

MASTER THESIS

Thesis submitted in fulfillment of the requirements for the degree of Master of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program AI Engineering

3D Spatio-Temporal Predictions with ConvLSTM and GAN models

By: DI Dr.techn. Markus Goldgruber

Student Number: 2210585005

Supervisors: Dr. rer. nat. Sharwin Rezagholi MSc.
DI (FH) Georg Brandmayr

Wien, May 31, 2024

Declaration

"As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsgesetz / Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool."

Wien, May 31, 2024

Signature

Kurzfassung

Die Vorhersage von Video Frames ist seit mehreren Jahren ein Forschungsthema, bei dem die allgemeine Idee darin besteht, eine Reihe von Video Frames aus der Vergangenheit zu verwenden und die nächsten Ereignisse im Video vorherzusagen, die in der Szene stattfinden werden. Obwohl es viele Veröffentlichungen und Fortschritte auf diesem Gebiet gibt, findet man kaum etwas über das Thema der Vorhersage des nächsten Frames von 3D-Daten, wie z. B. den Zustand oder die Topologieänderung von Objekten im Laufe der Zeit. Natürlich gibt es Anwendungen für solche Modelle, wie zeitabhängige physikalische Simulationen, Magnetresonanztomographie, CT-Scans, um nur einige zu nennen. Daher werden in dieser Arbeit die beiden am häufigsten verwendeten Ansätze, Convolutional Long Short-Term Memory (ConvLSTM) und (GAN)-basierte Netzwerke, trainiert und auf ihre Anwendbarkeit bei räumlich-zeitlichen 3D-Vorhersagen topologischer Veränderungen getestet. Der künstlich erzeugte akademische Datensatz, der verwendet wird, repräsentiert sich entwickelnde Hohlräume innerhalb eines begrenzten 3D-Gitterraums im Laufe der Zeit. Die Modelle werden anhand der Vorhersage des nächsten Frames und der langfristigen Vorhersageleistung mehrerer Frames, unter Verwendung der Jaccard-Distanz als Qualitätsmaß, bewertet. Zusätzlich werden drei verschiedene Datensatzformate ausgewertet, die kleinere und größere Frame-zu-Frame-Übergänge darstellen. Die Ergebnisse zeigen, dass größere Frame-Schritte kein Hindernis für die Modelle darstellten, sondern das Lernen des Übergangs von einem Frame zum Nächsten in beiden Modellen sogar gefördert haben. Von den beiden getesteten Ansätzen und Setups erbringt das ConvLSTM-Modell insgesamt die beste Leistung. Diese Arbeit zeigt außerdem, dass die 2D-Ansätze auf den 3D-Raum anwendbar sind und liefert Erkenntnisse darüber, wie viele zukünftige Frames mit einer akzeptablen Zuverlässigkeit, basierend auf den beiden Modellierungsansätzen und dem betrachteten Datensatz, vorhergesagt werden können.

Schlagworte: Prädiktive Modellierung, Vorhersage, LSTM, ConvLSTM, GAN, 3D, 4D

Abstract

Forecasting video frames is a research topic for several years, where the general idea is to use a set of video frames of the past and to predict the next event(s) that will happen in the scene. Although a lot of publications are available and advances in this field have been made, barely anything can be found on the topic of next frame predictions of 3D data, like the state or topology change over time of objects. Applications for such models obviously exist, like time-dependent physical simulations, Magnet Resonance Imaging, CT-scans, to name a few. Therefore, in this work the two most commonly used approaches, ConvLSTM and GAN-based networks, are trained and tested upon their applicability on 3D spatio-temporal predictions of topological changes. The artificially generated academic dataset that is used, represents developing cavities within a confined 3D grid space over time. The models are evaluated by means of single next frame prediction and long-term predictive performance utilizing the Jaccard Distance as a quality measure. Additionally, three different dataset formats are evaluated, representing smaller and larger frame to frame transitions. The results show that larger frame steps didn't produce an obstacle for the models, but have even encouraged learning the transition from one frame to the next in both. Out of the two approaches and setups tested, the ConvLSTM model showed the best performance overall. This work shows that the 2D approaches are applicable to the 3D space and delivers insights on how many future frames can be predicted with an acceptable confidence, based on the two modelling approaches and the dataset under consideration.

Keywords: Predictive Modelling, Forecasting, LSTM, ConvLSTM, GAN, 3D, 4D

Contents

1	Introduction	1
1.1	Hardware and Framework	3
2	4D Dataset Preparation	4
3	3D Object Reconstruction Method	9
4	ConvLSTM-based Predictive Model	10
4.1	ConvLSTM Network	11
4.2	Loss Function	12
4.3	Optimizer Settings	13
5	GAN-based Predictive Model	14
5.1	Single Frame Input Model	15
5.2	Multiple Frame Input Model	16
5.3	Encoder, Generator and Discriminator Networks	17
5.3.1	Encoder	18
5.3.2	Generator	18
5.3.3	Discriminator	19
5.4	Loss Function	21
5.5	Optimizer settings	22
6	Quality Metrics for Predicted Objects	23
6.1	Mean Squared Error	23
6.2	Jaccard Distance	23
6.3	Plausibility Thresholds for Model Quality Assessment	24
7	Results of the ConvLSTM-based Model	25
7.1	Training Performance	25
7.2	Next Frame Prediction Quality	26
7.3	Long-Term Prediction Quality	28
8	Results of the GAN-based Model	30
8.1	Training Performance	30
8.2	Next Frame Prediction Quality	32
8.3	Long-Term Prediction Quality	35

9 Result Comparison of All Models	38
10 Conclusion	40
10.1 Limitations and Outlook	40
Bibliography	42
List of Figures	45
List of Tables	47
List of Source Codes	48
List of Abbreviations	49
A ConvLSTM 3D Next Frame Prediction	50
A.1 ConvLSTM3D Training Algorithm	50
A.2 ConvLSTM3D Training Algorithm - Data Parser	60
A.3 ConvLSTM3D Prediction Algorithm	63
A.4 ConvLSTM3D Long-Term Prediction Algorithm	65
B GAN-based 3D Next Frame Prediction	67
B.1 Single Frame Input Training Algorithm	67
B.2 Single Frame Input Training Algorithm - Data Parser	88
B.3 Single Frame Input Prediction Algorithm	91
B.4 Single Frame Input Long-Term Prediction Algorithm	93
B.5 Multiple Frame Input Training Algorithm	95
B.6 Multiple Frame Input Training Algorithm - Data Parser	117
B.7 Multiple Frame Input Prediction Algorithm	120
B.8 Multiple Frame Input Long-Term Prediction Algorithm	122
B.9 Long-Term Prediction - Data Parser	125
C Utility Functions	128
C.1 Function to Plot Losses and Save them to a File	128
C.2 Functions to Generate 3D Objects from Grid-based data	132
C.3 Logger Class	135
C.4 *.npz to *.tfrecords Converter	136
C.5 Split Data into Train-, Validation- and Testsets	139

1 Introduction

Future video frame predictions utilizing machine learning is a research topic for several years and many researchers have dedicated their work on this topic. The general idea is to use a set of video frames of the past and predict the next event that will happen in the scene. In the simplest setting predicting just one, or even better, predicting several next frames. Several advances in this field have been made over the last decade applying different approaches for general video frame predictions or weather forecasting, to name a few [26], [17], [15], [3], [21], [2], [33], [13]. Based on all this work done on video frames and the advances in 3D generative networks over the last years [34], [1], [6], [36], including comprehensive surveys by [29], [27], an obvious step is to extend next frame predictions to the 3D space.

To date barely any publication has been dealing with a fully 3D object changing its topology over time, namely 3D spatio-temporal models. Works that are at least dealing with 3D spatio-temporal data are human motion predictions in 3D by [7] and [5], which are not predicting a full body, but the 3D coordinates of joints of the bodies. However, real-world applications that would profit from such models do exist and demand machine learning-based solutions. Concrete applications of such algorithms are for example

- time-dependent physical simulations in 3D space,
- time-dependent measurements or scans, e.g. Magnetic Resonance Imaging (MRI) and Computed Tomography (CT) scans, in 3D space,
- physics in computer games and
- everything that grows,

to name a few.

The reason for the lack of research in this field could originate from two possible reasons. First, 4D datasets are very rare due to their complex structure and non-generalized dataformats compared to images and videos, secondly, training such datasets is quite demanding regarding the hardware. A huge amount of data needs to stored and operated on during training, making a high end hardware setup necessary and therefore not that easy to handle. Temporal predictions of two dimensional data like videos already comes at a high cost due to the added third dimension of time and its grid based nature, where all positions in the grid store information, even though it might be empty space filled with zeros. Logically even more costly regarding the data are 3D grid-based temporal problems. Nevertheless, the grid-based format comes with a lot of advantages in combinations with neural networks and their treatment within the GPU, allowing for straight forward implementation of vectorized tensor operation, without an encoding

or conversion into a compatible format of the 3D object. An open source grid-based dataset of that kind (3D-spatio-temporal format) is used and provided by [11]. This dataset is a artificially generated dataset of 32000 datapoints, each in the format 128x128x128x128 and readily prepared in python format to be used for training of such models.

The two most commonly found deep learning architectures are Generative Adversarial Network (GAN)-based and Convolutional Long Short-Term Memory (ConvLSTM) networks. GAN-based models for generating new videos analogous to GANs for generating images were first introduced by [24], utilizing and enhancing the idea of 3D convolutional networks from [32]. Aigner et al. [3] took a step further and introduced "FutureGAN" an encoder-decoder GAN architecture build with progressively growing layers, which were first introduced by [19], to predict future frames from past frames, see figure 1.1. In such a setting the GAN becomes a supervised technique instead of an unsupervised one, where past frames deal as the features to be learned from. Linkermann [21] investigated different ideas, based on "FutureGAN", on how to feed the GAN the temporal information of the past to improve the forecast performance, for instance, different encoder architectures and varying length of past input frames. These approaches in combination with the advances on 3D GANs lead to the idea within this work for 3D spatio-temporal predictive modelling.

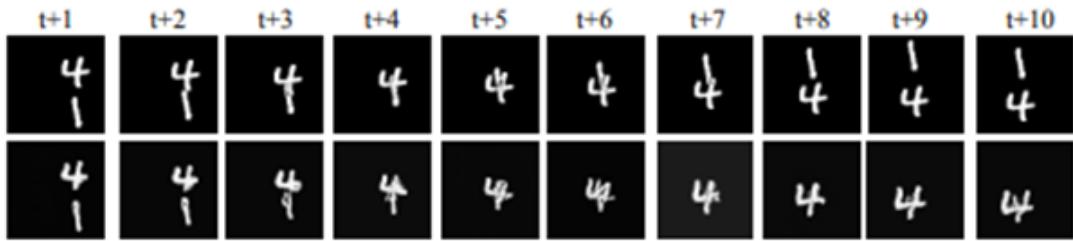


Figure 1.1: 10 next frame predictions with FutureGAN on the Moving MNIST dataset (top row: ground truth; bottom row: FutureGAN). Source: [3]

In contrast to the GAN-approach, another common approach are ConvLSTM models, which were first introduces for the purpose of future frame predictions by [26], see figure 1.2. Based on Long Short-Term Memory (LSTM) models, which are part of the recurrent neural network family, originally introduced by Hochreiter and Schmidhuber [14] and are specifically designed to learn from sequential data, ConvLSTMs are utilizing convolutional layers within the Long Short-Term Memory blocks in the hidden layers of a Recurrent Neural Network (RNN). Further works on this approach have been done by Adler [2] by introducing so-called peepholes into the LSTM blocks and Hosseini et al. [15], who added inception layers, first introduced by [28], with the goal to make the networks wider instead of deeper.

Both approaches, GAN-based and ConvLSTMs, have shown promising results in recent years for next video frame predictions and even for longer forecast than just a single next frame. Therefore, this work should deliver insights on the questions if the 2D approaches are applicable to 3D cases and furthermore, how far can we look into the future with an acceptable confidence.

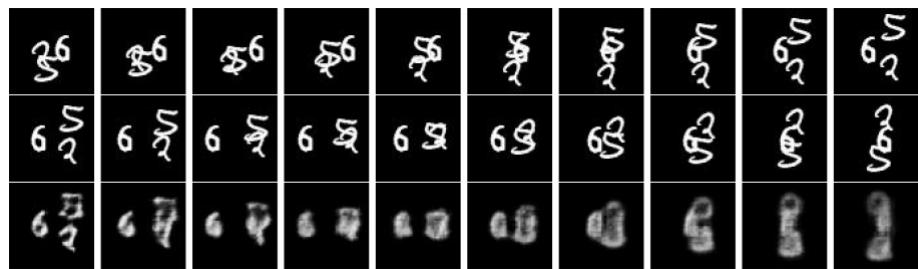


Figure 1.2: 10 next frame predictions with ConvLSTM; top: input, middle: ground truth, bottom: prediction. Source: [26]

1.1 Hardware and Framework

Data preparation and training of all models has been performed on a Windows 11 Desktop PC with a NVIDIA GeForce RTX 3090 with 24GB of Memory, a 12th Gen Intel(R) Core(TM) i9-12900KF Processor and 64GB RAM. The framework used for building the models is tensorflow 2.10.1 [30].

2 4D Dataset Preparation

Since 4D grid-based data is quite resource demanding the data needs to be prepared in an efficient way to not be the bottleneck of the training procedure due to an reading-freeing-memory overhead. As mentioned in chapter 1 the two methods that are used to train and predict next 3D object frames are the ConvLSTM and GAN approaches. Both of them need the data to be prepared in a different way, specifically talking about the right format of the pairs of input frames and target frames. Input frames in this manner correspond to past time frames and targets to next time frames.

The dataset used in all trainings is the dataset from [11] with a total of 32000 3D objects of a size of 128x128x128 over a time of 128 time-points. The objects are a random number of combinations of ball and tori shape variations, generated by parameterized analytical equations of varying complexity to represent different orientations and volume over time. A comprehensive description of these hypervolumes and their parameterization can also be found in [11]. Figure 2.1 shows a sample of 3D objects over 18 equally spaced time points.

All the data has been converted into the TFrecords format [31] to allow reading the data in the most efficient format when needed and pre-loading (prefetching) data during training for the next loop, because the dataset is too large to keep all of it in memory during training.

Since a grid-based approach is chosen for building the ConvLSTM and GAN models, the neural network gets big pretty fast due to the operations within the layers, especially due to convolutional layers and their corresponding number of filters. A reasonable compromise needed to be found between an object resolution still covering differences of relevant features and the training time on a single GPU setup. Therefore, all the 32000 data-points were converted from 128x128x128x128 to the size per sample of 128x16x16x16 and split into training-, validation- and testing sets, see table 2.1. Figure 2.2 shows the two resolutions next to each other for a random sample at a random point in time.

Table 2.1: Training, validation and test split.

Total	Training	Validation	Testing
100%	80%	10%	10%
32000	25600	3200	3200

The data from the 128 time-points per sample is extracted and prepared for training in 3 different formats, which are:

- T=20, TINC=1: Input length equals 20 time frames, from which every single frame is

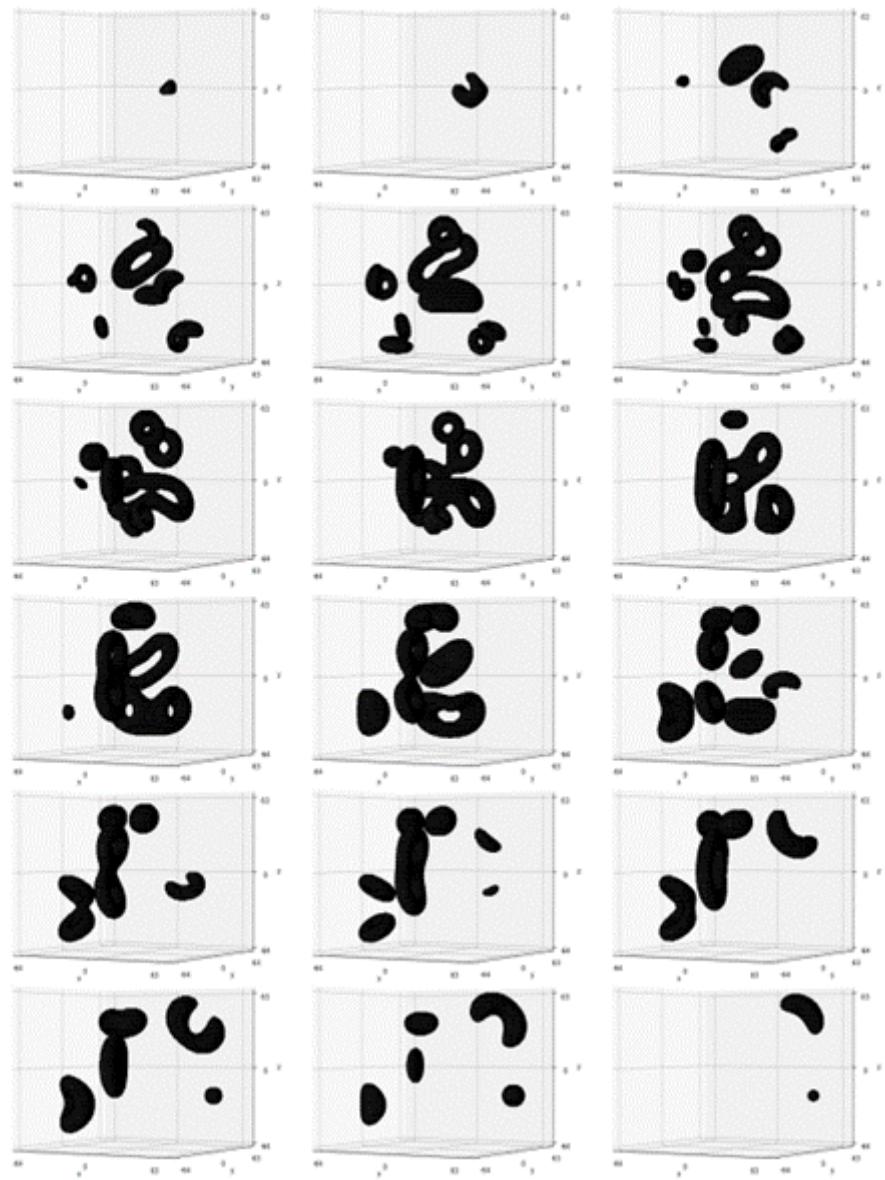


Figure 2.1: Cavities within a $128 \times 128 \times 128 \times 128$ sample, 18 equally-spaced 3D slices along the time-axis from left to right and top to bottom. Source: [11]

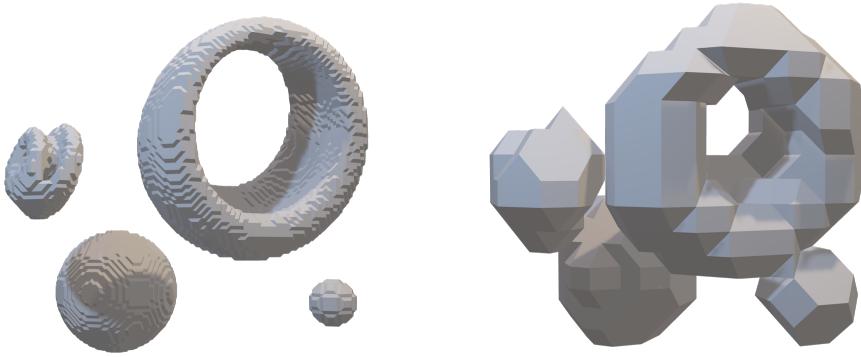


Figure 2.2: Resolution comparison of a random sample at a random time-point. Left: Original Size of 128x128x128; Right: Max-Pooled Size of 16x16x16

taken.

- T=40, TINC=2: Input length equals 40 time frames, from which every second frame is taken.
- T=80, TINC=4: Input length equals 80 time frames, from which every forth frame is taken.

All of them are extracting exactly 20 frames in total by cropping the time at the beginning and end, just with different temporal differences. E.g. T=80 and TINC=4 would result in taking the middle 80 consecutive time-points, where the first 24 and the last 24 time-points are cropped and from these only every forth frame. This extraction is implemented because most of the 4D data-points are having no objects yet visible in the beginning and end, which would lead to 4D tensors as input with zeros only.

These three formats are defined to illustrate the effect of change from one frame to the next and how much harder it is to learn greater change. For instance, the T=20 and TINC=1 correspond to very little change between two frames, whereas T=80 and TINC=4 are sets of consecutive frames with clear differences.

Since LSTM models are sequence based models per definition, they expect sequences as inputs. Therefore, the samples are prepared with pairs of input and target sequences, where the target sequence has an offset of one time frame. Figure 2.3 illustrates the extraction procedure of a sample for the ConvLSTM approach for exemplary choices of total time T and time increment TINC.

In contrary to the ConvLSTM, the temporal GAN approach doesn't use the full time history in one go during training, but either one or many past time-points as inputs (features) and one (next) time-point as output (target). Two different models are trained and tested, both of which require the samples to be prepared in two different formats:

- Single Frame Input Model

The simplest training approach is to use one time-point, e.g. at t_0 , as input (feature),

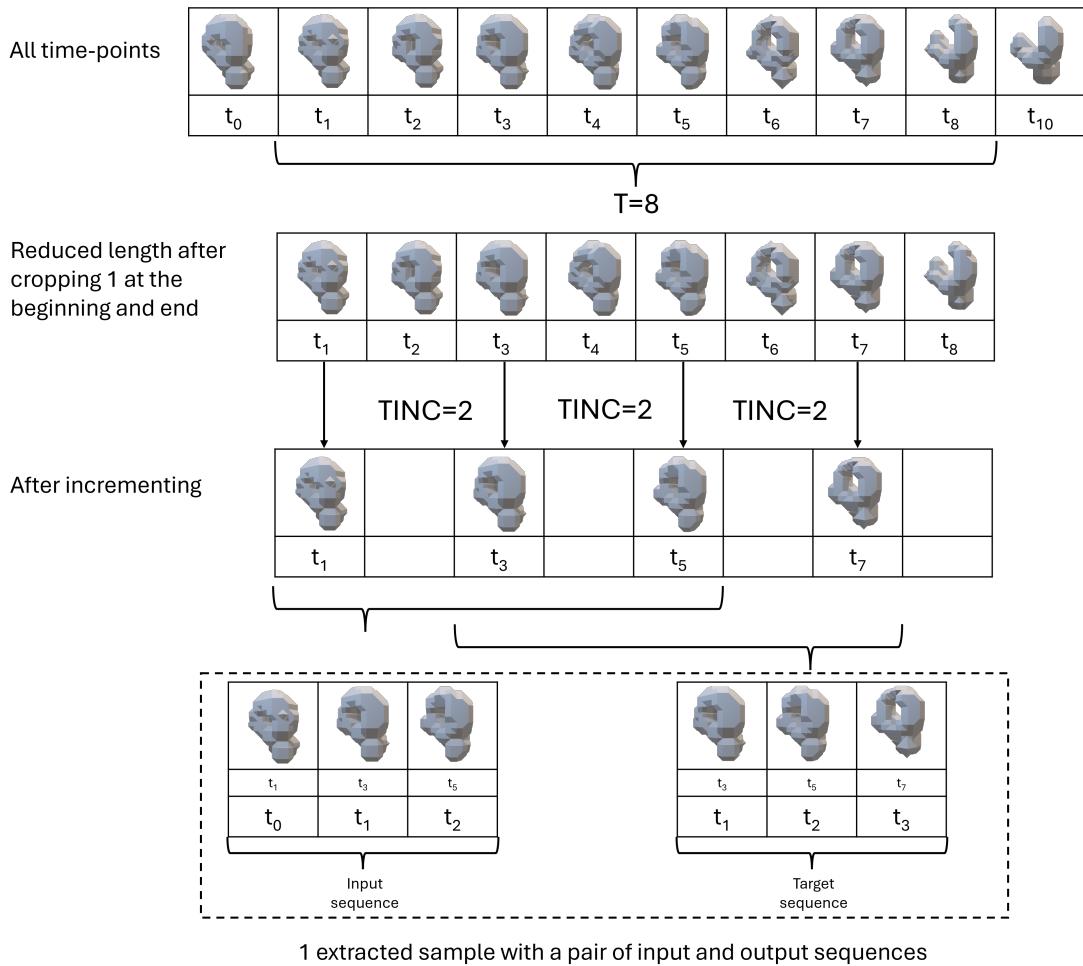


Figure 2.3: Illustration of how parameters T and $TINC$ influence the sample extraction for the ConvLSTM models.

which is then trained on the next time instant t_1 (target). In this case the data format is $2 \times 16 \times 16 \times 16$, which consist of the $1 \times 16 \times 16 \times 16$ input object and $1 \times 16 \times 16 \times 16$ target object.

- **Multiple Frame Input Model**

Another approach is to use multiple past time-points as inputs, specifically in this study three former time-points are used at times t_{-2}, t_{-1}, t_0 , which are then trained on the next time instant t_1 (target). The advantage of using multiple time-points is that the model can learn better from the past. In this case the dataformat is $4 \times 16 \times 16 \times 16$, which consist of the $3 \times 16 \times 16 \times 16$ past input objects and $1 \times 16 \times 16 \times 16$ target object.

3 3D Object Reconstruction Method

The objects in the dataset are prepared in grid-based format, where each position in the grid is either one or zero, analogous the objects generated by the generator network in the GAN-based model or by the ConvLSTM model are 3D tensors of size $16 \times 16 \times 16 \times 1$ with float values between zero and one. Displaying these values in a 3D cloud plot would allow to see the objects shape and features at least to some extend, but it won't be a 3D object including surfaces per definition. Therefore, a reconstruction algorithm (C.2), namely Marching Cubes algorithm from [25] is used to generate an actual object out of the 3D grid-based point cloud in STL format using the numpy-stl library. Figure 3.1 shows the point cloud and the reconstructed object next to each other. For the reconstruction a threshold of 0.5 is used to convert the floats into binary format. After this conversion values below 0.5 are set to zero and values above are set to one, corresponding to the support points for the reconstruction of the object.

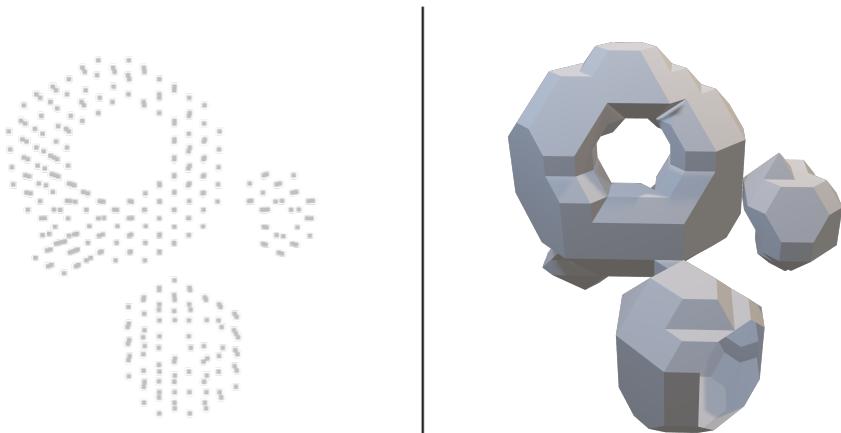


Figure 3.1: Left: Point cloud of an object; Right: Reconstructed object from points with Marching Cubes algorithm.

4 ConvLSTM-based Predictive Model

LSTM models in general are build for training and predicting sequences, like time history data, texts, video and any other sequential data. It was invented back in 1997 by Hochreiter and Schmidhuber [14] and is part of the family of Recurrent Neural Networks (RNN) with the speciality of its so-called memory cell, see figure 4.1. Short-term refers to to memory cell activations that can store information over "long" distances of sequences.

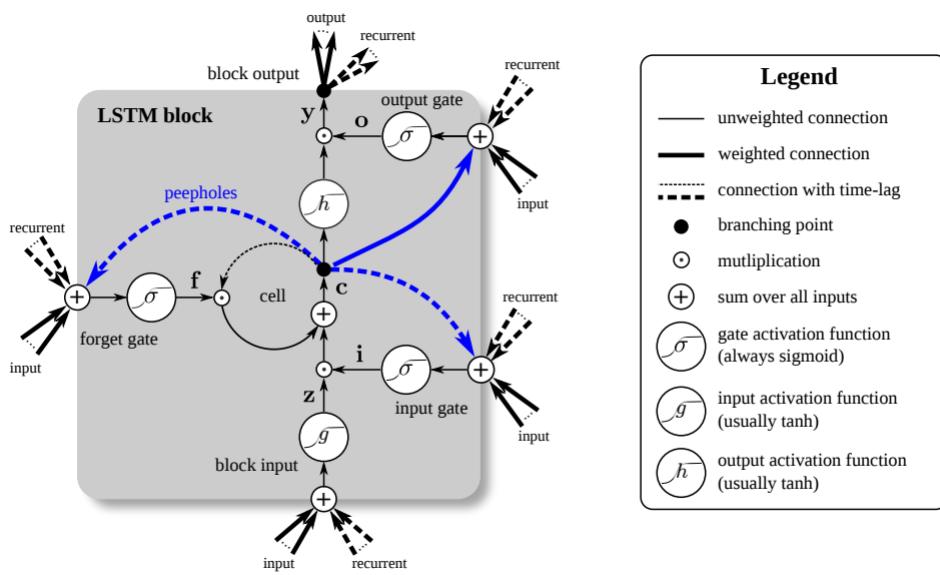


Figure 4.1: LSTM block. Source: [9]

LSTMs are one of the most used RNNs for translations, speech and language recognition, but got pushed into the background in recent years due to the rise of transformer-based models. They are very flexible models that can be used to learn and predict different sequence problems as depicted in figure 4.2.

In the case of next 3D frame prediction the "many to many" mode is the appropriate choice, since the input is the time sequence of a 3D object that's changing its state in each subsequent time frame and the output is the consecutive next time frame for each input time frame. Convolutions proved to be very effective in image analysis, which is why incorporating convolutions into LSTMs seemed natural and was first done by [26] and therefore called ConvLSTM. Figure 4.3 shows the ConvLSTM model from [26] illustrating the stacking of 2 ConvLSTM blocks consecutively, similar to deep convolutional networks in, for instance, image classification tasks. Further investigations on next frame predictions with ConvLSTM were done by [2], giving further prove of the applicability of this approach.

