universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Forecasting Irregularly Sampled CT Image Time Series of Lung Cancer Patients Using Deep Learning

verfasst von | submitted by

Alina Behrens

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 645
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Data Science
Betreut von | Supervisor:	Dipl.-Ing. Dr. Georg Langs

Affiliation

This thesis was conducted in the Christian Doppler Laboratory for Machine Learning Driven Precision Imaging, Department of Biomedical Imaging and Image-Guided Therapy, Medical University of Vienna.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Univ.-Prof. Dr. Georg Langs, for his invaluable guidance, encouragement, and continuous support throughout the course of this work. I am also grateful to my colleagues, Daniel Sobotka, Konstantin Miloserdov, and Stefan Brandstätter, for their helpful discussions, collaborative spirit, and technical assistance. Additionally, I thank Assoc.-Prof. Dr. Helmut Prosch and Dr. Lucian Beer, for their medical insight and support in guiding the clinical aspects of this work.
The financial support by the Austrian Federal Ministry of Labour and Economy, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

Abstract

Early detection and well-informed treatment decisions are crucial for improving lung cancer outcomes. Computed Tomography (CT) imaging of lung nodule progression provides physicians with valuable insights that contribute to timely and effective interventions. The ability to predict tumor development from a limited number of existing CT scans may significantly reduce exposure to radiation and facilitate earlier treatment decisions. To this end, we propose a deep learning model to forecast a future 3D CT image at any given time point based on a sequence of past 3D CT images of the patient. In the context of routine clinical practice, CT scans are not typically obtained at regular intervals, but rather in response to clinical necessity. Therefore, the proposed deep learning model accepts a time series with a variable number of scans as input, where the scans may have been taken at arbitrary time points. Our model is based on the existing work in the area of video processing. We propose a two-step model in which a Vector-Quantized Generative Adversarial Network is employed to compress the high-dimensional CT images and produce a structured latent space first. Then, a sequence model forecasts future unseen images. For the sequence model, we compare a transformer and a Selective State Space Model, namely Mamba. The irregular sampling of the input scans is integrated through a triplet representation, where embeddings of latent values, 3D positions, and time stamps are combined. The model is trained and evaluated on an in-house dataset of a heterogeneous routine cohort of lung cancer patients.

Kurzfassung

Frühzeitige Erkennung und fundierte Behandlungsentscheidungen sind entscheidend für die Heilungschancen von Lungenkrebs. Dabei liefern Computertomographie (CT)-Bildgebungen von der Entwicklung von Lungenrundherden Ärzten wertvolle Erkenntnisse, die zu rechtzeitigen und wirksamen Interventionen beitragen. Die Möglichkeit, Tumorentwicklungen anhand einer begrenzten Anzahl vorhandener CT-Scans vorherzusagen, könnte die Strahlenbelastung für Patienten erheblich verringern und frühere Behandlungsentscheidungen ermöglichen. Deswegen präsentieren wir in dieser Arbeit ein Deep-Learning-Modell, das basierend auf einer Sequenz vergangener 3D CT-Bilder eines Patienten ein zukünftiges 3D CT-Bild zu einem beliebigen Zeitpunkt vorhersagen können soll. Im klinischen Alltag werden solche CT-Scans in der Regel nicht in regelmäßigen Abständen, sondern nur bei klinischer Notwendigkeit erstellt. Das vorgeschlagene Deep-Learning-Modell kann daher eine Zeitreihe mit einer variablen Anzahl von Scans verarbeiten, wobei diese Scans zu beliebigen Zeitpunkten aufgenommen worden sein können. Unser Modell basiert auf bestehender Arbeit im Bereich der Videoverarbeitung. Wir entwickeln ein zweistufiges Modell, bei dem zuerst ein Vector-Quantized Generative Adversarial Network eingesetzt wird, um die hochdimensionalen CT-Bilder zu komprimieren und einen strukturierten latenten Raum zu erzeugen. In diesem Raum sagt ein Sequenzmodell dann zukünftige unbekannte Bilder vorher. Für das Sequenzmodell vergleichen wir einen Transformer mit einem selektiven Zustandsraummodell, genauer gesagt Mamba. Die unregelmäßigen Aufnahmezeitpunkte der Bilder werden durch eine Repräsentation integriert, in der Werte, 3D-Positionen und Zeitpunkte kombiniert werden. Das Modell wird auf einem internen Datensatz einer heterogenen Routine-Kohorte bestehend aus Lungenkrebspatienten trainiert und evaluiert.

Contents

Mathematical Notations

$\mathbf{v}$ A subvolume from the whole lung in the pixel space.

$(\mathbf{v}_p^{(t_i)})_{i=1}^{T_p}$ A sequence of subvolumes from a patient p with T_p CT images at different time points t_i.

l_v The size of a cubic lung subvolume in terms of the number of pixels along one axis.

$\mathbf{z}$ A lung subvolume encoded to the latent space, represented by an array of codebook indices.

$(\mathbf{z}_p^{(t_i)})_{i=1}^{T_p}$ A sequence of encoded latent patches of a patient p.

l_z The size of a cubic lung subvolume in the latent space in terms of the number of codebook indices along one axis.

z One latent pixel (codebook index) of a latent space patch $\mathbf{z}$.

t The timepoint of an observation.

c The channel, namely the spatial position, of a specific codebook index in the 3D patch.

$\mathbf{x}$ Triplet representation of a specific codebook index in a 3D patch, containing codebook index z, channel c, and timepoint t.

X A set of triplets that represents a time series in the latent space.

T The length of the input sequence for the forecasting models, in terms of the number of input CT scans.

$T_{\max}$ The length of the input sequence that the forecasting model requires. If the sequence has fewer entries, it is padded.

$\mathbf{q}$ Tuple representation of a specific query, consisting of the channel c and timepoint t.

Q The set of all query tuples for the target patch.

$\mathbf{y}$ Forecasting target made up of codebook indices.

$\mathbf{y}_q$ Forecasting target with quantized codebook vectors.

$\mathbf{z}_e$ A lung subvolume encoded to the latent space, but not yet quantized, represented by an array of vectors in a continuous space.

$\mathbf{z}_q$ A lung subvolume encoded to the latent space, represented by an array of quantized embedding vectors.

$\mathcal{E}$ The encoder that transforms a lung subvolume to its latent space representation.

$\mathcal{D}$ The decoder that transforms a latent space volume to its pixel space representation.

$\mathbf{q}(\cdot)$ The quantizer function that maps a vector in the continuous VQ-VAE latent space to the closest embedding vector.

$\mathbf{e}$ An embedding vector or codebook vector of a VQ-VAE or VQ-GAN.

n_{cb} The number of embedding vectors in the VQ-VAE latent space.

d_{cb} The dimension of the embedding vectors in the VQ-VAE latent space.

d_{model} The embedding dimension of the forecasting model.

1. Introduction

Lung cancer remains the primary cause of cancer-related mortality worldwide. It is predominantly diagnosed at an advanced stage, resulting in only approximately 25% of individuals diagnosed with lung and bronchus cancer surviving for five years or longer [1]. In recent years, mortality has declined not only due to a decline in smoking, but also because of earlier detection and advances in treatment [2]. When cancer is detected in an early stage, the five-year survival rates increase notably to approximately 60% [1]. Screenings for lung cancer have proven to be effective in enhancing early detection and reducing cancer-related deaths [3]. Nonetheless, they also carry the risk of detecting benign or slow-growing lung nodules that pose no threat to the patient [4]. The repercussions of such overdetection are quite substantial; it can result in unwarranted diagnostics or treatments, increase patient anxiety, inflate costs, and unnecessarily consume medical resources [4]. Thus, it is important to reliably evaluate malignancy during the initial consultations, preventing disproportionate exposure of the patient to additional radiation or other potentially harmful diagnostic techniques.
Alongside early detection, therapeutic interventions are crucial for lowering mortality rates. Currently, a variety of treatment options are available to patients, including immunotherapy, targeted therapy, chemotherapy, chemoradiotherapy, antiangiogenic therapy, and combinations of therapies. Each therapy category comprises a wide array of approved drugs [5]. The multiplicity of options complicates physicians' ability to make personalized treatment decisions, as predicting outcomes for individual patients and evaluating treatment response remain difficult due to tumor heterogeneity and the complexity of response assessment.

1.1. Motivation

An essential aspect of guiding treatment decisions and evaluating malignancy is monitoring the progression of nodules over time [6], [3]. Diagnostic approaches and progression evaluation thereby largely rely on imaging techniques such as Computed Tomography (CT), Position Emission Tomography (PET), or Magnetic Resonance Tomography (MRT) [5]. To enable early diagnosis and treatment decisions through progression analysis, while simultaneously reducing the quantity of necessary imaging data over extended time periods to an absolute minimum, it would be beneficial to be able to forecast future cancer development [7]. To this day, utilizing a patient's medical history for forecasting disease progression remains difficult. Predicting when the present treatment will become ineffective, requiring additional diagnostics and treatment decisions, poses significant challenges for physicians. Recent advances in deep learning in fields such as image analysis, time series forecasting, or video prediction have the potential to facilitate the utilization of data from large lung cancer populations for the purpose of making personalized predictions regarding disease progression.
In the following work, we address the task of forecasting the course of lung cancer based

on multiple past CT scans using current deep learning methods, to investigate whether CT images generated through deep learning techniques could be beneficial for physicians. Specifically, the objective is to predict a 3D CT image at a designated future time point, based on a variable number of past images. In comparison to predicting simpler measures like the nodule growth rate, the prediction of a future CT scan allows doctors to assess various characteristics such as morphology and texture, instead of directly reducing the prediction to a single dimension. An important constraint of this thesis is the exclusive use of past CT images, excluding other significant data sources such as patient demographics, medical records, or treatment information. This choice simplifies the task to a single technical challenge – forecasting CT image sequences – prior to tackling additional variables. The incorporation of such multimodal data is left for future research.

1.2. Challenges

From a technical point of view, the forecasting of CT image time series poses the following main challenges:

- **High dimensionality:** For the analysis of image sequences, especially with high-resolution 3D images, deep learning methods require expensive computational resources. Many deep learning approaches do not scale linearly, leading to even higher time and memory consumption.

- **Sparse input data**: Few time points are provided for each forecasting task due to the high cost of CT scans and their radiation exposure to patients. It is challenging to extrapolate a high-dimensional output from only few samples in time.

- **Irregularly sampled time points**: Like in most medical routine data, CT images are not conducted regularly, but based on necessity (e.g., before and after surgery), and patients' and doctors' availability. Time intervals between two consecutive CT scans can therefore range between days and years. These time gaps have to be incorporated and learned by deep learning models.

Additional challenges arise from the data. The employed dataset for this task is extracted from a large heterogeneous routine population of the Vienna general hospital (Allgemeines Krankenhaus, AKH) that includes patients with available imaging data diagnosed with lung cancer from 2002 to 2022. Given the considerable heterogeneity of the data, caused by, e.g., multiple CT scanners, different lung cancer progressions, and various treatments, it is necessary to perform extensive pre-processing. Existing literature on the task of lung nodule progression forecasting on an image level requires manual tumor localization [8], [9], [7], high quality tumor segmentation masks [9], [10], or other measures like tumor volume and diameter that have to be manually annotated [11]. All of these are rarely available and expensive to attain, and also not part of our routine population dataset. Therefore, an additional challenge is not to rely on precise segmentation masks, but to use rough automatic segmentations for tumor localization.

1.3. Contributions

We propose a deep learning model that is based on both video and time series literature to tackle the aforementioned challenges. The model is composed of two parts: a Vector-Quantized Generative Adversarial Network (VQ-GAN), a model based on a Variational Autoencoder (VAE) that produces a quantized latent space, to reduce the dimensionality of individual CT scans, and a forecasting model that operates solely on this latent space. Two distinct architectures for the forecasting model are considered: an encoder-decoder transformer model and a Selective State Space Model (SSM) with a cross-attention block. The time gaps are incorporated into the forecasting model through the use of learnable triplet embeddings. Extensive experiments on the mentioned in-house dataset show that the proposed models struggle to produce high-quality predictions when trained on the available highly heterogeneous data. We therefore investigate various alterations to the models that could potentially improve the results. We introduce loss functions designed to overcome the model's inability to predict temporal continuous image sequences, and we come up with a simplified interpolation task which the model is meant to then transfer to the extrapolation task. However, none of the alterations can significantly improve the prediction accuracy.

1.4. Outline

The thesis begins with a chapter that summarizes the most important deep learning models for this thesis (Chapter 2), providing an in-depth description of the utilized neural network architectures, such as VAEs and transformers.
Chapter 3 explores recent literature related to the task, focusing on papers that specifically discuss nodule growth prediction, as well as studies on time series and video forecasting, which are also related to the task.
In the subsequent Chapter 4, the internal data set and its processing steps are elaborated.
Chapter 5 outlines the proposed latent space forecasting model and its variants, including the model architectures, training procedures, and various loss functions.
In chapter 6, the prediction performance is assessed and model variants are compared through a number of experiments. The Vector-Quantized Variational Autoencoder (VQ-VAE) and forecasting model are analyzed individually, before discussing the results of the combined model. This chapter also includes the results of the experiments that were conducted.
Finally, the experiments are discussed in Chapter 7, wherein the results are thoroughly analyzed and the underlying causes of the model's suboptimal performance are investigated.
The thesis concludes with a brief summary and provides an outlook on future research directions.

2. Deep Learning Models

This chapter reviews deep learning models that form the basis of the proposed models. It starts by giving a general overview of neural networks and a short explanation of convolutional neural networks. Thereafter, we give an introduction to generative neural networks, such as Generative Adversarial Networks (GANs) [12], Variational Autoencoders (VAEs) [13], and variations of it, including Vector-Quantized Variational Autoencoders (VQ-VAEs) [14] and Vector-Quantized Generative Adversarial Networks (VQ-GANs) [15]. These networks will play a role in generating realistic future images. Lastly, we describe sequential models, focusing on transformer models [16] and Selective State Space Models (SSMs) [17], which will be used to process the sequence of past latent images to infer a future image in this work.

2.1. Neural Network

Neural networks are powerful models that can learn rules from data [18, p. 23]. The main building block of a neural network is the *artificial neuron* [19], a specific example being the *perceptron* [20], [18, pp. 157–158]. The neuron can be described through the formula

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{2.1}$$

where the x_i are the inputs to the neuron, w_i the neuron's weights, b the bias, and φ is an activation function [18, pp. 158, 179]. Typically, neurons are organized into layers and linked together to create the neural network [18, p. 43]. One example is the Multilayer Perceptron (MLP) [21], in which neurons are arranged into multiple layers and all neurons of two consecutive layers are connected [18, p. 178].

Activation functions. In general, the activation function φ in a neuron can be any (continuous) function [22, p. 428]. In practice, certain properties, like non-saturating functions, have been shown to avoid problems like vanishing gradients [23], [22, p. 429]. The most common activation function is the Rectified Linear Unit (ReLU) [24], which is defined as

$$\mathrm{ReLU}(x) = \max(x, 0). \tag{2.2}$$

There are also many alterations to the ReLU activation, for example, to overcome its differentiability issue at 0, such as the Sigmoid Linear Unit (SiLU) activation [25] which is defined as

$$\mathrm{SiLU}(x) = x \cdot \mathrm{sigmoid}(x). \tag{2.3}$$

Supervised and unsupervised learning. Depending on the type of data the neural network receives, the learning process is either called *supervised* or *unsupervised* [26, pp. 102–103]. When the data contains samples that are associated with a target, the model performs supervised learning, where it typically learns the distribution $p(y|x)$ that maps the input sample x to the target y. An example of a supervised learning task is classification, where the model learns a rule to assign categories to input samples [26, p.98]. In the other case, in which the model receives samples without associated targets, the model usually tries to learn the probability distribution $p(x)$, which is called unsupervised learning [26, p. 103]. For example, an estimation of the distribution $p(x)$ can be used for generating new data points by sampling from it [26, p. 651].

Forward pass. After defining the network architecture, i.e., the arrangement of neurons and choice of activation functions, as well as the input data the network receives, the data samples x can be passed through the network. This process is referred to as forward pass. It begins by calculating the outputs of all neurons in the input layer. These outputs are then forwarded to the subsequent layer, continuing this pattern until the output of the final layer is determined [21]. The final output layer depends on the task. For example, the output of multi-class classification tasks is usually computed by as many neurons as classes in the output layer, where each neuron's output is transformed into a probability via the *softmax* function [27], which is defined as $\sigma(\mathbf{y})_i = \frac{e^{y_i}}{\sum_{j=1}^{C} e^{y_j}}$, where $\mathbf{y}$ is the vector of all outputs, also referred to as logits, and C is the number of classes. The *softmax* function thereby typically replaces the activation function in the output layer [28, p. 39].

Maximum likelihood estimation and loss functions. At the beginning of the training, all weights of the neural network are initialized randomly, and the output of the forward pass is not meaningful [26, p. 298]. To train the network by adapting its weights, a measure of output quality is required [26, p. 101]. For supervised tasks, the aim is to approximate the probability distribution $p(y|x)$ as accurately as possible using a network with the parameter set θ, which contains all network weights. For unsupervised tasks, we instead try to estimate the distribution $p(x)$ [26, p. 103]. To do so, the likelihood is often used, which is defined as the probability of observing a given dataset given a model with a particular set of parameters. Mathematically, it is formulated as $L(\theta|\mathbf{X})$, where $\mathbf{X}$ is the observed dataset and θ is the parameter set [28, p. 12]. The goal is to find parameters that maximize the likelihood, which is called *maximum likelihood estimation* [28, p. 13]. The maximum likelihood estimator θ^* is defined as

$$\begin{aligned} \theta^* &= \arg\max_{\theta} L(\theta|\mathbf{X}) \\ &= \arg\max_{\theta} \prod_{i=1}^{n} p_{\text{model}}(x^{(i)};\theta) \\ &= \arg\max_{\theta} \sum_{i=1}^{n} \log p_{\text{model}}(x^{(i)};\theta). \end{aligned} \tag{2.4}$$

For supervised tasks, one would usually write $p_{\text{model}}(y;\theta,x)$ instead of $p_{\text{model}}(x;\theta)$. Note that the maximum likelihood estimator could also be interpreted as the minimum of a Kullback-Leibler Divergence, which means as minimum of the difference between the empirical distribution of the dataset and the distribution modeled by the parameter set

(p_{model}) [26, p. 130]. The Kullback-Leibler Divergence will be defined and explained in Section 2.6. This minimization is also equivalent to the minimization of Cross Entropy (CE) between the two specified distributions [26, p. 130].
When making specific assumptions for the model distribution, we arrive at well-known loss functions, some of which are listed below:

- **Mean Squared Error (MSE) Loss**: A Gaussian model leads to the MSE loss, where $\mathcal{L}_{\text{MSE}}(\hat{y}, y) = ||\hat{y} - y||_2^2$ is minimized over all samples, where y is the target and $\hat{y}$ is the output estimated by the model. Note that the conventional MSE is defined as $\text{MSE}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{N}||\mathbf{y} - \hat{\mathbf{y}}||_2^2$, where an additional scaling factor $\frac{1}{N}$ is used to take the mean over all N samples collected in the vector $\mathbf{y}$. This scaling is not necessary in a loss function, since it does not affect the gradient's direction. [29], [22, p. 115]

- **L1 Loss**: A similar loss, where the Gaussian model is replaced by a Laplacian model, is the L1 loss, defined as $\mathcal{L}_{\text{L1}}(\hat{y}, y) = ||\hat{y} - y||_1$. This loss is related to the Mean Absolute Error (MAE), which is defined as $\text{MAE}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| = \frac{1}{N}||\mathbf{y} - \hat{\mathbf{y}}||_1$. [29]

- **CE Loss**: The CE loss is used for Bernoulli or softmax distributions and is defined as $\mathcal{L}_{\text{CE}}(\hat{\mathbf{p}}, y) = -\sum_{i=1}^{C} y_i \log(\hat{p}_i)$, where $\hat{p}_i$ is the estimated probability for the i_{th} class and y_i is 0 or 1, indicating whether the sample belongs to class i. [29], [22, pp. 342, 351–352]

Backpropagation and Optimization. Having defined an objective, namely maximizing the likelihood or, equivalently, minimizing the loss function, the next question is how to use this measure to optimize the weights of the network. The common optimization method for neural networks is *stochastic gradient descent* ([30], [31], [20]), where a random sample or a small batch of random samples is drawn from the dataset to calculate the loss and estimate the gradient from it. Then, a small step toward the steepest descent, so in the direction of the negative gradient, is taken to improve the current parameters. This is repeated until convergence, or based on some other criteria, such as the validation loss that will be described later in this section. The algorithm can be described as

$$\theta_{t+1} = \theta_t - \eta_t g_t, \tag{2.5}$$

where θ_t are the parameters, i.e., the weights of the network, at update step t, and g_t is the current gradient estimation with regard to θ. The scalar η is the so-called *learning rate* that determines the step size in the negative gradient direction [22, pp. 284, 292]. It is a hyperparameter of the network that can either be a fixed constant η or change depending on the step t (η_t).
In practice, the stochastic gradient estimation is further improved by including momentum and adaptive learning rates, the most popular implementation being ADAM [32]. It combines the current gradient estimate with past estimates, and adapts the learning rate based on past squared gradients, in order to reach better convergence and avoid getting stuck in local minima [22, pp. 300].
As described, a gradient estimate for the loss is required for optimization. To get this estimate, *Backpropagation* is used. After forward propagation of an input, the loss is

calculated, and its gradient with regard to all weights needs to be determined. This is done using the chain rule, where the gradient with respect to the parameters of a specific layer depends on the values calculated in the subsequent layer, which is why the gradient is calculated "backward" starting in the output layer. [22, pp. 438]

Data split and Overfitting. We have now defined how the network is optimized to approximate a target distribution given a dataset. It is important to note, however, that the goal is not only for the model to perform well on data samples it was trained on, but it should generalize to new samples [26, p. 102]. This is the reason why the available data is typically divided into three different sets: the training set, the validation set, and the test set. The training set is used to train the model, which means to adapt its weights using the methods described above. The validation set functions as an independent set to choose hyperparameters like the learning rate, or to stop the training process before the model overfits [26, pp. 118–119]. Overfitting is the process of improving the model on the training data, while at the same time, the error on unseen data grows [26, pp. 108–110]. The error on unseen data is called generalization error and is typically estimated using the test set that was not involved in any training, or model and hyperparameter choices [26, pp. 108]. Neural networks are prone to overfitting due to their high number of parameters. There are many ways to avoid overfitting, two of which are *Early stopping* [33] and *Dropout* [34].

- **Early stopping**: During the training process, the training and validation errors are monitored. As soon as the validation error starts to grow, the training process is stopped. [22, pp. 126–127]
- **Dropout**: Another more advanced technique to regularize the model is dropout. At training time, a random subset of neurons in the network is "turned off", which means the neuron's activations are set to zero. The remaining neurons in a layer are scaled to ensure that the expected sum of activations remains the same when not using dropout at inference time. [28, pp. 54–55]

2.2. Convolutional Neural Network

Convolutional Neural Networks (CNNs) [35] are a specific kind of neural networks that employ convolutions [26, p. 326]. CNNs have shown to be very powerful in, for example, image processing tasks. In comparison to fully connected layers, they have several advantages. Among other benefits, they are more parameter efficient, as they utilize shared weights instead of linking each pixel to each input neuron individually, and they ensure translation equivariance [26, pp. 329–335].

Convolution. The definition of the convolution operation in 2D is

$$[\mathbf{F} \circledast \mathbf{X}](i,j) = \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} f_{u,v} x_{i+u,j+v}, \tag{2.6}$$

where $\mathbf{F}$ is the *filter* or *kernel* with dimensions $H \times W$ and $\mathbf{X}$ is the input image. It can be interpreted as sliding a filter over an image and computing a weighted sum of filter values and pixel values. This operation can be easily generalized to a third dimension and/or to

multiple image channels by increasing the dimension of the filter. [22, pp. 470-471, 473]
Since convolving an image with a smaller kernel usually results in a smaller output image, a *padding* around the image is often added according to the kernel size. This can be achieved through different methods, such as padding with a fixed value, mirroring the image, or employing any other appropriate strategy. [22, p. 471–472]
Another adaptation that can be made is the step size with which the filter is moved over the image. If this step size is larger than one, the convolution operation is called *strided convolution*. This operation is often used to *downsample* input images, that is, reduce their size. [22, p. 473]
In order to be able to upsample an image again, we need a special kind of convolution — the *transposed convolution*. Instead of multiplying the kernel with a piece of the input image of the same size, every single input pixel is multiplied by the whole kernel. This is done by keeping the kernel dimension and multiplying each kernel value by the pixel value of the input image. The intermediate image piece that results from multiplying the kernel with the upper left image pixel is placed in the upper left corner of the output image. Then, with every step in the input image, the resulting kernel is added to the current output after moving it in the same direction as the step on the input image. In the transposed convolution case, the stride defines the step size on the output image, which means how far away the intermediate results are placed on the output image. [22, pp. 486–487]
Transposed and strided transposed convolutions are visualized in Figure 2.1 and 2.2 respectively.

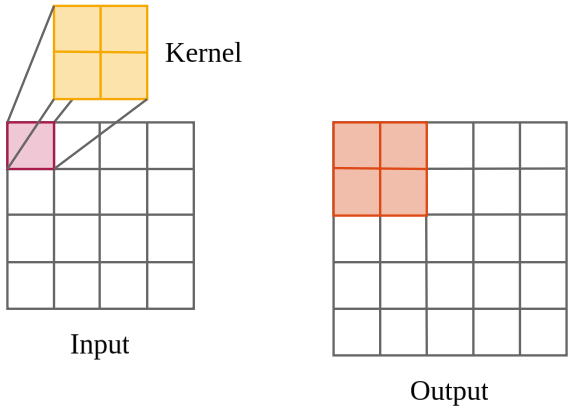


Figure 2.1.: Transposed convolution.

Convolutions in a neural network The described convolution filters are used as the weights of the CNN, and the weights are, as claimed above, *shared*, because the same parameter values are used for different positions on the image, in comparison to a fully connected layer where each input pixel gets assigned an individual weight. In CNNs, there are usually multiple convolutional filters in one layer that learn different features. Like fully connected layers, convolutional layers are often stacked together to form a deeper neural network. [22, p. 467]

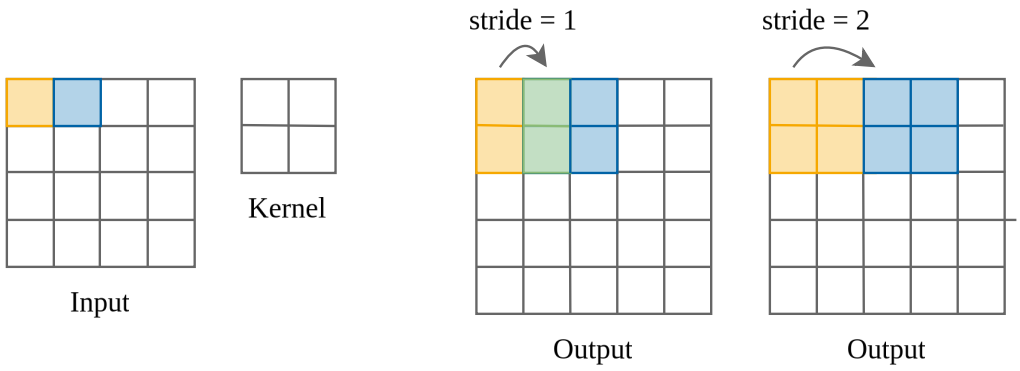


Figure 2.2.: Strided transposed convolution.

Residual Block. The so-called *Residual block* was proposed by He et al. as part of the ResNet [36]. In addition to the described convolutional layers, it introduces a skip connection, where the original data is added to the output. More precisely, the input of the residual layer is passed through one convolutional layer, followed by an activation, followed by another convolutional layer, after which the input is added to the current output, and then a last activation is applied. This residual unit enables an improved gradient flow as the gradient can skip over convolutional operations. [22, pp. 452, 483–484]

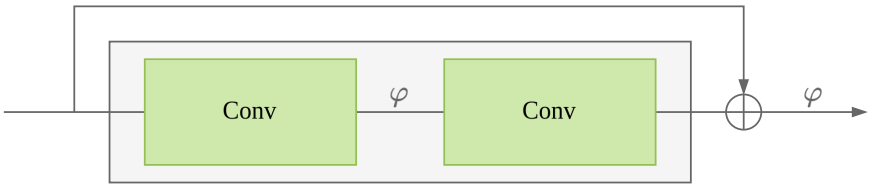


Figure 2.3.: Residual block.

2.3. Generative Network

Generative neural networks are a subclass of neural networks with a specific task: they generate new realistic data samples [28, p. 1]. Examples are the creation of text (such as [16], [37]) or the generation of realistic images (such as [12], [38]), e.g., faces of persons that do not exist. To do so, generative networks approximate the underlying distribution $p_{\text{data}}(x)$ from which the given training data was drawn, which is, in most cases, complex and unknown. The estimate of the true distribution $p_{\text{data}}(x)$ learned by the model is usually referred to as $p_{\text{model}}(x)$ [28, p. 9].
It is also possible to generate samples based on conditions, for example, to create an image conditioned on a text prompt that describes the image. The data distribution the model then approximates is called $p_{\text{data}}(x|c)$, where c is the condition [39, p. 774].
Depending on the employed model, the data distribution can be estimated in two different ways: explicitly or implicitly [39, p. 773]. In explicit density estimation, the model provides a well-defined mathematical representation of the data distribution. In contrast,

implicit models learn to generate data without explicitly assigning likelihoods to individual samples [39, p. 773]. A prominent example of explicit density estimation is the Variational Autoencoder (VAE) [13]. On the other hand, models such as Generative Adversarial Networks (GANs) [12], diffusion models, and transformer-based generative models do not explicitly compute the data distribution, but instead focus on learning the underlying data generation process.
Another key distinction between generative models is whether they generate data in an *autoregressive* manner [39, p. 821]. In autoregressive models, data is generated sequentially, with each step conditioned on the previous outputs [39, p. 821]. This approach is common in text generation (see, e.g., [16], [37]), where each predicted word depends on the preceding ones, but is also frequently used in image generation, where pixels or patches are generated step by step based on prior predictions (like in [38]). In contrast, non-autoregressive models generate data in parallel, producing entire samples at once without relying on sequential dependencies (see, e.g., [12]).

2.4. Generative Adversarial Network

Generative Adversarial Networks (GANs) are one prominent type of generative networks that has been proposed by Goodfellow et al. [12]. They will be an important part of the model we propose.

Components of the GAN. The Generative Adversarial Network (GAN) consists of two players: the generator and the discriminator.

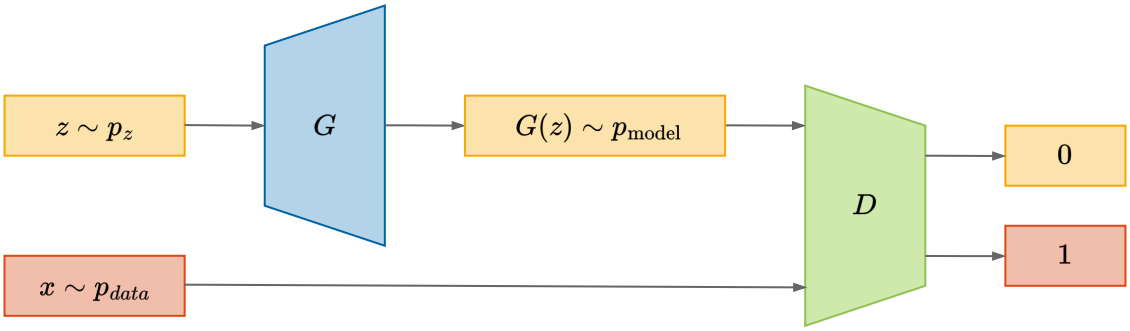


Figure 2.4.: The general framework of a GAN.

The generator G receives a random input $z \sim p_z(z)$, where $p_z(z)$ is a simple probability distribution, e.g., a standard Gaussian, from which the generator tries to construct a sample that appears to be drawn from $p_{data}(x)$. That is, the distribution of the generator output $p_{\text{model}}(G(z))$ is meant to approximate the data distribution $p_{data}(x)$ [12]. The discriminator D is fed with samples that are either real or produced by the generator and aims to accurately predict which of them it is. This can be mathematically formulated as the following min-max optimization problem, which uses a binary CE loss.

$$\begin{aligned} \theta^* &= \min_G \max_D \mathcal{L}_{GAN}(G, D), \\ \mathcal{L}_{GAN}(G, D) &= \mathbb{E}_{x \sim p_{\text{data}}}[\log(D(x))] + \mathbb{E}_{z \sim p_z}[1 - \log(D(G(z)))] \end{aligned} \tag{2.7}$$

The discriminator aims to maximize the first term of Equation 2.7, which corresponds to correctly assigning ones to real data points. It also tries to minimize the case where generated inputs are classified as ones, which can be seen in the second term, where one minus this case is maximized [12]. The generator, on the other hand, approximates the true data distribution, i.e., its target is to deceive the generator, which is why it minimizes the second term. It can be shown that the global optimum of the min-max problem 2.7 is attained if and only if $p_{\text{model}} = p_{\text{data}}$ [12]. This optimum, where the generator produces perfect fakes and the discriminator can no longer distinguish between true and generated images, is also known as *Nash Equilibrium* [40].

Training of the GAN. In the training scheme proposed by the authors [12], the discriminator and generator are optimized in turns. In each training step, a given number of samples is taken from the training data, and the same number of samples is constructed by feeding the generator with random inputs. First, all real and generated samples are used to update the discriminator weights using stochastic gradient ascent. Thereafter, the weights of the discriminator are fixed, and the generator is updated with a given amount of random inputs using gradient descent, whereby the generator is optimized to fool the discriminator. Note that the gradient update of the generator depends on the current state of the discriminator, which can be interpreted as the discriminator serving as a loss function that becomes increasingly complex in the course of the training. [12]

Inference on the trained GAN. In order to generate a new sample from the implicitly estimated data distribution, a random input is drawn from the distribution $p_z(z)$ and handed to the generator. Thus, during inference, the discriminator is usually no longer needed. [12]

Least Squares GAN. A variant of the explained GAN that uses a CE loss is the Least Squares GAN [41] that instead formulates the loss as a MSE as shown in the equation below.

$$\begin{aligned} \mathcal{L}_{LSGAN}(D) &= \frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}}\left[(D(x)-1)^2\right] + \frac{1}{2}\mathbb{E}_{z\sim p_z}\left[D(G(z))^2\right] \\ \mathcal{L}_{LSGAN}(G) &= \frac{1}{2}\mathbb{E}_{z\sim p_z}\left[(D(G(z))-1)^2\right] \end{aligned} \tag{2.8}$$

Both of these loss terms need to be minimized. Such Least Squares GANs have been shown to be able to generate images with higher quality and be more stable during training. [41]

PatchGAN discriminator. Another variant of the GAN, specifically an adaptation of the discriminator, is used for generating images. A typical issue is that GANs produce blurry images, which is caused by the use of the MSE or L1 loss. To improve quality also in the high-frequency components of the image, Isola et al. [42] propose a PatchGAN discriminator. Instead of outputting a single score for the whole image, the discriminator gives scores to smaller (overlapping) sub-patches of the image with a given size, which are averaged at the end. As a result, the PatchGAN discriminator focuses on fine-grained textures and local structures. [42]

2.5. Autoencoder

A popular deep learning approach to dimensionality reduction is the autoencoder. The concept of autoencoders was already introduced decades ago; some of the earliest publications are the PhD thesis of LeCun with the title "Connexionist learning models" [43], and the chapter "Learning Internal Representations by Error Propagation" by Rumelhart et al. [21]. While autoencoders are not necessarily generative models, they form an important basis for Variational Autoencoders (VAEs) [13] that will be reviewed in the next section. The idea of autoencoders is to reduce dimensionality by reconstructing the input employing an unsupervised neural network that contains a low-dimensional or otherwise constrained hidden layer serving as a bottleneck [26, pp. 499-500]. Typically, the neural network is thought of as two parts: the encoder, which encodes the input to an internal representation h, and the decoder, which decodes this representation back to the original input [26, p. 499]. The input and output layers have the same shape, and the loss function is a so-called reconstruction loss that measures the similarity between input and output [26, p. 500]. The network can have various architectures, it could be, e.g., a simple feed-forward network or a CNN. The network is constructed and trained in a way to grant useful properties to the internal representation h additional to its reconstruction abilities [26, p. 501]. Besides the low dimensionality of the internal representation, other constraints can confer beneficial properties, e.g., sparsity or robustness. To enforce such constraints, the reconstruction loss function is complemented by additional penalties [26, pp. 501–504].

2.6. Variational Autoencoder

A special type of autoencoder that is also a generative network is the Variational Autoencoder (VAE) introduced by Kingma et al. [13]. A variant of the VAE will play an important role in our proposed model. Next to its ability to compress high-dimensional input data into a lower-dimensional representation, its structured latent space can also be used to generate new samples.
Instead of learning a fixed latent vector for each data point like a vanilla autoencoder, the VAE learns parameters of a distribution from which the latent vector is then sampled [13]. This leads to a regularized continuous latent space from which random variables can be drawn that produce meaningful samples when decoding them. Compared to the simple autoencoder, this allows the VAE to be used as a generative model, as arbitrary samples can be drawn from the parameterized distribution that should produce meaningful data [39, p. 794]. In contrast to the GAN, the VAE models the approximated data distribution p_{data} explicitly [39, p. 773].

Model assumptions. The underlying assumption of the VAE is that a data point x is generated by a random process that depends on a hidden random variable z, i.e., a data point x is drawn from a distribution $p_\theta(x|z)$ after z is drawn from some prior distribution $p_\theta(z)$. The complete generative model is thus defined as $p_\theta(z, x) = p_\theta(x|z)p_\theta(z)$ [13]. Furthermore, it is assumed that both distributions come from a parametric family of distributions with unknown parameters θ [13]. The model is meant to learn this generation process, specifically the translation of latent variables z to data samples x, that is, the distribution $p_\theta(x|z)$. At the same time, the model aims to learn how to infer the latent

variables z given an input x, i.e., to learn $p_\theta(z|x)$, which is referred to as posterior inference [39, p. 792]. The inference of the posterior distribution $p(z|x)$ is usually intractable, as $p(z|x) = \frac{p(x,z)}{p(x)} = \frac{p(x,z)}{\int p(x,z)dz}$, where the integral over all latent variables is taken, cannot be evaluated. To overcome intractability, a model $q_\varphi(z|x)$ with learnable parameters φ is introduced to approximate the posterior $p_\theta(z|x)$ [13].

Components of the VAE. The architecture of the VAE consists of an encoder network and a decoder network. The encoder network projects random variables x to corresponding parameters φ that characterize the distribution $q_\varphi(z|x)$. It thus learns to map the input x to a reduced latent variable z by sampling from the approximate posterior $q_\varphi(z|x)$. The decoder network, on the other hand, reconstructs an output x' from the latent variable z, that is, it learns the distribution $p_\theta(x|z)$. [39, p. 792]
In the original paper [13], the authors propose to use a multivariate standard normal distribution as the prior, i.e., $p_\theta(z) = \mathcal{N}(0, I)$. They assume the posterior distribution to be approximately Gaussian with a diagonal covariance matrix, i.e., $q_\varphi(z|x) = \mathcal{N}(z; \mu_\varphi(x), \text{diag}(\sigma^2_\varphi(x)))$. Note that in the provided formula, the mean vector $\mu_\varphi(x) \in \mathbb{R}^d$ is a vector, which characterizes the mean of a multivariate Gaussian with d dimensions, where d is a hyperparameter of the model and $\text{diag}(\sigma^2_\varphi(x)) \in \mathbb{R}^d$ is the diagonal of the covariance matrix.
The architecture is visualized in Figure 2.5.

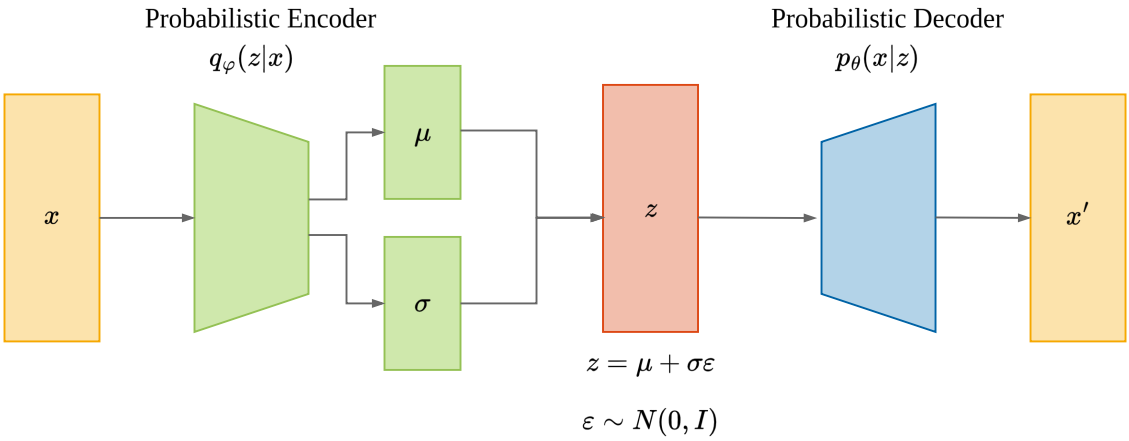


Figure 2.5.: Overview of the VAE.

