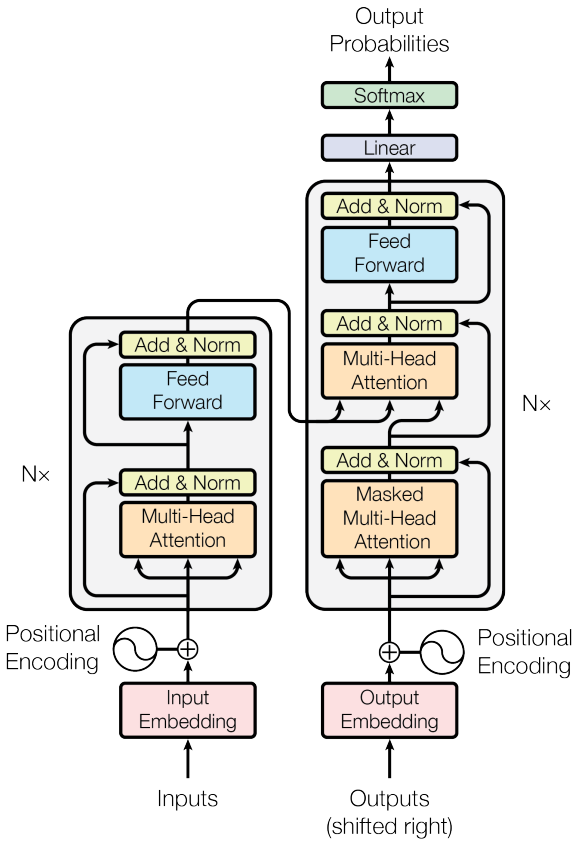


Figure 8: The Transformer model architecture. We only use the encoder on the left side of the image. *Image taken from [30].*

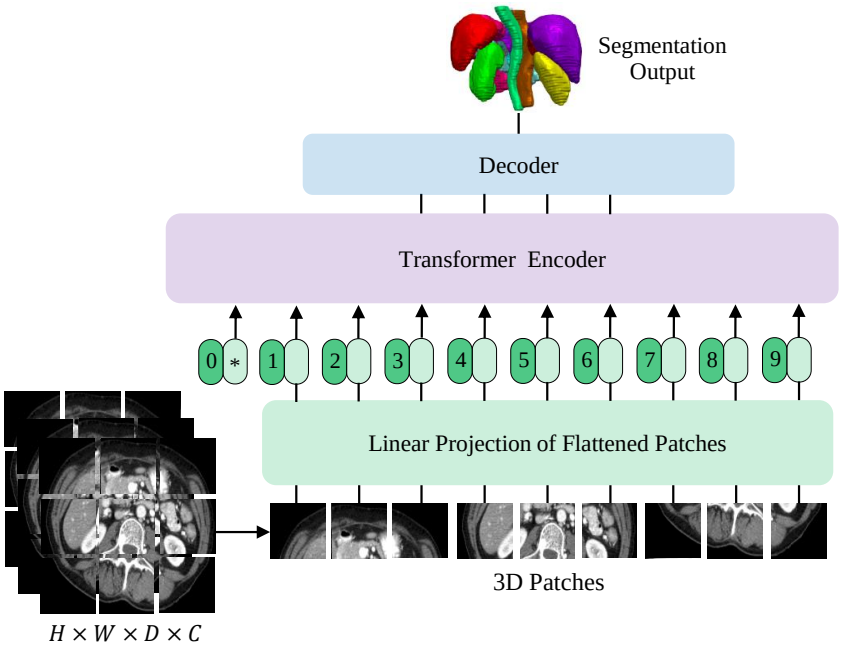


Figure 9: Schematic view of ViT and how it could be used for segmentation. The decoder can be any structure that can turn a sequence of embeddings into an image. *Image taken from [32], used under CC BY-NC.ND 4.0.*

sequence of word embeddings in NLP and can, therefore, be fed into a series of transformer layers without additional processing. The transformer layers calculate attention between all patch embeddings, represented as many-dimensional vectors. These patch embeddings and their transformed representations created by the transformer layers capture important features within the image. ViT can, therefore, use global context within the image to create a rich feature representation that can be used for downstream tasks such as classification [31]. See Figure 9 for a schematic view of how ViT could be used for semantic segmentation.

One downside of the ViT architecture is that it makes less use of reasonable assumptions about images than CNNs do. For example, it is often reasonable to assume that certain image features can occur in different places in an image, an assumption that convolutions are able to take advantage of due to them applying the same filter to the entire image. It is also often reasonable to assume that the exact location of a feature is not vital for the output, an assumption that max-pooling takes advantage of. ViT makes less use of these assumptions than CNNs [31]. Due to its patch embedding layer, each patch is downsampled to a single point in a high-dimensional space in a single transformation. This point must eventually be upsampled to its original patch size to output a per-pixel segmentation. Unless the embedding dimension is very high in relation to the patch size, capturing the fine details of the image in this manner can be challenging [33].

Since attention is calculated between all patches in the image, ViT's time complexity is highly dependent on the number of patches in the image. This means that the time complexity is highly dependent on the ratio of image resolution to patch size, particularly in the 3D case where, for a cubic image with a side of N pixels and a patch size of M, the number of patches is $\left(\frac{N}{M}\right)^3$. Since attention has to be calculated between all patches, the time complexity

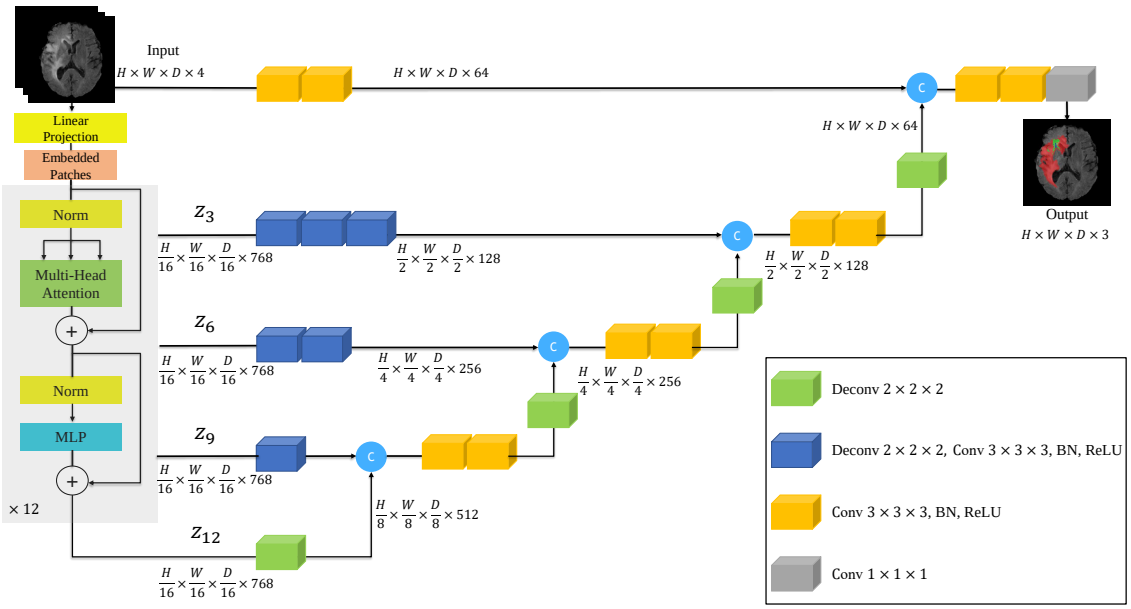


Figure 10: The UNETR architecture. *Image taken from [32], used under CC BY-NC.ND 4.0.*

of the transformer layers with respect to image and patch size is $O\left(\left(\frac{N}{M}\right)^6\right)$.

2.5.5 UNETR

UNETR is a U-net-like model utilizing a ViT backbone, designed for semantic segmentation of 3D medical images [32]. Instead of using convolutions and poolings to downsample the input image and extract features like U-net, UNETR uses ViT to perform downsampling and feature extraction. After every third transformer block, it takes an output from ViT and upsamples these to different scales before joining them together using skip connections in a U-net-like decoder. It also utilizes a skip connection directly from the input image to the output of the decoder before outputting a segmentation, see Figure 10 [32].

2.5.6 Swin Transformer

Swin Transformer is a transformer-based backbone for image analysis inspired by ViT but designed to solve some of its drawbacks. It has linear time complexity with regard to image size and outputs features at several resolutions, allowing fine detail to be captured and used in downstream tasks such as segmentation [33].

Like ViT, Swin Transformer uses a patch embedding scheme to turn patches of the image into meaningful high-dimensional representations that can be fed into a series of transformer blocks. Swin Transformer achieves linear time complexity by limiting attention to non-overlapping windows in the image. For three-dimensional data, a window is defined as a block of $M \times M \times M$ patches. Cross-window connections are added by computing attention twice within each transformer block, with the second attention layer using windows that have been shifted by $\frac{M}{2} \times \frac{M}{2} \times \frac{M}{2}$ patches such that two windows in the first layer will have

attention computed between their adjacent regions in the second layer, see Figure 11b [33]. For an input feature map $\mathbf{z}^{l-1}$, the equation for a Swin transformer block is

$$\begin{aligned}
\hat{\mathbf{z}}^{l} &= \text{W-MSA}\left(\text{LN}\left(\mathbf{z}^{l-1}\right)\right) + \mathbf{z}^{l-1}, \\
\mathbf{z}^{l} &= \text{MLP}\left(\text{LN}\left(\hat{\mathbf{z}}^{l}\right)\right) + \hat{\mathbf{z}}^{l}, \\
\hat{\mathbf{z}}^{l+1} &= \text{SW-MSA}\left(\text{LN}\left(\mathbf{z}^{l}\right)\right) + \mathbf{z}^{l}, \\
\mathbf{z}^{l+1} &= \text{MLP}\left(\text{LN}\left(\hat{\mathbf{z}}^{l+1}\right)\right) + \hat{\mathbf{z}}^{l+1},
\end{aligned} \tag{6}$$

with $\mathbf{z}^{l+1}$ being the final output of the layer. W-MSA and SW-MSA denote windowed self-attention and shifted window self-attention, respectively, and LN denotes layer norm, see Figure 11c. The multi-layer perceptron (MLP) has two layers with a GELU in between.

Instead of a positional encoding like in the original Transformer model, the Swin Transformer uses a learned positional bias $\boldsymbol{B}$ for each attention head when computing similarity:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{T}}{\sqrt{d}} + \boldsymbol{B}\right)\boldsymbol{V}. \tag{7}$$

The Swin Transformer has stages to output features at several resolution scales. Before the first stage, the input image of size $\boldsymbol{H} \times \boldsymbol{W} \times \boldsymbol{D}$ is divided into patches of size $2 \times 2 \times 2$, with $\boldsymbol{C}$ embedding dimensions. In the first Swin stage, the embeddings are fed into several Swin transformer blocks in series. The transformed embeddings are then reshaped into an image of size $\frac{H}{2} \times \frac{W}{2} \times \frac{D}{2}$, with $\boldsymbol{C}$ channels. Patch merging is then used, where blocks of $2{\times}2{\times}2$ patch embeddings are merged by concatenating their embeddings to $8\boldsymbol{C}$-dimensional vectors, which are then reduced to $2\boldsymbol{C}$ dimensions with a linear projection. This effectively doubles the size of the patches in each direction and doubles the embedding dimension, yielding an image of size $\frac{H}{4} \times \frac{W}{4} \times \frac{D}{4}$ with $2\boldsymbol{C}$ channels. All subsequent Swin stages have the same pattern of calculating attention and merging patches while doubling the number of embedding dimensions. See Figure 11a and 11d.

2.5.7 Swin UNETR

Swin UNETR is a model for semantic segmentation inspired by UNETR but utilizing a Swin Transformer backbone. It takes a feature map from the Swin Transformer backbone after each Swin stage and progressively upsamples and concatenates these feature maps with transpose convolutions, similar to a regular U-net-like model [5]. The number of Swin stages, the number of transformer blocks in each stage, and the number of attention heads in each attention are all tuneable hyperparameters of the model. The number of channels in each convolution depends on the initial embedding dimension $\boldsymbol{C}$, see Figure 12, where $\boldsymbol{C}$ is 48. There is also a modification called SwinUNETR-V2, which adds a residual convolution block at the start of each Swin stage in the encoder, see Figure 13 [34].

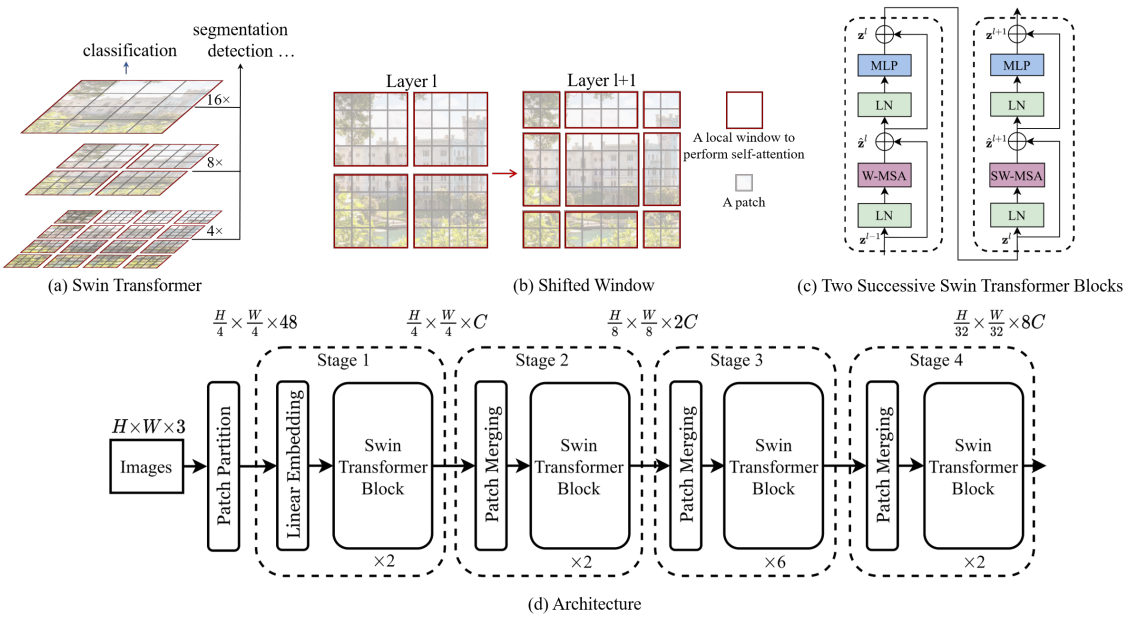


Figure 11: (a) How Swin Transformers divide images into windows and patches. (b) How attention is computed in consecutive Swin transformer blocks using shifted windows. (c) Two consecutive Swin transformer blocks and their structure. (d) Example architecture of chained Swin transformer blocks. *Image reproduced under MIT license from the Swin Transformer GitHub page [33].*

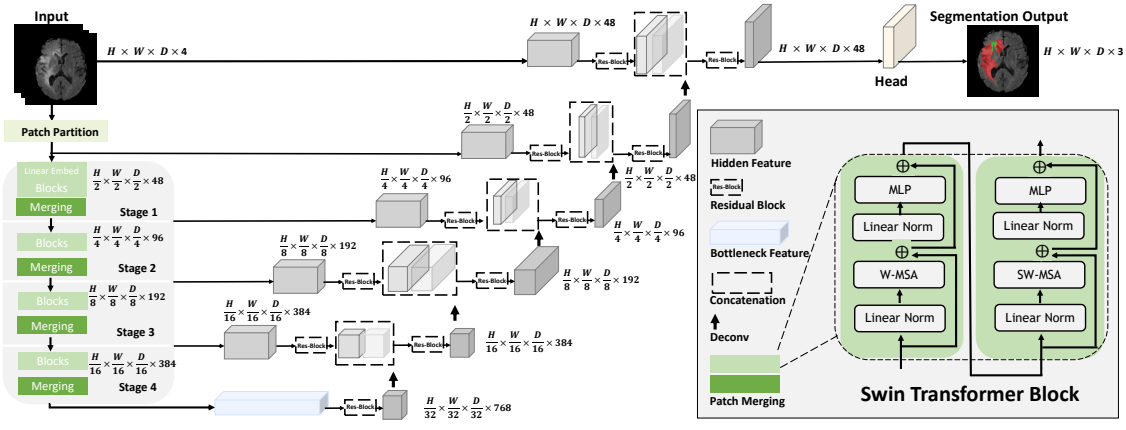


Figure 12: Example architecture of Swin UNETR and the structure of Swin transformer blocks. *Image taken from [5], used under CC BY-NC.ND 4.0.*