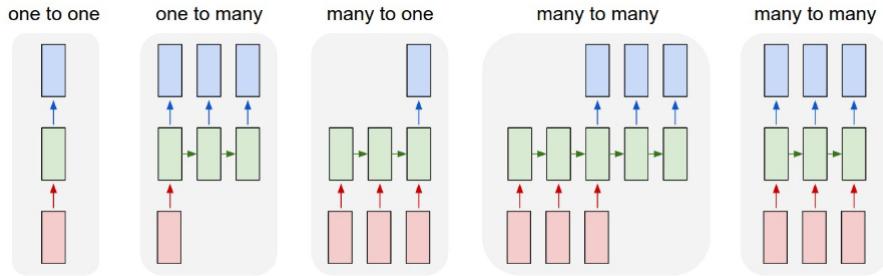


Figure 4.2: LSTM modes. Source: [18]

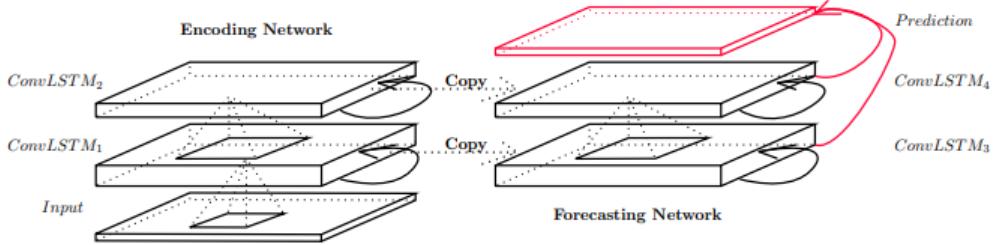


Figure 4.3: Illustration of the ConvLSTM model. Source: [26]

4.1 ConvLSTM Network

The ConvLSTM network is based on the code published in [22]. A couple of adaptions are done to account for the 3D grid space compared to the 2D next-frame video prediction from this example and are as follows:

- The number of filters in the ConvLSTM3D are reduced to 32 from 64, which showed to be enough to learn the features of the objects.
- The kernel size of the last convolution is set to be 5x5x5 instead of 3x3x3, which is too small to cover the rather coarse features.
- In the code basis the last layer is a 3D convolution applied to the sequential image frames from the ConvLSTM2D, which means that this 3D convolutional operation is a convolution over time with the first dimension to be the time axis. Since no 4D convolution is yet available in the tensorflow/keras library to perform the same operation on time frames of 3D objects a 3D convolutional operation was applied instead to the output sequence from the last ConvLSTM3D layer in a frame-wise way. Hence, each frame gets a convolutional operation applied to it, separately.

The ConvLSTM network is depicted in figure 4.4, which takes a sequence of frames of 3D objects as input. The input size is the sequence of all frames N to learn from, except the last one, e.g. if the total sequence is 20, the input sequence consists of the first 19 frames. The core of the network are three stacks of ConvLSTM3D operations. Within these the kernel size

decreases from initially five, to three, to one, consecutively. Each of them have Rectified Linear Unit (ReLU) activation applied to it. Batch normalization is added to the first two ConvLSTM3D layers but not after the third one. The last layer is a Conv3D layer applied to the each individual frame coming out of the last ConvLSTM3D layer. Finally, Sigmoid activation is added, which produces a sequence of output objects with floats between zeros and ones. The output sequence, equal to the input sequence, has a total size of $N - 1$ frames. In case of the target frames to learn from, these are the last $N - 1$ frames from the total sequence. In other words, its a many to many mapping (Figure 4.2) with the input and the output sequence shifted by one frame. The ConvLSTM network consists of 761,889 trainable parameters in total.

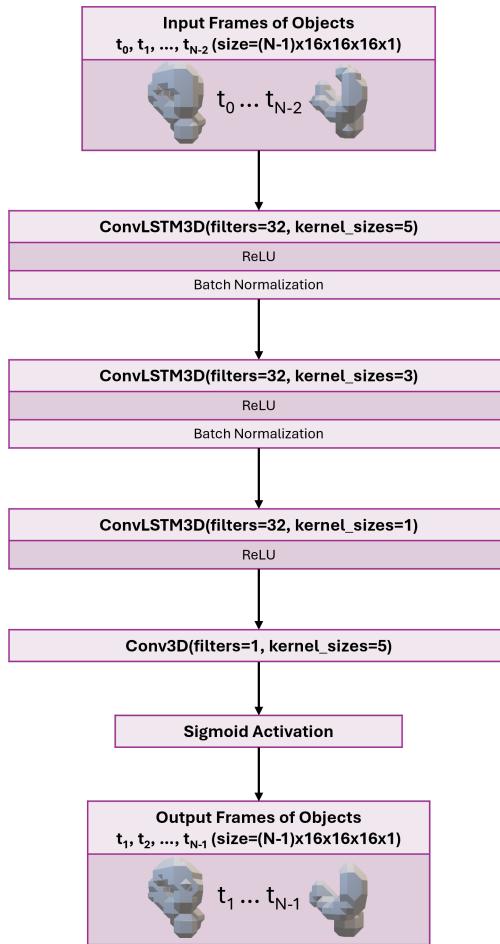


Figure 4.4: ConvLSTM network.

4.2 Loss Function

The loss functions used in the ConvLSTM model is the Binary Cross-Entropy (BCE) loss or Log-Loss. It is ideally suited for grid-based datasets which are described by zeros and ones

only, as in this dataset and defined as

$$L_{BCE} = -\frac{1}{N} \sum_{n=1}^N \left[\mathbf{x}_{t_1} \log \tilde{\mathbf{x}}_{t_1} + (1 - \mathbf{x}_{t_1}) \log(1 - \tilde{\mathbf{x}}_{t_1}) \right]. \quad (1)$$

\mathbf{x}_{t_1} and $\tilde{\mathbf{x}}_{t_1}$ are the next frame 3D tensors of target object and the generated object, respectively. N is the batch size in one training iteration. Therefore, the Binary Cross-Entropy loss per iteration is the mean over all samples in the batch.

4.3 Optimizer Settings

The Adaptive moment estimation (Adam) [20] optimizer is used in all trainings. The optimizer parameters are kept at their default values and are summarized in table 4.1.

Table 4.1: ConvLSTM Optimizer Settings

Optimizer: Adam				
Model	Learning Rate	β_1	β_2	ϵ
ConvLSTM	0.001	0.9	0.999	1e-07

5 GAN-based Predictive Model

For the GAN-based approach two models are built to compare their performance and applicability to next time frame predictions of 3D grid based objects, which are the

- Single Frame Input Model and the
- Multiple Frame Input Model.

Both models are basically the same with the only difference of the input sequence size, which are either 1 past frame or 3 past frames, respectively. The architecture of the neural network of both is based on the 2D Deep Convolutional ResNet by [10] and adapted to the 3D grid space, although a couple of differences are worth mentioning:

- Instead of two consecutive convolutions in the encoder and discriminator and transposed convolutions in the generator in [10] within the so-called Pre-Residual units originally from [12], only one convolution is used, see figure 5.1 (weight = convolutional operation).

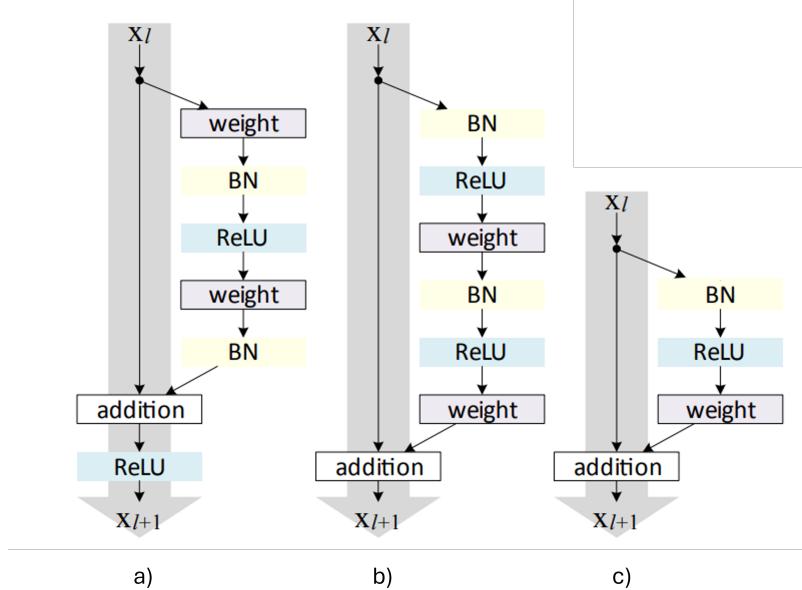


Figure 5.1: a) Residual Unit; b) Pre-Residual Unit; c) Pre-Residual Unit used in the GAN-based models in this work. BN=Batch Normalization; ReLU=Rectified Linear Unit Activation; weight=weights of the layer, e.g. weights of the convolutional layer. Source: [12]

- In [10] the recommended number of training loops for the discriminator is 5, before updating the generator. Nevertheless, for the case of 3D data and slight architecture changes

using 5 iteration loops didn't improve the training outcome compared to using only 1. Hence, for all trainings performed and documented in section 8 the discriminator was kept to be updated also only once per training loop.

- The general architecture was adapted from the approaches by [3] and [21] where the generator consists of a preceding encoder, which encodes the input object or a sequence and a decoder, which corresponds to standard generators in GANs. The progressive growing used in [3] wasn't adapted in the models used in this work. The architecture therefore follows the setup by [21] with the difference that an Auto-Encoder (AE) was used to decode the input object into a latent representation, instead of a Variational Auto-Encoder (VAE), since there isn't any advantage to use a more complex VAE, which would in contrary lead to the inclusion of an additional loss term for the Kullback-Leibler (KL) Divergence, making the training even harder.

Further details about the two models, like the architecture, can be found in sections 5.1 and 5.2.

5.1 Single Frame Input Model

The single frame input model, as the name suggests, only uses one past frame as input to learn and predict from. Figure 5.2 shows the general setup of this model. On the very left one can see the encoder picking up a single object, namely input object at t_0 . This object gets encoded with an AE, in contrary to the approach called EncGAN in [21] where a VAE was used. The encoding results in a latent vector of size 256 as a result of multiple strided convolutional operations. This latent vector deals as the input to the generator. The generator translates this encoding back to an object by multiple consecutive upsampling and transposed convolutions. As found in [21] and adapted here for the 3D grid space a condition for the discriminator is key to learn properly and fast. Tests without such a condition resulted in the network learning very slow, or not the next frame but trying to reproduce the past input frame. In contrary to [21], where the past frame was added as the condition, here it was decided to directly use the target frame at t_1 as the conditional information. Therefore, the generated object at frame t_1 coming from the generator is concatenated with the target object at frame t_1 and fed into the discriminator as the "fake" object. At the same time the target object at frame t_1 is concatenated with itself in this case. Hence, the "real" input to the discriminator is twice the target object at frame t_1 . The outcome of the discriminator for both "real" and "fake" is then fed into the Wasserstein-GAN with Gradient Penalty (WGAN-GP) loss function to be minimized, see section 5.4. For details of the encoder, generator and discriminator architecture, see section 5.3.

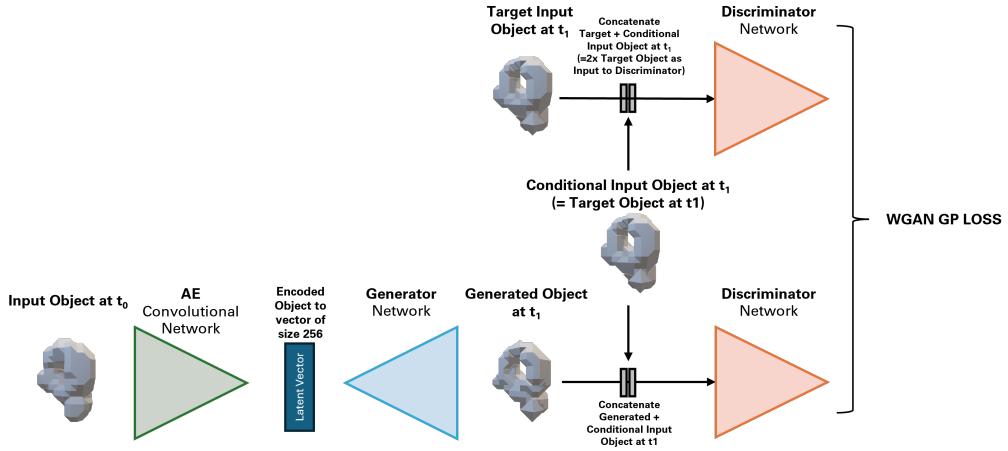


Figure 5.2: Single Frame Input Model.

5.2 Multiple Frame Input Model

The multiple frame input model is build analogous to the single frame input model. A condition for the discriminator is also key in this setup for the model to learn properly. The only difference is that three past frames t_{-2} , t_{-1} and t_0 are used as input to predict the next frame at t_1 . Each of these three objects are going through the same encoder separately, producing three different latent encoded representations, respectively. Hence, weights are shared by the encoder for all three past frames. The encodings are then concatenated to a latent vector of size $3 * 256 = 768$ and fed in to the generator. All following steps and the rest of the model stays the same as in the single frame input model, see section 5.1.

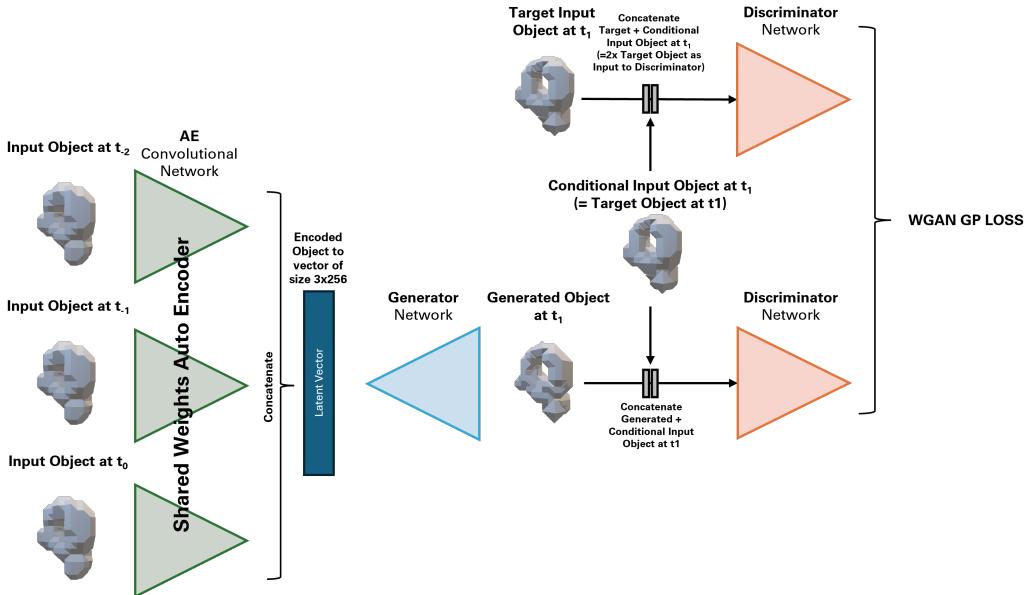


Figure 5.3: Multiple Frame Input Model.

5.3 Encoder, Generator and Discriminator Networks

All three networks have common settings which are as follows:

- Networks are constructed to work on an object grid space dimension of 16x16x16x1, where the last dimension is the channel. In case of images this dimension would correspond to the RGB color values. Since we don't have colors, but only 0 or 1 at each position in the grid, indicating a construction point of the 3D geometry, there is only 1 channel. The encoder and the discriminator expect inputs of this size and the generator produces objects of this size.
- The minimum size, or base size, is 4x4x4. This is the size from which the generator starts to increase the size step wise, using pairs of upsampling (doubling the size) and transposed convolutions (with stride=1), to finally result in the original size 16x16x16. A resolution of 2x2x2 was also taken under consideration but didn't learn anything, since the resolution was too small to start from to learn any features for the objects from the dataset. Also, a resolution of 8x8x8 was tested but didn't learn the next frame and got stuck to reproduce the input frame. The probable cause for that is that the networks are less deep with only one upsampling operation to be necessary to end up at the original size of 16x16x16.
- The kernel size of the 3D convolutional operations is the same in all networks and is chosen to be 5x5x5. Higher and lower sizes were found to learn slower.
- L1 regularization and dropout of 0.2 is added in all networks to prevent overfitting and improve generalization.
- Glorot-Uniform initialization is used for all weights in the networks.
- Bias is used in all networks and initialized with zeros.

5.3.1 Encoder

The encoder network is depicted in figure 5.4. It takes a single object of size $16 \times 16 \times 16 \times 1$ as input and consists in total of 4 convolutional operations and 9,620,416 trainable parameters. The output results in 256 units in case of the single frame input model and 768 units for the multiple frame input model, respectively, representing the encoded objects.

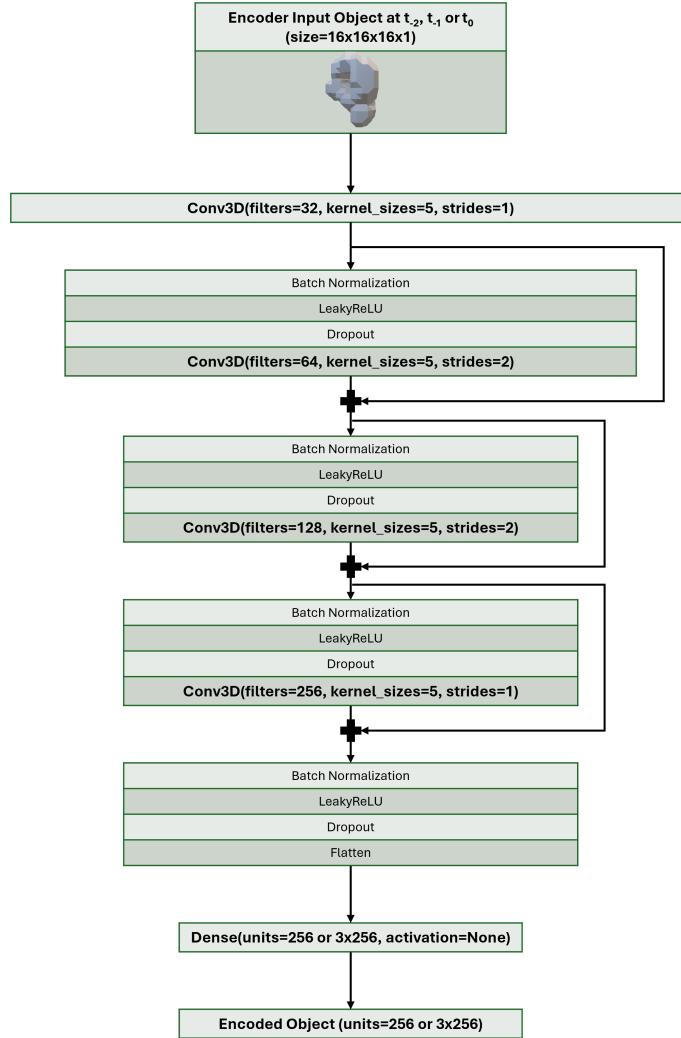


Figure 5.4: Encoder network.

5.3.2 Generator

The generator network is depicted in figure 5.5. Depending on the model used, the generator takes an encoded object of size 256 or 768 as input. The final convolution has a single filter leading to the corresponding output object's size of $16 \times 16 \times 16 \times 1$ and is followed by a Sigmoid activation, which produces outputs between zeros and ones. The generator network of the Single and Multiple Frame Input Model consists of 9,636,065 and 18,024,673 trainable parameters,

respectively.

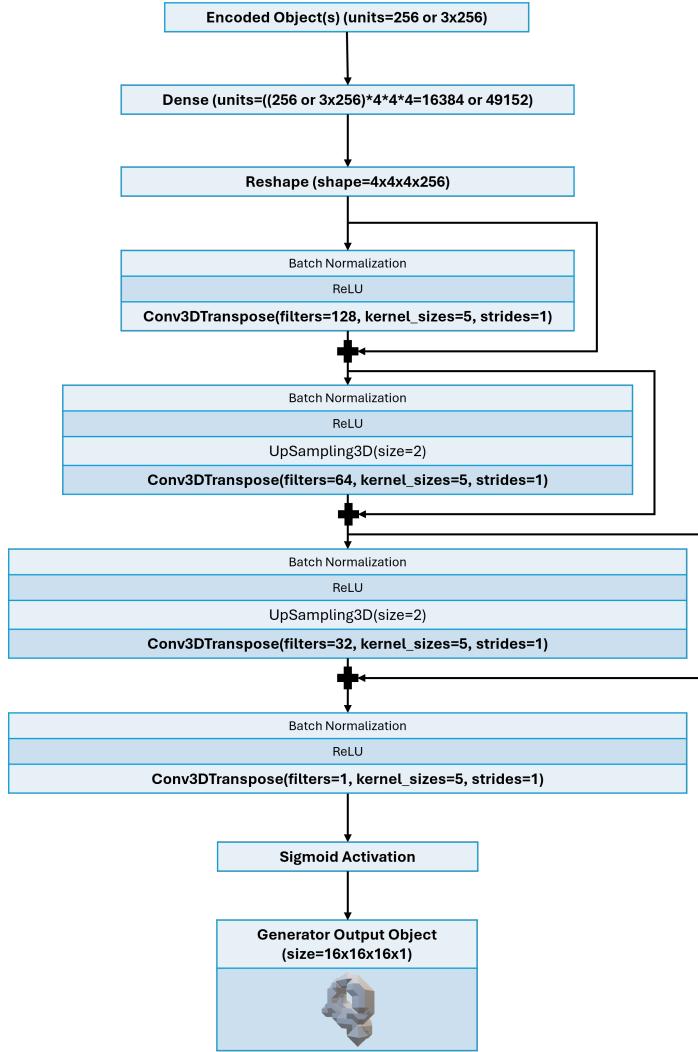


Figure 5.5: Generator network.

5.3.3 Discriminator

The discriminator network is depicted in figure 5.6. The discriminator takes as input a concatenated pair of the conditional input, which is always the next frame object at t_1 , and the target or generated object from the generator. The output is a dense layer with one unit, without an activation function, corresponding to the discriminators score of the generated or target object. The discriminator network consists of 5,445,281 trainable parameters in total.

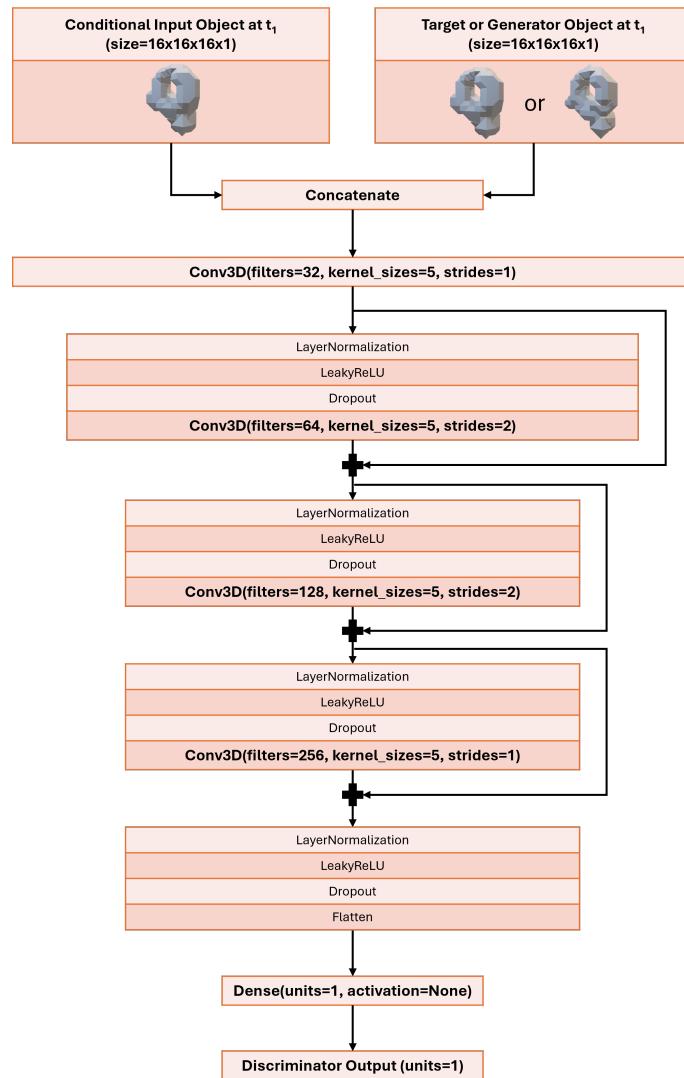


Figure 5.6: Discriminator network.

5.4 Loss Function

The loss in GANs is a minimax game between the generator and the discriminator. The generator learns to produce an object as close to the target as possible and the discriminator tries to distinguish if the object is fake (coming from the generator) or actually a real object (target object, coming from the dataset). During the training the discriminator always has to be evaluated twice per (batch) loop. Once for the generated object coming from the generator, producing the generated object loss (fake loss) value and another time running the real object through it, resulting in the target object loss value (real loss). The final loss calculated from these losses depends on the loss function that is used. Loss functions for GANs is a research topic since the very first GAN paper [8] came out. The reason for that is that the original GAN loss is not stable in training. Training mostly suffers from an unbalance when the discriminator learns faster than the generator, or vice versa, or converging not at all. Additionally, so-called Mode Collapse might occur where the generator will only learn a subset of objects. Different approaches for new loss functions came up to counter these problems, but after all, the improved WGAN-GP from [10] has become established in the world of GANs, where a gradient penalty is added to circumvent exploding or vanishing gradients from the original Wasserstein GAN [4]. The WGAN-GP was adapted for the 3D case and used for all GAN-based trainings. The loss for the discriminator (called critic in the Wasserstein papers [4], [10], because its not classifying anymore, but can be thought of a score) looks as follows, including the epsilon penalty term from [3]

$$L_D = \underbrace{\mathbb{E}_{\mathbf{x} \sim \mathbb{P}_r} [D(\mathbf{x})] - \mathbb{E}_{\tilde{\mathbf{x}} \sim \mathbb{P}_g} [D(\tilde{\mathbf{x}})]}_{WGAN \text{ loss}} + \underbrace{\lambda \mathbb{E}_{\hat{\mathbf{x}} \sim \mathbb{P}_{\hat{\mathbf{x}}}} [(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)]}_{gradient \text{ penalty}} + \underbrace{\epsilon \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_r} [D(\mathbf{x})^2]}_{epsilon \text{ penalty}}, \quad (2)$$

where \mathbb{P}_r is the distribution of the data, \mathbb{P}_g is the distribution of the generated data implicitly defined by $\tilde{\mathbf{x}} = G(\mathbf{z})$, $\tilde{\mathbf{x}} \sim p(\tilde{\mathbf{x}})$, λ is the gradient penalty term, $\|\dots\|_2$ is the norm and ϵ is the epsilon penalty, as in [3]. Since a preceding AE is used to utilize a GAN for next frame predictions, \mathbf{z} isn't drawn from a distribution anymore, like in standard GANs, but is now

$$\mathbf{z}_{t-1} = E(\mathbf{x}_{t-1}) \quad \text{or} \quad \mathbf{z}_{t-2,t-1,t_0} = E(\mathbf{x}_{t-2}) \| E(\mathbf{x}_{t-1}) \| E(\mathbf{x}_{t_0}), \quad (3)$$

where $E(\cdot)$ denotes the encoder model and $\|$ the concatenation operation. Correspondingly, the loss for the generator to be minimized and learned from looks as follows:

$$L_G = - \mathbb{E}_{\tilde{\mathbf{x}} \sim \mathbb{P}_g} [D(\tilde{\mathbf{x}})] \quad (4)$$

It's worth mentioning that the gradient penalty is set to 100 in all trainings, instead of the default value of 10 proposed in [10], which lead to faster training. Also, the epsilon penalty

was implemented in the algorithms B.1 and B.5, but is set to zero, which lead to more stable convergence during training.

5.5 Optimizer settings

The optimizer used is the Adam [20] optimizer and it's settings are the same as within [10]. Same settings are used for the Encoder, Generator and Discriminator and are summarized in table 5.1.

Table 5.1: GAN Optimizer Settings

Optimizer: Adam					
Model	Learning Rate	β_1	β_2	ϵ	
Encoder, Generator and Discriminator	0.0001	0.5	0.999	1e-07	

6 Quality Metrics for Predicted Objects

To quantify the quality of the generated objects two metrics have been implemented in both approaches for next time frame predictions, which are the Mean-Squared-Error and Jaccard Distance. Both are described in the upcoming sections 6.1 and 6.2.

6.1 Mean Squared Error

The Mean-Squared-Error (MSE) is a commonly used metric for comparison of n-dimensional tensors. It simply is the mean of the sum of squared difference of every pairwise entry in \mathbf{x}_{t_1} and $\tilde{\mathbf{x}}_{t_1}$, which are the next frame 3D tensors of target object and the generated object, respectively. The closer to zero the MSE is the more both objects look alike.

$$\text{MSE}(\mathbf{x}_{t_1}, \tilde{\mathbf{x}}_{t_1}) = \frac{1}{n} \sum_{i=1}^n (x_{i,t_1} - \tilde{x}_{i,t_1})^2. \quad (5)$$

6.2 Jaccard Distance

The Jaccard Index [16] is used to compare the similarity between two sample sets and is defined as the division of the size of the intersection and the size of the union and can mathematically be expressed for two sample sets A and B as

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}. \quad (6)$$

The Jaccard Index would yield one for two perfectly aligned sample sets and zero in case of no overlap. Its also known as Intersection over Union (IoU) and commonly used in object localization and detection tasks in computer vision. To express the diminishing difference during learning, one would like to express it in terms of an error. In this case the so-called Jaccard Distance was implemented and can be expressed as

$$d_J(\mathbf{x}_{t_1}, \tilde{\mathbf{x}}_{t_1}) = 1 - J(\mathbf{x}_{t_1}, \tilde{\mathbf{x}}_{t_1}) = \frac{|\mathbf{x}_{t_1} \cup \tilde{\mathbf{x}}_{t_1}| - |\mathbf{x}_{t_1} \cap \tilde{\mathbf{x}}_{t_1}|}{|\mathbf{x}_{t_1} \cup \tilde{\mathbf{x}}_{t_1}|}. \quad (7)$$

A Jaccard Distance of zero would mean a perfect match between the next frame 3D tensors of target object and the generated object \mathbf{x}_{t_1} and $\tilde{\mathbf{x}}_{t_1}$, respectively. The implementation is a bit-wise comparison between the target and the generated object. Therefore, since the generated object coming out of the generator network, or ConvLSTM network contains only float values

between zero and one, it's converted to binary format of clean zeros and ones by simply applying a threshold of 0.5 to the 3D grid data of size 16x16x16x1, like it's done for the reconstruction of the objects in section 3. Hence, values below 0.5 are set to zero and values above are one.

6.3 Plausibility Thresholds for Model Quality Assessment

The error against the target is the standard error used to quantify the closeness or similarity of the predicted object to the real next frame 3D object:

- Errors against the Target \mathbf{x}_{t_1}

$$\text{MSE}(\mathbf{x}_{t_1}, \tilde{\mathbf{x}}_{t_1}) \quad (8)$$

$$d_J(\mathbf{x}_{t_1}, \tilde{\mathbf{x}}_{t_1}) \quad (9)$$

Since the objects can be very similar between 2 frames and visually not that easy to be separated, an additional error measure, like testing against the input frame is helpful. Comparing these two errors tells if the predicted object is closer to the input or target.

- Errors against the Input \mathbf{x}_{t_0}

$$\text{MSE}(\mathbf{x}_{t_0}, \tilde{\mathbf{x}}_{t_1}) \quad (10)$$

$$d_J(\mathbf{x}_{t_0}, \tilde{\mathbf{x}}_{t_1}) \quad (11)$$

A lower error for the test against the target than to the input means that the object predicted is actually closer to the next frame than to the last input object frame. This deals as an indicator if the model was able to learn the transition from one frame to the next.