Loss function of the VAE. Starting with the encoder, the aim is to learn a distribution $q_\varphi(z|x)$ that follows the true posterior distribution $p_\theta(z|x)$ as closely as possible. This can be mathematically expressed as minimizing the Kullback-Leibler Divergence (D_{KL}) [44] between $q_\varphi(z|x)$ and $p_\theta(z|x)$ [13].

$$\min_\varphi D_{KL}(q_\varphi(z|x)||p_\theta(z|x)) \tag{2.9}$$

The Kullback-Leibler divergence [44] is defined as $D_{KL}(P||Q) = \int p(x) \log \frac{p(x)}{q(x)} dx$ and measures the distance between two probability distributions. The minimization problem in Equation 2.9 can be reformulated and simplified, resulting in the following loss function [13].

$$\mathcal{L}_{\text{VAE}}(\varphi, \theta, x) = -D_{KL}(q_\varphi(z|x)||p_\theta(z)) + \mathbb{E}_q(\log p_\theta(x|z)). \tag{2.10}$$

The encoder function $q_\varphi(z|x)$ can be learned by maximizing $\mathcal{L}$ with regard to φ [13].
In the decoder, an output x should be reconstructed from a latent variable z. Maximizing the probability $p_\theta(x|z)$ refers to reconstructing x from z as accurately as possible. However, we want the model to perform well for any possible z, which is why we need to account for all latent variables z by marginalizing over them. Mathematically, this can be formulated as the integration $p_\theta(x) = \int p_\theta(x|z)p_\theta(z)dz$. Thus, our ultimate goal is to maximize $p_\theta(x)$ [13].
It can be shown that the loss defined in Equation 2.10 is a lower bound of the log probability $\log p(x)$, which is why it is often called Evidence Lower Bound (ELBO). Therefore, maximizing this loss with regard to θ also maximizes the data distribution $p_\theta(x)$ as much as possible (up to the lower bound) [13].
Consequently, Equation 2.10 is the loss that is used for VAEs, which is maximized with regard to both φ and θ. The first part, i.e., the Kullback-Leibler divergence, can be interpreted as a constraint on the latent space, which enforces the variables to closely follow a given prior distribution. The second term is the log likelihood, which can be regarded as (the negative of) the reconstruction loss. [13]
With the Gaussian assumptions on prior and posterior distributions mentioned above, the Kullback-Leibler divergence has a closed-form solution. The final estimation of the loss can be formulated as

$$\mathcal{L}_{\text{VAE}}(\varphi, \theta, x) \simeq \frac{1}{2} \sum_{j=1}^{d} \left[1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2\right] + \log p_\theta(x|z), \tag{2.11}$$

where $z \in \mathbb{R}^d \sim N(0, I)$ and μ and σ are the estimated parameters of the posterior for the given input x [13]. The log likelihood, i.e., the reconstruction loss, can be, for example, replaced by the negative of a MSE when making the additional assumption that $p_\theta(x|z)$ is Gaussian.

Optimization despite high variance of the gradient estimator. The minimization problem

$$\min_{\varphi,\theta} \mathcal{L}_{\text{VAE}}(\varphi, \theta, x) \tag{2.12}$$

is commonly solved by backpropagation when employing neural networks, which requires the gradient of $\mathcal{L}_{\text{VAE}}$. Usually, stochastic gradient descent is implemented, estimating the gradient by a small batch of samples. In the original VAE paper by Kingma et al. [13], they note that the naive estimator has very high variance, such that it is unsuited for solving the optimization problem at hand. To solve this issue, Kingma et al. introduce the so-called reparametrization trick. They replace the latent variable z by a function $g_\theta(\varepsilon, x)$, where ε follows some distribution $p(\varepsilon)$. Specifically, they parameterize z as $z = \mu(x) + \sigma(x) \cdot \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, I)$. This allows z to be expressed as a deterministic function of $\mu(x)$, $\sigma(x)$, and a random noise variable ε that is independent of the data x.

This trick enables backpropagation through the stochastic sampling process, resulting in gradient estimates that typically have lower variance. [13]

Training of the VAE. During training, data points x from the distribution $p(x)$ are handed to the encoder. The encoder outputs the mean and standard deviation of a multivariate Gaussian posterior. From this distribution, a latent variable z is sampled by using the reparameterization trick. The latent variable is then passed to the decoder that reconstructs the input data x from the latent variable z. On the output, the loss is calculated and backpropagated through the network to obtain a gradient that can be used to iteratively update the model's weights. [13]

Inference on the trained VAE. During inference, a new sample can be generated by sampling a random latent variable z and passing it to the decoder. This can be mathematically described by the following equation, where $\mathcal{D}$ is the decoder network of the trained VAE.

$$x_{\text{generated}} = \mathcal{D}(z), z \sim \mathcal{N}(0, I) \tag{2.13}$$

Due to the structured nature of the VAE latent space, it is also possible to encode a specific data point x and use its latent variable z to sample closely around it to receive similar data points, or to interpolate between latent variables of two data points. [13]

2.6.1. Vector-Quantized Variational Autoencoder

While the presented VAE creates a continuous latent space, it can be beneficial for some tasks to have discrete representations. For example, transformer models have originally been developed for language which is inherently discrete [16]. Furthermore, having a categorical uniform prior $p(z)$ avoids a phenomenon known as *posterior collapse* (see, e.g., [45]), where the latent variables z become meaningless [39, pp. 806–807, 815].
Therefore, Oord et al. [14] present the Vector-Quantized Variational Autoencoder (VQ-VAE), a VAE with discrete latent representations, which will be a main part of the proposed model. The main components of the VQ-VAE are the same as in the VAE, but the quantization has implications on the loss function and the optimization of the model, which will be detailed below.

Model assumptions. To obtain discrete latent representations, the authors [14] exchange the Gaussian prior and posterior with categorical distributions. Specifically, they define a categorical embedding space with n_{cb} categories, which are represented by d_{cb}-dimensional vectors, referred to as *embedding vectors* or *codebook vectors*. The posterior distribution is chosen to be the categorical distribution in Equation 2.14, where $\mathbf{z}_e(\mathbf{x}) \in \mathbb{R}^{d_{cb}}$ represents the still continuous encoder output and $\mathbf{e}_j$ are the n_{cb} embedding vectors.

$$q(z = k|\mathbf{x}) = \begin{cases} 1 \text{ if } k = \text{argmin}_{j=0..n_{cb}-1} ||\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_j||_2 \\ 0 \text{ otherwise} \end{cases} \tag{2.14}$$

That is, the distribution is equal to one if the given latent variable k matches the index of the embedding vector closest to the encoder output for the given $\mathbf{x}$, and zero for all other values. Thus, the discrete latent representations z are integers between 0 and $n_{cb} - 1$,

i.e., the indices of the corresponding embedding vector, which we refer to as *codebook indices*. The input $\mathbf{x}$ is usually not compressed into a single latent variable z but into an arrangement of multiple latent variables, resulting in a latent representation $\mathbf{z}$, which would imply that the encoder output and decoder input are collections of vectors, i.e., matrices. Figure 2.6 gives an overview of the whole VQ-VAE.

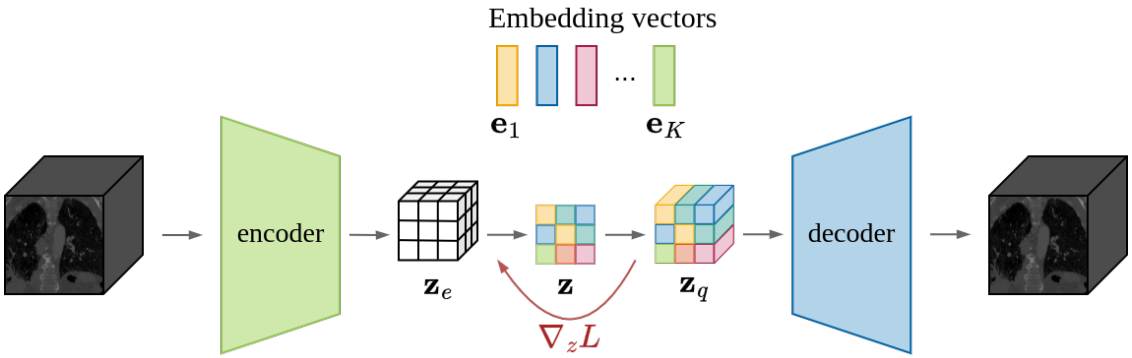


Figure 2.6.: **Overview of the VQ-VAE**. An input $\mathbf{x}$ is mapped to a continuous representation $\mathbf{z}_e$. All components of $\mathbf{z}_e$ are quantized by assigning them to the closest embedding vector $\mathbf{e}$. The latent representation $\mathbf{z}$ contains identification indices of the assigned embedding vectors, referred to as codebook indices. For the decoder input $\mathbf{z}_q$, the indices are mapped back to the corresponding vectors. From it, the decoder reconstructs the original input.

Quantization and differentiability. The quantized inputs to the decoder network, referred to as $\mathbf{z}_q(\mathbf{x})$, are the embedding vectors that correspond to the assigned indices, so the vectors closest to the encoder output, that is

$$\mathbf{z}_q(\mathbf{x}) = \mathbf{e}_k \text{ where } k = \text{argmin}_j ||\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_j||_2. \tag{2.15}$$

Equation 2.15 poses a problem for backpropagation, since the argmin operation is not differentiable. In order to circumvent this issue, the decoder gradient is handed to the encoder without adjusting it based on the argmin operation in between [14]. This is called a straight-through estimator [46].

VAE loss with a uniform categorical prior. For model training, the prior $p(z)$ is chosen to be uniform, i.e., $p(z = k) = \frac{1}{n_{cb}}$ [14]. Combined with the posterior distribution given above in Equation 2.14, the Kullback-Leibler divergence employed in the VAE loss 2.10 is constant, as shown below.

$$\begin{aligned} D_{KL}(q(z = k|\mathbf{x})||p(z = k)) &= \sum_{k=0}^{n_{cb}-1} q(z = k|\mathbf{x}) \log \frac{q(z = k|\mathbf{x})}{p(z = k)} \\ &= -\log p(k^*) \\ &= \log n_{cb} \end{aligned} \tag{2.16}$$

It can therefore be discarded from the loss function, such that only the reconstruction loss remains [14].

$$\mathcal{L}_{\text{VAE uniform}}(\varphi, \theta, x) = \mathbb{E}_q(\log p_\theta(x|z)). \tag{2.17}$$

Two additional loss terms. The remaining reconstruction loss in Equation 2.17 does not alter the latent embedding vectors, because the decoder gradient is passed straight to the encoder without affecting the quantization in between. Therefore, additional loss terms need to be added. The authors of the original VQ-VAE paper [14] employ Vector Quantization (VQ) [47], which is an algorithm that maps a continuous vector space into a finite set of vectors, called codeword (in the VQ-VAE paper [14] referred to as *embedding vectors* or *codebook vectors*). This is done by initializing random codewords and moving them to minimize the Euclidean distance between the data points and the corresponding closest codeword. In the VQ-VAE, this is realized by extending the loss by the following *codebook loss* term:

$$||sg[\mathbf{z}_e(\mathbf{x})] - \mathbf{e}||_2^2 \tag{2.18}$$

The expression $sg[\cdot]$ stands for the stop gradient operation, meaning that the encoder outputs $z_e(x)$ are fixed and not altered by taking the gradient. Summing up, this loss term moves the embedding vectors such that they align with the encoder output.
The other loss term that is added is the so-called *commitment loss*. This term is introduced to avoid the norm of the encoder outputs from growing arbitrarily, causing the embedding vectors to grow with them as they try to minimize the loss term from Equation 2.18. The commitment loss is defined as

$$||\mathbf{z}_e(\mathbf{x}) - sg[\mathbf{e}]||_2^2 \ . \tag{2.19}$$

It moves the encoder outputs towards the embedding vectors, i.e., the encoder outputs are encouraged to *commit* to an embedding vector. This ensures that the encoder output and the embedding vectors adapt to each other such that the encoder output cannot increase arbitrarily but is restrained by the embedding vectors.
The overall loss can be summarized by

$$\mathcal{L}_{VQ} = -\log p(\mathbf{x}|\mathbf{z}_q(\mathbf{x})) + ||sg[\mathbf{z}_e(\mathbf{x})] - \mathbf{e}||_2^2 + \beta||\mathbf{z}_e(\mathbf{x}) - sg[\mathbf{e}]||_2^2, \tag{2.20}$$

which is the sum of the reconstruction loss from the original VAE and the two loss terms defined above, where the commitment loss can be weighted by a hyperparameter β. [14]

Replacing the codebook loss term with a moving average. The codebook loss in Equation 2.18 is optimal when the embedding vector $\mathbf{e}$ is the average of all its assigned vectors $\mathbf{z}_e$. This is why an exponential moving average update on each data batch can replace this loss term. This update works by updating the cluster sizes and weighted sums of the input data as defined below:

$$\begin{aligned} N_i^{(t)} &:= N_i^{(t-1)} \cdot \gamma + n_i^{(t)}(1-\gamma) \\ \mathbf{m}_i^{(t)} &:= \mathbf{m}_i^{(t-1)} \cdot \gamma + \sum_j \mathbf{z}_{i,j}^{(t)}(1-\gamma) \\ \mathbf{e}_i^{(t)} &:= \frac{\mathbf{m}_i^{(t)}}{N_i^{(t)}} \end{aligned} \tag{2.21}$$

The variable $n_i^{(t)}$ is the number of input vectors $\mathbf{z}_{i,j}^{(t)}$ that are assigned to the embedding vector $\mathbf{e}_i$ in update step t. The hyperparameter γ can be chosen empirically between zero and one. [14]

After training the VQ-VAE model, the prior over the latent variables can be learned separately to allow generative sampling from the latent space. In the original paper [14], the authors suggest fitting an autoregressive prior to the latent space. They, for example, employ a PixelCNN [48] for images, an autoregressive model that generates images pixel by pixel. [14]

2.6.2. Vector-Quantized Generative Adversarial Network

Esser et al. [15] propose the Vector-Quantized Generative Adversarial Network (VQ-GAN) as an image synthesis model, which extends the VQ-VAE in two ways. First, they adapt the loss function by incorporating an adversarial framework and replacing the usually employed MSE reconstruction loss with a perceptual loss. Second, they propose using a transformer model to learn a prior distribution over the latent space. The adaptations are meant to enable the synthesis of high-resolution images. [15]

Embedding the VQ-VAE in an adversarial framework. Starting with the first point, when embedding the original VQ-VAE in a GAN, the whole VQ-VAE, consisting of the encoder, the quantization step, and the decoder, can be regarded as the generator, as it constructs images that are meant to follow the data distribution p_{data}. In comparison to the vanilla GAN described in Section 2.4, the input of the VQ-VAE generator is not random but a real image, which the decoder tries to reconstruct from a constrained latent space. As the discriminator, the authors use the aforementioned PatchGAN [42] (Section 2.4), which tries to distinguish between real and reconstructed images on a patch level. To formulate the loss function, the VQ-VAE encoder is labeled with $\mathcal{E}$, the decoder with $\mathcal{D}$, and the GAN discriminator with D.

$$Q^\star = \arg\min_{\mathcal{E},\mathcal{D}} \max_D \mathbb{E}_{x\sim p(x)}[\mathcal{L}_{VQ}(\mathcal{E},\mathcal{D}) + \lambda \mathcal{L}_{GAN}(\mathcal{E},\mathcal{D},D)] \tag{2.22}$$

Embedding the VQ-VAE in an adversarial framework is meant to improve the perceptual quality of the reconstructed images. [15]

Loss of the VQ-GAN. For the reconstruction loss, the authors employ a perceptual loss, which is a MSE that is not applied to the pixel space, but in some defined layer of a pre-trained CNN, which allows measuring similarity on a perceptual higher level. [15]

Learning the prior distribution. The second proposal concerns the prior distribution $p(z)$. An image is made up of several "latent space pixels", meaning several codebook indices z. When composing an image, naturally, these indices are not uniformly distributed but depend on each other. Esser et al. propose an autoregressive distribution for image synthesis, i.e., the probability distribution depends on all latent indices that have been selected before. They model this distribution using transformer models that will be introduced in Section 2.7. The transformer is trained on real images that were encoded into the latent space, such that it later can be used to construct latent code sequences

that reconstruct into meaningful images. [15]
An overview of both adaptations of the VQ-VAE, that is, the embedding in the adversarial framework and the integration of a transformer model is given in Figure 2.7.

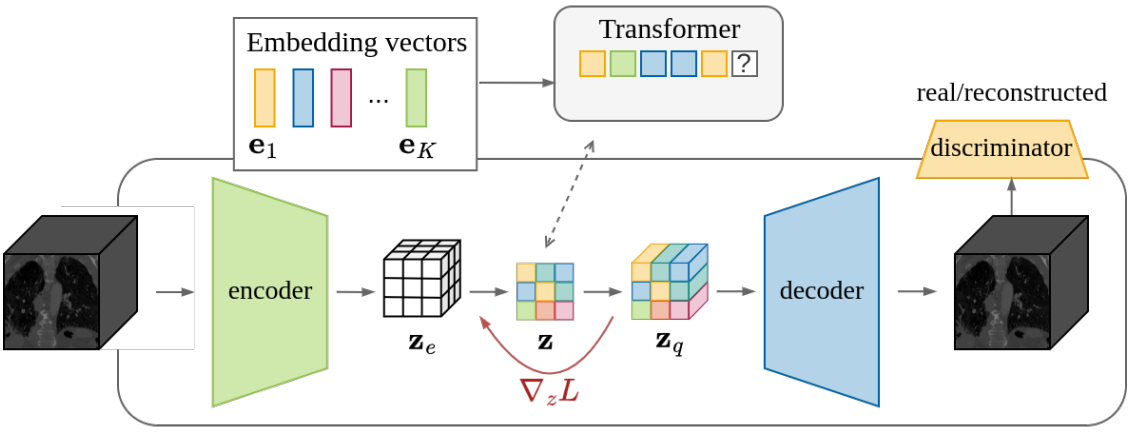


Figure 2.7.: **The VQ-GAN extending the VQ-VAE.** The illustration is inspired by Esser et al. [15].

2.7. Transformer

Transformer models are capable of effectively handling sequential data types, including natural language and time series data. Unlike Recurrent Neural Networks (RNNs) [21], which were the leading models for sequence learning prior to the introduction of transformers (see, e.g., [49], [50]), the transformer training can be parallelized. The transformer model was initially introduced by Vaswani et al. [16] as an encoder-decoder framework tailored for language translation tasks. In this model, the encoder compresses the input sequence into a compact representation, which the decoder then utilizes to produce the output sequence incrementally, token by token, based on the tokens previously generated [16]. Figure 2.8 illustrates the overall architecture.
At a high level, the elements of the input sequence are embedded in meaningful vector representations, called embeddings, and handed to the encoder that consists of N layers with a Multi-Head self-attention block and a Multilayer Perceptron (MLP). The encoder's output is handed to the decoder, as well as the embedded sequence of previous outputs that is meant to be completed. Inside the decoder, the output sequence first passes through a self-attention layer, which is then succeeded by a cross-attention layer that combines the output sequence representation with the encoder output. The decoder steps are also repeated N times. The processed data is then passed through an additional MLP. The output of the final decoder layer proceeds through a linear layer and a softmax activation, producing a probability distribution for the next output token. Between all blocks in the encoder and decoder, the authors use residual connections and layer normalizations. [16]
Next to encoder-decoder architectures, one can also use only encoder layers or only decoder layers (such as [51] or [37]).
The main building blocks of the transformer model are discussed in more detail below.

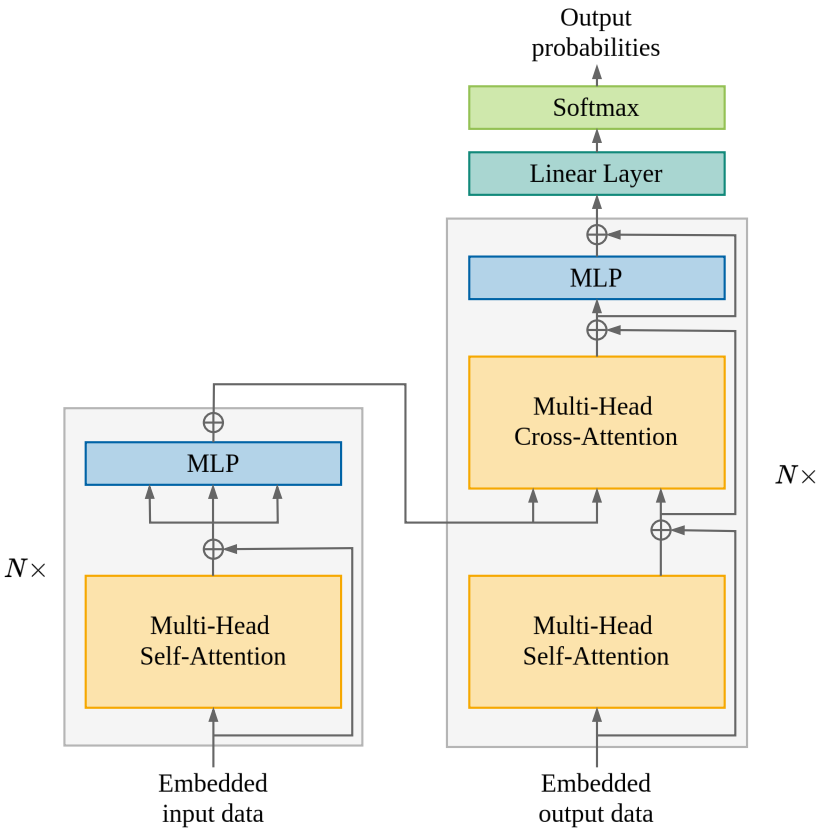


Figure 2.8.: **Overview of the transformer architecture**. The illustration is inspired by Vaswani et al. [16].

Attention. The central component of the transformer is the attention mechanism. It captures the relationships between different tokens of a sequence and is used to extract information from a sequence. In particular, the authors use what they call "Scaled Dot-Product Attention" that can be calculated using the following formula:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.23}$$

The matrix $Q = XW_Q \in \mathbb{R}^{T \times d_Q}$ consitsts of so-called queries, $K = XW_K \in \mathbb{R}^{T \times d_K}$ are the keys, $V = XW_V \in \mathbb{R}^{T \times d_V}$ are the values, and $X \in \mathbb{R}^{T \times d_{model}}$ are the input sequences with fixed sequence length T and embedding dimension d_{model}. The matrices W are learnable weights, whose dimensions d_Q, d_K and d_V are hyperparameters of the model. Intuitively, the dot product calculates similarities between keys (learned properties of the tokens) and queries (learned inquiries for each token) that are used to weigh the learned values that are added to the previous sequence representation to enrich it. The attention can be applied to an individual sequence, where the keys, values and queries are taken from the same sequence, as defined above, which is called *self-attention*. It can also be used such that the keys and values are taken from the input sequence, but the queries are coming from the output sequence ($Q = YW_Q$), which is then called *cross-attention*. It is used in the decoder to combine the sequence representation of the input sequence

with the preceding output tokens. To prevent the cross-attention block from taking all output tokens into account and not only those that precede the token that is predicted, the authors employ a causal mask. That is, they set all elements of the upper triangle of the matrix within the softmax to minus infinity, such that they become zero after softmax.

$$\text{Attention}(Q, K, V, M) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}} + M)V,\ M_{ij} = \begin{cases} -\infty & \text{if } ij \text{ is masked,} \\ 0 & \text{otherwise} \end{cases} \tag{2.24}$$

A similar mask can also be used to ignore padded tokens when processing sequences of unequal lengths. [16]

Multi-Head Attention. The authors discovered that it is advantageous to perform the attention mechanism on multiple query, key and value matrices that are linearly projected with different weight matrices. They call the attention mechanism on a single projection *attention head*. Specifically, the authors project Q, K and V with h different linear projections to the dimensions d_k, d_k and d_v respectively. One head operation can be described as

$$head_i = \text{Attention}(QW_Q^{(i)}, KW_K^{(i)}, VW_V^{(i)}). \tag{2.25}$$

The resulting matrices of dimensions $T \times d_v$ from all heads are then concatenated to obtain a matrix of dimension $T \times (d_v \cdot h)$ which is linearly projected by W_O to the output dimension $T \times d_{model}$.

$$MultiHead(Q, K, V) = \text{concat}(\text{head}_1, \ldots, \text{head}_h)W_O \tag{2.26}$$

[16]

Multilayer Perceptron. In the transformer encoder and decoder layers, there is a MLP at the end. As explained in Section 2.1, it is simply a collection of fully-connected layers. The number of layers and neurons in each layer are hyperparameters of the transformer model.

Layer Normalization. Layer normalization means that all activations of the neurons of one layer are normalized by subtracting the mean layer activation and dividing the result by the standard deviation. Layer normalization can improve the stability and efficiency of the training process. [52]

Input Embedding. Before the input can be passed to the transformer model, it has to be embedded into numerical vectors. The authors use an embedding neural network with learnable weights to project the input (in their case text tokens) to the embedding dimension d_{model}. [16]

Output. After passing embedded sequences through the encoder and decoder layers, the transformer model usually outputs a probability distribution over all possible tokens, indicating how likely a specific token is the target token, which could be, for example, the next translated word in a translation setting. Usually, encoder-decoder transformer architectures are used in an autoregressive setting, meaning that an input and output

sequence is given to the model, which always predicts the next token based on all previously predicted tokens. It is also possible for the decoder to predict multiple tokens independently at once, which, however, does not make much sense in a language setting. [16]

2.8. Selective State Space Model — Mamba

Like transformer models, Selective State Space Models (SSMs) can handle sequential input and output data. In this section, we will specifically focus on the Selective SSM Mamba [17], although there exist various other variants. The Mamba model consists of Mamba blocks that can be stacked onto one another similar to transformer layers. Specifically, a Mamba block takes an input sequence and linearly projects all tokens with two different matrices. The result of the first projection is passed through a convolutional layer followed by a SiLU activation and is then input to a Selective SSM that will be explained in the next paragraphs. The second matrix is passed through a linear layer and a SiLU activation. The two results are combined by either multiplication or addition with a non-linear activation, which is followed by a final linear projection. [17]
An overview of the Mamba block can be seen on the left side of Figure 2.9.

State Space Model. The building block of the Mamba block [17] that is responsible for the sequence processing is the Selective SSM, which is an improvement of the State Space Model (SSM) [53]. A SSM is defined by two equations, where $x(t)$ is a continuous input signal and $y(t)$ a continuous output signal:

$$h'(t) = Ah(t) + Bx(t) \tag{2.27}$$

$$y(t) = Ch(t) \tag{2.28}$$

The first equation is the *state equation* that defines the change of the state at a time point t as the sum of the weighted state at time point t and the weighted input at time point t. The second equation describes how the state is transformed into an output, namely by a linear projection. The matrices A, B, and C are parameters, which means that they are learned to describe the relation between $x(t)$ and $y(t)$. [53]
As the input and output sequences that need to be processed are often discrete (e.g., text), the matrices A and B can be altered such that the SSM can handle discrete signals. To do so, a discrete step size Δ is defined. The discrete input signal can be mapped to a continuous signal using, e.g., the zero-hold technique, and the continuous output signal is discretized by taking samples with step size Δ. Mathematically, these operations can be integrated into the matrices A and B as defined below:

$$\begin{aligned} \overline{A} &= \exp(\Delta A) \\ \overline{B} &= (\Delta A)^{-1}(\exp(\Delta A) - I) \cdot \Delta B \end{aligned} \tag{2.29}$$

The discrete state and output equations can be written as follows

$$\begin{aligned} \mathbf{h}_t &= \overline{A}\mathbf{h}_{t-1} + \overline{B}\mathbf{x}_t \\ \mathbf{y}_t &= C\mathbf{h}_t, \end{aligned} \tag{2.30}$$

where t is now a discrete time step. [17]
Discretized SSMs can be solved by recurrent or convolutional operations. Solving them

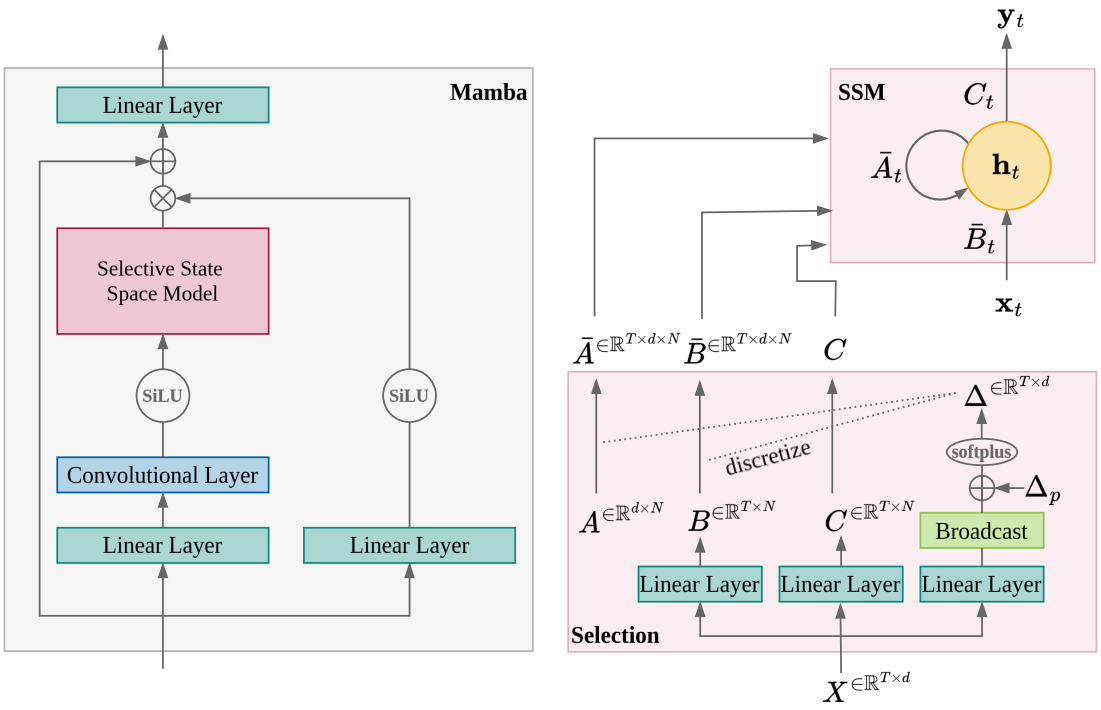


Figure 2.9.: **Overview of a Mamba block.** On the left side, the overall Mamba block is visualized, and the Selective SSM (pink block) is visualized on the right in more detail. The plot on the lower right corner shows the selection mechanism, that is, the calculation of the input-dependent matrices A, B, and C, and the discretization with the input-dependent step sizes Δ. The upper right plot then shows the SSM in its recursive representation. Input tokens $\mathbf{x}_t$ of the input sequence X are passed iteratively (or using the parallel scan) through the SSM, where $\overline{A}_t$, $\overline{B}_t$ and C_t are the respective matrices dependent on $\mathbf{x}_t$.

recurrently comes with the advantage of efficient inference because each token in the sequence needs to be passed to the model only once. Convolutional operations, on the other hand, can be parallelized and therefore enable efficient training. It is therefore common to change the representation and computation strategy depending on whether training or inference is performed. [54]
Furthermore, for efficient computation, it is crucial that the matrix A fulfills certain properties: SSMs are sensitive regarding initialization, which is why A is initialized in a certain way (with a HiPPO-N matrix), and a diagonal structure of A has to be ensured [55].

Selective State Space Models. A notable disadvantage of SSMs is their difficulty in focusing or forgetting specific tokens within a sequence, which can be explained by their time invariance property, meaning that all weights are independent of the input token [56]. Therefore, the authors of the Mamba paper [17] propose to make the matrices B and C as well as the step size Δ dependent on the input. That means that the matrices

get an additional dimension T, so there is a weight matrix for each input token $\mathbf{x}_t$ in a sequence of length T. To obtain the dependent matrices B and C, the input sequence $X \in \mathbb{R}^{T \times d_{model}}$ is linearly projected with two different layers, resulting in the dimensions $T \times N$ for both B and C. For the step sizes Δ, X is also passed through a linear layer, this time with one output neuron, before it is broadcasted to d_{model} dimensions. The result is added to a weight matrix Δ_p, and the sum is passed through a softplus activation, which is a smooth approximation of the ReLU activation. The resulting step size matrix Δ has the dimensions $T \times d_{model}$. This matrix is then used to discretize A and B, to obtain the tensors $\overline{A} \in \mathbb{R}^{T \times d_{model} \times N}$ and $\overline{B} \in \mathbb{R}^{T \times d_{model} \times N}$. [17]
The selection process is visualized in the bottom right corner of Figure 2.9.
The now time-variant tensors $\overline{A}$, $\overline{B}$ and C are passed to an SSM, which is visualized in the upper right corner of Figure 2.9. Note that, due to the dependence of the matrices $\overline{A}_t$, $\overline{B}_t$ and C_t on the input $\mathbf{x}_t$, where $t = 1..T$ is the current time step, the SSM can only be solved recurrently, and a calculation with convolutional operations is no longer possible. The algorithm now lacks the possibility of parallel training, making it as inefficient as a regular recurrent neural network. To enable efficient parallel training, the authors [17] propose three methods: a parallel scan algorithm, kernel fusion and recomputation. The parallel scan algorithm ([55], [57]) exploits the associativity of the linear operation $h_t = \overline{A}h_t + \overline{B}x_t$ to parallelize this operation, similar to a parallel scan for a prefix sum. For details of the parallel scan for the SSM please refer to [55]. To further reduce computational resources, the authors combine multiple operations, i.e., the discretization step, the scan algorithm, and the multiplication with C, into an operation that can be computed by a single compute kernel, meaning an operation specifically designed to be efficiently processed by the GPU [17]. Lastly, the authors [17] employ recomputation, meaning that intermediate states are not stored but instead recomputed during the backward pass, which improves speed as it would take longer to load stored intermediate results from a relatively slow memory than recomputing them. [17]

Complexity. Overall, Mamba's computational complexity during training is linear with respect to the sequence length T [17], whereas, in comparison, the transformer model's complexity is quadratic due to the self-attention mechanism. At inference time, generating a single token in Mamba is independent of sequence length T, as only the single input token has to be entered to get the corresponding output [17], while in a transformer, the time complexity grows linearly with T due to self-attention processing all previous tokens. Note that, naively, the transformer inference would be quadratic in T, which can be avoided by so-called Key-Value Caching [58][1]. The Mamba model, therefore, has significant advantages in computational complexity compared to the transformer, meaning it uses less time and memory, especially when processing long sequences.

2.9. Summary

Summing up, we reviewed multiple deep learning models that will play an essential role in the proposed model. The GAN is a powerful generative network that allows for creating new images by decoding noise inputs [12]. While the GAN is generally capable of

[1] An explanation on Key-Value Caching can be found here: `https://huggingface.co/docs/transformers/kv_cache` (Accessed on April 10, 2025)

generating high-quality images, the latent space is not constrained and therefore often regarded as unstructured, as it does not reliably support simple linear interpolations of images or the consistent sampling of images similar to a particular one [59]. In contrast, the generative VAE usually creates a highly structured latent space. Compared to the GAN, the VAE has the additional capabilities of an autoencoder — it can compress high-dimensional images to lower-dimensional representations [13]. In our proposed model, we make use of the generative and compression abilities of the VAE. However, the VAE sometimes suffers from a phenomenon known as posterior collapse, which results in an unstructured latent space [39, p. 807]. The VQ-VAE avoids this issue by employing a uniform categorical prior [39, p. 815]. A more expressive prior can be later learned by a powerful model, such as a transformer, to represent the semantics of the data by capturing complex dependencies and relationships between the discrete tokens, enabling the generation of high-quality images [14]. Images produced by the VAE and VQ-VAE are still often slightly blurry [39, pp. 797–798], [15]. Embedding the VQ-VAE in an adversarial framework like in the VQ-GAN can help produce more crisp and realistic images [15]. Thus, the VQ-GAN is an enhancement of both the VAE and the VQ-VAE, which is why we use a variant of it in our model.

Transformers can also be employed as generative models to synthesize text (such as [37]) or images (such as [60]). In contrast to GANs and VAEs, they interpret input data as sequences, which makes them suitable for tasks such as predicting the next word in a sentence or the next token of a time series. They can be regarded as inherently conditional since predictions always rely on a sequence of input tokens [16]. These are the reasons why we use a transformer model for the specific task of predicting future tokens based on the given sequence of images in our proposed model. Finally, we observed that Mamba is able to solve similar sequential tasks as the transformer model while being more resource-efficient [17], which is why we investigate exchanging the transformer model with Mamba in later sections.

3. State of the Art of Forecasting Models

This chapter focuses on current deep learning literature directly related to the task of predicting future lung CT images based on a sequence of past scans that are irregularly sampled in time. The task is composed of multiple challenges:

- The general task of **forecasting time series data** has to be solved,
- while dealing with **high-dimensional images**,
- that are additionally **irregularly sampled** over time.
- In addition, it is essential for the predicted images to be **medically coherent**, i.e., the predicted image showing the disease progression in the given lung image should be a reasonable and medically meaningful solution.

Tackling the first challenge, the task could be regarded as multivariate time series forecasting, interpreting different pixels or patches of the CT image as variables that can be extrapolated over time. Papers on time series forecasting, also including papers that specifically tackle the challenge of irregularly sampled time series, are described in Section 3.1. Furthermore, the forecasting of image series is closely related to the task of video forecasting, which addresses the challenge of working with high-dimensional image sequences. Current literature on video forecasting using deep learning models will be reviewed in Section 3.2. Another line of research focuses specifically on predicting pulmonary nodule growth using one or more past CT scans, combining all of the above-mentioned challenges. Papers on this topic are discussed in Section 3.3. The last Section 3.4 provides an overview of other papers dealing with medical image sequence forecasting that are not specifically related to lung cancer.
Please note that, in the following, the terms 'forecasting' and 'prediction' are, like in the related literature, used interchangeably.

3.1. Time Series Forecasting

The task of forecasting CT image time series can be more generally interpreted as a multivariate time series prediction task, where individual image tokens are considered as time series. While time series papers usually do not focus on image data, several papers tackle the challenge of irregularly sampled time series. This section therefore begins with an overview of current work on time series forecasting, before detailing literature that specifically addresses irregularly sampled sequences.
In recent years, research in the field of time series prediction with deep learning models has gained considerable momentum. Of particular note is the superior performance demonstrated by transformer-based models in time series forecasting tasks compared to other architectures like CNN-based or RNN-based models, which Wang et al. determined

in their deep time series model survey [61]. This review therefore mainly focuses on transformer-based models, describing a selection of popular time series forecasting models.

Transformer models for time series forecasting. The transformer model [16] has been applied and adapted in many ways to overcome challenges associated with time series prediction tasks, such as quadratic memory and runtime complexity of the attention module, intricate seasonal patterns and long-term trends, or missing data points.
Autoformer [62] approaches multivariate long-term forecasting by decomposing the series into a trend-cycle and a seasonal part by introducing a series decomposition block to the transformer architecture. Furthermore, it replaces self-attention by an autocorrelation block.
FEDformer [63] also employs a seasonal-trend decomposition. Additionally, time series are projected to a subset of frequency components attained by either the Discrete Fourier Transform or the Discrete Wavelet Transform, and attention is applied to these sparse components in the frequency domain. Both models use a transformer encoder-decoder structure.
Crossformer [64] tackles the correlation between different variables explicitly by applying attention to both the time and the variate dimension separately.
The series are divided into segments in the time dimension, where first attention is applied across the time segments of univariate series, and then the results of all variates are combined by an attention block across the variables. For the latter, a router mechanism is employed to reduce computational resources for high-dimensional data. These attention blocks are integrated into a hierarchical encoder-decoder transformer, allowing for capturing different time scales.
PatchTST [65] does not adapt the transformer architecture, but tokenizes time series into overlapping and non-overlapping patches, and hands channels independently to a transformer-encoder.
Goswami et al. [66] employ a model architecture also based on patching and a simple transformer-encoder, but propose to pre-train it on a large dataset composed of publicly-available data, using a patch-based masking scheme. The resulting model-family MOMENT can be used for various time series tasks, not only for forecasting, and works well on many different datasets. Variables of multivariate time series are passed independently to the model.
CHRONOS [67] addresses univariate time series forecasting by transforming values into a fixed vocabulary that is then handed to existing language models, which are pre-trained on public datasets.
Whereas most of the time series forecasting models apply attention on temporal tokens, iTransformer [68] inverts the time and variate dimension of the input, and instead uses attention on the variate dimension, effectively capturing multivariate correlations in the attention block. Other than the inversion of the input, the authors Liu et al. make no further changes to the employed transformer-encoder.

Alternatives to Transformers for Forecasting. It is worth mentioning that current papers suggest that Selective State Space Models, like Mamba [17], [69], can outperform transformers in various tasks, including time series forecasting [70], [71], [72]. Notably, these models have the benefit of not scaling quadratically in time and memory with the

number of input tokens.

Adaptations for Irregularly Sampled Time Points. While all aforementioned models incorporate some form of positional information into the input time series to inform the model of the ordering of the sequence, none of the papers specifically address challenges like irregularly sampled data and the inclusion of relative or absolute time points. For this reason, the following is a discussion of papers that are specifically concerned with these issues, although they do not explicitly address forecasting.
Tipirneni et al. [73] address classification based on irregularly sampled clinical time series, where different variables can have different sampling time points. They also employ a self-supervision task that forecasts the time series. To address the irregular time points, they propose a triplet representation of a time series, where a single observation is stored as a triplet, which contains the observed value, the variable it corresponds to, and a time point. The triplets are embedded separately in time, values, and variables, where the variables are embedded through a simple lookup, while the times and values are passed through a continuous embedding that the authors present. This continuous value embedding has a single input neuron to which a continuous value can be passed, which is then projected to $\sqrt{d_{model}}$ hidden neurons, where a $\tanh(\cdot)$ activation is applied, followed by a linear projection to d_{model} dimensions. The three embeddings are summed together before applying a transformer-encoder with an additional self-attention block to embed complete time series.
Yalavarthi et al. [74] propose a similar triplet representation to address the probabilistic interpolation of irregularly sampled time series. Their Tripletformer model also represents time series as triplets of time, variable, and value, but unlike the embedding employed by Tipirneni et al., their triplet embedding is constructed by first concatenating time, value, and the one-hot encoded channel to a single vector, which is then passed through a feed-forward layer with a $ReLU(\cdot)$ activation. The embeddings are handed to an encoder-decoder transformer. The inputs for the decoder are embeddings of target time and variable tuples, allowing it to predict values at specific time points and for specific variables.
Chen et al. [75] propose ContiFormer, which tackles various tasks such as the interpolation and forecasting of irregularly sampled time series. They combine transformer models and NeuralODEs to introduce continuity in the time dimension to the usually discrete transformer model.

Although the task of predicting images from past images has many similarities to time series forecasting, there are also several significant differences. In the mentioned literature, time series are long sequences of points that are mostly regularly sampled, apart from the papers specifically focusing on irregular time series. Furthermore, while the different channels of multivariate time series can depend on each other, they have no spatial relation like patches or pixels of an image have. Some of the described papers even omit the dependencies of channels by introducing channel independence, as explained before. Computational resources only become an issue when sequences are very long, but are usually not associated with high-dimensional input, as images or even volumes would be.