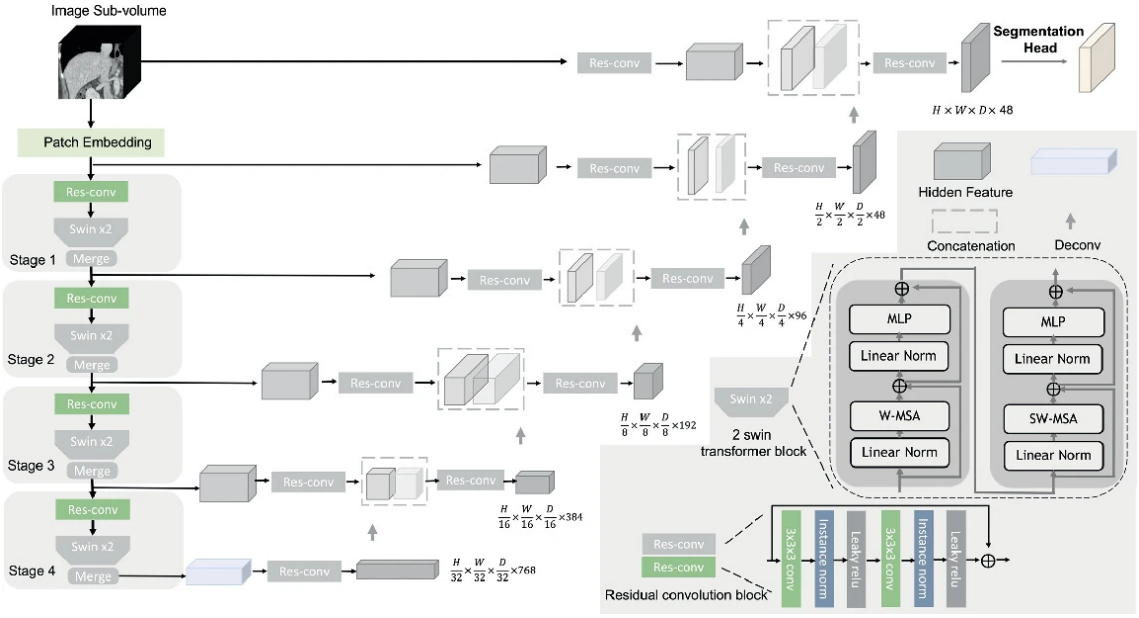


Figure 13: Example architecture of SwinUNETR-V2, the structure of Swin transformer blocks, and the structure of residual convolution blocks. *Image taken from [34]. Reproduced with permission from Springer Nature.*

3 Method

3.1 Method summary

In this work, we used datasets consisting of low-dose CT images, that will be described in Section 3.2. Using this data, we trained two types of vision transformers to segment organs and tissues: a slice-based model of our own design, that we describe in Section 3.3, and versions of Swin UNETR-V2, described in Section 3.4. To perform instance segmentation of ribs and vertebrae, we used a discriminative loss function as well as connected component labeling, described in Section 3.5.

We trained and evaluated the following networks:

- Swin UNETR-V2 with two decoders, one outputting semantic segmentation and one outputting 32-dimensional voxel embeddings for instance segmentation of ribs and vertebrae. This was evaluated on validation and test data.
- Slice model with two decoders, one outputting semantic segmentation and one outputting 16-dimensional voxel embeddings for instance segmentation of ribs and vertebrae. This was evaluated on validation and test data.
- Swin UNETR-V2 with two decoders, one outputting semantic segmentation and one outputting 32-dimensional voxel embeddings for instance segmentation of vertebrae. Instance segmentation of ribs was performed using connected component labeling. This was evaluated on validation and test data.
- Slice model with two decoders, one outputting semantic segmentation and one outputting 16-dimensional voxel embeddings for instance segmentation of vertebrae. Instance segmentation of ribs was performed using connected components. This was evaluated on validation and test data.
- Swin UNETR-V2 with one decoder, outputting semantic segmentation of all labels in the training set except lumbar vertebra 6. This was evaluated on validation data only.
- Slice model with one decoder, outputting semantic segmentation of all labels in the training set except lumbar vertebra 6. This was evaluated on validation data only.

The training was done with randomly cropped and augmented pieces of the training data, as described in Section 3.6. When evaluating the networks' performance, sliding window inference was used to segment images that were larger than the networks' input image size. Postprocessing was also performed to remove small segments. Sliding window inference and removing small objects are described in Section 3.7. Section 3.8 describes how model performance was measured.

3.2 Data

This project used three datasets: training data, validation data, and test data. The training and validation data came from the same source, whereas the test data came from a different source. The training data was used to tune the models' parameters, while the validation data was used to evaluate the models' performance during development and training. The test data was only used after the models had been finalized to compute final performance metrics.

Validation and training data

The training and validation data comprised 183 CT images of patients with known or suspected tumors. All images were collected at Sahlgrenska University Hospital in Gothenburg, Sweden, with the same model of CT camera. They were reconstructed using filtered back projection with a slice thickness and spacing of 3 mm and pixel spacing within slices of 1.37 mm. All images were low-dose images with 30 mAs exposure and 120 kV peak voltage, except four that had an exposure greater than 100 mAs with 120 kV peak voltage.

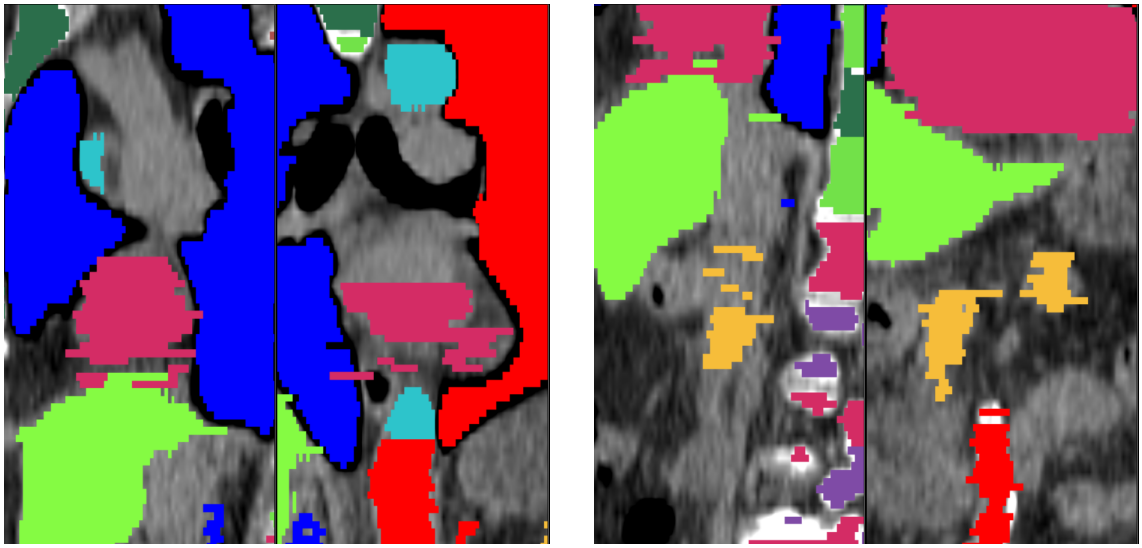
The organ labels were initially created by a team of radiologists and nuclear medicine physicians and then expanded upon by a team of machine learning engineers at Exini Diagnostics AB. Some images had missing labels, particularly the pancreas, for which only 133 images were labeled. Additionally, some patients had missing organs, such as kidneys, and 19 patients had an extra lumbar vertebra. For a complete list of all labels and counts, see Appendix A.

We observed that the pancreas often had poorly drawn labels, and some labels appeared to have been drawn slice for slice, making them look inaccurate when viewed from the front or side. See the heart and pancreas in the ground truth of Figure 14 for an example of these types of labels. The pancreas's shape also varies significantly between patients, and it has poor contrast with surrounding tissues. We also noted that the size of the bladder varied between patients, likely because of different degrees of fullness. It also had varying intensity in the CT images because some patients had been given contrast agents that had accumulated in the bladder.

Of the 183 images, five were excluded due to irregularities. Two of these had one more thoracic vertebra and two more ribs than normal, one had a missing lumbar vertebra, one had a completely fluid-filled right lung, and one had a massively inflated aorta with a stent. Because of these irregularities, the annotators had chosen not to annotate all organs in these patients. The remaining 178 images were randomly split 80/20 into training and validation sets, resulting in 143 images to be used for training and 35 to be used for validation. Of the patients in the training set, 44 were female and 99 were male, whereas 9 were female and 26 were male in the validation set. Most had a field of view ranging from the base of the skull to mid-thigh, but 11 training and 2 validation images were whole-body scans. Some patients had implants like hip prostheses or tooth fillings that were clearly visible, which could affect the segmentation performance on those patients.

Test data

The test data set consisted of 19 patients from three American hospitals. The patients were men undergoing prostate cancer screenings. They were manually segmented by a nuclear



(a) Heart label.

(b) Pancreas label.

Figure 14: Zoomed in examples of poorly drawn labels. (a) The pink middle label is the heart. (b) The orange middle label is the pancreas.

medicine physician. They were collected with similar exposure and peak voltage as the training and validation images, but with several different CT cameras and varying resolution. Unlike the training and validation data, almost all images ranged from the vertex of the skull to the mid-thigh, except one that started at the base of the skull.

Preprocessing

We performed preprocessing on the images before they were used for training or inference. Due to GPU memory limitations, the images had to be downsampled to a slice resolution of 128x128 pixels while resampling the height to 3 mm. To retain as much information as possible in the images, they were first cropped to remove the air around the patient. This was done by creating a bounding box around the patient, using a threshold value of -200 HU. Cropping air in this manner creates images that are not square in width and depth, which causes different images to be differently stretched when they are subsequently downsampled to be square. The image intensities were then rescaled with a linear scaling that mapped -700 HU to 0 and 1400 HU to 1. The values -700 HU and 1400 HU were chosen because they were approximately the minimum and maximum values of major tissues in the body, and we wanted the relevant voxels in the image to have values between 0 and 1.

3.3 Slice model

One of the models we used in this project was a slice-based transformer model that we developed ourselves, inspired by ViT and UNETR. It uses a patch embedding like ViT, but instead of being cubic, the patches have the shape of an entire axial slice of a preprocessed image. Because an entire slice contains more voxels than a small cubic patch like the ones used in UNETR, we used a high number of embedding dimensions to let the model encode

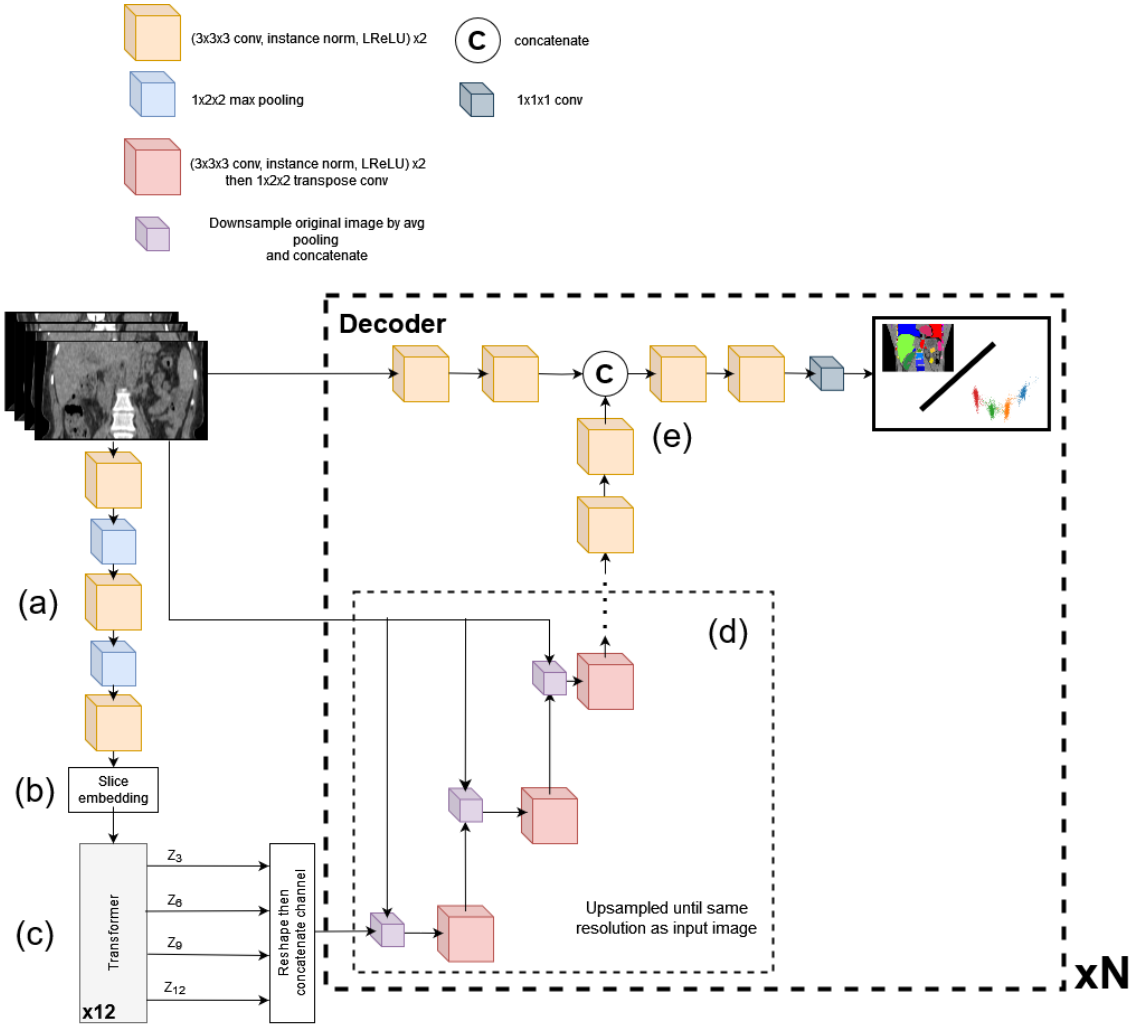


Figure 15: The architecture of the slice model. N is the number of decoders, 1 or 2, depending on if instance segmentation is performed. (a) Pre-embedding convolutions. (b) Slice embedding layer. (c) Transformer layers. (d) Decoder that upsamples the image back to the original size, and concatenates the original image downsampled with average pooling. (e) Final convolutions for getting the output.

detailed information about the slice's content. To help the model do this, we added several convolutional layers with max pooling before the slice embedding, helping the network turn an entire slice into a meaningful encoding. Attention is then calculated between all embedded slices.

The decoder is similar to UNETR's decoder, but with the addition that at every resolution, a copy of the original image downsampled to match the current resolution is added as an extra channel to help the model incorporate fine details that were lost when the slices were embedded. The architecture of the slice model can be seen in Figure 15. The network has one decoder for producing the semantic segmentation output. For the versions of the network that perform instance segmentation, another almost identical decoder is added to produce voxel embeddings. The difference between the two decoders is the number of output channels, which correspond to the number of semantic classes and the number of dimensions in the voxel embeddings, respectively. The transformer block consists of 12 transformer encoders, and its outputs, $Z_{3,6,9,12}$, are the outputs of the corresponding transformer encoders. The components of the model are described in detail below.

3.3.1 Model summary

The slice model divides an image into a sequence of axial slices. It embeds these into a many-dimensional space with the help of a series of convolutions to create meaningful embeddings. The sequence of embedded slices is then fed into a series of transformer blocks, computing attention between them. This enables the model to include context from the entire image in the representation of each slice. For example, when creating an embedding of a slice containing a piece of rib, information about the corresponding vertebra located several slices higher up can be included in the embedding.

The slice embeddings outputted by the transformer layers are then reshaped into a feature map with the same number of slices as the input image, but each slice has a resolution of 1×1 pixels and 4096 channels. This low-resolution feature map is inputted into a decoder.

The decoder progressively upsamples the slices in the feature maps using convolutions until they reach the original input image resolution. During each stage of the upsampling, a copy of the input image, downsampled to the resolution of that stage, is added as an extra image channel to add fine detail. After a final series of convolutions, the decoder gives a final output.