Additionally, another so-called Null Model was implemented to compare the target prediction error also against it. The Null Model is simply a fictional model that will just produce a perfect last frame 3D object, where $\tilde{\mathbf{x}}_{t_0} = \mathbf{x}_{t_0}$. This last frame is now evaluated against the target frame (next frame):

- Null Model

$$\text{MSE}(\mathbf{x}_{t_1}, \mathbf{x}_{t_0}) \quad (12)$$

$$d_J(\mathbf{x}_{t_1}, \mathbf{x}_{t_0}) \quad (13)$$

In other words, the error of this fictional model evaluates the inter-temporal variability between two frames. This error reflects the maximum error of one frame to the next and the actual prediction error against the target must not be larger than this error, otherwise the model can be categorized as not sufficiently accurate to be used as a predictive model. Overall, the error of a model must be below these two thresholds, the error against the input and the Null Model error, to be classified as a valid model for next frame(s) predictions.

7 Results of the ConvLSTM-based Model

The following sections show the training performance and results of the ConvLSTM-based prediction models for a single next 3D frame predictions and long-term 3D frame predictions. Three models are trained and tested upon their prediction performance, where each is fed with the three different data formats as described in section 2, including the evaluation using the two quality metrics from section 6.

7.1 Training Performance

All three training runs were performed with the setup described in section 4. The training dataset is a subset of 10000 data points out of the total 32000, consisting of pairs of sequences of 19 input frames and 19 target frames. For the validation during training 1000 data points from the formerly split dataset are used. A batch size of 16 is used in each training iteration and all trainings are run for 10000 iterations (processed batches).

Figure 7.1 shows the BCE loss, MSE loss and Jaccard Distance Loss during training for 10000 iterations for the training and validation set, where all are converging to a specific level.

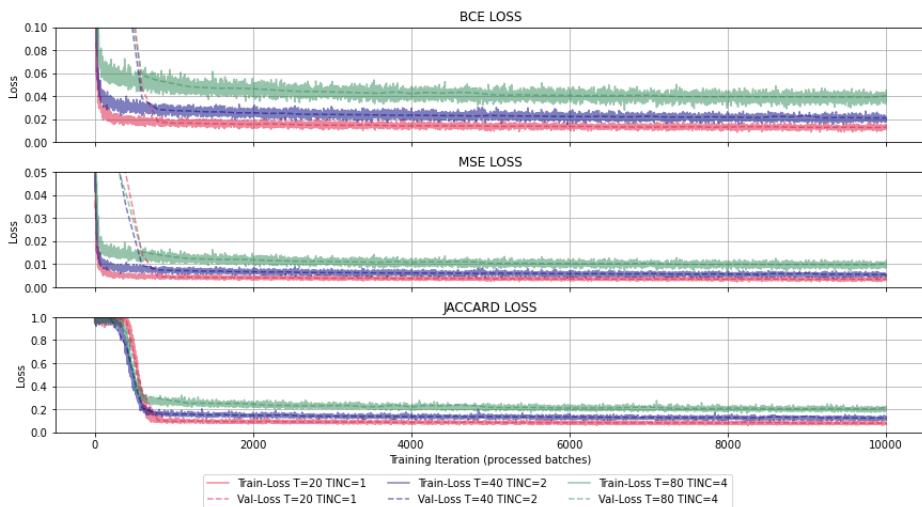


Figure 7.1: Training and validation loss of the BCE, MSE and Jaccard Distance.

One can observe that the Jaccard Distance loss starts decreasing only after a couple hundred simulations, which speaks for this metric compared to the two other, since these are kind of smeared errors and not comparing actual similarity as the Jaccard Distance. Therefore, already significantly dropped BCE and MSE losses don't mean that the objects are actually already that

similar. Although, for the Jaccard Distance, low values indicate similarity per se, which are within a range of 0.08 and 0.2, depending on the underlying frame step (TINC) used for extracting 20 training frames.

7.2 Next Frame Prediction Quality

Figure 7.2 shows the histograms and boxplots of the errors described in section 6 on 1000 test-set datapoints of the ConvLSTM Model.

Since the ConvLSTM models are processing an input sequence $\mathbf{x}_{t_0} \dots \mathbf{x}_{t_{19}}$ to predict the next sequence $\mathbf{x}_{t_1} \dots \mathbf{x}_{t_{20}}$, instead of a single value, the generally valid equations from section 6.3 need to be adopted. Therefore, the errors in the figures represent the mean over a predicted sequence:

$$\frac{1}{n} \sum_{i=1}^n \text{MSE}(\mathbf{x}_{t_1} \dots \mathbf{x}_{t_{20}}, \tilde{\mathbf{x}}_{t_1} \dots \tilde{\mathbf{x}}_{t_{20}})_i \quad \text{and} \quad \frac{1}{n} \sum_{i=1}^n d_J(\mathbf{x}_{t_1} \dots \mathbf{x}_{t_{20}}, \tilde{\mathbf{x}}_{t_1} \dots \tilde{\mathbf{x}}_{t_{20}})_i \quad (14)$$

Consequently, the MSE and Jaccard Distance of the prediction to input and the Null Model for the ConvLSTM model are calculated analogous to section 6.3 by:

- Errors against the Input $\mathbf{x}_{t_0} \dots \mathbf{x}_{t_{19}}$

$$\frac{1}{n} \sum_{i=1}^n \text{MSE}(\mathbf{x}_{t_0} \dots \mathbf{x}_{t_{19}}, \tilde{\mathbf{x}}_{t_1} \dots \tilde{\mathbf{x}}_{t_{20}})_i \quad \text{and} \quad \frac{1}{n} \sum_{i=1}^n d_J(\mathbf{x}_{t_0} \dots \mathbf{x}_{t_{19}}, \tilde{\mathbf{x}}_{t_1} \dots \tilde{\mathbf{x}}_{t_{20}})_i \quad (15)$$

- Null Model

$$\frac{1}{n} \sum_{i=1}^n \text{MSE}(\mathbf{x}_{t_1} \dots \mathbf{x}_{t_{20}}, \mathbf{x}_{t_0} \dots \mathbf{x}_{t_{19}})_i \quad \text{and} \quad \frac{1}{n} \sum_{i=1}^n d_J(\mathbf{x}_{t_1} \dots \mathbf{x}_{t_{20}}, \mathbf{x}_{t_0} \dots \mathbf{x}_{t_{19}})_i \quad (16)$$

Figure 7.2 shows the histogram and boxplot of the errors described in section 5 on 1000 test-set datapoints of the ConvLSTM Model. The model performs well in all setups. Models with data formats of larger change between frames (T=80, TINC=4) perform significantly below the mean of the input to prediction and Null Model thresholds, indicating that the model has learned to predict the next 3D object frame with a mean MSE of 0.01 and a mean Jaccard Distance of 0.188. Models that used data formats of low change from frame to frame have although reached much lower values of 0.0035 MSE and 0.0743 Jaccard Distance, but struggle to separate from the input frame.

Exemplary next frame sequences of 3D object predictions up to 5 time frames are depicted in Figure 7.3 showing good optical similarities between the prediction and the target supporting the numbers from the histogram and boxplot.

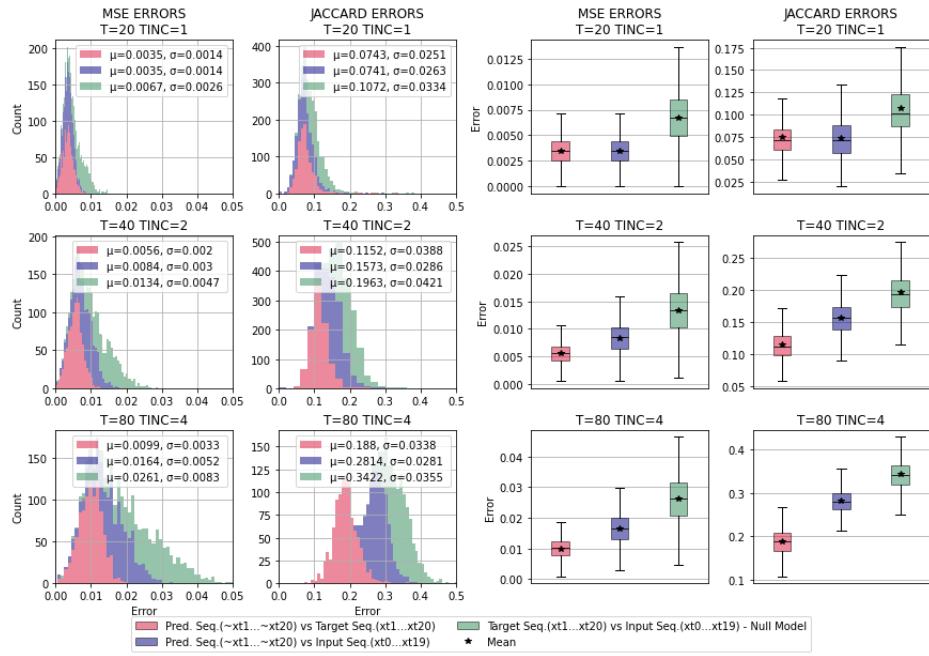


Figure 7.2: ConvLSTM Model - Histogram and boxplot of the MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T=20-INC=1, T=40-INC=2 and T=80-INC=4.

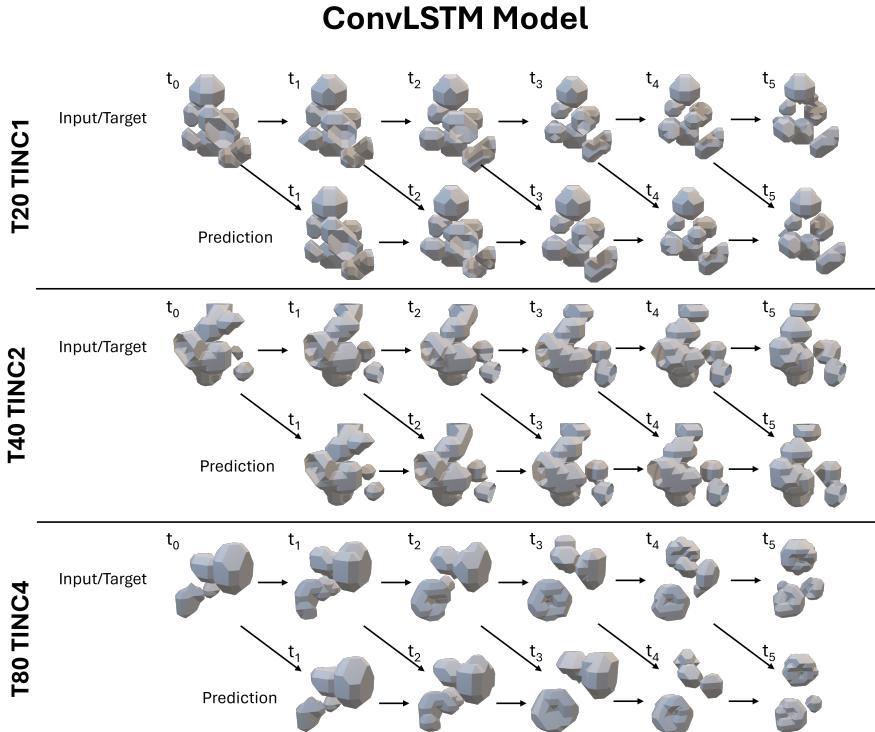


Figure 7.3: ConvLSTM Model - Exemplary generated predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

7.3 Long-Term Prediction Quality

The first 5 frames are used as input sequence $x_{t_0} \dots x_{t_5}$ to predict the one-step ahead sequence $\tilde{x}_{t_1} \dots \tilde{x}_{t_6}$. From this prediction the last frame \tilde{x}_{t_6} is concatenated to the former input frames to predict the next frame with the newly updated input sequence $x_{t_1} \dots x_{t_5}, \tilde{x}_{t_6}$ and so on and so forth. This is done within a loop 5 times in total. In other words, the prediction deals as the input in the next forecasting step. Therefore, the better the prediction is the better the model should be in predicting the next frame over and over. If the prediction is already inaccurate, all the the next predictions will be even worse. Hence, errors are propagated through time.

Figure 7.4 shows the mean MSE and Jaccard Distance errors including the bands of the standard deviation for the ConvLSTM Model. As observed in the single next frame predictions the models trained on larger frame to frame step data formats perform better also here. However, only after two forecast time frames the the mean of the thresholds of prediction to input and the Null Model are already exceeded. From the second time frame on the prediction to target and prediction to input errors are on bar, which means that the predicted object is as different, equal or similar to the same extend to the input and the target.

The green line, representing the Null Model, is basically a straight line, because the inter-temporal variability between two frames stays more or less constant over time in the dataset. Figure 7.5 shows exemplary picks of the long-term predictions for the three data formats. Optically it can be concluded that the ConvLSTM model does a good job in long-term predictions. Up to 5 time frames ahead the predicted objects are clearly similar to the targets, although the numbers in the histogram and boxplots indicate larger deviations to be expected.

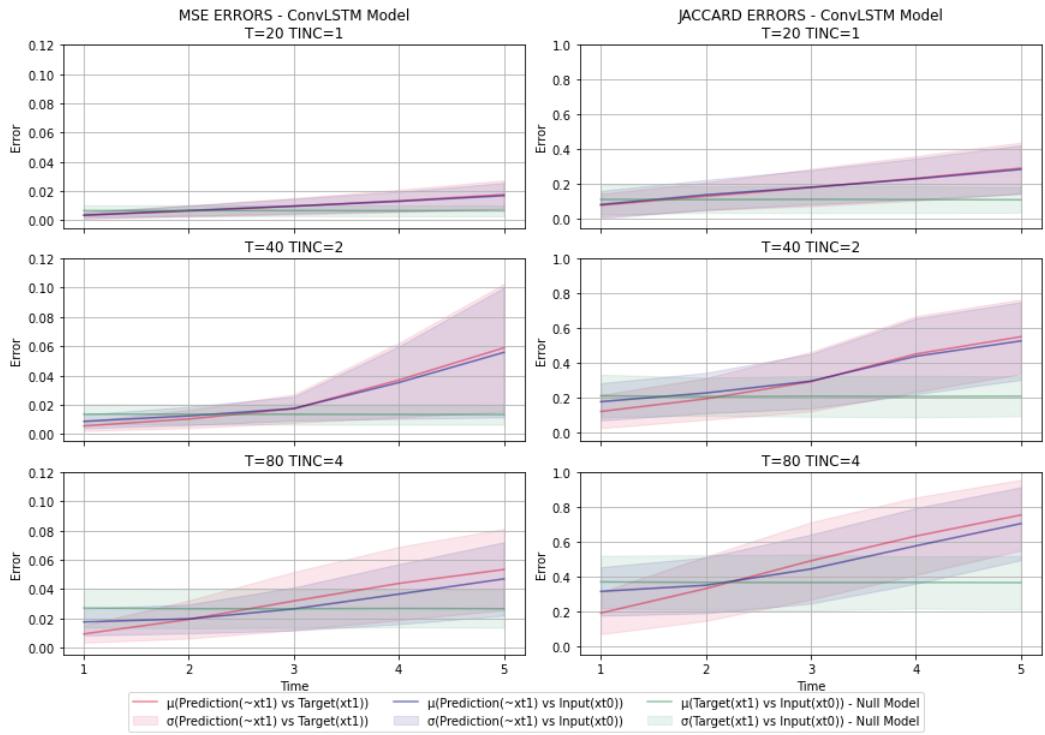


Figure 7.4: ConvLSTM Model - Long-term MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T=20-INC=1, T=40-INC=2 and T=80-INC=4.

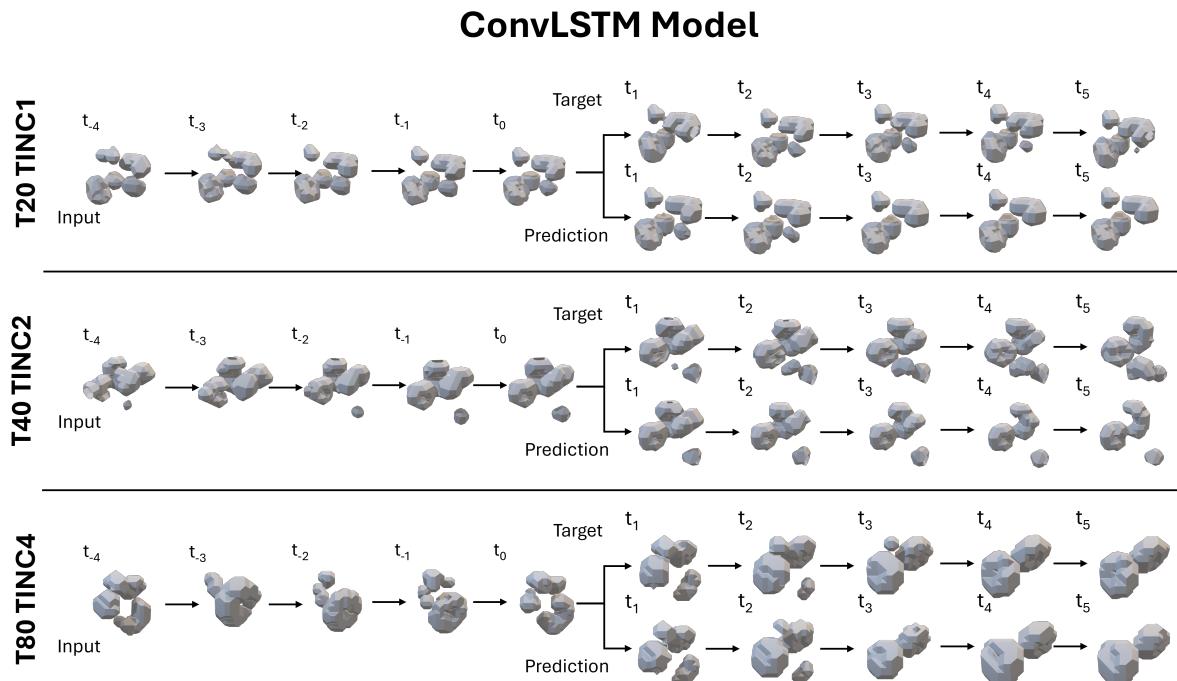


Figure 7.5: ConvLSTM Model - Exemplary generated long-term predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

8 Results of the GAN-based Model

The following sections show the training performance and results of the GAN-based models for a single next 3D frame prediction and long-term 3D frame predictions. In total, 6 models are trained and tested with the GAN-based approach upon their prediction performance. These are the two base models, the single and multiple frame input models, where each is fed with three different data formats as described in section 2, including the evaluation using the two quality metrics from section 6.

8.1 Training Performance

All 6 training runs were performed with the setup described in section 5. The training dataset is a subset of 10000 data points out of the total 32000. For the validation during training 1000 data points from the formerly split dataset are used. A batch size of 32 is used in each training iteration and all trainings are run for 20000 iterations (processed batches).

Figure 8.1 shows the generator and discriminator loss during training for 20000 iterations. As discussed in [4] and [10] the generator loss isn't really interpretable, in contrary to the discriminator (or critic) loss, which corresponds to the quality of an image or object and is naturally converging to a specific value. Generally, the lower the (negative) discriminator (or critic) loss is the better the quality of the object. In all trainings the discriminator converges to a value of about 5. Furthermore, one can already see the effects of the three different frame split setups whereby the larger the frame step (TINC) is the more noisy the training becomes.

Figure 8.2 shows the MSE and Jaccard Distance during training for 20000 iterations for the training and validation set. All trainings converge to a specific validation error level, where one can see that the single frame input model delivers worse training and validation errors (MSE: $0.01 < \text{Error} < 0.025$; Jaccard: $0.2 < \text{Error} < 0.4$) than the multiple frame input model (MSE: $0.01 < \text{Error} < 0.015$; Jaccard: $0.18 < \text{Error} < 0.3$). In the Multiple Frame Input Model an offset of the Jaccard validation error to the training error can be observed, showing signs of overfitting.

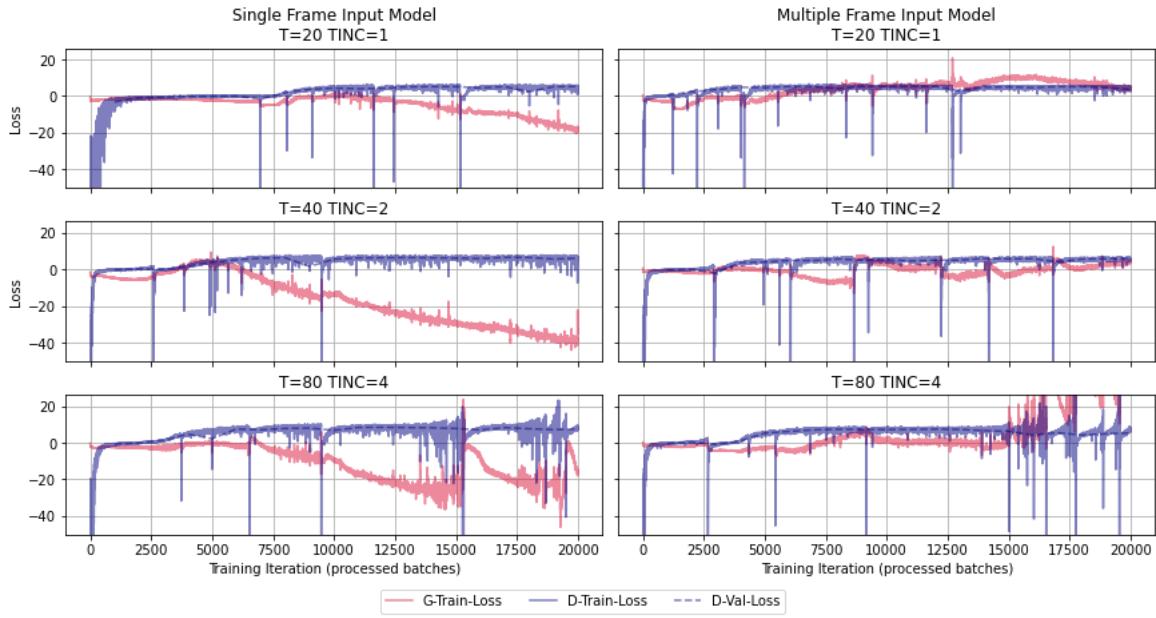


Figure 8.1: Training and validation loss of the Generator and Discriminator.

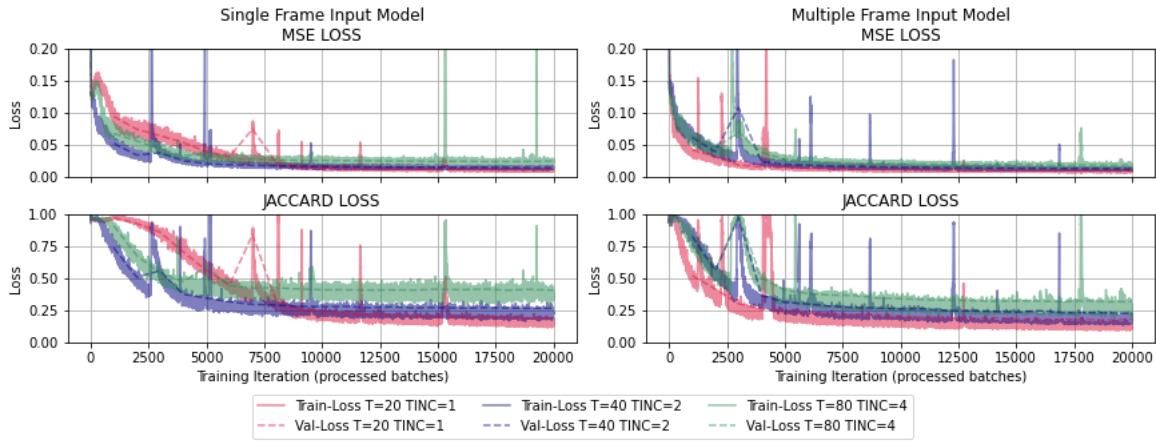


Figure 8.2: Training and validation loss of the MSE and Jaccard Distance.

8.2 Next Frame Prediction Quality

Figure 8.3 shows the histograms and boxplots of the errors described in section 6 on 1000 test-set datapoints of the Single Frame Input Model.

It can be observed that the model delivers a poor performance regarding the relative comparison of the prediction to target error against the two thresholds (see section 6.3), the error against the input and the Null Model, respectively. Exemplary next frame 3D object predictions are depicted in Figure 8.4 where a strong similarity between the input object and the prediction can be observed. This indicates that the errors between prediction and input must be low, which can be easily seen in the histogram and boxplots too. Especially in the T=80, TINC=4 model, the distance of the mean of the prediction to target and prediction to input is high, which means that the frame step width is too large to learn for this model and it gets stuck in reproducing the input frame. Therefore, it can be concluded that Single Frame Input Model isn't a valid predictive model.

Looking only at the mean prediction to target errors, the model quality ranges between 0.0105 and 0.0235 for the MSE and between 0.1839 and 0.4047 for the Jaccard Distance.

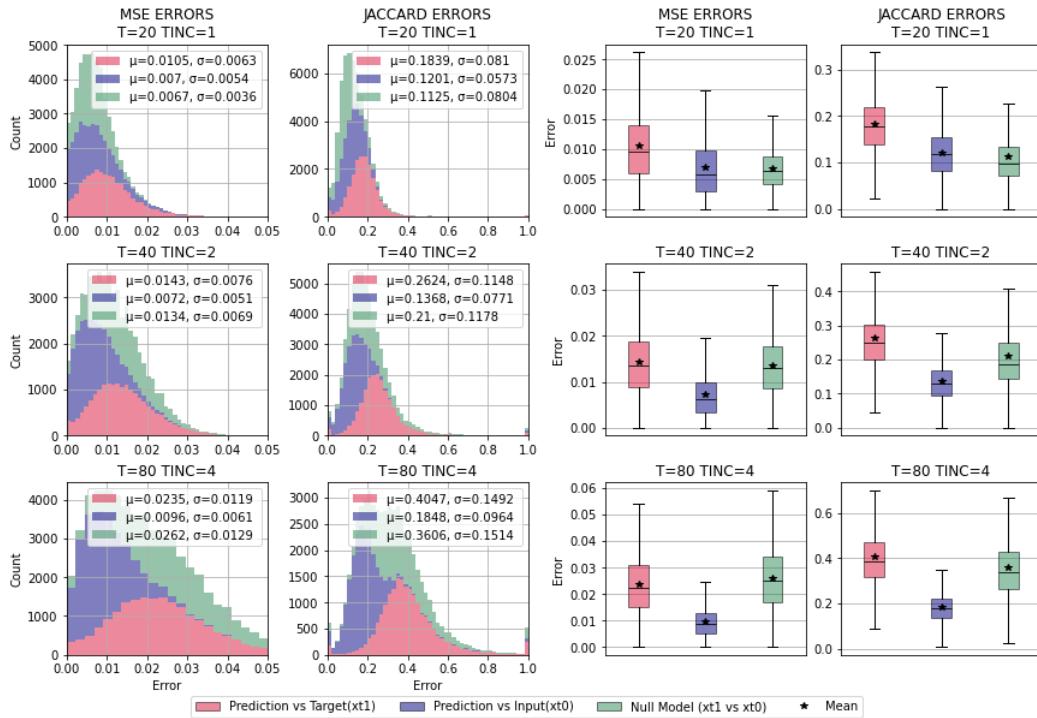


Figure 8.3: Single Frame Input Model - Histogram and Boxplot of the MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

Single Frame Input Model

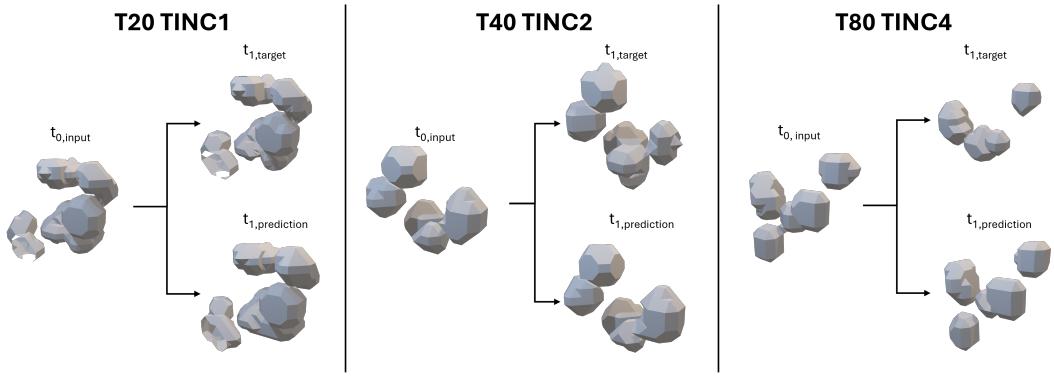


Figure 8.4: Single Frame Input Model - Exemplary generated predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

Figure 8.5 shows the histograms and boxplots of the errors described in section 6 on 1000 test-set datapoints of the Multiple Frame Input Model. Compared to the Single Frame Input Model the qualities are overall better, but only the $T=80$, $TINC=4$ model MSE and Jaccard Distance errors are below the thresholds of the prediction to input and Null Model. Exemplary next frame 3D object predictions are depicted in Figure 8.6 where again a similarity between the input object and the prediction can be observed. Therefore, it can be concluded that Multiple Frame Input Model performs better than the Single Frame Input Model, but only the $T=80$, $TINC=4$ model fulfills the thresholds. Although, this is the model with lowest quality in predictions of all with a mean MSE of 0.0177 and a Jaccard Distance of 0.3137.

Overall, the mean prediction to target errors, the model quality ranges between 0.0096 and 0.0177 for the MSE and between 0.171 and 0.3137 for the Jaccard Distance.

