Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Image Sequence Analysis with CNNs and LSTMs on Healing MNIST dataset

Lab Rotation at the Department of Quantitative Biomedicine

Melissa June Ensmenger

June 14, 2023

Contents

1 Introduction

In this project, we aim to develop a machine learning model that can effectively analyze longitudinal patient data. The model should be capable of extracting key features from patient data that can be used as low-dimensional patient representations and subsequent similarity analysis, or as meaningful input for a downstream task, such as classification or prediction.

Due to the limited availability of patient data and the complexity of the necessary preprocessing steps [1], we decided to represent each patient by a single image (single timepoint) or by a sequence of images. Both patient data and images are high-dimensional and the temporal aspect of patient data can be captured by expanding a single image into a sequence of images through image rotation [2]. Since this dataset shares some important characteristics of electronic health records, such as temporality, heterogeneity and high dimensionality, we hope to easily transfer our findings to real patient data.

2 Material and Methods

2.1 Description of datasets

For this study, we used and adapted the well-known MNIST handwritten digit dataset, consisting of 60'000 training images and 10'000 test images [3]. Following the example of the 'Healing MNIST' implementation from Krishnan et al., we created a sequence of digits from a single original image by rotating the original image sequentially by either a random or pre-specified angle. Additionally, we sometimes included a square of varying dimensions to the top-left corner of a subset of successive images in the sequence [2], as shown in Figure 1. We extended these modifications by also allowing the square to be dynamically sized over the course of the sequence, i.e. growing and then shrinking in size. This modified dataset should simulate longitudinal patient electronic health records, where patient features are recorded at distinct time-points. The square in the corner represents the presence of a dominant feature (such as the seasonal flu or other ailments) that is treated and disappears again [2].

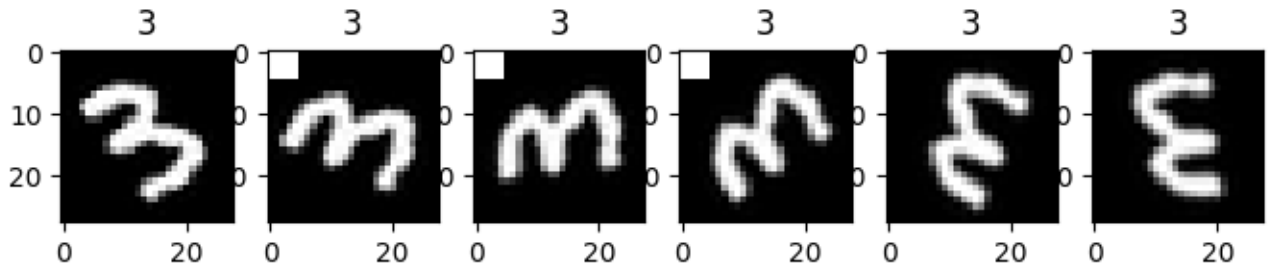


Figure 1: Example sequence of rotated digits with square in top left corner

2.2 Introduction of deep learning models

This section covers the theory behind the models that were used in this project. We focused on two main classes of deep learning models namely convolutional neural networks (CNNs) which

are particularly useful for image feature extraction [4], as well as Long-Short-Term-Memory networks (LSTMs) which are suited for the analysis of temporal/sequential data [5]. We also leveraged the concept of autoencoders to create low-dimensional embeddings of our input data [6].

2.2.1 Convolutional Neural Network

Convolutional neural networks, compared to classical neural networks such as multilayer perceptrons (MLPs), have the advantage of capturing spatial information from images and not treating each pixel independently. This makes them especially suited for deep learning tasks with images as input. One purpose of CNNs can be feature extraction from images generating a meaningful latent embedding that can be used as less complex input for downstream tasks such as classification, reconstruction or prediction [7].
In the following, we shortly discuss the different layer types typically found in CNNs.

Convolutional Layer A convolutional layer is the building block of a convolutional neural network (CNN). It applies a set of filters, or kernels, to the input data in order to extract features that are relevant for the task at hand. Each filter is a small matrix of weights that slides over the input data, producing a feature map highlighting regions of the input that are important for that filter. Common convolutional filter sizes are either 3 or 5. The feature maps generated by the convolutional layer are then passed on to subsequent layers in the network [7, 8].

Pooling Layer A pooling layer is a type of layer that typically follows a convolutional layer in a CNN. It reduces the spatial dimensions of the feature maps generated by the convolutional layer while retaining the most important information. Max-pooling with filter size 2 is a commonly used pooling technique, where the maximum value within each region of the feature map is computed. Pooling helps to reduce the number of parameters in the network, which can prevent overfitting and speed up training [7, 8].

Activation Layer An activation layer is a non-linear function applied to the output of a layer in a neural network. It is used to introduce non-linearity into the network, which enables it to learn complex relationships between inputs and outputs. The Rectified Linear Unit (ReLU) function is a commonly used activation function in deep learning, as it is simple to compute and avoids the vanishing gradient problem that can occur with other activation functions [9, 8].

Regularization Layer A regularization layer is a layer added to a neural network to reduce overfitting. Dropout is a commonly used regularization technique, where a proportion of the neurons in the previous layer are randomly dropped out during training. This helps to prevent the network from relying too heavily on only one feature, and encourages it to learn more robust representations [10]. Additionally, batch normalization can be used to normalize the inputs in a layer, which can help the network to converge faster [11].

2.2.2 Convolutional Autoencoder

A classical autoencoder is an unsupervised deep learning model that encodes high-dimensional input data into a lower-dimensional latent representation and then decodes it back to the original space with minimal loss of information [12]. The autoencoder consists of an encoder network mapping the input to the latent representation, and a decoder network reconstructing the input from the latent representation. The goal of an autoencoder is to learn a compressed representation of the input data that captures its essential features in a lower-dimensional space. The low-dimensional embedding produced by the encoder is typically the most interesting aspect of the autoencoder, as it represents a compact and informative representation of the input data [6]. This embedding can be used for a variety of downstream tasks, such as clustering, classification, or visualization. In our case, the low-dimensional embedding is used as a tool to assess patient similarity by using clustering algorithms on the low-dimensional embeddings.

In the case of a convolutional autoencoder, the input data is typically a 2D or 3D image, and the encoder and decoder networks contain convolutional layers. The encoder network maps the input image to a lower-dimensional latent representation, while the decoder network reconstructs the input image from the latent representation [13, 14]. The encoder typically consists of a series of convolutional layers followed by pooling layers, which progressively reduce the spatial dimensions of the input image while increasing the number of channels. The output of the final convolutional layer is flattened and passed through one or more fully connected layers to produce the latent representation. The decoder network is the reverse of the encoder network, consisting of one or more fully connected layers followed by a series of transposed convolutional layers, which gradually increase the spatial dimensions of the latent representation while reducing the number of channels [13, 14]. The output of the final transposed convolutional layer is the reconstructed image. During training, the autoencoder is optimized to minimize the reconstruction loss between the input image and its reconstruction. The loss function typically involves a measure of pixel-wise difference between the input and reconstructed images, such as mean squared error (MSE) or binary cross-entropy [13, 14].

2.2.3 LSTM

LSTMs are a type of recurrent neural networks (RNNs) used for processing sequential data[15]. LSTMs are less prone to vanishing gradients than traditional RNNs when trained on long sequences of data. LSTMs use memory cells that store information over time, and gates that regulate the flow of information into and out of the cells. These gates include an input gate that controls the information that enters the cell, a forget gate that controls the information that is retained or discarded from the cell, and an output gate that controls the information that is output from the cell. This gating mechanism enables LSTMs to selectively remember or forget information from the past, allowing them to better capture long-term dependencies in sequential data [16]. In addition to a single-layer LSTM, it is also common to use stacked LSTM layers to further improve the network's ability to model complex sequential data. A stacked LSTM consists of multiple layers of LSTM units, where the output of one layer is fed as input to the next layer. This allows the network to learn hierarchical representations of the input sequence,

with each layer capturing higher-level features and dependencies [17]. The output of the last hidden layer of the LSTM gives a vectorial representation of the sequential input data and can be subsequently used for downstream analysis tasks.

2.2.4 Training and Evaluation Procedure

All the models used in this study were trained using a similar training procedure. We used mini-batch processing for the input data to prevent out-of-memory issues. Following the principle of empirical risk minimization, we optimized the parameters of the models by computing the gradient of the loss function and performing backpropagation. We used the Adam optimizer which combines the advantages of two other popular optimization algorithms, namely RMSprop and stochastic gradient descent with momentum (SGDM) [18, 19]. Adam uses adaptive learning rates for each parameter, which are based on the first and second moments of the gradients. The learning rate is thus dynamically adjusted during training, depending on the gradients observed at each iteration. This avoids the need for manual tuning of the learning rate, and leads to faster convergence [20]. Adam also uses momentum, which helps to smooth out the gradient updates and reduce the impact of noisy gradients. This helps the optimizer to escape local minima and saddle points in the loss landscape, and leads to better performance [20].

We used the 60'000 images from the training dataset for training and evaluated the performance of the models by computing the loss functions and through other performance metrics on the 10'000 images from the test set. We used a validation set when performing cross-validation (CV) for hyperparameter tuning.

3 Implementation and Results of CNN

To become familiarized with the architecture and training procedure of a convolutional neural network, we implemented a CNN classifier taking a single-digit image as input and learning the digit label (0 to 9). The proposed CNN architecture (Figure 2) consists of 2 convolutional layers each followed by an activation layer with the ReLU activation function and a pooling layer. The output of the last layer is mapped to 10 nodes each representing a digit class [21]. From these nodes, the class probability is obtained by applying a soft-max function. The predicted class label is the digit class with the highest class probability. Since the MNIST dataset is already very well explored and a lot of suitable model architectures are available online, the implemented architecture for classification achieved a high accuracy of 98% on the test dataset.

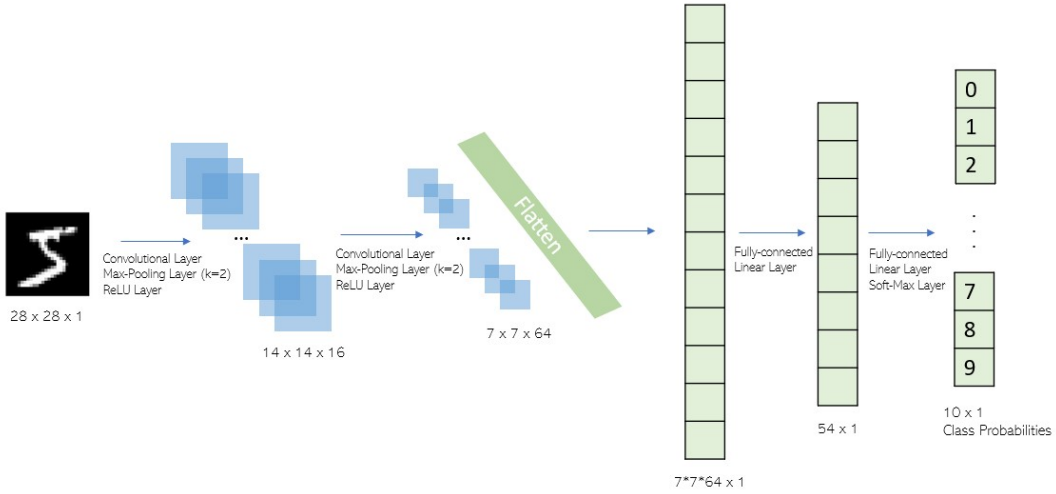


Figure 2: Architecture of CNN classifier for MNIST dataset consisting of 2 convolutional layers and 2 fully connected layers and generating a class probability for each of the 10 labels as output

4 Implementation and Results of Convolutional Autoencoder

4.1 Overview

As a next step, we extracted the most defining features of the image representation of the patient (digit). As outlined in subsubsection 2.2.2, we used convolutional layers in the encoder as well as in the decoder part of the autoencoder to capture the spatial relationships characteristic of image data. We hypothesized that the encoder network of this convolutional autoencoder creates a low-dimensional embedding of the input image that not only represents the most important features for subsequent reconstruction but also is representative of generally important features. To further consolidate this hypothesis, we used the low-dimensional embedding instead of the original image as input for a simple MLP classifier to predict the class labels from the embeddings. We compared the accuracy of this model to the accuracy of the CNN classifier from section 3. If the features extracted by the encoder are a meaningful representation of the original image, the simple classifier should be able to perform well also on the reduced input. Additionally, the low-dimensional representation was mapped onto a 2D plane using the first two components of a t-SNE dimensionality reduction and color-coded according to the class labels to get a visual representation of the low-dimensional embeddings. To determine how the feature representation changes when also being optimized for the classification task, the autoencoder and classifier were trained sequentially as well as jointly, i.e. in the same training loop. Our goal was to get an intuition about how much the latent representation of the digits depends on the downstream analysis task.

