VAE-based CT-Scan Segmentation

Detecting and Outlining Tumors in Lung Tissue

Nicklas Vraa (au592379) - 11. Dec. 2022

Abstract: Based on my findings during the course of this project, which was carried out in collaboration with David Felsager (au649158), I conclude that VAE-based models are indeed a viable option for image segmentation, based on the fact that our trained models successfully segmented tumors in CT-scans of lungs of patients, with a reasonable degree of accuracy. The full project, including trained models, is publicly available on GitHub. Our notebooks are well-documented, and are meant for allowing reproducibility of our results. One needs a copy of the used dataset, which is linked in the repository. Independent performance gauging is also possible, if one has their own dataset. I encourage the reader to exactly this and report their findings on the project's GitHub-page.

Repository link: VAE-based Segmentation

1 Introduction

In this project, I test the viability of using an extended variational autoencoder for semantic segmentation of images, specifically computed tomography (CT) scans of the human body, for the purpose of identifying and outlining cancerous tissue. I focus on CT-scans, where the resolution of the lung-structure of the patients have been enhanced via ingestion of a contrasting medium. I also explore the possibility of applying the model to other organs in the body.

1.1 Motivation

Automated semantic image segmentation is an active field of research in deep learning and one very important application of this technology is found in the field of medical imaging and diagnostics, such as determining where in an organ, there exists cancerous tissue, i.e. tumors.

This is traditionally done by having a highly specialized doctor examine the CT-scans of the patient in question, which is a time-consuming and laborious task all about pattern-recognition, in which deep learning models - like convolutional neural networks - excel.

Recently, the U-Net architecture has gained popularity for medical image segmentation. U-Net has the benefit of performing well on tasks, where labelled data is scarce. This is due to its encoder-decoder architecture. One problem with U-Net is that it uses skip-connections to fast-track low-level information from encoder to decoder. One could argue that this may lead to a loss of information in the compressed representation, i.e. the latent space of the model, which can result in the final model not being a good generalizer. A general model is exactly what is being requested by the medical community, which posed the medical segmentation challenge from which we have our dataset. The reason being that patients vary greatly in terms of size and build and medical data is notoriously hard come by, in part due to privacy concerns.

1.2 The Dataset

Our dataset is publicly available as part of a medical segmentation challenge. It contains 96 3D CT-scans, each comprised of 2D slices which adds up to 17657 slices. The lung-structure is contrasted, which in 1 is seen as the central, darker structure. The dataset comes with labels in the form of manually performed segmentation of each slice, performed by an expert. The data is provided raw, so we defined a routine to load in CT-images-slices in PyTorch and normalize every pixel-value between 0 and 1. Due to limits in hardware, we also scaled down each image to 256x256 pixels.

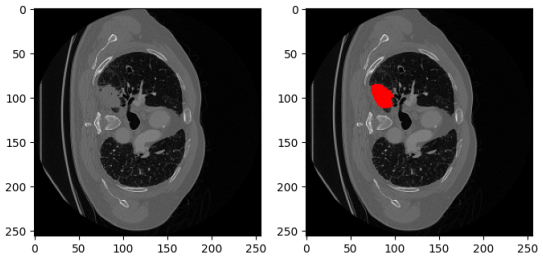

Figure 1: A single preprocessed slice paired with itself, but superimposed with its label. Red indicates cancerous tissue.

2 Theory

Before designing our segmentation solution, we must first establish a theoretical basis to build upon. This section describes the variational autoencoder, and how it achieves surprisingly robust performance using a very simple approach.

2.1 The Variational Autoencoder

Variational autoencoders are probabilistic generative models which build on its predecessor, the autoencoder, but rather than representing each latent attribute using discrete values, it encodes each latent attribute for a given input as a Gaussian distribution over the latent space. The encoder should learn to output μ and σ. The decoder can then sample a feature vector z from that distribution. This sampling is referred to as reparameterization, and follows equation 1:

$$z = \mu + \sigma\epsilon \text{ where } \epsilon = N(0, I_{z\times z}) \tag{1}$$

Naturally, the latent space should be filled, which is done by pushing the learned distributions to fill the space. This is called the Kullback-Leibler method, and is analogous to smearing out a Gaussian. This creates gradients from property to property.

$$D_{KL} = \sum_{i=1}^{n} \sigma_i^2 + \mu_i^2 - \log(\sigma_i) - 1 \tag{2}$$

This gives rise to a new loss-measure, which is referred to as the Kullback-Leibler divergence (KLd).

$$KL(P||Q) = \sum_x P(x) \log\left(\frac{P(x)}{Q(x)}\right) \tag{3}$$

It is a measure of how one probability distribution differs from another. KL-divergence says that: If we use distribution P to predict the behavior of a random process, and the process actually happens with distribution Q, how much worse did we do by using P as an estimate vs. if we had access to Q directly?

We use KLd summed with a reconstruction loss to define a new loss-function, which in the literature is called ELBO (evidence-lower-bound). The reconstruction loss is often chosen to be the mean-square-error (MSE) or cross-entropy (BCE in our case).