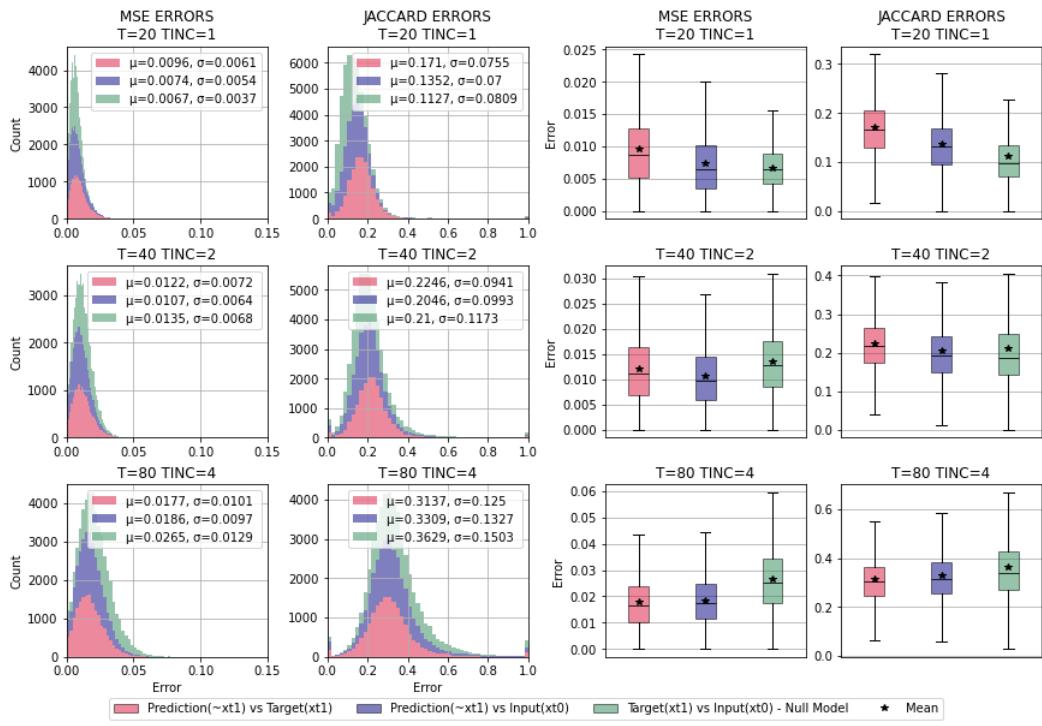


Figure 8.5: Multiple Frame Input Model - Histogram and Boxplot of the MSE and Jaccard Distance between prediction-target, prediction-input and the null model for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

Multiple Frame Input Model

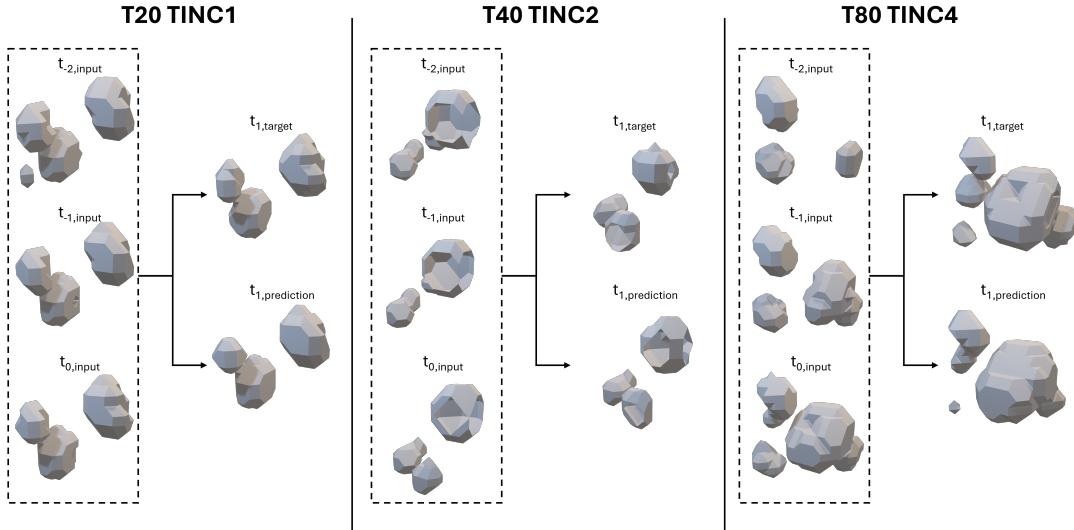


Figure 8.6: Multiple Frame Input Model - Exemplary generated predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

8.3 Long-Term Prediction Quality

Apart from single next frame predictions, long-term predictions on the two GAN-based model are also tested. In this setup, the first input is x_{t_0} to predict \tilde{x}_{t_1} , followed by a loop subsequently setting $\tilde{x}_{t_1} = \tilde{x}_{t_0} = x_{t_0}$ 5 times in total. In other words, the prediction deals as the input in the next forecasting step. Consequently, the better the prediction is the better the model should be in predicting the next frame over and over. If the prediction is already inaccurate, all the the next predictions will be even worse. Therefore, errors are propagated through time.

Figure 8.7 shows the mean MSE and Jaccard Distance errors including the bands of the standard deviation for the Single Frame Input Model. It's clearly observable that the prediction to target errors are not only growing per time step, but are also always above the mean of the thresholds of prediction to input and the Null Model. The green line, representing the Null Model, is basically a straight line, because the inter-temporal variability between two frames stays more or less constant over time in the dataset.

Figure 8.8 shows exemplary picks of the long-term predictions for the three data formats. The same issue as with the single next frame predictions can be observed that already the first step prediction basically reproduces the input frame. Following steps are just getting "stuck" and reproduce the same frame over and over, without barely any change over time. Therefore, it's obvious that the Single Frame Input Model is neither suitable for single next frame predictions, nor for long-term predictions.

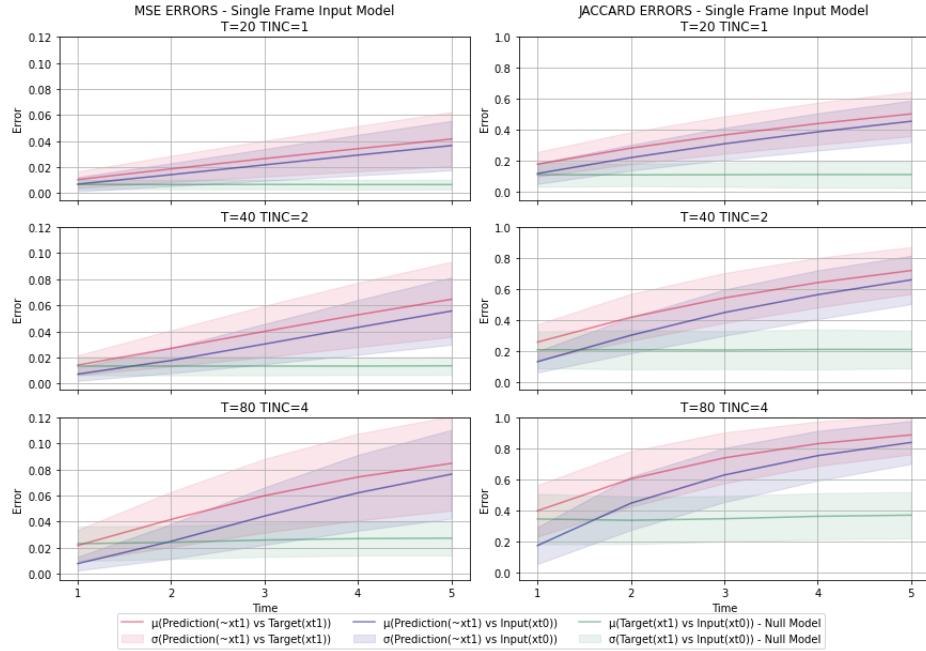


Figure 8.7: Single Frame Input Model - Long-term MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

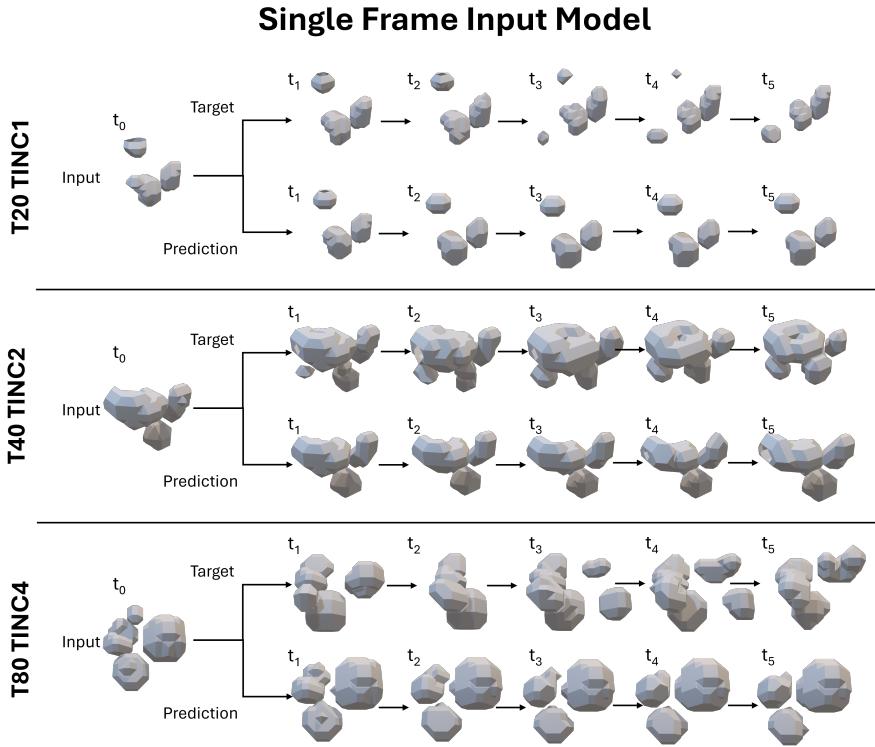


Figure 8.8: Single Frame Input Model - Exemplary generated long-term predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

The Multiple Frame Input Model long-term prediction performance is depicted in figure 8.9. Similarly to the Single Frame Input Model the errors are growing with each step and not even the second time step can fulfill the mean of the thresholds of prediction to input and Null Model. The exemplary long-term predictions in figure 8.10 although are showing a change along time frames compared to the input, but not following the target predictions. The error in the first step leads to predictions to take a wrong turn in the forecasting.

Single next frame predictions with Multiple Frame Input Model are learned at least for the $T=80$, $TINC=4$, but further steps are already far from an acceptable accuracy and therefore not suitable for predictions longer than one step ahead.

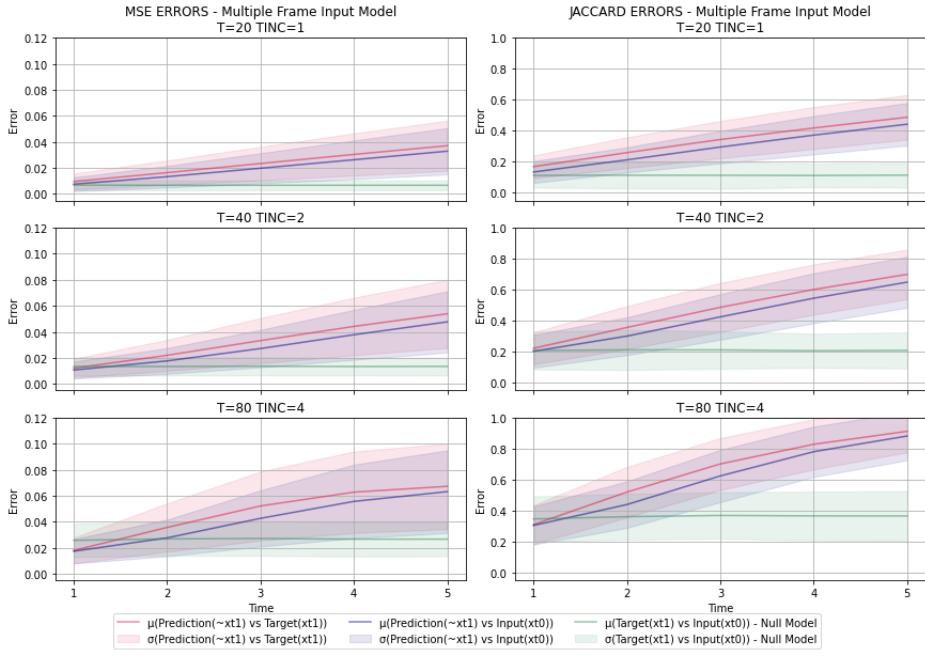


Figure 8.9: Multiple Frame Input Model - Long-term MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

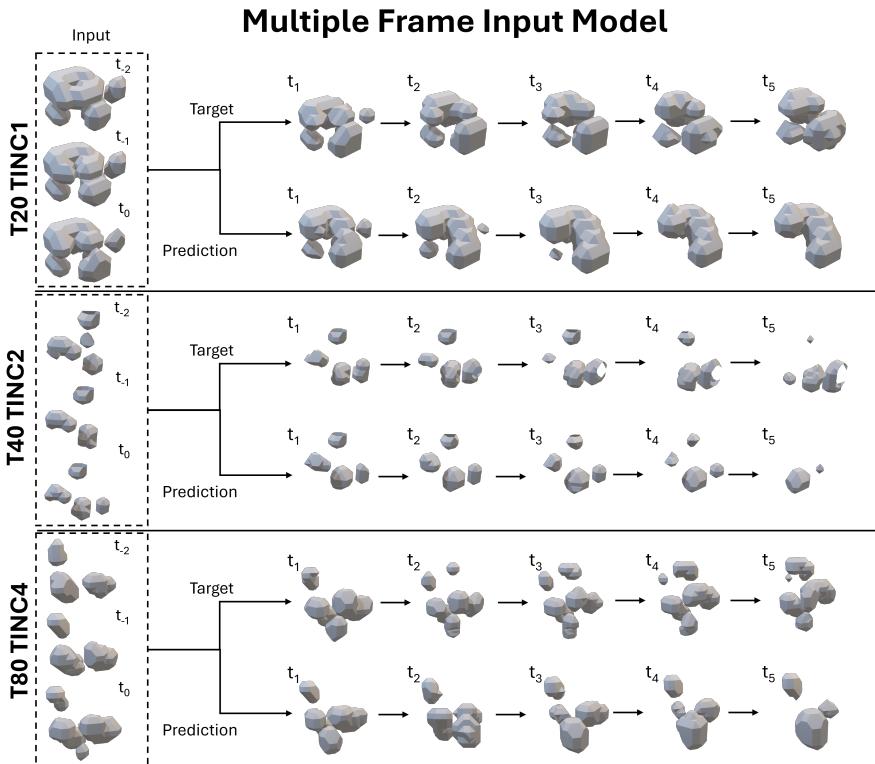


Figure 8.10: Multiple Frame Input Model - Exemplary generated long-term predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

9 Result Comparison of All Models

Figure 9.1 and figure 9.2 compare the prediction performance of all 9 trained models, which are the

- ConvLSTM model and the
- GAN-based models, namely
 - Single Frame Input Model and
 - Multiple Frame Input Model.

Each of them were trained on dataset formats of

- $T=20$, $TINC=1$,
- $T=40$, $TINC=2$ and
- $T=80$, $TINC=4$.

For single next frame predictions (Figure 9.1) the ConvLSTM model clearly outperforms the GAN-based models in all setups. Not only that they are below the Null Model threshold, but also better than the prediction to input error based on the mean value. The GAN-based models are failing mostly against prediction to input error indicating that the prediction is closer to the input than to the target, therefore barely learning the transition from the last to the next frame. Furthermore, looking at the quantiles in the boxplot the ConvLSTM spread of predicted objects is much smaller than the quantiles of the GAN-based models with rather high uncertainties in prediction error.

In case of long-term predictions the ConvLSTM also delivers the best predictions out of the 9 evaluated models. The Single Frame Input Model is not able to predict a single next frame, hence no longer predictions also. Compare to the later the Multiple Frame Input Model performs slightly better, but also fails on all datasets, except the $T=80$, $TINC=4$ setups to fulfill both thresholds and can only predict a single next 3D object frame. Although the ConvLSTM model is clearly better due to its recurrent nature of the network, learning temporal changes, predicting more than two next frames is also not possible within an acceptable quality regarding the mean values against the predefined prediction to input and Null Model thresholds.

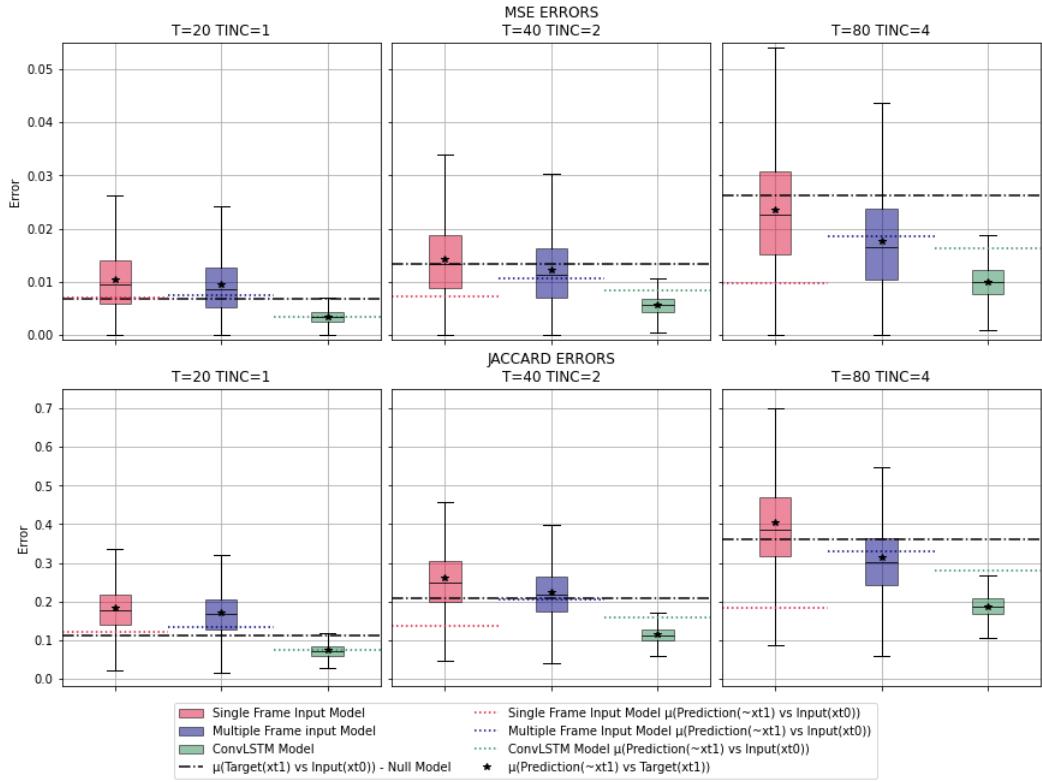


Figure 9.1: Boxplot comparison of all models for the MSE and Jaccard Distance between prediction-target, prediction-input and the null model for the three data formats T=20-TINC=1, T=40-TINC=2 and T=80-TINC=4.

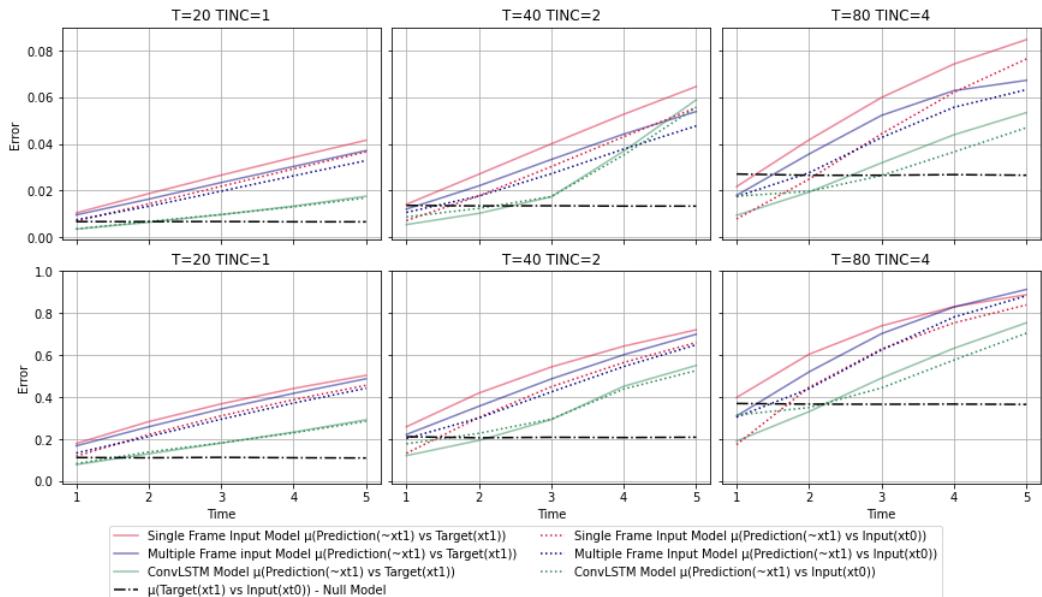


Figure 9.2: Long-term prediction comparison of all models for the MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T=20-INC=1, T=40-INC=2 and T=80-INC=4.

10 Conclusion

In this work, two general approaches for 3D spatio-temporal predictive models are trained and evaluated, the ConvLSTM (Section 4) and the GAN-based (Section 5) approach.

The ConvLSTM model outperforms the GAN-based approach in all setups due to its recurrent nature of the network. The plausibility thresholds are fulfilled for all setups tested, especially for the datasets prepared with larger frame to frame changes the results are clearly below. This indicates that the transition from the input to the target frame has obviously been learned. However, the long-term predictions struggle to predict frames within the acceptable threshold further ahead than 2 steps, although the optical comparison of the prediction to target shows to be not too far away from the target.

The GAN-based approach was tested with a single frame and multiple frames as inputs to learn from. The Single Frame Input Models were not able to produce meaningful predictions in any of the setups for single next frame or long-term predictions, due to the lack of information of past time frames, representing the temporal change. The Multiple Frame Input Models perform better on the single next frame predictions, especially on the datasets prepared with larger frame to frame changes, where they closely fulfill the plausibility thresholds described in section 6.3. The additional information of past frames clearly helps to learn the next 3D object frame. On the contrary, the long-term predictions fail to predict acceptable object qualities already at the second time step.

In conclusion, out of the two approaches and setups tested, the ConvLSTM model with the larger frame to frame dataset format showed the best performance overall. The larger transition within frames didn't produce an obstacle for the models, but has even encouraged learning the transition from one frame to the next in both model bases.

10.1 Limitations and Outlook

Although, the grid-based 4D format comes with a lot of advantages in combinations with neural networks and their treatment within the GPU, allowing for straight forward implementation of vectorized tensor operation, without an encoding or conversion into a compatible format of the 3D object, it comes at the cost of data inefficient storage of information, where all positions in the grid store information, even though it might be empty space filled with zeros. The original 4D dataset size of a single datapoint of $128 \times 128 \times 128 \times 128$ (268,435,456 entries of zeros and ones) is a huge amount (even not taking into account possible color information or something similar yet), not processable on high end workstation, and even a challenge on high end clusters,

which is why it was reduced to 128x16x16x16 in this work to be able to be trained on the available hardware described in section 1.1. A possible remedy and different approach to grid-based data are networks architectures, like in [23], [35], [36], which can deal with unstructured 3D object data, like point clouds or meshes, with variable input length.

For the GAN-based approach surely more research is necessary to improve their prediction performance. Especially a way of introducing temporal information is necessary to enhance the forecast quality. The encoder with shared weights, implemented in this work and already used in a similar way in [21], improved the quality to a reasonable extend.

Proper 4D data (3D + time axis) is hardly available. Hence, all trainings and tests in this work were performed solely on the 4D dataset from [11]. This dataset is readily prepared in python format to be used in machine learning for training of models. However, the dataset represents a toy dataset with barely any relation to real world problems. Looking at the time-wise change of the objects, the data has a high rate of spatio-temporal variability, including periodically appearance and disappearance of objects. Most real world problems although, like simulation results, CT scans, growing of plants, etc., might have smaller changes time-wise with physical relations or are steadily expanding and not appearing or disappearing, including more conditional information present to learn from. Therefore, learning and testing the approaches from within this work on a real dataset is a logical next step.

Bibliography

- [1] Panos Achlioptas et al. *Learning Representations and Generative Models for 3D Point Clouds*. 2018. arXiv: [1707.02392 \[cs.CV\]](https://arxiv.org/abs/1707.02392).
- [2] Thomas Adler. “Convolutional LSTM for Next Frame Prediction”. Available at <https://epub.jku.at/obvulihs/download/pdf/1825342?originalFilename=true>. Master’s thesis. Linz, Austria: Johannes Kepler University Linz, Feb. 2017.
- [3] Sandra Aigner and Marco Körner. *FutureGAN: Anticipating the Future Frames of Video Sequences using Spatio-Temporal 3d Convolutions in Progressively Growing GANs*. 2018. arXiv: [1810.01325 \[cs.CV\]](https://arxiv.org/abs/1810.01325).
- [4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: [1701.07875 \[stat.ML\]](https://arxiv.org/abs/1701.07875).
- [5] Emad Barsoum, John Kender, and Zicheng Liu. *HP-GAN: Probabilistic 3D human motion prediction via GAN*. 2017. arXiv: [1711.09561 \[cs.CV\]](https://arxiv.org/abs/1711.09561).
- [6] Andrew Brock et al. *Generative and Discriminative Voxel Modeling with Convolutional Neural Networks*. 2016. arXiv: [1608.04236 \[cs.CV\]](https://arxiv.org/abs/1608.04236).
- [7] Baptiste Chopin et al. *Human Motion Prediction Using Manifold-Aware Wasserstein GAN*. 2021. arXiv: [2105.08715 \[cs.CV\]](https://arxiv.org/abs/2105.08715).
- [8] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: [1406.2661 \[stat.ML\]](https://arxiv.org/abs/1406.2661).
- [9] Klaus Greff et al. “LSTM: A Search Space Odyssey”. In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (Oct. 2017), pp. 2222–2232. ISSN: 2162-2388. DOI: [10.1109/TNNLS.2016.2582924](https://doi.org/10.1109/TNNLS.2016.2582924). URL: <http://dx.doi.org/10.1109/TNNLS.2016.2582924>.
- [10] Ishaan Gulrajani et al. *Improved Training of Wasserstein GANs*. 2017. arXiv: [1704.00028 \[cs.LG\]](https://arxiv.org/abs/1704.00028).
- [11] Khalil Mathieu Hannouch and Stephan Chalup. *Topology Estimation of Simulated 4D Image Data by Combining Downscaling and Convolutional Neural Networks*. 2023. arXiv: [2306.14442 \[cs.CV\]](https://arxiv.org/abs/2306.14442).
- [12] Kaiming He et al. *Identity Mappings in Deep Residual Networks*. 2016. arXiv: [1603.05027 \[cs.CV\]](https://arxiv.org/abs/1603.05027).
- [13] Justin Hellermann and Stefan Lessmann. *Leveraging Image-based Generative Adversarial Networks for Time Series Generation*. 2023. arXiv: [2112.08060 \[cs.LG\]](https://arxiv.org/abs/2112.08060).

- [14] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: [10.1162/neco.1997.9.8.1735](https://doi.org/10.1162/neco.1997.9.8.1735). eprint: <https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf>. URL: <https://doi.org/10.1162/neco.1997.9.8.1735>.
- [15] Matin Hosseini et al. *Inception-inspired LSTM for Next-frame Video Prediction*. 2020. arXiv: [1909.05622 \[cs.CV\]](https://arxiv.org/abs/1909.05622).
- [16] Paul Jaccard. “Lois de distribution florale dans la zone alpine”. In: *Bull Soc Vaudoise Sci Nat* 38 (1902), pp. 69–130.
- [17] Nal Kalchbrenner et al. *Video Pixel Networks*. 2016. arXiv: [1610.00527 \[cs.CV\]](https://arxiv.org/abs/1610.00527).
- [18] Andrej Karpathy. *The unreasonable effectiveness of recurrent neural networks*. 2015. URL: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>.
- [19] Tero Karras et al. *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. 2018. arXiv: [1710.10196 \[cs.NE\]](https://arxiv.org/abs/1710.10196).
- [20] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: [1412.6980 \[cs.LG\]](https://arxiv.org/abs/1412.6980).
- [21] Tanja Linkermann. “Motion Prediction with the Improved Wasserstein GAN”. Available at <https://diglib.tugraz.at/download.php?id=6144a220477d4&location=browse>. Master’s thesis. Graz, Austria: Graz University of Technology, Apr. 2019.
- [22] *Next-Frame Video Prediction with Convolutional LSTMs*. https://keras.io/examples/vision/conv_lstm/.
- [23] Charles R. Qi et al. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. 2017. arXiv: [1612.00593 \[cs.CV\]](https://arxiv.org/abs/1612.00593).
- [24] Masaki Saito, Eiichi Matsumoto, and Shunta Saito. *Temporal Generative Adversarial Nets with Singular Value Clipping*. 2017. arXiv: [1611.06624 \[cs.LG\]](https://arxiv.org/abs/1611.06624).
- [25] *scikit-image - Image processing in Python*. <https://scikit-image.org/>.
- [26] Xingjian Shi et al. “Convolutional LSTM Network: a machine learning approach for precipitation nowcasting”. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. NIPS’15. Montreal, Canada: MIT Press, 2015, pp. 802–810.
- [27] Zifan Shi et al. *Deep Generative Models on 3D Representations: A Survey*. 2023. arXiv: [2210.15663 \[cs.CV\]](https://arxiv.org/abs/2210.15663).
- [28] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: [1409.4842 \[cs.CV\]](https://arxiv.org/abs/1409.4842).
- [29] Satya Pratheek Tata and Subhankar Mishra. *3D GANs and Latent Space: A comprehensive survey*. 2023. arXiv: [2304.03932 \[cs.CV\]](https://arxiv.org/abs/2304.03932).
- [30] *tensorflow 2.10 API*. https://www.tensorflow.org/versions/r2.10/api_docs/python/tf.

- [31] *TFrecords*. https://www.tensorflow.org/tutorials/load_data/tfrecord.
- [32] Du Tran et al. *Learning Spatiotemporal Features with 3D Convolutional Networks*. 2015. arXiv: [1412.0767 \[cs.CV\]](https://arxiv.org/abs/1412.0767).
- [33] Ruifu Wang et al. “Improvement and Application of a GAN Model for Time Series Image Prediction—A Case Study of Time Series Satellite Cloud Images”. In: *Remote Sensing* 14 (Nov. 2022), p. 5518. DOI: [10.3390/rs14215518](https://doi.org/10.3390/rs14215518).
- [34] Jiajun Wu et al. *Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling*. 2017. arXiv: [1610.07584 \[cs.CV\]](https://arxiv.org/abs/1610.07584).
- [35] Wenzuan Wu, Zhongang Qi, and Li Fuxin. *PointConv: Deep Convolutional Networks on 3D Point Clouds*. 2020. arXiv: [1811.07246 \[cs.CV\]](https://arxiv.org/abs/1811.07246).
- [36] Xiaohui Zeng et al. *LION: Latent Point Diffusion Models for 3D Shape Generation*. 2022. arXiv: [2210.06978 \[cs.CV\]](https://arxiv.org/abs/2210.06978).