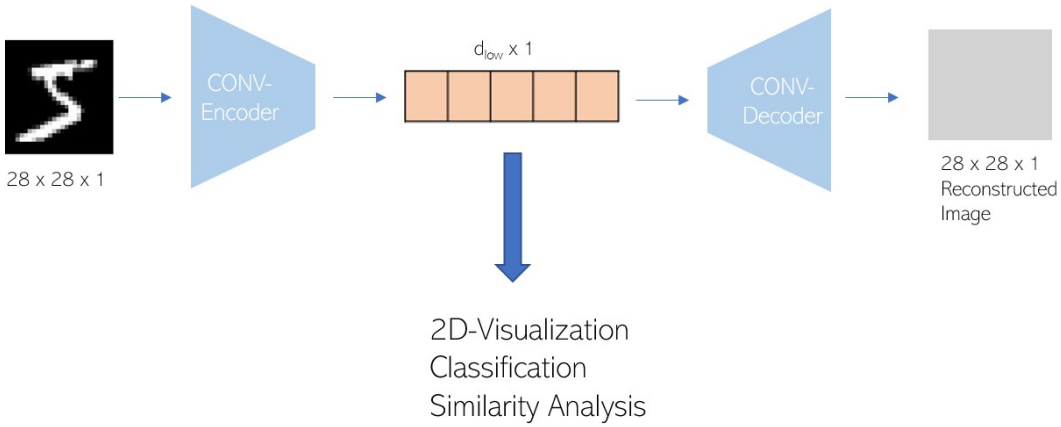


Figure 3: Convolutional Autoencoder: The encoder module maps the input image via convolutional layers to a latent space. The decoder module reconstructs the image from the latent space via transposed convolutional layers.

4.2 Implementation of convolutional autoencoder for classical MNIST data

Encoder Architecture

The encoder part (Figure 4) of the proposed convolutional autoencoder consists of 3 convolutional layers each followed by a ReLU activation layer. Instead of adding a pooling layer after each convolutional layer in order to downsize the image, the dimensionality is reduced by choosing a stride size of 2 and a kernel size of 3 for each of the convolutional layers. After the second convolutional layer a batch normalization layer is added. After the 3 convolutional layers with filters of dimensions 8, 16 and 32 respectively, the last hidden representation is flattened to produce a linear layer. This serves as input to 2 fully connected layers, the last of which maps the input to the specified dimensionality of the low-dimensional embeddings [22].

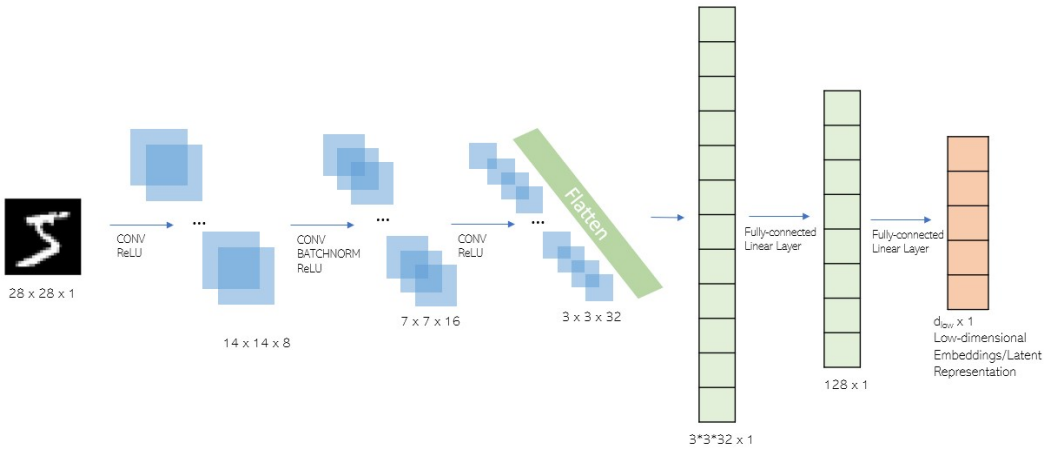


Figure 4: Architecture of Encoder: The encoder consists of 3 convolutional layers with number of filters = 8, 16, 32 respectively and each followed by a ReLU activation layer. The output of the last convolutional layer is flattened and mapped to the latent space via 2 fully connected layers of size 288 and 128 respectively.

Decoder Architecture

The decoder part (Figure 5) of the convolutional autoencoder mirrors the layers of the encoder network. The low-dimensional embedding serves as input to 2 fully connected layers. The last hidden layer is "unflatted" to attain the dimensions of a 2D image plus a third dimension representing the color channels. Via 3 transposed convolutional layers with a stride of 2, the input is expanded to the original dimensionality of the input image. In the last step, the data is fed through a sigmoid layer to obtain normalized pixel values between 0 and 1 which represent the reconstructed image of the digit [22].

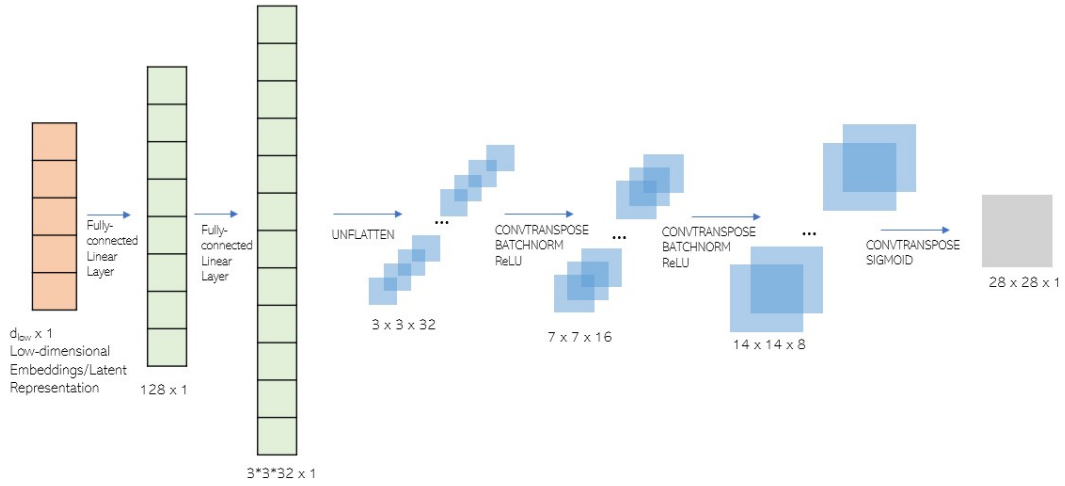


Figure 5: Architecture of Decoder: The decoder reverses the architecture of the encoder taking the low-dimensional embeddings and expanding them via 2 fully connected layers of size 128 and 288 respectively. The output is converted to 3 dimensions and expanded to the original dimension of the image via 3 transposed convolutional layers.

During training the original input image and the reconstructed image as the output of the decoder network were compared pixel-wise via the mean squared error loss. The Adam optimization algorithm with a learning rate of 0.01 was used to optimize the weights of both the encoder and decoder simultaneously.

Performance on reconstruction and classification

The implemented autoencoder performed very well on the original MNIST dataset with a MSE of only around 0.02. After training was completed, the low-dimensional embedding served as input to a very simple MLP classifier which achieved still a solid accuracy of 92% for a low-dimensional embedding of $d = 5$ after training. The class-specific accuracy was lower for more complex digits such as digit 4 of 5 for example and higher on simpler digits such as 0 or 1.

Visualization of latent representation

The visualization of the low-dimensional embedding (Figure 6) shows a clear separation and a clustering behaviour corresponding to the distinct class labels, which have not been explicitly learned by the network. From the visualization, we can guess which labels might have a higher

risk of misclassification. We concluded that the important features for reconstruction play also a big role in distinguishing between the different classes.

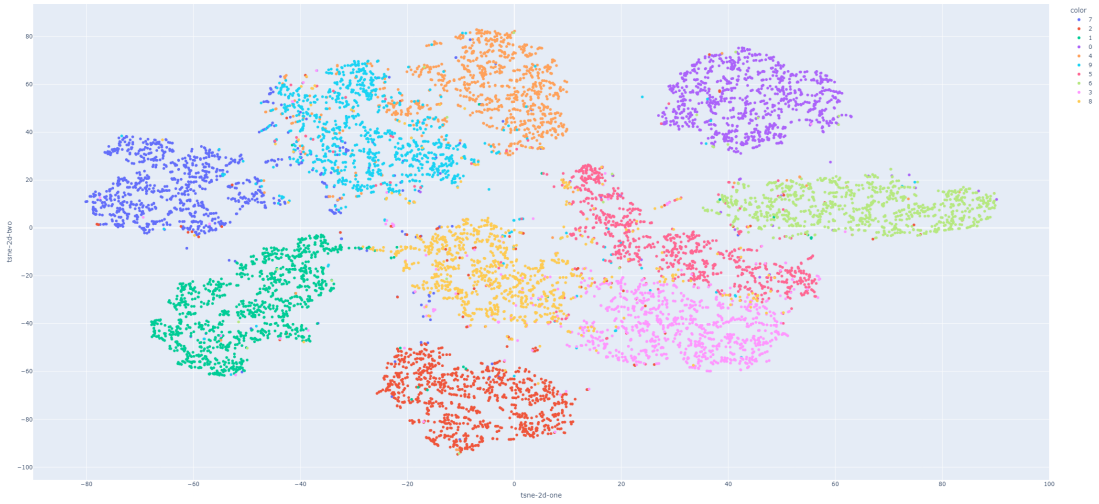


Figure 6: T-SNE coordinates of low-dimensional embedding (d=5) coloured by digit labels

4.3 Simultaneous training of autoencoder and classifier on classical MNIST data

Instead of applying the autoencoder to the input images and using the low-dimensional embeddings to train a classifier separately like in subsection 4.2, we trained the autoencoder and the classifier in the same training loop via a single loss function. For this, we took the weighted sum of the two losses (MSE-loss for the autoencoder and the cross-entropy loss for the classifier) using a weighting hyperparameter λ.

The goal of this experiment was to evaluate whether optimizing simultaneously the embeddings for both tasks significantly improves the classification accuracy and/or impacts the reconstruction ability and also if there are any significant visual changes in the low-dimensional embeddings. Significant changes in either the reconstruction error or classification accuracy would indicate that the low-dimensional embeddings generated by the autoencoder from subsection 4.2 can only be used for reconstruction and not for any downstream tasks.

Loss Function

One caveat for this approach is that the MSE-loss for training of the autoencoder and the cross-entropy-loss for training of the classifier operate on different scales. The cross-entropy loss is two order of magnitudes larger than the MSE loss. To address this imbalance, the weighting of the losses can be set using cross-validation. An alternative approach transforms both losses to the same value by weighting each loss by a term consisting of the current value of the other loss divided by the sum of the current values of both losses. Using this approach, we dynamically set the loss weights, depending on the value of the losses in the previous iteration.