3 Methodology

This section explains how we planned to use a variational autoencoder for image segmentation, as this is not its usual purpose. However, autoencoders are known for being good anomaly-detectors, so it seems only natural to have it search for anomalies, e.g. tumors.

As seen in figure 2, we will first be training a slightly non-standard autoencoder to be able to compress images into the lower-dimensional latent space, from which it should be able to reconstruct the input image. At this stage, the images are their own labels. When the reconstructions reach a satisfactory degree of precision, we switch off the decoder and instead pass the latent space vector to a new decoder. At this stage, the labels are introduced to train this new decoder to instead reconstruct the label, i.e. a segmentation mask instead of the original image.

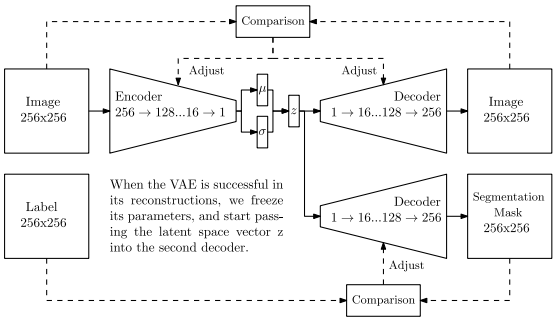

Figure 2: Conceptual structure of our model.

Exactly how our variational autoencoder is "non-standard", will become clear in the following section on implementation.

4 Implementation

This section explains how we implemented our designs using the PyTorch library. The implementation-process was an iterative approach, and many practical obstacles had to be overcome to reach our final models (VAE and segmentation-decoder). This section also outlines our training metrics, along with my observations on various aspects, which deviated from what could be expected, given the established literature on the subject of variational autoencoders. I encourage the reader to read our interactive notebooks, which are provided in the accompanying GitHub repository, as they walk through the entire development process.

4.1 Model Architecture

The architecture of our variational encoder consists of an encoder and a decoder, each built of mainly two blocks, I shall refer to as the Conv-block and the ConvTranspose-block.

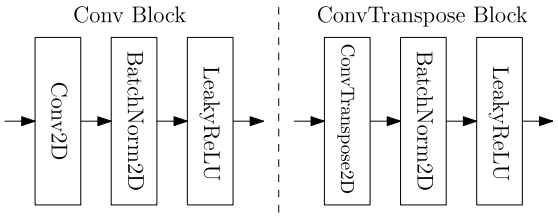

Figure 3: Basic building block of our VAE.

The Conv-block is simply a 2D convolution, followed by a batch normalization, followed by a leaky ReLU activation. For the ConvTranspose-block, we simply swap the convolution with its transpose equivalent, as seen in figure 3. Most convolutions use a 3x3 kernel, a padding of 1, and alternate between a stride of 2 and 1.

The encoder is a cascade of these blocks, where the output dimensions (width and height) are continuously decreased, while the number of channels increase, until we have a 1024-channel, 1-row, 1-column tensor, i.e. a vector. From this vector, we have a fully connected layers (implemented using convolution for speed), which will draw out two vectors of length 512, that represent μ and σ. These are then reparameterized into the vector z using eq. 1, which is our latent space representation of the input image. Note that the information within the image has been compressed $256^2/512 = 128$ times. The architecture of the encoder is summarized in table 1.

Lay. Type	Output Shape	Params
Conv	[-1,16,128,128]	144+32
Conv	[-1,32,128,128]	4608+64
Conv	[-1,32,64,64]	9216+64
Conv	[-1,32,64,64]	9216+64
Conv	[-1,32,32,32]	9216+64
Conv	[-1,64,32,32]	18432+128
Conv	[-1,64,16,16]	36864+128
Conv	[-1,64,16,16]	36864+128
Conv	[-1,64,8,8]	36864+128
Conv2d	[-1,1024,1,1]	4195328
LeakyReLU	[-1,1024,1,1]	0

Table 1: Simplified encoder summary. For parameters, the smaller number is contributed by the batch normalization.

The decoder effectively reverses what the decoder has carried out, so naturally the structure looks like its mirror image. The architecture of the decoder is summarized in table 2.

Lay. Type	Output Shape	Params
Conv2d	[-1,1024,1,1]	525312
ConvT	[-1,64,8,8]	4194304+128
Conv	[-1,64,8,8]	36864+128
ConvT	[-1,64,16,16]	65536+128
Conv	[-1,64,16,16]	36864+128
ConvT	[-1,64,32,32]	65536+128
Conv	[-1,32,32,32]	18432+64
ConvT	[-1,32,64,64]	16384+64
Conv	[-1,32,64,64]	9216+64
ConvT	[-1,32,128,128]	16384+64
Conv	[-1,16,128,128]	4608+32
ConvT	[-1,16,256,256]	4096+32
Conv2d	[-1,1,256,256]	145
Sigmoid	[-1,1,256,256]	0

Table 2: Simplified decoder summary. ConvT is short for ConvTranspose.

As is evident from the summaries, we ended up including intermediary Conv layers - which do not compress - at every other layer. We found this to improve accuracy, at the expense of training time and a larger model footprint. These may not be necessary, given more time to train and tune hyper-parameters. The second decoder that does segmentation is identical to first decoder. Final parameters from the VAE-decoder were used to set the initial parameters of the other, i.e. transfer learning. For all the nuances of the models, please see 'models.py' in the repository.