3.3.2 Pre-embedding convolutions

Before the slice embedding stage, a series of convolutional layers and max poolings are used to create feature maps, see Figure 15a. All convolutions have kernels of size $3 \times 3 \times 3$, enabling limited feature mixing between adjacent slices. They have a stride of one voxel, 21 output channels, and 21 input channels, except the first convolution, which only has one input channel because its input is the original single-channel CT image. The max poolings have kernel $1 \times 2 \times 2$ and stride $1 \times 2 \times 2$, meaning that we only downsample the slices, not the number of slices. A leaky ReLU activation function and a layer normalization follow each convolution.

The convolutions and max poolings are ordered as follows:

$$\text{conv} \times 2 \rightarrow \text{ max pool} \rightarrow \text{conv} \times 2 \rightarrow \text{max pool} \rightarrow \text{conv} \times 2,$$

meaning the resolution of the slices is downsampled by a factor of four in both dimensions. The number of channels in the convolutions was kept low because the number of parameters in the subsequent slice embedding layer would otherwise become very large.

3.3.3 Slice embedding

The slice embedding layer is similar to the one in ViT, but instead of cubic patches, we use a patch size equal to the size of an axial image slice after the pre-embedding convolutions. See Figure 16 for examples of axial slices in a CT image. The slice embedding layer can be seen in Figure 15b. The number of dimensions in the embedding is 1024. The embedding is performed using convolutions with a learned convolution with a kernel the size of a slice, a stride of one, 21 input channels, and 1024 output channels. A learned, one-dimensional positional encoding is added to the embeddings to add information about the relative positions of slices.

Figure 16: Examples of axial slices in a CT image.

3.3.4 Transformer layers

After embedding, the slices are sent into a stack of transformer encoders, see Figure 15c. We use 12 transformer encoders in series, with 8 attention heads and an MLP dimension of 1024. An output is taken from every third transformer encoder to obtain multiple embedded sequences. The embedded sequences are reshaped such that every embedding in the sequence becomes a slice in a feature map with a resolution of $h \times 1 \times 1$ and 1024 channels, where h is the number of slices in the input image. With 12 transformer layers, four outputs are reshaped into feature maps, and their channels are concatenated to produce a feature map with 4096 channels.

3.3.5 Decoder

The decoder consists of a series of convolutions and transposed convolutions. The convolutions have kernels of $3 \times 3 \times 3$ and a stride of 1, and the transposed convolutions have kernels

and strides of $1 \times 2 \times 2$ to upsample each slice by a factor of two in both dimensions. The convolutions are layered as

conv → layer norm → LReLU → conv → layer norm → LReLU → transposed conv

and repeated until the image has been upsampled to its original size. These convolutions are represented by the red blocks in the decoder in Figure 15d. All convolutions have 170 input channels, or 171 if they occur after a downsampled version of the original image has been concatenated. All convolutions have 170 output channels.

3.3.6 Downsampling of original image

During the decoder's upsampling, a module downsamples the slices in the original image to match the resolution of the various stages of the decoder. The downsampling is done by average pooling, and after each upsampling of the feature map, a downsampled version of the original image is added as an extra channel to the new feature map. This downsampling and concatenation is represented by the purple boxes in Figure 15d.

3.3.7 Final output

After the feature maps have been upsampled to the resolution of the original image, they are concatenated with a feature map constructed by a series of four convolutions performed on the original image, see Figure 15e. These convolutions have $3 \times 3 \times 3$ kernels, stride one, and 42 input and output channels, except the first one, which has one input channel. Like all convolutions in the model, each one is followed by layer normalization and leaky ReLU. After the concatenation, the feature map is fed into four additional convolutions with 84 channels. The final output of the model is obtained by convolving the feature map with a $1 \times 1 \times 1$ kernel with stride one and a number of output channels equal to the number of classes in the data, including the background class, or, in the instance decoder, the number of embedding dimensions. From the output of the semantic decoder, class probabilities for each pixel can be obtained with the softmax function. From the output of the instance decoder, voxel embeddings can be extracted directly.

3.4 Swin UNETR

The second model we used in this project was SwinUNETR-V2 [34]. We used four Swin stages with two Swin Transformer blocks each and 3, 6, 12, and 24 attention heads in each stage, respectively. The initial patch embedding dimension was 48, and we used a window size of 7 patches. We used two separate decoder paths, one for semantic segmentation and one for instance segmentation with discriminative loss, with the decoder path for discriminative loss having 32 output channels corresponding to a 32-dimensional embedding. Because the model must upsample the patches in the deepest encoder layer by a factor of 32, the size of the input image must be a multiple of 32 in every spatial dimension.

3.5 Instance segmentation

We performed instance segmentation of ribs and vertebrae by implementing discriminative loss, presented by De Brabandere et al. [35]. To perform instance segmentation with the discriminative loss function, the network was modified to have an additional output. The second output is a 16- or 32-dimensional pixel embedding for each voxel in the image. In one experiment setup, discriminative loss was used to segment the lumbar and thoracic vertebrae and the right ribs and left ribs into instances. In another setup, discriminative loss was only used on the vertebrae, and another method was used for the ribs. When using discriminative loss, the classes that are segmented into instances still have to be semantically segmented, but all instances belonging to the same class are given the same label. This means that the lumbar and thoracic vertebrae and the right and left ribs are divided into four different semantic classes. Only the embeddings of the voxels corresponding to these classes are of interest to the instance segmentation, and the semantic outputs for these classes are used as a mask to find which voxels are of interest.

The idea behind the loss is for the network to cluster the embeddings of voxels belonging to the same instance while increasing the distance between clusters of different instances. The loss consists of three terms that are weighted and summed together to get the final loss:

- A variance term, L_{var}, that depends on the distance between the pixel embeddings and the corresponding cluster center. A cluster center is the mean embedding of all pixels in the same instance. This term decreases as the pixel embeddings get closer to the mean embedding, reducing the variance of the clusters. The variance term is hinged, meaning the loss is set to zero for an embedding if the distance between it and the cluster center is smaller than δ_{v}.
- A distance term, L_{dist}, that depends on the distance between the cluster centers of different instances. This term decreases as the distance between the cluster centers increases. The distance term is also hinged, meaning that the loss for two clusters is set to zero if the distance between them is greater than $2\delta_{\mathrm{d}}$.
- A regularization term, L_{reg}, that depends on how far away all the cluster centers are from the origin. This term decreases as the distance between the cluster centers and the origin decreases.

A visualization of how the embeddings are affected by the distance and variance terms can be seen in Figure 17. The loss function is expressed as

$$L_{var} = \frac{1}{C}\sum_{c=1}^{C}\frac{1}{N_c}\sum_{i=1}^{N_c}\left[\|\mu_c - x_i\| - \delta_{\mathrm{v}}\right]_+^2, \tag{8}$$

$$L_{dist} = \frac{1}{C(C-1)}\sum_{\substack{c_A=1\\ c_A\neq c_B}}^{C}\sum_{c_B=1}^{C}\left[2\delta_{\mathrm{d}} - \|\mu_{c_A} - \mu_{c_B}\|\right]_+^2, \tag{9}$$

$$L_{reg} = \frac{1}{C}\sum_{c=1}^{C}\|\mu_c\|, \tag{10}$$

$$L = \alpha \cdot L_{var} + \beta \cdot L_{dist} + \gamma \cdot L_{reg}, \tag{11}$$

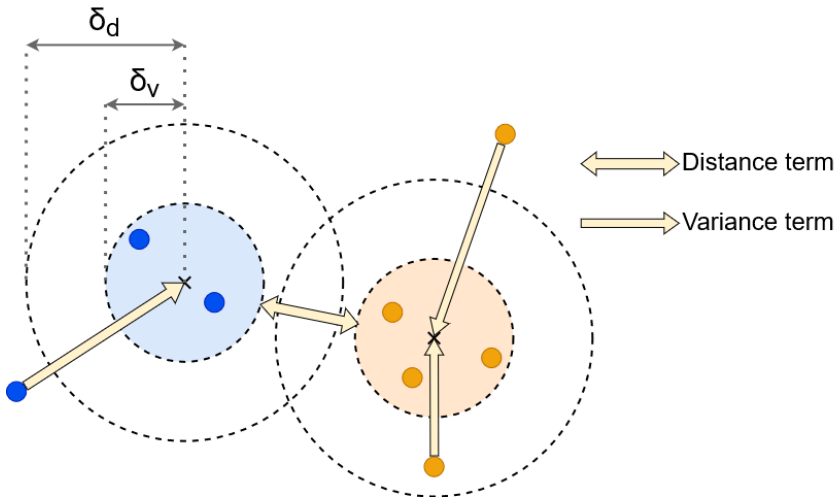


Figure 17: Visualization of how discriminative loss affects the embeddings. The variance term pulls embeddings of the same instances together, and the distance term pushes the embeddings of different instances away from each other.

where C is the number of clusters in the image, N_c is the number of pixels in cluster c, x_i is the i:th pixel embedding in a cluster, μ_c is the cluster center for cluster c. $[x]_+ = \max(0, x)$ denotes the hinge and δ_v and δ_d denote the margins for the hinge in the variance and distance term respectively. We used $\delta_v = 1.5$ and $\delta_d = 3$. $\|\cdot\|$ is the Euclidean distance in our implementation. α, β and γ are the weights for the different loss terms. We used $\alpha = 1.5$, $\beta = 1$, and $\gamma = 0.0001$ in this work.

Postprocessing is needed to get the final instance segmentation. Only the embeddings of the voxels in the mask of the instance classes created by the semantic segmentation are used for the post-processing. A simple variant of the K-means algorithm is used to cluster the embeddings into instances [36]. The pixels belonging to the same instance are found by picking out all pixels whose distance from the corresponding cluster center is less than the distance δ. We used $\delta = 3$. The cluster centers are found by randomly picking out a pixel embedding and then iteratively calculating the mean embedding of all embeddings within δ and using the embeddings within δ around this new point to calculate a new mean. This is repeated until the cluster mean convergences or for a maximum number of iterations. After convergence, all embeddings within δ of the mean are considered to be part of the same instance. After an instance is found, its embeddings are removed and not considered when finding the next cluster center. This process is repeated until every voxel has been assigned to an instance. The discriminative loss divides the ribs and vertebrae into instances within the blocks of 64 slices before the sliding window inference is performed.

To visualize the embeddings, we performed principal component analysis (PCA) to reduce their dimensionality to two-dimensional while preserving as much of the structure of the embedding space as possible. For a given image, PCA was calculated using the embeddings outputted by the network for all voxels marked as belonging to the type of instance being examined (left ribs, right ribs, lumbar vertebrae, or thoracic vertebrae) in the ground truth. The dimension-reduced embeddings were then scatter plotted, with different colors for voxels belonging to different instances.

3.5.1 Instance segmentation using connected component labeling

The second method used for segmenting the ribs into instances was using connected component labeling on the predicted semantic classes of the ribs. This means that all non-contiguous segments in the semantic class are given separate labels. The connected component labeling is performed on the entire output image after the sliding window inference. The ribs are thus handled the same as the semantic classes during the sliding window inference. Note that when this method was used to separate the ribs, discriminative loss was still used to separate the vertebrae into instances.

3.6 Training the network

We implemented multiple versions of both networks, that all were made for slightly different tasks. One task was pure semantic segmentation of all classes. The other two versions were both to do instance segmentation on the ribs and vertebrae, with two different approaches for the ribs. The networks were trained on the preprocessed images. When training the network, random blocks of 64 consecutive axial slices from each image were chosen and used as input. With a voxel height of 3 mm, the blocks had a height of 19.2 cm. Every epoch, a new random block chosen for each image to ensure training was done on all parts of the body for all patients. Data augmentation was performed on the random blocks before inputting them into the network. We used a batch size of one due to memory constraints. Because of the random cropping, each batch contains different organs and parts of the body, and with a batch size of only one, this leads to the network training on different organs in each batch. When the networks are presented with input images that are larger than 64 axial slices, sliding window inference is used to input 64 slices at a time and stitch together the outputs. The overlap between the windows is handled differently for semantic and different instance classes.

The optimizer used for the final training of all models was AdamW, with an initial learning rate of 0.001. The initial batch accumulation number (described below) used was 32. After the training seemed to have converged, the learning rate was reduced to 0.0002 and the gradient accumulation was increased to 72. Dropout of 0.2 was used on some parts of the networks.

3.6.1 Semantic segmentation

Soft Dice loss was used to train the network's semantic segmentation part. The smoothing terms α and β, as seen in Equation 2, were set to 0 and 10^{-5}, respectively. The background was included as a class in the calculation. The loss was reduced to a scalar by calculating the mean Dice of all classes. Because not all classes were labeled in some patients, we did not want to include these missing classes in the loss. They were excluded by multiplying the loss for all missing classes with zero before taking the mean, so the loss value for that class would not depend on the network's parameters, and the parameters would not be updated based on these classes for the patients with missing labels.

3.6.2 Data augmentation

During training, we used data augmentation to reduce the amount of overfitting and to make the model able to handle small variations from the norm. The augmentations were applied in the following order:

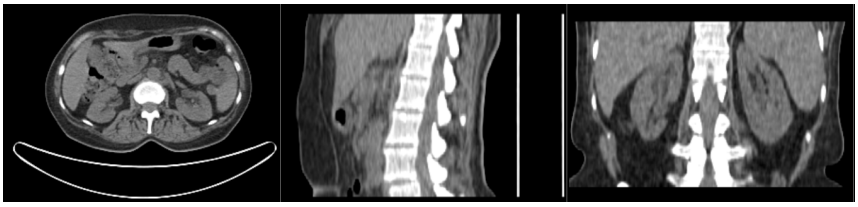
- With a probability of 75%, we perform a random elastic deformation consisting of all the following transformations:
  - Random rotation about all three image axes, with the amount of rotation in radians drawn from a uniform distribution in the range $[-0.08, 0.08]$ independently for each axis.
  - Random scaling along all three image axes, with the scaling factor drawn from a uniform distribution in the range $[0.9, 1.1]$ independently for each axis.
  - Random shearing of all three image axes, with the shear angle in radians being drawn from a uniform distribution in the range $[-0.1, 0.1]$ independently for each axis.
  - Random elastic deformation, with the random offsets on the grid being from a uniform distribution in the range $[0, 100]$ and the random offset grid being smoothed with a Gaussian kernel with a standard deviation drawn from a uniform distribution in the range $[4, 8]$.
- With a probability of 90%, we add Gaussian noise with a mean of 0 and standard deviation of 0.01.
- With a probability of 50%, we perform a gamma adjustment with the gamma being drawn from a uniform distribution in the range $[0.85, 1.15]$.
- With a probability of 25%, we perform a random histogram shift to the image intensities. The histogram shift is done with a random nonlinear intensity mapping with 25 control points.

An example of the preprocessing and data augmentation can be seen in Figure 18.

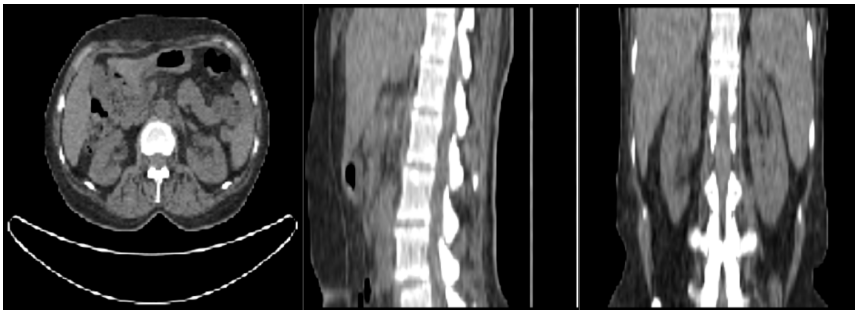
3.6.3 Performance considerations

All training was done on an Nvidia GeForce GTX 1080 Ti GPU with 11GB VRAM, using PyTorch. Since the images were three-dimensional, they and the feature maps constructed by the networks used a large amount of VRAM.