3.2. Video Forecasting

While the aforementioned time series literature addresses irregularly sampled data, this is not typically an issue associated to videos, as frames are usually sampled at regular and closely spaced time intervals. Nevertheless, the literature on video prediction is an important foundation for this work, as it specifically addresses the high dimensionality of the input data and provides ideas for generating future images based on past images. A multitude of deep learning tasks are associated with videos, including, but not limited to, video classification, video summarization, and the generation of videos conditioned on text prompts, or on one or more images. Of these, the latter, namely, the prediction of video frames based on images, is the task most closely related to the task at hand. Currently, two main model types are used to address video generation tasks: plain transformer models and diffusion models based on various architectures.

Video Transformers. In their video transformer survey, Selva et al. [76] concentrate on adaptations to transformer models that are specific to the video domain. For transformer models, the main challenge associated with video data is the quadratic scaling of computational resources with the number of input tokens, which has been addressed in several ways. A first step that is usually taken to reduce the high dimensionality of video data is the input embedding. Selva et al. divide the approaches into two main methods: *Minimal embeddings*, where first the video data is tokenized (e.g., into patches, like in the Vision Transformer (ViT) [60]), before it is passed to a few linear layers, and *embedding networks*, e.g., CNNs, that compress the data in the spatial or spatio-temporal dimension, by exploiting their locality inductive bias. In the current literature, models based on VQ-GAN [15] are commonly used as embedding networks to perform video tasks in the lower-dimensional latent space [77]. After embedding, the inputs are passed to the transformer model. Typically, encoder-only architectures are used for fixed output sizes, while decoder-only architectures are suited for variable-length outputs. As the quadratic complexity of the attention blocks remains a challenge even in reduced latent space, there have been several advances in restricting attention to specific areas. Selva et al. mention options such as restricting attention to certain local neighborhoods, restricting attention to certain axes, e.g., by decomposing temporal and spatial attention, or using sparse attention, e.g,. by using fixed strided patterns. [76]
To name a few video generation transformers, Yan et al. [78] propose VideoGPT, a model based on a 3D VQ-VAE embedding network, and an autoregressive transformer model to predict the latent codes of the next frame based on previous frames.
Gupta et al. [79] employ VQ-GAN and a bidirectional transformer with decomposed windowed spatial and spatio-temporal attention. During training, they propose a masking schedule to mask random patches in future frames, such that the resulting model is able to perform iterative decoding, so predicting a few patches in the future first, and then sequentially filling in the gaps.
Yu et al. [80] present a more general model, which is able to address various video tasks. They also use an autoencoder based on VQ-GAN and a bidirectional transformer, but introduce several masking schemes that allow for multitask learning.

Video Diffusion Models. Next to transformer models, diffusion models have shown strong generation capabilities. Diffusion models are a type of generative model that

gradually transforms simple noise into complex data, such as images, by learning the reverse process of a progressive noise addition. During training, they learn how to reverse the noisy degradation of data by modeling the probability distribution of the original data at each step of a gradual noise schedule. At inference, the model starts with random noise and iteratively denoises it, generating a realistic sample that mimics the training data. [77]
The following papers address the topic of video generation conditioned on images, among other things, by employing diffusion models. Ho et al. [81] use a 3D U-Net model with diffusion, to which they add factorized spatial and temporal attention. For conditioning the model on past frames, they propose "reconstruction-guided sampling", which allows the authors to autoregressively expand the video during inference.
Zhou et al. [82] perform the diffusion process in a lower-dimensional latent space obtained by a VAE that encodes videos frame-wise. The U-Net diffusion model then learns the video latent distribution of key frames, which are later interpolated.
Wang et al.'s diffusion model [83] also operates on a latent space. The authors present a "Spatio-Temporal Condition encoder" based on VQ-GAN that can encode various conditions such as images, sketches, or motion vectors. The different conditions are added up and concatenated to the input noise. Text-guidance is given by CLIP model embeddings, which are handed to the model through cross-attention blocks.
For a more detailed overview of diffusion models in video tasks, please refer to the review of Melnik et al. [77].

3.3. Lung Nodule Growth Prediction

The task of predicting the growth of lung nodules has been addressed by various researchers, also, like in this thesis, by generating future CT images.
Zhang et al. [9] address the task of predicting a future 3D CT image from two past CT images with irregular time intervals. The CT images contain nodules extracted and registered manually by experts. The authors employ a convolutional Long Short-Term Memory (LSTM) [84] network, which they extend to handle both 3D spatial and temporal information. Each of the LSTM units processes a single image slice and receives information from neighboring slices in both the spatial and temporal direction. The units are all embedded in a convolutional encoder-decoder structure, where the time interval between the consecutive images is concatenated to the encoded input.
Sheng et al. [8] address a similar task, also taking two nodule CT images as input and predicting a future nodule image. They approach the task with a U-Net-like network that processes the difference image between the first two time points and estimates the difference between the second and target time point by calculating the displacement field between the two difference images and then warping the input difference image. The two associated time intervals are concatenated to the encoded input.
Unlike the above-mentioned papers, the following papers establish their predictions of future images on the basis of a single past image, as opposed to a sequence of previous images.
Li et al. [10] propose a combination of two networks based on U-Net: a warp network that estimates the future shape of a nodule by estimating the displacement field of both the nodule image and its binary nodule mask, and a texture network that predicts the

change in pixel intensities. In both networks, a temporal encoding of the time interval to the target based on sinusoids is concatenated to the internal nodule representation.
Fang et al. [11] present a two-step approach, where they first estimate the future volume and mass of a nodule estimated by a learned parameterized Gompertz function, which are then used to guide a second network that predicts the future image. Like in the above paper by Li et al., the authors divide this second network into a shape estimation and a texture estimation network. The time interval is implicitly embedded into the future volume and mass that are predicted for a specific time point, which serve as additional input to the shape and texture networks.
Wang et al. [7] address a slightly simplified task of predicting future nodules based on a 2D CT image in one year's time. They employ a Wasserstein GAN with a U-Net-like predictor network. Performance is assessed by evaluating the results of a lung cancer risk prediction model, comparing its output given a single image, two consecutive real images, and one real image and one predicted image. The results demonstrate that the supplementary predicted image can enhance performance and is comparable to the outcomes of two real images, suggesting that predicted future images can help doctors make earlier treatment decisions.

3.4. Other Medical Image Forecasting Models

Next to papers explicitly addressing lung nodules, there are similar papers on other medical image progression forecasting tasks, like the development of tumors in other regions of the body.
Yoon et al. [85] address the task of generating medical images based on a sequence of past images, which can have various lengths and missing frames. They propose a diffusion model that is conditioned on past images processed by an attention block. They factorize attention into temporal and spatial attention and use the resulting compressed sequence representation to condition the model through classifier-free guidance. To overcome irregular sampling, missing frames are imputed by the model.
Petersen et al. [86] tackle glioma growth prediction by adapting Neural Processes in the imaging domain by adding a convolutional encoder-decoder structure and attention. The proposed model implicitly handles continuous-time input data.
Bai et al. [87] propose a model that regresses an image sequence with potentially missing time points. They describe the image deformation over time using a first-order ordinary differential equation that is formulated through a neural network. This Neural ODE is optimized in a lower-dimensional latent space using a pre-trained encoder and decoder.

4. Data

This chapter details the data that will be used for training and validating the deep learning model for forecasting future CT images. It starts with a general description of the internal dataset. Thereafter, all preprocessing steps are explained, encompassing the selection of suitable images, respacing of images, and image registration, as well as nodule detection. Finally, the split of the data into training, validation, and test set is outlined.

4.1. Data Description

The dataset originates from a retrospective clinical routine cohort at the Vienna General Hospital, assembled by the Medical University of Vienna. It encompasses patients diagnosed with lung cancer from 2002 to 2022, each having at least one CT scan. The specific diagnosis codes based on ICD-10 [88] comprise C34.0, C34.1, C34.2, C34.3, C34.8, C34.9 and D02.2. Codes starting with C34 denote cancerous tumors in the lung, with the last digit (0, 1, 2, 3, 8, 9) indicating the location, and D02.2 represents carcinoma in situ in the lung. The CT scanners utilized come from a range of manufacturers, including Siemens, Philips, GE Healthcare, and Toshiba.
Overall, the dataset comprises 4,730 patients with 65,154 CT images. Some of the patients underwent surgeries or received other treatments, but this data is not considered in this thesis, as it is still incomplete. The dataset contains all CT images taken of the selected patients, including images of the whole body and even images that do not show the lung. The images can have different orientations (depending on the patient's position in relation to the scanner) and various resolutions. Each patient can have CT scans taken on one or multiple dates. For one specific date, there can exist multiple scans for a patient. This is caused by, e.g., rescans due to motions or other artifacts, additional scans focusing on specific regions, or different image reconstructions of a single scan. The reconstruction process of a CT image, that is, the reconstruction of a 3D image from the raw scanner data, defines the quality of the reconstructed image [89]. The choice of the convolutional kernel in the reconstruction controls the trade-off between sharpness and noise in the image [89]. Often, multiple image reconstructions are created for different types of assessments [89]. The intensity values in the reconstructed CT images are given in Hounsfield units that measure radio density [90, p. 409]. A value of $-1,000$ corresponds to the density of air and zero to the density of water [90, p. 411].

4.2. Data Preprocessing

The original $4,730$ patients and $65,154$ CT images have to be filtered and preprocessed to use them as input to neural networks. The goal is to create two different datasets:

- **Dataset A: Lung CT images.** For training an autoencoder, as will be described in the next chapter, all CT images that show a lung can be used.

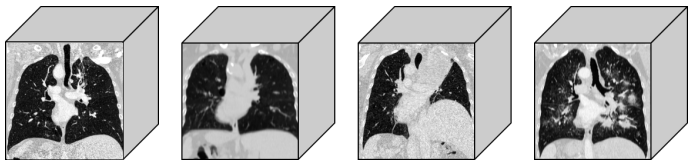


Figure 4.1.: **Dataset A: Lung CT images.** This dataset consists of all available lung CT images. Four random examples are shown in this figure.

- **Dataset B: Coherent time series of CT images**: The forecasting task requires sequences of lung images, where the images are registered over time and have little variablility in image characteristics.

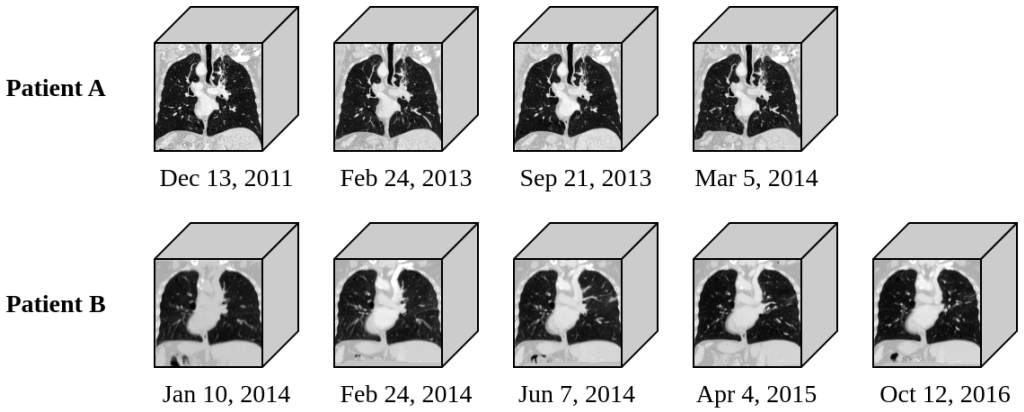


Figure 4.2.: **Dataset B: Time series of lung CT images.** This dataset contains registered patient-wise time series like shown for two examplary patients in this figure.

In order to create these datasets, several processing steps are taken. An overview of the complete process with details on how many images are discarded in each step is given in the flowchart in Figure 4.3.

Data Availability. CT images are not included if they were missing in the internal database or did not have a lung segmentation. Segmentations of the left and right lung were previously extracted using the code on github[1] by Hofmanninger et al. [91].

Cropping and respacing. All remaining images are rescaled and linearly interpolated to ensure isotropic resolution, meaning that they have a resolution of 1mm in each spatial direction. The images are cut out based on bounding boxes of the lung segmentations.

Orientation. For many images that were not in the standard orientation, the conversion from DICOM[2] to NIfTI[3] posed a major issue, and the segmentations couldn't be aligned correctly, which is why all images with non-standard orientation were removed. In general,

[1] `https://github.com/JoHof/lungmask`
[2] DICOM standard: `https://www.dicomstandard.org/`
[3] Neuroimaging Informatics Technology Initiative (NIfTI): `https://nifti.nimh.nih.gov/`

it is possible to align images to a standard orientation; however, because of the mentioned technical issues, this step was skipped to expedite the process.

Removal of non-lung CT images. As described earlier, not all images show the lung. These images are filtered out based on the size of the cut-out bounding box (lower threshold 10,000,000 mm^3), the volume of the segmentations (lower threshold 2,000,000 mm^3), and the intersection of the segmentation with the image boundary (before cutting, upper threshold 2,500 mm^3). The latter step is meant to prevent lungs that are partly cut off from being accepted, as they could cause issues during registration. The resulting isotropic and bounded lung CT images are all part of the first dataset for the autoencoder, namely *Dataset A*.

Time series for forecasting. For the forecasting, we require at least three time points for each image sequence: two for interpolation of the past, and one as future target point for training and validation. Therefore, after each of the following processing steps, sequences that contain less than three images are removed. Sequences are extracted patient-wise: there is exactly one sequence for each patient. Subsampling of these sequences to extend training data is done in a later step explained in Section 5.6.2.

Kernel selection. To prevent the forecasting model from managing scanner-specific variations in images, a single kernel is chosen for each sequence, specifically the one appearing in the most time points. If multiple images with the same kernel are present for a time point, the one with the highest original resolution is selected. If the selected kernel is not present in the CT images at a specific acquisition date, this date with the corresponding images will not be part of the dataset.

Registration. All images in the resulting sequences are registered to the first image of each sequence using the "Advanced Normalization Tools in Python" (ANTsPy)[4] library. In the first step, the lung segmentations are registered using an affine transformation. If the quality of the registration result is low, measured by a Dice score between the fixed and moving segmentation (lower threshold 0.5), the time point is discarded from the data. If the quality is sufficiently high, the lung images are registered using a symmetric normalization composed of an affine and a deformable transformation, with mutual information as an optimization metric. The transformation is initialized with the transformation resulting from the registration of the associated segmentations. If the mutual information of the result is too low (lower threshold 0.4), the time point is discarded. The transformations and thresholds were chosen empirically, based on visual registration results. The registered time series form the forecasting dataset, that is, *Dataset B*.

4.2.1. Extraction of Lung Subvolumes

To maintain a relatively high resolution of $1mm^3$ while staying within the constraints of available computational resources, the prediction of future CT images is not conducted on the whole lung CT images but on smaller, cubic subvolumes of the images. This also comes with the advantage that CT images with different sizes do not have to be cropped

[4] `https://github.com/ANTsX/ANTsPy`

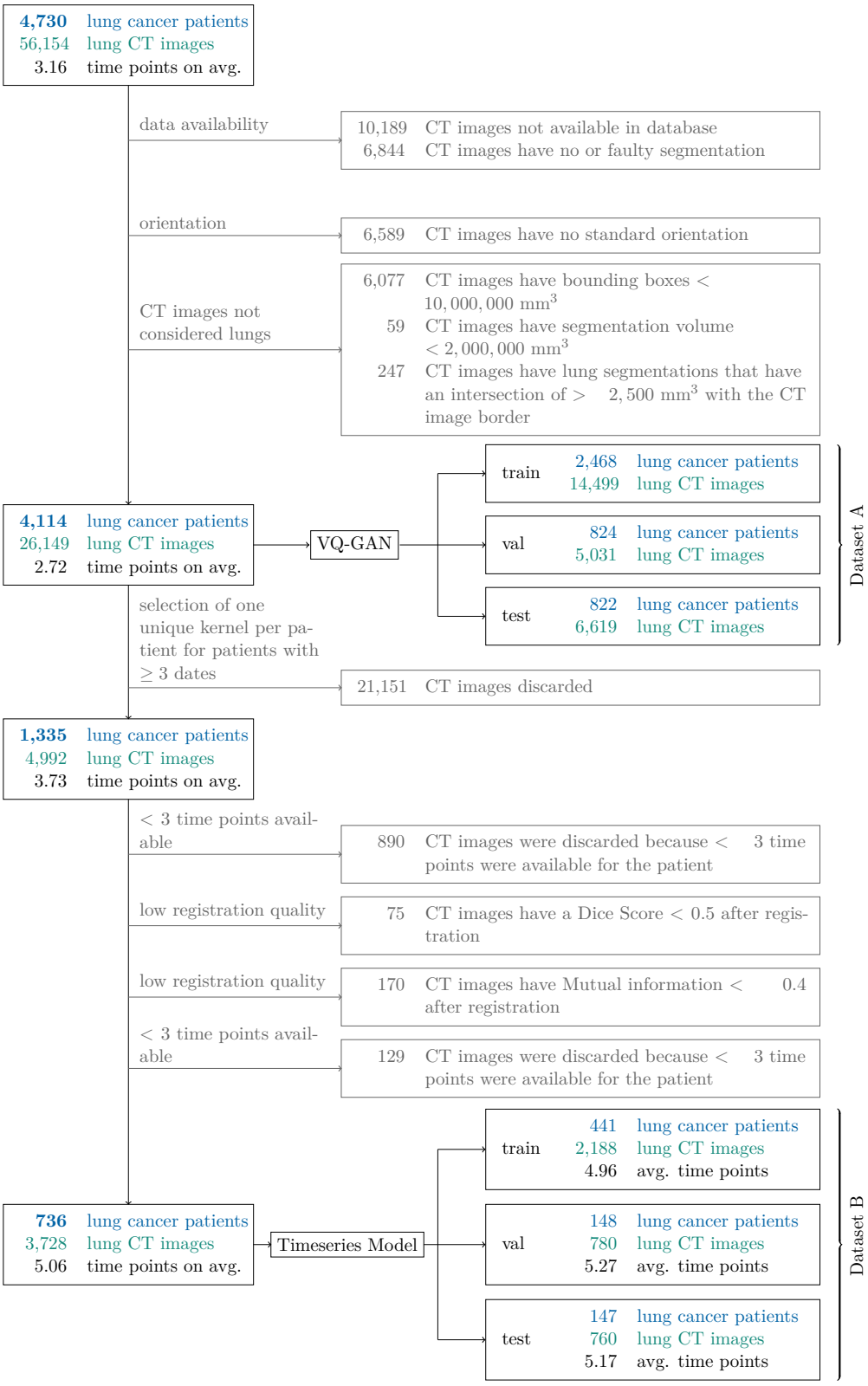


Figure 4.3.: Flowchart of the data preprocessing pipeline.

or padded to a fixed size. The length l_v of one side of the cubic subvolume will be a hyperparameter of the model. The process of obtaining the subvolumes (subsequently called **v**) can be done through the **extraction at random lung positions** or **extraction at the positions of lung nodules**, depending on their intended application within the model.

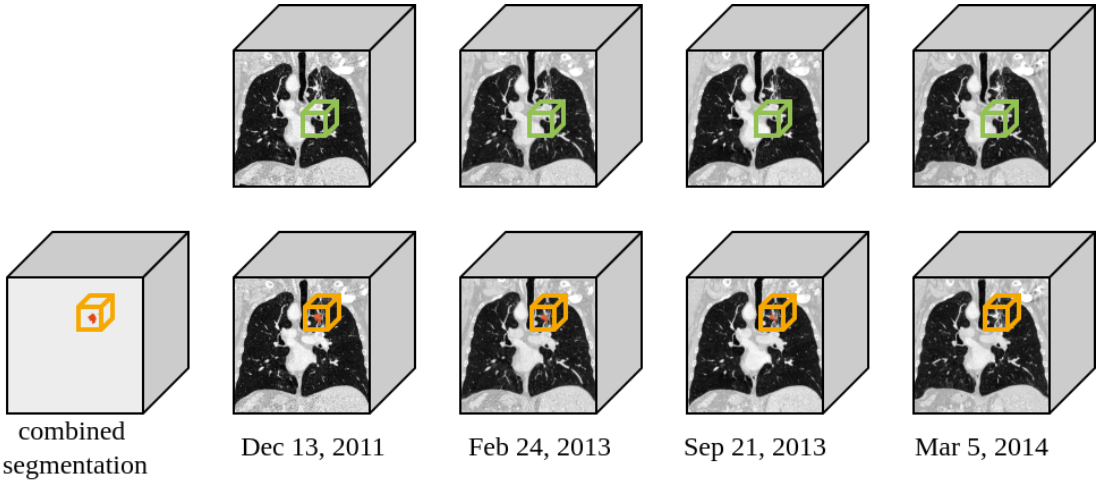


Figure 4.4.: **Subvolume extraction.** The top row shows the extraction of a subvolume at a randomly chosen position in the lung. The same subvolume is extracted for all available time points. The bottom row shows the extraction of a subvolume of a position based on available nodule segmentations. All segmentations of all time steps are combined into one (volume on the left), which is the basis for the choice of the subvolume position. The subvolume is centered around the found nodule. The same subvolume position is extracted at all time points.

Extraction of subvolumes at random lung positions. To extract a subvolume, a position within the lung (as indicated by the segmentation mask) is randomly drawn as the center of the subvolume. Furthermore, to ensure that all patches show at least a specific ratio of lung tissue, a subvolume is only accepted if at least 50% of all pixels included are inside the lung according to the lung segmentation mask.

Extraction of subvolumes at the position of lung nodules based on automatic nodule segmentations. Random subvolumes can bias the model toward predicting healthy, non-changing tissue. Therefore, we also extract volumes that are more likely to contain lung nodules. This extraction is based on the automatic segmentation of tumors.
The automatic tumor segmentation is performed by a segmentation model, namely a 3D nnUnet [92], which was trained by our research group. The data that was used for the segmentation model consists of CT images of 22 patients with a total of 37 visits and three CT images per visit reconstructed with different kernels, where lung tumors and lung metastases were annotated by physicians. In addition, 262 lung tumor CT images from public datasets [93]–[96] were used. The segmentation model was trained on 18

internal patients and 262 CT images from public datasets, and evaluated on four unseen patients (6 visits). The lung tumor segmentator achieved Dice scores between 0.72 and 0.73 depending on the kernel of the unseen patients.
We use this model to obtain nodule segmentations for all CT lung images in the time series dataset, i.e., *Dataset B*. The resulting segmentations from the segmentation model are binary masks indicating tumorous pixels. All masks of a sequence are combined into a single mask by a bitwise OR operation because we need to extract the same subvolumes from each image in the sequence. For our forecasting task, all connected components are identified in the combined masks, where connectivity is defined by a full neighborhood, meaning that all 26 adjacent pixels are considered connected. In the next step, for each connected component, a bounding box is calculated to contain the whole tumor. If the size of a connected component, that is, the number of connected pixels, is smaller than the subvolume size ($l_v{}^3$), a subvolume is extracted centered on this bounding box. If the segmented nodule is greater than the subvolume, multiple centers are randomly sampled along the border of the nodule. This is done because the center of a nodule is less informative than its boundaries for disease progression. The boundary pixels are extracted by taking the difference between each mask and its binary dilated version, from which random centers for the subvolumes are drawn. The extraction numbers for training, validation, and test set can be seen in Table 4.1. The split of the dataset into training, validation, and test set is detailed in Section 4.3.

	Training set	Validation set	Test set
# subvolume sequences containing nodules	7,062	2,232	2,906

Table 4.1.: Overall number of subvolume sequences extracted based on the automatic nodule segmentor in *Dataset B*.

Filtering of subvolume sequences extracted based on nodule segmentations. Manual inspection of time series of subvolumes extracted based on the nodule segmentor, as explained in the previous section, has revealed that some sequences are very discontinuous for various reasons: poor registration quality, effusions, or surgeries, etc. lead to sudden changes in the CT image sequences. As such sudden changes could hinder the learning process of the forecasting model, we decided to automatically remove some of these samples. To do so, for each subvolume sequence, the Structural Similarity Index Measure (SSIM) (explained in Section 6.1) is calculated for all adjacent subvolumes. A low SSIM value means that two images are very dissimilar. Thus, if the minimum of all SSIM values is below a defined threshold, the whole sequence is discarded. Table 4.2 shows the number of samples in training, validation and test set for different thresholds.

Validation data grouped based on the time gap between the last two images. For all experiments in Chapter 6, in which the *Dataset A* is used, nodule subvolumes filtered with a threshold of 0.1 are employed. For evaluations of extrapolation results, the validation set is additionally divided into groups based on the time gap in days between the second most recent and the most recent available image. We choose the following three groups:

	Training set	Validation set	Test set
No filtering	7,062	2,232	2,906
Threshold 0.1	6,474	2,051	2,745
Threshold 0.3	4,661	1,451	1,983

Table 4.2.: Number of nodule subvolume sequences in *Dataset B* with different thresholds on the minimal SSIM value of adjacent images in a time series.

1. The time gap between the last two images is less than or equal to 60 days.
2. The time gap is greater than 60 days but smaller than or equal to 180 days.
3. The gap is larger than 180 days.

Table 4.3 shows the distribution of the validation sample into the three groups.

	$\leq$ 60 days	61–180 days	$>$ 180 days
# samples	370	1,238	443

Table 4.3.: Number of validation time series in *Dataset B* in each group, where groups are based on the time distance between the two most recent images.

4.3. Training, Validation and Test Data

The two datasets, one containing CT images, namely *Dataset A*, and a subset containing coherent time series, namely *Dataset B*, are divided into training, validation, and test sets on patient level. The time series dataset (*Dataset B*) is divided into 60% training, 20% validation, and 20% test data. The 20% of patients in the test set of *Dataset B* are also part of the test set of *Dataset A*, such that the final evaluation can be made on data that has not been seen by either of the models. The remaining patients in *Dataset A* set are divided such that the resulting training, validation, and test data are also distributed 60%, 20%, and 20%. As further research on this project and data is planned, in this work, only the validation set is used for evaluations, such that the test set remains available for later, unbiased evaluation. Therefore, all validation metrics shown in the experiments can be slightly biased, since they might have been overfitted to the validation data.

5. Methodology

In this chapter, we present a medical time series model designed to solve the task of forecasting irregularly sampled time series of CT scans. Unlike previous work ([8], [10], [11]) that predicts lung nodule growth using deep learning models (described in Section 3.3), our model does not predict displacement fields. Instead, it is based mainly on the literature on video forecasting [78], as, to the best of our knowledge, these approaches have not been applied to the task at hand. Specifically, our model is inspired by VideoGPT [78], where a powerful model performs forecasting on a reduced latent space. This aims for the identification of complex sequential patterns within the data. To include irregularly sampled time intervals, we extend our model by a triplet embedding inspired by Yalavarthi et al. [74] and Tipirneni et al. [73]. The details of our model are described in the following sections.

5.1. Model Overview

The goal is to predict a future CT image based on a time series of past CT images. We denote each CT image subvolume as $\mathbf{v}_p^{(t_i)}$ where p is the patient identifier and t_i is the i^{th} time point of the available time points of patient p. That means $(\mathbf{v}_p^{(t_i)})_{i=1}^{T_p}$ is the time series of CT subvolumes of patient p for whom images were taken at T_p time points. Note that there can be multiple subvolumes at different spatial positions taken for a patient, which we do not include in the indices of the variable to improve readability.
We propose a two-step approach, inspired by VQ-GAN [15] and VideoGPT [78]. In the first step, an encoder-decoder model is trained, which is able to reduce the dimension of individual CT scans by projecting them into a constrained latent space and to reconstruct CT scans from latent space representations. The latent space learned by the encoder-decoder model needs to be structured such that newly created latent space representations can be meaningfully reconstructed to realistic CT subvolumes.
The encoder of the trained model is used to encode each subvolume $\mathbf{v}_p^{(t_i)}$ to a compressed latent space representation $\mathbf{z}_p^{(t_i)}$. Note that the encoder receives no information about time, patients, and connections between images.
In the second step, a transformer model is employed to forecast a latent space time series of encoded images. The input to the transformer model are sequences of encoded lung subvolumes $(\mathbf{z}_p^{(t_i)})_{i=1}^{T}$, flattened and enriched by spatial positions and time information. Note that we usually do not use all available images of a patient for the transformer input, as will be detailed later, which is why the length of the input time series is now denoted by T instead of T_p, where $T \leq T_p$ always holds. From this input, the transformer model predicts a future image $\mathbf{z}_p^{(t)}$ in the latent space at a given time point t.
The predicted future latent patch $\mathbf{z}_p^{(t)}$ can then be handed to the existing decoder to obtain the final prediction image $\mathbf{v}_p^{(t)}$ in the original pixel space. An overview of the model is given in Figure 5.1.

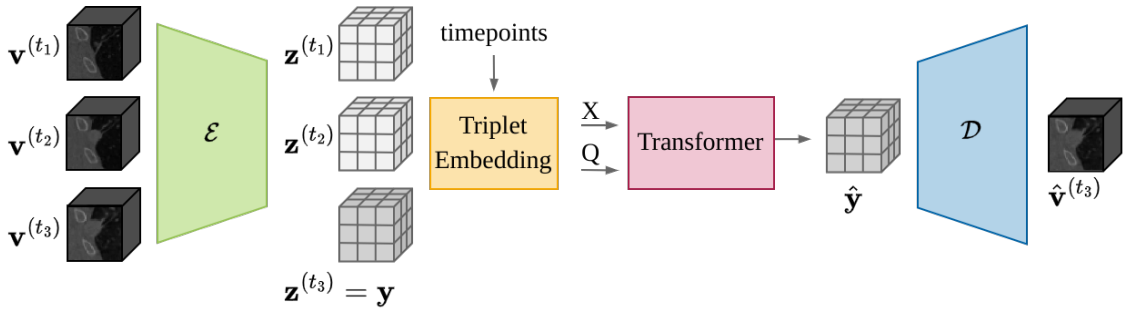


Figure 5.1.: **General overview of the proposed model.** A time series of three 3D lung subvolumes $\mathbf{v}$ at different time points t_1, t_2 and t_3 is given. During training, the image at the most recent time point t_3 will serve as the target. First, all three subvolumes are encoded into the latent space, resulting in the patches $\mathbf{z}^{(t_1)}$, $\mathbf{z}^{(t_2)}$ and $\mathbf{z}^{(t_3)}$. The patches $\mathbf{z}^{(t_1)}$ and $\mathbf{z}^{(t_2)}$ serve as input to the transformer model and are handed to the triplet embedding, while $\mathbf{z}^{(t_3)}$ is the transformer target, which we denote by $\mathbf{y}$. The triplet embedding combines the input images with the corresponding time points to form the input set X, and takes the target time point to form the query set $\mathbf{q}$. Both are handed to the transformer forecasting model that estimates the target $\mathbf{y}$ in the latent space. The prediction is then decoded back to the pixel space to get the final estimation of the future lung volume $\mathbf{v}^{(t_3)}$.

5.2. Latent Representation of Lung Subvolumes

In the first step, an autoencoder is used to compress all images $\mathbf{v}_p^{(t)}$ independently to lower-dimensional representations $\mathbf{z}_p^{(t)}$, which capture the most important image properties. Additionally, we aim to create a structured latent space from which we can also decode new meaningful images. Because the proposed encoder-decoder model compresses individual images independent of the patient and time series, we omit the indices p and t_i from the subvolumes $\mathbf{v}_p^{(t_i)}$ in the following section.

5.2.1. VQ-VAE in an Adversarial Framework: VQ-GAN

For the compression, we employ an encoder-decoder model for which we mostly follow the VQ-VAE [14] that is explained in Section 2.6.1. Specifically, we use the VQ-VAE implementation of the Medical Open Network for Artificial Intelligence (MONAI) framework[1] that provides a wide range of deep learning models for medical imaging, including generative models [97]. This VQ-VAE implementation also embeds the model into an adversarial framework, which is why we will refer to it as VQ-GAN instead.

VQ-GAN Architecture. The VQ-GAN consists of an encoder $\mathcal{E}$ that gets lung subvolumes $\mathbf{v} \in \mathbb{R}^{l_v \times l_v \times l_v}$ as input, applies downsampling convolutions, and finally outputs a tensor $\mathbf{z}_e \in \mathbb{R}^{l_z \times l_z \times l_z \times d_{cb}}$, where d_{cb} is the dimension of the embedding vectors in the latent space.

[1] `https://monai.io/`, VQ-VAE implementation from `https://github.com/Project-MONAI/GenerativeModels`

The latent size l_z can be altered by adapting the convolutions in terms of padding, stride, or number of layers. In the next step, each vector in the output tensor $\mathbf{z}_e$ is mapped to its closest embedding vector $\mathbf{e}$, where each of the embedding vectors is assigned to a number between zero and the number of codebook vectors (minus one as we start with zero) $n_{cb} - 1$, which we call the codebook index. The resulting quantized tensor $\mathbf{z}$ then has the dimension $l_z \times l_z \times l_z$. This operation is the Vector Quantization (VQ) operation, and will be denoted by $\mathbf{q}(\cdot)$. To decode a quantized tensor $\mathbf{z}$, each codebook index is mapped back to the corresponding codebook vector by a simple lookup. Refer back to Section 2.6.1 for more details. The resulting tensor $\mathbf{z}_q$ is then handed to upsampling convolution layers to finally arrive at a tensor $\hat{\mathbf{v}}$, the reconstruction of the input $\mathbf{v}$. The operations can be summarized by the following formulas.

$$\begin{aligned} \mathbf{z} &= \mathbf{q}(\mathcal{E}(\mathbf{v})), \\ \hat{\mathbf{v}} &= \mathcal{D}(\mathbf{z}) \end{aligned} \tag{5.1}$$

The encoder $\mathcal{E}$ consists of n_l downsampling convolutions that downsample the input by the choice of an appropriate stride, each followed by n_{rl} residual blocks as defined in Section 2.2. A final convolution is applied that keeps the dimension the same, but reduces the number of channels to the given embedding vector dimension d_{cb}. The decoder is chosen to be symmetric to the encoder, but applies upsampling convolutions, that is, transposed convolutions, instead of the downsampling convolutions applied in the encoder. The overall architecture of the VQ-GAN (without the adversarial framework) can be seen in Figure 5.2.

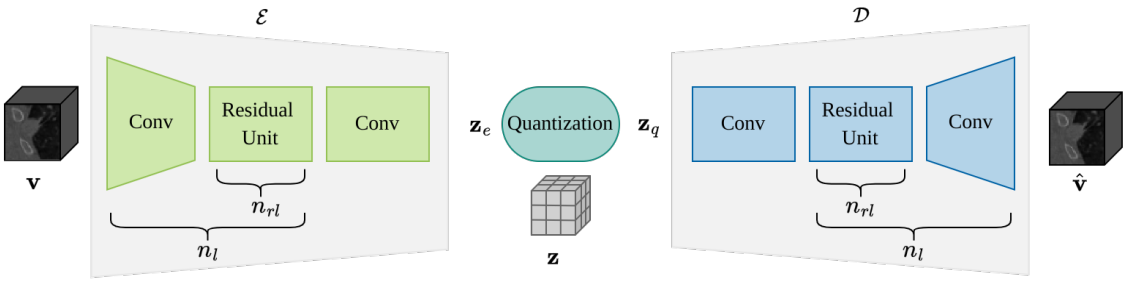


Figure 5.2.: **VQ-GAN architecture.** The VQ-GAN consists of an encoder composed of downsampling convolutions and residual units, a quantization step, and a decoder symmetrical to the encoder.

Similarly to the explained VQ-GAN paper (Section 2.6.2), the VQ-VAE is embedded in an adversarial framework. Specifically, we define the generator as

$$G(\mathbf{v}) = \mathcal{D}(\mathbf{q}(\mathcal{E}(\mathbf{v}))) = \hat{\mathbf{v}} \tag{5.2}$$

and use a PatchGAN discriminator $D(\mathbf{v})$ that learns to distinguish between real images $\mathbf{v}$ and reconstructed images $\hat{\mathbf{v}}$. In comparison to the VQ-GAN paper [15], we do not employ a perceptual loss, but an L1 loss on the pixel space images. The embedding of the VQ-VAE in the GAN framework is depicted in Figure 5.3.

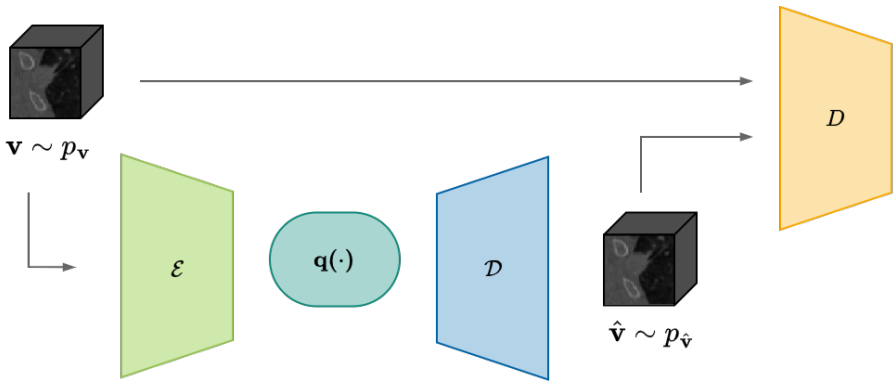


Figure 5.3.: **VQ-GAN: Embedding of the VQ-VAE in the adversarial framework.** The discriminator tries to distinguish between samples $\mathbf{v}$ that are drawn from the distribution of real lung subvolumes $p_{\mathbf{v}}$, and instances $\hat{\mathbf{v}}$ from the distribution of reconstructed samples $p_{\hat{\mathbf{v}}}$, that means samples that have been encoded, quantized and decoded back.

Loss of the VQ-GAN. The overall loss function is composed of the VQ-VAE loss (Equation 2.20) and the adversarial loss. As described in Section 2.6.1, the VQ-VAE loss encompasses the reconstruction loss, for which we employ an L1-loss, and the codebook loss, which moves the data points towards the assigned codebook vector. The codebook loss is replaced by exponential moving average updates, as explained in Section 2.6.1 and defined in Equation 2.8. The complete VQ-VAE loss is shown in Equation 5.3 and has to be minimized.

$$\mathcal{L}_{VQ} = \mathcal{L}_{L1}(\hat{\mathbf{v}}, \mathbf{v}) + \beta||\mathbf{z}_e(\mathbf{v}) - sg[\mathbf{e}]||_2^2 \tag{5.3}$$

For the GAN, we employ the Least Squares GAN, as explained in Section 2.4.

$$\begin{aligned} \mathcal{L}^D_{\text{LSGAN}} &= \frac{1}{2}\mathbb{E}_{\mathbf{v}\sim p_{\mathbf{v}}}\left[||D(\mathbf{v}) - \mathbf{1}||_2^2\right] + \frac{1}{2}\mathbb{E}_{\mathbf{v}\sim p_{\mathbf{v}}}\left[||D(G(\mathbf{v}))||_2^2\right] \\ \mathcal{L}^G_{\text{LSGAN}} &= \frac{1}{2}\mathbb{E}_{\mathbf{v}\sim p_{\mathbf{v}}}\left[||D(G(\mathbf{v})) - \mathbf{1}||_2^2\right] \end{aligned} \tag{5.4}$$

The overall generator loss is the sum of the VQ-VAE loss and the weighted Least Squares GAN generator loss, which is minimized during training.

$$\mathcal{L}_{\text{Gen}} = \mathcal{L}_{\text{VQ}} + \lambda_{\text{GAN}}\mathcal{L}^G_{\text{LSGAN}} \tag{5.5}$$

The discriminator loss is the weighted Least Squares GAN discriminator loss as shown below, which also needs to be minimized.

$$\mathcal{L}_{\text{Disc}} = \lambda_{\text{GAN}}\mathcal{L}^D_{\text{LSGAN}} \tag{5.6}$$

For the next step, namely the forecasting of the future, all images are projected into the latent space by the trained encoder and quantized into codebook indices. That is, after this first step, we end up with time sequences of latent space patches $(\mathbf{z}_p^{(t_i)})_{i=1}^{T_p}$, that is, encoded and quantized CT subvolumes, instead of the original CT subvolumes $(\mathbf{v}_p^{(t_i)})_{i=1}^{T_p}$.

The forecasting model solely operates on quantized latent space images that will from now on be referred to as *patches* $\mathbf{z}$. The latent space structure should thereby allow for the decoding of new latent patches that the forecasting model predicts. Note that, despite the mentioned structure in the VQ-GAN latent space, the codebook indices do not have an intuitive ordering, that is, codebook indices that are close to each other do not necessarily decode into similar pixel representations. This property will be addressed in Section 5.4.

5.3. Forecasting in the Latent Space

The starting point of this second step is a dataset consisting of time series of latent patches $(\mathbf{z}_p^{(t_i)})_{i=1}^T$, from which we aim to predict future latent patches $\mathbf{z}_p^{(t)}$, or, in other words, extrapolate the latent patch sequences. We refer to this second model that forecasts latent space sequences as the *forecasting model*. Given the structure the VQ-GAN enforces in the latent space, it should also be possible to decode the predicted new patches into meaningful CT images $\mathbf{v}_p^{(t)}$. However, the decoding step is not part of the forecasting model that will now be described, which solely operates on the latent space. The decoding step of the latent prediction is performed with the decoder model that was already trained for the image compression detailed in the previous section, and does not affect the training of the forecasting model.

5.3.1. Transformer

For the prediction of future latent images, we employ a transformer model. Unlike in VideoGPT [78], we do not predict the images autoregressively. Instead, we predict all latent pixels at the same time, which facilitates the inclusion of the irregularly sampled time points.
Like in any transformer model, we first require a matrix representation of our input. In our case, the input are timeseries of latent space images, which contain information on the values of the latent images as well as information on the time points at which the images were taken. The representation of values and time points in matrices is detailed in the paragraph **Triplet representation**. Following typical transformer models, the matrix representation is then embedded into a fixed-dimensional continuous space, aiming to produce dense, information-rich representation vectors. This step is explained in the paragraph **Embedding of triplet representations**. The resulting time series representations are then handed to a transformer model, for which the specific architecture and loss functions we choose are described in the last two paragraphs of this section.