4.2 Training

The VAE and segmentation-decoder were trained separately. As our dataset was very large, and our hardware limited, we utilized checkpointing to complete the full training-runs in chunks. Figure 4 shows the loss-curves of the last 45 epochs of training for our VAE-model. In total, it was trained for 70 epochs, which took around two hours. The losses for the first 25 epochs were unfortunately lost, but exhibited the same exponentially decaying loss. The variance, i.e. the difference in the loss for the training- and development-set, may look large, but this is due to the compressed loss-axis. However, there is some overfitting happening. It is also evident from the curve, that the model may not even have had time to converge fully.

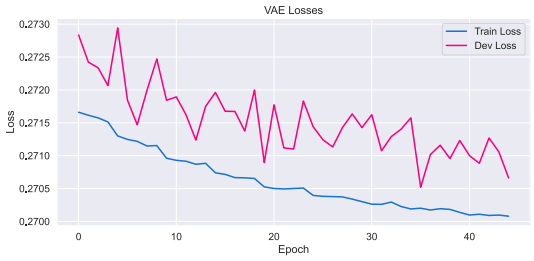

Figure 4: Loss-curves of both the training- and development-set, during training of the VAE.

Figure 5 show the loss-curve for the full training-run of our segmentation decoder. As the model is around half the size of the VAE, the training was done twice as fast. The model seems to generalize well, as the variance is low.

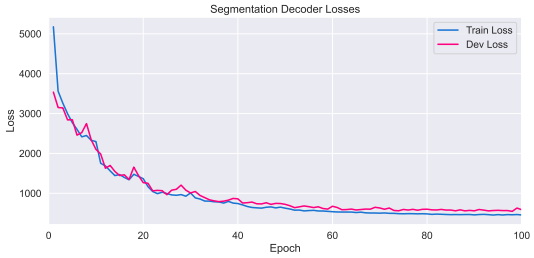

Figure 5: Loss-curves of both the training- and development-set, during training the segmentation decoder.

In the literature on the subject of VAE's, it is recommended that one sums the contribution of the reconstruction loss and KLd. In development, we found that we had to scale down KLd. Otherwise, the reconstruction quality suffered, i.e. blurry images, as seen in figure 6 (second set). This is due to the Kullback-Leibler method essentially smearing out our Gaussian distributions, as is described in eq. 2. Figure 6 also shows how using MSE as the reconstruction loss resulted in the model outputting the same reconstructed image (first set), regardless of input.

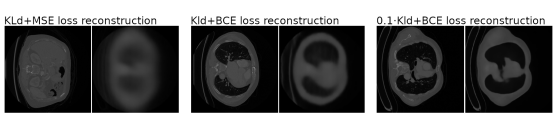

Figure 6: Comparing loss functions.

In conclusion, we scaled KLd by 0.1 and used binary-cross-entropy for the reconstruction loss, which resulted in a nice reconstruction (third set).

5 Results

Figure 7 shows a set of slices, each superimposed with their label in green and our model's prediction in red. Batches were taking randomly from our testing-set, which the model had not previously seen. I encourage the reader to run our notebooks to get an unbiased view of the performance. On the project repository, I have also provided a GIF showcasing real-time performance.

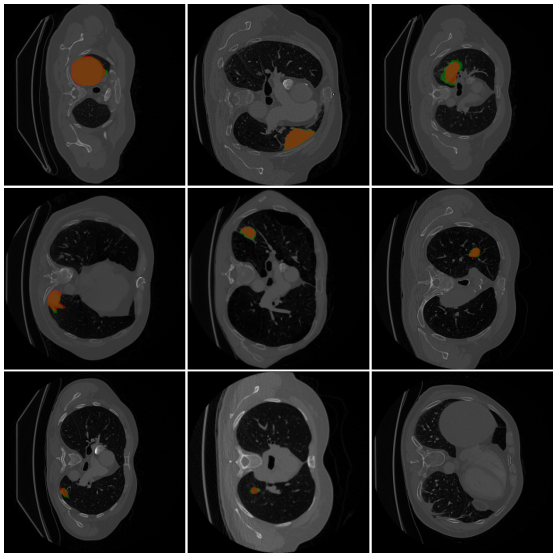

Figure 7: A set of predictions. Green is ground truth, and red is the predicition of our model. Visit the repository for more. Note: I recommend zooming in.

We measured the average IoU-score of slices, that contained a label or a prediction, the results of which is shown in table 3. The dataset was split 80/10/10.

Set Type	Training	Devel.	Testing.
Avg. IoU	0.6295	0.6368	0.615

Table 3: Intersection-over-union score for each split.

6 Conclusion

Based on the performance of our models, I conclude that a VAE-based approach is indeed a viable option for segmentation of biomedical imagery. Further development is required. I theorize, that since tumors tend to exhibit similar structural anomalies in comparison to healthy tissue, regardless of where they are in the body, tuning the models to other organs should pose little challenge.