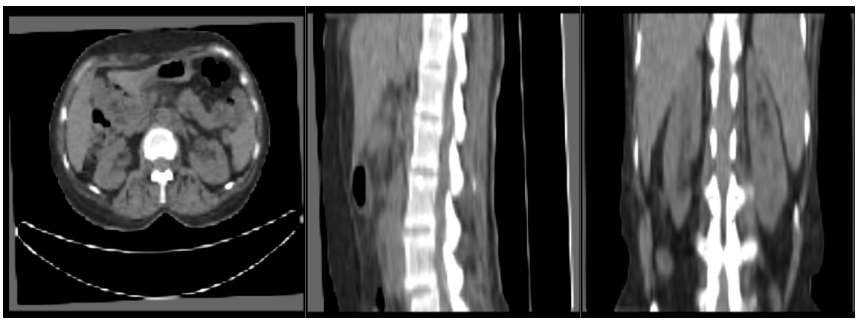
We could not use a batch size greater than one without running out of memory. To enable us to make use of the benefit of larger batch sizes, we did not apply a weight update after each backward pass but rather accumulated the calculated gradients with a technique called gradient accumulation [37]. After 32 batches, the accumulated gradients were divided by 32, and the weights were updated. This effectively simulated the effect of using a batch size of 32, except that we could not use batch normalization as implemented in PyTorch and had to use instance normalization instead. To further reduce memory usage, we used PyTorch's automatic mixed precision, reducing memory usage at the cost of numerical precision. Towards the end of each model's training, the number of accumulated gradients was increased to 72 (half the training set) to fine-tune the models.



(a) Input image.



(b) Preprocessed image.



(c) Augmented image.

Figure 18: (a) Input image with 64 axial slices in original resolution. The image was zoomed in and the air around the patient is thus not visible. (b) Preprocessed input image. Note that the image has been stretched and has a lower resolution. (c) Random augmentation of the preprocessed image.

3.7 Postprocessing

3.7.1 Sliding window inference

The networks in this work take input images with the shape $64 \times 128 \times 128$. The height of the input images can, however, be larger than 64 slices. For example, an image that ranges from the base of the skull to mid-thigh consists of about 300 slices. To do inference on the entire image, sliding window inference was used. We used sliding windows with the shape $64 \times 128 \times 128$, sliding from the top of the image to the bottom, with an overlap of 50%. For the semantic segmentation, the output values of overlapping voxels were weighted together with a Gaussian bell curve centered on the center of each window.

For the instance segmentation with discriminative loss, we implemented functions to decide which instance each voxel should belong to in the overlapping parts of the windows.

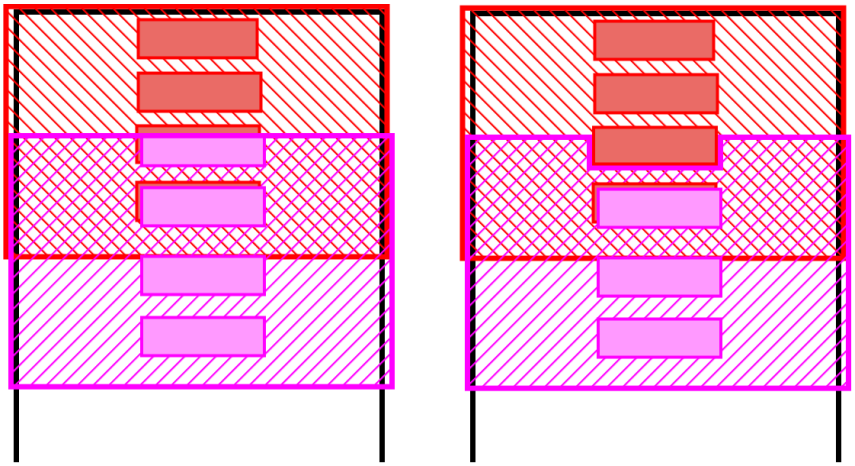


Figure 19: Illustration of sliding window overlap function for instances of vertebrae. The red and pink regions represent two windows. Instances that touch the top of a window do not overwrite the output of the previous window.

We implemented one function for the ribs and one for the vertebrae.

The implementation for the vertebrae assumes that each vertebra fits fully, without touching the edges, in at least one window. Since an overlap of 50% was used, the edge between two windows is always in the middle of a third window. For the assumption to be true with the height of slices we use, a vertebra has to be less than 9.6 cm tall. The function works by assigning all voxels with the value of the window furthest down, overwriting the prediction of the previous window. This is except for if an instance touches the top of the window. If there are any such instances, the voxels of these instances will not overwrite the previous window. Instead, the prediction from the previous window is kept in this part of the overlap. The first, uppermost, window is an exception in that instances "touch" the top of that window will be kept as is. This overlap function is illustrated in Figure 19. The windows in the figure have an offset in order to show the predictions of both predictions in the overlapping part. The left half shows the predictions for two windows. Since the top vertebra in the purple image touches the top of the window, it does not override the red prediction, the prediction of the red window is used instead. The right half shows the final image after the two windows have been stitched together.

The implementation for the ribs stitches together overlapping instances into one single instance. Given two overlapping windows, it first separates non-contiguous segments in each window into separate instances. Then, it checks for overlapping instances between the two windows recursively to create groups of instances where each instance in a group overlaps with at least one other instance in that group. The union of every instance in the group becomes one instance in the function's output. To ensure continuity with previous windows higher up in the image, the function checks if any of the instances in the group touch the top of the topmost window. If one does, the entire output instance will get the same label as that instance, ensuring that if the rib continues above the top window, it will not be split into two instances at the border. Two instances have to overlap more than 10 voxels to count as overlapping. See Figure 20, where the upper window (red) contains two instances belonging to the same rib, which extends beyond the top of the window. The lower window (pink) also

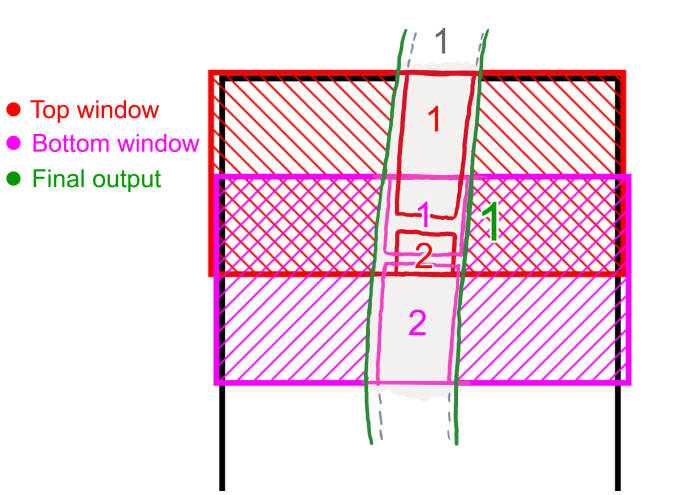


Figure 20: Illustration of sliding window overlap function for instances of ribs. Instances that overlap more than a set threshold are given the same label.

contains two instances belonging to the same rib. Since the four instances overlap in a chain, they get combined into a single instance (green) with the same label as the topmost instance in the upper window.

3.7.2 Removing small objects

The networks sometimes output small segments in wrong locations. We added a postprocessing step to remove such segments. After the model has outputted a segmentation, all connected components with a volume below a threshold are removed. The threshold is measured in voxels in the resolution of preprocessed images, and each class of object has its own threshold. For the semantically segmented classes, the threshold was set to 20% of the mean class volume in the training data. For the instance classes, threshold volumes were found by manually tuning them on the validation data. The final thresholds used can be seen in Table 1. If an object or instance touches the top or bottom of the image, it is not removed. This is done to not accidentally remove organs of which only a part is visible in the field of view of the image.

Table 1: The threshold volumes used for removing small objects in the instance segmentations. The volumes are measured in voxels in the resampled size.

Class	Threshold
Lumbar	3493
Thoracic	1873
Right ribs	42
Left ribs	42

3.8 Performance metrics

For the semantic segmentation, Dice scores, both the average over all classes, as seen in Equation 3 and the separate Dice score for each class, as seen in Equation 2, were used to evaluate performance.

A modified version of the panoptic quality introduced in [7] was used to evaluate the instance segmentation. Panoptic quality (PQ) is the product of segmentation quality (SQ) and recognition quality (RQ), and their modified definitions are shown in Equation 12. We changed how the overlap is measured in the metrics, from intersection over union (IoU) to using the Dice score. Both SQ and PQ were also measured individually when evaluating the segmentation. In Equation 12, p is a predicted segment, g is a segment in the ground truth, $Dice$ is the Dice score of two segments, and TP, FP, and FN are the number of true positives, false positives, and false negatives respectively in the image.

$$\mathrm{PQ} = \underbrace{\frac{\sum_{(p,g)\in TP} \mathrm{Dice}(p,g)}{|TP|}}_{\text{segmentation quality (SQ)}} \times \underbrace{\frac{|TP|}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}}_{\text{recognition quality (RQ)}} \tag{12}$$

RQ is the F1-score, and SQ is the mean Dice score of all true positive instances. True positive instances are defined as predicted segments that overlap more than 50% with a ground truth label, i.e., that they have a Dice score > 0.5. Because the segments in the ground truth do not overlap with each other, nor do the predicted segmentation masks, each ground truth segment can only be matched to one predicted segment at a time, and vice versa. If a predicted segment does not overlap with a ground truth more than 50%, it is considered a false positive. A ground truth segment that does not overlap with any predicted segment more than 50% is considered a false negative. The Dice score in SQ is only calculated for the segments and ground truth pairs considered true positives.

Dice score and PQ were used to measure model performance on validation and training data. Dice was used on the semantically segmented classes, and PQ on the instance segmented classes. These metrics were computed after our segmentations, which had a slice resolution of 128×128, were resampled to the original image resolution. As a result, our segmentations were more pixelated than the ground truth they were measured against.

4 Results

4.1 Semantic segmentation

The Dice scores for the models performing instance segmentation on ribs and vertebrae are in Table 2. Overall, both the slice model and the Swin model performed well, with Dice scores of 0.8 or above for most classes. Classes that the models performed poorly on include adrenal glands and pancreas. The models had inconsistent performance on the urinary bladder, sometimes missing more than half of it.

The slice and Swin models performed similarly on the semantic classes in the validation data, but the Swin model outperformed the slice model on the test data. This difference was larger for the models that performed instance segmentation of ribs with discriminative loss. The slice model performed much better on semantic segmentation of the test data when using connected component labeling for ribs compared to when using discriminative loss for ribs.

The results of the pure semantic segmentation, where each rib and vertebra is segmented as its own class, are shown in Table 3. Both models performed poorly at semantic segmentation of the ribs and vertebrae. Although some ribs and vertebrae had Dice scores upwards of 0.8, the segmentations were visually unappealing, with the vertebrae being smeared together and ribs being split into many classes. Examples of this can be seen in Figure 21 and 22.

4.2 Instance segmentation

The PQ, SQ, and RQ achieved by the models performing instance segmentation purely with discriminative loss are shown in Table 4. Examples of a few segmentations in frontal and sagittal planes are displayed in Figures 23, 24, and 25. An example of when the segmentation worked well on all organs is shown in 3D in Figure 26. Both the Swin model and the slice model achieved high SQ on the instance segmentation task, but RQ was generally lower, leading to reduced PQ, especially on the ribs. The low RQ was caused by the models splitting ribs into multiple instances, as seen in Figure 27. Both models tended to undersegment the vertebrae slightly, but the segmentations were visually appealing, with clear separation between different vertebrae. Sometimes, however, the segmentation of the vertebrae did not work as well, as seen in Figure 28. Just as in the semantic segmentation task, the slice model performed much worse on the test data than on the validation data, whereas the Swin model only performed slightly worse.

4.3 Instance segmentation with connected components

The PQ, SQ, and RQ of the models performing instance segmentation of ribs with connected component labeling are shown in Table 5. The scores are similar to the ones achieved by the models performing instance segmentation of the ribs using discriminative loss, except that the ribs have significantly higher RQ and, therefore, higher PQ. This is because, with this method of instance segmentation, ribs are not being split into multiple segments. Occasionally, two or more ribs are merged if they are so close to each other that the space between them cannot be resolved at the resolution we perform inference.

Table 2: Dice scores for organs and tissues for the models that performed instance segmentation. The first four columns are for the models that performed instance segmentation with discriminative loss for both ribs and vertebrae. The last four columns are the models that performed instance segmentation on ribs with connected components labeling and on vertebrae with discriminative loss. Since the test data had fewer labels than the training data, some Dice scores could not be computed. These were not included in the calculation of the mean.

	Ribs with discriminative loss				**Ribs with connected components**			
	Validation data		**Test data**		**Validation data**		**Test data**	
Organ/Tissue	**Slice**	**Swin**	**Slice**	**Swin**	**Slice**	**Swin**	**Slice**	**Swin**
Adrenal gland left	0.363	0.421	-	-	0.449	0.440	-	-
Adrenal gland right	0.414	0.390	-	-	0.457	0.435	-	-
Aorta abdominal part	0.773	0.785	0.619	0.733	0.794	0.782	0.737	0.731
Aorta thoracic part	0.831	0.867	0.804	0.851	0.856	0.873	0.857	0.864
Brain	0.935	0.963	0.749	0.932	0.953	0.961	0.907	0.746
Clavicle left	0.876	0.880	0.807	0.867	0.884	0.884	0.869	0.859
Clavicle right	0.880	0.883	0.845	0.861	0.889	0.880	0.880	0.839
Femur left	0.911	0.923	0.885	0.927	0.928	0.932	0.913	0.934
Femur right	0.923	0.933	0.858	0.928	0.931	0.933	0.899	0.934
Gallbladder	0.638	0.648	0.511	0.620	0.687	0.607	0.641	0.566
Gluteus maximus left	0.896	0.917	-	-	0.915	0.917	-	-
Gluteus maximus right	0.901	0.922	-	-	0.913	0.922	-	-
Heart	0.868	0.899	0.798	0.877	0.888	0.898	0.833	0.869
Hip bone left	0.903	0.909	0.821	0.917	0.911	0.910	0.903	0.920
Hip bone right	0.909	0.913	0.805	0.920	0.912	0.913	0.890	0.920
Humerus left	0.870	0.898	0.796	0.882	0.889	0.900	0.876	0.902
Humerus right	0.883	0.899	0.790	0.888	0.890	0.901	0.860	0.904
Kidney left	0.845	0.879	0.678	0.808	0.874	0.879	0.780	0.842
Kidney right	0.862	0.884	0.659	0.818	0.881	0.874	0.824	0.831
Liver	0.934	0.946	0.879	0.939	0.940	0.948	0.856	0.946
Lung left	0.942	0.948	0.780	0.943	0.946	0.950	0.928	0.933
Lung right	0.963	0.965	0.812	0.960	0.964	0.966	0.917	0.961
Mandible	0.807	0.817	0.758	0.784	0.8198	0.804	0.775	0.749
Pancreas	0.392	0.418	-	-	0.302	0.455	-	-
Sacrum and coccyx	0.901	0.921	0.884	0.915	0.920	0.922	0.900	0.920
Scapula left	0.835	0.841	0.745	0.841	0.848	0.845	0.834	0.841
Scapula right	0.841	0.842	0.775	0.830	0.844	0.841	0.820	0.845
Skull	0.806	0.814	0.702	0.777	0.824	0.817	0.788	0.791
Spleen	0.875	0.887	0.814	0.831	0.897	0.890	0.829	0.850
Sternum	0.853	0.859	0.859	0.856	0.871	0.859	0.878	0.863
Urinary bladder	0.755	0.727	0.503	0.596	0.773	0.765	0.614	0.670
Vertebra cervical all	0.821	0.857	0.734	0.802	0.857	0.854	0.832	0.832
Mean of all classes	**0.822**	**0.837**	**0.766**	**0.849**	**0.839**	**0.839**	**0.839**	**0.847**

Table 3: Separate Dice scores for various organs and tissues segmented with only semantic segmentation on validation data.