Pre-defined weights for loss function
The overall loss with a pre-defined weight parameter λ was calculated according to the following equation:

$$loss = \lambda * loss_{autoencoder} + (1 - \lambda) * loss_{classifier}$$

where $\lambda = 0$ means that the parameters of the network were only optimized for classification and conversely only for reconstruction if $\lambda = 1$.

Dynamically set weights for loss function
Weighting both the losses dynamically and equally in each training step was implemented by temporarily storing the reconstruction and the classification losses from the previous iteration and then weighting each of the losses with the value of the other loss divided by the sum of both losses.

$$\lambda_{reconstruction} = \frac{loss_{classifier}}{loss_{classifier} + loss_{reconstruction}}, \lambda_{classification} = \frac{loss_{reconstruction}}{loss_{classifier} + loss_{reconstruction}}$$

We evaluated the trade-off between the classifier accuracy and the reconstruction error of the autoencoder as well as the changes in the low-dimensional embeddings for either different sizes of the low-dimensional embedding or different weights for the different losses in the overall loss function.

4.3.1 Impact of hyperparameter: dimensionality

To evaluate the trade-off between classifier accuracy and reconstruction error for different sizes of the low-dimensional embedding, the losses were weighed with $\lambda = 0.95$ which in our initial setting of the low-dimensional embedding of $d = 5$ performed very well for both reconstruction loss and classification accuracy compared to other weights.

Performance regarding reconstruction and classification

dimension of embedding (d)	weight for reconstruction loss ($\lambda_{reconstruction}$)	weight for classification loss ($\lambda_{classification}$)	reconstruction loss (test)	classification loss (test)	total loss (test)	classification accuracy (test)
3	0.95	0.05	0.035	1.505	0.036	0.8558
4			0.03	1.591	0.032	0.869
5			0.027	1.501	0.028	0.9605
6			0.023	**1.494**	0.025	0.9673
8			0.019	**1.494**	0.021	**0.9675**
10			0.017	**1.494**	0.019	**0.9675**
12			0.015	1.499	0.017	0.9616
25			**0.014**	1.503	**0.016**	0.9582

Table 1: Results for varying dimensionality of low-dimensional embedding for constant loss weights

From Table 1 as well as from Figure 7 where each of the separate losses resp. the classification accuracy are plotted for each hyperparameter d, we can deduce that a low-dimensional embedding of at least dimension $d = 5$ is needed to reach a high performance for both the autoencoder and the classifier. Increasing the dimensionality leads to a continuous decrease in the reconstruction loss but the classification loss reached its optimum between $d = 6$ and $d = 8$.

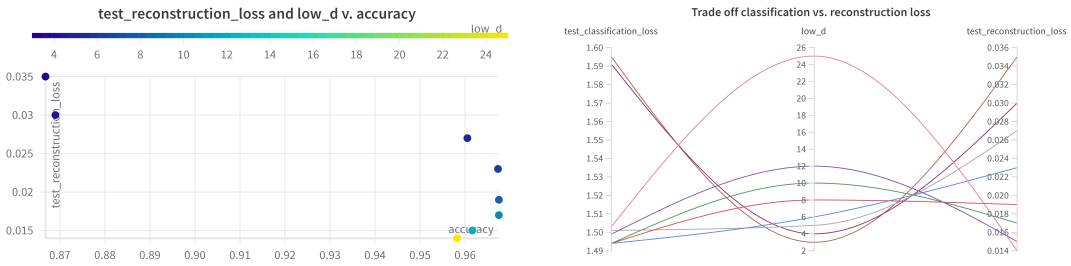


Figure 7: Trade-off between accuracy of classifier and reconstruction error for different dimensions of latent space

Visualization of latent representation

The first plot in Figure 8 shows the reconstruction ability of the trained autoencoder with $d = 5$ on the hold-out test set, which visually performs very well except for some minor confusions for the digit 5 in this example. The second plot in Figure 8 showing the t-SNE decomposition of the low-dimensional embeddings ($d = 5$) exhibits a very clear separation between the classes. Compared to the low-dimensional embeddings for $d = 5$ (Figure 6) from the autoencoder from subsection 4.2, the classes are slightly better separated but similar features are extracted. This is also reflected in the minor improvement in the accuracy of the classifier from 92% to almost 97%.

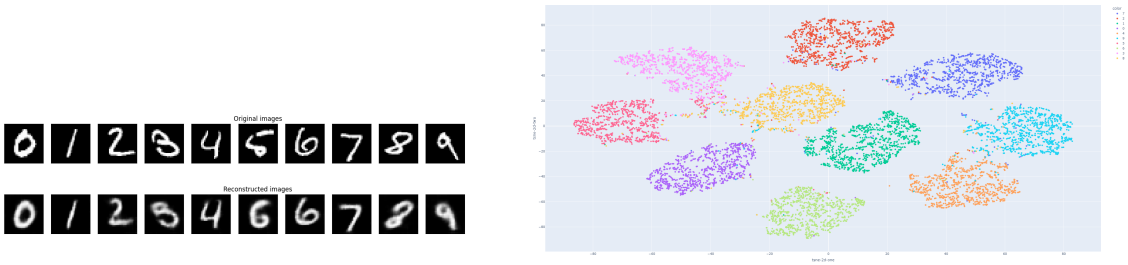


Figure 8: Visualization of reconstruction and low-dimensional embedding for $d = 5$ and $\lambda = 0.95$

4.3.2 Impact of hyperparameter: loss weighting

We tested out parameter values for λ ranging from $\lambda = 0$, for which only the classifier was trained, to $\lambda = 1$ which only trained the autoencoder. We also tracked the loss weights for the dynamically set weights over the iterations and found that they converged to $\lambda_{reconstruction} \approx 0.99$ and $\lambda_{classification} \approx 0.01$.

Performance regarding reconstruction and classification

dimension of embedding (d)	**weight for reconstruction loss**	**weight for classification loss**	reconstruction loss (test)	classification loss (test)	total loss (test)	classification accuracy (test)
5	0	1	0.231	2.072	0.233	0.2889
	0.2	0.8	0.046	1.662	0.047	0.7994
	0.5	0.5	0.04	1.696	0.042	0.7656
	0.8	0.2	0.038	1.577	0.039	0.8842
	0.9	0.1	0.029	1.504	0.031	0.9573
	0.95	0.05	0.026	**1.501**	0.028	**0.9608**
	0.99	0.01	**0.025**	1.521	**0.026**	0.9447
	1	0	0.026	2.303	0.028	0.098
	$\lambda_{reconstruction} \approx 0.99$	$\lambda_{classification} \approx 0.01$	0.026	1.929	0.28	0.6338

Table 2: Results for varying weights for the reconstruction and classification loss with a low-dimensional embedding of dimension $d = 5$

The lowest overall loss is achieved for $\lambda_{reconstruction} = 0.99$ for the reconstruction loss, which is close to the value that our dynamically set weights reached in the end. However, the classification as well as the reconstruction loss are located in the higher range, suggesting that even though the weights for the loss converged, the optimization of the network parameters did not work well. Giving only a very low weight - e.g. $\lambda_{classification}$ between 0.01 and 0.05 - to the loss of the classifier, greatly improved the performance of the classifier and did not significantly affect the reconstruction performance of the autoencoder. This can be explained by the fact that the loss of the classifier operates on a greater scale than the reconstruction loss. Moreover, it indicates that the low-dimensional representation generated by the encoder part of the convolutional autoencoder identifies features that are not only important for reconstruction but also for classification.

When looking at the trade-off plot in Figure 9, we observe that all the different weights (except for $\lambda_{reconstruction} = 0$) perform almost equally well on the reconstruction loss but the classification loss depends a lot more on the weight value. As expected the classification loss reached the maximum value for $\lambda_{reconstruction} = 1$. Furthermore, the classification loss reached its second highest value when only the classifier and not the autoencoder was trained. Indeed, in this setting, the classifier received more or less random embeddings as input which it may have found harder to learn from. This supports again the hypothesis that the autoencoder is able to extract meaningful features from the original images.

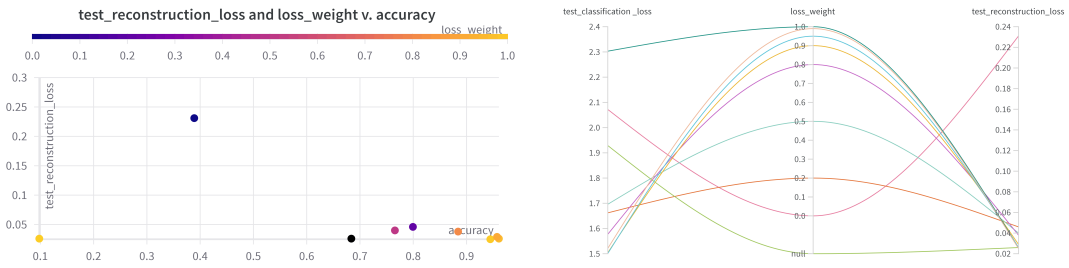
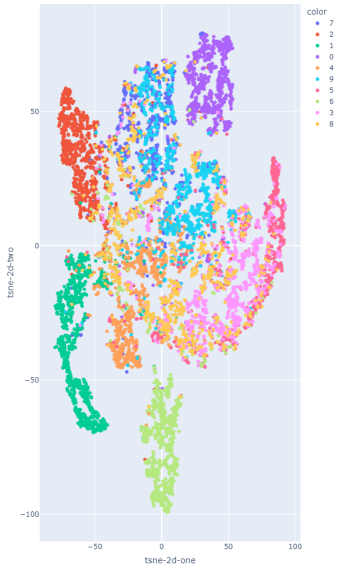


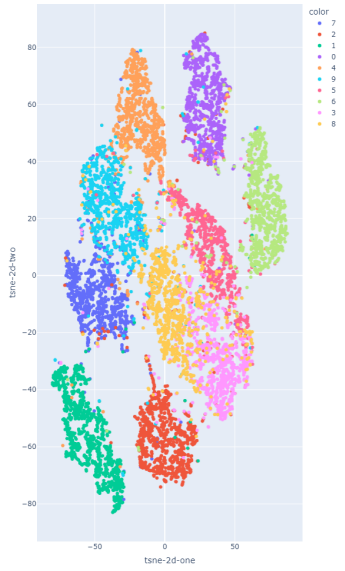
Figure 9: Trade-off between accuracy of classifier and reconstruction error for different weights $\lambda_{classification}$ and $\lambda_{reconstruction}$

Performance regarding reconstruction and classification

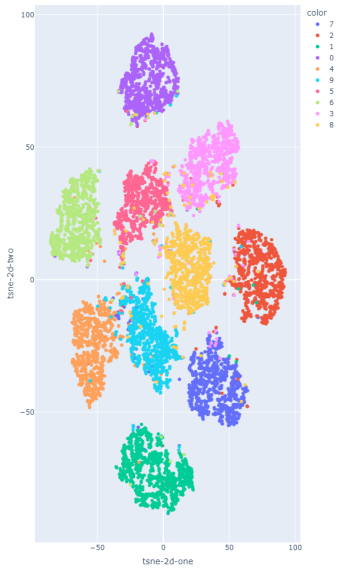
Comparing the low-dimensional embeddings and the visual performance of the reconstruction for the extreme cases of $\lambda_{reconstruction} = 0$ resp $\lambda_{reconstruction} = 1$ as well as the best performing $\lambda_{reconstruction} = 0.95$, we can observe some trends, especially in the t-SNE plots generated by the differently trained networks. As expected based on the performance in Table 2, the visual reconstruction does not differ greatly between the different weights except for $\lambda_{reconstruction} = 0$, where there has been no training of the autoencoder. Regarding the low-dimensional embeddings, we can observe a very clear separation between the class labels for the best-performing weight parameter $\lambda_{reconstruction} = 0.95$ for the classification. In general, it holds that the lower the performance of the classifier, the more we observe overlap between the classes in the low-dimensional embedding. Compared with the low-dimensional embedding of the autoencoder trained separately from the classifier (Figure 6), the visual representation of the embeddings does not change significantly. Therefore, we showed that by simultaneously training the autoencoder and the classifier the separation between the classes can be improved. However, the autoencoder itself captures the most important features also without this additional step in the training loop.