List of Figures

Figure 1.1	10 next frame predictions with FutureGAN on the Moving MNIST dataset (top row: ground truth; bottom row: FutureGAN). Source: [3]	2
Figure 1.2	10 next frame predictions with ConvLSTM; top: input, middle: ground truth, bottom: prediction. Source: [26]	3
Figure 2.1	Cavities within a 128x128x128x128 sample, 18 equally-spaced 3D slices along the time-axis from left to right and top to bottom. Source: [11]	5
Figure 2.2	Resolution comparison of a random sample at a random time-point. Left: Original Size of 128x128x128; Right: Max-Pooled Size of 16x16x16	6
Figure 2.3	Illustration of how parameters T and TINC influence the sample extraction for the ConvLSTM models.	7
Figure 3.1	Left: Point cloud of an object; Right: Reconstructed object from points with Marching Cubes algorithm.	9
Figure 4.1	LSTM block. Source: [9]	10
Figure 4.2	LSTM modes. Source: [18]	11
Figure 4.3	Illustration of the ConvLSTM model. Source: [26]	11
Figure 4.4	ConvLSTM network.	12
Figure 5.1	a) Residual Unit; b) Pre-Residual Unit; c) Pre-Residual Unit used in the GAN-based models in this work. BN=Batch Normalization; ReLU=Rectified Linear Unit Activation; weight=weights of the layer, e.g. weights of the convolutional layer. Source: [12]	14
Figure 5.2	Single Frame Input Model.	16
Figure 5.3	Multiple Frame Input Model.	16
Figure 5.4	Encoder network.	18
Figure 5.5	Generator network.	19
Figure 5.6	Discriminator network.	20
Figure 7.1	Training and validation loss of the BCE, MSE and Jaccard Distance.	25
Figure 7.2	ConvLSTM Model - Histogram and boxplot of the MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T=20-INC=1, T=40-INC=2 and T=80-INC=4.	27
Figure 7.3	ConvLSTM Model - Exemplary generated predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.	27
Figure 7.4	ConvLSTM Model - Long-term MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T=20-INC=1, T=40-INC=2 and T=80-INC=4.	29

Figure 7.5	ConvLSTM Model - Exemplary generated long-term predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.	29
Figure 8.1	Training and validation loss of the Generator and Discriminator.	31
Figure 8.2	Training and validation loss of the MSE and Jaccard Distance.	31
Figure 8.3	Single Frame Input Model - Histogram and Boxplot of the MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.	32
Figure 8.4	Single Frame Input Model - Exemplary generated predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.	33
Figure 8.5	Multiple Frame Input Model - Histogram and Boxplot of the MSE and Jaccard Distance between prediction-target, prediction-input and the null model for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.	34
Figure 8.6	Multiple Frame Input Model - Exemplary generated predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.	34
Figure 8.7	Single Frame Input Model - Long-term MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.	35
Figure 8.8	Single Frame Input Model - Exemplary generated long-term predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.	36
Figure 8.9	Multiple Frame Input Model - Long-term MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.	37
Figure 8.10	Multiple Frame Input Model - Exemplary generated long-term predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.	37
Figure 9.1	Boxplot comparison of all models for the MSE and Jaccard Distance between prediction-target, prediction-input and the null model for the three data formats T=20-TINC=1, T=40-TINC=2 and T=80-TINC=4.	39
Figure 9.2	Long-term prediction comparison of all models for the MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T=20-INC=1, T=40-INC=2 and T=80-INC=4.	39

List of Tables

Table 2.1 Training, validation and test split.	4
Table 4.1 ConvLSTM Optimizer Settings	13
Table 5.1 GAN Optimizer Settings	22

List of Source Codes

A.1 ConvLSTM3D Training Algorithm	50
A.2 ConvLSTM3D - Data Parser	60
A.3 ConvLSTM3D Prediction Algorithm	63
A.4 ConvLSTM3D Long-Term Prediction Algorithm	65
B.1 Single Frame Input Training Algorithm	67
B.2 Single Frame Input Training Algorithm - Data Parser	88
B.3 Single Frame Input Prediction Algorithm	91
B.4 Single Frame Input Prediction Algorithm	93
B.5 Multiple Frame Input Training Algorithm	95
B.6 Multiple Frame Input Training Algorithm - Data Parser	117
B.7 Multiple Frame Input Prediction Algorithm	120
B.8 Multiple Frame Input Prediction Algorithm	122
B.9 Long-Term Prediction - Data Parser	125
C.1 Function to Plot Losses and Save them to a File	128
C.2 Functions to Generate 3D Objects from Grid-based data	132
C.3 Logger Class	135
C.4 *.npz to *.tfrecords Converter	136
C.5 Split Data into Train-, Validation- and Testsets	139

List of Abbreviations

MRI Magnetic Resonance Imaging

CT Computed Tomography

GAN Generative Adversarial Network

LSTM Long Short-Term Memory

RNN Recurrent Neural Network

ConvLSTM Convolutional Long Short-Term Memory

AE Auto-Encoder

VAE Variational Auto-Encoder

KL Kullback-Leibler

ReLU Rectified Linear Unit)

WGAN-GP Wasserstein-GAN with Gradient Penalty

MSE Mean-Squared-Error

Adam Adaptive moment estimation

BCE Binary Cross-Entropy

A ConvLSTM 3D Next Frame Prediction

A.1 ConvLSTM3D Training Algorithm

```
1000 import os
1001 from shutil import copyfile
1002 import glob
1003 import time
1004 import sys
1005 import numpy as np
1006
1007 from keras import layers
1008 import tensorflow as tf
1009 from tensorflow.keras import Model
1010 from tensorflow.keras.optimizers import Adam
1011
1012 # OWN LIBRARIES
1013 from utils.loss_plotter import render_graphs
1014 from utils import dataIO as d
1015 from utils import parse_tfrecords_4D_16_for_training_ConvLSTM as p4d
1016 from utils.logger import Logger
1017
1018 ##### inputs #####
1019 #####
1020 #####
1021 #####
1022 N_EPOCHS = 10000 # Default: 10000 – Max. training epochs
1023 MAX_ITERATIONS = 10000 # Default: -1 --> Max. Training Iterations (Max. Number of processed Batches) before training stops
1024 TRAIN_DATASET_LOC = "Z:/Master_Thesis/data/00_4D_training_datasets"
1025 VALID_DATASET_LOC = "Z:/Master_Thesis/data/01_4D_validation_datasets"
1026 N_TRAINING = 10000 # Number of Files for Training: -1 == USE ALL
1027 N_VALIDATION = 1000 # Number of Files for Validation: -1 == USE ALL
1028 T = 80 # Default: 80 – Even Number of Length of Time History to use, MUST BE EVENLY DIVISIBLE BY TINC, e.g. 80 means that 80 steps out of all 128 steps are used, where 24 steps at the beginning and 24 steps at the end of the time history are cropped.
1029 TINC = 4 # Default: 1 – Allowed: (1, 2, 4, 8) – e.g.: 4 --> 80/4 = 20 Timepoints --> 19 PAIRS of T0->T1 --> TIME INCREMENT FOR EXTRACTING Timesteps per datapoint – (Check parse_tfrecords_4D_##_for_training for extraction details)
```

```

1030 NORMALIZE          = False           # NORMALIZE DATA to ZERO MEAN --> from [0;1]
     to [-0.5, 0.5]
LOGGING            = True             # LOG TRAINING RUN TO FILE (If Kernel get
     Interupted (user or error) Kernel needs to be restarted , otherwise log file is
     locked to be deleted!)
#####
1032 ##### RESTART TRAINING #####
# Restart Training?
1034 RESTART_TRAINING    = False
RESTART_ITERATION   = 50000          # the epoch number when the last training
     was stopped, has no influnce on loading the restart file , just used for tracking
     information numbering
1036 RESTART_FOLDER_NAME = '20240221_012216_aencgan_3dnext_16_t3'
#####
1038 # Output Settings
VAL_SAVE_FREQ       = 200            # Default: N_TRAINING//BATCH_SIZE - OFF: -1,
     every n training iterations (n batches) check validation loss
1040 WRITE_RESTART_FREQ = 1000          # Default: 100 - every n training iterations
     (n batches) save restart checkpoint
KEEP_BEST_ONLY      = True            # Default: True - Keep only the best model (
     KEEP_LAST_N=1)
1042 BEST_METRIC        = "JACCARD"       # Default: JACCARD - "JACCARD" or "MSE"
# if KEEP_BEST_ONLY = False define max. Number of last models to keep:
1044 KEEP_LAST_N        = 5               # Default: 10 - Max. number of last files to
     keep
N_SAMPLES           = 5               # Default: 10 - Number of object samples to
     generate after each epoch
#####
1046 ##### HYPERPARAMETERS #####
1048 #####
1049 BATCH_SIZE         = 16              # Default: 64 - Number of Examples per Batch
     / Probably Increase for better Generalization
1050 FEATUREMAPS        = 32              # Default: 5 - Max Featuremap size (Maybe
     increase for learning larger/more diverse datasets...)
#####
1052 # Optimizer Settings
LR                  = 0.001           # Default: 0.001
1054 #####
# DEFAULT - DONT TOUCH SETTINGS
1056 SIZE               = 16              # Default: 16 - Voxel Cube Size
#####
1058 ##### GET DATA/OBJECTS #####
1059 #####
1060 TRAINING_FILERAMES = tf.io.gfile.glob(TRAIN_DATASET_LOC+"/*.tfrecords")[:N_TRAINING]
VALIDATION_FILERAMES = tf.io.gfile.glob(VALID_DATASET_LOC+"/*.tfrecords")[:N_VALIDATION]
1062 training_set_size = len(TRAINING_FILERAMES)
validation_set_size = len(VALIDATION_FILERAMES)
1064 print()
print("Total Number of Samples to Train:", training_set_size)
print("Total Number of Samples for Validation:", validation_set_size)
1066

```

```

print()

1068 # Convert/Decode dataset from tfrecords file for training
1070 train_objects = p4d.get_dataset(TRAINING_Filenames, shuffle=True, batch_size=
    BATCH_SIZE, t=T, tinc=TINC, normalize=NORMALIZE )
1071 val_objects = p4d.get_dataset(VALIDATION_Filenames, shuffle=False, batch_size=
    BATCH_SIZE, t=T, tinc=TINC, normalize=NORMALIZE )

1072 #####
1074 ##### CONVOLUTIONAL LSTM NETWORK #####
1075 #####
1076 # DEFINE CONVLSTM NETWORK
1077 def ConvLSTM3D():
1078     inp = layers.Input(shape=(None, *(16,16,1)))

1080     x = layers.ConvLSTM3D(filters=FEATUREMAPS, kernel_size=(5, 5, 5), padding="same",
1081     , return_sequences=True, activation="relu")(inp)
1082     x = layers.BatchNormalization()(x)

1084     x = layers.ConvLSTM3D(filters=FEATUREMAPS, kernel_size=(3, 3, 3), padding="same",
1085     , return_sequences=True, activation="relu")(x)
1086     x = layers.BatchNormalization()(x)

1088     x = layers.ConvLSTM3D(filters=FEATUREMAPS, kernel_size=(1, 1, 1), padding="same",
1089     , return_sequences=True, activation="relu")(x)
1090     #x = layers.BatchNormalization()(x)

1092     x = layers.Conv3D(filters=1, kernel_size=(5, 5, 5), activation="sigmoid",
1093     , padding="same", input_shape=x.shape[2:])(x)

1094     model = Model(inp, x)

1096     #####
1097     ##### COMPILE NETWORK #####
1098     #####
1099     model = ConvLSTM3D()
1100     #Compile Model
1101     model.compile()
1102     # PRINT NETWORK SUMMARY
1103     print(model.summary())

1104     #####
1105     ##### Optimizer #####
1106     #####
1107     optimizer = Adam(learning_rate=LR)

1108     #####
1109     ##### LOSS #####
1110

```

```

#####
1112 @tf.function
1113 def BCE_LOSS(y_true , y_pred):
1114     bce=tf.keras.metrics.binary_crossentropy(y_true , y_pred , from_logits=False ,
1115     label_smoothing=0.0 , axis=-1)
1116     BCE_loss = tf.math.reduce_mean(bce)
1117
1118     return BCE_loss
1119
1120 #####
1121 @tf.function
1122 def MSE_LOSS(y_true , y_pred):
1123     mse = tf.keras.metrics.mean_squared_error(y_true , y_pred)
1124     MSE_loss = tf.reduce_mean(mse)
1125
1126     return MSE_loss
1127
1128 #####
1129 def JACCARD_LOSS(X, y):
1130     y_true = tf.dtypes.cast(y, tf.float32)
1131     y_pred = np.squeeze(model(X, training=False))
1132
1133     y_real = np.asarray(y_true>0.5, bool)
1134     y_pred = np.asarray(y_pred>0.5, bool)
1135
1136     j_and = tf.math.reduce_sum(np.double(np.bitwise_and(y_real , y_pred)), axis
1137     =[2,3,4])
1138     j_or = tf.math.reduce_sum(np.double(np.bitwise_or(y_real , y_pred)), axis
1139     =[2,3,4])
1140
1141     JACCARD_loss = tf.reduce_mean(np.nan_to_num(1.0 - j_and / j_or , nan=0.0))
1142
1143     return JACCARD_loss
1144 #####
1145 ###### TRAINING STEP #####
1146 #####
1147 @tf.function
1148 def train_step(X, y):
1149
1150     X = X[... ,np.newaxis]
1151     y_true = y[... ,np.newaxis]
1152
1153     with tf.GradientTape() as tape:
1154         y_pred = model(X, training=True)
1155         BCE_loss = BCE_LOSS(y_true , y_pred)
1156         MSE_loss = MSE_LOSS(y_true , y_pred)
1157
1158         gradients = tape.gradient(BCE_loss , model.trainable_variables)
1159         optimizer.apply_gradients(zip(gradients , model.trainable_variables))
1160
1161     return BCE_loss, MSE_loss

```

```

1158
1159     @tf.function
1160     def validation_step(X, y):
1161         X = X[..., np.newaxis]
1162         y_true = y[..., np.newaxis]
1163
1164         y_pred = model(X, training=False)
1165         BCE_loss = BCE_LOSS(y_true, y_pred)
1166         MSE_loss = MSE_LOSS(y_true, y_pred)
1167
1168         return BCE_loss, MSE_loss
1169
1170 ##### SOME UTILITY FUNCTIONS #####
1171
1172 def generate_sample_objects(objects, total_iterations, tag="pred"):
1173     # Random Sample at every VAL_SAVE_FREQ
1174     i=0
1175
1176     for X, y in objects:
1177         X = tf.dtypes.cast(X, tf.float32)
1178         y_pred = np.squeeze(model(X, training=False) > 0.5)
1179         y_pred_last = y_pred[-1]
1180         try:
1181             d.plotMeshFromVoxels(y_pred_last, obj=model_directory+tag+'_'+str(i)+_
1182             '_next'+'_iter_'+str(total_iterations))
1183         except:
1184             print(f"Cannot generate STL, Marching Cubes Algo failed for sample {i}")
1185             # print('''Cannot generate STL, Marching Cubes Algo failed: Surface
1186             # level must be within volume data range! \n
1187             # This may happen at the beginning of the training, if it still
1188             # happens at later stages epoch>10 --> Check Object and try to change Marching
1189             # Cube Threshold.''')
1190
1191         i+=1
1192
1193 def IOhousekeeping(model_directory, KEEP_LAST_N, VAL_SAVE_FREQ, RESTART_TRAINING,
1194     total_iterations, KEEP_BEST_ONLY, best_iteration):
1195     if total_iterations > KEEP_LAST_N*VAL_SAVE_FREQ and total_iterations % VAL_SAVE_FREQ == 0:
1196         if KEEP_BEST_ONLY:
1197             fileList_all = glob.glob(model_directory + '*_iter_*')
1198             fileList_best = glob.glob(model_directory + '*_iter_' + str(int(best_iteration)) + '*')
1199             fileList_del = [ele for ele in fileList_all if ele not in fileList_best]
1200
1201         else:
1202             fileList_del = glob.glob(model_directory + '*_iter_' + str(int(total_iterations -KEEP_LAST_N*VAL_SAVE_FREQ)) + '*')

```

```

1200     # Iterate over the list of filepaths & remove each file .
1202     for filePath in fileList_del:
1203         os.remove(filePath)

1204 ##### LAST PREPARATION STEPS BEFORE TRAINING STARTS #####
1205 #####
1206 #####
1207 #generate folders:
1208 if RESTART_TRAINING:
1209     model_directory = os.getcwd() + '/' + RESTART_FOLDER_NAME + '/'
1210 else:
1211     named_tuple = time.localtime() # get struct_time
1212     time_string = time.strftime("%Y%m%d_%H%M%S", named_tuple)
1213     model_directory = os.getcwd()+'/' +time_string+'_' + os.path.splitext(os.path.
1214 basename(__file__))[0] + '/'
1215     if not os.path.exists(model_directory):
1216         os.makedirs(model_directory)

1217 # Save running python file to run directory
1218 if RESTART_TRAINING:
1219     named_tuple = time.localtime() # get struct_time
1220     time_string = time.strftime("%Y%m%d_%H%M%S", named_tuple)
1221     copyfile(__file__ , model_directory+time_string +'_restart_'+os.path.basename(
1222 __file__))
1223 else:
1224     copyfile(__file__ , model_directory+os.path.basename(__file__))

1225 # Activate Console Logging :
1226 if LOGGING:
1227     sys.stdout = Logger(filename=model_directory+'log_out.txt')

1228 # save some real examples
1229 i=0
1230 train_objects_samples = []
1231 for X, y in train_objects.take(N_SAMPLES):
1232     X_last = X[0][-1]
1233     y_last = y[0][-1]

1235 try:
1236     sample_name = model_directory+'train_'+str(i)+'_init'
1237     d.plotMeshFromVoxels(np.array(X_last) , obj=sample_name)
1238     sample_name = model_directory+'train_'+str(i)+'_next'
1239     d.plotMeshFromVoxels(np.array(y_last) , obj=sample_name)
1240 except:
1241     print("Example Object couldnt be generated with Marching Cube Algorithm .
1242 Probable Reason: Empty Input Tensor, All Zeros")

1243 train_objects_samples.append([X[0][np.newaxis ,..., np.newaxis] , y[0][np.newaxis
1244 ,..., np.newaxis]]) # for validation of reconstruction during training , see
1245 generate_sample_objects():

```

```

    i+=1
1246
# save some validation examples
1248 i=0
val_objects_samples = []
1250 for X, y in val_objects.take(N_SAMPLES):
    X_last = X[0][-1]
    y_last = y[0][-1]

1254 try:
    sample_name = model_directory+'valid_'+str(i)+'_init'
1256     d.plotMeshFromVoxels(np.array(X_last), obj=sample_name)
    sample_name = model_directory+'valid_'+str(i)+'_next'
1258     d.plotMeshFromVoxels(np.array(y_last), obj=sample_name)
except:
    print("Example Object couldnt be generated with Marching Cube Algorithm.
Probable Reason: Empty Input Tensor, All Zeros")

1262 val_objects_samples.append([X[0][np.newaxis, ..., np.newaxis], y[0][np.newaxis,
..., np.newaxis]]) # for validation of reconstruction during training , see
generate_sample_objects():
    i+=1
1264 #####
1266 # Define Checkpoint
checkpoint_path = model_directory+'checkpoints/'
1268 checkpoint_dir = os.path.dirname(checkpoint_path)

1270 ckpt = tf.train.Checkpoint(optimizer=optimizer,
                             model=model)

1272 manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=1)
1274
def save_checkpoints(ckpt, model_directory):
    if not os.path.exists(model_directory+'checkpoints/'):
        os.makedirs(model_directory+'checkpoints/')
    manager.save()

1280 if RESTART_TRAINING:
    total_iterations = RESTART_ITERATION
1282 RESTART_EPOCH = training_set_size // BATCH_SIZE
    if manager.latest_checkpoint:
        ckpt.restore(manager.latest_checkpoint)
        print("\nRestored from {} \n".format(manager.latest_checkpoint))
1286 else:
    print("No Checkpoint found --> Initializing from scratch.")

1288 #####
1290 ##### TRAINING #####
#####
```

```

1292 print("-----")
1293 print('RUN TRAINING:\n')
1294 #####
1295 start_time = time.time()
1296 iter_arr, val_iter_arr = [], []
1297 BCE_losses_train = []
1298 MSE_losses_train = []
1299 JACCARD_losses_train = []
1300 BCE_losses_val_mean = []
1301 MSE_losses_val_mean = []
1302 JACCARD_losses_val_mean = []
1303 n_objects = 0
1304 total_iterations = 0
1305 best_iteration = VAL_SAVE_FREQ
1306 best_loss = 1e10
1307 for epoch in range(1, N_EPOCHS+1):
1308     #####
1309     # START OF EPOCH
1310     t_epoch_start = time.time()
1311     n_batch=0
1312     #####
1313
1314     for X, y in train_objects:
1315         # START OF BATCH
1316         batch_time = time.time()
1317         BCE_loss_train, MSE_loss_train = train_step(X, y)
1318         #Append arrays
1319         BCE_losses_train.append(BCE_loss_train)
1320         MSE_losses_train.append(MSE_loss_train)
1321
1322         JACCARD_loss_train = JACCARD_LOSS(X, y)
1323         JACCARD_losses_train.append(JACCARD_loss_train)
1324
1325         n_batch+=1
1326         total_iterations+=1
1327         iter_arr.append(total_iterations)
1328
1329         print("E: %3d/%5d | B: %3d/%3d | I: %5d | T: %5.1f | dt: %2.2f || LOSSES:
1330             BCE %.4f | MSE %.4f | JACCARD %.4f | "
1331             % (epoch, N_EPOCHS, n_batch, training_set_size//BATCH_SIZE,
1332                 total_iterations, time.time() - start_time, time.time() - batch_time,
1333                 BCE_loss_train, MSE_loss_train, JACCARD_loss_train))
1334
1335     #####  

1336     # AFTER N TOTAL ITERATIONS GET VALIDATION LOSSES, SAVE GENERATED OBJECTS AND  

1337     MODEL:  

1338     if (total_iterations % VAL_SAVE_FREQ == 0) and VAL_SAVE_FREQ != -1:  

1339         print('\nRUN VALIDATION, SAVE OBJECTS, MODEL AND PLOT LOSSES... ')
1340
1341     BCE_losses_val = []

```

```

1338     MSE_losses_val = []
1339     JACCARD_losses_val = []
1340     for X, y in val_objects:
1341         BCE_loss_val, MSE_loss_val = validation_step(X, y)
1342         BCE_losses_val.append(BCE_loss_val) # Collect Val loss per Batch
1343         MSE_losses_val.append(MSE_loss_val) # Collect Val loss per Batch
1344
1345         JACCARD_loss_val = JACCARD_LOSS(X, y)
1346         JACCARD_losses_val.append(JACCARD_loss_val) # Collect Val loss per
1347         Batch
1348
1349         BCE_losses_val_mean.append(np.mean(BCE_losses_val)) # Mean over batches
1350         MSE_losses_val_mean.append(np.mean(MSE_losses_val)) # Mean over batches
1351         JACCARD_losses_val_mean.append(np.mean(JACCARD_losses_val)) # Mean over
1352         batches
1353
1354         val_iter_arr.append(total_iterations)
1355         ##### # Output generated objects
1356         generate_sample_objects(train_objects_samples, total_iterations, 'train')
1357     )
1358         generate_sample_objects(val_objects_samples, total_iterations, 'valid')
1359         # # Plot and Save Loss Curves
1360         render_graphs(model_directory, BCE_losses_train, MSE_losses_train,
1361 JACCARD_losses_train, BCE_losses_val_mean, MSE_losses_val_mean,
1362 JACCARD_losses_val_mean, iter_arr, val_iter_arr, RESTART_TRAINING) #this will
1363         only work after a 50 iterations to allow for proper averaging
1364
1365         if KEEP_BEST_ONLY:
1366             if BEST_METRIC == "JACCARD":
1367                 BEST_LOSS_METRIC = JACCARD_losses_val_mean
1368                 KEEP_LAST_N = 1
1369             elif BEST_METRIC == "MSE":
1370                 BEST_LOSS_METRIC = MSE_losses_val_mean
1371                 KEEP_LAST_N = 1
1372             if len(BEST_LOSS_METRIC) > 1 and (BEST_LOSS_METRIC[-1] <= best_loss)
1373             :
1374                 best_loss = BEST_LOSS_METRIC[-1]
1375                 best_iteration = total_iterations
1376                 # Save models
1377                 model.save(model_directory+'_trained_ConvLSTM3D_'+iter_+str(
1378 total_iterations)+'.h5')
1379             else:
1380                 # Save models
1381                 model.save(model_directory+'_trained_ConvLSTM3D_'+iter_+str(
1382 total_iterations)+'.h5')
1383
1384                 # # Delete Model, keep best or last N models
1385                 IOhousekeeping(model_directory, KEEP_LAST_N, VAL_SAVE_FREQ,
1386 RESTART_TRAINING, total_iterations, KEEP_BEST_ONLY, best_iteration)

```

```

1378 #####
1380 # SAVE CHECKPOINT FOR RESTART:
1381 if total_iterations % WRITE_RESTART_FREQ == 0:
1382     print('WRITE RESTART FILE...\n')
1383     # Save Checkpoint after every 100 epoch
1384     save_checkpoints(ckpt, model_directory)

1386 #####
1387 # STOP TRAINING AFTER MAX DEFINED ITERATIONS ARE REACHED
1388 if total_iterations == MAX_ITERATIONS :
1389     break
1390 if total_iterations == MAX_ITERATIONS:
1391     break

1392 # Print Status at the end of the epoch
1393 dt_epoch = time.time() - t_epoch_start
1394 n_objects = n_objects + training_set_size
1395 print("\n-----")
1396 print('END OF EPOCH ', epoch, '| Total Training Iterations: ', int(
1397 total_iterations), '| Total Number of objects trained: ', int(n_objects/1000),
1398 'k', '| time elapsed:', str(int((time.time() - start_time) / 60.0)), "min")
1399 print("-----\n\n")
1400

1401 # END OF EPOCH
1402 #####
1403
1404 print('\n TRAINING DONE! \n')
1405 print("Total Training Time: ", str((time.time() - start_time) / 60.0), "min" )

```

Listing A.1: ConvLSTM3D Training Algorithm

A.2 ConvLSTM3D Training Algorithm - Data Parser

```
1000 import os
os.environ[ 'TF_CPP_MIN_LOG_LEVEL' ] = '1'
1002 import tensorflow as tf
1003 import numpy as np
1004
AUTOTUNE = tf.data.AUTOTUNE
1006
1007 def decode_object(example):
1008     object4d = tf.io.decode_raw(example[ 'object4d' ], tf.uint8)
1009     labelBetti = tf.io.decode_raw(example[ 'labelBetti' ], tf.uint8)
1010     labelObjs = tf.io.decode_raw(example[ 'labelObjs' ], tf.uint8)
1011
1012     object4d = tf.reshape(object4d, [128, 16, 16, 16])
1013     labelBetti = tf.reshape(labelBetti, [4])
1014     labelObjs = tf.reshape(labelObjs, [4])
1015
1016     return object4d, labelBetti, labelObjs
1017
1018 def read_tfrecord(example):
1019     global NORMALIZE
1020     global T
1021     global TINC
1022
1023     object_feature_description = {
1024         'object4d': tf.io.FixedLenFeature([], tf.string),
1025         'labelBetti': tf.io.FixedLenFeature([], tf.string),
1026         'labelObjs': tf.io.FixedLenFeature([], tf.string),
1027     }
1028
1029     example = tf.io.parse_single_example(example, object_feature_description)
1030
1031     object4d, labelBetti, labelObjs = decode_object(example)
1032
1033     CROP=(128-T)//2
1034     # Reshape into time slices of n time steps (axis=1)
1035     object4d = object4d[CROP:128-CROP:TINC]
1036     object4d = tf.reshape(object4d, [T//TINC, 16, 16, 16])
1037     x_object4d = object4d[0 : object4d.shape[0] - 1, :, :]
1038     y_object4d = object4d[1 : object4d.shape[0], :, :]
1039
1040     if NORMALIZE:
1041         object4d = tf.cast(object4d, tf.float32) - 0.5
1042
1043     return x_object4d, y_object4d #, tf.ones([ object4d.shape[0]-1,4], tf.uint8)*
1044     labelObjs
```

```

#return object4d , tf.ones(object4d.shape[0], tf.uint8)*labelBetti[-1]#, tf.ones([object4d.shape[0],4], tf.uint8)*labelObjs

1046
1047 def load_dataset(filenames):
1048     options = tf.data.Options()
1049     options.deterministic = False # disable order, increase speed
1050     dataset = tf.data.TFRecordDataset(filenames, compression_type="GZIP",
1051         num_parallel_reads=AUTOTUNE) # automatically interleaves reads from multiple
1052         files
1053     dataset = dataset.with_options(options) # uses data as soon as it streams in,
1054         rather than in its original order
1055     dataset = dataset.map(read_tfrecord, num_parallel_calls=AUTOTUNE)
1056     return dataset
1057
1058
1059 def get_dataset(filenames: list, shuffle: bool, batch_size: int, t: int, tinc: int,
1060 normalize: bool):
1061     global NORMALIZE
1062     global T
1063     global TINC
1064     NORMALIZE = normalize
1065     TINC = tinc
1066     T = t
1067     dataset = load_dataset(filenames)
1068
1069     if shuffle:
1070         dataset = dataset.shuffle(len(filenames)*(T//TINC)//2) # ALL: len(filenames)
1071             *(T//TINC)
1072     if batch_size is not None:
1073         dataset = dataset.batch(batch_size, drop_remainder=True)
1074     #dataset = dataset.prefetch(buffer_size=AUTOTUNE)
1075     #dataset = dataset.cache()
1076     return dataset
1077
1078
1079 # FOR TESTING:
1080 """
1081 folder_name="Z:/Master_Thesis/data/4D_128_32_topological_dataset_0-31_16_label8only"
1082
1083 FILENAMES = tf.io.gfile.glob(folder_name+"/*.tfrecords")[:7]
1084 split_ind = int(1 * len(FILENAMES))
1085 TRAINING_FILENOAMES, VALID_FILENOAMES = FILENAMES[:split_ind], FILENAMES[split_ind:]
1086
1087 #TEST_FILENOAMES = tf.io.gfile.glob(GCS_PATH + "/tfrecords/test*.tfrec")
1088 print("Train TFRecord Files:", len(TRAINING_FILENOAMES))
1089 print("Validation TFRecord Files:", len(VALID_FILENOAMES))
1090 #print("Test TFRecord Files:", len(TEST_FILENOAMES))
1091
1092 BATCH_SIZE = 64

```

```

1088 train_dataset = get_dataset(TRAINING_FILERAMES, shuffle=True, batch_size=BATCH_SIZE,
1089     t=80, tinc=4, normalize=False)
1090 #print(len(list(train_dataset)) * BATCH_SIZE)
1091 #object4d, labelBetti = next(iter(train_dataset))
1092 # print(object4d.shape, labelBetti.shape)
1093 # print(object4d[0], labelBetti)

1094 #print(tf.data.experimental.cardinality(train_dataset))

1096 def jaccard(x,y):
1097     x = np.asarray(x, bool) # Not necessary, if you keep your data
1098     y = np.asarray(y, bool) # in a boolean array already!

1100 #j = tf.math.reduce_sum(tf.square(differences), axis=[1,2,3,4])
1101 j_and = tf.math.reduce_sum(np.double(np.bitwise_and(x, y)), axis=[1,2,3])
1102 j_or = tf.math.reduce_sum(np.double(np.bitwise_or(x, y)), axis=[1,2,3])

1104 return tf.reduce_mean(1 - j_and / j_or)

1106 import dataIO
1107 for object4d, labelBetti in train_dataset.take(1):
1108     print(object4d.shape, labelBetti.shape)
1109     jaccard_ = jaccard(object4d[:,0], object4d[:,1])

1110     i=0
1111     for object4d_i, labelBetti_i in zip(object4d, labelBetti):
1112         print(object4d_i.shape, labelBetti_i.shape)
1113         #print(object4d_i.shape, labelBetti_i)

1116         for object4d_ii in object4d_i:
1117             dataIO.plotFromVoxels(object4d_ii.numpy(), [i, labelBetti_i])
1118             i+=1
1119
1120 ,,