Triplet representation. This paragraph provides a detailed description of the input and its representation that is handed to the transformer model. We start with a time series with $T+1$ latent representations of lung subvolumes, where each is associated with an acquisition date . The latent patches are denoted as $\mathbf{z}^{(t_i)}, i = 1..T+1$. For simplicity, we omit the patient identifier p from the patch identifier, meaning that $\mathbf{z}^{(t_i)}$ are considered to be patches coming from a time series of the same patient. Note that the time series with the length $T+1$ is a subsequence of the complete time series of the patient with length T_p. The subsampling of a sequence will be explained in Section 5.6.2. The patch associated with the most recent time point $T+1$ will serve as the target $\mathbf{y}$ during training,

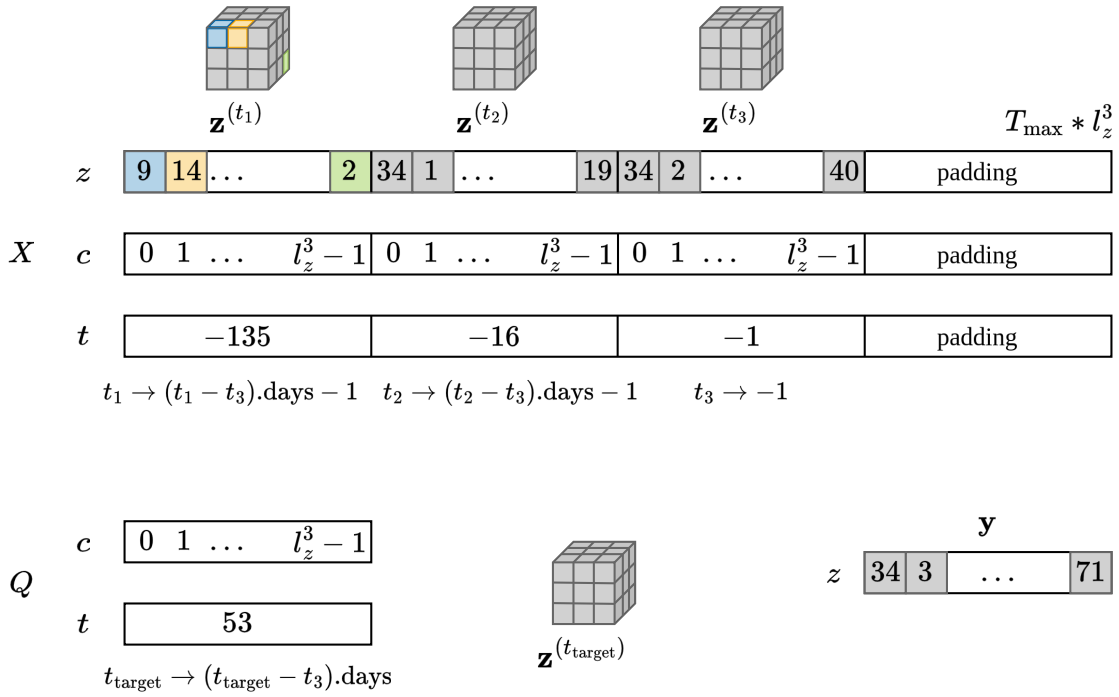


Figure 5.4.: **Triplet representation** In this example, the input consists of three latent space images taken on January 18, May 17, and June 1, all in 2019, and one target image taken on July 24, 2019. The input X to the transformer model consists of the flattened latent patches $\mathbf{z}$ of the three input images. The flattened sequence is padded up to the maximum length $T_{\max} * l_z^3$. The values z are paired with spatial positions c in the 3D volumes, which always simply count from 0 to $l_z^3 - 1$ to identify the position, and with the relative time differences in days to the last input (minus one for past images), in this case, the time difference to the third input date. The query Q consists of the channels and the target time, where the target $\mathbf{y}$ is composed of the flattened latent values of the target image (flattened according to the position order in the query).

while the remaining patches form the input time series of length T of the transformer. The dates in the input time series are converted into relative time points, where the most recent input image gets assigned $t = -1$, while each previous image gets assigned the negative time difference from the most recent input image in days minus one. Taking all relative time differences in the past minus one serves the purpose of not using zero for any real input, such that zero can later be used as a padding value. The relative target time point is the positive time difference between the target time and the most recent input patch time point for training. During inference, an arbitrary positive integer can be chosen.
The individual encoded images have the dimension $l_z \times l_z \times l_z$, as explained in the previous section. Each of the ${l_z}^3$ spatial positions in this patch is assigned a position index c between 0 and $l_z^3 - 1$.

The input time series is reformulated into a set of triplets, inspired by Yalavarthi et al. [74]. Each element z of the encoded image $\mathbf{z}$ is put together with its position c and its associated relative time point t to form a triplet $\mathbf{x} = (z, c, t)$. The whole time series can then be represented by the set of all triplets, which will be referred to as X.
When predicting a future target, the future image $\mathbf{y}$ is unknown, but the relative target time point t and all positions c of the target image are fixed. They are collected in the form of tuples $\mathbf{q} = (t, c)$ to form the query set Q and also serve as an input to the transformer model.
Since neural networks require fixed-sized matrices as input, the two sets X and Q have to be converted into this format. To do so, we define a maximal sequence length $T_{\max}$ for the network, i.e., the maximum number of input images that can be handed to the network. The triplet elements are concatenated into fixed length vectors (one for the values, one for the positions, and one for the times) of size $T_{\max} \cdot l_z^3$, which are padded at the end if fewer than $T_{\max}$ input images are present. Regarding the padding values, the values for the indices and positions are chosen as the maximal possible number plus one, so n_{cb} and l_z^3, respectively. The padding value for the time is more difficult to choose since there is no value that naturally cannot occur in a continuous time sequence. This is why we took all relative past time differences minus one, such that zero is never used as a time input. Thus, we take zero as the padding value. The target $\mathbf{y}$ is the flattened target image $\mathbf{z}$. An example of the final transformer input is shown in Figure 5.4.
To sum up, each patient's sequence of latent space patches $(\mathbf{z}_p^{(t_i)})_{i=1}^T$ is represented as a matrix X_p in which the codebook indices, the spatial positions, and the time points of the input sequence are collected. During training, one image is taken as the target, and thus not included in X_p. The target time and spatial positions of the target patch are collected in the query matrix Q_p.

Embedding of triplet representations. After handing a triplet representation X_p, that is, the representation of the time series of patient p, to the transformer, each triplet $\mathbf{x}$ is embedded by a learned network. As explained above, a triplet representation X_p is made up by triplets $\mathbf{x}$, each consisting of a single codebook index value z, its spatial position c in the 3D volume, and the time point associated to the latent patch in which it is located. We embed z, t and c separately. The values z and the positions c are embedded by a simple linear layer that projects the values to d_{model} dimensions. As the time dimension is continuous rather than discrete, we employ the Continuous Value Embedding proposed by Tipirneni et al. [73] to embed the relative time points t. As described in Section 3.1, this Continuous Value Embedding has a single input neuron for the one-dimensional time stamp, a hidden layer that projects the input to $\sqrt{d_{model}}$ dimensions, and an output layer with d_{model} dimensions. The authors use a $\tanh(\cdot)$ activation in the hidden layer. The embedding can be described by the following formula with weight matrices U and W and bias b, and a visualization is given in Figure 5.5.

$$Emb(t) = U \tanh(Wt + b) \tag{5.7}$$

The embedding networks are followed by dropout layers to avoid overfitting. The resulting embedded vectors for z, c, and t are added up to get a final embedding for each triplet $\mathbf{x}$. The queries $\mathbf{q}$ are embedded using the same embedding networks, but only the embeddings

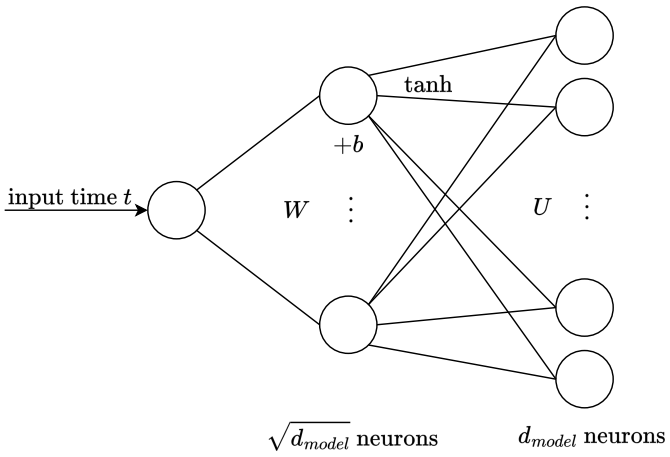


Figure 5.5.: Continuous Value Embedding architecture as proposed by Tipirneni et al. [73]

of t and c are added up, as z is not included in the query.
The result of the embedding step is a matrix of size $(T_{\max} \cdot {l_z}^3) \times d_{\text{model}}$ for the triplet representation matrix X_p, that is, each triplet is represented in one vector of size d_{model}, and a matrix of size ${l_z}^3 \times d_{\text{model}}$ for the query Q_p for each patient p.

Adapted transformer architecture. The embedded input is now handed to a transformer model that aims to predict the future latent patch. The transformer we use does not deviate much from the original transformer model presented by Vaswani et al. [16]. We will now explain the changes that we made to this architecture. Our implementation is largely based on the MONAI implementation[2] [97]. One of the changes in comparison to the original transformer is the application of the layer normalization before the attention blocks, as proposed by Xiong et al. [98].
The transformer encoder gets an embedded input sequence X_p as input and compresses the sequence into a compact representation. This representation is handed to the transformer decoder which combines this representation with the query $\mathbf{q}$ to make a prediction. In comparison to the original transformer, we drop the self-attention layer in the decoder, as the query is not a sequence, but is only used to inform the model of the target time.
As already mentioned, the transformer does not predict the future autoregressively. Instead, the final output $\hat{\mathbf{p}}$ of the transformer model has the dimensions $n_{cb} \times l_z^3$, which contains the estimated probability distribution over all codebook indices for every latent value of the target image. The final estimation $\hat{\mathbf{y}}$ of the target image is obtained by taking an argmax over each probability distribution. Optional dropout layers are added in the attention blocks. The adapted transformer architecture can be seen in Figure 5.6.

Loss of the forecasting model. While training the model, the output probability distributions for all positions should approach the real target indices. Therefore, we employ a multi-class CE loss on each position, which is defined in Section 2.1. A mean over all positions is taken to aggregate the resulting CE values. Overall, the loss can be

[2]Code on GitHub `https://github.com/Project-MONAI/GenerativeModels`

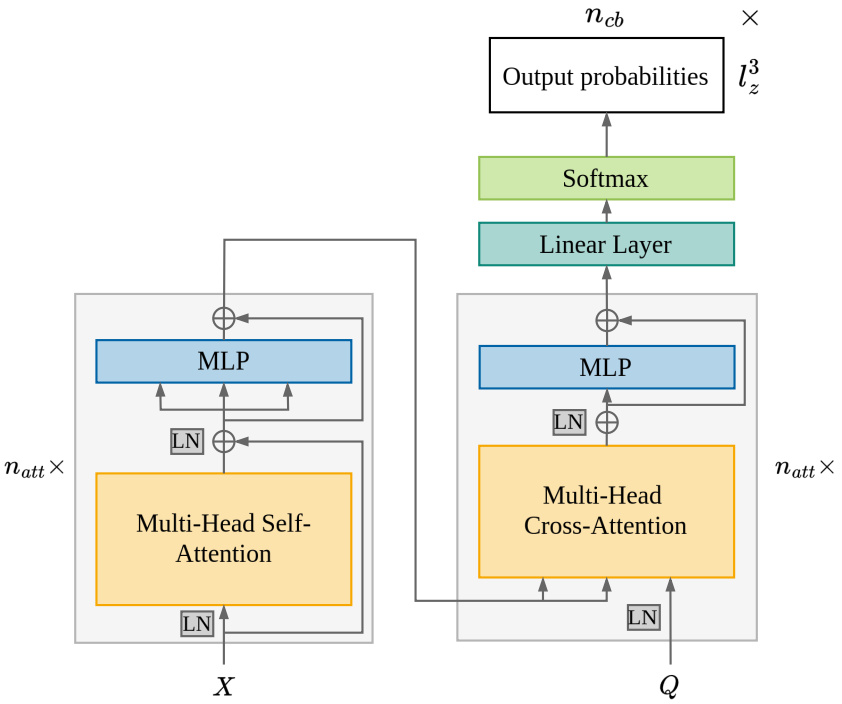


Figure 5.6.: Transformer architecture of the forecasting model.

formulated as shown in Equation 5.8.

$$\mathcal{L}_{\text{Forecasting CE}}(\hat{\mathbf{p}}, \mathbf{y}) = \frac{1}{l_z^3} \sum_{i=1}^{l_z^3} \left[- \sum_{j=0}^{n_{cb}-1} \mathbb{1}_{\{y_{[i]}=j\}} \log(\hat{p}_j^{(i)}) \right], \tag{5.8}$$

The matrix $\hat{\mathbf{p}} \in \mathbb{R}^{n_{cb} \times l_z^3}$ encodes the predicted class probabilities for each of the l_z^3 positions in the target. We denote the probability distribution corresponding at a specific spatial position as $\hat{\mathbf{p}}^{(i)}$, where $\hat{p}_j^{(i)}$ is the probability assigned to the codebook index j. The variable $y_{[i]}$ denotes the value of the target $\mathbf{y}$ at the spatial position i. The loss entails that the value of each position is classified independently of the others, as mentioned before.

The overall transformer model takes sequences of encoded subvolumes $(\mathbf{z}_p^{(t_i)})_{i=1}^T$ represented by the matrix X_p from which it aims to predict the future latent image $\hat{\mathbf{y}}$ at a specific time point embedded in the query matrix Q_p. It therefore estimates the probability distributions over all spatial positions i and codebook indices j, from which the final prediction is taken by choosing the indices with the maximum probability. More formally, we can write

$$\begin{aligned} &((\mathbf{z}_p^{(t_i)})_{i=1}^T, T+1) \xmapsto{\text{Triplet Representation}} (X_p, Q_p) \\ &F_{\text{Transformer}}(X_p, Q_p) \mapsto \hat{\mathbf{p}} \\ &\hat{\mathbf{z}}_p^{(T+1)} = \hat{\mathbf{y}} = \left(\hat{y}_{[i]}\right)_{i=1}^{l_z^3}, \text{where } \hat{y}_{[i]} = \arg\max_j \hat{p}_j^{(i)}, \end{aligned} \tag{5.9}$$

where $F_{\text{Transformer}}$ is the forecasting model. The prediction of the future latent image $\hat{\mathbf{y}}$ is, in the end, handed to the already trained VQ-GAN decoder to obtain the predicted future CT scan $\hat{\mathbf{v}}^{(T+1)}$.

5.3.2. Mamba

Recently, Selective State Space Models (SSMs) have emerged in the deep learning literature, which have shown to be able to outperform transformer models in some sequencing tasks, while using significantly lower computational resources. We therefore experiment with replacing the encoder part of the transformer model that compresses the long input sequences with a Mamba block [17]. The input to the Mamba block is the same as the one to the transformer encoder, that is, the same triplet representations X_p and Q_p are used that are embedded in the same way as in the transformer model. The Mamba block itself is implemented exactly as in the author's implementation[3] of their paper [17] and we use their default values for all hyperparameters. The Mamba block produces an output sequence of the same length as the input sequence. This output sequence is handed to a multi-head attention block, where it is used to calculate keys and values that are then, like in the transformer forecasting model described before, combined with the target times that form the queries. Like in the transformer forecasting model, a linear layer and softmax are applied at the end. The adapted architecture is depicted in Figure 5.7.

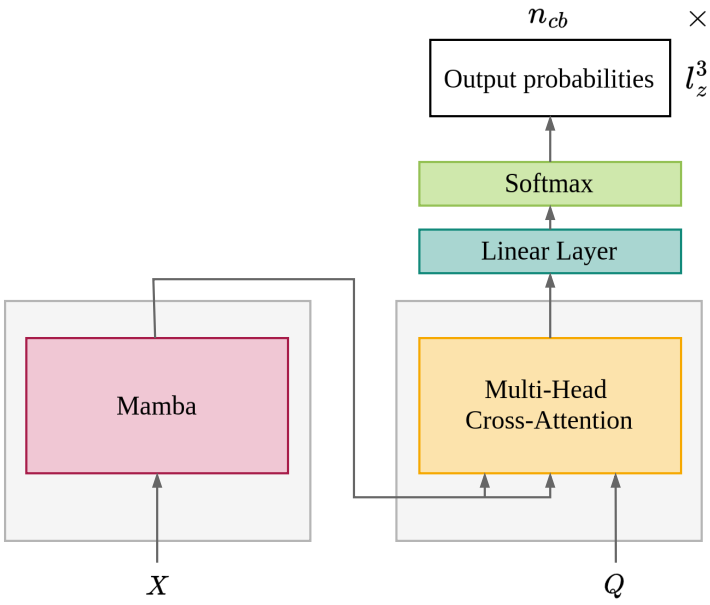


Figure 5.7.: Mamba replacing the transformer encoder in the forecasting model.

Summing up, the forecasting model including a Mamba block does not deviate much from the transformer model. Specifically, the transformer encoder self-attention block is replaced by the Mamba block. Compare Figures 5.6 and 5.7 for an overview. Like the transformer forecasting model, the Mamba forecasting model predicts probability distributions $\hat{\mathbf{p}}$, from

[3]Code on GitHub `https://github.com/state-spaces/mamba`

which the future latent patch $\mathbf{z}_p^{(T+1)}$ can be obtained, from a timeseries $(\mathbf{z}_p^{(t_i)})_{i=1}^T$ by using the same triplet respresentation and embedding:

$$F_{\text{Mamba}}(X_p, Q_p) \mapsto \mathbf{y} = \hat{\mathbf{p}} \tag{5.10}$$

The overall proposed model is capable of predicting future lung subvolumes $\mathbf{v}_p^{(T+1)}$ from a time series $(\mathbf{v}_p^{(t_i)})_{i=1}^T$. To do so, the VQ-GAN model is used to project all available subvolumes into latent space representations, resulting in latent time series $(\mathbf{z}_p^{(t_i)})_{i=1}^{T_p}$. The forecasting model (either a transformer or Mamba model) then takes a latent time series $(\mathbf{z}_p^{(t_i)})_{i=1}^T$ and predicts a future latent patch $\hat{\mathbf{z}}_p^{(T+1)}$. Finally, the VQ-GAN decoder is used to obtain the pixel space prediction $\hat{\mathbf{v}}_p^{(T+1)}$.

5.4. Towards a Meaningful Latent Space: Noisy Codebook Assignments in the VQ-GAN

As can be seen in the results (Section 6.3.3), one of the main challenges is to ensure continuity over time in the forecasting model. The goal is to be able to forecast lung subvolumes with dense temporal sampling, such that the predicted images display a smooth temporal progression, as opposed to exhibiting abrupt changes.
Since the codebook indices are arbitrarily assigned to the embedding vectors in the quantization step (refer back to Section 2.6.1) and the transformer only receives the indices as input, it does not receive any information about the structure and similarities in the VQ-GAN latent space. Note that the instances which we call *embedding vectors* or *codebook vectors*, refer to the codebook elements of the VQ-GAN (see Section 2.6.1) and are unrelated to the forecasting embedding explained in the transformer section. The forecasting model has to learn which codebook indices are usually similar and which are very different in pixel space. Such semantic similarity information could already be encoded in the latent vector space, which the forecasting model is unaware of.
To include semantic information also in the codebook indices, we employ an approach that was presented in the master's thesis of H. Coppock [99]. Instead of always assigning an encoded vector to the index of the closest embedding vector, the assignment is slightly randomized. That means discrete Gaussian noise centered around zero and with a standard deviation of σ is added to the codebook index of the closest embedding vector. The noise injection affects the exponential moving average updates, encouraging the data point to move closer to the noisily assigned vector, but is implemented such that it does not influence the commitment loss. That is, the embedding vectors are moved toward the points which were originally assigned to them before adding the noise. The result is, that the ordering of codebook indices has semantic meaning, where codebook indices that are close to each other are also similar in pixel space. H. Coppock demonstrates in his thesis that such a representation enables more meaningful linear interpolations in the quantized latent space.
The CE loss is not affected by the more meaningful ordering of the codebook indices, since it regards them as independent classes without any ordering. However, in the next sections, we explain strategies for using the semantic ordering of the codebook indices.

5.5. Exploring Different Loss Functions

We investigate different loss functions to improve prediction quality, which are detailed below. Most loss functions aim at improving temporal continuity in the predictions. An overview of all proposed loss functions is given in Figure 5.8.

Regression Loss on Latent Space Vectors

The proposed regression loss operates on codebook vectors instead of codebook indices, that is, it tries to move the codebook vectors at all spatial positions towards the target codebook vectors using a regression loss.

As mentioned in Section 5.4, one of the main challenges is achieving continuity in the time dimension. While in Section 5.4, we discussed a method that induces order to the codebook indices, the more intuitive idea would be to directly use the embedding vectors as input for the forecasting model instead of using the indices, since the vectors often have intrinsic semantic meaning. We therefore try to directly hand the latent patch $\mathbf{z}_q$, consisting of embedding vectors to the forecasting model, instead of the patch $\mathbf{z}$, which is made up of codebook indices. In the forecasting model's embedding explained above, triplet representations X of time series, consisting of codebook indices z, spatial positions c and time stamps t are passed through embedding layers to obtain a vector with dimension d_{model} for each triplet. Since we now work on codebook vectors instead of indices, the embedding layer for the values does not embed single indices, but linearly projects the embedding vectors to the dimension d_{model}. Again, note that the term *embedding vector* refers to instances of the VQ-GAN and not to instances of the embedding step of the forecasting model.
Since we also use embedding vectors instead of indices in the target, we cannot employ a CE loss anymore. Instead, we reformulate the task into a regression problem, where we try to minimize a distance, e.g., a MSE, between predicted embedding vector $\hat{\mathbf{e}}$ and real embedding vector $\mathbf{e}$. The loss function with a MSE is given in the following formula, where $\mathbf{y}_q \in \mathbb{R}^{l_z^3 \times d_{cb}}$ is the target image with quantized codebook vectors of dimension d_{cb}, and $\hat{\mathbf{y}}_q \in \mathbb{R}^{l_z^3 \times d_{cb}}$ is the prediction. The variable $\hat{\mathbf{e}}_i = (\hat{\mathbf{y}}_q)_{[i,\cdot]}$ denotes the predicted embedding vector at the spatial position i, and likewise $\mathbf{e}_i = (\mathbf{y}_q)_{[i,\cdot]}$ the target embedding vector at position i.

$$\mathcal{L}_{\text{Regression}}(\hat{\mathbf{y}}_q, \mathbf{y}_q) = \frac{1}{l_z^3} \sum_{i=1}^{l_z^3} ||\hat{\mathbf{e}}_i - \mathbf{e}_i||_2^2 \tag{5.11}$$

Since the target contains codebook vectors instead of indices, the forecasting output does not have the dimension $n_{cb} \times l_z^3$, but instead $d_{cb} \times l_z^3$, which is realized by altering the last linear layer of the forecasting model accordingly.
In order to meaningfully decode the resulting regressed embedding vectors, they each have to be one of the available embedding vectors, since the VQ-GAN only learned to reconstruct those during training. Therefore, before decoding, but after calculating the forecasting loss, the predicted vectors are quantized by mapping them to the closest embedding vector.

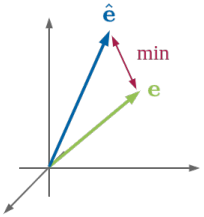
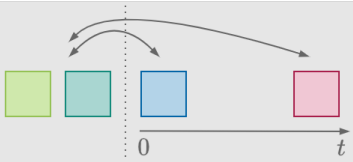
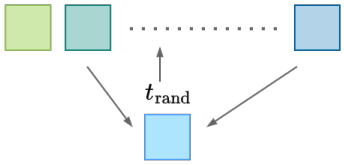
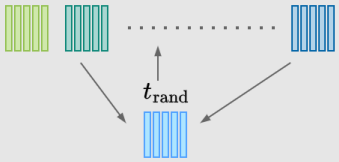
	Operates on	Requires NCB
CE Loss minimizes the CE between the predicted probability distribution and target patch constisting of codebook indices. $\mathcal{L}_{\text{Forecasting CE}}(\hat{\mathbf{p}}, \mathbf{y}) = \frac{1}{l_z^3} \sum_{i=1}^{l_z^3} \left[- \sum_{j=0}^{n_{cb}-1} \mathbb{1}_{\{y_{[i]}=j\}} \log(\hat{p}_j^{(i)}) \right]$	Codebook indices	No
Regression Loss minimizes the distance between estimated and target codebook vectors. $\mathcal{L}_{\text{Regression}}(\hat{\mathbf{y}}_q, \mathbf{y}_q) = \frac{1}{l_z^3} \sum_{i=1}^{l_z^3} \|\hat{\mathbf{e}}_i - \mathbf{e}_i\|_2^2$ 	Codebook vectors	No
Regularized CE loss punishes large deviations from the last input based on the time distance between the last input and the target.  $\mathcal{L}_{\text{Regularized CE}}(\hat{\mathbf{p}}, \mathbf{y}, \mathbf{z}^{(T)}, \Delta_{\text{time}}) = \mathcal{L}_{\text{Forecasting CE}}(\hat{\mathbf{p}}, \mathbf{y}) + \mathcal{R}(\hat{\mathbf{p}}, \mathbf{z}^{(T)}, \Delta_{\text{time}})$	Codebook indices	Yes
Weighted CE loss allows for training samples between available images by constructing a target by interpolating the codebook indices of the neighboring images.  $\mathcal{L}_{\text{Weighted CE}}(\hat{\mathbf{p}}^{(t_{\text{rand}})}, \mathbf{z}^{(t_k)}, \mathbf{z}^{(t_{k+1})})$ $= \beta \cdot \mathcal{L}_{\text{Forecasting CE}}(\hat{\mathbf{p}}^{(t_{\text{rand}})}, \mathbf{z}^{(t_{k+1})}) + (1-\beta) \cdot \mathcal{L}_{\text{Forecasting CE}}(\hat{\mathbf{p}}^{(t_{\text{rand}})}, \mathbf{z}^{(t_k)})$	Codebook indices	Yes
Weighted regression loss allows for training samples between available images by constructing a target by interpolating the codebook vectors of the neighboring images.  $\mathcal{L}_{\text{Weighted Regression}}(\hat{\mathbf{z}}_q^{(t_{\text{rand}})}, \mathbf{z}_q^{(t_k)}, \mathbf{z}_q^{(t_{k+1})})$ $= \beta \cdot \mathcal{L}_{\text{Regression}}(\hat{\mathbf{z}}_q^{(t_{\text{rand}})}, \mathbf{z}_q^{(t_{k+1})}) + (1-\beta) \cdot \mathcal{L}_{\text{Regression}}(\hat{\mathbf{z}}_q^{(t_{\text{rand}})}, \mathbf{z}_q^{(t_k)})$	Codebook vectors	Yes
Pixel loss minimizes the dissimilarity between target and prediction in the pixel space. $\mathcal{L}_{\text{Pixel}}(\hat{\mathbf{y}}, \mathbf{v}_{\text{target}}) = \mathcal{L}_{\text{pixel}}(\mathcal{D}(\hat{\mathbf{y}}), \mathbf{v}_{\text{target}})$ 	Pixels	No

Figure 5.8.: Overview of all proposed loss functions.

Regularized Cross Entropy Loss

The goal of the regularized CE loss is to penalize large deviations between the predicted output and the most recent input image, particularly when the prediction time point is temporally proximate to that of the last input.

Instead of working on embedding vectors, we now return to a CE loss on the codebook indices. As discussed in Section 5.4, it might be beneficial to introduce a loss function that encourages continuity of the predictions over time. That is, there should be no sudden changes in the pixel space predictions when predicting multiple images closely sampled in time. To do so, the idea is to penalize large deviations from the image of the previously available time step, if the time distance to this last time step is small. For example, if the prediction date is only one day away from the last available input, the prediction should be very similar to this last input, while if a year has passed, the prediction can deviate more from the last input.
To access the property of closeness in pixel space also in the latent space, we employ the VQ-GAN with noisy codebook assignments, as explained in Section 5.4. We can then exploit the induced pixel similarity of neighboring indices in a loss function. In particular, we formulate the following regularization term that will be added to the CE loss.

$$\mathcal{R}(\hat{\mathbf{p}}, \mathbf{z}^{(T)}, \Delta_{\text{time}}) = w(\Delta_{\text{time}}) \cdot d(\hat{\mathbf{p}}, \mathbf{z}^{(T)}) \tag{5.12}$$

The function $w(\Delta_{\text{time}})$ returns a scalar to weight the dissimilarity between the predicted probability distribution and the last available input image $\mathbf{z}^{(T)}$ based on their time difference. We define $w(\Delta_{\text{time}})$ as

$$w(\Delta_{\text{time}}) = \exp(-\alpha \Delta_{\text{time}}), \tag{5.13}$$

where α is a hyperparameter which controls how high the penalty is. The smaller α, the higher the penalty and the more the model is encouraged not to deviate from the previous input.
The dissimilarity between the predicted probability distributions and the last input image is measured by $d(\hat{\mathbf{p}}, \mathbf{z}^{(T)})$, where $\hat{\mathbf{p}} \in \mathbb{R}^{n_{cb} \times l_z^3}$ contains probability distributions for all l_z^3 positions in the prediction obtained by applying a softmax to the output of the forecasting model. The probability distribution of a specific position is denoted by $\hat{\mathbf{p}}^{(i)}$. We define the dissimilarity between the prediction and last input $d(\hat{\mathbf{p}}, \mathbf{z}^{(T)})$ as follows:

$$d(\hat{\mathbf{p}}, \mathbf{z}^{(T)}) = \frac{1}{l_z^3} \sum_{i=1}^{l_z^3} \left[\frac{1}{n_{cb}} \sum_{j=0}^{n_{cb}-1} \hat{p}_j^{(i)} \cdot d_{index}(j, z_i^{(T)}) \right] \tag{5.14}$$

In the inner sum over all codebook indices, a dissimilarity measure $d_{index}(\cdot, \cdot)$ between each index j and the index of the last available image $z_i^{(T)}$ is multiplied by the estimated probability $\hat{p}_j^{(i)}$ for this index j. The product results in high values for probabilities that are high for indices that are far away from the 'real' index $z_i^{(T)}$ (of the past image, j^* from now on), while low probabilities and probabilities that are close to the index j^* produce small values. This behavior is the aim of the regularization term — to penalize indices that deviate much from the past image. The outer sum is part of a mean over all codebook indices in the prediction and last image.

The dissimilarity measure over the indices $d_{index}(j, j^*)$ can be chosen in several ways. We came up with three different options: A MSE and an L1 loss, and a dissimilarity measure based on a Gaussian distribution around the true index. For the latter, we center a Gaussian distribution around the 'real' index j^*, with a standard deviation σ, and take one minus this distribution to arrive at a dissimilarity score that is low around the 'real' index and equally high everywhere else. Refer to Figure 5.9 for a visualization of the different options. For experiments in Chapter 6 we always employ a MSE as dissimilarity measure.

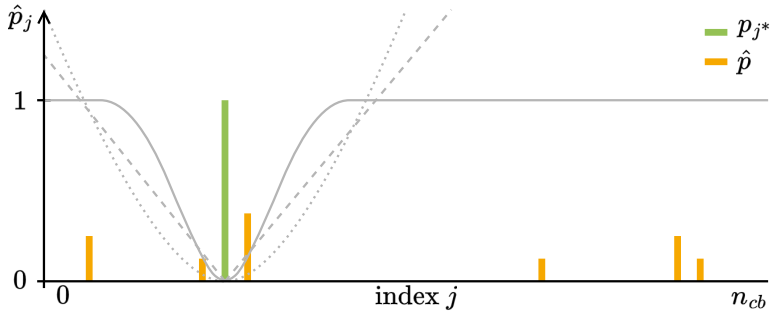


Figure 5.9.: **Example for the calculation of the penalty for one index in the previous input and the corresponding predicted probability distribution.** The green bar marks an example index of the last available input image. The orange bars show the probability distribution of the prediction index, resulting from the forecasting model output. The gray lines show the different options for the dissimilarity measure between indices: an L1 loss, a MSE loss and the explained measure constructed using a Gaussian. The orange bars next to the green one produce low penalties, as the index dissimilarity is low, while the bars farther away are multiplied by higher values, resulting in higher penalties.

To sum up, we end up with the final loss function

$$\mathcal{L}_{\text{Regularized CE}}(\hat{\mathbf{p}}, \mathbf{y}, \mathbf{z}^{(T)}, \Delta_{\text{time}}) = \mathcal{L}_{\text{Forecasting CE}}(\hat{\mathbf{p}}, \mathbf{y}) + \mathcal{R}(\hat{\mathbf{p}}, \mathbf{z}^{(T)}, \Delta_{\text{time}}), \tag{5.15}$$

where $\mathcal{L}_{\text{Forecasting CE}}$ is the original CE loss defined in Equation 5.8 and $\mathcal{R}$ is the regularization term defined above that penalizes large deviations from the past image.

Weighted Cross Entropy Loss for Interpolation

The weighted CE loss allows for using interpolation samples in between available patches by implicitely interpolating codebook indices of the neighboring patches.

As will be explained in Section 5.6.2, we experiment with pre-training the model with an interpolation task. Concretely, an image is removed from the input time series and used as target. For details, please refer to Section 5.6.2. To additionally be able to create interpolation samples at time points at which no image is available, we introduce a weighted CE loss. Instead of removing an image from the input time series, we randomly

sample a time point between the available images and try to predict the lung patch at this time point. However, during training, a target or another way to evaluate the predicted image is required. We choose the linear interpolation between the codebook indices of latent patches at the two neighboring time points as the target. Note that the linear interpolation between latent patches is only meaningful if codebook indices that are close to each other produce similar outputs in pixel space. Thus, this loss is only suitable when using the VQ-GAN with noisy codebook assignment. Instead of explicitly creating the linearly interpolated patch, the resulting image is evaluated by a weighted CE loss defined by the formula below.

$$\begin{aligned}\mathcal{L}_{\text{Weighted CE}}(\hat{\mathbf{p}}^{(t_{\text{rand}})}, \mathbf{z}^{(t_k)}, \mathbf{z}^{(t_{k+1})}) = \beta \cdot \mathcal{L}_{\text{Forecasting CE}}(\hat{\mathbf{p}}^{(t_{\text{rand}})}, \mathbf{z}^{(t_{k+1})}) \\ + (1-\beta) \cdot \mathcal{L}_{\text{Forecasting CE}}(\hat{\mathbf{p}}^{(t_{\text{rand}})}, \mathbf{z}^{(t_k)})\end{aligned} \tag{5.16}$$

The random target time is denoted by t_{rand}, where the previously available time point is denoted by t_k and the time point after by t_{k+1}. The weight β is defined as $\beta = \frac{t_{\text{rand}} - t_k}{t_{k+1} - t_k}$, that means, as the distance from the random time point to the previous time point divided by the total distance between the time points. Thus, the weighted CE loss replaces the linear interpolation between the images at time points t_k and t_{k+1} by an implicit interpolation in the loss function. The weighted CE loss is visualized in Figure 5.10.

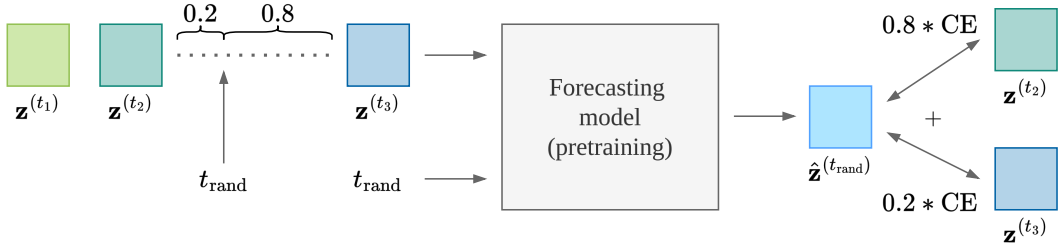


Figure 5.10.: **Example of a random time point between available images and the weighted CE loss.** A time series is handed to the forecasting model. For the query time point, a time point t_{rand} is randomly drawn between the available ones, in this case, between t_2 and t_3. The weighting parameter β is 0.2, so the weighted CE loss on the prediction is calculated by $0.2 * \mathcal{L}_{\text{Forecasting CE}}(\hat{\mathbf{p}}^{(t_{\text{rand}})}, \mathbf{z}^{(t_3)}) + 0.8 * \mathcal{L}_{\text{Forecasting CE}}(\hat{\mathbf{p}}^{(t_{\text{rand}})}, \mathbf{z}^{(t_2)})$.

It remains to define the sampling strategy for the time point t_{rand}. We employ two different sampling strategies. In both strategies, first, an available time point t_k is randomly selected.

- **Uniform sampling**: In the first strategy, a random time point is then sampled between t_k and the following time point t_{k+1} uniformly at random. The disadvantage of this approach is that the farther away the random point in time is from the available images, the less certain we can be that the interpolated image is still meaningful.
- **Sampling close to available images**: Therefore, our second strategy is to sample images close to the available ones with a higher probability. To do so, there are two

normal distributions centered around t_k and t_{k+1}, from one of which (with a 50-50 chance) a sample is taken. Samples outside the range $[t_k; t_{k+1}]$ are discarded, and new samples are drawn until one falls into this range.

Weighted Regression Loss

The weighted regression loss allows for using interpolation samples in between available patches by implicitly interpolating codebook vectors of the neighboring patches.

Instead of implicitly interpolating the codebook index space with the loss function, one can also interpolate in the embedding vector space, i.e., interpolate between the quantized embedding vectors that are not mapped to indices. The weighted CE loss on the indices is therefore replaced by a weighted regression loss on the embedding vectors. The regression loss on embedding vectors has already been explained in Section 5.5. Like in the weighted CE loss, a time point t_{rand} between two existing images is randomly sampled. To evaluate the prediction image at this time point, a loss similar to the weighted CE loss defined in Equation 5.16 is employed as defined below:

$$\begin{aligned}\mathcal{L}_{\text{Weighted Regression}}(\hat{\mathbf{z}}_q^{(t_{\text{rand}})}, \mathbf{z}_q^{(t_k)}, \mathbf{z}_q^{(t_{k+1})}) = \beta \cdot \mathcal{L}_{\text{Regression}}(\hat{\mathbf{z}}_q^{(t_{\text{rand}})}, \mathbf{z}_q^{(t_{k+1})}) \\ + (1-\beta) \cdot \mathcal{L}_{\text{Regression}}(\hat{\mathbf{z}}_q^{(t_{\text{rand}})}, \mathbf{z}_q^{(t_k)})\end{aligned} \tag{5.17}$$

The weight β is again defined as $\beta = \frac{t_{\text{rand}} - t_k}{t_{k+1} - t_k}$, the regression loss $\mathcal{L}_{\text{Regression}}$ is the one defined in Equation 5.11, and ${\mathbf{z}_q}^{(t)}$ are latent space images at time point t consisting of quantized embedding vectors. Note that, like in the simple regression loss in Section 5.5, the regressed predictions are quantized before decoding them back to pixel space, but this operation does not affect model training. We employ the same sampling strategies for the time point t_{rand} as in the weighted CE loss (Section 5.5).

Loss in Pixel Space

An approach to improve the prediction results on the pixel level is to calculate the loss on the decoded image in the pixel space. The idea is simple at first glance: an error, like MSE or SSIM (explained in Section 6.1) is calculated on the decoded prediction and the target image and then propagated back through the frozen decoder and the forecasting model. However, this approach comes with a differentiability problem. The translation of the predicted probability distribution into a volume of codebook indices that can be decoded is not differentiable, since this is usually done by an argmax. To circumvent this, we employ a Gumbel-softmax straight-through estimator as proposed by Jang et al. [100]. This estimator uses the discrete argmax operation in the forward pass, to ensure that the samples can be handed to the VQ-GAN decoder, but it approximates the gradient using continuous Gumbel-softmax samples during the backward pass. These samples can be formulated as

$$\hat{y}_j = \frac{\exp(\log(\hat{p}_j) + g_j)}{\sum_{k=1}^{n_{cb}} \exp(\log(\hat{p}_k) + g_k)}, \tag{5.18}$$

where $\hat{y}_j$ is the relaxed probability for codebook index j at a fixed spatial position i of the prediction patch $\hat{\mathbf{y}}$. That is, we have a one-hot encoded vector $\text{h}(y_{[i]})$ of a codebook index from the target patch $\mathbf{y}$ at the spatial position i, where $\text{h}(k) = (\mathbb{1}_{\{k=j\}})_{j=0}^{n_{cb}}$ converts

a codebook index in a one-hot encoded vector, which we try to approximate using the Gumbel-softmax estimate, where $\hat{y}_j$ is the estimated value for the j^{th} position of $\text{h}(y_{[i]})$. The variables $\hat{p}_j$ are the class probabilities of each codebook element j and g_j are samples from the standard Gumbel distribution. [100]
This estimation trick gives us the possibility to directly apply the loss on the decoded images in pixel space, as formulated below, where $\mathcal{L}_{\text{P}}$ is any suitable distance function for the pixel space.

$$\mathcal{L}_{\text{Pixel}}(\hat{\mathbf{y}}, \mathbf{v}_{\text{target}}) = \mathcal{L}_{\text{P}}(\mathcal{D}(\hat{\mathbf{y}}), \mathbf{v}_{\text{target}}) \tag{5.19}$$

5.6. Training

After having discussed model architectures and loss functions, this section details the training procedure of the proposed model, with a focus on which and how the training data is used.
The proposed two-step forecasting model is trained in two phases: First, the VQ-GAN is trained to reconstruct lung patches. Afterward, the VQ-GAN weights are frozen, so not changed in the remaining training procedure. The forecasting model is then trained with time series data encoded to the fixed latent space.