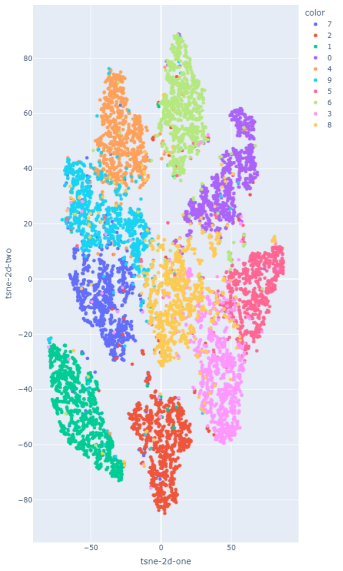
(a) $\lambda = 0$



(b) $\lambda = 1$



(c) $\lambda = 0.95$



(d) λ dynamically set ($\lambda_{inverse}$)

Figure 10: T-SNE representation of low-dimensional embedding for $\lambda = 0, 1, 0.95, \lambda_{inverse}$ and $d = 5$

(a) $\lambda = 0$

(b) $\lambda = 1$

(c) $\lambda = 0.95$

(d) λ dynamically set ($\lambda_{inverse}$)

Figure 11: Visual reconstruction for $\lambda = 0, 1, 0.95, \lambda_{inverse}$ and $d = 5$

4.4 Implementation of convolutional autoencoder for extended MNIST data

Since the architecture of the autoencoder that we implemented in subsection 4.2 worked well for the conventional MNIST dataset and extracts the most important features for an example downstream task such as classification, we assessed the performance of this particular autoencoder on a variation of the Healing MNIST dataset introduced in subsection 2.1. The images of the digits were modified according to the augmentation steps that we eventually used to generate the Healing MNIST image sequence dataset.

The dataset was augmented by several operations:

- Squares of constant size appearing in a specified proportion of the original images
- Squares of random size (between a minimum and maximum) appearing in a specified proportion of the original images
- Random rotations between 0° and 180° degrees in a specified proportion of the original images
- Combination of squares and rotations

In the final augmented dataset, we implemented a rotation of a random angle between 0° and 180° degrees in 50% randomly selected images as well as the addition of a square of size $n \times n$ (n varying between 3 and 8) in again 50% randomly selected images.

We assessed the performance of the model using a similar architecture and training procedure as insubsection 4.2. The size of the latent representation was increased to $d = 25$ in order to account for the more complex features contained in the augmented dataset. Since the dimensionality of the latent space constitutes the bottleneck for further downstream analysis/prediction tasks, we need to retain enough information from the original 28x28 2D-representation. Additionally, the learning rate was changed from $\lambda_{lr} = 0.01$ to $\lambda_{lr} = 0.001$ to achieve better results.

4.4.1 Performance of convolutional autoencoder on Healing MNIST dataset

The model was able to learn meaningful low-dimensional representations of the Healing MNIST images and produced visually accurate reconstructions (Figure 12). Most importantly, it seems to be able to distinguish between images that contain a square in the top-left corner and those without a square and even correctly reproduced the size of this square. Moreover, it also correctly reproduces the digit with the correct rotation angle even though the contours of the digit are less accurately reproduced as in the case of unmodified digits (Figure 8).

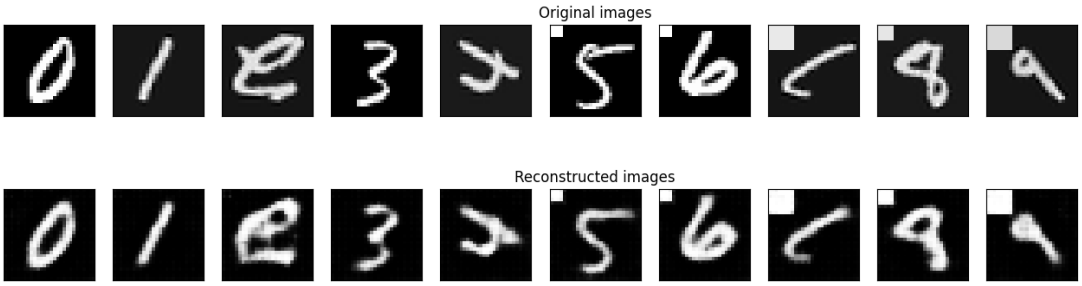


Figure 12: Reconstruction of single Healing MNIST images with random square size and random angle

Visualization of the latent representation

The t-SNE dimensionality reduction (first plot of Figure 15) shows a less clear separation between the class labels compared to the unmodified dataset. This can be attributed to the random rotation that was introduced. For some digit classes, we observe a large cluster representing the class label separated into several subclusters representing the different rotations angles or the square that have been introduced. This can be observed for example for digits of class 1 (green). In other cases, the distance between the unmodified and modified digits of the same class is much greater. In the case of digit label 8 for example, we observe that there is no overall cluster but rather a separate cluster for digits with an additional square and/or rotation. The second plot in Figure 15 shows the distribution of images containing a square and images without a square. There, the model seems to learn to some degree how to distinguish between those two classes as the digits with the square are concentrated more in the center of the 2D-representation. However, at the edges, there is still a significant overlap. Also, the degree of the separation in the embedding subspace depends on the digit class.

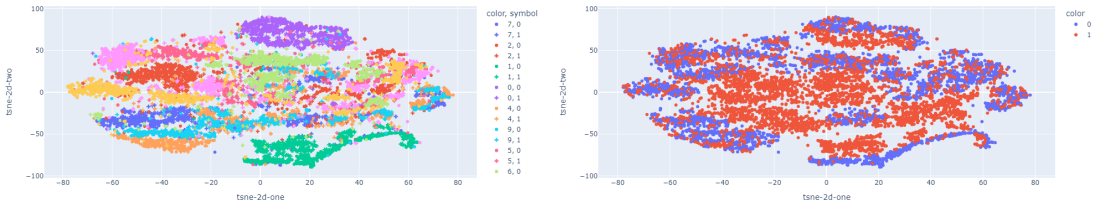


Figure 13: t-SNE representation of low-dimensional embedding for $d = 25$ coloured according to class label or indicator for presence of square

Since the separation between the classes is visible to a certain degree, we hypothesize that a downstream classifier should be able to predict the class labels with acceptable accuracy. We therefore trained the same classifier as for the analysis in subsection 4.2 which takes the output of the encoder (latent representation) as input and learns to predict the class labels.

This classifier achieved an overall accuracy of 84% after training which is significantly lower than the accuracy of around 95% on the unmodified MNIST dataset. As already observed in the low-dimensional embedding visualization, there are more regions where classes cannot be distinguished clearly anymore, making it harder for classifier to learn the decision boundary

Class	Accuracy (%)
0	93.16
1	97.71
2	79.55
3	85.64
4	78.31
5	71.52
6	87.89
7	82.39
8	80.08
9	81.96
Overall	84.11

Table 3: Accuracy per class and overall test accuracy on testset

correctly. This overlap is much more significant than what we observed in the visualization of the low-dimensional embeddings of the unmodified digits (Figure 6). The differences in the representation of different digit classes is also reflected in the class-wise accuracy, which is very high for well separated digit classes such as 0 and 1 (up to 97%) and much lower for more complex digits such as 4 and 5 (down to 71%).

5 Implementation and Results of CNN-LSTM for Healing MNIST data

5.1 Image sequence analysis in the literature

To analyze a sequence of images, the model needs to be able to capture and utilize the temporal component of the input data for which LSTMs are a natural choice. A conventional LSTM usually takes a 1-D sequential as input. Since the input data in our case is a sequence of images, we either have to extend the original LSTM model to also work for images and/or implement suitable pre-processing steps of the image data. Some work has already been done on the analysis of sequential images [23], mostly in the context of either video sequence analysis [24, 23], activity analysis based on sensor data [25, 26], satellite image analysis [27, 28, 29] or analysis of ecological/meteorological phenomena such a sea ice motion [30, 31], typhoon movement [32] or precipitation prediction [33]. The proposed models were developed for different tasks such as regression, classification, reconstruction or next image prediction.

Paper: Long-term recurrent convolutional networks for visual recognition and description
In this work, frames of a video sequence are passed to a network to classify the type of activity shown in the video. The authors propose to firstly feed each frame firstly through an CNN whose output then serves as input to an LSTM. The hidden outputs of each LSTM-module in the network are then averaged and fed through a classifier module [24].

Paper: Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting
This paper presents a model for next-image prediction with convolutional operations replacing the usual element-wise multiplications in the LSTM implementation. This enables the authors to directly take 2-D images as input for the LSTM. The architecture of the model is divided into an encoding network and a forecasting network. This so-called ConvLSTM was tested out on a synthetic dataset called MovingMNIST where groups of MNIST digits were moved (translation and rotation) from one image in the sequence to the next one and the K next images in the sequence were predicted [33].

Paper: Deep Learning Model for Global Spatio-Temporal Image Prediction
In this work, the authors compared the performance of two different models, both combining properties from LSTMs and CNNs, for sequence-to-one prediction and sequence-to-sequence prediction. The first model consists of convolutional LSTM layers where similarly to the ConvLSTM model proposed by Shi et al. [33], the output of the LSTM gates is calculated using convolutional operators. The second model consists of a sequential CNN-LSTM where the images first pass through some convolutional layers, are flattened out and subsequently fed through an LSTM for prediction. Both models perform similarly with the CNN-LSTM exhibiting better results in case of sequence-to-sequence prediction and the ConvLSTM achieving better results

for sequence-to-one prediction [29].

5.2 CNN-LSTM for next image prediction

Healing MNIST sequences

We chose to develop a model that is able to perform prediction of the next image in the Healing MNIST sequence. As described in subsection 2.1, the Healing MNIST sequence consists of consecutive images of the same digit which is rotated by a certain angle in each step (same angle within a given sequence). To increase the complexity of the sequence, a square can be added to the top left corner for a subsequence within the whole sequence, mimicking a certain event that only persists for a limited number of time steps. Since the end point of this subsequence of squares should be learned by the model, there needs to be some reasoning behind the appearance of the square. The length of this subsequence of squares can either be constant with a random starting point or both the starting and ending point of the subsequence can be random. For the latter scenario, we let the square expand and shrink back again within this subsequence to indicate the beginning and ending of the subsequence. To test the limitations of the model, we implemented datasets of increasing complexity regarding the rotations and subsequences of squares with fixed or randomly chosen lengths between $s = 3$ and $s = 6$.

The following table and figure show the different types of Healing MNIST sequences the model was trained and tested on for next image prediction.

<table>
<tr><th>Constant sequence length (s = 6)</th><th>Variable sequence length ($3 \leq s \leq 6$)</th></tr>
<tr><td>Randomly* chosen angle, no square</td><td>Randomly* chosen angle, no square</td></tr>
<tr><td>Constant angle of 30°,
subsequence of 3 squares of constant size 5
starting at random point</td><td rowspan="2">Randomly chosen angle,
subsequence of 3 squares
with growing and shrinking size
starting at random point</td></tr>
<tr><td>Constant angle of 30°,
subsequence of random number of squares
with growing and shrinking size
starting at random point</td></tr>
<tr><td>Randomly* chosen angle,
subsequence of random number of squares
with growing and shrinking size
starting at random point</td><td>Randomly* chosen angle,
subsequence of random number of squares
with growing and shrinking size starting at random point</td></tr>
</table>

*randomly chosen from the set of 30, 45, 60, 75, 90 degrees

Table 4: List of different datasets generated for analysis of sequences with constant and variable lengths with increasing complexity

Interpretation of rotation

The rotation aims to simulate the development of the patient over time, e.g. the progression of the disease. Different angles were introduced to better capture the heterogeneity of the patients, with different disease evolutions. The model therefore not only needs to learn in what

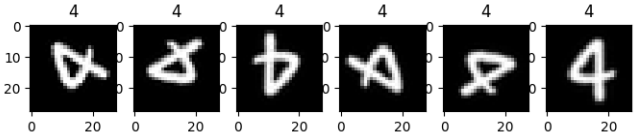
"direction" the digit/patient moves but also the speed of this movement.