```

Listing A.2: ConvLSTM3D - Data Parser

A.3 ConvLSTM3D Prediction Algorithm

```
1000 import os
1001 import tensorflow as tf
1002 from keras.models import load_model
1003 import numpy as np
1004 # OWN LIBRARIES
1005 from utils import parse_tfrecords_4D_16_for_training_ConvLSTM as p4d
1006 from utils import dataIO as d

1008 base_dir = r"Z:\Master_Thesis\code\99_FINAL_MODELS_FOR_THESIS"
1009
1010 MODEL_LOCATION      = os.path.join(base_dir, "convlstm3d_16_T20_TINC1")
1011
1012 TEST_DATASET_LOC    = "Z:/Master_Thesis/data/02_4D_test_datasets"
1013
1014 T                  = int(MODEL_LOCATION[-8:-6])
1015 TINC                = int(MODEL_LOCATION[-1])
1016 N_PREDICT          = 5
1017
1018 if not os.path.exists(os.path.join(base_dir, f'objects/T{T}_TINC{TINC}')):
1019     os.makedirs(os.path.join(base_dir, f'objects/T{T}_TINC{TINC}'))
1020 #####
1021 ##### GET Data/Objects #####
1022 #####
1023 #####
1024 TEST_Filenames = tf.io.gfile.glob(os.path.join(TEST_DATASET_LOC, "*tfrecords"))
test_set_size = len(TEST_Filenames)
1026
1027 print()
1028 print("Total Number of TFRecord Files to Test : ", test_set_size)
1029 print()
1030
1031 # Convert/Decode dataset from tfrecords file for training
1032 test_objects = p4d.get_dataset(TEST_Filenames, shuffle=True, batch_size=1, t=T, tinc=TINC, normalize=False)
1033
1034 convlstm3d_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*ConvLSTM3D*.h5"))
convlstm3d = load_model(convlstm3d_model[-1])
1036
1037 def predictions(objects):
1038     n=0
1039     for X, y in objects:
1040         d.plotFromVoxels(np.squeeze(X[0,0]), f"{n}_T{T}_TINC{TINC}_t-1")
1041         d.plotMeshFromVoxels(np.squeeze(X[0,0]), obj=f"objects/T{T}_TINC{TINC}/"
1042                             convlstm_{n}_T{T}_TINC{TINC}_t0")
1043         d.plotFromVoxels(np.squeeze(X[0,1]), f"{n}_T{T}_TINC{TINC}_t-2")
1044         d.plotMeshFromVoxels(np.squeeze(X[0,1]), obj=f"objects/T{T}_TINC{TINC}/"
1045                             convlstm_{n}_T{T}_TINC{TINC}_t1")
```

```

1044     d.plotFromVoxels(np.squeeze(X[0,2]), f"{{n}}_T{T}_TINC{TINC}_t0")
1045     d.plotMeshFromVoxels(np.squeeze(X[0,2]), obj=f"objects/T{T}_TINC{TINC}/
1046     convlstm_{n}_T{T}_TINC{TINC}_t2")
1047     d.plotFromVoxels(np.squeeze(X[0,3]), f"{{n}}_T{T}_TINC{TINC}_t1")
1048     d.plotMeshFromVoxels(np.squeeze(X[0,3]), obj=f"objects/T{T}_TINC{TINC}/
1049     convlstm_{n}_T{T}_TINC{TINC}_t3")
1050     d.plotFromVoxels(np.squeeze(X[0,4]), f"{{n}}_T{T}_TINC{TINC}_t1")
1051     d.plotMeshFromVoxels(np.squeeze(X[0,4]), obj=f"objects/T{T}_TINC{TINC}/
1052     convlstm_{n}_T{T}_TINC{TINC}_t4")
1053     d.plotFromVoxels(np.squeeze(X[0,5]), f"{{n}}_T{T}_TINC{TINC}_t1")
1054     d.plotMeshFromVoxels(np.squeeze(X[0,5]), obj=f"objects/T{T}_TINC{TINC}/
1055     convlstm_{n}_T{T}_TINC{TINC}_t5")

1056     # Encode & Generate
1057     X = tf.dtypes.cast(X[:,0:5], tf.float32)
1058     y_pred = np.squeeze(convlstm3d(X[...,:np.newaxis], training=False))
1059     y_pred = np.asarray(y_pred>0.5, bool)

1060     d.plotFromVoxels(y_pred[0], f"{{n}}_T{T}_TINC{TINC}_t1_pred")
1061     d.plotMeshFromVoxels(y_pred[0], obj=f"objects/T{T}_TINC{TINC}/convlstm_{n}_T
1062     {T}_TINC{TINC}_t1_pred")
1063     d.plotFromVoxels(y_pred[1], f"{{n}}_T{T}_TINC{TINC}_t1_pred")
1064     d.plotMeshFromVoxels(y_pred[1], obj=f"objects/T{T}_TINC{TINC}/convlstm_{n}_T
1065     {T}_TINC{TINC}_t2_pred")
1066     d.plotFromVoxels(y_pred[2], f"{{n}}_T{T}_TINC{TINC}_t1_pred")
1067     d.plotMeshFromVoxels(y_pred[2], obj=f"objects/T{T}_TINC{TINC}/convlstm_{n}_T
1068     {T}_TINC{TINC}_t3_pred")
1069     d.plotFromVoxels(y_pred[3], f"{{n}}_T{T}_TINC{TINC}_t1_pred")
1070     d.plotMeshFromVoxels(y_pred[3], obj=f"objects/T{T}_TINC{TINC}/convlstm_{n}_T
1071     {T}_TINC{TINC}_t4_pred")
1072     d.plotFromVoxels(y_pred[4], f"{{n}}_T{T}_TINC{TINC}_t1_pred")
1073     d.plotMeshFromVoxels(y_pred[4], obj=f"objects/T{T}_TINC{TINC}/convlstm_{n}_T
1074     {T}_TINC{TINC}_t5_pred")

1075     n+=1
1076     if n==N_PREDICT:
1077         break

1078 predictions(test_objects)

```

Listing A.3: ConvLSTM3D Prediction Algorithm

A.4 ConvLSTM3D Long-Term Prediction Algorithm

```
1000 import os
1001 import tensorflow as tf
1002 from keras.models import load_model
1003 import numpy as np
1004 # OWN LIBRARIES
1005 from utils import parse_tfrecords_4D_16_for_training_ConvLSTM as p4d
1006 from utils import dataIO as d

1008 base_dir = r"Z:\Master_Thesis\code\99_FINAL_MODELS_FOR_THESIS"
1009
1010 MODEL_LOCATION      = os.path.join(base_dir, "convlstm3d_16_T20_TINC1")
1011
1012 TEST_DATASET_LOC    = "Z:/Master_Thesis/data/02_4D_test_datasets"
1013
1014 T                  = int(MODEL_LOCATION[-8:-6])
1015 TINC                = int(MODEL_LOCATION[-1])
1016 N_PREDICT          = 5
1017 MAX_TIME            = 5

1018 if not os.path.exists(os.path.join(base_dir, f'objects_longterm/T{T}_TINC{TINC}')):
1019     os.makedirs(os.path.join(base_dir, f'objects_longterm/T{T}_TINC{TINC}'))
1020 #####
1021 ##### GET Data/ Objects #####
1022 #####
1023 #####
1024 TEST_Filenames = tf.io.gfile.glob(os.path.join(TEST_DATASET_LOC, "*tfrecords"))[:N_TEST]
1025 test_set_size = len(TEST_Filenames)

1026 print()
1027 print("Total Number of TFRecord Files to Test : ", test_set_size)
1028 print()

1029 # Convert/Decode dataset from tfrecords file for training
1030 test_objects = p4d.get_dataset(TEST_Filenames, shuffle=True, batch_size=1, t=T, tinc=TINC, normalize=False)
1031
1032 convlstm3d_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*ConvLSTM3D*.h5"))
1033 convlstm3d = load_model(convlstm3d_model[-1])

1034 def predictions(objects):
1035     n=0
1036     for X, y in objects:
1037         d.plotFromVoxels(np.squeeze(X[0,0]), f'{n}_T{T}_TINC{TINC}_t-1')
1038         d.plotMeshFromVoxels(np.squeeze(X[0,0]), obj=f"objects_longterm/T{T}_TINC{TINC}/convlstm_{n}_T{T}_TINC{TINC}_t0")
1039         d.plotFromVoxels(np.squeeze(X[0,1]), f'{n}_T{T}_TINC{TINC}_t-2')
```

```

1044     d.plotMeshFromVoxels(np.squeeze(X[0,1]), obj=f"objects_longterm/T{T}_TINC{  
TINC}/convlstm_{n}_T{T}_TINC{TINC}_t1")  
    d.plotFromVoxels(np.squeeze(X[0,2]), f"{n}_T{T}_TINC{TINC}_t0")  
    d.plotMeshFromVoxels(np.squeeze(X[0,2]), obj=f"objects_longterm/T{T}_TINC{  
TINC}/convlstm_{n}_T{T}_TINC{TINC}_t2")  
    d.plotFromVoxels(np.squeeze(X[0,3]), f"{n}_T{T}_TINC{TINC}_t1")  
    d.plotMeshFromVoxels(np.squeeze(X[0,3]), obj=f"objects_longterm/T{T}_TINC{  
TINC}/convlstm_{n}_T{T}_TINC{TINC}_t3")  
    d.plotFromVoxels(np.squeeze(X[0,4]), f"{n}_T{T}_TINC{TINC}_t1")  
    d.plotMeshFromVoxels(np.squeeze(X[0,4]), obj=f"objects_longterm/T{T}_TINC{  
TINC}/convlstm_{n}_T{T}_TINC{TINC}_t4")  
    d.plotFromVoxels(np.squeeze(X[0,5]), f"{n}_T{T}_TINC{TINC}_t1")  
    d.plotMeshFromVoxels(np.squeeze(X[0,5]), obj=f"objects_longterm/T{T}_TINC{  
TINC}/convlstm_{n}_T{T}_TINC{TINC}_t5")  
    d.plotFromVoxels(np.squeeze(X[0,6]), f"{n}_T{T}_TINC{TINC}_t1")  
    d.plotMeshFromVoxels(np.squeeze(X[0,6]), obj=f"objects_longterm/T{T}_TINC{  
TINC}/convlstm_{n}_T{T}_TINC{TINC}_t6")  
    d.plotFromVoxels(np.squeeze(X[0,7]), f"{n}_T{T}_TINC{TINC}_t1")  
    d.plotMeshFromVoxels(np.squeeze(X[0,7]), obj=f"objects_longterm/T{T}_TINC{  
TINC}/convlstm_{n}_T{T}_TINC{TINC}_t7")  
    d.plotFromVoxels(np.squeeze(X[0,8]), f"{n}_T{T}_TINC{TINC}_t1")  
    d.plotMeshFromVoxels(np.squeeze(X[0,8]), obj=f"objects_longterm/T{T}_TINC{  
TINC}/convlstm_{n}_T{T}_TINC{TINC}_t8")  
    d.plotFromVoxels(np.squeeze(X[0,9]), f"{n}_T{T}_TINC{TINC}_t1")  
    d.plotMeshFromVoxels(np.squeeze(X[0,9]), obj=f"objects_longterm/T{T}_TINC{  
TINC}/convlstm_{n}_T{T}_TINC{TINC}_t9")  
  
1062    X = tf.dtypes.cast(X[:,0:5], tf.float32)  
1063    for t in range(4,y.shape[1]-1):  
1064        y_pred = np.squeeze(convlstm3d(X[...,np.newaxis], training=False))  
1065        y_pred = np.asarray(y_pred>0.5, bool)[-1]  
  
1066        d.plotFromVoxels(y_pred, f"{n}_T{T}_TINC{TINC}_t{t+1}_pred")  
1067        d.plotMeshFromVoxels(y_pred, obj=f"objects_longterm/T{T}_TINC{TINC}/  
convlstm_{n}_T{T}_TINC{TINC}_t{t+1}_pred")  
  
1070        if t==MAX_TIME+4-1:  
1071            break  
  
1072    X = np.concatenate((X, np.expand_dims(y_pred[np.newaxis,...], axis=1)),  
axis=1)[0,1:6][np.newaxis,...]  
1074  
1075    n+=1  
1076    if n==N_PREDICT:  
1077        break  
1078  
1080 predictions(test_objects)

```

Listing A.4: ConvLSTM3D Long-Term Prediction Algorithm

B GAN-based 3D Next Frame Prediction

B.1 Single Frame Input Training Algorithm

```
1000 import os
1001 from shutil import copyfile
1002 import glob
1003 import sys
1004 import numpy as np
#np.set_printoptions(threshold=sys.maxsize)
1006 import time
1007 import tensorflow as tf
1008 from tensorflow.keras import Model
from tensorflow.keras.layers import Dense, BatchNormalization, LayerNormalization,
    LeakyReLU, Reshape, Conv3DTranspose, Conv3D, Flatten, ReLU, Dropout, Concatenate
    , Input, UpSampling3D, AveragePooling3D
1009 from tensorflow.keras.activations import sigmoid, tanh
1010 from tensorflow.keras.optimizers import Adam
1011 import tensorflow_addons as tfa
1012
1013 # OWN LIBRARIES
1014 from utils.loss_plotter import render_graphs
1015 from utils import dataIO as d
1016 from utils import parse_tfrecords_4D_16_for_training as p4d
1017 from utils.logger import Logger
1018
1019 ##### INPUTS #####
1020 #####
1021 #####
1022 N_EPOCHS          = 10000           # Default: 10000 – Max. training epochs OR
    Define Max. Iterations
1023 MAX_ITERATIONS     = 20000           # Default: -1 – Max. Training Iterations (
    Max. Number of processed Batches) before training stops, if "-1" Training stops
    after reaching N_EPOCHS
1024 TRAIN_DATASET_LOC  = "Z:/Master_Thesis/data/00_4D_training_datasets"
1025 VALID_DATASET_LOC  = "Z:/Master_Thesis/data/01_4D_validation_datasets"
1026 N_TRAINING         = 10000           # Number of Files for Training: -1 == USE
    ALL
1027 N_VALIDATION       = 1000            # Number of Files for Validation: -1 == USE
    ALL
1028 T                  = 80              # Default: 80 – Even Number of Length of
    Time History to use, MUST BE EVENLY DIVISIBLE BY TINC, e.g. 80 means that 80
    steps out of all 128 steps are used, where 24 steps at the beginning and 24
```

```

    steps at the end of the time history are cropped.
1030 TINC          = 4           # Default: 1 – Allowed: (1, 2, 4, 8) – e.g.:
    4 --> 80/4 = 20 Timepoints --> 19 PAIRS of T0->T1 --> TIME INCREMENT FOR
    EXTRACTING TIMESTEPS PER DATAPoint – (Check parse_tfrecords_4D_##_for_training
    for extraction details)
NORMALIZE      = False        # NORMALIZE DATA to ZERO MEAN --> from [0;1]
    to [-0.5, 0.5]
1032 PRINT_D_LOSSES = False     # LOG DISCRIMINATOR LOSSES SPLIT UP IN Fake,
    Real, GP and EPS Losses
LOGGING        = True         # LOG TRAINING RUN TO FILE (If Kernel get
    Interupted (user or error) Kernel needs to be restarted, otherwise log file is
    locked to be deleted!)
#####
# Restart Training?
1036 RESTART_TRAINING = False
RESTART_ITERATION = 50000       # the epoch number when the last training
    was stopped, has no influnce on loading the restart file, just used for tracking
    information numbering
1038 RESTART_FOLDER_NAME = "20240221_012216_aengan3d_16"
#####
# Output Settings
VAL_SAVE_FREQ   = 1000        # Default: 1000 – OFF: -1, every n training
    iterations (n batches) check validation loss
1042 WRITE_RESTART_FREQ = 1000   # Default: 1000 – every n training
    iterations (n batches) save restart checkpoint
KEEP_BEST_ONLY  = True        # Default: True – Keep only the best model (
    KEEP_LAST_N=1)
1044 BEST_METRIC    = "JACCARD" # Default: JACCARD – "JACCARD" or "MSE"
# if KEEP_BEST_ONLY = False define max. Number of last models to keep:
1046 KEEP_LAST_N   = 5          # Default: 5 – Max. number of last files to
    keep
N_SAMPLES       = 5          # Default: 5 – Number of object samples to
    generate after each epoch
1048 #####
##### HYPERPARAMETERS #####
1050 #####
GD_ARCHITECTURE = "PreRes"    # Default: "PreRes" – "PreRes" or "Classic"
    Convolutions
1052 BATCH_SIZE     = 32         # Default: 32 – Number of Examples per Batch
    / Probably Increase for better Generalization
Z_SIZE           = 256         # Default: 256 – Latent Random Vector Size
1054 FEATUREMAP_BASE_DIM = 256   # Default: 256 – Max Featuremap size (Maybe
    increase for learning larger/more diverse datasets...)
#####
# WGAN-GP Settings
1056 GP_WEIGHT      = 100        # Default: 10 – WGAN-GP gradient penalty
    weight
1058 EPS_WEIGHT     = 0.0        # Default: 0.001 – WGAN-GP epsilon penalty
    weight (real logits penalty)
D_STEPS          = 1           # Default: 5 – WGAN-GP Disc. training steps

```

```

1060 #####
1061 # WGAN-GP Optimizer Settings
1062 G_LR = 0.0001 # Default: 0.0001 – Generator Learning Rate
1063 D_LR = 0.0001 # Default: 0.0001 – Discriminator Learning
1064 Rate
1065 CLR = False # Default: False – Use Cyclic Learning Rate,
1066 minLR=G_LR or D_LR, maxLR=10*G_LR or 10*D_LR
1067 #####
1068 # Dropout
1069 DROPOUT_RATE = 0.2 # Default: 0.2
1070 #####
1071 # weight_initializer & regularizer
1072 weight_initializer = "glorot_uniform" # "glorot_uniform" "glorot_normal" or "
1073 he_uniform"
1074 weight_regularizer = "L1" # tf.keras.regularizers.L1(l1=0.01) #
1075 Default l1=0.01
1076 #####
1077 # USE BIAS IN GENERATOR
1078 USE_BIAS = True
1079 bias_initializer = "zeros" # DEFAULT: "zeros" – "ones" or tf.keras.initializers.
1080 Constant(0.5)
1081 #####
1082 # DEFAULT – DONT TOUCH SETTINGS
1083 SIZE = 16 # Default: 16 – Object Grid Size
1084 BASE_SIZE = 4 # Default: 4 – Base Resolution to start from
1085 for generator and to end with for discriminator
1086 KERNEL_SIZE = 5 # Default: 5 – Lower values learn pretty bad
1087 , higher than 5 doesnt change training much
1088 #####
1089 ##### NETWORKS #####
1090 #####
1091 # latent vector input
1092 latent_vector = Input(shape=(Z_SIZE,))
1093 #-----
1094 ##### ENCODER #####
1095 #-----
1096 def obj_input(object_dim=(SIZE, SIZE, SIZE, 1)):
1097     obj_in = Input(shape=object_dim)
1098     return obj_in
1099 #####
1100 # Encoder PreResidual Block
1101 def EncoderPreResBlock(x, in_filters, out_filters, kernel_size, strides,
1102 kernel_initializer, kernel_regularizer, padding):
1103     # Define Shortcut
1104     if strides == 1 and in_filters == out_filters: # Identity Skip Connection
1105         shortcut = x
1106     elif strides == 2 and in_filters == out_filters: # Identity Skip Connection
1107         shortcut = AveragePooling3D(pool_size=(strides, strides, strides), strides=(
1108             strides, strides, strides))(x)

```

```

    elif strides == 1 and in_filters != out_filters: # Change of Number of Filters
        Skip
        shortcut = Conv3D(filters=out_filters, kernel_size=1, strides=1,
                           kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
                           padding="same")(x)
    elif strides == 2 and in_filters != out_filters: # Skip with Change in Size and
        Number of Filters
        shortcut = AveragePooling3D(pool_size=(strides, strides, strides), strides=(strides,
                           strides, strides))(x)
        shortcut = Conv3D(filters=out_filters, kernel_size=1, strides=1,
                           kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
                           padding="same")(shortcut)

# Define Activation on Input
x = BatchNormalization()(x)
x = LeakyReLU()(x)
x = Dropout(DROPOUT_RATE)(x)
x = Conv3D(filters=out_filters, kernel_size=kernel_size, strides=strides,
                           kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
                           padding=padding)(x)

return shortcut + x
#####
# DEFINE ENCODER NETWORK
def ClassicEncoder():
    obj_in = obj_input()

    x = Conv3D(filters=FEATUREMAP_BASE_DIM//4, kernel_size=KERNEL_SIZE, strides=2,
                           kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
                           padding="same")(obj_in)
    x = BatchNormalization()(x)
    x = LeakyReLU()(x)
    x = Dropout(DROPOUT_RATE)(x)

    x = Conv3D(filters=FEATUREMAP_BASE_DIM//2, kernel_size=KERNEL_SIZE, strides=2,
                           kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
                           padding="same")(x)
    x = BatchNormalization()(x)
    x = LeakyReLU()(x)
    x = Dropout(DROPOUT_RATE)(x)

    x = Conv3D(filters=FEATUREMAP_BASE_DIM, kernel_size=KERNEL_SIZE, strides=1,
                           kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
                           padding="same")(x)

    x = Flatten()(x)

    out_layer = Dense(Z_SIZE, activation=None)(x)

# define model

```

```

1136     model = Model(obj_in , out_layer)
1137     return model
1138
1139 def PreResEncoder():
1140     obj_in = obj_input()
1141
1142     # Begin Activation
1143     x = Conv3D(filters=FEATUREMAP_BASE_DIM//8 , kernel_size=KERNEL_SIZE, strides=1,
1144     input_shape=[SIZE, SIZE, SIZE, 1], kernel_initializer=weight_initializer ,
1145     kernel_regularizer=weight_regularizer , padding="same")(obj_in)
1146
1147     # PreRes Blocks
1148     x = EncoderPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//8 , out_filters=
1149     FEATUREMAP_BASE_DIM//4 , kernel_size=KERNEL_SIZE, strides=2, kernel_initializer=
1150     weight_initializer , kernel_regularizer=weight_regularizer , padding="same")
1151     x = EncoderPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//4 , out_filters=
1152     FEATUREMAP_BASE_DIM//2 , kernel_size=KERNEL_SIZE, strides=2, kernel_initializer=
1153     weight_initializer , kernel_regularizer=weight_regularizer , padding="same")
1154     x = EncoderPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//2 , out_filters=
1155     FEATUREMAP_BASE_DIM , kernel_size=KERNEL_SIZE, strides=1, kernel_initializer=
1156     weight_initializer , kernel_regularizer=weight_regularizer , padding="same")
1157
1158     x = BatchNormalization()(x)
1159     x = LeakyReLU()(x)
1160     x = Dropout(DROPOUT_RATE)(x)
1161
1162     # Flatten (reshape) and Linear (dense) Layer
1163     x = Flatten()(x)
1164
1165     out_layer = Dense(Z_SIZE, activation=None)(x)
1166
1167     # define model
1168     model = Model(obj_in , out_layer)
1169     return model
1170
1171 #####
1172 #
1173 ###### GENERATOR #####
1174 #
1175
1176 def latent_input(latent_vector=latent_vector , BASE_SIZE=BASE_SIZE):
1177     base = BASE_SIZE * BASE_SIZE * BASE_SIZE
1178
1179     latent_dense = Dense(units = FEATUREMAP_BASE_DIM*base)(latent_vector)
1180     latent_reshape = Reshape((BASE_SIZE, BASE_SIZE, BASE_SIZE, FEATUREMAP_BASE_DIM))(latent_dense)
1181     return latent_reshape
1182
1183 #####

```

```

# Generator PreResidual Block
1178 def GPreResBlock(x, in_filters, out_filters, kernel_size, up_size,
    kernel_initializer, kernel_regularizer, padding, use_bias, bias_initializer):
    # Define Shortcut
    1180 if up_size == 1 and in_filters == out_filters: # Identity Skip Connection
        shortcut = x
    1182 elif up_size == 2 and in_filters == out_filters: # Identity Skip Connection
        shortcut = UpSampling3D(size=(up_size, up_size, up_size))(x)
    1184 elif up_size == 1 and in_filters != out_filters: # Change of Number of Filters
        # Skip
        1185     shortcut = Conv3D(filters=out_filters, kernel_size=1, strides=1,
            kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
            padding=padding, use_bias=use_bias, bias_initializer = bias_initializer)(x)
    1186 elif up_size == 2 and in_filters != out_filters: # Skip with Change in Size and
        # Number of Filters
        1187     shortcut = UpSampling3D(size=(up_size, up_size, up_size))(x)
        1188     shortcut = Conv3D(filters=out_filters, kernel_size=1, strides=1,
            kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
            padding=padding, use_bias=use_bias, bias_initializer = bias_initializer)(
                shortcut)

    1190 # Define Activation on Input
    1191 x = BatchNormalization()(x)
    1192 x = ReLU()(x)
    1193 if up_size != 1:
    1194     x = UpSampling3D(size=(up_size, up_size, up_size))(x)
    1195 x = Conv3DTranspose(filters=out_filters, kernel_size=kernel_size, strides=1,
        kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
        padding=padding, use_bias=use_bias, bias_initializer = bias_initializer)(x)
    1196
    1197     return shortcut + x
    1198 #####
    1199 # DEFINE GENERATOR NETWORK
    1200 def ClassicGenerator():
        latent_in = latent_input()
        1201 x = BatchNormalization()(latent_in)
        x = ReLU()(x)
    1203
        x = Conv3DTranspose(filters=FEATUREMAP_BASE_DIM, kernel_size=KERNEL_SIZE,
            strides=1, kernel_initializer=weight_initializer, kernel_regularizer=
            weight_regularizer, padding="same", use_bias=USE_BIAS, bias_initializer =
            bias_initializer)(x)
        x = BatchNormalization()(x)
        x = ReLU()(x)
    1207
        x = UpSampling3D(size=(2, 2, 2))(x)
        x = Conv3DTranspose(filters=FEATUREMAP_BASE_DIM//2, kernel_size=KERNEL_SIZE,
            strides=1, kernel_initializer=weight_initializer, kernel_regularizer=
            weight_regularizer, padding="same", use_bias=USE_BIAS, bias_initializer =
            bias_initializer)(x)

```

```

1212     x = BatchNormalization()(x)
1213     x = ReLU()(x)

1214     x = UpSampling3D(size=(2, 2, 2))(x)
1215     x = Conv3DTranspose(filters=FEATUREMAP_BASE_DIM//4, kernel_size=KERNEL_SIZE,
1216                          strides=1, kernel_initializer=weight_initializer, kernel_regularizer=
1217                          weight_regularizer, padding="same", use_bias=USE_BIAS, bias_initializer =
1218                          bias_initializer)(x)
1219     x = BatchNormalization()(x)
1220     x = ReLU()(x)

1221     x = Conv3DTranspose(filters=1, kernel_size=KERNEL_SIZE, strides=1,
1222                           kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
1223                           padding="same", use_bias=USE_BIAS, bias_initializer = bias_initializer)(x)

1224 if NORMALIZE:
1225     out_layer = tanh(x)
1226 else:
1227     out_layer = sigmoid(x)

1228 model = Model(latent_vector, out_layer)

1229 return model

1230 def PreResGenerator():
1231     x = latent_input()

1232     # PreRes Blocks
1233     x = GPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM      , out_filters=
1234                       FEATUREMAP_BASE_DIM//2, kernel_size=KERNEL_SIZE, up_size=1, kernel_initializer=
1235                       weight_initializer, kernel_regularizer=weight_regularizer, padding="same",
1236                       use_bias=USE_BIAS, bias_initializer = bias_initializer)
1237     x = GPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//2, out_filters=
1238                       FEATUREMAP_BASE_DIM//4, kernel_size=KERNEL_SIZE, up_size=2, kernel_initializer=
1239                       weight_initializer, kernel_regularizer=weight_regularizer, padding="same",
1240                       use_bias=USE_BIAS, bias_initializer = bias_initializer)
1241     x = GPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//4, out_filters=
1242                       FEATUREMAP_BASE_DIM//8, kernel_size=KERNEL_SIZE, up_size=2, kernel_initializer=
1243                       weight_initializer, kernel_regularizer=weight_regularizer, padding="same",
1244                       use_bias=USE_BIAS, bias_initializer = bias_initializer)

1245     x = BatchNormalization()(x)
1246     x = ReLU()(x)
1247     # Final Convolution
1248     x = Conv3DTranspose(filters=1, kernel_size=KERNEL_SIZE, strides=1,
1249                         kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
1250                         padding="same", use_bias=USE_BIAS, bias_initializer = bias_initializer)(x)

1251     # End Activation
1252     if NORMALIZE:

```

```

        out_layer = tanh(x)
    else:
        out_layer = sigmoid(x)

1248 model = Model(latent_vector, out_layer)
1250
1252 return model

1254 #####
1255 #
1256 ##### DISCRIMINATOR #####
1257 #

1258 def real_obj_input_discriminator(object_dim=(SIZE, SIZE, SIZE, 1)):
1259     real_obj_in = Input(shape=object_dim)
1260     return real_obj_in
1261 def cond_obj_input_discriminator(object_dim=(SIZE, SIZE, SIZE, 1)):
1262     cond_obj_in = Input(shape=object_dim)
1263     return cond_obj_in

1266 ##### Discriminator PreResidual Block
1267 def DPreResBlock(x, in_filters, out_filters, kernel_size, strides,
1268   kernel_initializer, kernel_regularizer, padding):
1269     # Define Shortcut
1270     if strides == 1 and in_filters == out_filters: # Identity Skip Connection
1271         shortcut = x
1272     elif strides == 2 and in_filters == out_filters: # Identity Skip Connection
1273         shortcut = AveragePooling3D(pool_size=(strides, strides, strides), strides=(strides, strides, strides))(x)
1274     elif strides == 1 and in_filters != out_filters: # Change of Number of Filters
1275         # Skip
1276         shortcut = Conv3D(filters=out_filters, kernel_size=1, strides=1,
1277           kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
1278           padding="same")(x)
1279     elif strides == 2 and in_filters != out_filters: # Skip with Change in Size and
1280       # Number of Filters
1281         shortcut = AveragePooling3D(pool_size=(strides, strides, strides), strides=(strides, strides, strides))(x)
1282         shortcut = Conv3D(filters=out_filters, kernel_size=1, strides=1,
1283           kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
1284           padding="same")(shortcut)

1285     # Define Activation on Input
1286     x = LayerNormalization()(x)
1287     x = LeakyReLU()(x)
1288     x = Dropout(DROPOUT_RATE)(x)
1289     x = Conv3D(filters=out_filters, kernel_size=kernel_size, strides=strides,
1290       kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,

```

```

padding=padding)(x)

1286     return shortcut + x
#####
1288 # DEFINE DISCRIMINATOR NETWORK
def ClassicDiscriminator():
1290     real_obj_in = real_obj_input_discriminator()
1291     cond_obj_in = cond_obj_input_discriminator()

1292     # merge label_conditioned_generator and latent_input output
1294     x = Concatenate()([real_obj_in, cond_obj_in])

1296     x = Conv3D(filters=FEATUREMAP_BASE_DIM//4, kernel_size=KERNEL_SIZE, strides=2,
1297                 kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
1298                 padding="same")(x)
1299     x = LayerNormalization()(x)
1300     x = LeakyReLU()(x)
1301     x = Dropout(DROPOUT_RATE)(x)

1303     x = Conv3D(filters=FEATUREMAP_BASE_DIM//2, kernel_size=KERNEL_SIZE, strides=2,
1304                 kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
1305                 padding="same")(x)
1306     x = LayerNormalization()(x)
1307     x = LeakyReLU()(x)
1308     x = Dropout(DROPOUT_RATE)(x)

1310     x = Conv3D(filters=FEATUREMAP_BASE_DIM, kernel_size=KERNEL_SIZE, strides=1,
1311                 kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
1312                 padding="same")(x)

1314     x = Flatten()(x)

1316     out_layer = Dense(1, activation=None)(x)

1318     # define model
1319     model = Model([real_obj_in, cond_obj_in], out_layer)
1320     return model

1322 def PreResDiscriminator():
1323     real_obj_in = real_obj_input_discriminator()
1324     cond_obj_in = cond_obj_input_discriminator()

1326     # merge label_conditioned_discriminator and latent_input output
1327     x = Concatenate()([real_obj_in, cond_obj_in])

1329     # Begin Activation
1330     x = Conv3D(filters=FEATUREMAP_BASE_DIM//8, kernel_size=KERNEL_SIZE, strides=1,
1331                 input_shape=[SIZE, SIZE, SIZE, 1], kernel_initializer=weight_initializer,
1332                 kernel_regularizer=weight_regularizer, padding="same")(x)