5.6.1. VQ-GAN

The VQ-GAN gets lung subvolumes as input which it tries to reconstruct while also trying to fool the patch discriminator into labeling them as real images. For this, as described in Section 4.2, lung subvolumes are extracted from all available lung CT images regardless of the kernel or the time series. In particular, for the VQ-GAN training, the subvolumes are extracted randomly from all available CT scans, collected in *Dataset A*.
Hyperparameters of the VQ-GAN training are

- the patch size l_v, that means how many pixels (= mm) each side of the extracted lung cube has,
- the number of random patches that we draw in each epoch,
- the number of training epochs,
- the batch size, and
- the learning rates for the generator and discriminator.

The final VQ-GAN model is chosen based on the minimum value of the validation loss that is calculated after each training epoch.

Fine-tuning on Nodules

Because we are primarily interested in the development of cancerous lung tissue, it is crucial that the VQ-GAN can not only reconstruct healthy subvolumes but also subvolumes containing anomalies. As cancerous regions usually only make up a small ratio of the lung, randomly drawn subvolumes contain mostly healthy tissue. To ensure that the VQ-GAN also reconstructs unhealthy tissue well, we investigate adding fine-tuning epochs after

the training explained above, where the model is fed with subvolumes positioned around potential lung nodules as described in Section 4.2.1. In particular, we use all subvolumes containing nodules from the training set of *Dataset B*, as we put only those through the segmentor to receive nodule masks. The number of fine-tuning epochs is an additional hyperparameter of the model.

5.6.2. Forecasting Model

The forecasting model training requires a time series of lung subvolumes, where the latest subvolume is used as the target. Like for the VQ-GAN, the easiest choice for extracting lung subvolumes is the extraction at random lung positions, where the same subvolume is extracted from each of the registered lung CT scans of one patient.

Training on Nodules

Because most of the lung does not change over time even in lung cancer patients, sampling random patches introduces a considerable bias towards subvolumes that do not change at all or only very little over time. As the aim is to model the development of lung nodules, the forecasting model needs to focus on nodules specifically to learn whether a nodule remains the same, grows, or shrinks. Therefore, for the forecasting model, we extract subvolumes that potentially contain nodules by employing a nodule segmentor as described in Section 4.2.1.

Subsampling Long Time Series

To enrich training data, specifically the timeseries data in *Dataset B*, and to deal with time series that exceed the maximal window size of the forecasting model input, we implement a subsampling strategy. Therefore, we first define how many samples we intend to create for each time series. We denote this number of samples by s and the length of a time series by T_p. There are three different cases on how to generate samples:

- If the number of samples s is equal to one, we simply take the whole time series and cut off all images that exceed the maximal input size T_{max} of the forecasting model, keeping only the T_{max} most recent scans.

Figure 5.11.: **Subsampling Long Time Series: Example for number of subsamples $s = 1$, time series length $T_p = 6$ and maximal input length $T_{\text{max}} = 5$.** If only one subsample is taken from a time series and the the time series length exceeds the maximal input length, the first images are discarded. In this case, the first image is discarded, since only five images fit in the forecasting window.

- If the number of possible combinations of length ≥ 3 is smaller than or equal to the number of samples s and $T \leq T_{\text{max}}$, all these combinations are included into the training data.

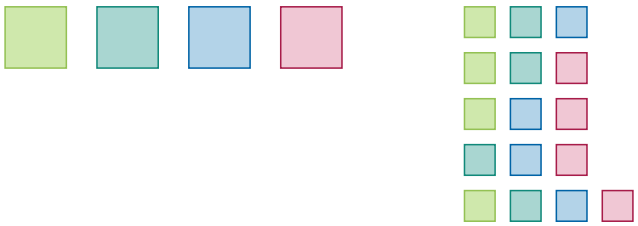


Figure 5.12.: **Subsampling Long Time Series: Example for number of subsamples $s = 5$, time series length $T_p = 4$ and maximal input size $T_{\text{max}} = 5$.** If there are fewer or equal possible combinations of the sequence elements than samples that are meant to be drawn, all combinations are taken. In this case, the five possible combinations are all included in the training data.

- If the number of samples s is smaller than the number of possible combinations or $T_p > T_{\text{max}}$, we draw s random samples as follows. First, we draw a random sequence length T_{rand} between 3 and T_{max}. Then we draw T_{rand} scans from the available time series. We repeat this procedure s times to generate s random training samples from the time series.

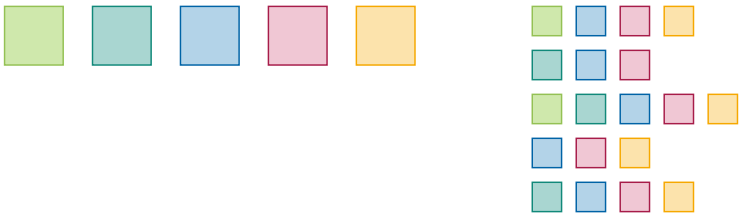


Figure 5.13.: **Subsampling Long Time Series: Example for number of subsamples $s = 5$, time series length $T_p = 5$ and maximal input size $T_{\text{max}} = 5$.** If the number of combinations of sequence elements is larger than the number of subsamples to be drawn or $T_p > T_{\text{max}}$, we draw random sequences. The sequence lengths and images are randomly sampled to generate $s = 5$ training samples, for example, as shown in the figure.

Subsampling training sequences enrich the training dataset, but s has to be chosen with care, as a large s can lead to overfitting when the model sees a similar training sample too often and memorizes it.

Training Data Augmentations

Another common approach to enriching training data is augmentation. Augmentation describes the process of altering existing data to generate new samples. For our image sequences, we employ two different augmentations: rotating and flipping the 3D images in pixel space. For rotation, two dimensions are drawn randomly, around which the lung subvolumes are rotated by 90 degrees. For flipping, we draw a random axis on which to flip the subvolumes. To create a new training sample, the same augmentation needs to be

applied to all images of a time series. The number of randomly generated rotated and randomly flipped samples per time series is determined by hyperparameters.

Interpolation Pre-training

Extrapolating time series of CT scans is very challenging, due to several reasons, like complicated tumor growth patterns, high dimensionality and heterogeneity of the training data. These challenges impede the training process and the models tend to overfit quickly. Therefore, we experiment with pre-training the models on a slightly easier but closely related task: time series interpolation. Before training the model on the forecasting task, the model receives time series as input where one lung patch is missing in between and tries to reconstruct the missing patch. The patch that is removed from the time series and used as target is sampled randomly in each time series. This process is repeated for a fixed number of epochs. Refer to Figure 5.14 for a visual explanation of the interpolation training.
After interpolation training, the model is trained to forecast images as described earlier. For the weighted loss functions (Section 5.5 and 5.5), we additionally have to define how

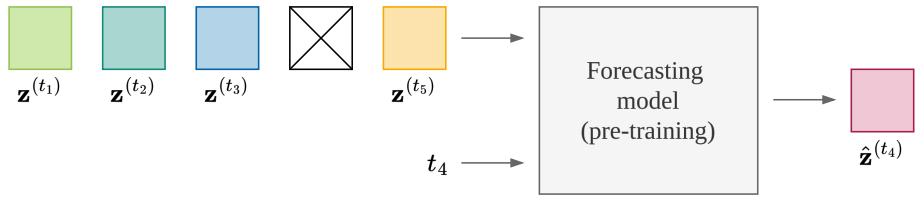


Figure 5.14.: **Interpolation pre-training.** The model tries to predict a missing time point in the time series.

many time points t_{rand} between existing images are drawn for each training sample.

Overall, the training of the forecasting model has the following important hyperparameters, apart from hyperparameters determining the model architecture:

- **General hyperparameters** are the number of training epochs, the batch size, and the learning rate.
- A **subsampling hyperparameter** is the number of subsamples s of a time series.
- An **augmentation hyperparameter** is the number of flipped and/or rotated images that should be created for each time series.
- **Pre-training hyperparameters** are whether the model is pre-trained with the interpolation task, the number of pre-training epochs, and the number of interpolation samples per time series.

6. Experiments

This section details the experiments that were conducted to evaluate the model and the different methodological options described above. First, experiments on the VQ-GAN are explained. Second, experiments on the forecasting model only, which means experiments on the latent space, are discussed. The third section then describes experiments conducted on the combined model, where evaluations are performed on pixel space predictions. The last section contains a technical experiment on computational resources.
Each experiment is structured into a research question that it aims to answer, the quality measures to evaluate the results, the data that was input into the model, and the relevant hyperparameters of the model. The results directly follow the descriptions of the experiments.

6.1. Experiment 1: CT Image Reconstruction using a VQ-GAN

The first experiment aims to evaluate the compression and reconstruction accuracy of the VQ-GAN. We investigate whether it is suitable for the latent representation of lung CT images that can be used in the forecasting model.

Question. Can the VQ-GAN compress the CT image subvolumes into low-dimensional quantized patches and still reconstruct them well?

Quality Measures. As a quantitative measure, the mean reconstruction loss (L1 loss), the mean Peak Signal-to-Noise Ratio (PSNR) value, and the mean Structural Similarity Index Measure (SSIM) value on all validation images are used to assess the quality of the reconstructed images. The PSNR is defined as

$$\mathrm{PSNR}(\mathbf{a}, \mathbf{b}) = 10 \log_{10} \frac{R^2}{\mathrm{MSE}(\mathbf{a}, \mathbf{b})}, \tag{6.1}$$

where $\mathbf{a}$ and $\mathbf{b}$ are two images and R is the maximum possible range of the image values, in our case 4,095. This number results from the minimum intensity value of $-1,024$ and the maximum intensity value of $3,071$ present in our data. The PSNR can range between 0 and ∞ in units of decibels (dB), where a higher value indicates a better reconstruction. In practice, values between 40 and 50dB indicate high-quality image reconstructions. The Structural Similarity Index Measure (SSIM) as developed by Wang et al. [101] aims to measure perceived errors by exploring structural information in the image. It can range between -1 and 1, where 1 corresponds to perfect similarity, 0 for no similarity at all, and -1 for perfect anti-correlation.
For a qualitative evaluation, we randomly show reconstructions of subvolumes from the validation set.

Data. For training the VQ-GAN model, *Dataset A* as described in Section 4.2 is used, in particular the training split consisting of 14,499 CT images. The corresponding validation split is used for model selection and also for the shown evaluations. The test set is not used in this thesis as it is meant to be used for unbiased evaluations in further research. From each CT image, 128 subvolumes of size $32 \times 32 \times 32$ pixels with at least 50% lung volume included are randomly drawn in each training and validation epoch.

Hyperparameters. For this experiment, we use the VQ-GAN model as defined in Section 5.2.1. We choose its hyperparameters as listed below:

- **Training**: The model is trained for five epochs, and the model state yielding the lowest validation loss is retained as the final model. The batch size is 256, and the generator and discriminator learning rates are 10^{-4} and $2 \cdot 10^{-4}$ respectively, and the adversarial weight $\lambda_{\text{GAN}} = 10$.
- **Architecture**: The encoder consists of $n_l = 3$ downsampling convolutions, each cutting the size of the input in half, and followed by $n_{rl} = 2$ residual units. The downsampling convolutions each have 256 channels (kernels) of size four. As explained earlier, a final convolution reduces the number of kernels to the defined embedding dimension d_{cb}. The decoder is chosen symmetric to the encoder. Refer back to Figure 5.3 for a visualization of the architecture.
- **Embedding space**: The number of embedding vectors is chosen to be $n_{cb} = 256$ with dimension $d_{cb} = 32$.

The chosen hyperparameters result in reducing the original subvolumes of side length $l_v = 32$ to the patch size $l_z = 4$.

Results. The mean L1 loss measured on the reconstructions of subvolumes from the validation set of *Dataset A* is **62.74**, the mean PSNR **33.36**, and the mean SSIM **0.806**. As a reference, a PSNR above 30dB usually indicates reasonable, but not high-quality reconstructions. The value of the SSIM also indicates this.
The shown random reconstruction examples from the validation dataset visualized in Figure 6.1 are in line with this assumption: the overall reconstructions look reasonable, but some of the details are lost, e.g., the white line in the bottom center of the first image in the left that is missing in the reconstruction, or the small white details in the image in the third column that are not visible in the reconstructed image.
Figure 6.2 shows some additional reconstruction examples, this time drawn from subvolumes that contain nodules which are part of the validation set of *Dataset B*. It can be seen that, even for unhealthy tissue, the reconstructions preserve the overall structure, although not all details can be reconstructed. In the most left image in Figure 6.2, for example, the light circle in the upper left corner is reconstructed at a slightly different location and with lower intensity.

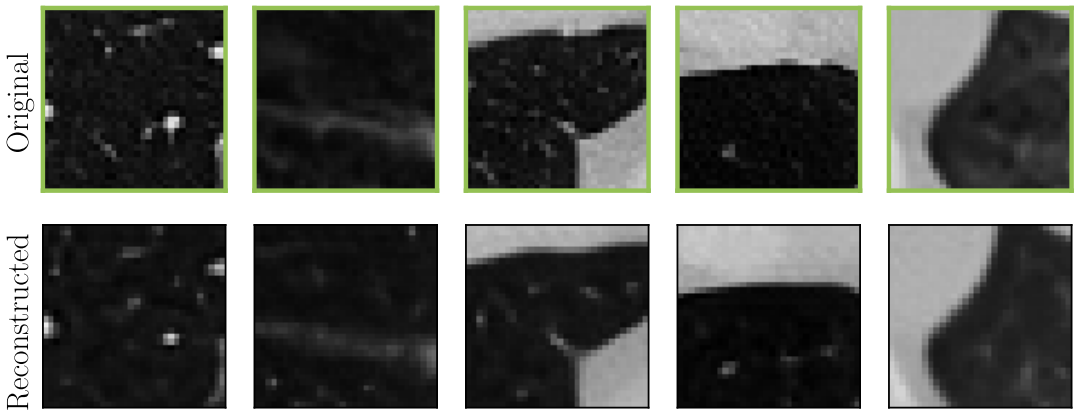


Figure 6.1.: **VQ-GAN reconstruction examples.** The original CT images are shown at the top, together with the corresponding reconstructions underneath. Samples were randomly drawn from *Dataset A*. All visualizations show the central slice of the sagittal plane.

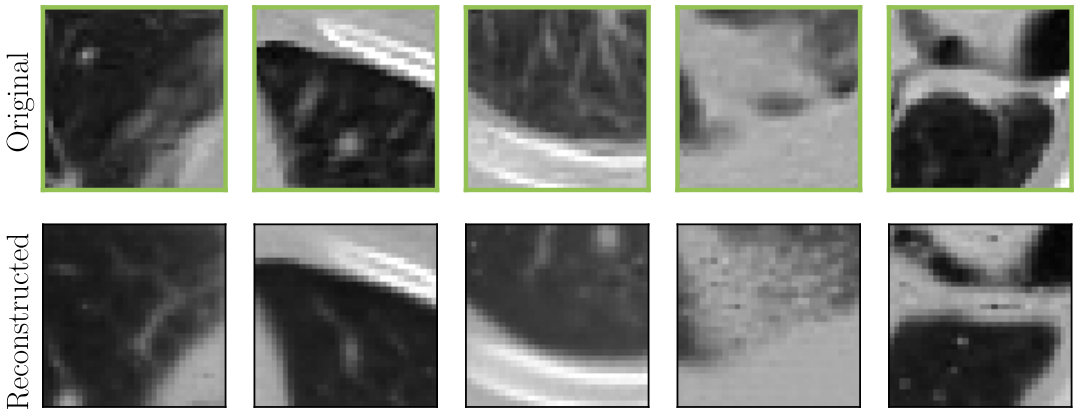


Figure 6.2.: **VQ-GAN reconstruction examples.** Samples were randomly drawn from subvolumes containing nodules from the validation set of *Dataset B*.

6.1.1. Experiment 1.1: Optimizing Adversarial-Reconstruction Balance in VQ-GANs

The embedding of the VQ-VAE in the adversarial framework is meant to improve image quality, i.e., avoid blurry images caused by the reconstruction loss [15]. In practice, it is important to find a balance between low reconstruction loss and low generator and discriminator loss. This balance can be adjusted by the hyperparameter λ_{GAN}, as used in the equations 5.5 and 5.6. An imbalance between the discriminator and the generator can cause unstable training, while, on the other hand, a weak adversarial part does not contribute to improving image quality. We therefore evaluate which values for λ_{GAN}

achieve a good balance between reconstruction accuracy and realistic-looking, sharp results.

Question. Can the integration of the VQ-VAE into an adversarial framework improve results compared to a VQ-VAE which is not trained with an additional discriminator? How can the adversarial weight λ_{GAN} be chosen to optimize reconstruction quality?

Quality Measures. To assess reconstruction quality, we inspect the reconstruction (L1) loss, the SSIM and the PSNR values for different hyperparameter choices. In addition, we visually investigate the generator and discriminator loss during training. For a qualitative evaluation, random images from the validation set are reconstructed by different models and manually analyzed.

Data. We use the same data as described in Experiment 6.1.

Hyperparameters. All hyperparameters of the VQ-GAN are identical to the model described in Experiment 6.1; we only vary the adversarial weight λ_{GAN}, investigating the values 0, 1, 10, and 100 for the adversarial weight.

Results. The values for the three reconstruction quality measures are shown in Table 6.1 for different choices of λ_{GAN}. One can observe that for the values 0, 1 and 10 of λ_{GAN} the SSIM and PSNR are very similar, although they are highest for $\lambda_{\text{GAN}} = 1$. The same can also be observed for the mean L1 distances.

	$\lambda_{GAN} = 0$	$\lambda_{GAN} = 1$	$\lambda_{GAN} = 10$	$\lambda_{GAN} = 100$
Reconstruction (L1) Loss	63.69	**57.88**	62.74	72.44
SSIM	33.01	**33.79**	33.36	31.94
PSNR	0.8269	**0.8299**	0.806	0.7705

Table 6.1.: Experimental results with varying λ_{GAN}. All values were calculated on the validation set.

Looking at the training loss curves shown in Figure 6.3, it can be seen that for $\lambda_{GAN} = 0$ (in blue), the discriminator loss and generator loss remain constant as expected, since they are not optimized. At the same time, the training reconstruction loss is the lowest. In the model training with $\lambda_{GAN} = 1$ (in green), one can observe that the discriminator loss decreases while at the same time the generator loss increases and is not able to fool the strong discriminator. The reconstruction loss is similar to the one with $\lambda_{GAN} = 0$. For $\lambda_{GAN} = 10$, the discriminator and the generator loss both decrease and seem to find a balance, although the discriminator is still stronger. The reconstruction loss does not suffer significantly. Finally, for $\lambda_{GAN} = 100$ (in red), the generator loss decreases even further, and we can observe a balance between generator and discriminator loss. However, the reconstruction loss is significantly higher than for the other models.
Finally, we also look at the visual difference of reconstructions for different values of λ_{GAN}. In Figure 6.4, three random subvolumes that are drawn from the validation set are shown on the left side, together with their reconstruction for different choices of

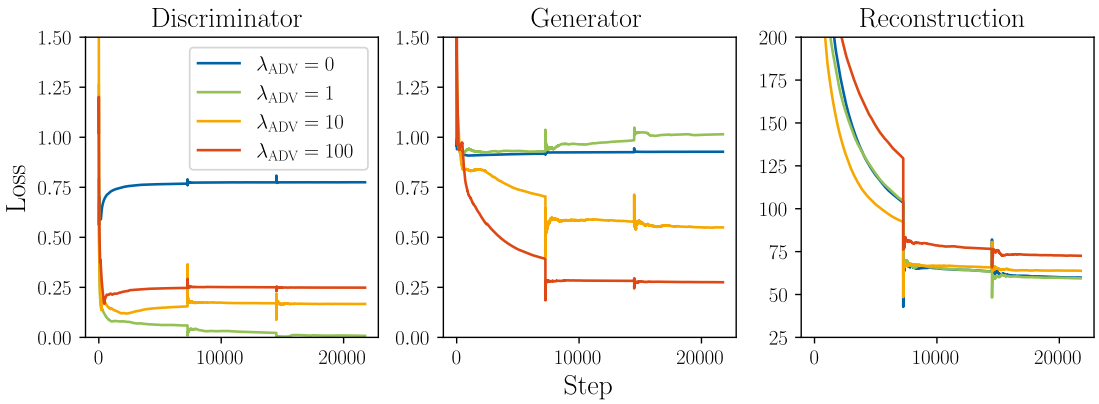


Figure 6.3.: Training loss curves for different choices of λ_{ADV}.

λ_{GAN}. We can observe that, for $\lambda_{GAN} = 0$ and $\lambda_{GAN} = 1$, the reconstructions are slightly blurry, while they become increasingly sharper and more detailed for $\lambda_{GAN} = 10$ and $\lambda_{GAN} = 100$. However, especially for the second sample, the reconstruction from the model with $\lambda_{GAN} = 100$ contains details that do not exist in the original image (e.g., in the upper right corner).
Based on the visual results combined with the training loss analysis and reconstruction quality measure, we decided to use a model with $\lambda_{GAN} = 10$ for all other experiments.

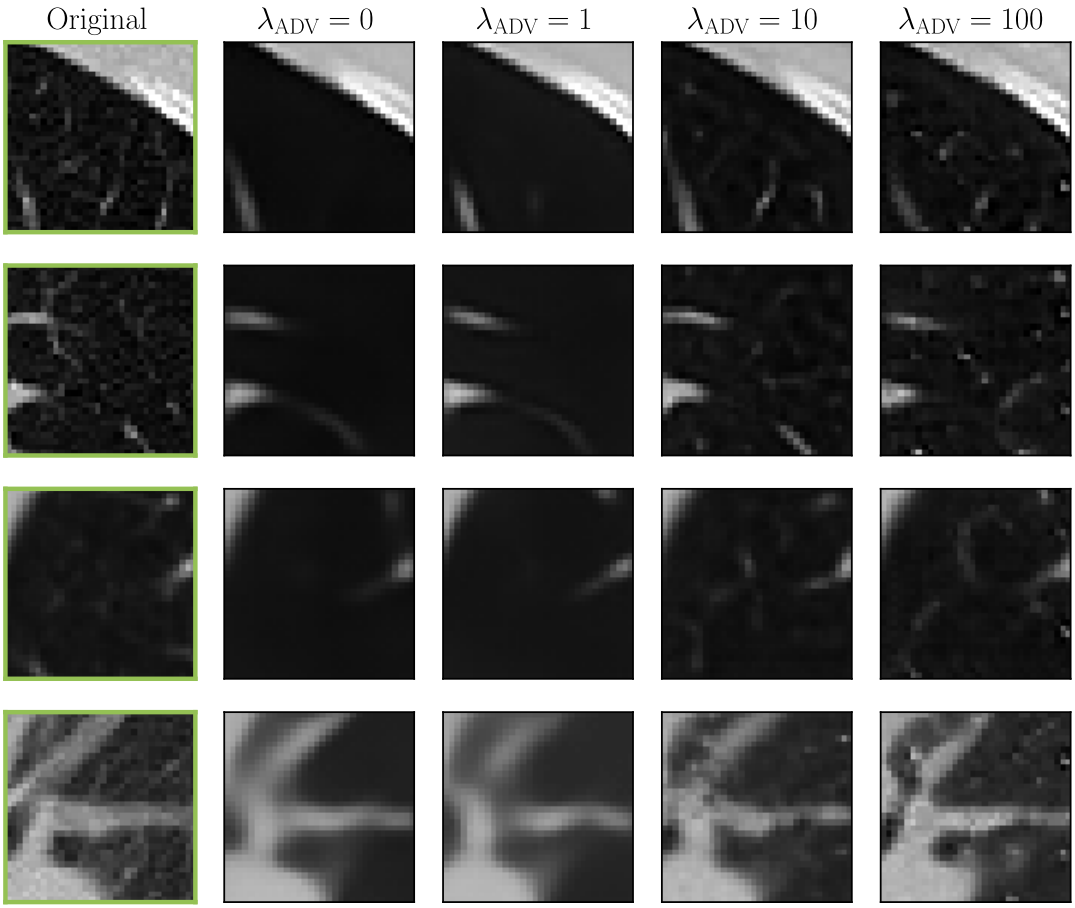


Figure 6.4.: **Reconstruction examples for different choices of λ_{ADV}.** The first column shows original volumes, and the other columns contain reconstructions of different models.

6.1.2. Experiment 1.2: The Effect of Fine-Tuning on Nodule Reconstructions

As described in Section 5.6.1, fine-tuning the VQ-GAN with subvolumes potentially containing nodules could improve reconstruction quality for unhealthy lung subvolumes, which is investigated in this experiment.

Question. Does fine-tuning the VQ-GAN on nodule subvolumes improve the reconstruction quality of nodule subvolumes compared to a VQ-GAN that is only trained on randomly drawn subvolumes?

Quality Measures. In order to assess the reconstruction quality of unhealthy tissue, the Reconstruction (L1) loss, the PSNR and the SSIM on the subvolumes containing nodules from the validation set of *Dataset B* are compared for a VQ-GAN with and without fine-tuning.

Data. The model is first trained on the same data as in Experiment 6.1, consisting of 128 randomly drawn subvolumes for each image of the training set of *Dataset A*. For fine-tuning, the training set of *Dataset B* is used, from which 6,474 subvolumes containing nodules are extracted based on automatic nodule segmentation as detailed in Section 4.2.

Hyperparameters. We use the same hyperparameters as in Experiment 6.1. The model is trained for three epochs and then fine-tuned for two more epochs. A comparison is performed between the model achieving the lowest validation loss within the first three epochs and the model achieving the lowest validation loss within the final two epochs.

Results. The results of the reconstruction loss, the PSNR and the SSIM for the VQ-GAN with and without fine-tuning are shown in Table 6.2. It can be seen that fine-tuning on nodules can improve all reconstruction quality measures, although they only improve by a small amount.

	VQ-GAN without fine-tuning	VQ-GAN with fine-tuning
Reconstruction (L1) loss	67.2	**61.08**
PSNR	32.8	**33.41**
SSIM	0.7899	**0.8179**

Table 6.2.: **Comparison of reconstruction quality of a VQ-GAN with and without fine-tuning.** All metrics were calculated on the validation subvolumes that contain nodules from *Dataset B*.

Visualizing some samples containing nodules from the validation set of *Dataset B*, shown in Figure 6.5, reveals that reconstructions after fine-tuning are more blurry than reconstructions without fine-tuning. Visually, fine-tuning does not achieve great improvement, which is why we do not use fine-tuned models in the following experiments.

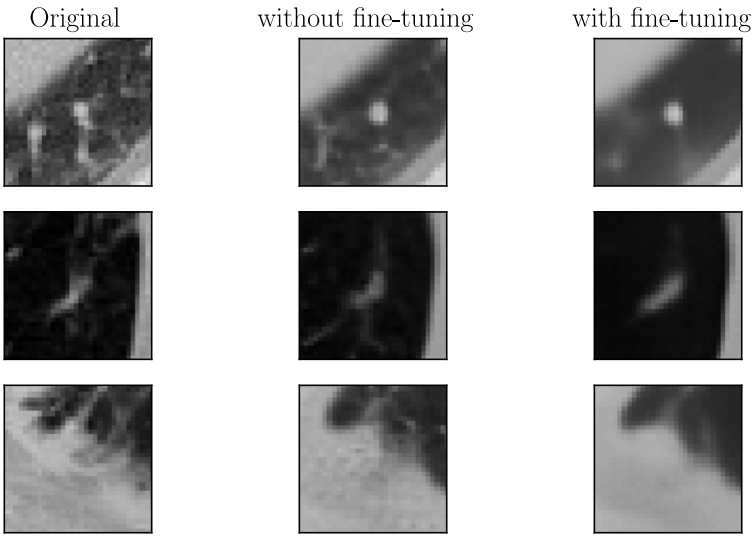


Figure 6.5.: Reconstruction examples of VQ-GAN with and without fine-tuning epochs on cancerous subvolumes.

6.1.3. Experiment 1.3: Latent Space Analysis: Comparing Latent Space Properties with and without Noisy Codebook Assignments

Closeness in the latent space does not necessarily correspond to similarity in the reconstructed images. However, this property of the latent space is crucial for some of the loss functions explained above: For the regression loss and weighted regression loss, latent vectors that are close to each other need to correspond to similar pixel reconstructions for the loss functions to work. For the regularized and the weighted CE loss, the distance between codebook indices needs to be correlated with the distance of reconstructed pixels to achieve meaningful results. That is why we are interested in the structure of the codebook vectors as well as in the structure of codebook indices with regard to reconstruction similarities.

Question. How structured is the latent space regarding the correspondence of embedding vector distances or index distances with distances in the pixel space when using the original VQ-GAN? Can we improve this correspondence when employing noisy codebook assignment? Does noisy codebook assignment also improve linear interpolation results?

Quality Measures. To evaluate the structure of the latent space, we project it to 2D using Principal Component Analysis (PCA) and analyze the result visually. Additionally, we look at the correlation between latent space distances and pixel space L1 distances. Furthermore, we create linear interpolations between latent space representations of randomly chosen images from the validation set and compare the results for a model with and without noisy codebook assignment.

Data. We use the same data as described in Experiment 6.1.

Hyperparameters. All hyperparameters of the VQ-GAN are identical to the ones of the model described in Experiment 6.1. We include noisy codebook assignment and set the standard deviation of the random discrete Gaussian noise to one.

Results. First, we look at the latent space structure in terms of embedding vectors with a model without noisy codebook assignment. Specifically, we investigate the correlation between L1 distances between pairwise embedding vectors and the corresponding L1 distances between their reconstructed counterparts. Note that this is not a perfect analysis since we only reconstruct individual embedding vectors to the pixel space, which are padded with zero vectors on all sides. Usually, reconstructed images contain l_z^3 embedding vectors, and neighboring vectors influence the reconstruction of a specific embedding vector. Still, reconstructions from individual embedding vectors allow for some insights. The Pearson correlation between latent vector distances and pixel reconstruction distances is 0.98, meaning that the distances are highly correlated, which is also confirmed by the visualization in Figure 6.6, in which we can see a strong linear correlation.

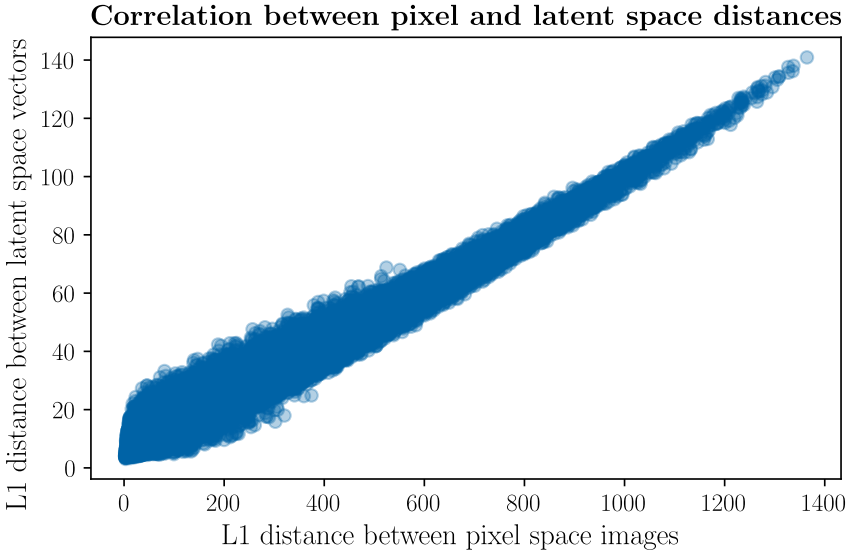


Figure 6.6.: Scatter plot showing the correlation between distances in decoded pixel space images and distances between latent space vectors.

Next, we look at the structure of the latent space with regard to codebook indices. The latent embedding vectors of the model with and without codebook indices are projected to two dimensions using PCA, so that they can be visualized. Figure 6.7 shows these projections, in which the individual projected embedding vectors are colored by their corresponding codebook indices. While for the model without noisy codebook assignment, the codebook indices do not follow any pattern, i.e., they are assigned randomly to the embedding vectors, the model with noisy codebook assignments achieves a latent space that is highly structured with regard to the codebook indices. A linear order of codebook indices can be observed in the projected latent space.
To investigate whether this induced order of the codebook indices has a meaningful effect in pixel space, we linearly interpolate the first and last images of a known time series and

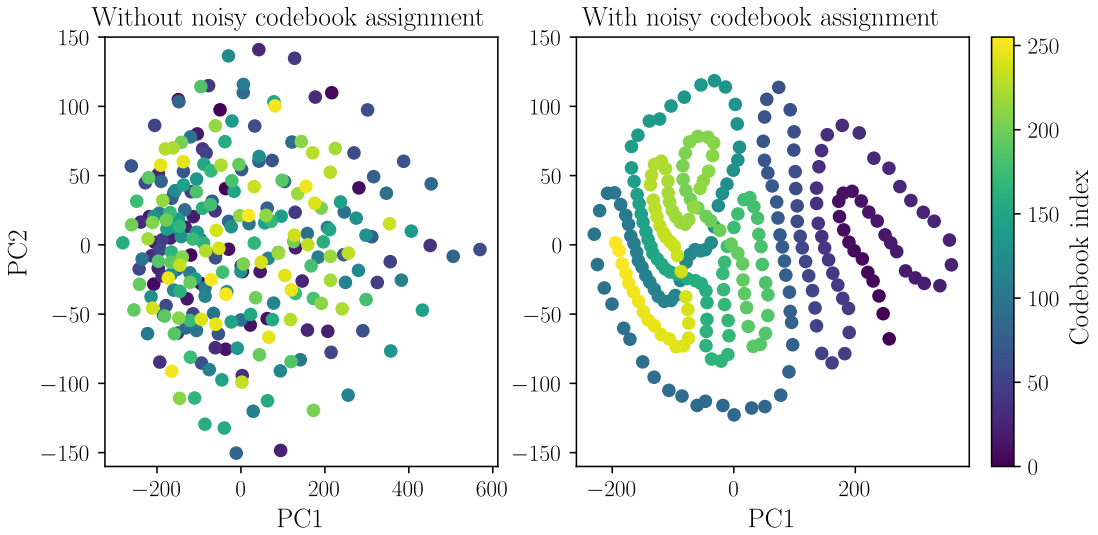


Figure 6.7.: **Visualization of the latent space embedding vectors projected to two PCA components of a VQ-GAN without (left) and with noisy codebook assignment (right).** The embedding vectors are colored by their assigned indices.

compare the results to the actual development of the subvolume. Linear interpolations are converted to integers before decoding, which means that any decimal digits are cut off. Figure 6.8 shows a manually selected time series from the validation set of *Dataset B*, which was chosen due to its comprehensible development. We can observe that, as expected, the interpolation of codebook indices is not meaningful at all without noisy codebook assignment. Although the noisy codebook assignment slightly improves the interpolation, it is still not very similar to the real development of the nodule, even though the nodule does not evolve in a complicated way. Additional visual examples, which can be found in Appendix A.1, show similar results.
In a final analysis, instead of interpolating the codebook indices, we interpolate the codebook vectors and visualize the results. We perform this analysis on both the model with and without noisy codebook assignment, although we do not expect an improvement for the model with noisy codebook assignment, since the noisy codebook assignment is primarily meant to structure codebook indices. Linearly interpolated vectors are mapped to the closest embedding vectors before decoding. Figure 6.9 shows the results of linear interpolations in the latent vector space. We can observe that linearly interpolating embedding vectors creates more meaningful results than interpolating codebook indices. Both models show a slightly different interpolation, but it is hard to tell which is medically more accurate. It should also be noted that the dates are not equally spaced because they are based on medical necessity. When looking closely, one can see that the original sequence shows rather an exponential growth of the nodule, whereas the linear interpolations, naturally, show a linear growth. Additional examples of linear latent space interpolations can be found in Appendix A.1.

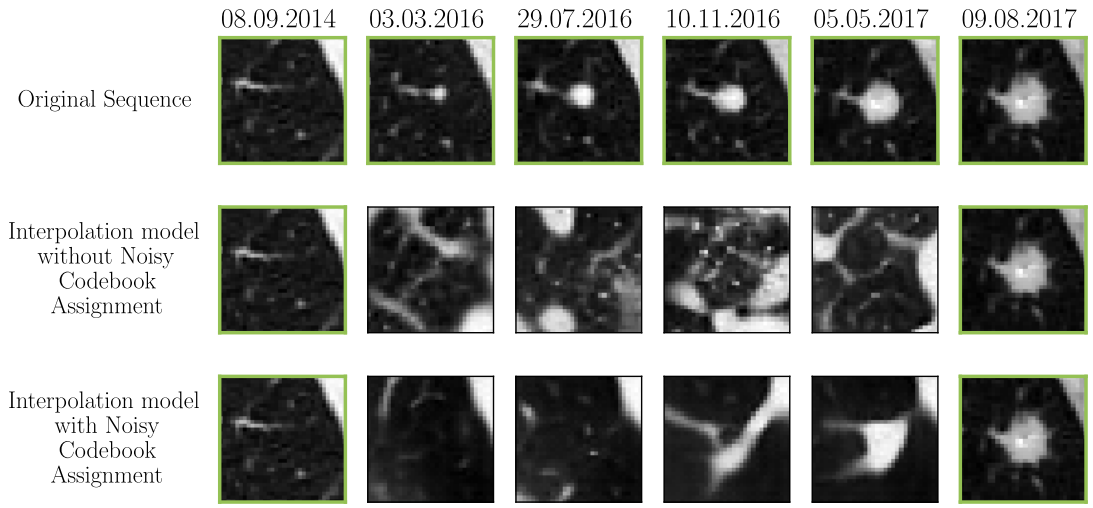


Figure 6.8.: **Linear interpolation of codebook indices.** The first row shows the real development of a nodule. The subvolume was manually chosen from the validation set of *Dataset B* because of the clear nodule development. In the second row, the latent space representations consisting of codebook indices of the first and last subvolumes are linearly interpolated by a VQ-GAN without noisy codebook assignment and plotted at the same dates as the real sequence. The first and last images are the original images. The third row shows interpolations by a VQ-GAN with noisy codebook assignment.

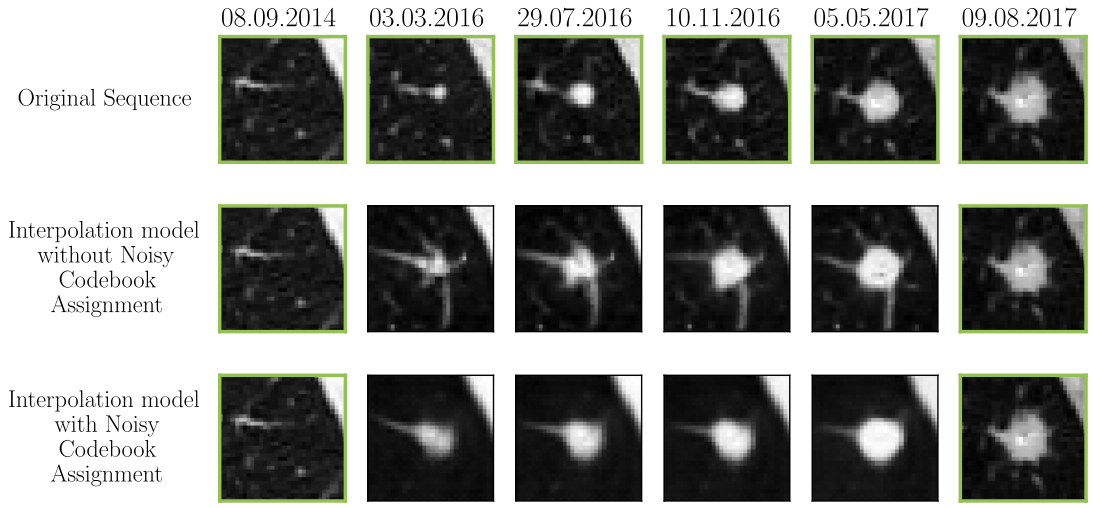


Figure 6.9.: **Linear interpolation of codebook vectors.** This is the same figure as Figure 6.8, with the difference that the linear interpolation is not performed on the codebook indices but on the codebook vectors. Interpolated vectors are mapped back to the closest embedding vectors before decoding.

6.2. Experiment 2: Forecasting Image Time Series in the Latent Space Using a Transformer Model

In this experiment, we investigate the performance of the forecasting model on latent space patches. We choose a trained VQ-GAN, as detailed in the following hyperparameter section, freeze its weights, and project all input and target subvolumes to the latent space. The forecasting transformer model with a CE loss is trained and evaluated on the latent patches.

Question. How does the proposed transformer model perform when predicting future patches in the latent space? Does it learn enough to accurately predict codebook indices, and to reach low CE values on its predictions?

Quality Measures. To answer these questions, we look at the CE loss on samples from the validation set of *Dataset B* projected to the latent space. We furthermore calculate the ratio of predicted codebook indices that match the target indices, as defined below:

$$r_{\text{indices}} = \frac{\sum_{i=1}^{l_z^3} \mathbb{1}_{\{\hat{\mathbf{z}}_i = \mathbf{z}_i\}}}{l_z^3} \tag{6.2}$$

This measure quantifies the prediction quality after the argmax function was applied to the logits to analyze the target patch in the state in which it is handed to the decoder.

Data. For training the VQ-GAN used for encoding all the following data, the data and hyperparameters listed in Experiment 6.1 are used. For the forecasting model training, the *Dataset B*, as explained in Section 4.2, is employed. Specifically, as detailed in Section 5.6.2, subvolumes are extracted based on a nodule segmentor. We filter the subvolume sequences with a minimum pairwise SSIM threshold of 0.1 and finally obtain 6,474 subvolume sequences from the training set and 2,051 subvolume sequences from the validation set. Each subvolume sequence is augmented with one random rotation and one random flipping operation, resulting in three samples per sequence. All resulting subvolumes are passed through the VQ-GAN encoder to obtain sequences of latent patches. Each patch sequence is subsampled as described in Section 5.6.2 by drawing $s = 5$ random sequences with a maximum length of $T_{\max} + 1 = 15$. For interpolation pre-training, one sample is generated per sequence, for which one random patch is dropped and used as the target. During inference, the validation set with segmented nodules is used, which are also filtered according to a minimum pairwise SSIM threshold of 0.1. On the validation sequences, we do not apply subsampling strategies, but instead take the last $T_{\max} + 1 = 15$ elements, where the last one is used as the target.