Interpretation of square

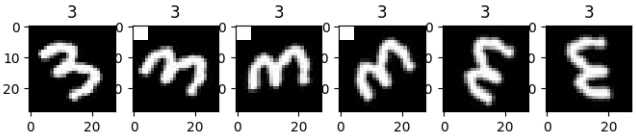
The square serves to simulate a marker such as fever or pregnancy that is similar across patients that may appear and disappear again over time. Since this marker has no relation with either the rotation or the digit class, there needs to be some shared characteristics between patients for the model to learn about the onset and offset of this marker. Either this marker only appears for a pre-defined number of timepoints such as pregnancy that always lasts 9 months, or it starts to develop gradually and then also gradually disappears. This was mimicked by the growing and shrinking size of the square. This implementation comes also with the advantage that missing measurements or inconsistent duration of this marker between patients could possibly be picked up by the model.

Interpretation of variable sequence length

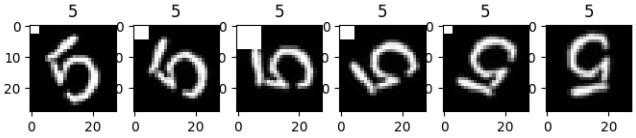
We analyzed whether the implemented model is also able to handle sequences of varying lengths. In real-life datasets, patients have varying numbers of visits and different collected measurements. Our model should ideally be able to extract the most characteristic features from a patient sequence independently of its length.



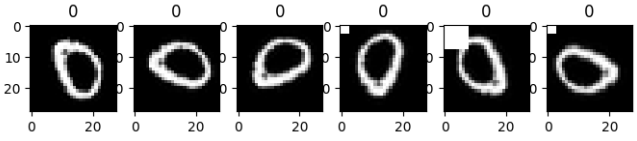
(a) Randomly chosen angle, no square



(b) Constant angle of 30°, subsequence of 3 squares of constant size 5 starting at random point



(c) Constant angle of 30°, subsequence of random number of squares with growing and shrinking size starting at random point



(d) Randomly chosen angle, subsequence of random number of squares with growing and shrinking size starting at random point

Figure 14: Example sequences with constant sequence length $s = 6$

Model implementation

Based on the previous analyses and the example implementations in the literature, we decided to implement a Convolutional-LSTM-Autoencoder which is based on the convolutional autoencoder described in section 4. We pre-trained the previously implemented encoder modules on single images and froze their weights. Then, we passed each image of the sequences through the encoder modules, outputting one lower dimensional representation per image. Then, the sequence of low-dimensional representations was passed through a second network module consisting of a linear layer expanding the size of the features to $d = 128$ and then a stacked LSTM (3 layers) with a hidden state of dimension $d = 25$.

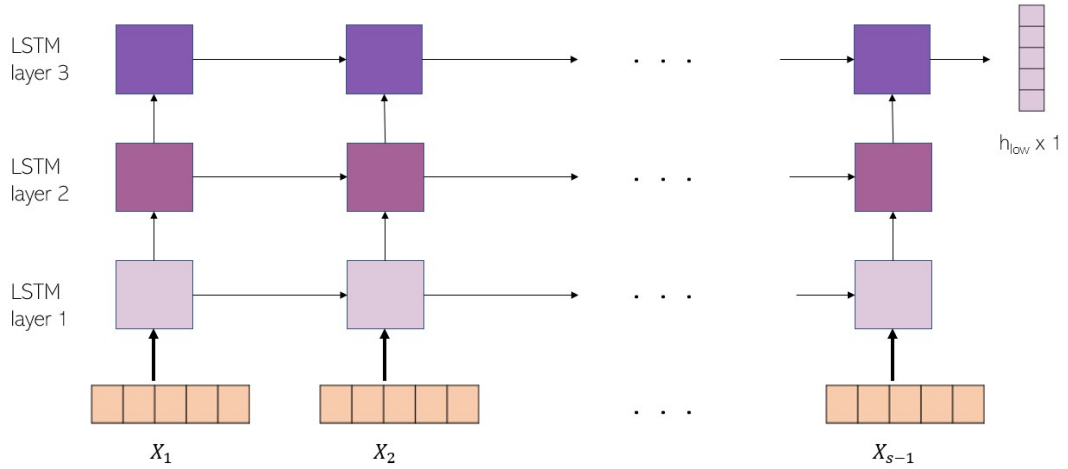


Figure 15: LSTM architecture: LSTM module consisting of 3 stacked LSTM layers; each layer mapping input to hidden state of dimension $h = 25$. The hidden state of the last layer is taken as low-dimensional embedding of the input sequence

The last hidden state of the last LSTM layer can be interpreted as a low-dimensional latent embedding of the whole sequence. The embedding represents therefore not only a single time-point of information about a patient but the entire timeline of information that is available for this patient. This latent representation can again be visualized on a 2D-plane using t-SNE dimensionality reduction and thus enables comparison of the patients.

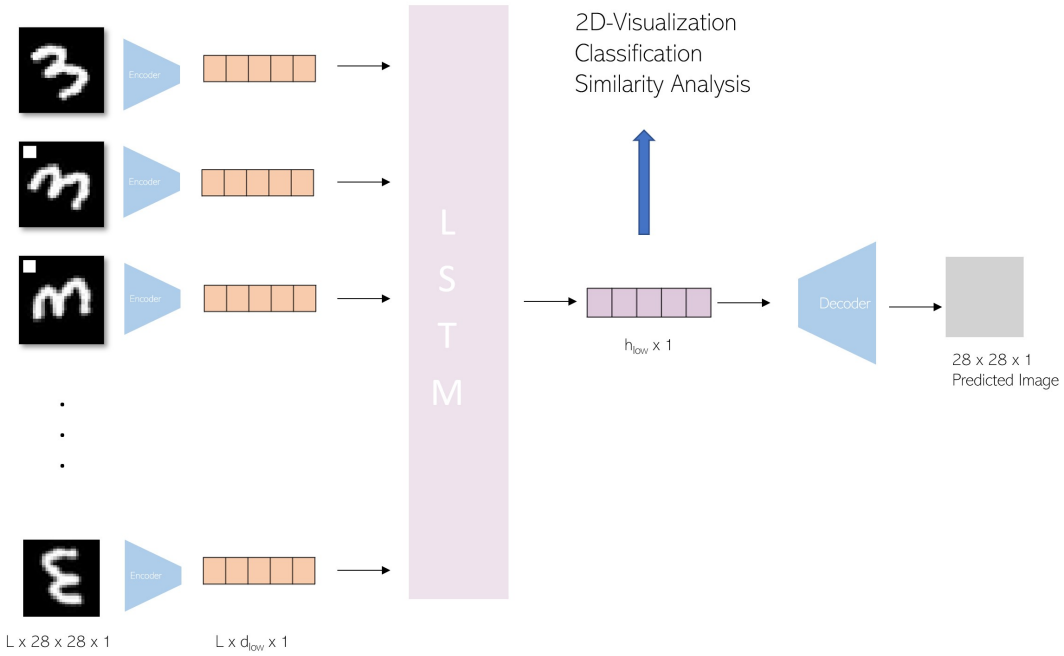


Figure 16: Convolutional-LSTM-Autoencoder: For each image of the input sequence, a low-dimensional embedding of size $d = 25$ is generated by the pre-trained encoder module. The low-dimensional embeddings are fed as a sequence into a LSTM module which outputs a low-dimensional embedding of the entire sequence. From this embedding, the predicted image is constructed using a convolutional decoder module.

5.2.1 Results for CNN-LSTM on Healing MNIST data

Results for constant sequence length

The network was trained on 60'000 sequences each of length $s = 6$ generated from the original images of the MNIST dataset and tested on 10'000 sequences generated from the original MNIST test dataset. We set the number of epochs to $n = 12$ at which point the test loss converged to the value of the training loss and plateaued. From Figure 17 we can see that both the test and training loss have a slight downward trend after the last epoch, indicating that the prediction could get even more accurate if training for more epochs and that there is no overfitting at the present stopping point. For computational reasons, we nevertheless only trained for twelve epochs.

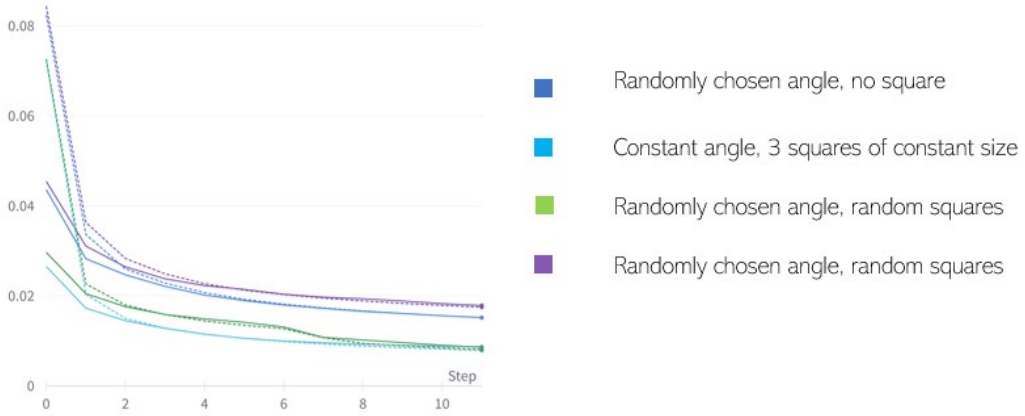
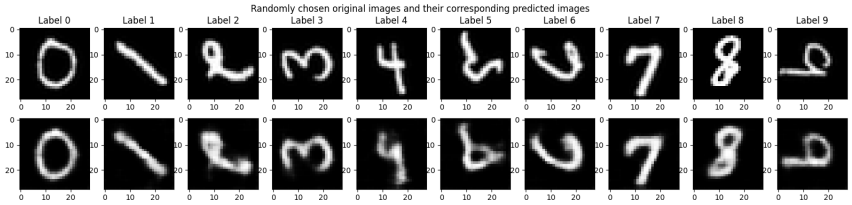


Figure 17: Training (full line) and test loss (dotted line) recorded during training of the proposed CNN-LSTM on the 4 different types of sequences with constant length

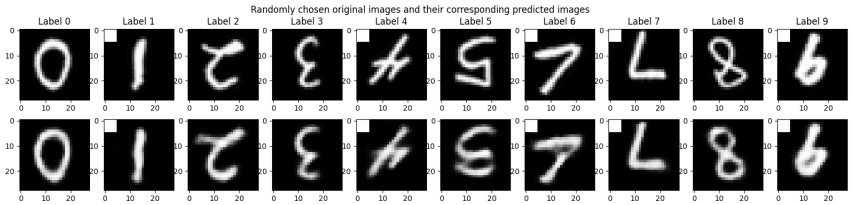
The prediction error (MSE loss) for all the different types of sequences achieves a low value, comparable to the value that was attained in section 4 for the reconstruction loss. We can also observe that there is a gap between the loss for the sequences where the angle is randomly selected from 30, 45, 60, 75 or 90 degrees and the sequences rotated by the same angle of 30 degrees. Both the training and test loss converge to a higher value for the random rotation, indicating that the randomness of the angle makes it more difficult for the model to accurately predict the next image.