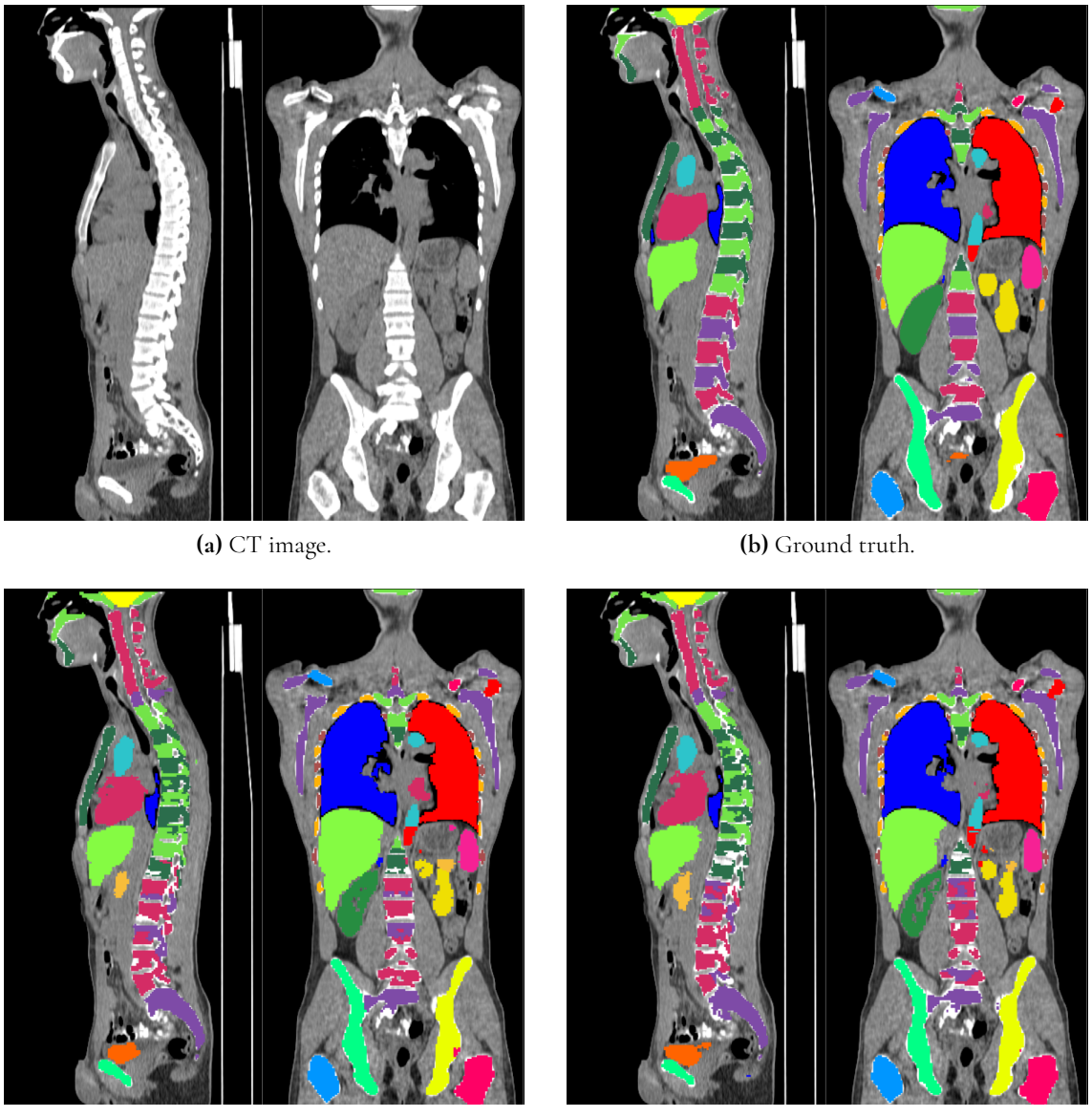
Organ/Tissue	Slice model	Swin model	Organ/Tissue	Slice model	Swin model
Adrenal gland left	0.449	0.436	Rib right 1	0.790	0.800
Adrenal gland right	0.460	0.428	Rib right 2	0.792	0.800
Aorta abdominal part	0.780	0.784	Rib right 3	0.738	0.770
Aorta thoracic part	0.846	0.867	Rib right 4	0.659	0.733
Brain	0.926	0.961	Rib right 5	0.636	0.674
Clavicle left	0.881	0.876	Rib right 6	0.650	0.627
Clavicle right	0.879	0.879	Rib right 7	0.672	0.624
Femur left	0.912	0.908	Rib right 8	0.657	0.633
Femur right	0.918	0.926	Rib right 9	0.581	0.656
Gallbladder	0.702	0.669	Rib right 10	0.630	0.692
Gluteus maximus left	0.902	0.920	Rib right 11	0.694	0.718
Gluteus maximus right	0.903	0.918	Rib right 12	0.638	0.673
Heart	0.868	0.891	Sacrum and coccyx	0.901	0.908
Hip bone left	0.903	0.909	Scapula left	0.839	0.846
Hip bone right	0.905	0.910	Scapula right	0.838	0.842
Humerus left	0.890	0.891	Skull	0.803	0.812
Humerus right	0.887	0.891	Spleen	0.895	0.892
Kidney left	0.866	0.882	Sternum	0.859	0.859
Kidney right	0.860	0.890	Vertebra cervical all	0.837	0.846
Liver	0.933	0.942	Vertebra lumbar 1	0.641	0.802
Lung left	0.938	0.949	Vertebra lumbar 2	0.746	0.782
Lung right	0.958	0.963	Vertebra lumbar 3	0.781	0.907
Mandible	0.801	0.808	Vertebra lumbar 4	0.758	0.786
Pancreas	0.544	0.550	Vertebra lumbar 5	0.776	0.777
Urinary bladder	0.777	0.766	Vertebra thoracic 1	0.726	0.813
Rib left 1	0.784	0.785	Vertebra thoracic 2	0.730	0.805
Rib left 2	0.772	0.793	Vertebra thoracic 3	0.728	0.793
Rib left 3	0.722	0.732	Vertebra thoracic 4	0.697	0.771
Rib left 4	0.706	0.728	Vertebra thoracic 5	0.681	0.748
Rib left 5	0.611	0.686	Vertebra thoracic 6	0.657	0.744
Rib left 6	0.636	0.675	Vertebra thoracic 7	0.679	0.687
Rib left 7	0.680	0.684	Vertebra thoracic 8	0.622	0.674
Rib left 8	0.651	0.658	Vertebra thoracic 9	0.565	0.596
Rib left 9	0.599	0.704	Vertebra thoracic 10	0.503	0.621
Rib left 10	0.657	0.735	Vertebra thoracic 11	0.566	0.720
Rib left 11	0.641	0.752	Vertebra thoracic 12	0.629	0.735
Rib left 12	0.648	0.674	**Mean for all classes**	**0.746**	**0.775**

Table 4: Instance metrics for the models that segment both ribs and vertebrae with discriminative loss.

	Validation data						Test data					
	Swin model			Slice model			Swin model			Slice model		
Class	**PQ**	**SQ**	**RQ**	**PQ**	**SQ**	**RQ**	**PQ**	**SQ**	**RQ**	**PQ**	**SQ**	**RQ**
Left ribs	0.605	0.770	0.785	0.520	0.753	0.691	0.557	0.784	0.710	0.404	0.749	0.539
Right ribs	0.584	0.769	0.760	0.565	0.766	0.738	0.532	0.784	0.679	0.390	0.752	0.518
Lumbar	0.845	0.882	0.957	0.762	0.848	0.900	0.813	0.864	0.941	0.650	0.806	0.807
Thoracic	0.796	0.833	0.956	0.708	0.790	0.896	0.794	0.825	0.962	0.645	0.756	0.854
All classes	**0.672**	**0.802**	**0.838**	**0.604**	**0.778**	**0.776**	**0.632**	**0.806**	**0.784**	**0.472**	**0.759**	**0.621**

Table 5: Instance metrics for the models that segment ribs with connected component labeling and vertebrae with discriminative loss.

	Validation data						Test data					
	Swin model			Slice model			Swin model			Slice model		
Class	**PQ**	**SQ**	**RQ**	**PQ**	**SQ**	**RQ**	**PQ**	**SQ**	**RQ**	**PQ**	**SQ**	**RQ**
Left ribs	0.766	0.796	0.962	0.747	0.797	0.937	0.692	0.807	0.917	0.738	0.794	0.929
Right ribs	0.756	0.796	0.951	0.767	0.805	0.954	0.692	0.793	0.873	0.669	0.793	0.844
Lumbar	0.822	0.879	0.935	0.745	0.848	0.879	0.808	0.871	0.928	0.684	0.809	0.845
Thoracic	0.789	0.816	0.967	0.749	0.797	0.940	0.800	0.812	0.985	0.734	0.787	0.932
All classes	**0.777**	**0.813**	**0.957**	**0.753**	**0.806**	**0.935**	**0.740**	**0.807**	**0.917**	**0.708**	**0.793**	**0.893**



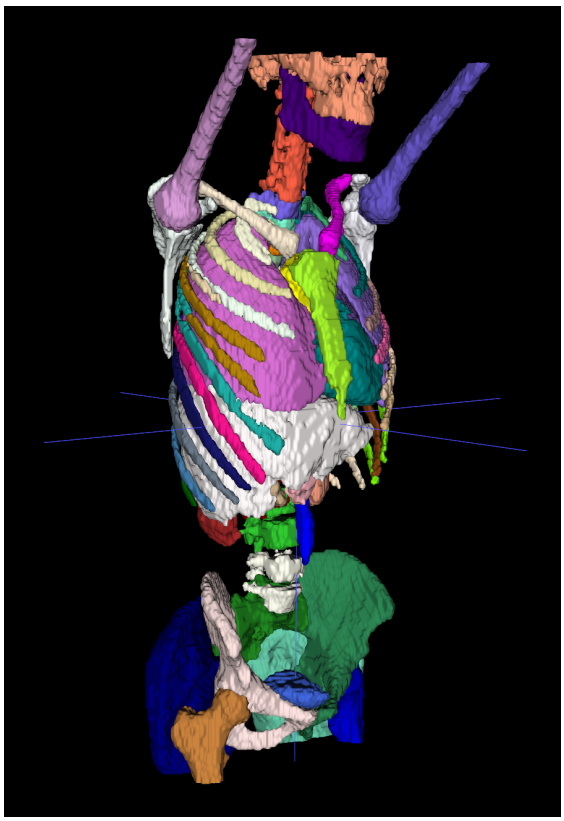
(a) CT image.

(b) Ground truth.

(c) Swin model.

(d) Slice model.

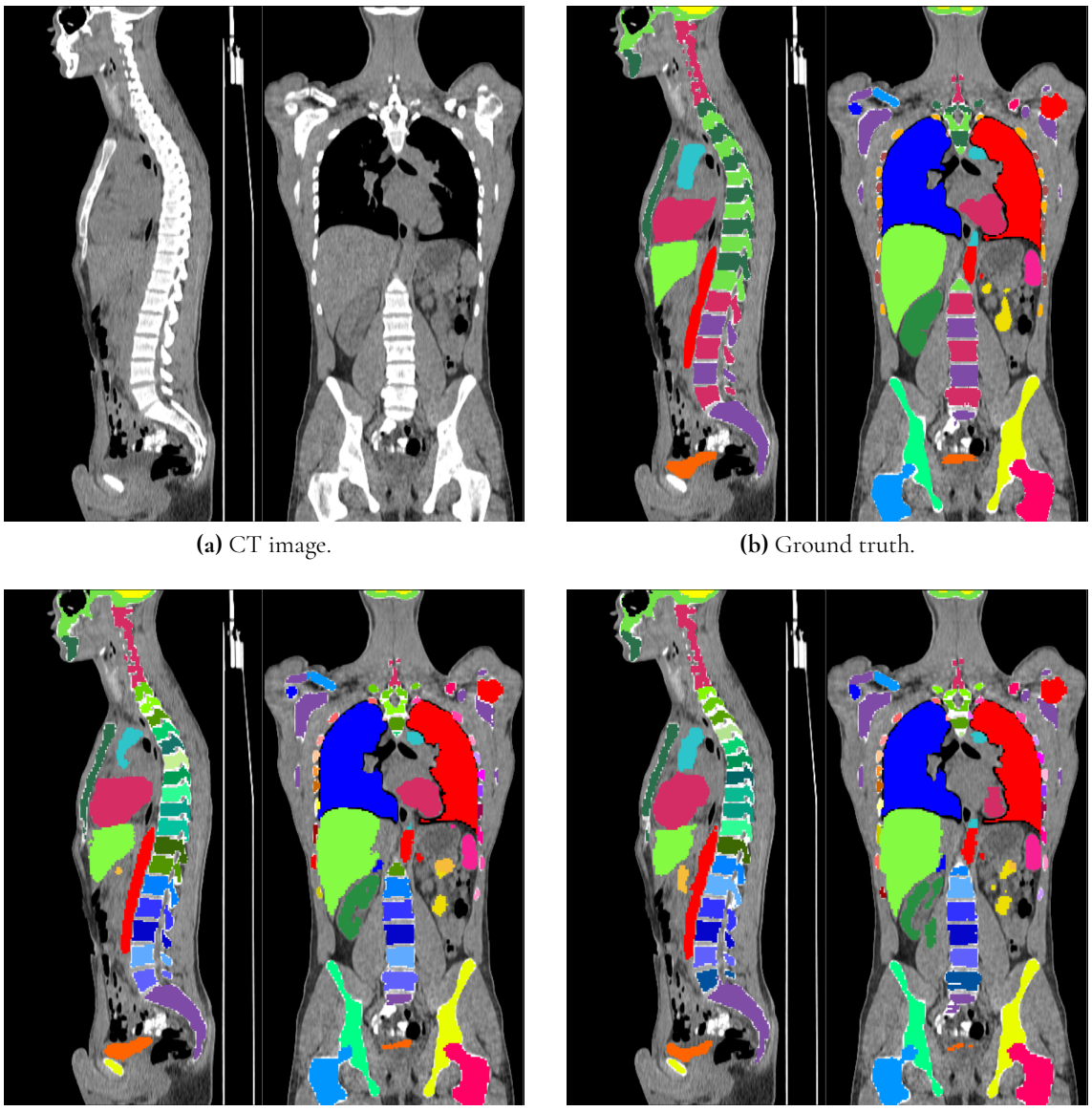
Figure 21: The segmentations of the models with only semantic segmentation on a patient from the validation dataset. Note that in this image, the pancreas was not annotated in the ground truth but was segmented by the network.



(a) Note the undersegmented ribs.

(b) Note the split ribs and vertebrae.

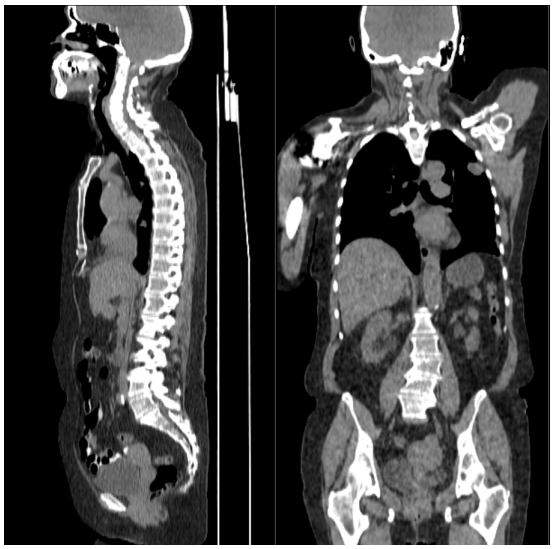
Figure 22: An example of when the semantic segmentation of ribs and vertebrae did not work well. The segmentation is from the Swin model. The holes visible in the scapulae are not due to undersegmentation of the model but rather due to how the 3D rendering works.



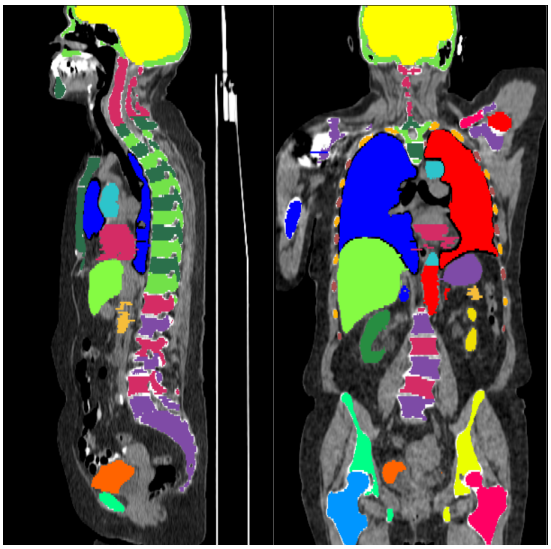
(a) CT image. **(b)** Ground truth.

(c) Swin model. **(d)** Slice model.