Hyperparameters. The employed VQ-GAN is the same as the one described in Experiment 6.1. The transformer model takes padded input sequences with a length of $T_{\max} = 14$ as input that are represented as triplets. The input triplets are embedded into vectors of size 768. The transformer forecasting model consists of $n_{att} = 4$ encoder layers, where each self-attention block has twelve heads. The same is true for the decoder; it has four layers with twelve cross-attention heads each. The MLP blocks consist of two layers with 3,072

neurons, each followed by a dropout layer. Embedding, MLP, and attention dropout rates are set to 0.2. The model is pre-trained on interpolation samples for three epochs, before training for another three epochs on forecasting samples. The final forecasting model is selected based on the epoch with the lowest validation loss achieved during forecasting training. The batch size is chosen to be 32, and the learning rate is set to 10^{-4}.

Results. The mean CE loss calculated on images from the validation set of *Dataset B* is **4.46** and the mean matching codebook index ratio $r_{\text{indices}} = \mathbf{0.1084}$.
To put the CE value into perspective, we look at a baseline CE value, namely the CE value when predicting labels uniformly at random. It can be approximated by $-\log(\frac{1}{n_{cb}})$. For $n_{cb} = 256$, this results in $-\log(\frac{1}{256}) \approx 5.55$. Although a CE value of 4.46 indicates predictions that are better than random guessing, the model still has a high degree of uncertainty. A matching codebook element ratio of only 11% also does not indicate high-quality predictions.

6.2.1. Experiment 2.1: Comparing Prediction Performance: Transformer Model vs. Mamba

As described in Section 5.3.2, we investigate whether the transformer model can be replaced by a more efficient (regarding time and memory consumption) Mamba model.

Question. Can the forecasting model perform equally well, or better, if a Mamba model is used instead of a transformer model?

Quality Measures. Like before, we look at the CE loss and the ratio of correctly classified codebook indices on the validation set to evaluate the models.

Data. The same data as described in Experiment 6.2 is used for model comparison.

Hyperparameters. For the transformer model, we use the model from the last experiment (Experiment 6.2). For the Mamba model, we choose the same embedding dimension and dropout rates. For the decoder part of the Mamba forecasting model, we use four layers with twelve attention heads for each cross-attention block, like in the transformer. We use the default values as in the original Mamba implementation[1] for the Mamba model parameters. Unlike in the transformer model training, we pre-train the Mamba forecasting model for five epochs, as it seems to learn more slowly. All other training hyperparameters are chosen to be the same for the Mamba and transformer forecasting models.

Results. Table 6.3 shows the CE value and matching codebook index ratio for the transformer model compared to the Mamba forecasting model. One can observe that the Mamba model achieves comparable, slightly better values.
Due to slightly higher performance and resource efficiency (see results in Experiment 6.4), we use the Mamba forecasting model instead of the transformer in the following experiments.

[1] Mamba implementation: `https://github.com/state-spaces/mamba`

	Transformer model	Mamba model
CE loss	4.46	**4.35**
Matching codebook index ratio	0.1084	**0.1184**

Table 6.3.: **Comparison of the forecasting quality in the latent space using a forecasting model with a transformer architecture versus a forecasting model with a Mamba encoder block.** The values show the mean performance on the validation set of *Dataset B*.

6.3. Experiment 3: Combined VQ-GAN and Forecasting Model for Pixel Space Results

While the performance of the forecasting model on the latent space is an important aspect, we are ultimately interested in the quality of the predicted images in the pixel space, as this is the output that can be useful for physicians. Therefore, this experiment combines the two-step model: the input images are projected to the latent space using the VQ-GAN, then the forecasting model is used to predict the future latent patch, which is finally projected back to pixel space using the VQ-GAN decoder. On these pixel space predictions, we can measure the overall prediction quality.

Question. Can the two-step forecasting model predict future pixel space images that are close to the target images?

Quality Measures. To measure forecasting quality, predicted future subvolumes are compared to their corresponding target subvolumes. We employ three quantitative measures: the L1 distance between prediction and target, the PSNR, and the SSIM. We divide the validation samples into groups based on the time difference between the most recent input image and the target image because we assume that predicting the future becomes more difficult for higher time differences. We choose three groups: the first group encompasses targets that are within 60 days, so approximately two months after the last input, the second group targets between 61 and 180 days, which corresponds to about two to six months, and the third targets that are further away than 180 days. To judge whether the resulting measures indicate high or low quality, we employ two benchmark models:

- **Copy Benchmark**: The copy benchmark model copies the last input image to predict the target; that is, we measure the similarity between the target image and the last input image.
- **Linear Latent Space Extrapolation**: As a second benchmark, we linearly extrapolate the two most recent latent space inputs in the codebook vector space to generate a future image.

A qualitative analysis is included in Experiment 6.3.1.

Data. We use the same data as in Experiment 6.2.

Hyperparameters. To reduce time and memory consumption, all following experiments are performed with the Mamba forecasting model, as it shows similar performance to the transformer forecasting model (see Table 6.3). All hyperparameters are chosen as in Experiment 6.2.1.

Results. Table 6.4 and Figure 6.10 show the similarities of prediction and target image of the two benchmark models and the Mamba forecasting model with CE loss grouped by the time distances between last input and target images in days. One can see that all

	L1 distance			PSNR			SSIM		
	≤60	61–180	>180	≤60	61–180	>180	≤60	61–180	>180
Copy Benchmark	**166.72**	**153.23**	188.97	**25.88**	**26.71**	**25.62**	0.4820	0.5620	0.5303
Linear LS Extrapolation	209.04	256.14	347.70	24.15	22.83	20.73	0.3872	0.3987	0.3107
Latent CE Mamba	176.12	153.65	**187.64**	25.55	26.54	25.55	**0.4821**	**0.5670**	**0.5395**

Table 6.4.: **Forecasting quality in the pixel space.** Quality measures are calculated on subvolumes with nodules from the validation set of *Dataset B*.

three measures show similar trends. With regard to the baseline models, it can be seen that copying the last image achieves significantly better results than linear extrapolation predictions. As one would expect, the extrapolation baseline gets worse the farther away the output image is from the last input. Contrary to expectation, this is not the case for the copy baseline.
One can observe that, in general, the forecasting model has difficulty beating the baseline models. The difference between the copy benchmark performance and the forecasting performance is very small for all measures.

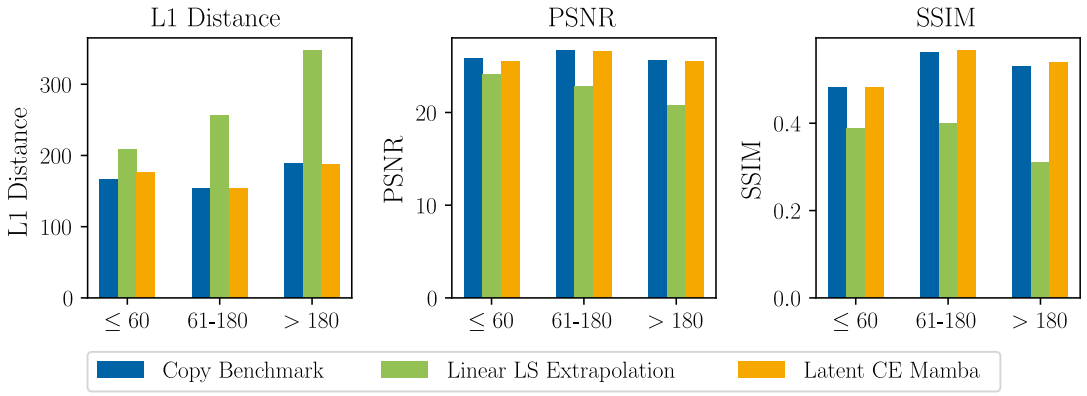


Figure 6.10.: **Forecasting quality in the pixel space.** The similarities of the target pixel images and the prediction images decoded to the pixel space are shown, grouped by the time distances between the last input and target. The values of three different similarity measures are depicted. The bar plots show the same data as Table 6.4.

6.3.1. Experiment 3.1: Comparing Prediction Performance: Models with Different Loss Functions

As detailed in Section 5.5, we exchange the CE loss on the codebook indices with various loss functions to improve prediction quality. This experiment analyzes which loss functions achieve predictions close to the target images. In a subsequent experiment (Experiment 6.3.3), we will also look at the prediction quality regarding temporal continuity.

Question. Which loss functions help to produce prediction images at future timepoints that are more similar to the target images?

Quality Measures. Like in the last experiment, we look at the L1 distance, the PSNR and the SSIM of the prediction and target images grouped by how far in the future the predictions are. We also visualize a time series with predicted future images for each employed loss variant.

Data. Again, we use the data as detailed in Experiment 6.2.

Hyperparameters. In general, the hyperparameters are chosen as in Experiment 6.3, that is, we take a Mamba forecasting model and exchange the loss function. Refer to Section 5.5 for details on the loss functions. Hyperparameters that are specific to the loss function, as well as hyperparameters that were altered for a loss function, are enumerated in the following list.

- **Regression Loss on Latent Space Vectors**: Two versions of the regression loss model are trained: one based on images encoded with the default VQ-GAN and one with the VQ-GAN with noisy codebook assignment. The number of pre-training epochs is increased to ten since the validation loss keeps decreasing until roughly the tenth epoch.

- **Regularized Cross Entropy Loss**: The weight of the regularizer α (refer to Equation 5.13) is set to 0.1. Furthermore, the regularization loss term is only meaningful when the images have been encoded by the VQ-GAN with noisy codebook assignments since it assumes a similarity of pixel decodings for closeby codebook indices, so it is trained on these latent patches.

- **Weighted Cross Entropy Loss for Interpolation**: The same is also true for the weighted CE loss, so it is also trained with samples encoded by the VQ-GAN with noisy codebook assignment. The specific hyperparameters for this loss, namely the number of interpolation samples that have real targets and the number of interpolation samples with targets between two existing images per training sequence, are chosen to be two and three, respectively. Two versions of the model are trained: one with the uniform sampling scheme and one with sampling close to available images. Refer back to Section 5.5 for details on the sampling schemes. The interpolation loss is used for the three pre-training epochs before it reduces to a simple CE loss for the remaining three forecasting training epochs.

- **Weighted Regression Loss for Interpolation**: The model with the weighted regression loss is trained on images encoded by a VQ-GAN with noisy codebook assignment. Like the model with the weighted CE loss, it takes two interpolation samples with available targets and three with interpolated targets. It is also trained with both sampling schemes for the target times between available images. The number of pre-training epochs is increased to five, but not to ten as for the regression model without weighted samples, since one epoch now contains five times as many training samples.

- **Loss in Pixel Space**: Due to time constraints, the pixel space model is only trained for one pre-training epoch and one forecasting training epoch. Experiments that are not shown in this work have shown that training for more epochs leads to an increasing validation loss.

Results. The grouped performance metrics are shown in Table 6.5 for the two benchmark models, the Mamba model as used until now, and Mamba models with the described loss function adaptations. Additionally, Figure 6.11 visualizes the SSIM data from the table to facilitate analysis.
In the table, one can observe that four of the proposed loss functions, namely the regression loss applied to data encoded by a VQ-GAN with noisy codebook assignments, the two versions of weighted CE, and the weighted regression loss with time points samples close to available images, can slightly improve results compared to the CE loss for almost all metrics and groups. However, no model significantly outperforms the baselines, and there is no model that has a clear advantage over the others. The pixel loss model performs poorly (SSIM values 0.09–0.11), which is why it is not included in further experiments. Unexpectedly, the regression loss works significantly better when using noisy codebook assignment (SSIM 0.34–0.38 without vs 0.51–0.58 with NCB). This is why the weighted regression loss model was also trained on samples encoded with the noisy codebook assignment encoder. The sampling scheme makes a difference in the performance of the weighted regression loss: when the time points in between available images are sampled close to the real images, the model performs better than when they are drawn uniformly at random (SSIM 0.51–0.59 vs 0.45–0.56). For the weighted CE loss, this difference in sampling schemes cannot be observed. Most models perform best on the group with targets 61–180 days away from the last input image.
Since the performance measures are hard to interpret, we now look at a visual example (Figure 6.12) of how the different models predict the development of a nodule. We again use the same sequence as chosen in Experiment 6.1.3 because of its comprehensible development. The predictions for future time points by models with different loss functions are shown in Figure 6.12. The first row shows the original sequence, the following rows visualize the predictions based on the first three images of the original sequence. Predictions are generated for the dates at which the real images have been taken and at a date three years after the last available image. Additional examples that were randomly drawn from a manually selected subset of the validation set are shown in Appendix A.2.
Looking at the predictions, it is apparent that none of the models is able to accurately forecast the growth of the nodule. Instead, all forecasting models predict images that are static over time and resemble one of the three input images rather than the target image. The predictions look very similar for different loss functions, apart from the

	L1 distance			PSNR			SSIM		
	≤60	61–180	>180	≤60	61–180	>180	≤60	61–180	>180
Copy Benchmark	166.72	153.23	188.97	25.88	26.71	25.62	0.4820	0.5620	0.5303
Linear LS Extrapolation	209.04	256.14	347.70	24.15	22.83	20.73	0.3872	0.3987	0.3107
Latent CE Mamba	176.12	153.65	187.64	25.55	26.54	25.55	0.4821	0.5670	0.5395
Latent Regression Mamba	249.74	284.60	273.98	23.15	22.55	22.64	0.3369	0.3849	0.3849
Latent Regression Mamba (NCB)	172.01	167.18	178.51	25.98	26.50	25.99	0.5082	0.5840	**0.5636**
Regularized CE	**160.06**	**147.66**	177.90	**26.20**	26.89	25.90	0.5064	0.5856	0.5549
Weighted CE, close to real sampling (NCB)	164.43	148.00	**176.32**	26.07	26.86	**26.06**	0.5071	0.5854	0.5595
Weighted CE, uniform sampling (NCB)	161.63	147.73	176.54	**26.20**	**26.90**	26.01	**0.5140**	0.5876	0.5588
Weighted Regression, close to real sampling (NCB)	170.05	164.46	177.68	26.06	26.61	26.01	0.5097	**0.5891**	0.5619
Weighted Regression, uniform sampling (NCB)	189.47	177.37	179.82	25.30	26.11	25.93	0.4507	0.5606	0.5603
Pixel loss	372.07	430.93	415.48	19.35	18.32	18.58	0.0909	0.1096	0.1038

Table 6.5.: **Extrapolation Metrics for Different Loss Functions.** The (NCB) indicates that data encoded by a VQ-GAN with noisy codebook assignment is used.

latent regression Mamba trained with images encoded by a VQ-GAN without noisy codebook assignment, which does not even capture the structure of the input images. The regression loss on data encoded with noisy codebook assignment produces images that appear slightly more blurry than images produced by models with CE loss variations. The linear interpolation baseline model predicts reasonable results for closer target dates, but starts to predict unrealistic images in the far future. This can especially be observed in the additional examples in Appendix A.2.

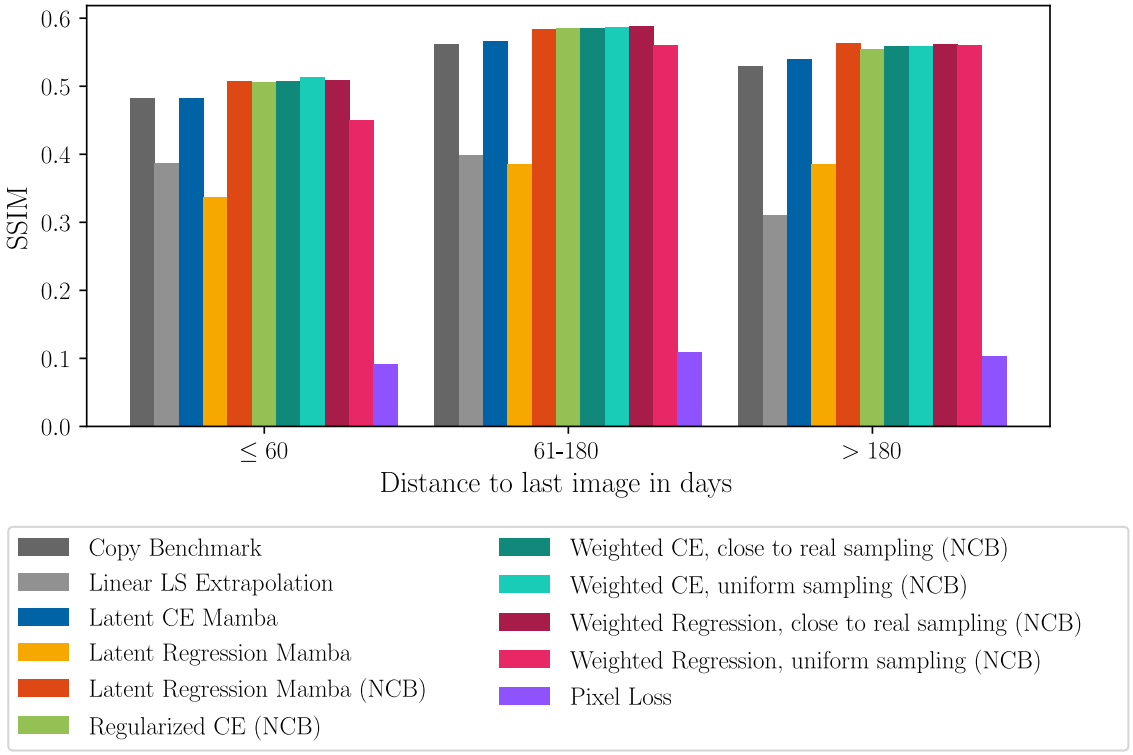


Figure 6.11.: **Extrapolation Metrics for Different Loss Functions.** The similarities of the target pixel images and the prediction images in terms of the SSIM are shown for models with different loss functions, grouped by the time distances between the last input and target. The bar plot shows SSIM values as also listed in Table 6.5.

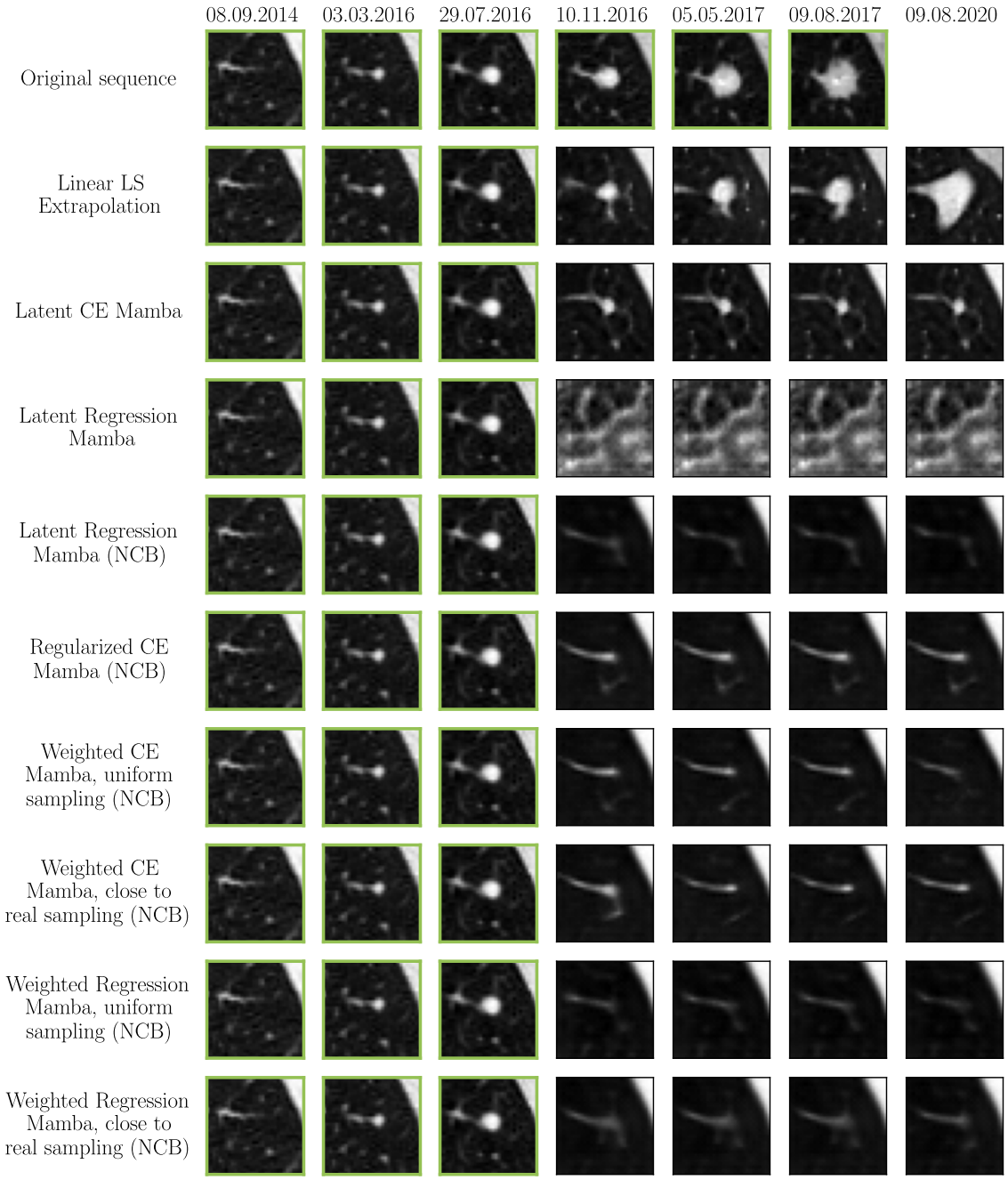


Figure 6.12.: **Extrapolation results for models with different loss functions.** Each model received the first three images of the shown sequence as input. The top row shows the original sequence. All real images are marked by a green frame.

6.3.2. Experiment 3.2: Comparing Prediction Performance: Interpolation Pre-training

As described in Section 5.6.2, we investigate how interpolation pre-training affects the prediction quality. Specifically, we compare models with a CE loss and regression loss with pre-training to the same models without pre-training. The other loss functions are not meaningful without interpolation pre-training.
Furthermore, we are interested in how well the models perform in the interpolation task. All loss functions will be included in this analysis.

Question. Does interpolation pre-training positively affect extrapolation results? How well do the models perform on interpolation tasks after interpolation pre-training?

Quality Measures. To assess prediction quality, we again look at the three pixel space similarities between predictions and targets, namely the L1 distance, the PSNR, and the SSIM. They are used to compare models with and without pre-training. Furthermore, we look at the interpolation quality of models with different loss functions, using the same metrics. Note that the linear extrapolation baseline model becomes a linear interpolation model, interpolating the nearest patch before and after the target, and the copy benchmark copies the image before the interpolation target. For a qualitative analysis, we visualize an interpolated sequence.

Data. We use the same data as detailed in Experiment 6.2. For interpolation validation, we take the validation set of *Dataset B* and, for each time series, use every image except the first and the last one as the target once.

Hyperparameters. The hyperparameters are used as defined in Experiment 6.3.1. For the model with the latent regression loss, we use the version trained on samples encoded by the VQ-GAN with noisy codebook assignment, since it performed better in the previous experiment. For the models without pre-training, we simply set the pre-training epochs to zero. For the interpolation performance analysis, models are selected based on the pre-training epoch yielding the lowest validation loss on the interpolation task.

Results. Table 6.6 contains the similarity measures between the target and predicted future image for models with and without pre-training, respectively. The data indicates that pre-training does not achieve better results for the model with CE loss, while it significantly increases the prediction quality for the regression model. Because the performance of the model with CE loss does not deteriorate much, pre-training is still performed for all models in all experiments, since the models will also be analyzed regarding their interpolation performance.
Looking at the interpolation results presented in Table 6.7 and Figure 6.13, reveals a structure similar to that observed in the extrapolation results (Table 6.5): Like in the extrapolation task, the regression model trained on data encoded by a model without noisy codebook assignment performs poorly on interpolations, while all other models achieve results very similar to the baselines. Again, no clear winner can be determined. One difference that can be seen is the slightly better performance of the linear interpolation baseline compared to the copy benchmark in terms of L1 distance and SSIM.

	L1 loss		PSNR		SSIM	
Pre-training	✓	x	✓	x	✓	x
Latent CE Mamba	165.05	**164.18**	26.15	**26.20**	0.5458	**0.5493**
Latent Regression Mamba (NCB)	**170.50**	273.60	**26.30**	22.64	**0.5659**	0.3917

Table 6.6.: Comparison of extrapolation performance of models with and without pre-training.

	L1 distance	PSNR	SSIM
Copy Benchmark	120.31	**29.08**	0.6473
Linear LS Interpolation	117.50	28.86	0.6507
Latent CE Mamba	120.48	28.55	0.6500
Latent Regression Mamba	287.69	22.48	0.4448
Latent Regression Mamba (NCB)	124.67	28.67	0.6644
Regularized CE (NCB)	**115.94**	28.91	0.6669
Weighted CE, close to real sampling (NCB)	116.54	28.86	0.6612
Weighted CE, uniform sampling (NCB)	116.99	28.87	0.6625
Weighted Regression, close to real sampling (NCB)	118.35	29.02	**0.6744**
Weighted Regression, uniform sampling (NCB)	156.69	27.42	0.6482

Table 6.7.: Interpolation Metrics for Different Loss Functions.

Again, we also look at a visualization to assess interpolation prediction qualities. Figure 6.14 shows the interpolations for the same original sequence as used in previous experiments. The interpolation models received the first and last image of the original sequence as input. In comparison to the extrapolation results, this time, the predictions are not all static over time. The CE model first predicts images that are close to the first input, and for the final prediction outputs a slightly larger nodule. The latent regression model without noisy codebook assignment in the encoder shows poor performance, as already indicated by the performance measures. The latent regression model (NCB) and the regularized CE model both predict very static images that are similar to the first input image. The weighted CE model and the weighted regression model with uniform sampling produce the most visually convincing interpolations. The models with the sampling scheme that samples target times close to available images show a course where the nodule is rather shrinking than growing, as opposed to what the last input would indicate. Although these observations are only true for this one example, when also

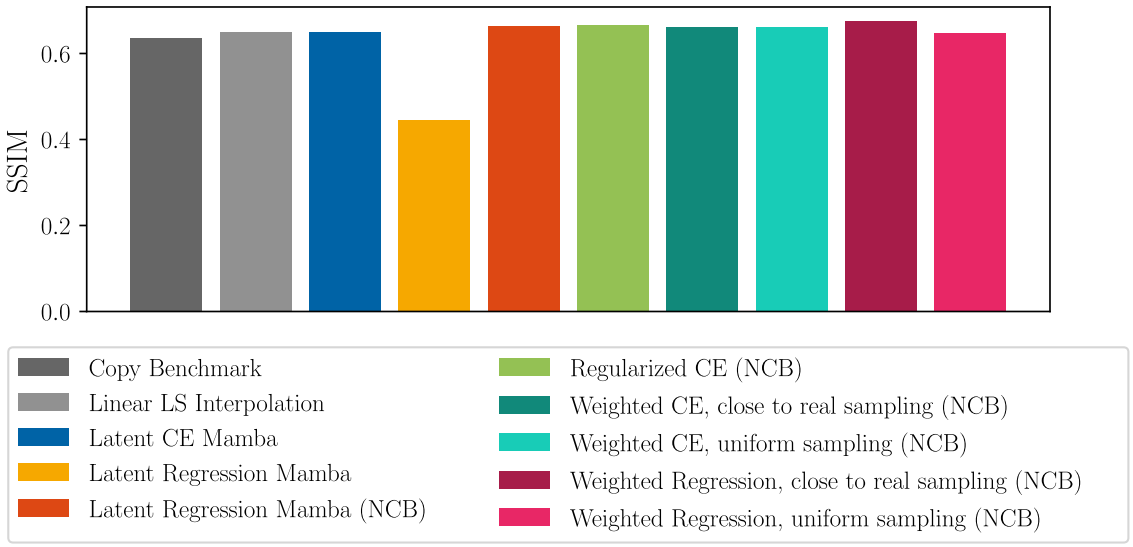


Figure 6.13.: **Interpolation Metrics for Different Loss Functions.** The bar plot shows the SSIM values as presented in Table 6.7.

looking at additional examples as shown in Appendix A.3, one can see a few trends: The regularized CE model predicts rather static developments that stay close to the first image. The weighted regression model with uniform sampling seems to make the most reasonable predictions, which is also supported by the SSIM value discussed above.

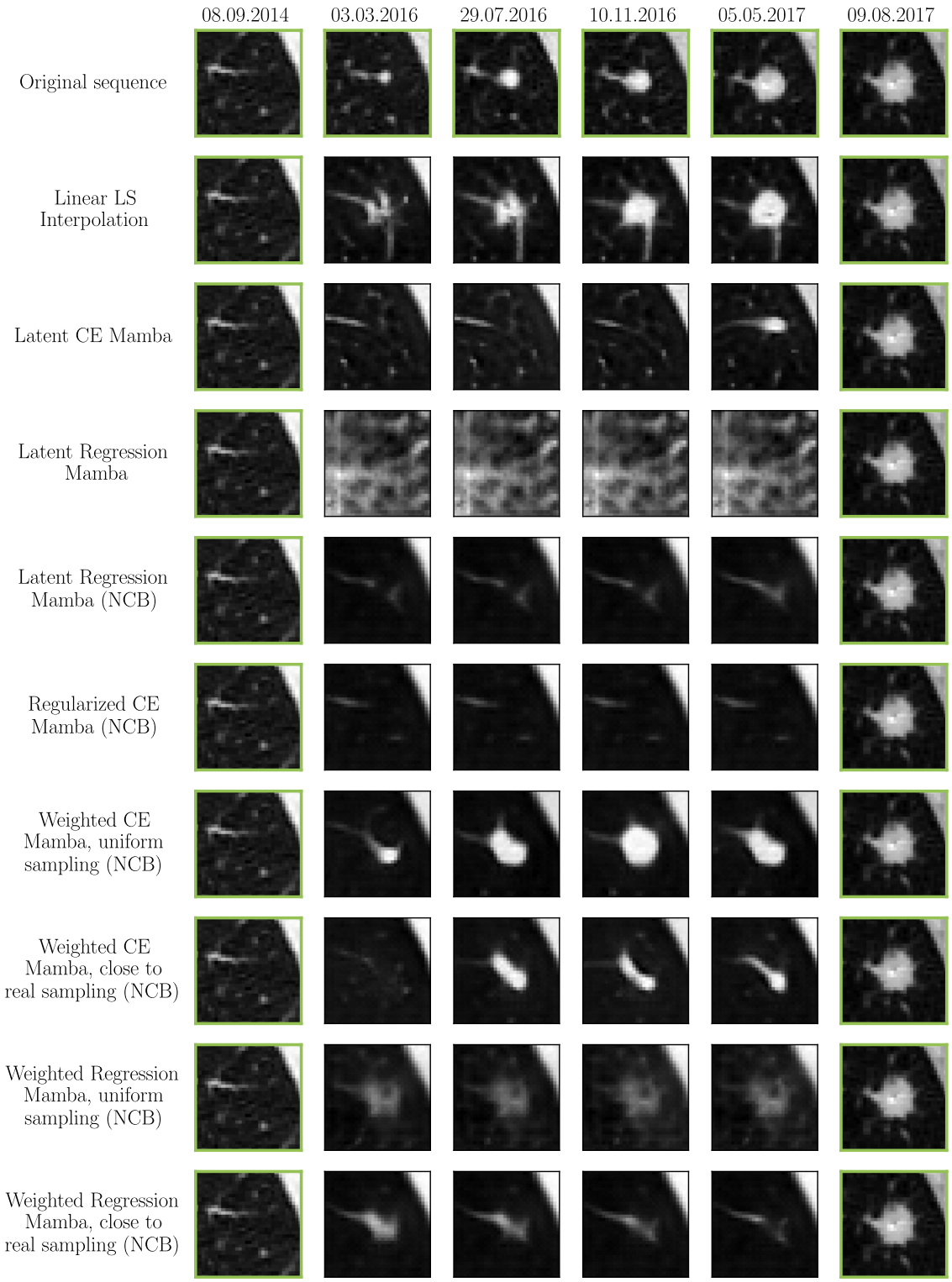


Figure 6.14.: **Interpolation results for models with different loss functions.** Each model received the first and last image of the shown sequence as input.

6.3.3. Experiment 3.3: Exploring Temporal Continuity in Forecasting

One property that we naturally expect from interpolation and extrapolation models is temporal continuity. Lung nodules, like other real-world processes, evolve gradually over time, with smooth changes in size, shape, and texture rather than abrupt variations. Predicting a CT image at a specific time-point accurately requires the model to understand the time dimension and its implications for continuity. However, this concept is difficult for the model to learn, in particular, because the input data does not contain closely spaced images that could teach the model the correspondence between small time steps and small, smooth changes in the image. Moreover, the proposed model does not contain an explicit mechanism to ensure or encourage continuity.
The regularized and weighted loss functions are meant to aid the model in learning temporal continuity. The regularized loss does so by penalizing large changes in small time steps in the codebook index space. The weighted CE loss tries to support the model by providing more closely spaced samples by implicitly linearly interpolating the codebook indices. The weighted regression loss tries to achieve the same, but by interpolating in the embedding vector space. We investigate whether the proposed loss functions achieve the goal of improving the continuity of predictions over time.

Question. How continuous are the predicted interpolations over time? Do the regularized and weighted loss functions improve continuity?

Quality Measures. To assess temporal continuity, we look at a specific example from which we take the first and last image to interpolate them. Interpolation images are predicted daily after the first until the last date, and the SSIMs are calculated between the prediction and the first image, and the prediction and the last image. Continuity is evaluated by visually determining sudden jumps in plots visualizing the SSIMs.

Data. For model training, the same data as detailed in Experiment 6.2 is used. For the qualitative analysis, the same time series as in the previous experiment (see Figure 6.13) is used, and additional examples randomly extracted from a manually selected subset of the validation set of *Dataset B* are shown in Appendix A.4.

Hyperparameters. The hyperparameters are used as defined in Experiment 6.3.1.

Results. Figure 6.15 shows plots based on interpolations between the first and last image of the time series shown in Figure 6.13. The first plot shows the similarity, measured by the SSIM, between all images shown in Figure 6.13 and the first image as blue cross marks and the similarity between all images and the last image as green marks. Note that the marks have been linearly interpolated to visualize the approximate development, which is unknown but assumed to be smooth. All other plots show the SSIM between interpolation images estimated in one-day intervals and the first and last images, respectively. It can be observed that the linear interpolation of the latent space provides a relatively smooth interpolation trajectory. The models with CE loss and weighted CE loss functions produce progressions with sudden jumps that appear as steps in the visualizations. The regularized loss produces a progression path that stays constantly close to the first image before

suddenly changing into an image closer to the last one. In comparison, all regression loss functions achieve relatively smooth results. Similar observations can also be made on the additional plots provided in Appendix A.4.

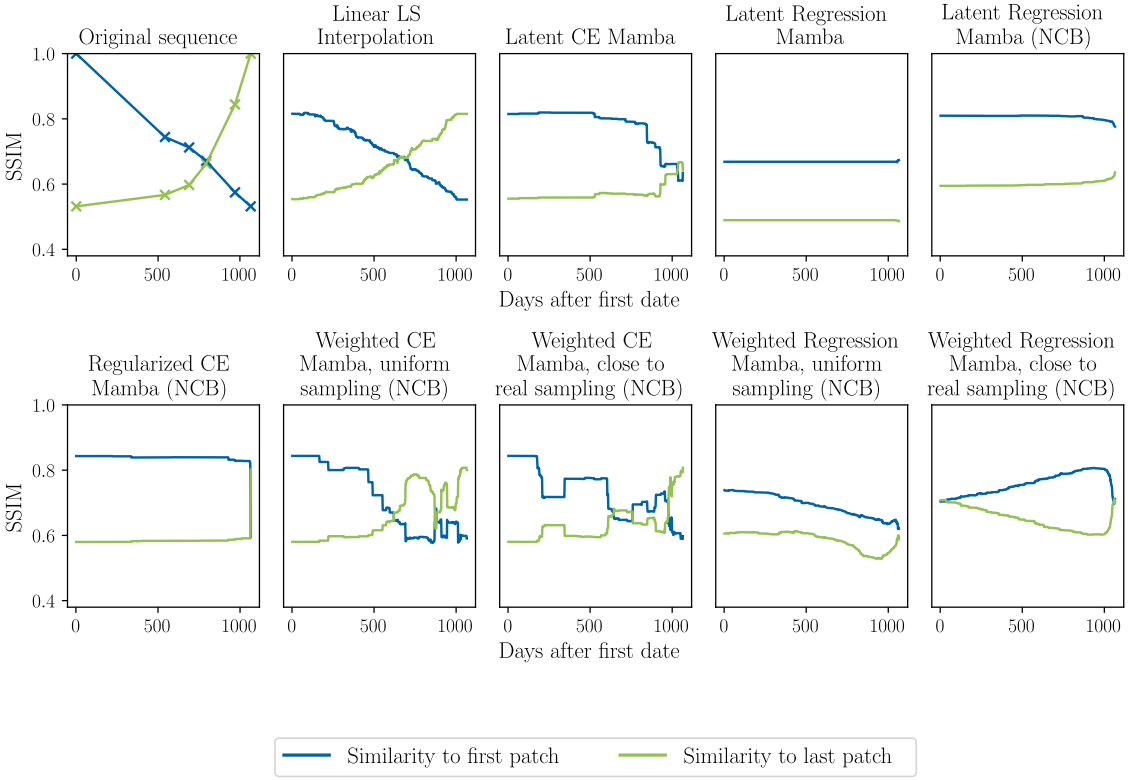


Figure 6.15.: **Continuity analysis of interpolations over time.** The plots show the SSIM to the first image (in blue) and to the second image (in green), when interpolating two images. Specifically, the same two images as employed in Figure 6.14 are also utilized for this analysis.

6.4. Experiment 4: Time and Memory Usage: Transformer vs. Mamba

In this final experiment, we briefly look at the time and memory consumption of the transformer model compared to the Mamba model. This experiment is meant to investigate the claim that the Mamba model is more efficient, especially in the proposed setting, and to justify the choice of using the Mamba model for most experiments.

Question. Does the Mamba model require fewer computational resources than the transformer model in the proposed setting?

Quality Measures. Since this experiment is not central to this work, time and memory requirements are only roughly approximated and not accurately assessed. We measure the training time for one epoch by calculating the difference between the time stamp before

and after the epoch. For assessing memory consumption, the GPU memory tracked by the metadata tracker AIM[2] is used, from which we extract the peak memory usage.

Data. The training data described in Experiment 6.2 is used for this experiment.

Hyperparameters. We compare the transformer model and Mamba model from Experiment 6.2.1.

Results. Regarding run time, the transformer model takes about 22 minutes per epoch, while the Mamba model requires only 5 minutes. As the transformer model is trained for six epochs, the overall training process takes about 132 minutes. To achieve comparable results, the Mamba forecasting model is trained for two more epochs, so overall eight epochs, summing up to roughly 40 minutes of training, which is still significantly less time than the transformer model takes. During training, the transformer model requires a maximum of about 27 GB of GPU memory, while the peak memory usage of the Mamba model is around 6 GB. It is important to keep in mind, however, that the transformer model in our configuration has significantly more parameters (around 57,000,000) than the Mamba model (around 7,000,000), which explains at least part of the difference in memory requirements.

[2] `https://aimstack.io/`

7. Discussion

The last section showed that, in general, the proposed model cannot accurately predict future images based on a time series of irregularly sampled 3D CT images. The model seems to fail to understand the time dimension and is unable to learn the necessary rules to make predictions of the future that resemble the target CT images. None of the proposed alterations to the loss functions can significantly improve the interpolation or extrapolation results.
The VQ-GAN allows for samples to be projected into a low-dimensional latent space with a high compression rate and reconstruct the main components. It thereby produces a highly structured embedding vector space. An additional noisy codebook assignment can also impose structure on the codebook index space. Linear interpolations in the embedding vector space appear promising.
However, predicting the future in this latent space presents a greater challenge. The forecasting model is not capable of capturing the patterns necessary to make high-quality predictions. Furthermore, it also struggles with the simpler task of interpolating image sequences.
There are several potential reasons for this. The data contains highly heterogeneous disease trajectories coming from patients who received various treatments. The progression of the disease is influenced by several external factors that are not accessible to the model. This may render certain changes impossible to predict. It is also plausible that the amount of training data is not sufficient for the model to learn intricate nodule developments, leading the model to overfit on the training data. Furthermore, the proposed model architecture may not be optimal for the task at hand. The lack of available closely sampled sequences hinders the model's ability to grasp the concept of temporal continuity. Additionally, the model does not incorporate any prior medical knowledge.
This chapter provides a thorough examination of the model's performance and offers explanations for the observed phenomena. The chapter is structured into separate discussions of each experiment. The last section gives an overview of limitations.

7.1. CT Image Reconstruction using a VQ-GAN (Experiment 1)

Question. Can the VQ-GAN compress the CT image subvolumes into low-dimensional quantized patches and still reconstruct them well?

We compress lung subvolumes with a rate of 1:512, which is a very high compression rate. The quantitative and qualitative results (Figure 6.1) both show that the low-dimensional representations can be approximately reconstructed, but some details are lost during the process. Lower compression rates would likely lead to better reconstructions, but the latent space forecasting model would then require substantially more memory. We did not investigate other compression rates. Although the reconstructions appear to be of reasonable quality, further assessment by clinicians is necessary.

7.1.1. Optimizing Adversarial-Reconstruction Balance in VQ-GANs (Experiment 1.1)

Question. Can the integration of the VQ-VAE into an adversarial framework improve results compared to a VQ-VAE which is not trained with an additional discriminator? How can the adversarial weight λ_{GAN} be chosen to optimize reconstruction quality?

Visual analysis of examples (Figure 6.4) suggests that the embedding of the model in an adversarial framework can indeed help to generate less blurry reconstructions. However, there is a trade-off between a low reconstruction loss and realistic-looking, sharp samples (refer to Table 6.1). For the experiments, the adversarial weight $\lambda_{\text{GAN}} = 10$ is used since the visual results look sharper, while the reconstruction loss does not suffer significantly. The improvement is rather subtle, producing only slightly sharper edges and preserving a few details.

7.1.2. The Effect of Fine-Tuning on Nodule Reconstructions (Experiment 1.2)

Question. Does fine-tuning the VQ-GAN on nodule subvolumes improve the reconstruction quality of nodule subvolumes compared to a VQ-GAN that is only trained on randomly drawn subvolumes?