Visual performance of next image prediction

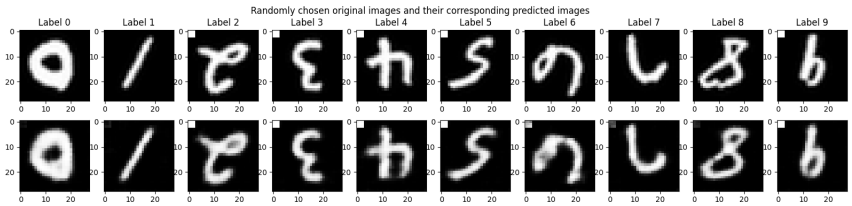
The visual results of the prediction task look satisfactory for all the types of sequences, except for the last version of the training and test datasets where the rotation angle, as well as the starting point and the number of squares with growing and shrinking square size, is picked randomly (Figure 18d). Instead of differentiating between predicted images that should contain a square and those that should not, the model predicts a square with low intensity in all the output images instead of learning that the square should end at a certain point when the size decreases again.



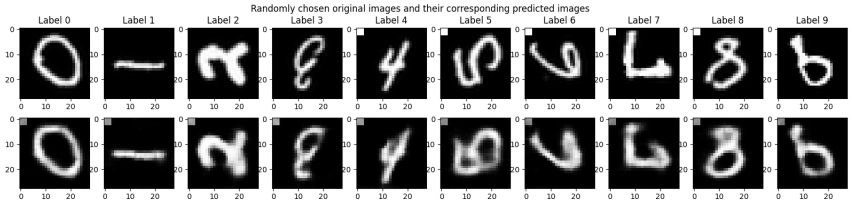
(a) Randomly chosen angle, no square



(b) Constant angle of 30°, subsequence of 3 squares of constant size 5 starting at random point



(c) Constant angle of 30°, subsequence of random number of squares with growing and shrinking size starting at random point



(d) Randomly chosen angle, subsequence of random number of squares with growing and shrinking size starting at random point

Figure 18: Predicted final images for sequences with constant sequence length (upper row: ground truth; lower row: predicted image)

To improve the model, the same architecture but using a bidirectional LSTM was implemented. The main advantage of a bidirectional LSTM is its ability to capture long-range dependencies and context from both directions [34]. By considering future information, the model can make more informed decisions at each time step. As expected, the improved model was now able to predict the presence or absence of the square correctly (Figure 19) and the loss went down significantly compared with the unidirectional model applied to the same dataset. Since the bidirectional LSTM performed better when more randomness was involved, we decided to carry out the remaining experiments with the improved model.

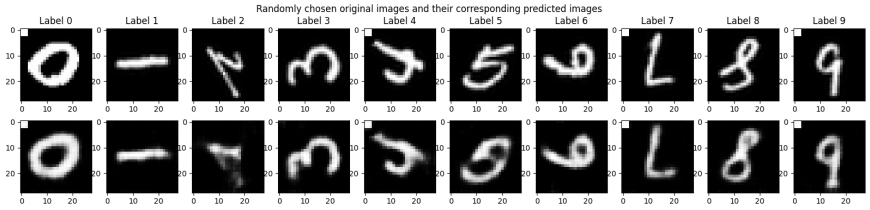


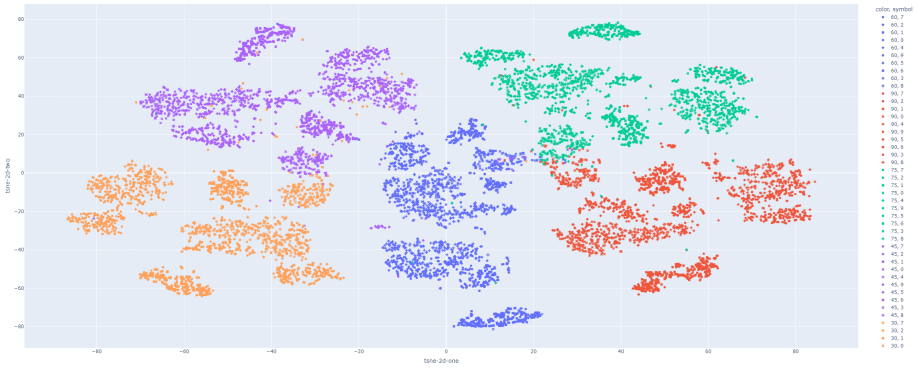
Figure 19: Predicted image with bidirectional LSTM architecture

Visualization of the latent representation

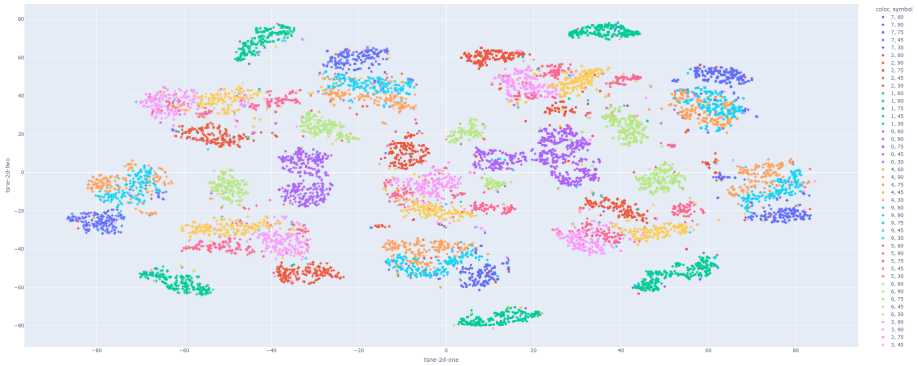
We visualized 5 different characteristics of the sequences, namely the rotation angle, presence/absence of the square in the predicted image, sequence length, number of squares as well as the original digit label. Similar to section 4, we computed and visualise the t-SNE decomposition of the last hidden state of the last (forward) layer of the LSTM layers of the model.

Figure 20a shows the distribution of different angles on the 2D plane. We clearly observe that sequences that are rotating with the same angle (i.e. similar disease progression speed) clearly cluster together and can be very well separated. When we take Figure 20b which shows the distribution of the class (digit) labels into account, we notice that every "rotation angle" cluster can be divided into 10 subclusters corresponding to the class labels. Similar to the case where each embedding corresponds to a single image, there are classes such as digit 1 and 0 that are very well separated and classes such as digit 5 or 8 that cluster more together and are more difficult to learn. We therefore hypothesize that a downstream classifier would be able to predict both the rotation angle and the class label from the low-dimensional embedding of the sequences.

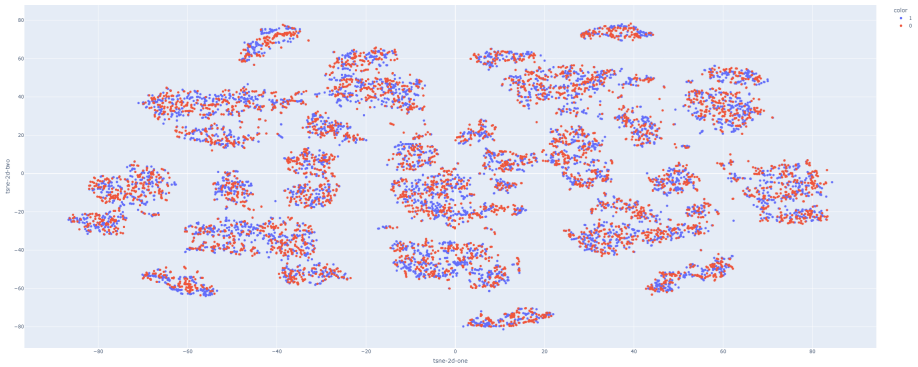
For both the number of squares or the binary indicator if a square is present or absent in the predicted image there is no clear separation or trend visible (Figure 20c and Figure 20d). However, there is not a complete overlap, indicating there might be some separation in a slightly higher dimensional decomposition.



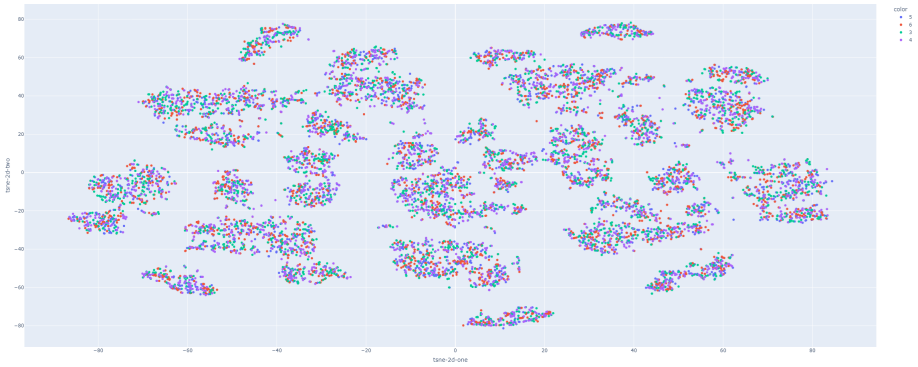
(a) Visualization of rotation angle



(b) Visualization of digit label



(c) Visualization of absence/presence of square in predicted image



(d) Visualization of number of squares

Figure 20: t-SNE plots for a sequence of length $s = 6$ with a randomly chosen rotation angle, random number of squares with growing and shrinking size with random starting point

Visualization of sequences with constant vs. random rotations

Since the embeddings visualized in Figure 20 originate from a dataset that contains random rotations, we want to learn which features the model learns differently in case the rotation angle is the same for all the sequences in the dataset. In Figure 21, we visualize the embeddings of the sequences with a constant rotation angle of 30°, but featuring random square sequences. The biggest difference that we observe is how the datapoints are grouped together with respect to the class labels. The large clusters now correspond to the digit labels instead of the different rotation angles. Therefore, the most prominent features extracted for next image prediction are associated with the original class label.

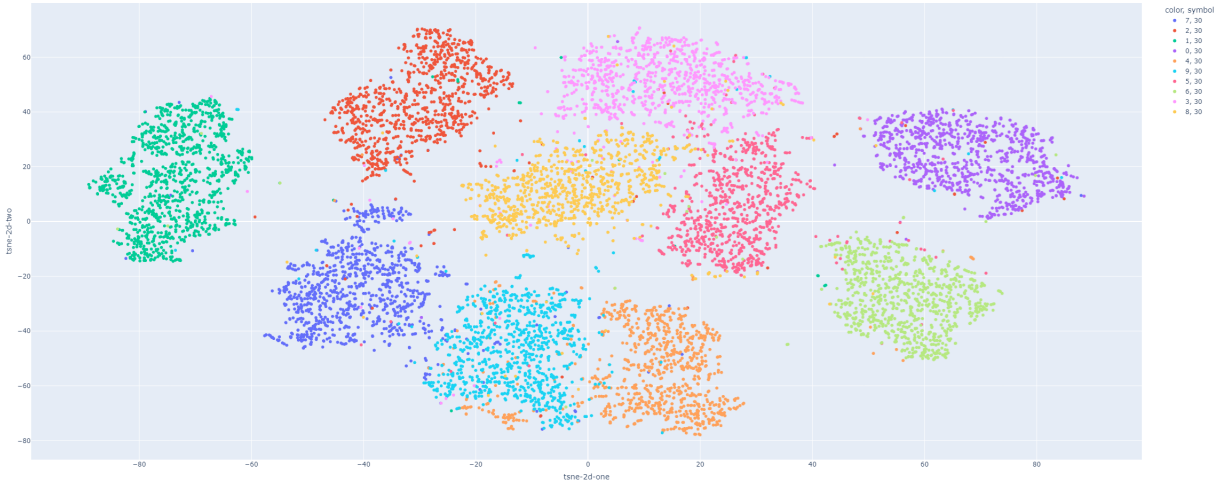


Figure 21: Visualization of digits for dataset with constant angle

Visualization of embeddings from unidirectional vs. bidirectional LSTM module

Lastly, using the most complex dataset, we compared the low-dimensional embeddings generated by the model featuring a unidirectional LSTM module to the embeddings generated by the bidirectional LSTM. The model with the unidirectional LSTM was not able to correctly predict the square in the final images, whereas the bidirectional model could learn this feature. Comparing the distribution of patient sequence embeddings with a present/absent square in the predicted image computed by the unidirectional model in Figure 22 and the bidirectional model in Figure 20c, neither of the models exhibits higher degrees of separation between the two cases. Therefore the different performances of the models cannot be explained at this stage.