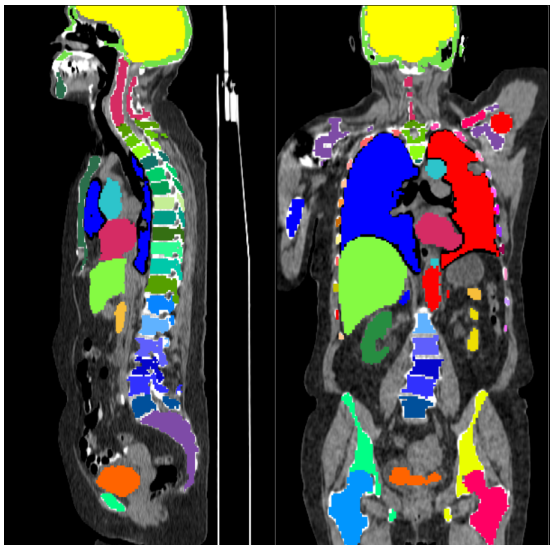
Figure 23: The segmentations of the models with instance segmentation with discriminative loss on both the ribs and the spine on a patient from the validation dataset. Note that in this image, the pancreas was not annotated in the ground truth but was segmented by the network.



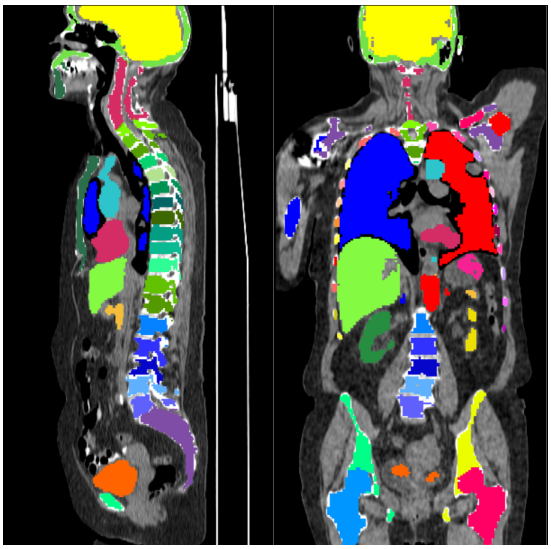
(a) CT image.



(b) Ground truth.



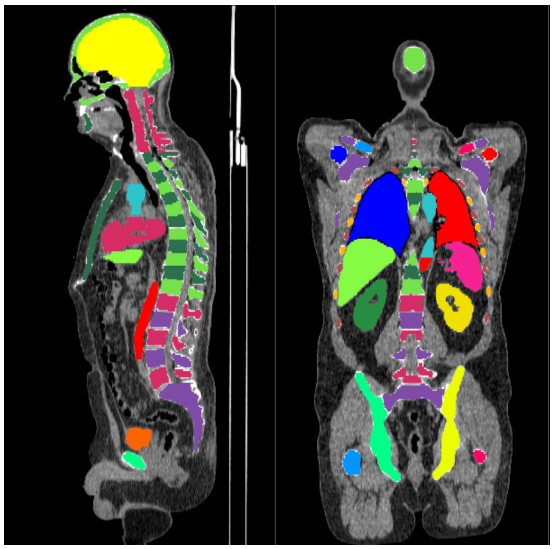
(c) Swin model.



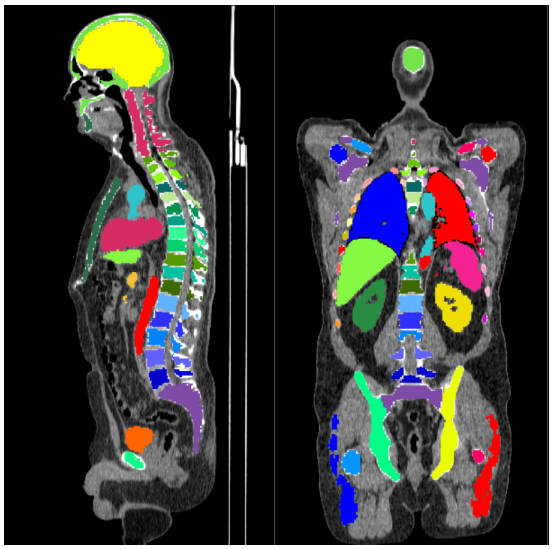
(d) Slice model.

Figure 24: The segmentations of the models with instance segmentation with discriminative loss on both the ribs and vertebrae on a patient from the validation dataset with 6 lumbar vertebrae. The stomach is segmented in the ground truth, an organ that our models were not trained to segment. It is the purple class under the left (red) lung. Also note that the slice model incorrectly segmented the topmost lumbar vertebra as a thoracic vertebra.

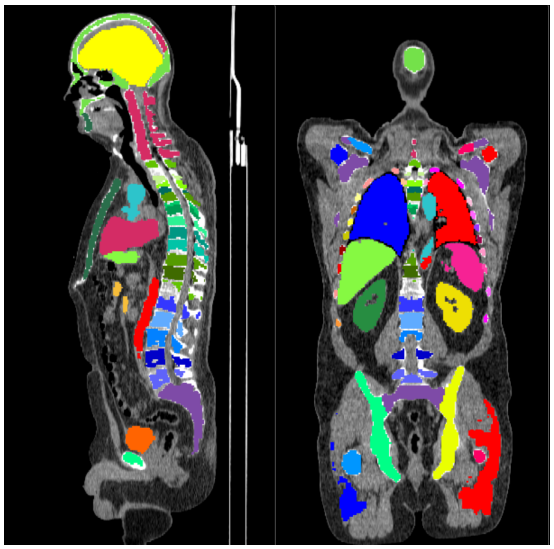
(a) CT image.



(b) Ground truth.



(c) Swin model.



(d) Slice model.

Figure 25: The segmentations of the models with instance segmentation with discriminative loss on both the ribs and the spine on a patient from the test dataset. Note that some classes are not labeled in the ground truth.

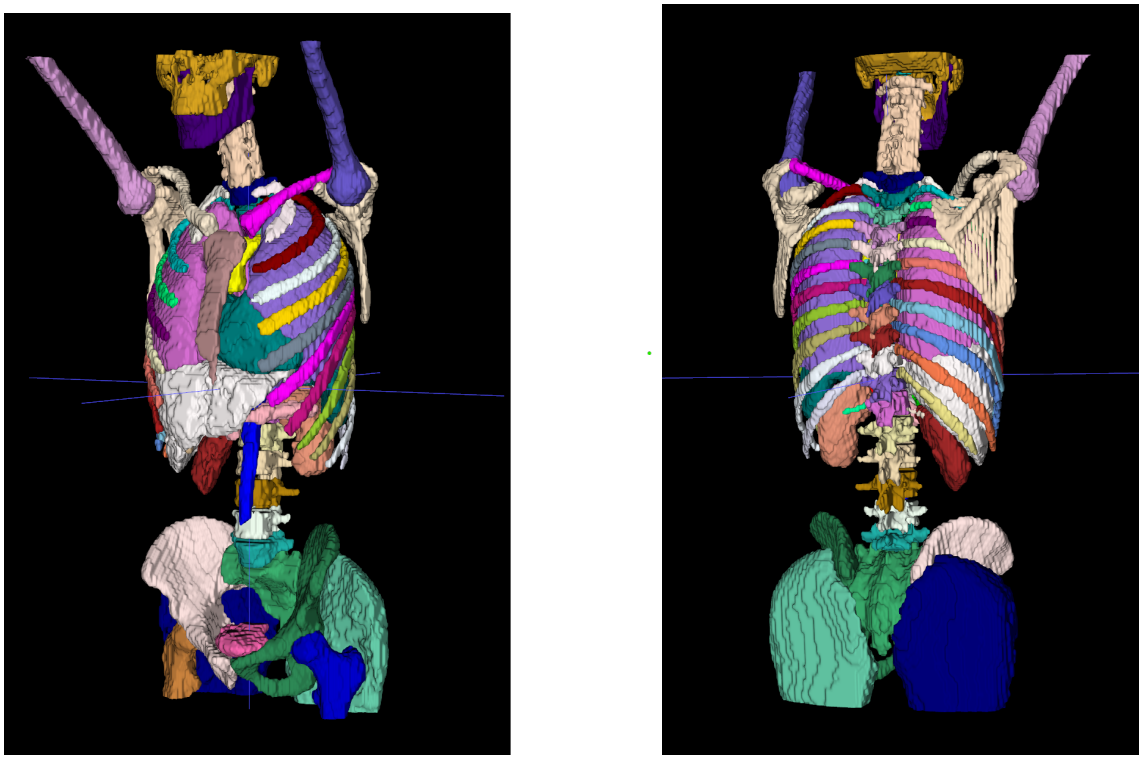


Figure 26: Example of how the instance segmentation of ribs and spine worked with discriminative loss. The segmentation shown is from the Swin model performing instance segmentation with discriminative loss on ribs and vertebrae. The holes visible in the scapulae are not due to undersegmentation of the model but rather due to how the 3D rendering works.

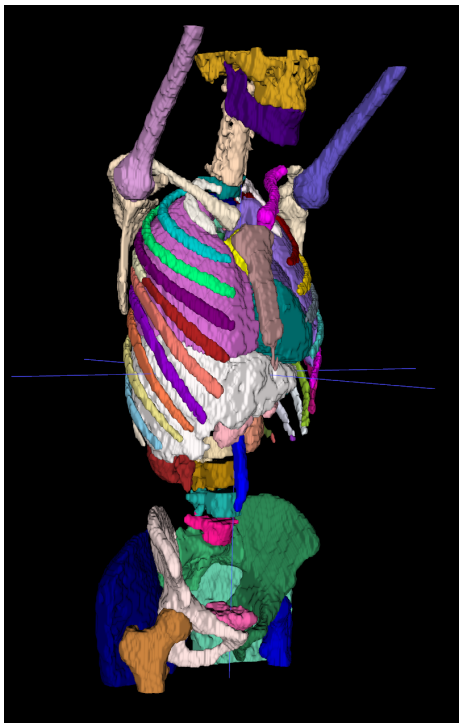
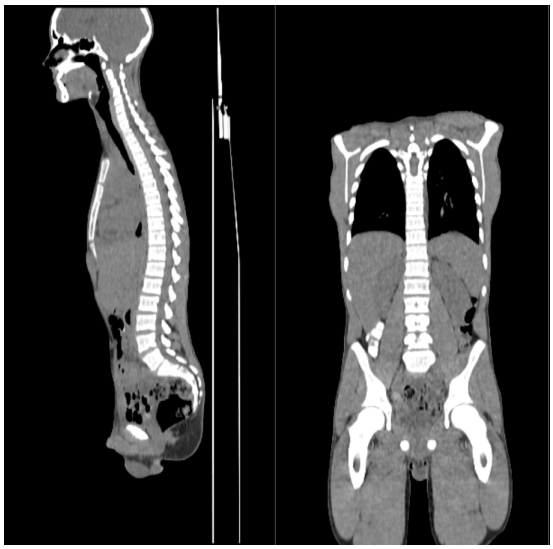
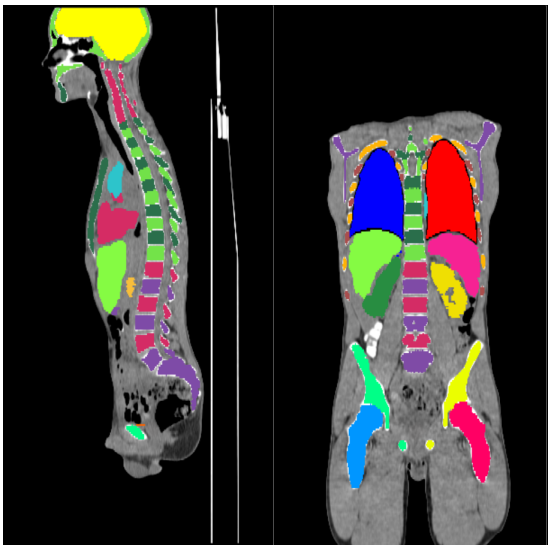


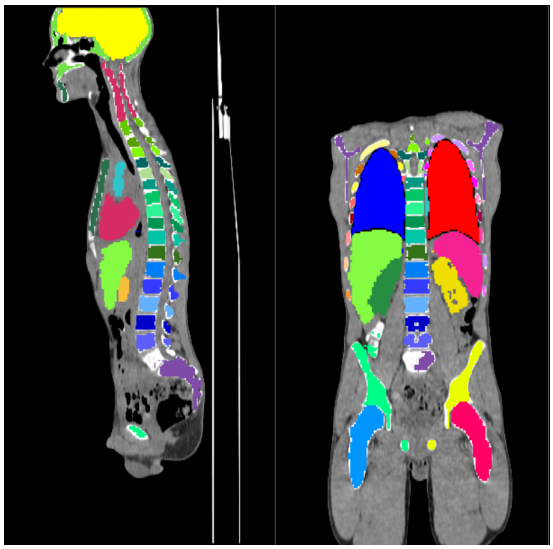
Figure 27: Example of a rib being split into multiple instances, see the red and beige rib. The segmentation shown is from the slice model performing instance segmentation with discriminative loss on ribs and vertebrae.



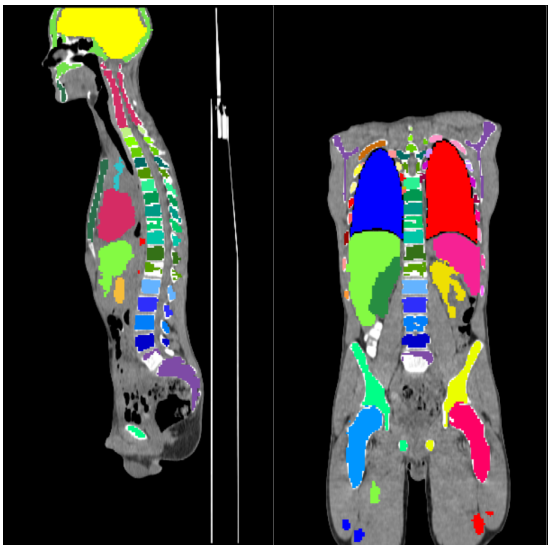
(a) CT image.



(b) Ground truth.



(c) Swin model.

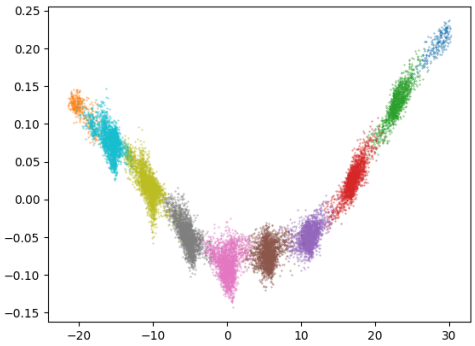


(d) Slice model.

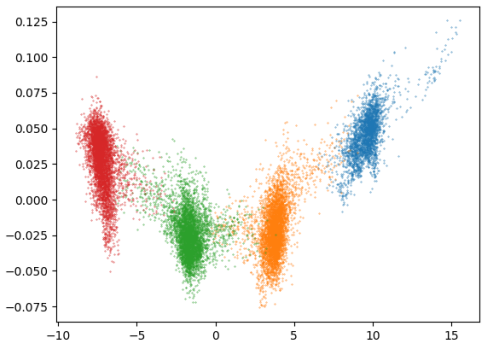
Figure 28: An example of where the presence of a sixth lumbar vertebra caused a poor segmentation of the lumbar vertebrae. The segmentations shown are from models performing instance segmentation with discriminative loss on ribs and vertebrae.

4.4 PCA of voxel embeddings for instance segmentation

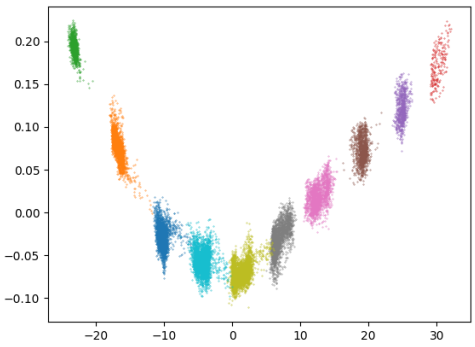
The voxel embeddings of an image in the validation set, outputted by the Swin and slice models with 32 and 16 embedding dimensions, respectively, and reduced to two principal components, are displayed in Figures 29 and 30. Note the different axis scales, indicating that the first principal component is much more significant than the second, especially for the Swin model. PCA to three dimensions (not shown here) revealed that the third principal component had a range roughly 10 times smaller than the second. Even in this dimension-reduced projection, the clusters are fairly well separated, and it is easy to see approximately where the boundaries between clusters should be placed, although perfect separation is not always possible, especially for the vertebrae.