Although the quantitative reconstruction measures (Table 6.2) suggest slightly improved results after training the model on additional subvolumes containing nodules, the visualizations (Figure 6.5) show that the reconstructions are again more blurry after fine-tuning. The reason could be that the reconstruction loss overpowers the adversarial loss in the fine-tuning epoch, where the discriminator loss is likely decreasing further, while the generator loss suffers. Fine-tuning has not been further investigated and has not been employed in other experiments.

7.1.3. Latent Space Analysis: Comparing Latent Space Properties with and without Noisy Codebook Assignments (Experiment 1.3)

Question. How structured is the latent space regarding the correspondence of embedding vector distances or index distances with distances in the pixel space when using the original VQ-GAN? Can we improve this correspondence when employing noisy codebook assignment? Does noisy codebook assignment also improve linear interpolation results?

The VQ-GAN without noisy codebook has a highly structured embedding vector space, meaning that distances in between latent space vectors are meaningful also for their reconstructed pixel decodings. This is reflected in the high Pearson correlation between the distances, as well as in the linear interpolation plots (Figure 6.9) that reflect a comprehensible nodule progression. The indices and their relations are randomly assigned in the VQ-GAN, such that interpolations have no meaning in the index space. Introducing noisy codebook assignment to the VQ-GAN is meant to change this by inducing an order to the codebook indices. As shown in Figure 6.7, noisy codebook assignment leads to a significantly more structured latent space with respect to the codebook indices. However, in the visualized interpolation of codebook indices (Figure 6.8), one can see that the nodule progression does look more reasonable, but is still not smooth. Our assumption is

that the pixel space volumes are too complex to put their latent representations into a linear order that results in meaningful interpolations. The vector representation structure does not seem to be negatively affected by the noisy codebook assignment and still produces comprehensible interpolation results (Figure 6.9), at least in the examples we investigated. Note that linear interpolations generally do not accurately represent nodule developments, since nodule growth can often be better described by other functions, such as the Gompertz function [11].

7.2. Forecasting Image Time Series in the Latent Space Using a Transformer Model (Experiment 2)

Question. How does the proposed transformer model perform when predicting future patches in the latent space? Does it learn enough to accurately predict codebook indices, and to reach low CE values on its predictions?

The transformer forecasting model does not achieve promising results, neither regarding the CE value around 4.5, which indicates results only slightly better than random guessing, nor regarding the ratio of correctly predicted codebook indices, which is at about 11 %. This can have various reasons. Neural networks learn complex patterns that do not necessarily match human intuition, making it difficult to understand how they arrive at their results. The lack of explicit causal mechanisms makes it difficult to determine whether a failure is due to insufficient data, suboptimal architecture, or other deficiencies. As a result, the following explanations remain speculative.
Firstly, the low ratio of correctly classified indices might be due to a high number of different classes and, at the same time, multiple indices that result from very similar encoded pixels. This likely makes it hard for the model to find specific patterns, as similar developments in the pixel space training sequences could be represented by completely different codebook index sequences, while the model does not get any information about this. It is punished equally much for indices that produce very similar pixel space reconstructions and indices that produce highly dissimilar reconstructions. However, in Figure 6.12 of a subsequent experiment, we can observe that at least the "background" pixels, that is, pixels that are not nodules, seem to be predicted accurately despite the very low ratio of matching codebook indices, which reinforces the hypothesis that different codebook indices can decode into similar pixel representations. The pixel reconstructions suggest that the forecasting model learns some patterns, presumably a pattern such as always predicting the same codebook indices as in the last input image.
A second point that could cause insufficient prediction quality is the non-autoregressive prediction. From an intuitive point of view, the predicted pixels depend on each other, so the model might benefit from autoregressive prediction similar to VideoGPT [78] or iterative improvement of the prediction by reintroducing it to the model like in MaskViT [79].
Thirdly, another shortcoming could be the embedding that is handed to the transformer model. The model might struggle to learn informative embeddings from time, channels, and values, or it might be difficult for it to disentangle the three very different concepts, which are summed up in a single vector.
Finally, the encoder-decoder architecture may not be perfectly suited, since the decoder is usually used to combine a sequence of variable length with the encoder's output. We,

on the other hand, do not enter a meaningful sequence to the decoder, but rather a fixed-length sequence of embeddings that inform the model about prediction positions and time.
There may also be several other reasons for poor model performance, some of which are discussed in the following paragraphs.

7.2.1. Comparing Prediction Performance: Transformer Model vs. Mamba (Experiment 2.1)

Question. Can the forecasting model perform equally well, or better, if a Mamba model is used instead of a transformer model?

The results show that the Mamba model achieves a mean CE and matching index ratio on the validation set comparable to that of the transformer model (see Table 6.3). However, as seen before, the transformer model performs rather poorly, which would probably allow many models to achieve similar accuracy. Nevertheless, we decided to use the Mamba model for the other experiments, as it performs well in replacing the transformer model in other research and saves computational resources.

7.3. Combined VQ-GAN and Forecasting Model for Pixel Space Results (Experiment 3)

Question. Can the two-step forecasting model predict future pixel space images that are close to the target images?

As indicated in the previous section, the forecasting model already has difficulties in predicting the future in the latent space, which naturally also affects the reconstructions of the predictions in the pixel space. The performance metrics (Table 6.4) all show that the forecasting Mamba model cannot significantly improve the prediction quality compared to the baseline models.

General reasons for the poor forecasting performance. Following up on the reasons for the poor performance, as started in the previous section, we now focus on the CT image sequences in the pixel space. A major challenge for the model is the high heterogeneity of the sequences since they come from different patients with various treatments. In order for the model to be able to learn the progression of such heterogeneous sequences, it would likely require more training data. Related to this is the issue of overfitting. When observing the validation loss after each training epoch, the validation loss starts to increase significantly after only few epochs, while the model would likely need more epochs to learn the complicated nodule progressions. Instead, it starts to memorize the training samples. Furthermore, a substantial obstacle for successful training is the unpredictable nature of some samples: For example, it is impossible for the model to predict registration errors, while already small discrepancies can affect the corresponding codebook indices. Another aspect that cannot be forecast are changes in imaging properties, such as contrast, noise, or resolution. Furthermore, we do not include treatment data in the model because it is not yet available to us. Therefore, a model cannot predict changes due to surgeries

or changes in nodule progression caused by different treatments. Sudden changes in the course of the patient's disease, for example, effusions, are also hard to predict from only a sequence of subvolumes without further information.
Aside from unpredictable changes, predicting nodule progression is still a challenge because it depends on many external factors. Visual inspection of CT image does not often allow a human, at least without a medical background, to infer the further nodule progression, which indicates that forecasting nodule progressions is an inherently difficult task — a task that the model needs to learn from scratch since it does not get any prior knowledge. Other deep learning models that aim to solve this task, as described in Section 3.4, do include prior knowledge, e.g., through smoothness constraints on deformations or through detailed, accurate tumor masks that are additionally input to the model. The proposed architecture is not well suited to introduce such prior knowledge, since it operates on a discrete latent space that is difficult to interpret.

Detailed analysis of pixel forecasting results. After having discussed general reasons for the difficulties in predicting nodule progressions, we now look at the results in more detail. The quality of the predictions is measured in three different groups: the target is 1-60 days away from the last image, 61–180 days away, or more than 180 days away. One would expect that the farther into the future the prediction is made, the less accurate it becomes. The results (Table 6.4) partly contradict this assumption: the model performs worse for the first group than for the second group. We hypothesize that closely spaced CT images often have an external cause, such as sudden deterioration of the patient's condition, imaging before and after surgery, or other events that require immediate imaging. Otherwise, one would probably avoid frequent exposure of the patient to radiation. This hypothesis is supported by the observation that the copy benchmark also performs worse on the closer group, although one would intuitively expect that copying the last image would work better if the target is closer to it.
Looking at the linear latent vector space extrapolation baseline, one can see that its prediction gets worse the farther away the target is from the last image. This suggests that the linear extrapolation is only meaningful near real images, while it deteriorates for images farther away.

7.3.1. Comparing Prediction Performance: Models with Different Loss Functions (Experiment 3.1)

Question. Which loss functions help to produce predictions that are more similar to the target images?

Overall, most adaptations of the loss function can only yield a marginal improvement in results compared to the CE loss (see Table 6.5). The performance differences are so small that they may be attributable to lucky random initializations. Besides, it has to be kept in mind that all results were calculated on the validation set on which we also performed all model choices, such that it is likely that the model slightly overfits on the validation set, which means that the results would probably be worse on an unseen test set.
The following discussion will be arranged into short analyses of the individual loss functions.

- **Regression loss:** The idea of the regression loss is to exploit the structural information contained in the embedding vector latent space. Interestingly, the regression

loss works significantly better for models trained on samples encoded by a model with noisy codebook assignment. The reasons for this observation are unclear to us. The superior performance might be due to the fact that the vector structure of the latent space is also influenced by the noisy codebook assignment, as can be observed in Figure 6.7. This adapted structure seems to be beneficial during training, producing more meaningful regressions of embedding vectors.
Furthermore, the experiment results (Figure 6.12) suggest that the decoded predictions of the model with regression loss (NCB) are slightly more blurry than those of CE loss models. Like in the VQ-VAE that is not embedded into an adversarial framework, this is likely due to the employed MSE loss, which averages over multiple possible outcomes for a sequence, resulting in blurry contours.
Overall, one can say that the additional structural information does not add a significant improvement in comparison to the CE loss model that does not receive this information.

- **Regularized CE loss:** The regularized CE loss is meant to avoid large changes in short time intervals. Therefore, it penalizes deviations from the previous input image, with the weight of the penalty depending on the time distance to the previous image — the larger the distance, the less deviations are penalized. One hypothesis on the impact of this loss is that the model is encouraged to copy the last input image if the target time is close to it, since in this case, the regularization term of the loss would be zero. When looking at the results in Table 6.5, we can observe that the regularized CE model outperforms the copy benchmark, unlike many other models. Since we expected the model to be encouraged to copy the last image, we would have expected the model performance to be closer to the copy benchmark than the results of other models. We do not have an explanation for the superior performance of this specific loss. The model could benefit from learning to copy faster and then being able to focus on deviations from copying, but its superiority could also just be due to a fortunate random initialization.

- **Weighted CE loss:** The weighted CE loss implicitly extends the interpolation training data by allowing for target time points for which no images are available. To replace missing targets, the codebook indices of the neighboring latent patches are linearly interpolated. As seen in Experiment 6.1.3, these linear interpolations are not necessarily realistic, which is why we did not expect the model to generate superior results compared to the other models, which it does for some metrics and groups. Based on the non-optimal interpolations, we would furthermore have expected that the model that uses additional targets that are sampled close to available images, which should be more realistic, would outperform the one with uniform sampling. This is not the case, but the results are very similar. We hypothesize that both models struggle to learn patterns in the data and that the better performance is due to random chance.

- **Weighted regression loss:** Different than the weighted CE loss, the weighted regression loss linearly interpolates codebook vectors instead of indices. Experiment 6.1.3 suggests that these interpolations are more meaningful, which is why we expected this loss to allow the model to benefit from the extended training data. Opposed to expectations, the model does not significantly outperform the latent

regression Mamba (NCB) or the weighted CE losses. This could be related to the observation that the linear interpolation baseline in Experiment 6.3.2 is not much better than the copy baseline, even if the example image sequence in Figure 6.14 would indicate that. Furthermore, adding linearly interpolated samples causes the model to learn linear progressions instead of the actual progressions, which could degrade performance.
For the weighted regression loss, the uniform sampling scheme does produce worse results, which could be explained by the hypothesis stated in the weighted CE loss point: interpolations further away from the real images could become less accurate.

- **Loss in pixel space:** Finally, the pixel space loss performs very poorly. This is likely due to the complicated patterns the model would have to learn to correctly predict every individual pixel. Additionally, the loss is challenging to backpropagate, and the straight-through estimator might impede the training process.

Overall, none of the loss functions helps to generate high-quality predictions. The loss functions do not affect some of the issues discussed for the CE loss model: unpredictable samples do not become predictable when changing the loss functions, the models still do not include prior knowledge, despite the regularized CE loss that tries to constrain sudden changes, and the training data is still very heterogeneous.

7.3.2. Comparing Prediction Performance: Interpolation Pre-training (Experiment 3.2)

Question. Does interpolation pre-training positively affect extrapolation results? How well do the models perform on interpolation tasks after interpolation pre-training?

The task of interpolating a sequence is related to extrapolation, but it should be significantly easier. Interpolation pre-training allows us to use more images of a sequence as a target, which considerably enlarges the training data. When looking at the results in Table 6.6, we can see that the regression model significantly benefits from the pre-training, while the CE model does not. For some unknown reason, the regression loss seems to allow for a better transition between interpolation and extrapolation. It is also possible that the heterogeneity of the training data does not allow for much better predictions than copying the last input. The CE model reaches this baseline without pre-training, which is why it might not benefit from it.
The interpolation results (Table 6.6) show that, again, all models struggle to outperform the interpolation benchmarks and stay very close to them. Since the interpolation task should be significantly easier, the poor results suggest that either the training data is not well suited for this problem and contains many non-trivial nodule progressions, or the proposed model is not optimal for this type of task. As mentioned above, it is difficult to pinpoint the exact reason why a deep learning model fails.

7.3.3. Exploring Temporal Continuity in Forecasting (Experiment 3.3)

Question. How continuous are the predicted interpolations over time? Do the regularized and weighted loss functions improve continuity?

Figure 6.15 shows that the model with the CE loss produces discontinuous interpolations, indicated by sudden steps. To evaluate why this is the case, we need to consider if and how the model is able to learn continuity. As the model is inspired by video generation models, for which continuity plays a very important role, we assume that the model is generally capable of learning continuity. However, we hypothesize that for videos, the model learns continuity from very closely spaced frames, where each frame differs only slightly from the previous one. Our data does not contain such closely spaced images from which the model could grasp continuity, and the progressions between successive images are unclear. The model also does not receive any additional explicit knowledge on continuity, such as, for example, in the ContiFormer [75], which employs a NeuralODE that explicitly enforces a continuous progression over time. Some of the alternative loss functions are meant to aid the model in learning continuity, nevertheless. Discussions on the loss function alterations and their specific results are listed below.

- **Regularized CE loss**: Like intended, the regularized CE loss causes the prediction to stay close to the past image. However, the prediction then suddenly jumps to the target instead of smoothly transitioning. The reason could be that the regularization term has too much weight assigned.

- **Weighted CE loss**: The weighted regression loss is meant to improve continuity by extending the data by closely spaced data points, which we hypothesize are necessary for the model to grasp the concept of continuity. However, the results show that the weighted CE loss cannot achieve this. This can be explained by the suboptimal codebook index interpolations that do not create smooth progressions, such that the model consequently also does not learn smooth progressions.

- **Weighted regression loss**: Again, additional data points are meant to enable the model to learn continuity. Since the vector space interpolations are considerably smoother than the index interpolations, the regression model generates smoother progressions. In this regard, the weighted regression loss is beneficial. However, individual predictions are still not accurate, as discussed in previous sections.

7.4. Time and Memory Usage: Transformer vs. Mamba (Experiment 4)

Question. Does the Mamba model require fewer computational resources than the transformer model in the proposed setting?

Yes, it does. The Mamba model requires significantly less time and memory resources than the transformer model. However, it also has considerably fewer parameters. A fair comparison would involve models with similar performance, which is the case for the Mamba and transformer models. However, both models perform poorly, making the comparison less meaningful — especially since the copy benchmark, which requires almost no computational resources, achieves similar results.

7.5. Limitations

The results indicate that the proposed model did not work as intended. This section discusses limitations that may have contributed to its shortcomings.

Limitations associated with the data. The first set of limitations is related to the data. As mentioned several times, the data comes from a heterogeneous routine cohort and therefore contains many very different disease trajectories. The nodules have not been manually segmented, and the results of the automatic segmentation model have not been verified on samples of the dataset used. Therefore, it is possible that some of the extracted patches do not contain nodules. Additionally, unlike other models described in Section 3.4, our model does not have access to the tumor masks because we do not want to rely on potentially suboptimal segmentation results. Furthermore, all sequences were automatically registered based on the entire lung volumes, while a manual registration or a second registration of the extracted subvolumes could have potentially improved the quality of registrations. Another important limitation is that we did not include treatment data, although surgery, chemotherapy, immunotherapy, etc., are likely to have a large impact on tumor progression.

Limitations associated with the proposed model architecture. Focusing on the proposed model, it should be noted that it was not feasible to perform extensive hyperparameter tuning, although some hyperparameter changes could have considerable effects on model performance. Furthermore, the choice of the general model architecture can be questioned. We decided to use a rather simple architecture with well-known components to solve the task at hand. Although this architecture performs reasonably well on video forecasting tasks, the discrete nature of the latent space may not be optimal for handling irregularly sampled sequences that do not contain closely spaced images. The discretization in the latent space poses challenges in enforcing continuity constraints. In general, the latent space is challenging to interpret and not well suited for explicitly integrating prior medical knowledge, such as constraints on deformations. Consequently, the model must learn all forecasting rules entirely from the data, without leveraging prior information.

Limitations of the evaluations. Lastly, we focus on limitations of the evaluation process. As previously explained, an unbiased test set is not employed in the evaluation process. The justification for this approach is based on the poor results — they would most likely not improve on the test set. Consequently, we have opted to reserve this unbiased set for future research and utilize it once the model's performance has been enhanced. In addition to presenting quantitative results, we also offer qualitative visualizations. It is to be noted that these visualizations can only show a small part of the truth. The visualizations have not been extracted randomly, but rather, they have been selected based on image sequences that a person with no medical background would consider comprehensible. Furthermore, it should be noted that the visualizations are limited in scope, as they depict a single 2D slice, which may obscure crucial insights.

The discussions on the model failure remain rather speculative. To investigate the phenomenon of model failure further, it would be advisable to include additional experiments. For instance, a potential approach would involve substituting the existing dataset

with a more elementary synthetic dataset, characterized by deterministic and predictable alterations over time. This would assist in exploring whether the model failure is due to the model architecture or the data. Such research will be conducted in future work.

8. Conclusion and Outlook

8.1. Conclusion

This thesis attempted to solve the task of forecasting a future 3D CT lung image from a given time sequence of past CT images, where the images are acquired at arbitrary time points, based on clinical necessity. Predicting the future progression of lung nodules would potentially allow for earlier, more accurate diagnoses, more informed treatment decisions, and less exposure to radiation.
To enable the forecasting of nodule progression, we proposed a two-step model: A VQ-GAN encodes individual CT subvolumes to a lower-dimensional latent space. The latent space representations are enriched by time and position information using a triplet embedding. The embedded latent patches are passed to a forecasting model together with information on the target time point. We experimented with two different architectures for the forecasting model, namely a transformer and a Mamba model. Several loss function adaptations have been proposed to allow the models to predict the future more accurately and to enable the models to learn temporal continuity.
However, the experiments show that the proposed model cannot accurately predict future CT images. While the VQ-GAN manages to reconstruct images from a low-dimensional latent space representation reasonably well, the latent space forecasting model is not able to accurately infer future development. It is not completely clear whether this failure is caused by the data, the model architecture, or the inherent complexity of the task.

8.2. Outlook

The future goal is to enhance the model's performance through either thorough data cleaning or further improvements on the model architecture. To investigate whether the data or the model choice is the main impeding factor, it would be interesting to apply the model to a homogeneous dataset, for which it is ensured that progressions are predictable, such as a synthetically created dataset. Furthermore, it would be insightful to apply the model to a dataset that has been used for other similar models, such as the National Lung Screening Trial (NLST) dataset [102] used in some models ([11], [7]) detailed in Section 3.4. Possible improvements are, among others, the shift from non-autoregressive to autoregressive training or the inclusion of an iterative refinement scheme, additional hyperparameter tuning, and advanced data augmentations.
As soon as the model performs reasonably well in forecasting lung image sequences, additional modalities will be included, such as treatment data and demographic data of the patients.

Bibliography

[1] National Cancer Institute, SEER Program, *Cancer stat facts: Lung and bronchus cancer*, https://seer.cancer.gov/statfacts/html/lungb.html, Accessed: 2024-11-29, 2024.

[2] R. L. Siegel, A. N. Giaquinto and A. Jemal, 'Cancer statistics, 2024', *CA: A Cancer Journal for Clinicians*, vol. 74, no. 1, pp. 12–49, 2024. DOI: https://doi.org/10.3322/caac.21820. eprint: https://acsjournals.onlinelibrary.wiley.com/doi/pdf/10.3322/caac.21820. [Online]. Available: https://acsjournals.onlinelibrary.wiley.com/doi/abs/10.3322/caac.21820.

[3] S. J. Adams, E. Stone, D. R. Baldwin, R. Vliegenthart, P. Lee and F. J. Fintelmann, 'Lung cancer screening', *The Lancet*, vol. 401, no. 10374, pp. 390–408, 2023, ISSN: 0140-6736. DOI: https://doi.org/10.1016/S0140-6736(22)01694-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140673622016944.

[4] J. Brodersen, T. Voss, F. Martiny, V. Siersma, A. Barratt and B. Heleno, 'Overdiagnosis of lung cancer with low-dose computed tomography screening: Meta-analysis of the randomised clinical trials', *Breathe*, vol. 16, no. 1, 2020. DOI: 10.1183/20734735.0013-2020. eprint: https://publications.ersnet.org//content/breathe/16/1/200013.full.pdf. [Online]. Available: https://publications.ersnet.org//content/breathe/16/1/200013.

[5] Q. Guo, L. Liu, Z. Chen *et al.*, 'Current treatments for non-small cell lung cancer', *Frontiers in Oncology*, vol. 12, p. 945 102, 2022. DOI: 10.3389/fonc.2022.945102. [Online]. Available: https://doi.org/10.3389/fonc.2022.945102.

[6] D. E. Wood, E. A. Kazerooni, S. L. Baum *et al.*, 'Lung cancer screening, version 3.2018, NCCN clinical practice guidelines in oncology', *Journal of the National Comprehensive Cancer Network J Natl Compr Canc Netw*, vol. 16, no. 4, pp. 412–441, 2018. DOI: 10.6004/jnccn.2018.0020. [Online]. Available: https://jnccn.org/view/journals/jnccn/16/4/article-p412.xml.

[7] Y. Wang, C. Zhou, L. Ying *et al.*, 'Enhancing early lung cancer diagnosis: Predicting lung nodule progression in follow-up low-dose ct scan with deep generative model', *Cancers*, vol. 16, no. 12, 2024, ISSN: 2072-6694. DOI: 10.3390/cancers16122229. [Online]. Available: https://www.mdpi.com/2072-6694/16/12/2229.

[8] J. Sheng, Y. Li, G. Cao and K. Hou, 'Modeling nodule growth via spatial transformation for follow-up prediction and diagnosis', in *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–7. DOI: 10.1109/IJCNN52387.2021.9534163.

[9] L. Zhang, L. Lu, X. Wang *et al.*, 'Spatio-temporal convolutional lstms for tumor growth prediction by learning 4d longitudinal patient data', *IEEE Transactions on Medical Imaging*, vol. 39, no. 4, pp. 1114–1126, 2020. DOI: 10.1109/TMI.2019.2943841. [Online]. Available: https://doi.org/10.1109/TMI.2019.2943841.

[10] Y. Li, J. Yang, Y. Xu *et al.*, 'Learning tumor growth via follow-up volume prediction for lung nodules', in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2020*, A. L. Martel, P. Abolmaesumi, D. Stoyanov *et al.*, Eds., Cham: Springer International Publishing, 2020, pp. 508–517, ISBN: 978-3-030-59725-2.

[11] J. Fang, J. Wang, A. Li *et al.*, 'Parameterized Gompertz-guided morphological autoencoder for predicting pulmonary nodule growth', *IEEE Transactions on Medical Imaging*, vol. 42, no. 12, pp. 3602–3613, 2023. DOI: 10.1109/TMI.2023.3297209.

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza *et al.*, 'Generative adversarial networks', *Commun. ACM*, vol. 63, no. 11, pp. 139–144, Oct. 2020, ISSN: 0001-0782. DOI: 10.1145/3422622. [Online]. Available: https://doi.org/10.1145/3422622.

[13] D. P. Kingma and M. Welling, 'Auto-encoding variational bayes', in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014. [Online]. Available: http://arxiv.org/abs/1312.6114.

[14] A. van den Oord, O. Vinyals and K. Kavukcuoglu, 'Neural discrete representation learning', in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6309–6318, ISBN: 9781510860964.

[15] P. Esser, R. Rombach and B. Ommer, 'Taming transformers for high-resolution image synthesis', in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2021, pp. 12 868–12 878. DOI: 10.1109/CVPR46437.2021.01268. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.01268.

[16] A. Vaswani, N. Shazeer, N. Parmar *et al.*, 'Attention is all you need', in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[17] A. Gu and T. Dao, 'Mamba: Linear-time sequence modeling with selective state spaces', *arXiv preprint arXiv:2312.00752*, 2023.

[18] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[19] W. S. McCulloch and W. Pitts, 'A logical calculus of the ideas immanent in nervous activity', *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943. DOI: 10.1007/BF02478259.

[20] F. Rosenblatt, 'The perceptron: A perceiving and recognizing automaton', Project PARA, Cornell Aeronautical Laboratory, Report, Jan. 1957.

[21] D. E. Rumelhart, G. E. Hinton and R. J. Williams, 'Learning internal representations by error propagation', in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362, ISBN: 026268053X.

[22] K. P. Murphy, *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. [Online]. Available: http://probml.github.io/book1.

[23] S. Hochreiter, 'The vanishing gradient problem during learning recurrent neural nets and problem solutions', *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 6, no. 2, pp. 107–116, Apr. 1998, ISSN: 0218-4885. DOI: 10.1142/S0218488598000094. [Online]. Available: https://doi.org/10.1142/S0218488598000094.

[24] V. Nair and G. E. Hinton, 'Rectified linear units improve restricted boltzmann machines', in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10, Haifa, Israel: Omnipress, 2010, pp. 807–814, ISBN: 9781605589077.

[25] S. Elfwing, E. Uchibe and K. Doya, 'Sigmoid-weighted linear units for neural network function approximation in reinforcement learning', *Neural Networks*, vol. 107, pp. 3–11, 2018, Special issue on deep reinforcement learning, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2017.12.012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608017302976.

[26] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[27] J. Bridle, 'Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters', in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2, Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1989/file/0336dcbab05b9d5ad24f4333c7658a0e-Paper.pdf.

[28] D. Foster, *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*. O'Reilly Media, 2019, ISBN: 9781492041917.

[29] J. Terven, D.-M. Cordova-Esparza, J.-A. Romero-González, A. Ramírez-Pedraza and E. A. Chávez-Urbiola, 'A comprehensive survey of loss functions and metrics in deep learning', *Artificial Intelligence Review*, vol. 58, no. 7, p. 195, 2025, ISSN: 1573-7462. DOI: 10.1007/s10462-025-11198-7. [Online]. Available: https://doi.org/10.1007/s10462-025-11198-7.

[30] H. Robbins and S. Monro, 'A Stochastic Approximation Method', *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951. DOI: 10.1214/aoms/1177729586. [Online]. Available: https://doi.org/10.1214/aoms/1177729586.

[31] J. Kiefer and J. Wolfowitz, 'Stochastic Estimation of the Maximum of a Regression Function', *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952. DOI: 10.1214/aoms/1177729392. [Online]. Available: https://doi.org/10.1214/aoms/1177729392.

[32] D. P. Kingma and J. Ba, 'Adam: A method for stochastic optimization', *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6628106.

[33] N. Morgan and H. Bourlard, 'Generalization and parameter estimation in feedforward nets: Some experiments', in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2, Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1989/file/63923f49e5241343aa7acb6a06a751e7-Paper.pdf.

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, 'Dropout: A simple way to prevent neural networks from overfitting', *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[35] Y. LeCun, B. Boser, J. Denker *et al.*, 'Handwritten digit recognition with a back-propagation network', in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2, Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

[36] K. He, X. Zhang, S. Ren and J. Sun, 'Deep residual learning for image recognition', in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[37] T. Brown, B. Mann, N. Ryder *et al.*, 'Language models are few-shot learners', in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[38] A. Ramesh, M. Pavlov, G. Goh *et al.*, 'Zero-shot text-to-image generation', in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 18–24 Jul 2021, pp. 8821–8831. [Online]. Available: https://proceedings.mlr.press/v139/ramesh21a.html.

[39] K. P. Murphy, *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023. [Online]. Available: http://probml.github.io/book2.

[40] J. F. Nash, 'Equilibrium points in n-person games', *Proceedings of the National Academy of Sciences*, vol. 36, no. 1, pp. 48–49, 1950. DOI: 10.1073/pnas.36.1.48. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.36.1.48. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.36.1.48.

[41] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang and S. P. Smolley, 'Least Squares Generative Adversarial Networks', in *2017 IEEE International Conference on Computer Vision (ICCV)*, Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2017, pp. 2813–2821. DOI: 10.1109/ICCV.2017.304. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCV.2017.304.

[42] P. Isola, J.-Y. Zhu, T. Zhou and A. A. Efros, 'Image-to-image translation with conditional adversarial networks', in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5967–5976. DOI: 10.1109/CVPR.2017.632.

[43] Y. Lecun, *PhD thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models)*, English (US). Universite P. et M. Curie (Paris 6), Jun. 1987.

[44] S. Kullback and R. A. Leibler, 'On Information and Sufficiency', *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951. DOI: 10.1214/aoms/1177729694. [Online]. Available: https://doi.org/10.1214/aoms/1177729694.

[45] X. Chen, D. P. Kingma, T. Salimans *et al.*, 'Variational lossy autoencoder', in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=BysvGP5ee.

[46] Y. Bengio, N. Léonard and A. Courville, 'Estimating or propagating gradients through stochastic neurons for conditional computation', *arXiv preprint arXiv:1308.3432*, 2013.

[47] R. Gray, 'Vector quantization', *IEEE ASSP Magazine*, vol. 1, no. 2, pp. 4–29, 1984. DOI: 10.1109/MASSP.1984.1162229.

[48] A. Van Den Oord, N. Kalchbrenner and K. Kavukcuoglu, 'Pixel recurrent neural networks', in *International conference on machine learning*, PMLR, 2016, pp. 1747–1756.

[49] I. Sutskever, O. Vinyals and Q. V. Le, 'Sequence to sequence learning with neural networks', in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence and K. Weinberger, Eds., vol. 27, Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/5a18e133cbf9f257297f410bb7eca942-Paper.pdf.

[50] A. Graves, A.-r. Mohamed and G. Hinton, 'Speech recognition with deep recurrent neural networks', in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6645–6649. DOI: 10.1109/ICASSP.2013.6638947.

[51] J. D. M.-W. C. Kenton and L. K. Toutanova, 'Bert: Pre-training of deep bidirectional transformers for language understanding', in *Proceedings of naacL-HLT*, Minneapolis, Minnesota, vol. 1, 2019.

[52] J. L. Ba, J. R. Kiros and G. E. Hinton, 'Layer normalization', *arXiv preprint arXiv:1607.06450*, 2016.

[53] R. E. Kalman, 'A new approach to linear filtering and prediction problems', *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, Mar. 1960, ISSN: 0021-9223. DOI: 10.1115/1.3662552. eprint: https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/82/1/35/5518977/35_1.pdf. [Online]. Available: https://doi.org/10.1115/1.3662552.

[54] A. Gu, K. Goel and C. Ré, 'Efficiently modeling long sequences with structured state spaces', in *The International Conference on Learning Representations (ICLR)*, 2022.

[55] J. T. Smith, A. Warrington and S. Linderman, 'Simplified state space layers for sequence modeling', in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=Ai8Hw3AXqks.

[56] D. Y. Fu, T. Dao, K. K. Saab, A. W. Thomas, A. Rudra and C. Re, 'Hungry hungry hippos: Towards language modeling with state space models', in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=COZDy0WYGg.

[57] G. E. Blelloch, 'Scans as primitive parallel operations', *IEEE Transactions on computers*, vol. 38, no. 11, pp. 1526–1538, 1989.

[58] T. Wolf, L. Debut, V. Sanh *et al.*, 'Transformers: State-of-the-art natural language processing', in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[59] T. P. Van, T. M. Nguyen, N. N. Tran *et al.*, 'Interpreting the Latent Space of Generative Adversarial Networks using Supervised Learning', in *2020 International Conference on Advanced Computing and Applications (ACOMP)*, Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2020, pp. 49–54. DOI: 10.1109/ACOMP50827.2020.00015. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ACOMP50827.2020.00015.

[60] A. Dosovitskiy, L. Beyer, A. Kolesnikov *et al.*, 'An image is worth 16x16 words: Transformers for image recognition at scale', in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy.

[61] Y. Wang, H. Wu, J. Dong, Y. Liu, M. Long and J. Wang, 'Deep time series models: A comprehensive survey and benchmark', *arXiv preprint arXiv:2407.13278*, 2024.

[62] H. Wu, J. Xu, J. Wang and M. Long, 'Autoformer: Decomposition transformers with Auto-Correlation for long-term series forecasting', in *Advances in Neural Information Processing Systems*, 2021.

[63] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun and R. Jin, 'FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting', in *Proc. 39th International Conference on Machine Learning (ICML 2022)*, Baltimore, Maryland, 2022.

[64] Y. Zhang and J. Yan, 'Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting', in *International Conference on Learning Representations*, 2023.

[65] Y. Nie, N. H. Nguyen, P. Sinthong and J. Kalagnanam, 'A time series is worth 64 words: Long-term forecasting with transformers', in *International Conference on Learning Representations*, 2023.

[66] M. Goswami, K. Szafer, A. Choudhry, Y. Cai, S. Li and A. Dubrawski, 'Moment: A family of open time-series foundation models', in *International Conference on Machine Learning*, 2024.

[67] A. F. Ansari, L. Stella, C. Turkmen *et al.*, 'Chronos: Learning the language of time series', *arXiv preprint arXiv:2403.07815*, 2024.

[68] Y. Liu, T. Hu, H. Zhang *et al.*, 'Itransformer: Inverted transformers are effective for time series forecasting', *arXiv preprint arXiv:2310.06625*, 2023.

[69] T. Dao and A. Gu, 'Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality', in *International Conference on Machine Learning (ICML)*, 2024.

[70] X. Cai, Y. Zhu, X. Wang and Y. Yao, 'Mambats: Improved selective state space models for long-term time series forecasting', *arXiv preprint arXiv:2405.16440*, 2024.

[71] M. A. Ahamed and Q. Cheng, 'Timemachine: A time series is worth 4 mambas for long-term forecasting', *arXiv preprint arXiv:2403.09898*, 2024.

[72] Z. Wang, F. Kong, S. Feng *et al.*, 'Is mamba effective for time series forecasting?', *arXiv preprint arXiv:2403.11144*, 2024.

[73] S. Tipirneni and C. K. Reddy, 'Self-supervised transformer for sparse and irregularly sampled multivariate clinical time-series', *ACM Trans. Knowl. Discov. Data*, vol. 16, no. 6, Jul. 2022, ISSN: 1556-4681. DOI: 10.1145/3516367. [Online]. Available: https://doi.org/10.1145/3516367.

[74] V. Yalavarthi, J. Burchert and L. Schmidt-Thieme, 'Tripletformer for probabilistic interpolation of irregularly sampled time series', in *2023 IEEE International Conference on Big Data (BigData)*, Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2023. DOI: 10.1109/BigData59044.2023.10386325. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/BigData59044.2023.10386325.

[75] Y. Chen, K. Ren, Y. Wang, Y. Fang, W. Sun and D. Li, 'Contiformer: Continuous-time transformer for irregular time series modeling', in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=YJDz4F2AZu.

[76] J. Selva, A. S. Johansen, S. Escalera, K. Nasrollahi, T. B. Moeslund and A. Clapes, 'Video Transformers: A Survey', *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 45, no. 11, pp. 12 922–12 943, Nov. 2023, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2023.3243465. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TPAMI.2023.3243465.

[77] A. Melnik, M. Ljubljanac, C. Lu, Q. Yan, W. Ren and H. Ritter, 'Video diffusion models: A survey', *arXiv preprint arXiv:2405.03150*, 2024.

[78] W. Yan, Y. Zhang, P. Abbeel and A. Srinivas, 'Videogpt: Video generation using vq-vae and transformers', *arXiv preprint arXiv:2104.10157*, 2021.

[79] A. Gupta, S. Tian, Y. Zhang, J. Wu, R. Martín-Martín and L. Fei-Fei, 'Maskvit: Masked visual pre-training for video prediction', in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=QAV2CcLEDh.

[80] L. Yu, Y. Cheng, K. Sohn *et al.*, 'Magvit: Masked generative video transformer', in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 10 459–10 469. DOI: 10.1109/CVPR52729.2023.01008.

[81] J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi and D. J. Fleet, 'Video diffusion models', *arXiv preprint arXiv:2204.03458*, 2022.

[82] D. Zhou, W. Wang, H. Yan, W. Lv, Y. Zhu and J. Feng, 'Magicvideo: Efficient video generation with latent diffusion models', *arXiv preprint arXiv:2211.11018*, 2022.

[83] X. Wang, H. Yuan, S. Zhang *et al.*, 'Videocomposer: Compositional video synthesis with motion controllability', *arXiv preprint arXiv:2306.02018*, 2023.

[84] S. Hochreiter and J. Schmidhuber, 'Long short-term memory', *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735.

[85] J. S. Yoon, C. Zhang, H.-I. Suk, J. Guo and X. Li, 'Sadm: Sequence-aware diffusion model for longitudinal medical image generation', in *Information Processing in Medical Imaging*, 2023.

[86] J. Petersen, F. Isensee, G. Köhler *et al.*, 'Continuous-time deep glioma growth models', in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2021*, M. de Bruijne, P. C. Cattin, S. Cotin *et al.*, Eds., Cham: Springer International Publishing, 2021, pp. 83–92.

[87] H. Bai and Y. Hong, 'NODER: Image Sequence Regression Based on Neural Ordinary Differential Equations', in *proceedings of Medical Image Computing and Computer Assisted Intervention – MICCAI 2024*, vol. LNCS 15002, Springer Nature Switzerland, Oct. 2024.

[88] W. H. Organization, *International statistical classification of diseases and related health problems (icd-10), 2019 edition*, Accessed: 2024-12-04, 2019. [Online]. Available: https://icd.who.int/browse10/2019/en#.

[89] S. Schaller, J. Wildberger, R. Raupach, M. Niethammer, K. Klingenbeck-Regn and T. Flohr, 'Spatial domain filtering for fast modification of the tradeoff between image sharpness and pixel noise in computed tomography', *IEEE Transactions on Medical Imaging*, vol. 22, no. 7, pp. 846–853, 2003. DOI: 10.1109/TMI.2003.815073.

[90] J. T. Bushberg, J. A. Seiberg, E. M. Leidholdt Jr. and J. M. Boone, *The Essential Physics of Medical Imaging*. Philadelphia: Wolters Kluwer Health, 1st Jan. 2002, ISBN: 978-1-9751-0322-4. [Online]. Available: https://research.ebsco.com/linkprocessor/plink?id=a88dfbd0-96f7-3ef4-a604-224068b227b0.

[91] J. Hofmanninger, F. Prayer, J. Pan, S. Röhrich, H. Prosch and G. Langs, 'Automatic lung segmentation in routine imaging is primarily a data diversity problem, not a methodology problem', *European Radiology Experimental*, vol. 4, no. 1, Aug. 2020, ISSN: 2509-9280. DOI: 10.1186/s41747-020-00173-2. [Online]. Available: http://dx.doi.org/10.1186/s41747-020-00173-2.

[92] F. Isensee, P. F. Jaeger, S. A. Kohl, J. Petersen and K. H. Maier-Hein, 'nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation', *Nature methods*, vol. 18, no. 2, pp. 203–211, 2021.

[93] M. Antonelli, A. Reinke, S. Bakas *et al.*, 'The medical segmentation decathlon', *Nature communications*, vol. 13, no. 1, p. 4128, 2022.

[94] A. Madabhushi and M. Rusu, 'Fused radiology-pathology lung dataset', *The Cancer Imaging Archive*, 2018.

[95] R. S. Vanguri, J. Luo, A. T. Aukerman *et al.*, 'Multimodal integration of radiology, pathology and genomics for prediction of response to PD-(L)1 blockade in patients with non-small cell lung cancer', *Nature cancer*, vol. 3, no. 10, pp. 1151–1164, 2022.

[96] B. Zhao, L. H. Schwartz, M. G. Kris and G. J. Riely, 'Coffee-break lung ct collection with scan images reconstructed at multiple imaging parameters', 2015, Dataset (Version 3). DOI: 10.7937/k9/tcia.2015.u1x8a5nr.

[97] W. H. L. Pinaya, M. S. Graham, E. Kerfoot *et al.*, 'Generative AI for medical imaging: Extending the MONAI framework', *arXiv preprint arXiv:2307.15208*, 2023.

[98] R. Xiong, Y. Yang, D. He *et al.*, 'On layer normalization in the transformer architecture', in *Proceedings of the 37th International Conference on Machine Learning*, ser. ICML'20, JMLR.org, 2020.

[99] H. Coppock, 'Vector Quantised-Variational Autoencoders (VQ-VAEs) for Representation learning', M.S. thesis, Imperial College London, Department of Computing, Sep. 2020.

[100] E. Jang, S. Gu and B. Poole, 'Categorical reparameterization with gumbel-softmax', in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=rkE3y85ee.

[101] Z. Wang, A. Bovik, H. Sheikh and E. Simoncelli, 'Image quality assessment: From error visibility to structural similarity', *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. DOI: 10.1109/TIP.2003.819861.

[102] National Lung Screening Trial Research Team, *Data from the national lung screening trial (NLST)*, Data set, 2013. DOI: 10.7937/TCIA.HMQ8-J677. [Online]. Available: https://doi.org/10.7937/TCIA.HMQ8-J677.

Acronyms

D_{KL} Kullback-Leibler Divergence.

CE Cross Entropy.

CNN Convolutional Neural Network.

CT Computed Tomography.

GAN Generative Adversarial Network.

LSTM Long Short-Term Memory.

MAE Mean Absolute Error.

MLP Multilayer Perceptron.

MONAI Medical Open Network for Artificial Intelligence.

MRT Magnetic Resonance Tomography.

MSE Mean Squared Error.

PCA Principal Component Analysis.

PET Position Emission Tomography.

PSNR Peak Signal-to-Noise Ratio.

ReLU Rectified Linear Unit.