```

```

1326 # PreRes Blocks
1327 x = DPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//8, out_filters=
1328     FEATUREMAP_BASE_DIM//4, kernel_size=KERNEL_SIZE, strides=2, kernel_initializer=
1329     weight_initializer, kernel_regularizer=weight_regularizer, padding="same")
1330 x = DPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//4, out_filters=
1331     FEATUREMAP_BASE_DIM//2, kernel_size=KERNEL_SIZE, strides=2, kernel_initializer=
1332     weight_initializer, kernel_regularizer=weight_regularizer, padding="same")
1333 x = DPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//2, out_filters=
1334     FEATUREMAP_BASE_DIM , kernel_size=KERNEL_SIZE, strides=1, kernel_initializer=
1335     weight_initializer, kernel_regularizer=weight_regularizer, padding="same")
1336
1337 x = LayerNormalization()(x)
1338 x = LeakyReLU()(x)
1339 x = Dropout(DROPOUT_RATE)(x)
1340
1341 # Flatten (reshape) and Linear (dense) Layer
1342 x = Flatten()(x)
1343
1344 out_layer = Dense(1, activation=None)(x)
1345
1346 # define model
1347 model = Model([real_obj_in, cond_obj_in], out_layer)
1348 return model
1349
1350 ##### COMPILE NETWORKS #####
1351
1352 if GD_ARCHITECTURE == "PreRes":
1353     encoder, generator, discriminator = PreResEncoder(), PreResGenerator(),
1354     PreResDiscriminator()
1355 elif GD_ARCHITECTURE == "Classic":
1356     encoder, generator, discriminator = ClassicEncoder(), ClassicGenerator(),
1357     ClassicDiscriminator()
1358 else:
1359     ValueError("Generator/Disriminator Architecture not Available. Choose between 'PreRes' or 'Classic'!")
1360 encoder.compile()
1361 generator.compile()
1362 discriminator.compile()
1363
1364 ##### LOSS FUNCTION #####
1365
1366 @tf.function
1367 def encgen_loss(real_objects):
1368
1369     real_objects_t0 = tf.dtypes.cast(real_objects[:,0], tf.float32)
1370     real_objects_t1 = tf.dtypes.cast(real_objects[:,1], tf.float32)
1371
1372     # Latent Encoding

```

```

latent_code = encoder(real_objects_t0, training=True)
1368
pred_objects_t1 = generator(latent_code, training=True)
1370 fake_logits = discriminator([pred_objects_t1, real_objects_t1], training=True)
1371 loss = wgan_gp_generator_loss(fake_logits)

1372
mse = tf.keras.metrics.mean_squared_error(real_objects_t1, pred_objects_t1)
1374 MSE_loss = tf.reduce_mean(mse)

1376
return loss, MSE_loss

1378 @tf.function
def encdisc_loss(real_objects):
1380
real_objects_t0 = tf.dtypes.cast(real_objects[:,0], tf.float32)
1382 real_objects_t1 = tf.dtypes.cast(real_objects[:,1], tf.float32)

1384 # Latent Encoding
latent_code = encoder(real_objects_t0, training=True)
1386
pred_objects_t1 = generator(latent_code, training=True)
1388 fake_logits = discriminator([pred_objects_t1, real_objects_t1], training=True)
1389 real_logits = discriminator([real_objects_t1, real_objects_t1], training=True)
1390 D_loss, fake_loss, real_loss, gp, ep = wgan_gp_discriminator_loss(real_logits,
fake_logits, real_objects_t1, pred_objects_t1, real_objects_t1, GP_WEIGHT,
EPS_WEIGHT)

1392
return D_loss, fake_loss, real_loss, gp, ep

1394 @tf.function
def val_encdisc_loss(val_real_objects):
1396
val_real_objects_t0 = tf.dtypes.cast(val_real_objects[:,0], tf.float32)
1398 val_real_objects_t1 = tf.dtypes.cast(val_real_objects[:,1], tf.float32)

1400 # Latent Encoding
val_latent_code = encoder(val_real_objects_t0, training=False)
1402
val_pred_objects_t1 = generator(val_latent_code, training=False)
1404 val_fake_logits = discriminator([val_pred_objects_t1, val_real_objects_t1],
training=False)
1405 val_real_logits = discriminator([val_real_objects_t1, val_real_objects_t1],
training=False)
1406 D_loss, fake_loss, real_loss, gp, ep = wgan_gp_discriminator_loss(
val_real_logits, val_fake_logits, val_real_objects_t1, val_pred_objects_t1,
val_real_objects_t1, GP_WEIGHT, EPS_WEIGHT)

1408
return D_loss, fake_loss, real_loss, gp, ep

1410 # WGAN LOSSES

```

```

@tf.function
1412 def wgan_gp_generator_loss(fake_logits):
    G_loss = -tf.math.reduce_mean(fake_logits)
    return G_loss
@tf.function
1416 def wgan_gp_discriminator_loss(real_logits, fake_logits, real_objects, fake_objects,
    cond_objects, GP_WEIGHT, EPS_WEIGHT):
    """ Calculates the gradient penalty.

1418 This loss is calculated on an interpolated image
1420 and added to the discriminator loss.
    """
1422 # 1. Get the interpolated object
    alpha = tf.random.uniform([real_objects.shape[0], 1, 1, 1, 1], minval=0.0,
    maxval=1.0, seed=None)
    differences = fake_objects - real_objects
    interpolates = real_objects + (alpha*differences)
1426
# 2. Calculate the gradients w.r.t to this interpolated object.
1428 gradients = tf.gradients(discriminator([interpolates, cond_objects], training=
    False), [interpolates])[0]
    norm = tf.math.sqrt(tf.math.reduce_sum(tf.square(gradients), axis=[1,2,3,4])) # norm over all gradients of 3D Grid
1430
# 3. Calculate the norm of the gradients.
1432 gradient_penalty = tf.reduce_mean((norm-1.0)**2)

1434 # Add the gradient penalty to the original discriminator loss and MEAN over
1436 BATCH_SIZE + epsilon penalty
    fake_loss = tf.math.reduce_mean(fake_logits)
    real_loss = tf.math.reduce_mean(real_logits)
    gp = GP_WEIGHT * gradient_penalty
    ep = EPS_WEIGHT * tf.math.reduce_mean((real_logits)**2)

1440 D_loss = fake_loss - real_loss + gp + ep

1442 return D_loss, fake_loss, real_loss, gp, ep

1444 ##### TRAINING STEP #####
1446
@tf.function
1448 def train_step(obj_batch):

1450     real_objects = obj_batch[..., np.newaxis]

1452     for i in range(D_STEPS):
        with tf.GradientTape() as disc_tape:
            # discriminator loss
            EncDisc_loss = encdisc_loss(real_objects)

```

```

1456     gradients_of_discriminator = disc_tape.gradient(EncDisc_loss[0] ,
discriminator.trainable_variables)
1458     discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator ,
discriminator.trainable_variables))

1460 with tf.GradientTape() as gen_tape, tf.GradientTape() as enc_tape:
1461     # generator loss
1462     EncGen_loss = encgen_loss(real_objects)

1464     gradients_of_encoder = enc_tape.gradient(EncGen_loss[0], encoder.
trainable_variables)
1465     encoder_optimizer.apply_gradients(zip(gradients_of_encoder, encoder.
trainable_variables))

1466     gradients_of_generator = gen_tape.gradient(EncGen_loss[0], generator.
trainable_variables)
1467     generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.
trainable_variables))

1470 return EncGen_loss, EncDisc_loss

1472 @tf.function
1473 def validation_step(val_obj_batch):
1474     val_real_objects = val_obj_batch[..., np.newaxis]

1476     Val_EncDisc_Loss = val_encdisc_loss(val_real_objects)

1478     Val_MSE_reconstruction_loss = MSE_reconstruction_loss(val_real_objects)

1480 return Val_EncDisc_Loss, Val_MSE_reconstruction_loss

1482
1483 @tf.function
1484 def MSE_reconstruction_loss(real_objects):
1485     real_objects_t0 = tf.dtypes.cast(real_objects[:, 0], tf.float32)
1486     real_objects_t1 = tf.dtypes.cast(real_objects[:, 1], tf.float32)

1488     latent_code = encoder(real_objects_t0, training=False)
1489     pred_objects_t1 = generator(latent_code, training=False)

1490     mse = tf.keras.metrics.mean_squared_error(real_objects_t1, pred_objects_t1)
1491     MSE_loss = tf.reduce_mean(mse)

1494 return MSE_loss

1496 def JACCARD_reconstruction_loss(val_obj_batch):
1497     val_real_objects = val_obj_batch[..., np.newaxis]

1498     val_real_objects_t0 = tf.dtypes.cast(val_real_objects[:, 0], tf.float32)

```

```

1500    val_real_objects_t1 = tf.dtypes.cast(val_real_objects[:,1], tf.float32)

1502    val_latent_code = encoder(val_real_objects_t0, training=False)
    val_pred_objects_t1 = generator(val_latent_code, training=False)

1504
1505    if NORMALIZE:
1506        x = np.asarray(val_real_objects_t1 >0.0, bool)
        y = np.asarray(val_pred_objects_t1 >0.0, bool)
1507    else:
1508        x = np.asarray(val_real_objects_t1 >0.5, bool)
        y = np.asarray(val_pred_objects_t1 >0.5, bool)

1509
1510
1511    j_and = tf.math.reduce_sum(np.double(np.bitwise_and(x, y)), axis=[1,2,3,4])
    j_or = tf.math.reduce_sum(np.double(np.bitwise_or(x, y)), axis=[1,2,3,4])

1512    JACCARD_loss = tf.reduce_mean(np.nan_to_num(1.0 - j_and / j_or, nan=0.0))

1513
1514    return JACCARD_loss

1515
1516
1517    ##### SOME UTILITY FUNCTIONS #####
1518
1519
1520
1521
1522 def generate_sample_objects(objects, total_iterations, tag="pred"):
    # Random Sample at every VAL_SAVE_FREQ
1523    i=0
    generated_objects = []
1525
    for X in objects:
        X = tf.dtypes.cast(X, tf.float32)
        # Encode
        latent_code = encoder(X, training=False)

1527
        sample_object = generator(latent_code, training=False)
        if NORMALIZE:
            object_out = np.squeeze(sample_object>0.0)
        else:
            object_out = np.squeeze(sample_object>0.5)

1529
        try:
            d.plotMeshFromVoxels(object_out, obj=model_directory+tag+"_"+str(i)+"_"
1530            _next+"_"+str(total_iterations))
            except:
                print(f"Cannot generate STL, Marching Cubes Algo failed for sample {i}")
                # print("""Cannot generate STL, Marching Cubes Algo failed: Surface
1531                level must be within volume data range! \n
                # This may happen at the beginning of the training, if it still
1532                happens at later stages epoch>10 --> Check Object and try to change Marching
                Cube Threshold."""")

1533
        generated_objects.append(object_out)

```

```

1546         i+=1
1548
1549     print()
1550
1551     return generated_objects
1552
1552 def IOhousekeeping(model_directory , KEEP_LAST_N, VAL_SAVE_FREQ, RESTART_TRAINING,
1553 total_iterations , KEEP_BEST_ONLY, best_iteration):
1554     if total_iterations > KEEP_LAST_N*VAL_SAVE_FREQ and total_iterations % VAL_SAVE_FREQ == 0:
1555         if KEEP_BEST_ONLY:
1556             fileList_all = glob.glob(model_directory + "*_iter_*")
1557             fileList_best = glob.glob(model_directory + "*_iter_" + str(int(best_iteration)) + "*")
1558             fileList_del = [ele for ele in fileList_all if ele not in fileList_best]
1559
1560         else:
1561             fileList_del = glob.glob(model_directory + "*_iter_" + str(int(total_iterations -KEEP_LAST_N*VAL_SAVE_FREQ)) + "*")
1562
1562         # Iterate over the list of filepaths & remove each file .
1563         for filePath in fileList_del:
1564             os.remove(filePath)
1565
1566 #####
1566 ##### LAST PREPARATION STEPS BEFORE TRAINING STARTS #####
1567 #####
1568 #generate folders:
1569 if RESTART_TRAINING:
1570     model_directory = os.getcwd() + "/" + RESTART_FOLDER_NAME + "/"
1571 else:
1572     named_tuple = time.localtime() # get struct_time
1573     time_string = time.strftime("%Y%m%d_%H%M%S", named_tuple)
1574     model_directory = os.getcwd()+"/"+time_string+"_"+os.path.splitext(os.path.basename(__file__))[0] + "/"
1575     if not os.path.exists(model_directory):
1576         os.makedirs(model_directory)
1577
1578 # Save running python file to run directory
1579 if RESTART_TRAINING:
1580     named_tuple = time.localtime() # get struct_time
1581     time_string = time.strftime("%Y%m%d_%H%M%S", named_tuple)
1582     copyfile(__file__ , model_directory+time_string +"_restart_" +os.path.basename(__file__))
1583 else:
1584     copyfile(__file__ , model_directory+os.path.basename(__file__))
1585
1586 # Activate Console Logging:
1587 if LOGGING:
1588     sys.stdout = Logger(filename=model_directory+"log_out.txt")

```

```

1590
1591 # PRINT NETWORK SUMMARY
1592 print(encoder.summary())
1593 print(generator.summary())
1594 print(discriminator.summary())

1595 #####
1596 ##### GET DATA/OBJECTS #####
1597 #####
1598 TRAINING_Filenames = tf.io.gfile.glob(TRAIN_DATASET_LOC+"/*.tfrecords")[:N_TRAINING]
1599 VALIDATION_Filenames = tf.io.gfile.glob(VALID_DATASET_LOC+"/*.tfrecords")[:N_VALIDATION]

1600 training_set_size = len(TRAINING_Filenames)
1601 validation_set_size = len(VALIDATION_Filenames)
1602 training_samples = training_set_size * (T//TINC-1) // BATCH_SIZE * BATCH_SIZE
1603 validation_samples = validation_set_size * (T//TINC-1) // BATCH_SIZE * BATCH_SIZE
1604 print()
1605 print("Total Number of TFRecord Files to Train : ", training_set_size)
1606 print("Total Number of Samples to Train: ", training_samples)
1607 print()
1608 print("Total Number of TFRecord Files for Validation: ", validation_set_size)
1609 print("Total Number of Samples for Validation: ", validation_samples)
1610 print()

1611 # Convert/Decode dataset from tfrecords file for training
1612 train_objects = p4d.get_dataset(TRAINING_Filenames, shuffle=True, batch_size=BATCH_SIZE, t=T, tinc=TINC, normalize=NORMALIZE )
1613 val_objects = p4d.get_dataset(VALIDATION_Filenames, shuffle=False, batch_size=BATCH_SIZE, t=T, tinc=TINC, normalize=NORMALIZE )

1614 # save some real examples
1615 i=0
1616 test_objects = []
1617 for X in train_objects.take(N_SAMPLES):
1618     Xt0 = X[0][0]
1619     Xt1 = X[0][1]

1620     try:
1621         sample_name = model_directory+"train_"+str(i)+"_init"+str(t)
1622         d.plotMeshFromVoxels(np.array(Xt0), obj=sample_name)
1623         sample_name = model_directory+"train_"+str(i)+"_next"+str(t)
1624         d.plotMeshFromVoxels(np.array(Xt1), obj=sample_name)
1625     except:
1626         print("Example Object couldnt be generated with Marching Cube Algorithm. Probable Reason: Empty Input Tensor, All Zeros")

1627     test_objects.append([Xt0[... ,np.newaxis]]) # for validation of reconstruction during training , see generate_sample_objects():

1628     i+=1

```

```

1636 # save some validation examples
1637 i=0
1638 valid_objects = []
1639 for X in val_objects.take(N_SAMPLES):
1640     Xt0 = X[0][0]
1641     Xt1 = X[0][1]
1642
1643     try:
1644         sample_name = model_directory+"valid_"+str(i)+"_init"+str(t)
1645         d.plotMeshFromVoxels(np.array(Xt0), obj=sample_name)
1646         sample_name = model_directory+"valid_"+str(i)+"_next"+str(t)
1647         d.plotMeshFromVoxels(np.array(Xt1), obj=sample_name)
1648     except:
1649         print("Example Object couldnt be generated with Marching Cube Algorithm.
1650 Probable Reason: Empty Input Tensor, All Zeros")
1651
1652     valid_objects.append([Xt0[... ,np.newaxis]]) # for validation of reconstruction
1653     during training , see generate_sample_objects():
1654
1655 #####
1656 ##### OPTIMIZER #####
1657 #####
1658
1659 if CLR:
1660     steps_per_epoch = len(TRAINING_Filenames) // BATCH_SIZE
1661     clr_g = tfa.optimizers.CyclicalLearningRate(initial_learning_rate=G_LR,
1662                                                 maximal_learning_rate=G_LR*10,
1663                                                 scale_fn=lambda x: 1/(2.**(x-1)),
1664                                                 step_size=2 * steps_per_epoch
1665     )
1666     clr_d = tfa.optimizers.CyclicalLearningRate(initial_learning_rate=D_LR,
1667                                                 maximal_learning_rate=D_LR*10,
1668                                                 scale_fn=lambda x: 1/(2.**(x-1)),
1669                                                 step_size=2 * steps_per_epoch
1670     )
1671
1672     encoder_optimizer = Adam(learning_rate=clr_g , beta_1=0.5, beta_2=0.999, epsilon
1673 =1e-07)
1674     generator_optimizer = Adam(learning_rate=clr_g , beta_1=0.5, beta_2=0.999,
1675     epsilon=1e-07)
1676     discriminator_optimizer = Adam(learning_rate=clr_d , beta_1=0.5, beta_2=0.999,
1677     epsilon=1e-07)
1678
1679 else:
1680     encoder_optimizer = Adam(learning_rate=G_LR, beta_1=0.5, beta_2=0.999, epsilon=1
1681 e-07)
1682     generator_optimizer = Adam(learning_rate=G_LR, beta_1=0.5, beta_2=0.999, epsilon
1683 =1e-07)

```

```

discriminator_optimizer = Adam(learning_rate=D_LR, beta_1=0.5, beta_2=0.999,
epsilon=1e-07)

#####
# Define Checkpoint
checkpoint_path = model_directory+"checkpoints/"
checkpoint_dir = os.path.dirname(checkpoint_path)

ckpt = tf.train.Checkpoint(encoder_optimizer=encoder_optimizer,
                           generator_optimizer=generator_optimizer,
                           discriminator_optimizer=discriminator_optimizer,
                           encoder=encoder,
                           generator=generator,
                           discriminator=discriminator)

manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=1)

def save_checkpoints(ckpt, model_directory):
    if not os.path.exists(model_directory+"checkpoints/"):
        os.makedirs(model_directory+"checkpoints/")
    manager.save()

if RESTART_TRAINING:
    total_iterations = RESTART_ITERATION
    RESTART_EPOCH = training_samples // BATCH_SIZE
    if manager.latest_checkpoint:
        ckpt.restore(manager.latest_checkpoint)
        print("\nRestored from {} \n".format(manager.latest_checkpoint))
    else:
        print("No Checkpoint found --> Initializing from scratch.")

#####
##### TRAINING #####
#####

print("-----")
print("RUN TRAINING:\n")
#####
start_time = time.time()
iter_arr, val_iter_arr = [], []
G_losses = []
MSE_losses = []
JACCARD_losses = []
D_losses = []
D_val_loss, MSE_val_loss, JACCARD_val_loss = [], [], []
n_objects = 0
total_iterations = 0
best_iteration = VAL_SAVE_FREQ
best_loss = 1e10
for epoch in range(1, N_EPOCHS+1):
    #####

```

```

1726 # START OF EPOCH
1727 t_epoch_start = time.time()
1728 n_batch=0
1729 ######
1730 for obj_batch in train_objects:
1731     # START OF BATCH
1732     batch_time = time.time()
1733     G_loss, D_loss = train_step(obj_batch)
1734     #Append arrays
1735     G_losses.append(G_loss[0])
1736     MSE_losses.append(G_loss[1])
1737     D_losses.append(-D_loss[0])
1738
1739     n_batch+=1
1740     total_iterations+=1
1741     iter_arr.append(total_iterations)
1742
1743     JACCARD_loss = JACCARD_reconstruction_loss(obj_batch)
1744     JACCARD_losses.append(JACCARD_loss)
1745     print("E: %3d/%5d | B: %3d/%3d | I: %5d | T: %5.1f | dt: %2.2f || LOSSES: D
1746 %.4f | G %.4f | MSE %.4f | JACCARD %.4f" \
1747         % (epoch, N_EPOCHS, n_batch, np.ceil(training_set_size*(T//TINC-1)//
1748 BATCH_SIZE), total_iterations, time.time() - start_time, time.time() -
1749 batch_time, -D_loss[0], G_loss[0], G_loss[1], JACCARD_loss))
1750
1751     if PRINT_D_LOSSES:
1752         print("\nfake_loss: ", D_loss[1].numpy(), "real_loss: ", D_loss[2].numpy()
1753             (), "gp: ", D_loss[3].numpy(), "ep: ", D_loss[4].numpy(), "\n\n")
1754
1755     ######
1756     # AFTER N TOTAL ITERATIONS GET VALIDATION LOSSES, SAVE GENERATED OBJECTS AND
1757     # MODEL:
1758     if (total_iterations % VAL_SAVE_FREQ == 0) and VAL_SAVE_FREQ != -1:
1759         print("\nRUN VALIDATION, SAVE OBJECTS, MODEL AND PLOT LOSSES... ")
1760
1761     Val_losses = []
1762     Val_MSE_losses = []
1763     Val_JACCARD_losses = []
1764     for val_obj_batch in val_objects:
1765         Val_loss, MSE_recon = validation_step(val_obj_batch)
1766         Val_losses.append(-Val_loss[0]) # Collect Val loss per Batch
1767         Val_MSE_losses.append(MSE_recon) # Collect Val loss per Batch
1768         JACCARD_recon = JACCARD_reconstruction_loss(val_obj_batch)
1769         Val_JACCARD_losses.append(JACCARD_recon) # Collect Val loss per
1770
1771     Batch
1772
1773     D_val_loss.append(np.mean(Val_losses)) # Mean over batches
1774     MSE_val_loss.append(np.mean(Val_MSE_losses)) # Mean over batches
1775     JACCARD_val_loss.append(np.mean(Val_JACCARD_losses)) # Mean over batches
1776     val_iter_arr.append(total_iterations)

```

```

1770 #####
1771     # Output generated objects
1772     generate_sample_objects(test_objects, total_iterations, "train")
1773     generate_sample_objects(valid_objects, total_iterations, "valid")
1774     # # Plot and Save Loss Curves
1775     render_graphs(model_directory, G_losses, D_losses, D_val_loss,
1776     MSE_losses, MSE_val_loss, JACCARD_losses, JACCARD_val_loss, iter_arr,
1777     val_iter_arr, RESTART_TRAINING) #this will only work after a 50 iterations to
1778     allow for proper averaging
1779
1780
1781     if KEEP_BEST_ONLY:
1782         if BEST_METRIC == "JACCARD":
1783             BEST_LOSS_METRIC = JACCARD_val_loss
1784             KEEP_LAST_N = 1
1785         elif BEST_METRIC == "MSE":
1786             BEST_LOSS_METRIC = MSE_val_loss
1787             KEEP_LAST_N = 1
1788         if len(BEST_LOSS_METRIC) > 1 and (BEST_LOSS_METRIC[-1] <= best_loss)
1789         :
1790             best_loss = BEST_LOSS_METRIC[-1]
1791             best_iteration = total_iterations
1792             # Save models
1793             encoder.save(model_directory+"_trained_encoder_"+str(
1794             total_iterations)+".h5")
1795             generator.save(model_directory+"_trained_generator_"+str(
1796             total_iterations)+".h5")
1797         else:
1798             # Save models
1799             encoder.save(model_directory+"_trained_encoder_"+str(
1800             total_iterations)+".h5")
1801             generator.save(model_directory+"_trained_generator_"+str(
1802             total_iterations)+".h5")
1803
1804             # # Delete Model, keep best or last N models
1805             IOhousekeeping(model_directory, KEEP_LAST_N, VAL_SAVE_FREQ,
1806             RESTART_TRAINING, total_iterations, KEEP_BEST_ONLY, best_iteration)
1807
1808             #####
1809             # SAVE CHECKPOINT FOR RESTART:
1810             if total_iterations % WRITE_RESTART_FREQ == 0:
1811                 print("WRITE RESTART FILE ... \n")
1812                 # Save Checkpoint after every 100 epoch
1813                 save_checkpoints(ckpt, model_directory)
1814
1815             # END OF BATCH
1816             #####
1817             # STOP TRAINING AFTER MAX DEFINED ITERATIONS ARE REACHED
1818             if total_iterations == MAX_ITERATIONS :
1819                 break
1820             if total_iterations == MAX_ITERATIONS:

```

```

    break

1812
# Print Status at the end of the epoch
1814 dt_epoch = time.time() - t_epoch_start
n_objects = n_objects + np.ceil(training_set_size*(T//TINC-1)//BATCH_SIZE)*
BATCH_SIZE
1816 print("\n-----")
1817 print("END OF EPOCH ", epoch, " | Total Training Iterations: ", int(
total_iterations), " | Total Number of objects trained: ", int(n_objects/1000),
"K", " | time elapsed:", str(int((time.time() - start_time) / 60.0)), "min")
1818 print("-----\n\n")
1819 )

1820 # END OF EPOCH
1821 #####
1822
1823
1824 print("\n TRAINING DONE! \n")
print("Total Training Time: ", str((time.time() - start_time) / 60.0), "min" )

```

Listing B.1: Single Frame Input Training Algorithm

B.2 Single Frame Input Training Algorithm - Data Parser

```
1000 import os
os.environ[ 'TF_CPP_MIN_LOG_LEVEL' ] = '1'
1002 import tensorflow as tf
1003 import numpy as np
1004
AUTOTUNE = tf.data.AUTOTUNE
1006
1007 def decode_object(example):
1008     object4d = tf.io.decode_raw(example[ 'object4d' ], tf.uint8)
1009     labelBetti = tf.io.decode_raw(example[ 'labelBetti' ], tf.uint8)
1010     labelObjs = tf.io.decode_raw(example[ 'labelObjs' ], tf.uint8)
1011
1012     object4d = tf.reshape(object4d, [128, 16, 16, 16])
1013     labelBetti = tf.reshape(labelBetti, [4])
1014     labelObjs = tf.reshape(labelObjs, [4])
1015
1016     return object4d, labelBetti, labelObjs
1017
1018 def read_tfrecord(example):
1019     global NORMALIZE
1020     global T
1021     global TINC
1022
1023     object_feature_description = {
1024         'object4d': tf.io.FixedLenFeature([], tf.string),
1025         'labelBetti': tf.io.FixedLenFeature([], tf.string),
1026         'labelObjs': tf.io.FixedLenFeature([], tf.string),
1027     }
1028
1029     example = tf.io.parse_single_example(example, object_feature_description)
1030
1031     object4d, labelBetti, labelObjs = decode_object(example)
1032
1033     CROP=(128-T)//2
1034     # Reshape into time slices of n time steps (axis=1)
1035     object4d = object4d[CROP:128-CROP:TINC]
1036     object4d = tf.reshape(object4d, [T//TINC, 16, 16, 16])
1037     rearranged_objs4d = []
1038     for t in range(object4d.shape[0]):
1039         if t == object4d.shape[0] - 1:
1040             break
1041         rearranged_objs4d.append(object4d[t])
1042         rearranged_objs4d.append(object4d[t+1])
1043
1044     object4d = tf.reshape(tf.stack(rearranged_objs4d), [T//TINC-1, 2, 16, 16, 16])
1045
1046
```

```

1048     if NORMALIZE:
1049         object4d = tf.cast(object4d, tf.float32) - 0.5
1050
1051     return object4d#, tf.ones([object4d.shape[0],4], tf.uint8)*labelObjs
1052     #return object4d, tf.ones(object4d.shape[0], tf.uint8)*labelBetti[-1]#, tf.ones
1053     #([object4d.shape[0],4], tf.uint8)*labelObjs
1054
1055 def load_dataset(filenames):
1056     ignore_order = tf.data.Options()
1057     ignore_order.deterministic = False # disable order, increase speed
1058     dataset = tf.data.TFRecordDataset(filenames, compression_type="GZIP",
1059     num_parallel_reads=AUTOTUNE) # automatically interleaves reads from multiple
1060     files
1061     dataset = dataset.with_options(ignore_order) # uses data as soon as it streams
1062     in, rather than in its original order
1063     dataset = dataset.map(read_tfrecord, num_parallel_calls=AUTOTUNE)
1064     return dataset
1065
1066 def get_dataset(filenames: list, shuffle: bool, batch_size: int, t: int, tinc: int,
1067 normalize: bool):
1068     global NORMALIZE
1069     global T
1070     global TINC
1071     NORMALIZE = normalize
1072     TINC = tinc
1073     T = t
1074     dataset = load_dataset(filenames)
1075     dataset = dataset.unbatch()
1076     if shuffle:
1077         dataset = dataset.shuffle(len(filenames)*(T//TINC-1)//2) # ALL: len(
1078         filenames)*(T//TINC)
1079     if batch_size is not None:
1080         dataset = dataset.batch(batch_size, drop_remainder=True)
1081     dataset = dataset.prefetch(buffer_size=AUTOTUNE)
1082     #dataset = dataset.cache()
1083     return dataset
1084
1085
1086 # FOR TESTING:
1087 ''
1088 folder_name="Z:/ Master_Thesis/data/4D_128_32_topological_dataset_0-31_16_label8only"
1089
1090 FILENAMES = tf.io.gfile.glob(folder_name+"/*.tfrecords")[:7]
1091 split_ind = int(1 * len(FILENAMES))
1092 TRAINING_FILENOAMES, VALID_FILENOAMES = FILENAMES[:split_ind], FILENAMES[split_ind:]
1093
1094 #TEST_FILENOAMES = tf.io.gfile.glob(GCS_PATH + "/tfrecords/test*.tfrec")
1095 print("Train TFRecord Files:", len(TRAINING_FILENOAMES))
1096 print("Validation TFRecord Files:", len(VALID_FILENOAMES))
1097 #print("Test TFRecord Files:", len(TEST_FILENOAMES))