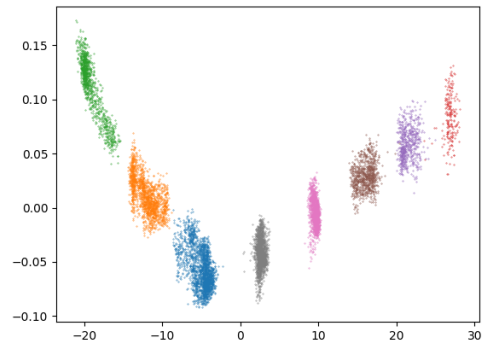
(a) Thoracic vertebrae.



(b) Lumbar vertebrae.

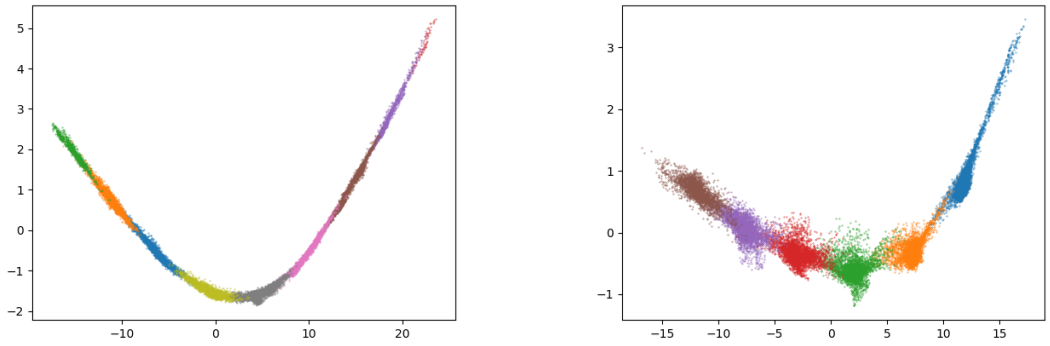


(c) Left side ribs.



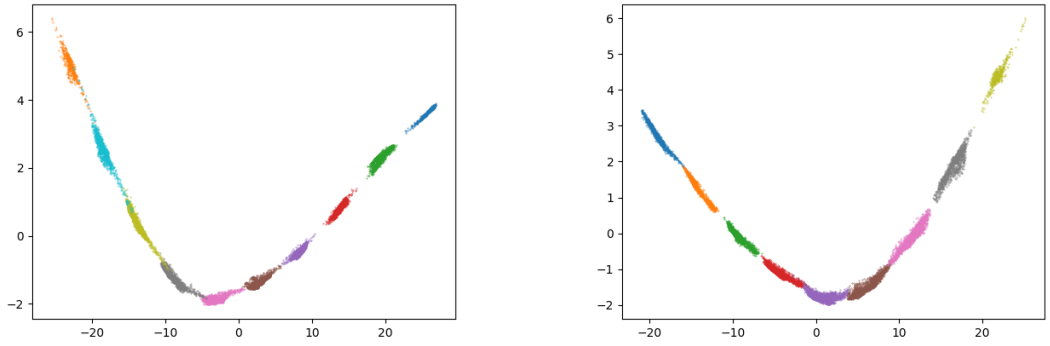
(d) Right side ribs.

Figure 29: The Swin model's voxel embeddings on ground truth, reduced to two dimensions with PCA. Each color corresponds to a different instance in the ground truth. The embeddings come from random blocks of 64 slices.



(a) Thoracic vertebrae.

(b) Lumbar vertebrae.



(c) Left side ribs.

(d) Right side ribs.

Figure 30: The slice model's voxel embeddings on ground truth, reduced to two dimensions with PCA. Each color corresponds to a different instance in the ground truth. The embeddings come from random blocks of 64 slices.

5 Discussion

The models we produced could segment arbitrary imaging ranges accurately, achieving high Dice scores on almost all semantic classes and good panoptic quality on the ribs and vertebrae when evaluated with sliding window inference on whole images. This allows the models to, for example, segment the thorax for a lung examination or the pelvis for a prostate examination. Accurate segmentations could be used to help clinicians when planning for surgery or dosimetry.

It is difficult to calculate any meaningful metrics on small fields of view, as the Dice and panoptic quality will vary depending on which organs are present in the image, and not all organs are equally easy to segment. However, when we evaluate on full-sized images, the network is still only allowed to look at one thin section of the image at a time, meaning that the performance is indicative of performance on smaller imaging ranges, with the added bonus that every organ is included equally many times. This, combined with the results of our visual evaluation on smaller images, shows that our models can accurately segment CT images of arbitrary imaging ranges.

Using models that incorporate attention mechanisms with vision transformers worked well but it is difficult to say exactly how the attention mechanisms affected the performance of the models compared to if they were non-transformer-based. We are, however, satisfied that we could create networks that perform both semantic and instance segmentation simultaneously with a rather simple implementation. The models are fairly lightweight, with the slice model being able to run inference on a whole-body CT scan in 10-15 seconds and the Swin model in 5-10 seconds on hardware with limited processing power and VRAM. The self-contained approach, where a single model performs the entire task, combined with the low hardware demands, means that our solution could be used to solve real-world problems, where complexity and hardware requirements are often limiting factors.

5.1 Semantic segmentation

We are satisfied with the results of the semantic segmentation of all organs except for the individual vertebrae and ribs. Some classes, like lungs and femurs, are large, have distinctive shapes, and have very high contrast with surrounding tissues. These classes were easy for the networks to segment, with Dice scores of over 0.9, except for the slice model on the test dataset.

The adrenal glands had low Dice scores, but since they are very small, a few incorrect voxels will have a large effect on the Dice score. In addition, they were difficult for us to manually locate in the CT images because of their small size and poor contrast with surrounding tissues. The fact that they are difficult to locate for (untrained) humans and that small mistakes

have large negative impacts on Dice scores leads us to conclude that the performance we achieved is reasonably good, given the difficulty of the problem.

The pancreas was also an outlier in terms of Dice score, with similar performance as the adrenal glands. We believe that this is because of its low contrast with surrounding tissues and irregular shape that varies significantly between patients and because roughly one-third of training images had no label for the pancreas. In addition, we observed that ground truth labels that we used for training and computing metrics were often poorly drawn for adrenal glands and pancreas. This means that not only was the network trained to make poor segmentations but also that the metrics computed for these organs cannot be trusted to any large degree. The performance on the urinary bladder was also noticeably worse than on most other classes. This can likely be explained by its varying size and HU value.

5.1.1 Ribs and vertebrae

We were not satisfied with the semantic segmentation of separate ribs and vertebrae, which is why we decided to implement instance segmentation. This decision was motivated by our belief that it would be better to have cleanly separated ribs and vertebrae without numbering than to have poorly drawn ribs and vertebrae with varyingly accurate numbering.

We think that the reason for the poor performance is that most vertebrae and ribs are almost identical to their neighbors. If the networks had been presented with whole-body images, they would have been able to 'count' the instances from top to bottom, or vice versa, to assign the correct label to each. When presented with only smaller slices, this strategy does not work, and the networks have to tell which is which based only on the shape of the ribs and vertebrae as well as their location in relation to close-lying tissues and organs. Since adjacent instances are similar in appearance and there is a large variability between patients in the location of the bones in relation to surrounding organs and tissues, this is a very difficult problem to solve. That roughly 10% of images in the training set have a non-standard number of vertebrae makes the problem even more difficult.

Despite the vertebrae being smeared together and the ribs split into multiple pieces, we are impressed with the Dice scores the networks could achieve. We believe that the transformer layers in the networks helped in this task since they enable the networks to take all other instances present in the image, as well as surrounding tissues and organs, into account when deciding which instance is which. For this reason, we hypothesize that the performance could be improved further if larger sections of the body could be inputted into the networks. In addition to a longer spatial attention range, larger windows also increase the likelihood that distinctive features are included in the image that can be used to tell which instance is which. Such features include the top and bottom of the spine and the transition between the thoracic spine and lumbar or cervical, which can be identified by the fact that only thoracic vertebrae have ribs attached to them. Such features can be used to determine a known point from where the network can 'count' ribs and vertebrae to correctly label all instances present in the image.

5.2 Instance segmentation

We chose to use instance segmentation with discriminative loss because we thought it would be relatively simple to implement and add to our models. Thus, we would not have to develop and train a new model just for separating ribs and vertebrae into instances. The paper introducing the discriminative loss function was not designed for CT images or even medical images, but we thought it was an interesting approach that we would be able to implement within the timeframe of this project, and that would allow us to perform both semantic and instance segmentation with a single model.

Overall, we are satisfied with the instance segmentation of ribs and vertebrae. The models can segment the instances in arbitrary axial ranges with better results than the semantic segmentation. The models sometimes have problems with the spine being under-segmented and a rib being split in two, but these are relatively small problems that could potentially be solved with further postprocessing.

Rib separation with connected component labeling gave similar SQ and much higher RQ than discriminative loss despite being a simpler approach, where the network has less to learn. Segmentation using connected component labeling will fail if the semantically segmented ribs touch, or if they are split into multiple parts. Both of these problems could potentially be solved by discriminative loss, but it would require a close-to-perfect clustering of the embeddings. Training a model to produce such embeddings was not possible for us, and therefore, the simpler and more explainable method of connected component labeling is a preferable approach.

Many other instance segmentation methods we researched were proposal-based, meaning they used networks trained to detect instances by placing bounding boxes around them. Then, a separate part of the network, or even an entirely separate network, would perform segmentation inside the bounding boxes. Proposal-based methods could potentially have problems with segmenting the ribs since their bounding boxes could overlap significantly, a problem that our approach does not have. In addition, the proposal-based methods were often more complex, multi-stage models. We are satisfied that we could create a network that performs both semantic segmentation and instance segmentation simultaneously, with a rather simple implementation.

5.2.1 Clustering of voxel embeddings

The clustering algorithm we used to turn voxel embeddings into instances was relatively simple. If the networks had learned to create perfect voxel embeddings where all points belonging to the same instance were located within δ_v of the cluster center, and all cluster centers had a distance of at least δ_d to each other, the clustering algorithm we used would have clustered the embedding space correctly, see Figure 17. However, since the models could not create perfect embeddings, our clustering algorithm often failed to accurately cluster the embedding space.

The clusters produced by the networks appeared to be non-circular in shape when projected to two dimensions, and were connected along a curved line. This means there were often points located between two cluster centers that were excluded from both clusters and assigned to their own small clusters. This resulted in under-segmented instances, as the resulting instances were removed for being too small. In the case of the vertebrae, these small

clusters were often located in the border regions between adjacent vertebrae, as well as in the vertebral arch.

5.2.2 Problems with segmenting ribs with discriminative loss function

As noted, the instance segmentation using discriminative loss performed poorly on the ribs, sometimes splitting one rib into multiple instances and sometimes marking the distal end of one rib and the proximal end of an adjacent rib as the same instance, as seen in Figure 27. It did not make the same type of mistake on the vertebrae. We speculate that this is because of the difference in shape between ribs and vertebrae. The vertebrae have compact shapes and do not extend far in any direction, whereas the ribs are long and thin. Since the two ends of each rib are located far from each other, it may be difficult for the models to make the connection that they are part of the same rib. The fact that the networks use the same weights to embed vertebrae as they do to embed ribs may also have an effect since, from the vertebrae, the models may learn that voxels close to each other in space belong to the same instance and voxels far apart belong to separate instances.

That rule does not apply to the ribs, which means the two tasks conflict with each other, and thus, weight sharing might not be appropriate. We chose to use weight sharing because we thought the two instance segmentation tasks were similar enough that weight sharing would be beneficial and because we thought that where the two tasks differed, it could use some of the embedding dimensions for one task and some for the other. It is possible that the instance segmentation could have been improved if the instance decoder branch of the networks had two embedding outputs, one for the vertebrae and one for the ribs.

5.3 Models

5.3.1 Slice model

We did not have time to evaluate how all parts of the model and their hyperparameters affected the performance of the models. However, we are satisfied with the final semantic and instance segmentation performance, and as a whole, the architectures for both the slice model and the Swin UNETR-based model worked well. The slice model performed worse on most classes for both semantic and instance segmentation, especially on the test data.

The idea behind using slices as patches is that we deemed it easier to extract meaningful information from an axial slice of the body than from a small three-dimensional patch, containing only a few grayscale values. Thus, by using slices, attention can be calculated between detailed representations of different pieces of the image. In addition, using slices as patches results in a one-dimensional sequence of embeddings, well suited to the sequential nature of transformers. We added the pre-embedding convolutions to the slice model to create rich feature maps for the slice embedding stage. We added this stage because we hypothesized that the single convolution of the embedding layer could not adequately capture all relevant information in a slice.

The slice embedding stage transforms the feature maps from the pre-embedding convolutions into points in a 1024-dimensional space. We hypothesized that it would be difficult for the model to upsample such a point into the original resolution of a slice, which contains 16384 pixels. To help the upsampling, we added the module that downsamples the original slices and concatenates them to the feature maps at various stages of the decoder. The idea is that the downsampled slices could help the network with the details of the image at each resolution. The downsampling is done with average pooling and thus only adds parameters to the extra channels of the kernels in the upsampling.

5.3.2 Comparison of the two models

On the test data, the slice model performance was inconsistent, with varying Dice scores on the semantic segmentation. However, the Swin model was consistently good on the test data, achieving Dice scores of above 0.8 on most organs. We do not know why the slice model was sometimes worse on the test data and sometimes not. We note that the validation data came from the same source as the training data, while the test data came from a different source. The test data was collected with different CT cameras with different image characteristics and from another patient population than the validation and training data. Thus, it is not unexpected that the performance could be worse on the test set than on the validation set. We also note that the test data does not contain the same amount of labeled classes, especially the most difficult classes (adrenal glands and pancreas) are missing. This could be the reason why the Swin model achieved better mean Dice scores on the test data. Overall, the Swin model seems to be a better model that consistently performs well, although more work is required to assess the slice model's performance.

5.4 Metrics and performance evaluation

That we perform the segmentation in low resolution but compute performance metrics using the high-resolution ground truth has an effect on Dice scores. Our low-resolution segmentations have choppy, pixelated edges in comparison to the smoother ground truth. This means that the overlap of segmentation and ground truth cannot match well around the surface of the segmentation. Therefore, classes that have a high surface-to-volume ratio may get significantly lower Dice score than they would have if the segmentation had been higher resolution. This includes organs and tissues that have thin structures, such as the skull, sternum, or scapulae, and small organs, such as adrenal glands. We noted that we achieved higher Dice scores on these organs when the metrics were computed with low-resolution ground truth.

We are happy with the Dice scores and panoptic quality we achieved. On most body parts, we achieved a Dice score above 0.8, which is very high. A Dice score of 1.0 is not a reasonable goal, as the ground truth is created by humans, who do not create perfect labels. In addition, due to limited resolution, noise, motion artifacts, etc., the exact borders of organs in the CT image are often ambiguous. A Dice score close to 1.0 means that the network made the same errors as the humans labeling the data and is not necessarily indicative of a well-trained network able to generalize well to new data. It is, therefore, difficult to make fair comparisons between different networks, especially if they have been evaluated on different data sets. At most, it is possible to state whether two networks have performance in the same ballpark.

In some setups, PQ suffered because of low RQ, although SQ was quite high, even on the ribs. This was because some ribs were split into several instances, and sometimes, two ribs were combined into one instance. This reduces RQ and PQ but is not necessarily a big problem for a physician, who would still be able to see the shape and location of the ribs, even though some have the same color and some are split into two colors. Therefore, we think that SQ is a more important metric in this case, and we are satisfied with the results.

5.4.1 Comparison with another model

A comparison with a model trained on similar data, presented in [38], reveals that our models had similar performance. That model was trained to segment a similar number of organs in CT images, also performing inference on a limited field of view and also using a custom solution for the vertebrae. The model achieved a mean Dice score of 0.88 on validation data and 0.90 on test data, although the test data only contained 10 labels. It struggled with the same organs our models struggled with, including adrenal glands and pancreas. It achieved a mean Dice of roughly 0.9 on vertebrae and ribs, although it performed semantic segmentation instead of instance segmentation on the vertebrae and ribs. It should also be noted that it was trained with roughly twice as much data as our models, and as stated, models evaluated on different datasets cannot be compared directly. Our models were significantly faster to run, running inference and postprocessing on a whole image in less than a minute, whereas the other model took two minutes on average, despite running on more powerful hardware.