RNN Recurrent Neural Network.

SiLU Sigmoid Linear Unit.

SSIM Structural Similarity Index Measure.

SSM State Space Model.

VAE Variational Autoencoder.

ViT Vision Transformer.

VQ Vector Quantization.

VQ-VAE Vector-Quantized Variational Autoencoder.

VQ-GAN Vector-Quantized Generative Adversarial Network.

A. Appendix

A.1. Additions to Experiment 6.1.3

Experiment 6.1.3 shows latent interpolations in the codebook index and vector space for a manually selected example. To supplement the qualitative analysis with more samples, we include three time series that were randomly drawn from a manually selected subset from *Dataset B* of sequences that show comprehensible nodule progressions.

Example 1.

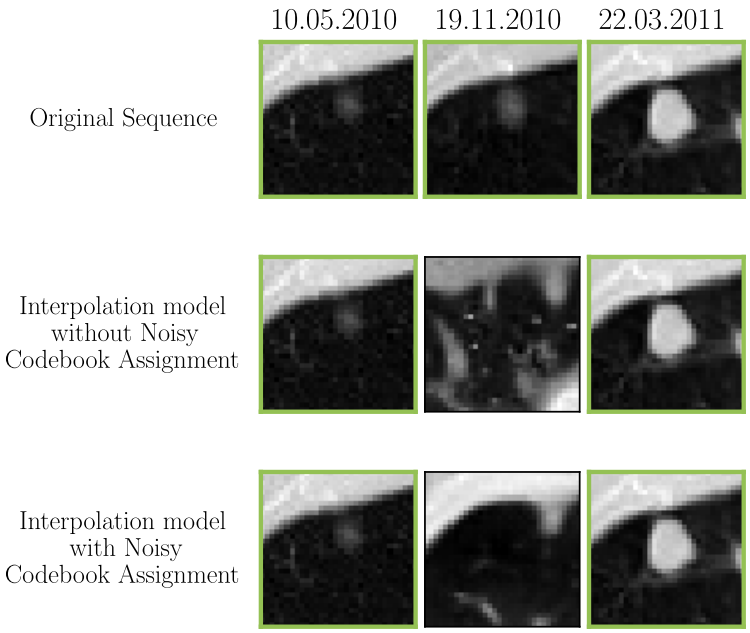


Figure A.1.: **Linear interpolation of codebook indices.** First and last images of the original sequence (first row) are used to linearly interpolate their corresponding codebook indices. The sample was drawn randomly from a manually selected subset in which comprehensible nodule development could be observed.

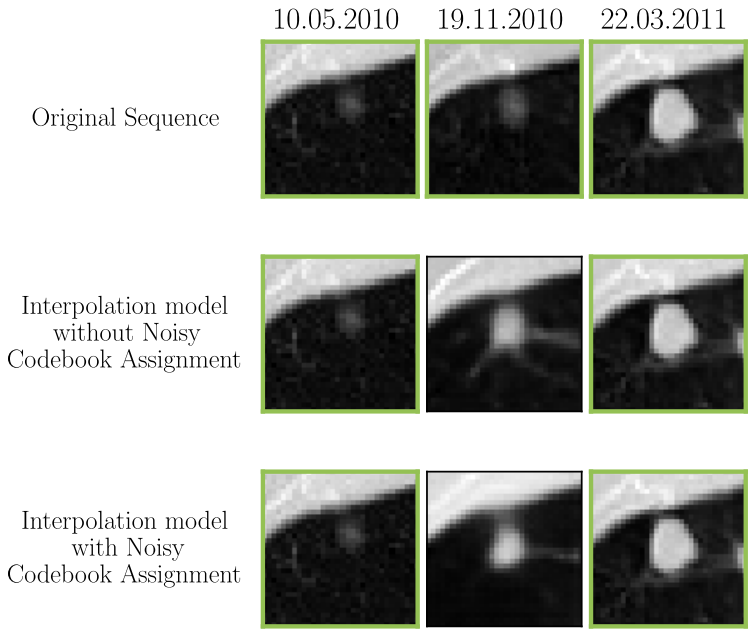


Figure A.2.: **Linear interpolation of codebook vectors.** First and last images of the original sequence (first row) are used to linearly interpolate their corresponding codebook vectors.

Example 2.

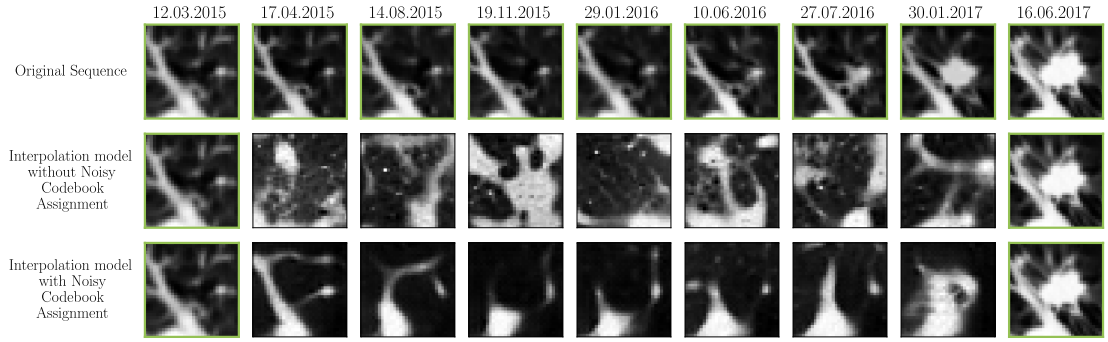


Figure A.3.: **Linear interpolation of codebook indices.** First and last images of the original sequence (first row) are used to linearly interpolate the codebook indices. The sample was drawn randomly from a manually selected subset in which comprehensible nodule development could be observed.

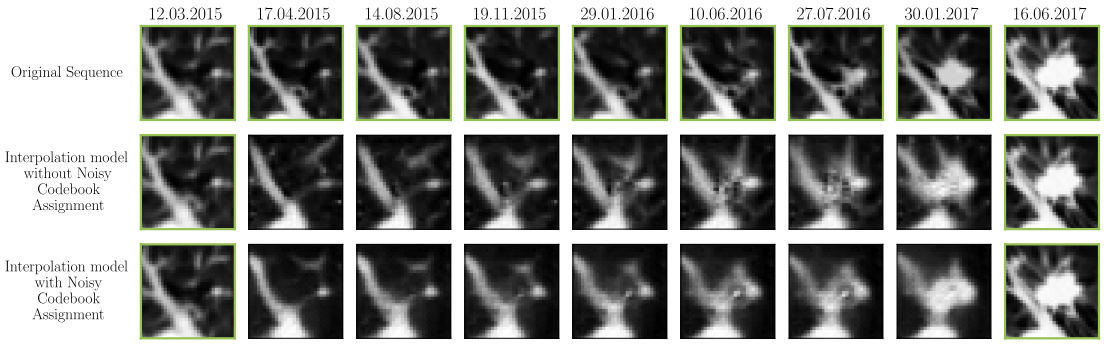


Figure A.4.: **Linear interpolation of codebook vectors.** First and last images of the original sequence (first row) are used to linearly interpolate their corresponding codebook vectors.

Example 3.

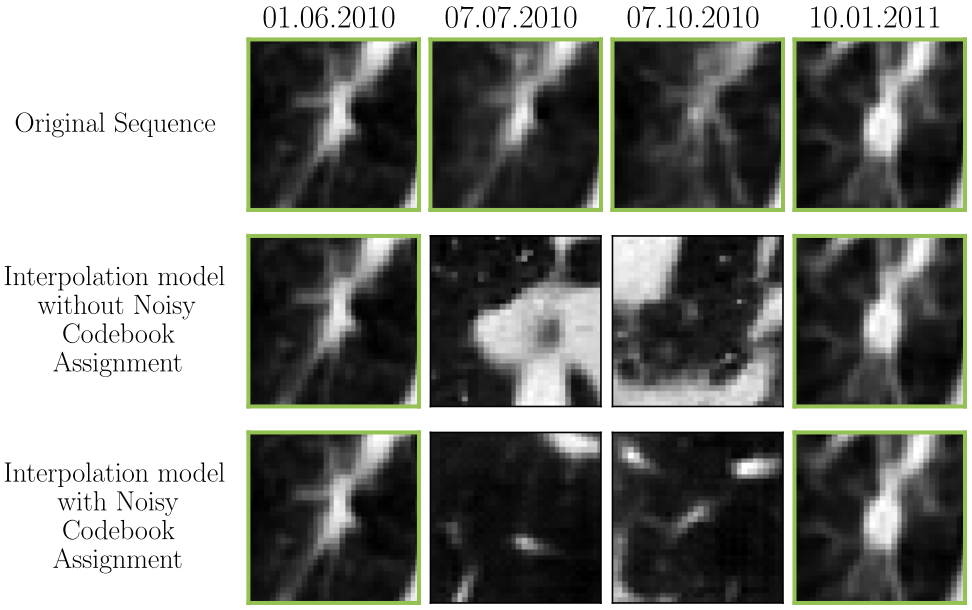


Figure A.5.: **Linear interpolation of codebook indices.** First and last images of the original sequence (first row) are used to linearly interpolate the codebook indices. The sample was drawn randomly from a manually selected subset in which comprehensible nodule development could be observed.

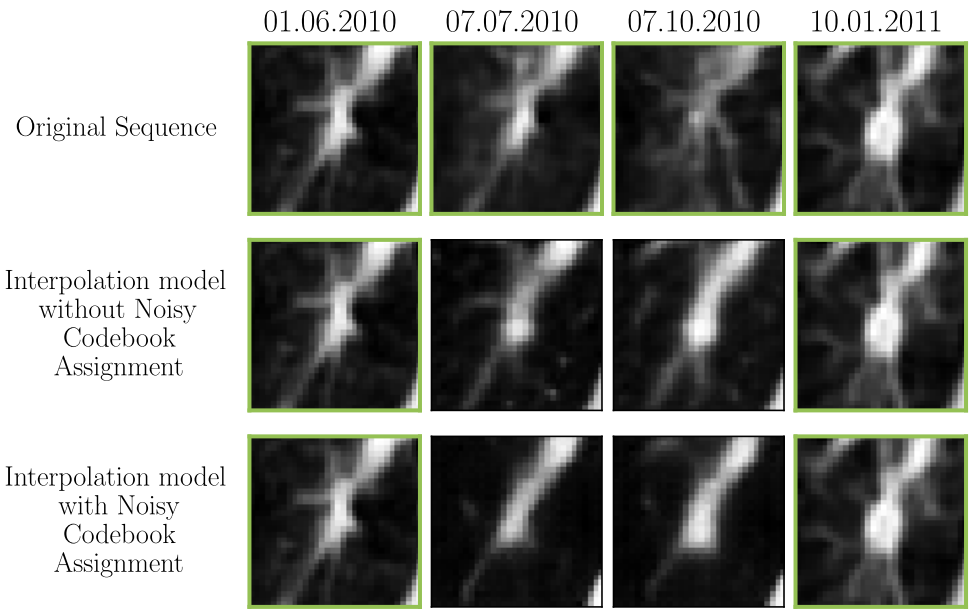


Figure A.6.: **Linear interpolation of codebook vectors.** First and last images of the original sequence (first row) are used to linearly interpolate their corresponding codebook vectors.

A.2. Additions to Experiment 6.3.1

Experiment 6.3.1 shows the qualitative prediction results of the models on a single time series. We complement this example with three more examples shown in this section. The sequences were randomly drawn from a manually selected subset of the validation set of *Dataset B*, in which the nodules show comprehensible trajectories.

Example 1.

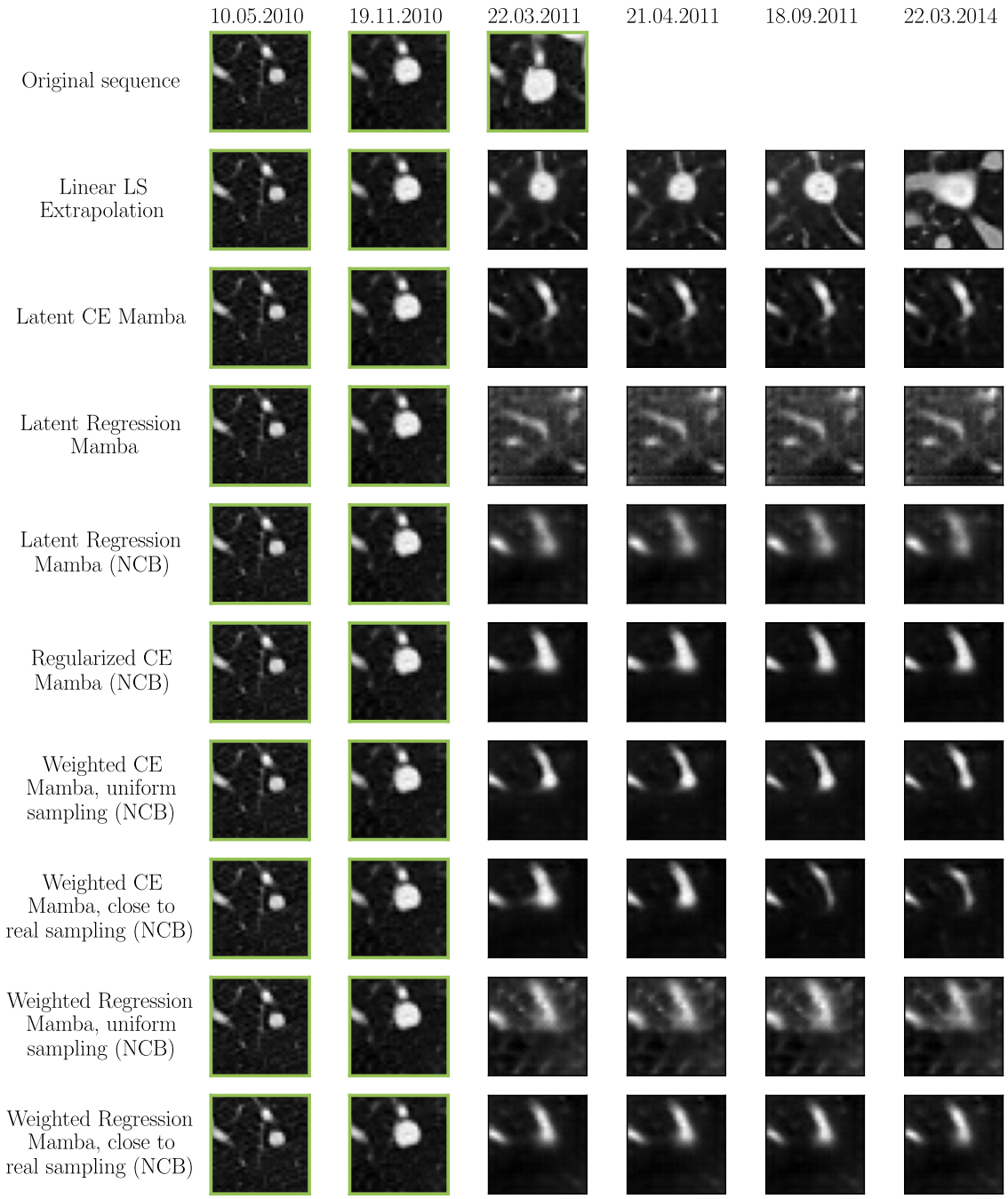


Figure A.7.: **Extrapolation results for models with different loss functions for a random example.** Each model received the first two thirds of the sequence as input.

Example 2.

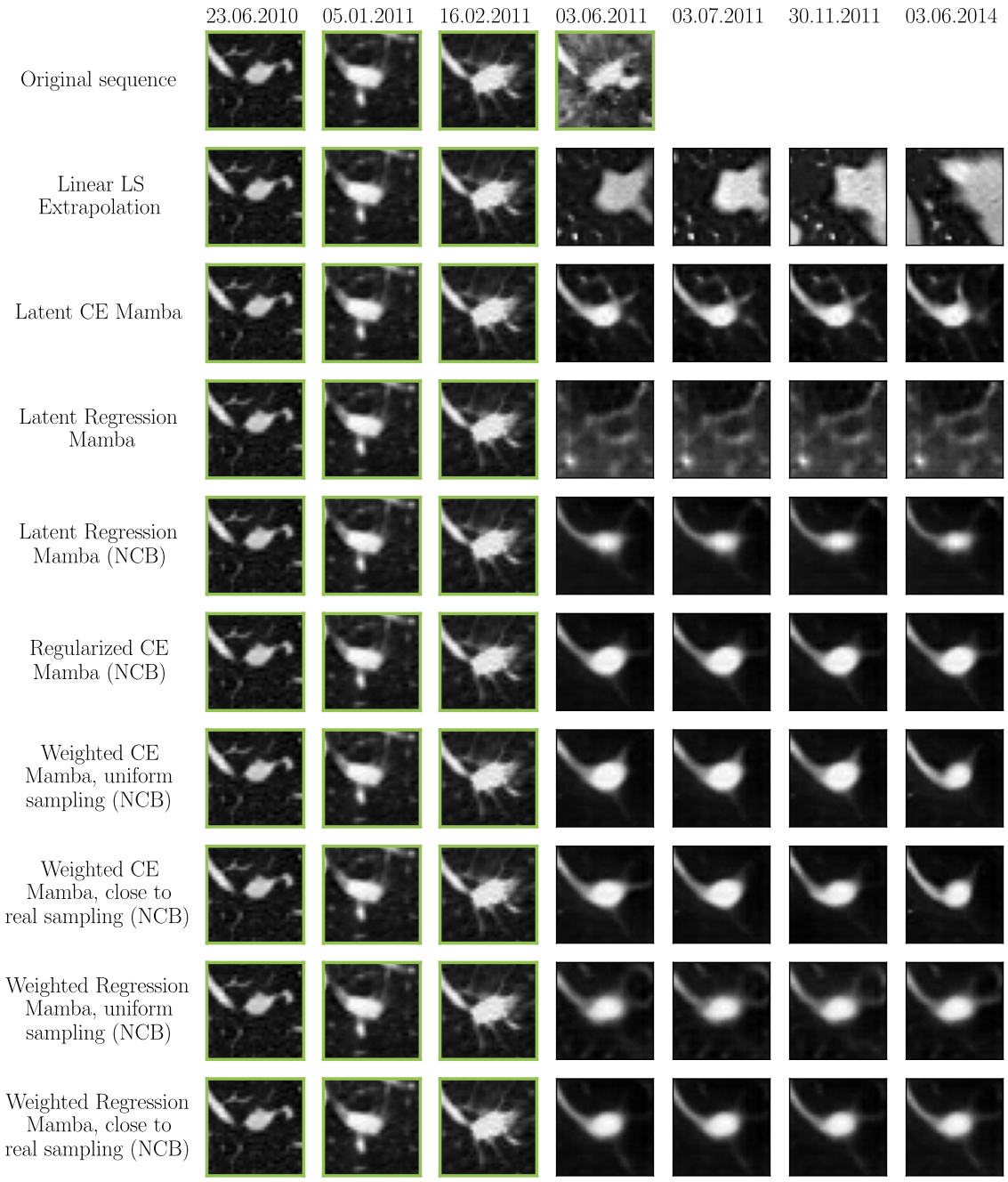


Figure A.8.: **Extrapolation results for models with different loss functions for a random example.** Each model received the first two thirds of the sequence as input.

Example 3.

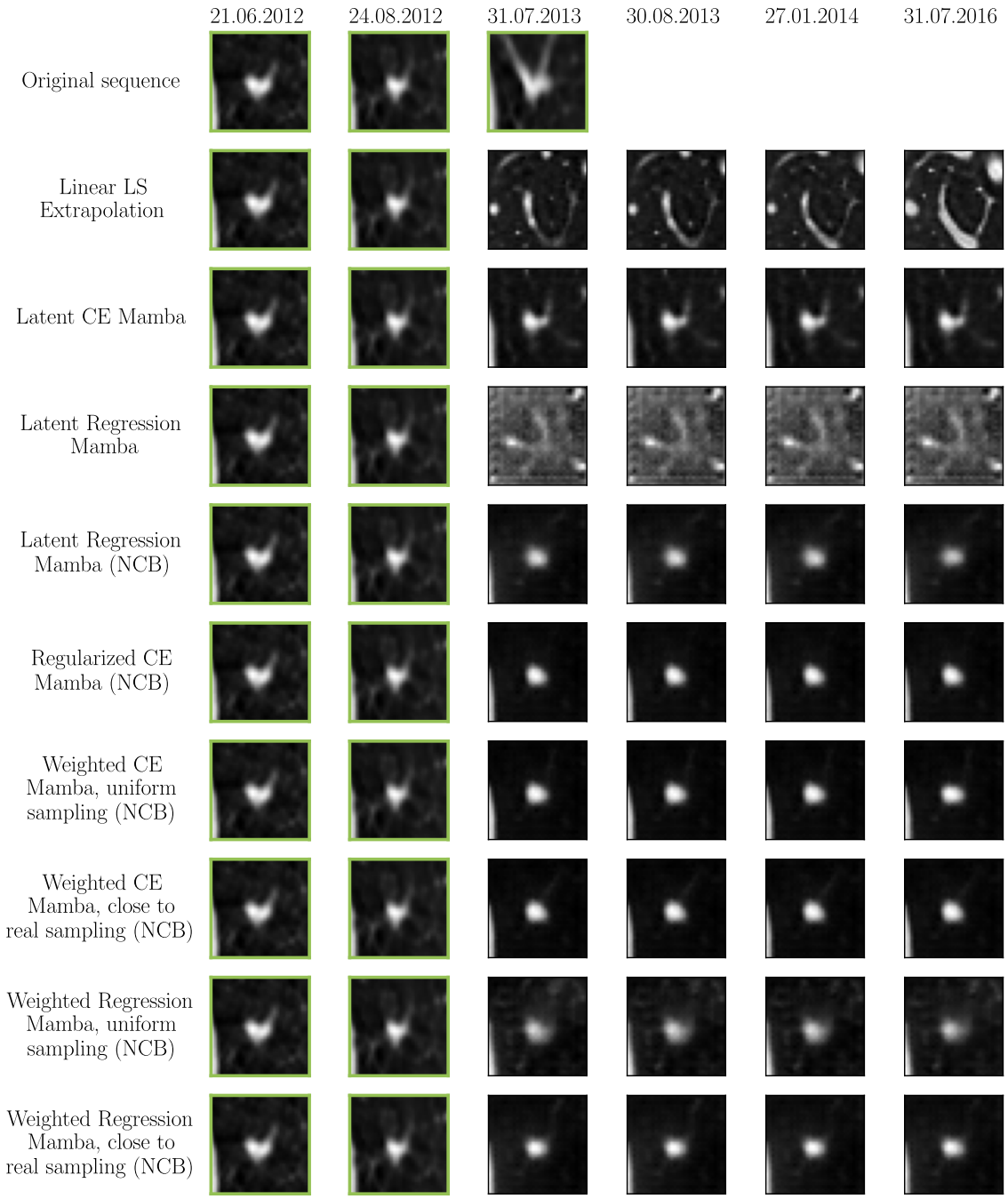


Figure A.9.: **Extrapolation results for models with different loss functions for a random example.** Each model received the first two thirds of the sequence as input.

A.3. Additions to Experiment 6.3.2

In Experiment 6.3.2, a visualization of an interpolation sample is given. We add interpolation visualizations for all sequences shown in Appendix A.2.

Example 1.

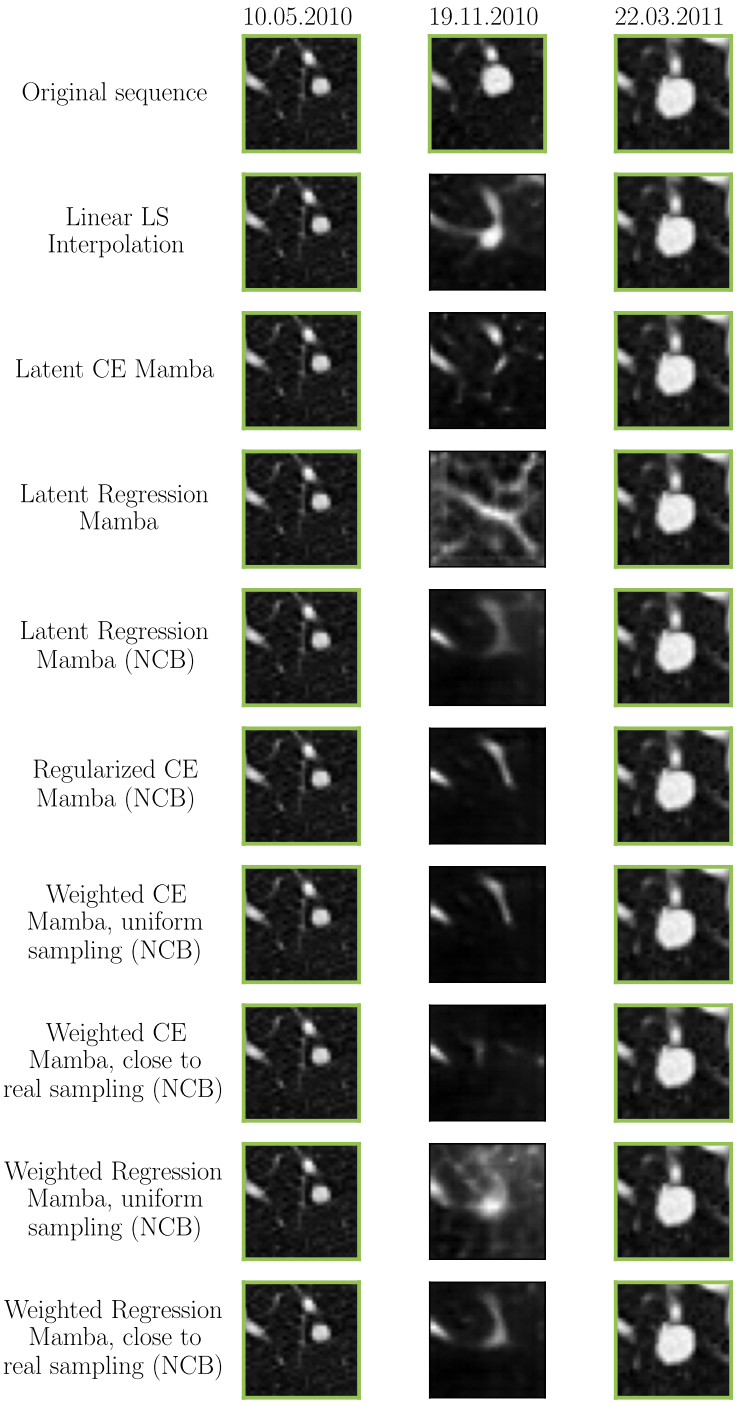


Figure A.10.: **Interpolation results for models with different loss functions for a random example.** Each model received the first and last image of the shown sequence as input.

Example 2.

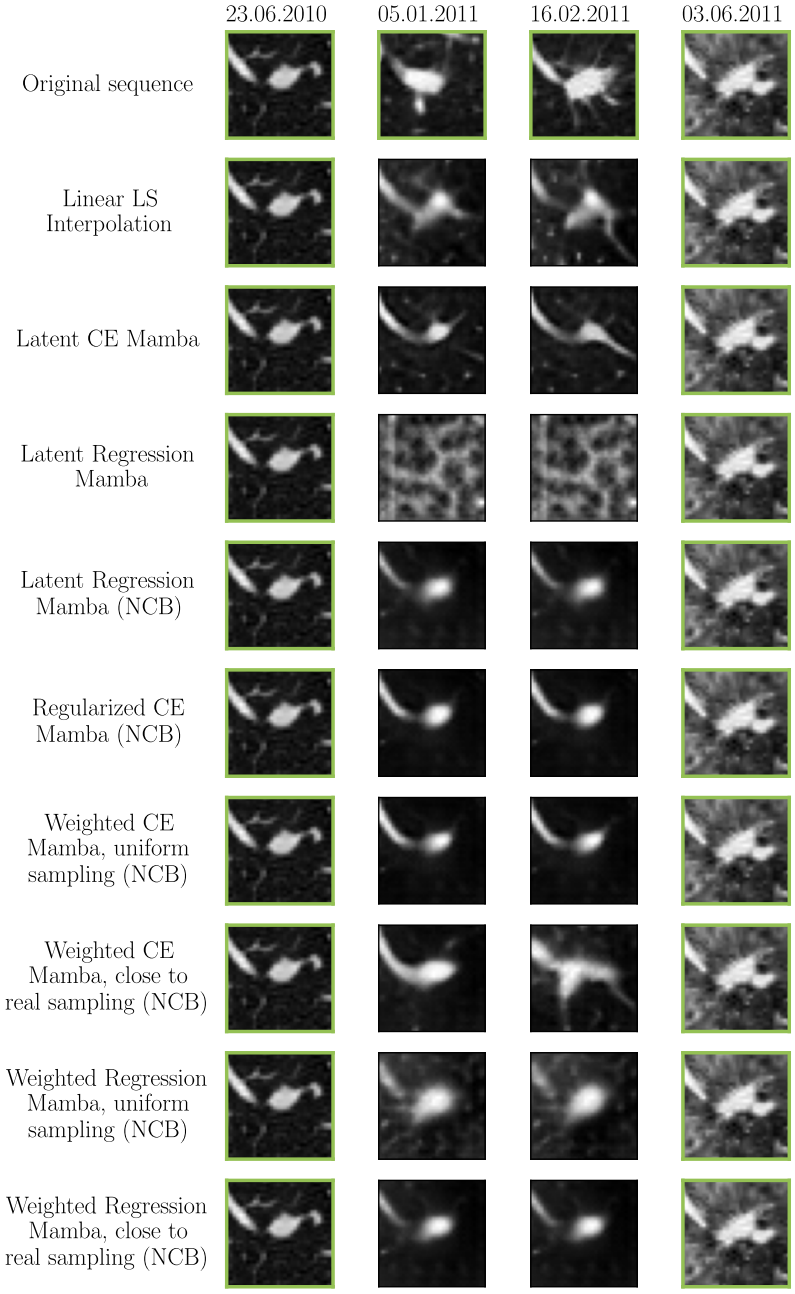


Figure A.11.: **Interpolation results for models with different loss functions for a random example.** Each model received the first and last image of the shown sequence as input.

Example 3.

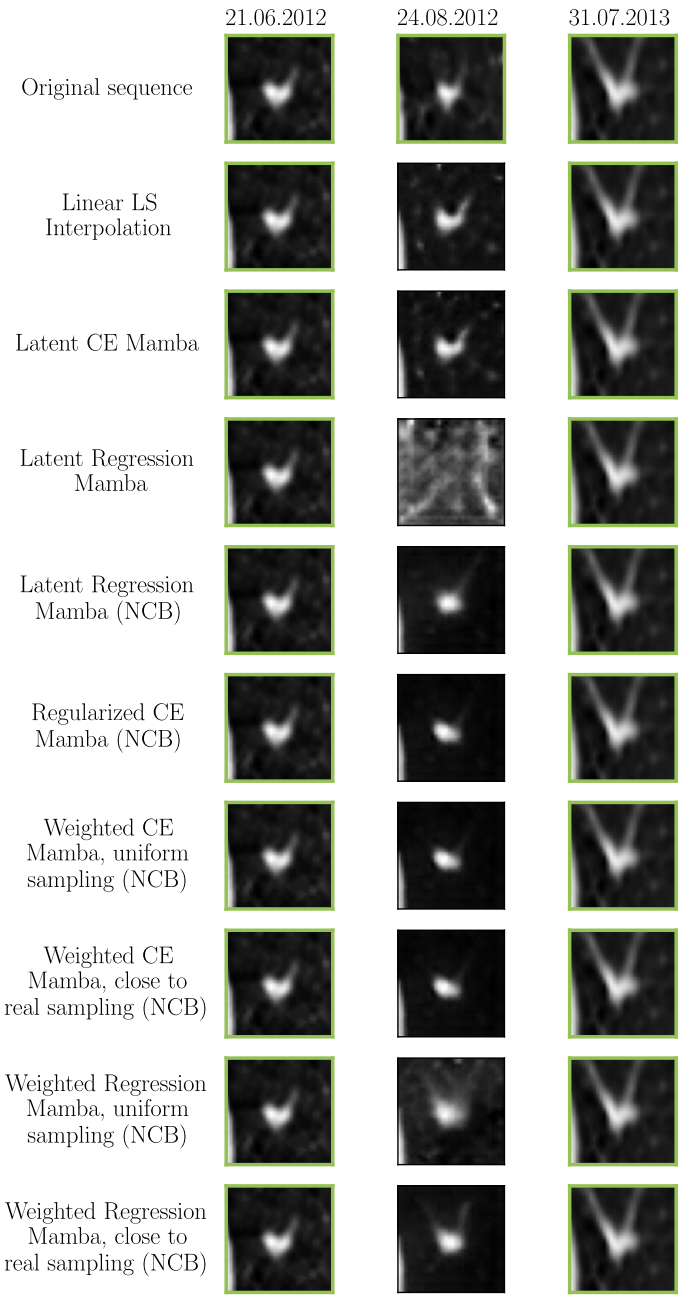


Figure A.12.: **Interpolation results for models with different loss functions for a random example.** Each model received the first and last image of the shown sequence as input.

A.4. Additions to Experiment 6.3.3

The following plots show the corresponding continuity analyses for the interpolation sequences shown in Appendix A.3.

Example 1.

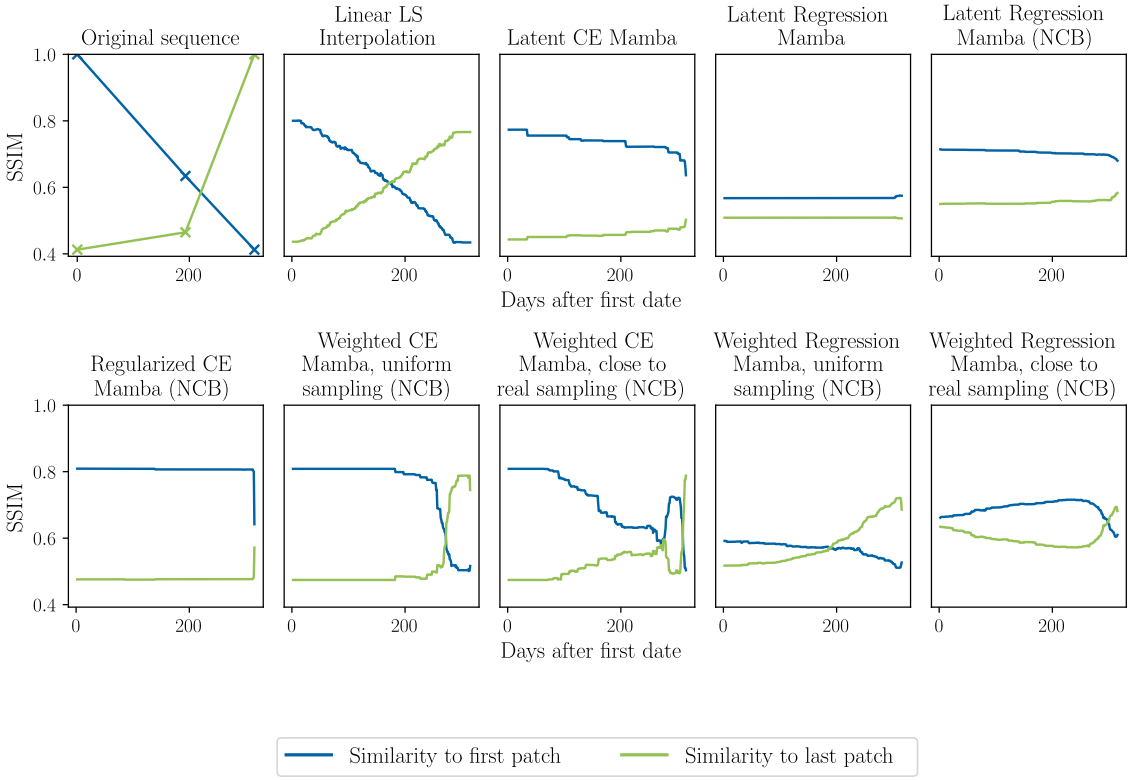


Figure A.13.: **Continuity analysis of interpolations over time.** The plots show the SSIM to the first image (in blue) and to the second image (in green), when interpolating two images. Specifically, the same two images as employed in Figure A.10 are also utilized for this analysis.

Example 2.

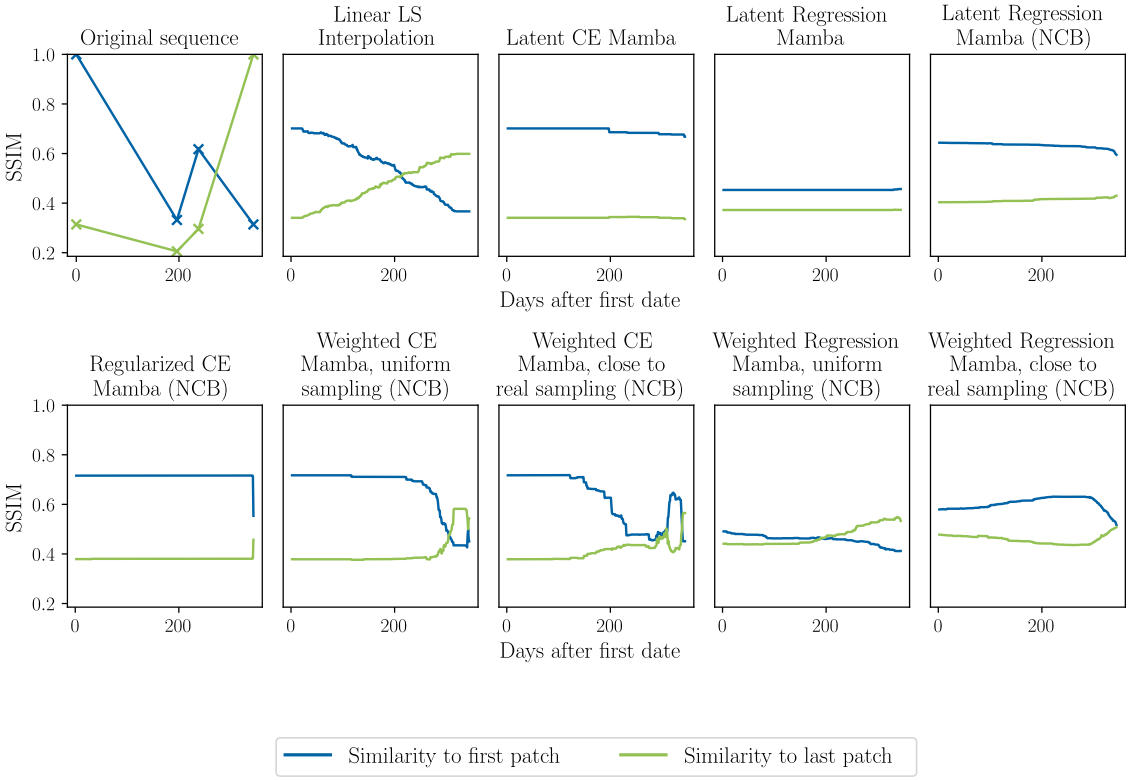


Figure A.14.: **Continuity analysis of interpolations over time.** The plots show the SSIM to the first image (in blue) and to the second image (in green), when interpolating two images. Specifically, the same two images as employed in Figure A.11 are also utilized for this analysis.

Example 3.

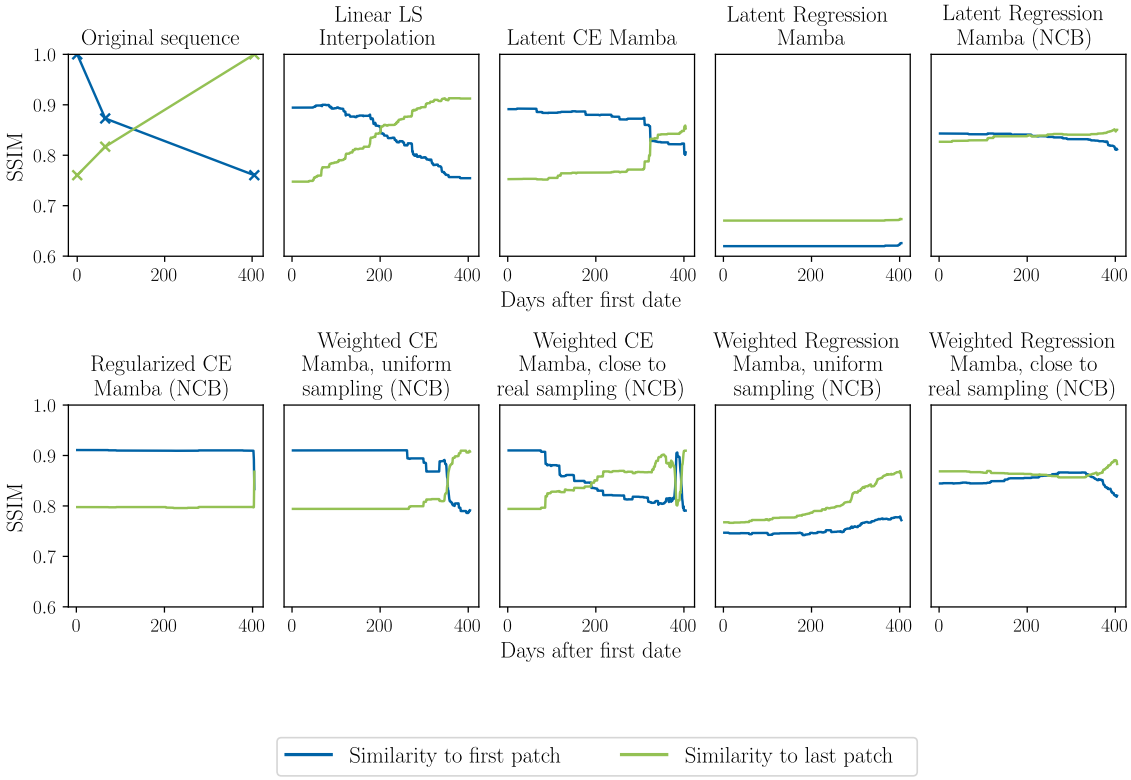


Figure A.15.: **Continuity analysis of interpolations over time.** The plots show the SSIM to the first image (in blue) and to the second image (in green), when interpolating two images. Specifically, the same two images as employed in Figure A.12 are also utilized for this analysis.