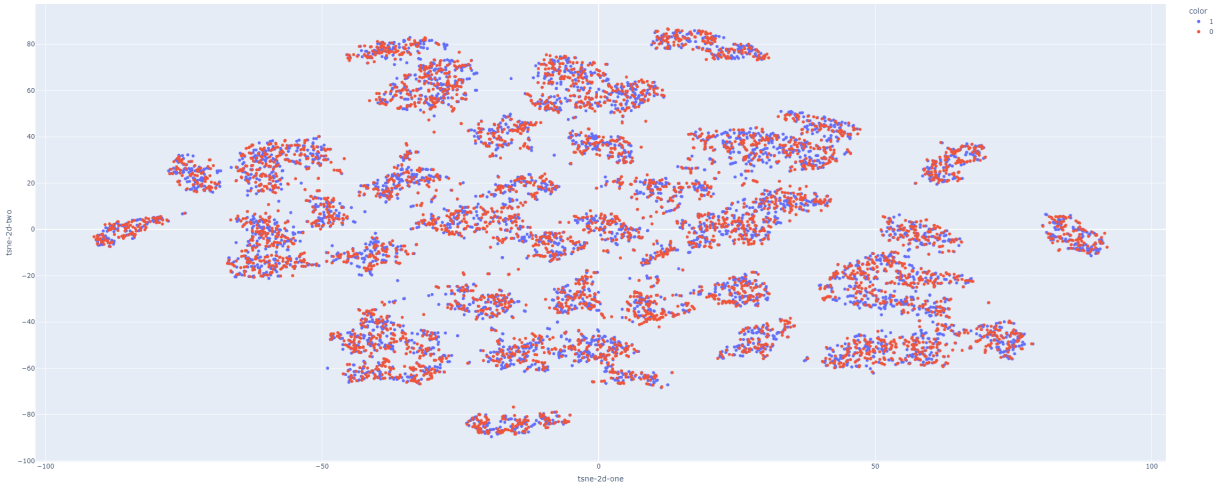


Figure 22: Visualization of absence/presence of square in predicted image for embeddings generated by unidirectional model

Results for variable sequence length

We trained the model featuring the bidirectional LSTM on the three different versions of sequences listed in Table 4, all generated from the original 60'000 training images of the MNIST dataset and tested out on the corresponding sequences generated from the original 10'000 test images. We directly used the bidirectional LSTM to enable the model to capture all the characteristics of the Healing MNIST sequence despite the added complexity of the random sequence length.

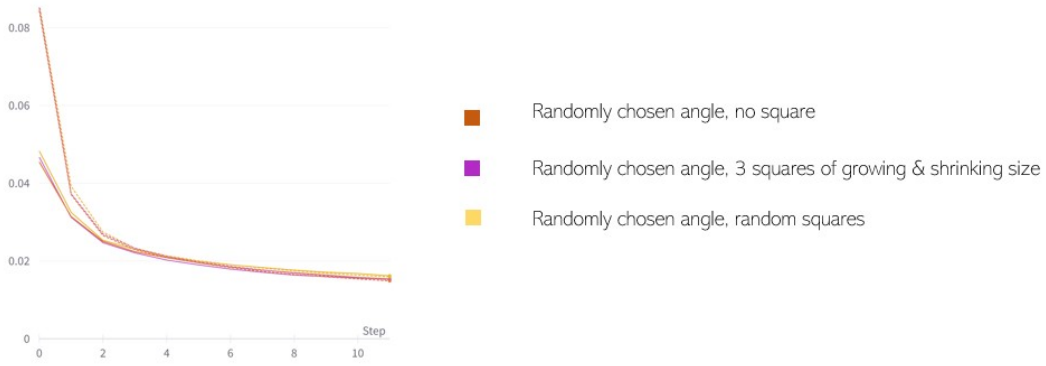
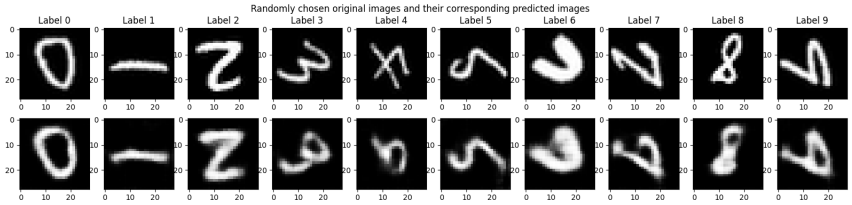


Figure 23: Training (full line) and test loss (dotted line) recorded during training of the proposed CNN-LSTM on the 3 different types of sequences with variable length

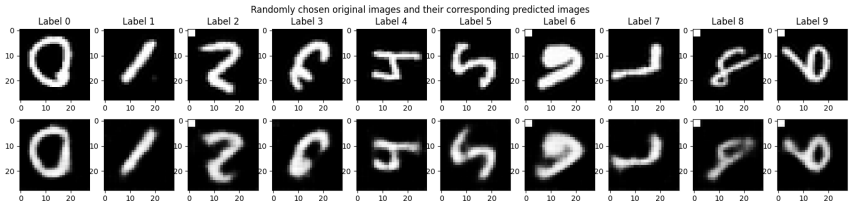
The prediction error (MSE loss) for all the different types of sequences for the training as well as the test dataset behaves very similarly during optimization indicating that the model handles the added complexity of the variable sequence length very easily (Figure 23). The value of the final losses is also comparable to the values obtained for the different sequences of constant length.

Visual performance of next image prediction

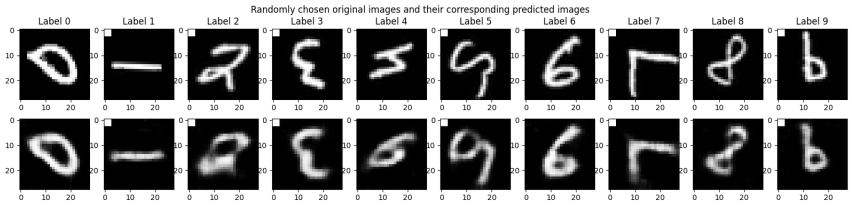
By comparing the predicted and original images in Figure 24, we see that the architecture with the bidirectional LSTM learned the presence/absence of the square as well as the rotation correctly for all the different sequence types. The exact contours of the digits however are sometimes not captured very precisely and leave some space for improvement. By training the model for more epochs and/or extending the complexity of the model architecture this could be improved. All in all, based on the visual results of its predictions, the model with the bidirectional LSTM also performs well when taking sequences of random length as input. Interestingly, when we used the training and test datasets with the sequences of random length, the simpler model featuring the unidirectional LSTM module correctly predicted both the rotation angle as well as the square. This is even the case for the most complex final dataset. It is not clear, why the unidirectional model only fails for the correct square prediction in the case of constant sequence length and random squares.



(a) Randomly chosen angle, no square



(b) randomly chosen angle, subsequence of 3 squares of growing and shrinking size starting at random point



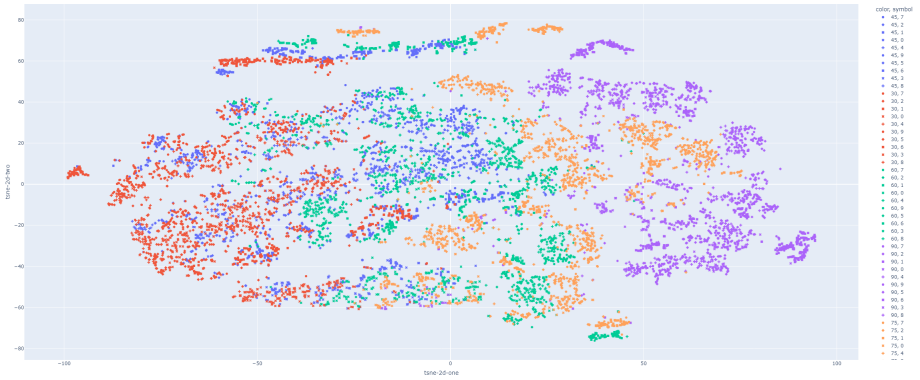
(c) randomly chosen angle, subsequence of random number of squares with growing and shrinking size starting at random point

Figure 24: Predicted final images for sequences with variable sequence length (upper row: ground truth; lower row: predicted image)

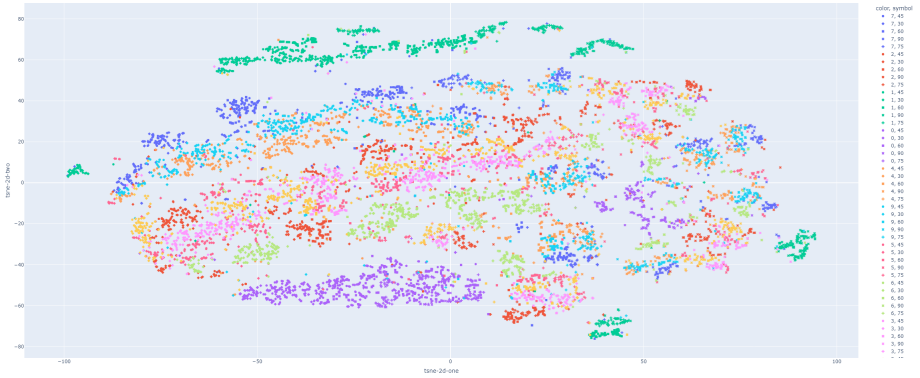
Visualization of the latent representations

In Figure 25, we visualize the latent embeddings generated by the model featuring a bidirectional LSTM on the last dataset with variable sequence length (random rotation angle and random number of squares with growing and shrinking sizes). Again, we analyze the distribution of the class labels, the rotation angle, the number and presence of squares and additionally the sequence length. In contrast to the scenario with constant sequence length, the rotation angles and digit

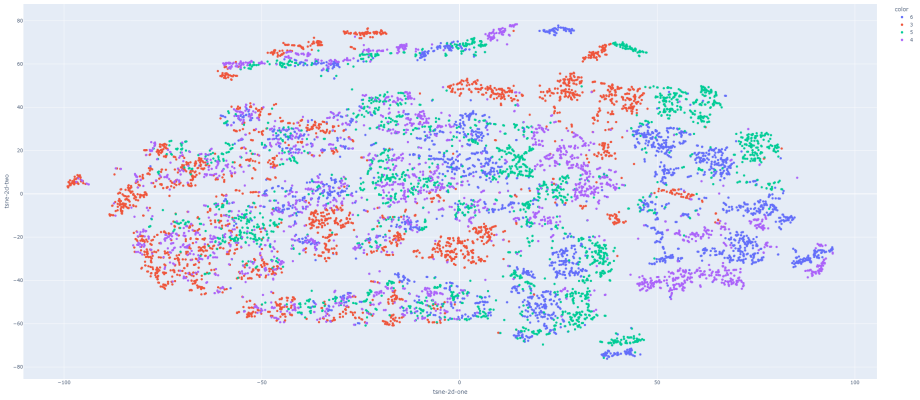
labels are not separated into clusters and subclusters anymore but are rather distributed in layers along the x- and y-axes respectively. In Figure 25a, the rotation angle increases from 30° to 90° in the positive x-direction, whereas in Figure 25b, the class labels are organized as stacked layers in the y-direction with a few outlying clusters. Each layer can be split in the x-direction according to the rotation angle. Again, some layers corresponding to simple digits are more distinctly separated than layers corresponding to digits that are harder to learn. For the different number of squares (Figure 25e) and the presence/absence of a square in the predicted image (Figure 25d), there is no clear pattern visible similarly to the case of sequences with constant length Figure 20. For the total sequence lengths visualized in Figure 25c on the other hand, we can observe that sequences with the same length tend to cluster together in multiple subclusters distributed across the entire 2D plane. Longer sequences such as $s = 5$ (green) and $s = 6$ (blue) are mostly clearly separated from the shorter sequences of length $s = 3$ (red) and $s = 4$ (purple).