5.5 Dice loss

We set the smoothing term in the numerator of the Dice loss to zero because we initially had problems with the networks not segmenting certain classes at all. The classes which it did not find differed for each training, but they were often small-volume classes. This problem occurred frequently when we performed purely semantic segmentation with 73 different classes. We believed this was because the networks would get stuck in a local minimum during optimization. At the beginning of the training, the overlap of the label and the predicted probabilities is random. If there is a smoothing term in the numerator and the overlap of a class is zero (or practically zero), the network will reduce the Dice loss by making a smaller prediction since the denominator will then have a smaller value. We hypothesized that this could be the reason why the network predicted that certain organs were not at all present in the images. Without the smoothing term in the numerator, the size of the prediction does not affect the loss if the overlap is zero. The network should thus not be rewarded for producing a smaller output when the overlap is zero. We hypothesized that setting the smoothing term to zero would thus reduce the risk of the network not segmenting some classes.

The segmentation seemed to improve without the smoothing term in the numerator, so we kept it for all models because it seemed to work better. However, we did not have enough time to do a systematic test of different smoothing terms for the final networks, so there might be a better combination of smoothing terms for the different networks.

Early during the project, we experimented with loss functions such as generalised Dice loss [39] and a Dice loss for missing or empty labels [40], both leading to poor performance. Because of our training method of randomly sampling small pieces of images, most labels

are missing from any given piece of training data. This creates problems for both those loss functions, as well as regular Dice loss, because if there is a smoothing factor in the numerator, the network will run into the problems described above. If there is no smoothing factor, the loss will be 0 for all missing labels, and the network will not learn anything at all because the gradients will be 0. Therefore, there is a need for a loss function for semantic segmentation better suited to data where most labels are missing in most training images.

5.6 Performance discussion

If we had a newer, more powerful GPU with more VRAM to train on, we could have used a larger model that could have performed better. We could also have used images of higher resolution, and larger windows. Larger windows could be helpful due to the network being allowed to include context from a larger region of the body. Higher resolution could also improve performance, particularly on the ribs when instance segmenting them with connected component labeling, where low resolution could cause two ribs to become connected if the gap between them is small.

On the other hand, the network's ability to run on limited hardware is a big advantage in terms of real-world usefulness. Running fast GPUs with large VRAM can be prohibitively expensive or impractical in many applications. In particular, hospitals that could potentially use medical devices incorporating this work may have limited computing resources, potentially running the model on a CPU only. Therefore, we are satisfied that we could create a network that can run on limited hardware.

5.7 Future work

We did not have enough time to test many combinations of hyperparameters, preprocessing, data augmentation, etc. Further tuning of hyperparameters and longer trainings with fine-tuning could improve the models' performance.

5.7.1 Dilating instance segmentation

The instance segmentation performed with discriminative loss divides all voxels that are semantically segmented as ribs or vertebrae into instances. We remove instances that are too small, which often leads to the spine being under-segmented, especially in the thinner posterior parts and at the edges between two vertebrae. This under-segmentation could potentially be fixed by dilating the instances until they fill the semantic segmentation since the semantic segmentation of the spine was consistently good. This would assign the label of the closest kept vertebra to every voxel that was removed while also making sure not to over-segment the vertebra by only dilating into the voxels marked as the corresponding class in the semantic segmentation. The same could be done for the ribs, but we did not find them to be as undersegmented.

5.7.2 Smarter clustering algorithms

Since it is easy to visually determine approximately where cluster boundaries should be placed even in two dimensions, it should be possible to separate the clusters better than our clustering algorithm did. There exist many variants of the K-means algorithm as well as many other clustering algorithms that are more advanced than the one we used [36, 41]. It is possible that another, more advanced algorithm that can adapt to the structure of the embedding space produced by our models could have resulted in better instance segmentation without changing or re-training the network.

5.7.3 Numbering of instances

In many imaging ranges, it could be possible to turn the instance segmentation into semantic segmentation through rules-based postprocessing by utilizing distinctive features of the spine and ribs. For example, if the transition between two classes of vertebrae, e.g. thoracic and lumbar, is visible in the imaging range, it would be possible to assign numbers to the instances based on their order in the image in relation to the point of transition. The same type of label assignment could be used if the top or bottom of the spine is visible in the imaging range. If no such distinctive features are visible, it could still be possible to infer which vertebra or rib is which through more complicated rules, but these rules could become very complex, and it may be better to use a machine learning algorithm to assign labels. This would partially defeat the purpose of our single-model approach, but perhaps a smaller, more explainable machine learning model like a decision tree could work well in this application.

6 Conclusions

The goal of this project was to develop a model that could segment tissues and organs in CT images of any imaging range using transformer-based deep learning models. Both the Swin model and the slice model performed well in this task, although the Swin model appears to generalize better to new data than the slice model. Our method of training with randomly sampled sections of whole-body images and using sliding window inference on larger images allows us to segment arbitrary imaging ranges as long as they are longer than 19.2 cm in height.

Semantically segmenting every vertebra and rib into separate classes is difficult when only a small region of the body has been imaged. Neither model could give satisfactory performance at this task. However, performing instance segmentation of these bones using a discriminative loss function allows a single network to perform both the semantic and instance segmentation tasks with good results. The model that performed the best was the Swin model using connected components labeling to separate the ribs into instances.

References

[1] Saeid Asgari Taghanaki et al. "Deep semantic segmentation of natural and medical images: a review". In: *Artificial Intelligence Review* 54.1 (Jan. 2021), pp. 137–178. ISSN: 1573-7462. DOI: 10.1007/s10462-020-09854-1.

[2] Anne I J Arens et al. "FDG-PET/CT Limited to the Thorax and Upper Abdomen for Staging and Management of Lung Cancer". en. In: *PLoS One* 11.8 (Aug. 2016), e0160539.

[3] Ronald Boellaard et al. "FDG PET/CT: EANM procedure guidelines for tumour imaging: version 2.0". en. In: *Eur J Nucl Med Mol Imaging* 42.2 (Dec. 2014), pp. 328–354.

[4] Wolfgang P Fendler et al. "PSMA PET/CT: joint EANM procedure guideline/SNMMI procedure standard for prostate cancer imaging 2.0". en. In: *Eur J Nucl Med Mol Imaging* 50.5 (Jan. 2023), pp. 1466–1486.

[5] Ali Hatamizadeh et al. *Swin UNETR: Swin Transformers for Semantic Segmentation of Brain Tumors in MRI Images*. 2022. DOI: https://doi.org/10.48550/arXiv.2201.01266. arXiv: 2201.01266 [eess.IV].

[6] Shijie Hao, Yuan Zhou, and Yanrong Guo. "A Brief Survey on Semantic Segmentation with Deep Learning". In: *Neurocomputing* 406 (2020), pp. 302–321. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.11.118.

[7] Alexander Kirillov et al. *Panoptic Segmentation*. 2019. DOI: https://doi.org/10.48550/arXiv.1801.00868. arXiv: 1801.00868 [cs.CV].

[8] Michael Young Shady Hermena. "CT-scan Image Production Procedures". In: *StatPearls* (2023).

[9] Joseph D. Bronzino. "Chapter 15 - Radiation Imaging". In: *Introduction to Biomedical Engineering (Third Edition)*. Ed. by John D. Enderle and Joseph D. Bronzino. Third Edition. Biomedical Engineering. Boston: Academic Press, 2012, pp. 995–1038. ISBN: 978-0-12-374979-6.

[10] Tami D DenOtter and Johanna Schubert. "Hounsfield Unit". en. In: *StatPearls*. Treasure Island (FL): StatPearls Publishing, Jan. 2024.

[11] Greenway K et al. "Hounsfield unit". In: *Radiopaedia.org* (2015). DOI: https://doi.org/10.53347/rID-38181.

[12] Gulgun Kayalioglu. "Chapter 3 - The Vertebral Column and Spinal Meninges". In: *The Spinal Cord*. Ed. by Charles Watson, George Paxinos, and Gulgun Kayalioglu. San Diego: Academic Press, 2009, pp. 17–36. ISBN: 978-0-12-374247-6. DOI: https://doi.org/10.1016/B978-0-12-374247-6.50007-9. URL: https://www.sciencedirect.com/science/article/pii/B9780123742476500079.

[13] Ying-Zhao Yan et al. “Rate of presence of 11 thoracic vertebrae and 6 lumbar vertebrae in asymptomatic Chinese adult volunteers”. en. In: *J Orthop Surg Res* 13.1 (May 2018), p. 124.

[14] D Dominguez et al. “Normative values for the L5 incidence in a subgroup of transitional anomalies extracted from 147 asymptomatic subjects”. en. In: *Eur Spine J* 25.11 (Jan. 2016), pp. 3602–3607.

[15] Kunio Yokoyama et al. “Spinopelvic alignment and sagittal balance of asymptomatic adults with 6 lumbar vertebrae”. en. In: *Eur Spine J* 25.11 (Oct. 2015), pp. 3583–3588.

[16] Niladri Kumar Mahato. “Morphological traits in sacra associated with complete and partial lumbarization of first sacral segment”. In: *The Spine Journal* 10.10 (2010), pp. 910–915. ISSN: 1529-9430. DOI: https://doi.org/10.1016/j.spinee.2010.07.392. URL: https://www.sciencedirect.com/science/article/pii/S1529943010009459.

[17] Wikipedia user DrJanaOfficial. *Segments of Vertebrae*. https://commons.wikimedia.org/wiki/File:Segments_of_Vertebrae.svg. License: CC BY-SA 4.0, Accessed: 2024-05-28. 2020.

[18] Wikipedia user Jmarchn. https://commons.wikimedia.org/wiki/File:718_Vertebra-en.svg. License: CC BY-SA 3.0, Accessed: 2024-05-28. Image was edited by us. This version is licensed under CC BY-SA 3.0 by Morris Thånell and Petter Melander. 2015.

[19] Andreas Lindholm et al. *MACHINE LEARNING, A First Course for Engineers and Scientists*. Cambridge University Press, 2022.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[21] Andrinandrasana David Rasamoelina, Fouzia Adjailia, and Peter Sinčák. “A Review of Activation Function for Artificial Neural Network”. In: *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*. 2020, pp. 281–286. DOI: 10.1109/SAMI48414.2020.9108717.

[22] Andrew L. Maas. “Rectifier Nonlinearities Improve Neural Network Acoustic Models”. In: 2013. URL: https://api.semanticscholar.org/CorpusID:16489696.

[23] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2023. arXiv: 1606.08415 [cs.LG].

[24] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. DOI: https://doi.org/10.48550/arXiv.1711.05101. arXiv: 1711.05101 [cs.LG].

[25] Jeroen Bertels et al. “Optimizing the Dice Score and Jaccard Index for Medical Image Segmentation: Theory and Practice”. In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*. Ed. by Dinggang Shen et al. Cham: Springer International Publishing, 2019, pp. 92–100. ISBN: 978-3-030-32245-8.

[26] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. “V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation”. In: *2016 Fourth International Conference on 3D Vision (3DV)*. 2016, pp. 565–571. DOI: 10.1109/3DV.2016.79.

[27] Adrian Galdran, Gustavo Carneiro, and Miguel Ángel González Ballester. *On the Optimal Combination of Cross-Entropy and Soft Dice Losses for Lesion Segmentation with Out-of-Distribution Robustness*. 2022. DOI: https://doi.org/10.48550/arXiv.2209.06078. arXiv: 2209.06078 [cs.CV].

[28] Ronneberger, Olaf and Fischer, Philipp and Brox, Thomas. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: (2015). Ed. by Nassir Navab et al., pp. 234–241. DOI: https://doi.org/10.1007/978-3-319-24574-4_28.

[29] Derya Soydaner. "Attention mechanism in neural networks: where it comes and where it goes". In: *Neural Computing and Applications* 34.16 (May 2022), pp. 13371–13385. ISSN: 1433-3058. DOI: 10.1007/s00521-022-07366-3. URL: http://dx.doi.org/10.1007/s00521-022-07366-3.

[30] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[31] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2020). DOI: https://doi.org/10.48550/arXiv.2010.11929. arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929.

[32] Ali Hatamizadeh et al. *UNETR: Transformers for 3D Medical Image Segmentation*. 2021. DOI: https://doi.org/10.48550/arXiv.2103.10504. arXiv: 2103.10504 [eess.IV].

[33] Ze Liu et al. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. Official implementation: https://github.com/microsoft/Swin-Transformer. 2021. DOI: https://doi.org/10.48550/arXiv.2103.14030. arXiv: 2103.14030 [cs.CV].

[34] Yufan He et al. "SwinUNETR-V2: Stronger Swin Transformers with Stagewise Convolutions for 3D Medical Image Segmentation". In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2023*. Ed. by Hayit Greenspan et al. Cham: Springer Nature Switzerland, 2023, pp. 416–426. ISBN: 978-3-031-43901-8.

[35] Bert De Brabandere, Davy Neven, and Luc Van Gool. *Semantic Instance Segmentation with a Discriminative Loss Function*. 2017. DOI: https://doi.org/10.48550/arXiv.1708.02551. arXiv: 1708.02551 [cs.CV].

[36] Abiodun M. Ikotun et al. "K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data". In: *Information Sciences* 622 (2023), pp. 178–210. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2022.11.139. URL: https://www.sciencedirect.com/science/article/pii/S0020025522014633.

[37] PyTorch Contributors. *CUDA Automatic Mixed Precision examples*. 2023. URL: https://pytorch.org/docs/stable/notes/amp_examples.html#gradient-accumulation.

[38] Elin Trägårdh et al. "RECOMIA-a cloud-based platform for artificial intelligence research in nuclear medicine and radiology". en. In: *EJNMMI Phys* 7.1 (Aug. 2020), p. 51.

[39] Carole H. Sudre et al. “Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations”. In: *Lecture Notes in Computer Science*. Springer International Publishing, 2017, pp. 240–248. ISBN: 9783319675589. DOI: 10.1007/978-3-319-67558-9_28. URL: http://dx.doi.org/10.1007/978-3-319-67558-9_28.

[40] Sofie Tilborghs et al. “The Dice Loss in the Context of Missing or Empty Labels: Introducing Φ and ϵ”. In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2022*. Springer Nature Switzerland, 2022, pp. 527–537. ISBN: 9783031164439. DOI: 10.1007/978-3-031-16443-9_51. URL: http://dx.doi.org/10.1007/978-3-031-16443-9_51.

[41] Absalom E. Ezugwu et al. “A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects”. In: *Engineering Applications of Artificial Intelligence* 110 (2022), p. 104743. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2022.104743. URL: https://www.sciencedirect.com/science/article/pii/S095219762200046X.

Appendices

A Organ counts

Table 6: Organ label counts in training and validation sets.

Organ	Count
Adrenal gland right	170
Adrenal gland left	170
Aorta, abdominal part	178
Aorta, thoracic part	178
Brain	178
Clavicle left	178
Clavicle right	178
Femur left	178
Femur right	178
Gallbladder	168
Gluteus maximus left	178
Gluteus maximus right	178
Heart	178
Hip bone left	178
Hip bone right	178
Humerus left	178
Humerus right	178
Kidney left	176
Kidney right	174
Liver	178
Lung left	178
Lung right	178
Mandible	178
Pancreas	133
Urinary bladder	178
Rib left 1	178
Rib left 2	178
Rib left 3	178
Rib left 4	178
Rib left 5	178
Rib left 6	178
Rib left 7	178
Rib left 8	178
Rib left 9	178
Rib left 10	178
Rib left 11	178
Rib left 12	178

Organ	Count
Rib right 1	178
Rib right 2	178
Rib right 3	178
Rib right 4	178
Rib right 5	178
Rib right 6	178
Rib right 7	178
Rib right 8	178
Rib right 9	178
Rib right 10	178
Rib right 11	178
Rib right 12	178
Sacrum and coccyx	178
Scapula left	178
Scapula right	178
Skull	178
Spleen	172
Sternum	178
Vertebra cervical all	178
Vertebra lumbar 1	178
Vertebra lumbar 2	178
Vertebra lumbar 3	178
Vertebra lumbar 4	178
Vertebra lumbar 5	178
Vertebra lumbar 6	19
Vertebra thoracic 1	178
Vertebra thoracic 2	178
Vertebra thoracic 3	178
Vertebra thoracic 4	178
Vertebra thoracic 5	178
Vertebra thoracic 6	178
Vertebra thoracic 7	178
Vertebra thoracic 8	178
Vertebra thoracic 9	178
Vertebra thoracic 10	178
Vertebra thoracic 11	178
Vertebra thoracic 12	178

Master's Theses in Mathematical Sciences 2024:E46
ISSN 1404-6342

LUTFMA-3546-2024

Computer Vision and Machine Learning
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/