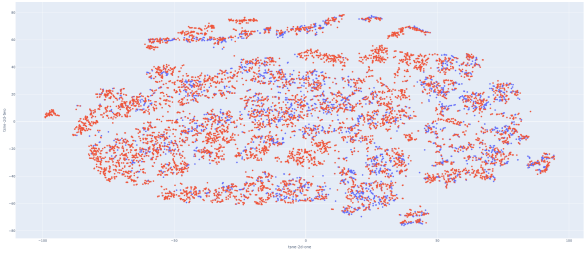
(a) Visualization of rotation angle



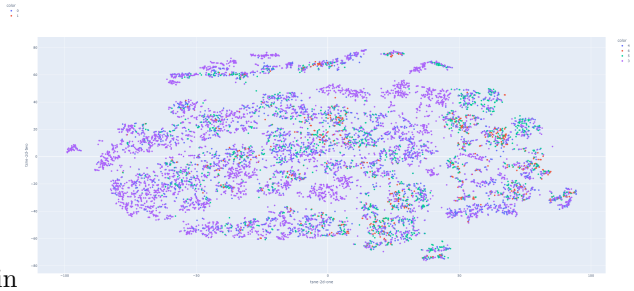
(b) Visualization of digit label



(c) Visualization of sequence length



(d) Visualization of absence/presence of square in predicted image



(e) Visualization of number of squares

Figure 25: t-SNE plots for sequences of variable length with a randomly chosen rotation angle, random number of squares with growing and shrinking size with random starting point

5.2.2 Robustness of CNN-LSTM on out-of-domain datasets

To assess the robustness and the generalizability of the implemented model, we used a pre-trained model with fixed weights to predict the next image of an out-of-domain dataset. We used the architecture featuring the bidirectional LSTM and trained the model again for 12 epochs on the dataset with random sequence length, random rotations and a random number of squares with growing and shrinking sizes.

The out-of-domain dataset was generated similarly to the pretraining dataset, except that we allowed a wider selection of rotation angles. We allowed rotation angles ranging from 10 to 90 degrees (factors of 10, i.e. 9 different angles in total), instead of the selection of the 5 different angles as described subsection 5.2.

From the visualization of the predicted image in Figure 26, we observe that the new rotation angles can be reproduced by the model. The direction of the digits is mostly aligned between the original and the predicted image. Compared to the predictions for the dataset the model was originally trained for (Figure 24c), the contours of the digits are not predicted with the same accuracy as before.

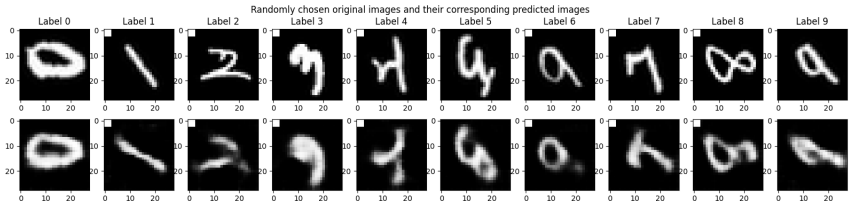
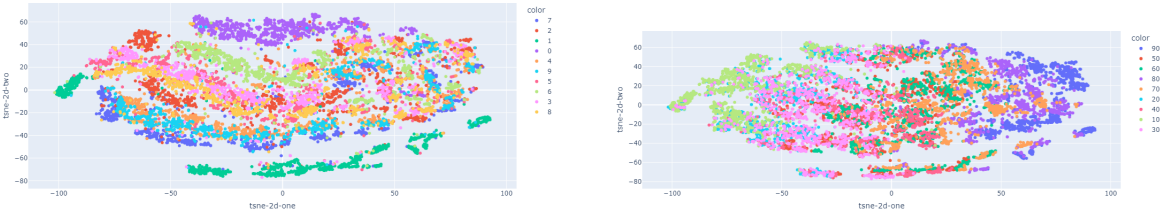


Figure 26: Visualization of predicted image for out-of-domain dataset with different rotation angles

The t-SNE plots (Figure 27) showing the distribution of the digit labels and the rotation angles greatly resemble the t-SNE plots for the in-domain dataset (Figure 25). We can again identify layers of the different class labels along the y-axis in Figure 27a and the different rotation angles are sorted in ascending order (from smallest to greatest) along the x-axis of the t-SNE plot in Figure 27b. This indicates that the model is still able to represent information about the digit type and the "speed" of the progression" in the latent representations of the out-of-domain sequences.



(a) Visualization of digit labels

(b) Visualization of rotation angle

Figure 27: t-SNE plots for out-of-domain run with larger set of rotation angles

6 Conclusion and Outlook

In this project, we proposed a Convolutional LSTM Encoder-Decoder model for next image prediction on the Healing MNIST dataset. The Healing MNIST dataset is a modified version of the original well-known MNIST dataset, where the single images of digits have been expanded to sequences of digits via rotations and further modifications aiming to mimic certain aspects and characteristics of longitudinal patient data.

The model architecture consists of a pre-trained convolutional encoder, which is part of a convolutional autoencoder. We showed that this encoder extracts important features for downstream tasks like classification and reconstruction. The preprocessed and feature-extracted image sequence is then fed into a bidirectional LSTM, which generates a latent representation of the whole sequences. Using this latent representation as input, a convolutional decoder predicts the next image.

The model successfully predicts the next image for sequences of random length, random rotation angles, and subsequences of squares with continuously changing sizes. Additionally, the latent representation of the entire sequence captures characteristics such as the digit class (interpreted as the type of disease in a patient) and the rotation angle (interpreted as the speed of disease progression). When projected onto a 2D plane through t-SNE decomposition, these characteristics form patterns that can be learned and utilized by further models for downstream analysis and used to assess/characterize patient similarity.

For future analyses, there are several potential downstream tasks that could be explored using a similar model architecture. We could extend the LSTM-Decoder model to predict the next K images, enabling long-term predictions for patients. Another option would be to implement an LSTM model for reconstructing the original input sequence, which may provide even more informative low-dimensional embeddings for patient similarity.

To enhance the dataset's resemblance to real patient data, further modifications could be made, such as introducing missing measurements (partial image availability) or missing time-points (random deletion of images within a sequence). A last step which is out of the scope of this project, would be to test out the model architecture on actual patient data. It is important to note that scaling up the model's complexity to handle actual patient data and implementing appropriate preprocessing steps would be necessary before testing the model architecture on real patient data.

Additionally, exploring alternative model classes could provide valuable insights. One example from the literature would be Convolutional LSTMs [33, 29] which directly incorporate convolutional operations in the LSTM layer. Furthermore, considering the success of deep learning models originally developed for Natural Language Processing, such as Transformers [35], it might be worth investigating their applicability to the healing MNIST dataset created for these experiments and compare the performance to our proposed Convolutional LSTM Encoder-Decoder model.

References

[1] John H. Holmes, James Beinlich, Mary R. Boland, Kathryn H. Bowles, Yong Chen, Tessa S. Cook, George Demiris, Michael Draugelis, Laura Fluharty, Peter E. Gabriel, Robert Grundmeier, C. William Hanson, Daniel S. Herman, Blanca E. Himes, Rebecca A. Hubbard, Charles E. Kahn, Dokyoon Kim, Ross Koppel, Qi Long, Nebojsa Mirkovic, Jeffrey S. Morris, Danielle L. Mowery, Marylyn D. Ritchie, Ryan Urbanowicz, and Jason H. Moore. Why is the electronic health record so challenging for research and clinical care? *Methods of Information in Medicine*, 60(01/02):032–048, may 2021.

[2] Rahul G. Krishnan, Uri Shalit, and David Sontag. Deep kalman filters. November 2015.

[3] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[4] Manjunath Jogin, Mohana, M S Madhulika, G D Divya, R K Meghana, and S Apoorva. Feature extraction using convolution neural networks (cnn) and deep learning. In *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pages 2319–2323, 2018.

[5] Benjamin Lindemann, Timo Müller, Hannes Vietz, Nasser Jazdi, and Michael Weyrich. A survey on long short-term memory networks for time series prediction. *Procedia CIRP*, 99:650–655, 2021.

[6] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[8] Stanford University. CS231n: Convolutional Neural Networks for Visual Recognition. `https://cs231n.github.io/convolutional-networks/`. Accessed on June 2, 2023.

[9] Abien Fred Agarap. Deep learning using rectified linear units (relu). March 2018.

[10] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37, pages 448–456, 2015.

[12] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. March 2020.

[13] Krushna Shinde, Vincent Itier, José Mennesson, Dmytro Vasiukov, and Modesar Shakoor. Dimensionality reduction through convolutional autoencoders for fracture patterns prediction. *Applied Mathematical Modelling*, 114:94–113, feb 2023.

[14] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, volume 28, pages 609–616, 2010.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, nov 1997.

[16] Christopher Olah. Understanding lstm networks. *colah's blog*, 2015.

[17] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[18] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[19] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2):26–31, 2012.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. December 2014.

[21] Nutan. Pytorch convolutional neural network with mnist dataset, May 2021. Accessed on June 2, 2023.

[22] Eugenia Anello. Convolutional autoencoder in pytorch on mnist dataset, June 2021. Accessed on June 2, 2023.

[23] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Yuyang Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, Francois-Xavier Aubet, Laurent Callot, and Tim Januschowski. Deep learning for time series forecasting: Tutorial and literature survey. *ACM Computing Surveys (2022)*, 55(6):1–36, dec 2020.

[24] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. November 2014.

[25] Francisco Ordóñez and Daniel Roggen. Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1):115, jan 2016.

[26] Dipanwita Thakur, Suparna Biswas, Edmond S. L. Ho, and Samiran Chattopadhyay. ConvAE-LSTM: Convolutional autoencoder long short-term memory network for smartphone-based human activity recognition. *IEEE Access*, 10:4137–4156, 2022.

[27] Waytehad Moskolaï, Wahabou Abdou, Albert Dipanda, and Dina Taiwe Kolyang. Application of lstm architectures for next frame forecasting in sentinel-1 images time series. September 2020.

[28] Waytehad Rose Moskolaï, Wahabou Abdou, Albert Dipanda, and Kolyang. Application of deep learning architectures for satellite image time series prediction: A review. *Remote Sensing*, 13(23):4822, nov 2021.

[29] Dušan P. Nikezić, Uzahir R. Ramadani, Dušan S. Radivojević, Ivan M. Lazović, and Nikola S. Mirkov. Deep learning model for global spatio-temporal image prediction. *Mathematics*, 10(18):3392, sep 2022.

[30] Zisis I. Petrou and Yingli Tian. Prediction of sea ice motion with convolutional long short-term memory networks. *IEEE Transactions on Geoscience and Remote Sensing*, 57(9):6865–6876, 2019.

[31] Fatemehalsadat Madaeni, Karem Chokmani, Rachid Lhissou, Saeid Homayouni, Yves Gauthier, and Simon Tolszczuk-Leclerc. Convolutional neural network and long short-term memory models for ice-jam predictions. *The Cryosphere*, 16(4):1447–1468, apr 2022.

[32] Peng Lu, Ao Sun, Mingyu Xu, Zhenhua Wang, Zongsheng Zheng, Yating Xie, and Wenjuan Wang. A time series image prediction method combining a CNN and LSTM and its application in typhoon track prediction. *Mathematical Biosciences and Engineering*, 19(12):12260–12278, 2022.

[33] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. June 2015.

[34] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.