```

```

1092 BATCH_SIZE = 128
1094 train_dataset = get_dataset(TRAINING_FILERAMES, shuffle=True, batch_size=BATCH_SIZE,
1095     t=80, tinc=4, normalize=False)
1096 #print(len(list(train_dataset)) * BATCH_SIZE)
1097 #object4d, labelBetti = next(iter(train_dataset))
1098 # print(object4d.shape, labelBetti.shape)
1099 # print(object4d[0], labelBetti)
1100 #print(tf.data.experimental.cardinality(train_dataset))
1102 def jaccard(x,y):
1103     x = np.asarray(x, bool) # Not necessary, if you keep your data
1104     y = np.asarray(y, bool) # in a boolean array already!
1106 #j = tf.math.reduce_sum(tf.square(differences), axis=[1,2,3,4])
1107 j_and = tf.math.reduce_sum(np.double(np.bitwise_and(x, y)), axis=[1,2,3])
1108 j_or = tf.math.reduce_sum(np.double(np.bitwise_or(x, y)), axis=[1,2,3])
1110 return tf.reduce_mean(1 - j_and / j_or)
1112 import dataIO
1113 for object4d, labelBetti in train_dataset.take(1):
1114     print(object4d.shape, labelBetti.shape)
1115     jaccard_ = jaccard(object4d[:,0], object4d[:,1])
1116
1117     i=0
1118     for object4d_i, labelBetti_i in zip(object4d, labelBetti):
1119         print(object4d_i.shape, labelBetti_i)
1120         #print(object4d_i.shape, labelBetti_i)
1121
1122         # for object4d_ii in object4d_i:
1123         #     dataIO.plotFromVoxels(object4d_ii.numpy(), [i, labelBetti_i])
1124         #     i+=1
1125     ,

```

Listing B.2: Single Frame Input Training Algorithm - Data Parser

B.3 Single Frame Input Prediction Algorithm

```
1000 import os
1001 import tensorflow as tf
1002 from keras.models import load_model
1003 import numpy as np
1004 # OWN LIBRARIES
1005 from utils import parse_tfrecords_4D_16_for_training as p4d
1006 from utils import dataIO as d

1008 base_dir = r"Z:\Master_Thesis\code\99_FINAL_MODELS_FOR_THESIS"
1009
1010 MODEL_LOCATION      = os.path.join(base_dir, "aencgan3d_16_T20_TINC1")
1011
1012 TEST_DATASET_LOC    = "Z:/Master_Thesis/data/02_4D_test_datasets"
1013
1014 T                  = int(MODEL_LOCATION[-8:-6])
1015 TINC                = int(MODEL_LOCATION[-1])
1016 N_PREDICT          = 5
1017
1018 if not os.path.exists(os.path.join(base_dir, f'objects/T{T}_TINC{TINC}')):
1019     os.makedirs(os.path.join(base_dir, f'objects/T{T}_TINC{TINC}'))
1020 #####
1021 ##### GET Data/Objects #####
1022 #####
1023 TEST_FILERAMES = tf.io.gfile.glob(os.path.join(TEST_DATASET_LOC, "*tfrecords"))
1024 test_set_size = len(TEST_FILERAMES)
1025
1026 print()
1027 print("Total Number of TFRecord Files to Test : ", test_set_size)
1028 print()
1029
1030 # Convert/Decode dataset from tfrecords file for training
1031 test_objects = p4d.get_dataset(TEST_FILERAMES, shuffle=True, batch_size=1, t=T, tinc=TINC, normalize=False)
1032
1033 encoder_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*encoder*.h5"))
1034 encoder = load_model(encoder_model[-1])
1035
1036 generator_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*generator*.h5"))
1037 generator = load_model(generator_model[-1])
1038
1039 def predictions(objects):
1040     n=0
1041     for X in objects:
1042
1043         d.plotFromVoxels(np.squeeze(X[0,0]), f"{n}_T{T}_TINC{TINC}_t0")
```

```

1046     d.plotMeshFromVoxels(np.squeeze(X[0,0]), obj=f"objects/T{T}_TINC{TINC}/
single_{n}_T{T}_TINC{TINC}_t0")
1047     d.plotFromVoxels(np.squeeze(X[0,1]), f"{n}_T{T}_TINC{TINC}_t1")
1048     d.plotMeshFromVoxels(np.squeeze(X[0,1]), obj=f"objects/T{T}_TINC{TINC}/
single_{n}_T{T}_TINC{TINC}_t1")
1049
1050     # Encode & Generate
1051     X0 = tf.dtypes.cast(X[:,0], tf.float32)
1052     latent_code = encoder(X0, training=False)
1053     pred_object = np.squeeze(generator(latent_code, training=False))
1054     pred_object = np.asarray(pred_object>0.5, bool)
1055
1056     d.plotFromVoxels(pred_object, f"{n}_T{T}_TINC{TINC}_t1_pred")
1057     d.plotMeshFromVoxels(pred_object, obj=f"objects/T{T}_TINC{TINC}/single_{n}_T
{T}_TINC{TINC}_t1_pred")
1058     if n==N_PREDICT:
1059         break
1060     n+=1
1061
1062 predictions(test_objects)

```

Listing B.3: Single Frame Input Prediction Algorithm

B.4 Single Frame Input Long-Term Prediction Algorithm

```
1000 import os
1001 import tensorflow as tf
1002 from keras.models import load_model
1003 import numpy as np
1004 # OWN LIBRARIES
1005 from utils import parse_tfrecords_4D_16_for_LongTerm_prediction as p4d
1006 from utils import dataIO as d

1008 base_dir = r"Z:\Master_Thesis\code\99_FINAL_MODELS_FOR_THESIS"
1009
1010 MODEL_LOCATION      = os.path.join(base_dir, "aencgan3d_16_T20_TINC1")
1011
1012 TEST_DATASET_LOC    = "Z:/Master_Thesis/data/02_4D_test_datasets"
1013
1014 T                  = int(MODEL_LOCATION[-8:-6])
1015 TINC                = int(MODEL_LOCATION[-1])
1016 N_PREDICT          = 5
1017 MAX_TIME            = 5

1018 if not os.path.exists(os.path.join(base_dir, f'objects_longterm/T{T}_TINC{TINC}')):
1019     os.makedirs(os.path.join(base_dir, f'objects_longterm/T{T}_TINC{TINC}'))
1020 #####
1021 ##### GET Data / Objects #####
1022 #####
1023 #####
1024 TEST_Filenames = tf.io.gfile.glob(os.path.join(TEST_DATASET_LOC, "*tfrecords"))
1025 test_set_size = len(TEST_Filenames)

1026 print()
1027 print("Total Number of TFRecord Files to Test : ", test_set_size)
1028 print()

1029 # Convert/Decode dataset from tfrecords file for training
1030 test_objects = p4d.get_dataset(TEST_Filenames, shuffle=False, batch_size=None, t=T,
1031                               tinc=TINC, normalize=False)
1032
1033 encoder_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*encoder*.h5"))
1034 encoder = load_model(encoder_model[-1])
1035
1036 generator_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*generator*.h5"))
1037 generator = load_model(generator_model[-1])

1038 def predictions(objects):
1039     n=0
1040     for X in objects:
1041         d.plotFromVoxels(np.squeeze(X[0]), f"{n}_T{T}_TINC{TINC}_t0")
1042
1043
1044
```

```

d.plotMeshFromVoxels(np.squeeze(X[0]), obj=f"objects_longterm/T{T}_TINC{TINC}
1046 }/single_{n}_T{T}_TINC{TINC}_t0")
d.plotFromVoxels(np.squeeze(X[1]), f"{n}_T{T}_TINC{TINC}_t1")
d.plotMeshFromVoxels(np.squeeze(X[1]), obj=f"objects_longterm/T{T}_TINC{TINC}
1048 }/single_{n}_T{T}_TINC{TINC}_t1")
d.plotFromVoxels(np.squeeze(X[2]), f"{n}_T{T}_TINC{TINC}_t2")
d.plotMeshFromVoxels(np.squeeze(X[2]), obj=f"objects_longterm/T{T}_TINC{TINC}
1049 }/single_{n}_T{T}_TINC{TINC}_t2")
d.plotFromVoxels(np.squeeze(X[3]), f"{n}_T{T}_TINC{TINC}_t3")
d.plotMeshFromVoxels(np.squeeze(X[3]), obj=f"objects_longterm/T{T}_TINC{TINC}
1050 }/single_{n}_T{T}_TINC{TINC}_t3")
d.plotFromVoxels(np.squeeze(X[4]), f"{n}_T{T}_TINC{TINC}_t4")
d.plotMeshFromVoxels(np.squeeze(X[4]), obj=f"objects_longterm/T{T}_TINC{TINC}
1052 }/single_{n}_T{T}_TINC{TINC}_t4")
d.plotFromVoxels(np.squeeze(X[5]), f"{n}_T{T}_TINC{TINC}_t5")
d.plotMeshFromVoxels(np.squeeze(X[5]), obj=f"objects_longterm/T{T}_TINC{TINC}
1054 }/single_{n}_T{T}_TINC{TINC}_t5")
#Loop over time per sample
X = tf.dtypes.cast(X, tf.float32)
for t in range(X.shape[1]-1):
    if t==0:
        X0 = X[0][np.newaxis ,...]
1060
1062
# Encode & Generate
X0 = tf.dtypes.cast(X0, tf.float32)
latent_code = encoder(X0, training=False)
pred_object = np.squeeze(generator(latent_code, training=False))
pred_object = np.asarray(pred_object>0.5, bool)
1068
d.plotFromVoxels(pred_object, f"{n}_T{T}_TINC{TINC}_t{t+1}_pred")
d.plotMeshFromVoxels(pred_object, obj=f"objects_longterm/T{T}_TINC{TINC}
1070 }/single_{n}_T{T}_TINC{TINC}_t{t+1}_pred")
1072
if t==MAX_TIME-1:
    break
1074
X0 = pred_object[np.newaxis ,...]
1076
1078
n+=1
if n==N_PREDICT:
    break
1080
1082
1084 predictions(test_objects)

```

Listing B.4: Single Frame Input Prediction Algorithm

B.5 Multiple Frame Input Training Algorithm

```
1000 import os
1001 from shutil import copyfile
1002 import glob
1003 import sys
1004 import numpy as np
1005 #np.set_printoptions(threshold=sys.maxsize)
1006 import time
1007 import tensorflow as tf
1008 from tensorflow.keras import Model
1009 from tensorflow.keras.layers import Dense, BatchNormalization, LayerNormalization,
1010     LeakyReLU, Reshape, Conv3DTranspose, Conv3D, Flatten, ReLU, Dropout, Concatenate
1011     , Input, UpSampling3D, AveragePooling3D
1012 from tensorflow.keras.activations import sigmoid, tanh
1013 from tensorflow.keras.optimizers import Adam
1014 import tensorflow_addons as tfa

1015 # OWN LIBRARIES
1016 from utils.loss_plotter import render_graphs
1017 from utils import dataIO as d
1018 from utils import parse_tfrecords_4D_16_t3_for_training as p4d
1019 from utils.logger import Logger

1020 ##### INPUTS #####
1021 #####
1022 #####
1023 N_EPOCHS = 10000          # Default: 10000 – Max. training epochs OR
1024 Define Max. Iterations
1025 MAX_ITERATIONS = 20000      # Default: -1 – Max. Training Iterations (
1026     Max. Number of processed Batches) before training stops, if "-1" Training stops
1027     after reaching N_EPOCHS
1028 TRAIN_DATASET_LOC = "Z:/Master_Thesis/data/00_4D_training_datasets"
1029 VALID_DATASET_LOC = "Z:/Master_Thesis/data/01_4D_validation_datasets"
1030 N_TRAINING = 10000          # Number of Files for Training: -1 == USE
1031     ALL
1032 N_VALIDATION = 1000          # Number of Files for Validation: -1 == USE
1033     ALL
1034 T = 20                      # Default: 80 – Even Number of Length of
1035     Time History to use, MUST BE EVENLY DIVISIBLE BY TINC, e.g. 80 means that 80
1036     steps out of all 128 steps are used, where 24 steps at the beginning and 24
1037     steps at the end of the time history are cropped.
1038 TINC = 1                      # Default: 1 – Allowed: (1, 2, 4, 8) – e.g.:
1039     4 --> 80/4 = 20 Timepoints --> 19 PAIRS of T0->T1 --> TIME INCREMENT FOR
1040     EXTRACTING TIMESTEPS PER DATAPOINT – (Check parse_tfrecords_4D_##_for_training
1041     for extraction details)
1042 NORMALIZE = False            # NORMALIZE DATA to ZERO MEAN --> from [0;1]
1043     to [-0.5, 0.5]
```

```

1032 PRINT_D_LOSSES      = False          # LOG DISCRIMINATOR LOSSES SPLIT UP IN Fake,
    Real , GP and EPS Losses
LOGGING           = True           # LOG TRAINING RUN TO FILE (If Kernel get
    Interupted (user or error) Kernel needs to be restarted , otherwise log file is
    locked to be deleted!)
#####
1034 ##### RESTART #####
# Restart Training?
1036 RESTART_TRAINING   = False
RESTART_ITERATION   = 50000          # the epoch number when the last training
    was stopped, has no influnce on loading the restart file , just used for tracking
    information numbering
1038 RESTART_FOLDER_NAME = "20240221_012216_aencgan3d_16_t3"
#####
1040 # Output Settings
VAL_SAVE_FREQ       = 1000          # Default: 1000 – OFF: -1, every n training
    iterations (n batches) check validation loss
1042 WRITE_RESTART_FREQ = 1000          # Default: 1000 – every n training
    iterations (n batches) save restart checkpoint
KEEP_BEST_ONLY      = True           # Default: True – Keep only the best model (
    KEEP_LAST_N=1)
1044 BEST_METRIC        = "JACCARD"      # Default: JACCARD – "JACCARD" or "MSE"
# if KEEP_BEST_ONLY = False define max. Number of last models to keep:
1046 KEEP_LAST_N        = 5             # Default: 5 – Max. number of last files to
    keep
N_SAMPLES           = 5              # Default: 5 – Number of object samples to
    generate after each epoch
#####
1048 ##### HYPERPARAMETERS #####
#####
1050 ##### ARCHITECTURE #####
GD_ARCHITECTURE     = "PreRes"        # Default: "PreRes" – "PreRes" or "Classic"
    Convolutions
1052 BATCH_SIZE         = 32            # Default: 32 – Number of Examples per Batch
    / Probably Increase for better Generalization
Z_SIZE               = 256            # Default: 256 – Latent Random Vector Size
1054 FEATUREMAP_BASE_DIM = 256          # Default: 256 – Max Featuremap size (Maybe
    increase for learning larger/more diverse datasets...)
#####
1056 # WGAN-GP Settings
GP_WEIGHT             = 100            # Default: 10 – WGAN-GP gradient penalty
    weight
1058 EPS_WEIGHT         = 0.0            # Default: 0.001 – WGAN-GP epsilon penalty
    weight (real logits penalty)
D_STEPS               = 1              # Default: 5 – WGAN-GP Disc. training steps
#####
1060 ##### OPTIMIZER #####
# WGAN-GP Optimizer Settings
1062 G_LR                = 0.0001        # Default: 0.0001 – Generator Learning Rate
D_LR                 = 0.0001        # Default: 0.0001 – Discriminator Learning
    Rate
1064 CLR                  = False          # Default: False – Use Cyclic Learning Rate ,
    minLR=G_LR or D_LR, maxLR=10*G_LR or 10*D_LR

```

```

#####
# Dropout
1066 DROPOUT_RATE      = 0.2                      # Default: 0.2
#####
# weight_initializer & regularizer
1068 weight_initializer = "glorot_uniform" # "glorot_uniform" "glorot_normal" or "
1070     he_uniform"
1071 weight_regularizer = "L1"                   # tf.keras.regularizers.L1(l1=0.01) #
1072     Default l1=0.01
#####
# USE BIAS IN GENERATOR
1074 USE_BIAS           = True
1075 bias_initializer   = "zeros" # DEFAULT: "zeros" - "ones" or tf.keras.initializers.
1076     Constant(0.5)
#####
# DEFAULT - DONT TOUCH SETTINGS
1078 SIZE                = 16                     # Default: 16 - Object Grid Size
1079 BASE_SIZE            = 4                      # Default: 4 - Base Resolution to start from
1080     for generator and to end with for discriminator
1081 KERNEL_SIZE          = 5                     # Default: 5 - Lower values learn pretty bad
1082     , higher than 5 doesnt change training much
#####
# NETWORKS #####
1084 # latent vector input
1085 latent_vector = Input(shape=(Z_SIZE*3,))
1086 #
#####
# ENCODER #####
1088 #
1089 def obj_input(object_dim=(SIZE, SIZE, SIZE, 1)):
1090     obj_in = Input(shape=object_dim)
1091     return obj_in
1092 #
#####
# Encoder PreResidual Block
1094 def EncoderPreResBlock(x, in_filters, out_filters, kernel_size, strides,
1095     kernel_initializer, kernel_regularizer, padding):
1096     # Define Shortcut
1097     if strides == 1 and in_filters == out_filters: # Identity Skip Connection
1098         shortcut = x
1099     elif strides == 2 and in_filters == out_filters: # Identity Skip Connection
1100         shortcut = AveragePooling3D(pool_size=(strides, strides, strides), strides=(strides, strides, strides))(x)
1101     elif strides == 1 and in_filters != out_filters: # Change of Number of Filters
1102         # Skip
1103         shortcut = Conv3D(filters=out_filters, kernel_size=1, strides=1,
1104             kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
1105             padding="same")(x)
1106     elif strides == 2 and in_filters != out_filters: # Skip with Change in Size and
1107         # Number of Filters

```

```

1104     shortcut = AveragePooling3D(pool_size=(strides, strides, strides), strides=(strides, strides, strides))(x)
1105     shortcut = Conv3D(filters=out_filters, kernel_size=1, strides=1,
1106                         kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
1107                         padding="same")(shortcut)
1108
1109 # Define Activation on Input
1110 x = BatchNormalization()(x)
1111 x = LeakyReLU()(x)
1112 x = Dropout(DROPOUT_RATE)(x)
1113 x = Conv3D(filters=out_filters, kernel_size=kernel_size, strides=strides,
1114                 kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
1115                 padding=padding)(x)
1116
1117     return shortcut + x
1118 #####
1119 # DEFINE ENCODER NETWORK
1120 def ClassicEncoder():
1121     obj_in = obj_input()
1122
1123     x = Conv3D(filters=FEATUREMAP_BASE_DIM//4, kernel_size=KERNEL_SIZE, strides=2,
1124                 kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
1125                 padding="same")(obj_in)
1126     x = BatchNormalization()(x)
1127     x = LeakyReLU()(x)
1128     x = Dropout(DROPOUT_RATE)(x)
1129
1130     x = Conv3D(filters=FEATUREMAP_BASE_DIM//2, kernel_size=KERNEL_SIZE, strides=2,
1131                 kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
1132                 padding="same")(x)
1133     x = BatchNormalization()(x)
1134     x = LeakyReLU()(x)
1135     x = Dropout(DROPOUT_RATE)(x)
1136
1137     x = Conv3D(filters=FEATUREMAP_BASE_DIM, kernel_size=KERNEL_SIZE, strides=1,
1138                 kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
1139                 padding="same")(x)
1140
1141     x = Flatten()(x)
1142
1143     out_layer = Dense(Z_SIZE, activation=None)(x)
1144
1145 # define model
1146 model = Model(obj_in, out_layer)
1147     return model
1148
1149 def PreResEncoder():
1150     obj_in = obj_input()
1151
1152 # Begin Activation

```

```

x = Conv3D(filters=FEATUREMAP_BASE_DIM//8, kernel_size=KERNEL_SIZE, strides=1,
           input_shape=[SIZE, SIZE, SIZE, 1], kernel_initializer=weight_initializer,
           kernel_regularizer=weight_regularizer, padding="same")(obj_in)

1144
# PreRes Blocks
1146 x = EncoderPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//8, out_filters=
FEATUREMAP_BASE_DIM//4, kernel_size=KERNEL_SIZE, strides=2, kernel_initializer=
weight_initializer, kernel_regularizer=weight_regularizer, padding="same")
x = EncoderPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//4, out_filters=
FEATUREMAP_BASE_DIM//2, kernel_size=KERNEL_SIZE, strides=2, kernel_initializer=
weight_initializer, kernel_regularizer=weight_regularizer, padding="same")
1148 x = EncoderPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//2, out_filters=
FEATUREMAP_BASE_DIM , kernel_size=KERNEL_SIZE, strides=1, kernel_initializer=
weight_initializer, kernel_regularizer=weight_regularizer, padding="same")

1150 x = BatchNormalization()(x)
x = LeakyReLU()(x)
1152 x = Dropout(DROPOUT_RATE)(x)

1154 # Flatten (reshape) and Linear (dense) Layer
x = Flatten()(x)

1156 out_layer = Dense(Z_SIZE, activation=None)(x)
1158
# define model
1160 model = Model(obj_in, out_layer)
return model

1162 #####
1164 #
1166 ##### GENERATOR #####
1168
def latent_input(latent_vector=latent_vector, BASE_SIZE=BASE_SIZE):
1170     base = BASE_SIZE * BASE_SIZE * BASE_SIZE

1172     latent_dense = Dense(units = FEATUREMAP_BASE_DIM*base)(latent_vector)
     latent_reshape = Reshape((BASE_SIZE, BASE_SIZE, BASE_SIZE, FEATUREMAP_BASE_DIM))
     (latent_dense)
1174     return latent_reshape

1176 #####
1178 # Generator PreResidual Block
def GPreResBlock(x, in_filters , out_filters , kernel_size , up_size ,
kernel_initializer , kernel_regularizer , padding , use_bias , bias_initializer):
# Define Shortcut
1180     if up_size == 1 and in_filters == out_filters: # Identity Skip Connection
         shortcut = x
     elif up_size == 2 and in_filters == out_filters: # Identity Skip Connection

```

```

    shortcut = UpSampling3D(size=(up_size, up_size, up_size))(x)
1184  elif up_size == 1 and in_filters != out_filters: # Change of Number of Filters
      Skip
        shortcut = Conv3D(filters=out_filters, kernel_size=1, strides=1,
kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
padding=padding, use_bias=use_bias, bias_initializer = bias_initializer)(x)
1186  elif up_size == 2 and in_filters != out_filters: # Skip with Change in Size and
      Number of Filters
        shortcut = UpSampling3D(size=(up_size, up_size, up_size))(x)
1188  shortcut = Conv3D(filters=out_filters, kernel_size=1, strides=1,
kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
padding=padding, use_bias=use_bias, bias_initializer = bias_initializer)(
shortcut)

1190  # Define Activation on Input
1191  x = BatchNormalization()(x)
1192  x = ReLU()(x)
1193  if up_size != 1:
1194    x = UpSampling3D(size=(up_size, up_size, up_size))(x)
1195  x = Conv3DTranspose(filters=out_filters, kernel_size=kernel_size, strides=1,
kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
padding=padding, use_bias=use_bias, bias_initializer = bias_initializer)(x)
1196
1197  return shortcut + x
1198 #####
# DEFINE GENERATOR NETWORK
1199 def ClassicGenerator():
1200   latent_in = latent_input()
1201   x = BatchNormalization()(latent_in)
1202   x = ReLU()(x)
1203
1204   x = Conv3DTranspose(filters=FEATUREMAP_BASE_DIM, kernel_size=KERNEL_SIZE,
strides=1, kernel_initializer=weight_initializer, kernel_regularizer=
weight_regularizer, padding="same", use_bias=USE_BIAS, bias_initializer =
bias_initializer)(x)
1205   x = BatchNormalization()(x)
1206   x = ReLU()(x)
1207
1208   x = UpSampling3D(size=(2, 2, 2))(x)
1209   x = Conv3DTranspose(filters=FEATUREMAP_BASE_DIM//2, kernel_size=KERNEL_SIZE,
strides=1, kernel_initializer=weight_initializer, kernel_regularizer=
weight_regularizer, padding="same", use_bias=USE_BIAS, bias_initializer =
bias_initializer)(x)
1210   x = BatchNormalization()(x)
1211   x = ReLU()(x)
1212
1213   x = UpSampling3D(size=(2, 2, 2))(x)
1214   x = Conv3DTranspose(filters=FEATUREMAP_BASE_DIM//4, kernel_size=KERNEL_SIZE,
strides=1, kernel_initializer=weight_initializer, kernel_regularizer=
weight_regularizer, padding="same", use_bias=USE_BIAS, bias_initializer =

```

```

1216     bias_initializer)(x)
1217     x = BatchNormalization()(x)
1218     x = ReLU()(x)
1219
1220
1221     x = Conv3DTranspose(filters=1, kernel_size=KERNEL_SIZE, strides=1,
1222     kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
1223     padding="same", use_bias=USE_BIAS, bias_initializer = bias_initializer)(x)
1224
1225     if NORMALIZE:
1226         out_layer = tanh(x)
1227     else:
1228         out_layer = sigmoid(x)
1229
1230     model = Model(latent_vector, out_layer)
1231
1232     return model
1233
1234 def PreResGenerator():
1235     x = latent_input()
1236
1237     # PreRes Blocks
1238     x = GPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM      , out_filters=
1239     FEATUREMAP_BASE_DIM//2, kernel_size=KERNEL_SIZE, up_size=1, kernel_initializer=
1240     weight_initializer, kernel_regularizer=weight_regularizer, padding="same",
1241     use_bias=USE_BIAS, bias_initializer = bias_initializer)
1242     x = GPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//2, out_filters=
1243     FEATUREMAP_BASE_DIM//4, kernel_size=KERNEL_SIZE, up_size=2, kernel_initializer=
1244     weight_initializer, kernel_regularizer=weight_regularizer, padding="same",
1245     use_bias=USE_BIAS, bias_initializer = bias_initializer)
1246     x = GPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//4, out_filters=
1247     FEATUREMAP_BASE_DIM//8, kernel_size=KERNEL_SIZE, up_size=2, kernel_initializer=
1248     weight_initializer, kernel_regularizer=weight_regularizer, padding="same",
1249     use_bias=USE_BIAS, bias_initializer = bias_initializer)
1250
1251     x = BatchNormalization()(x)
1252     x = ReLU()(x)
1253
1254     # Final Convolution
1255     x = Conv3DTranspose(filters=1, kernel_size=KERNEL_SIZE, strides=1,
1256     kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
1257     padding="same", use_bias=USE_BIAS, bias_initializer = bias_initializer)(x)
1258
1259     # End Activation
1260     if NORMALIZE:
1261         out_layer = tanh(x)
1262     else:
1263         out_layer = sigmoid(x)
1264
1265     model = Model(latent_vector, out_layer)
1266
1267     return model

```

```

1252 #####
1254 #-----
1256 ##### DISCRIMINATOR #####
1258 #-----
1259
1260 def real_obj_input_discriminator(object_dim=(SIZE, SIZE, SIZE, 1)):
1261     real_obj_in = Input(shape=object_dim)
1262     return real_obj_in
1263 def cond_obj_input_discriminator(object_dim=(SIZE, SIZE, SIZE, 1)):
1264     cond_obj_in = Input(shape=object_dim)
1265     return cond_obj_in
1266 #####
1267 # Discriminator PreResidual Block
1268 def DPreResBlock(x, in_filters, out_filters, kernel_size, strides,
1269   kernel_initializer, kernel_regularizer, padding):
1270     # Define Shortcut
1271     if strides == 1 and in_filters == out_filters: # Identity Skip Connection
1272         shortcut = x
1273     elif strides == 2 and in_filters == out_filters: # Identity Skip Connection
1274         shortcut = AveragePooling3D(pool_size=(strides, strides, strides), strides=(strides, strides, strides))(x)
1275     elif strides == 1 and in_filters != out_filters: # Change of Number of Filters
1276       Skip
1277         shortcut = Conv3D(filters=out_filters, kernel_size=1, strides=1,
1278           kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
1279           padding="same")(x)
1280     elif strides == 2 and in_filters != out_filters: # Skip with Change in Size and
1281       Number of Filters
1282         shortcut = AveragePooling3D(pool_size=(strides, strides, strides), strides=(strides, strides, strides))(x)
1283         shortcut = Conv3D(filters=out_filters, kernel_size=1, strides=1,
1284           kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
1285           padding="same")(shortcut)
1286
1287     # Define Activation on Input
1288     x = LayerNormalization()(x)
1289     x = LeakyReLU()(x)
1290     x = Dropout(DROPOUT_RATE)(x)
1291     x = Conv3D(filters=out_filters, kernel_size=kernel_size, strides=strides,
1292       kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
1293       padding=padding)(x)
1294
1295     return shortcut + x
1296 #####
1297 # DEFINE DISCRIMINATOR NETWORK
1298 def ClassicDiscriminator():
1299     real_obj_in = real_obj_input_discriminator()

```

```

cond_obj_in = cond_obj_input_discriminator()

1292
# merge label_conditioned_generator and latent_input output
1294 x = Concatenate()([real_obj_in, cond_obj_in])

1296 x = Conv3D(filters=FEATUREMAP_BASE_DIM//4, kernel_size=KERNEL_SIZE, strides=2,
kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
padding="same")(x)
x = LayerNormalization()(x)
1298 x = LeakyReLU()(x)
x = Dropout(DROPOUT_RATE)(x)

1300
1302 x = Conv3D(filters=FEATUREMAP_BASE_DIM//2, kernel_size=KERNEL_SIZE, strides=2,
kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
padding="same")(x)
x = LayerNormalization()(x)
x = LeakyReLU()(x)
1304 x = Dropout(DROPOUT_RATE)(x)

1306 x = Conv3D(filters=FEATUREMAP_BASE_DIM, kernel_size=KERNEL_SIZE, strides=1,
kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
padding="same")(x)

1308 x = Flatten()(x)

1310 out_layer = Dense(1, activation=None)(x)

1312 # define model
model = Model([real_obj_in, cond_obj_in], out_layer)
1314 return model

1316 def PreResDiscriminator():
1317     real_obj_in = real_obj_input_discriminator()
1318     cond_obj_in = cond_obj_input_discriminator()

1320 # merge label_conditioned_discriminator and latent_input output
1321 x = Concatenate()([real_obj_in, cond_obj_in])

1322 # Begin Activation
1323 x = Conv3D(filters=FEATUREMAP_BASE_DIM//8, kernel_size=KERNEL_SIZE, strides=1,
input_shape=[SIZE, SIZE, SIZE, 1], kernel_initializer=weight_initializer,
kernel_regularizer=weight_regularizer, padding="same")(x)

1326 # PreRes Blocks
1327 x = DPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//8, out_filters=
FEATUREMAP_BASE_DIM//4, kernel_size=KERNEL_SIZE, strides=2, kernel_initializer=
weight_initializer, kernel_regularizer=weight_regularizer, padding="same")
x = DPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//4, out_filters=
FEATUREMAP_BASE_DIM//2, kernel_size=KERNEL_SIZE, strides=2, kernel_initializer=
weight_initializer, kernel_regularizer=weight_regularizer, padding="same")

```

```

x = DPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//2, out_filters=
FEATUREMAP_BASE_DIM , kernel_size=KERNEL_SIZE, strides=1, kernel_initializer=
weight_initializer , kernel_regularizer=weight_regularizer , padding="same")

1330 x = LayerNormalization()(x)
1332 x = LeakyReLU()(x)
1334 x = Dropout(DROPOUT_RATE)(x)

1336 # Flatten (reshape) and Linear (dense) Layer
1338 x = Flatten()(x)

1340 out_layer = Dense(1, activation=None)(x)

1342 # define model
model = Model([real_obj_in , cond_obj_in], out_layer)
return model

1344 #####
1345 ##### COMPILE NETWORKS #####
1346 #####
1347 if GD_ARCHITECTURE == "PreRes":
1348     encoder, generator, discriminator = PreResEncoder(), PreResGenerator(),
1349     PreResDiscriminator()
1350 elif GD_ARCHITECTURE == "Classic":
1351     encoder, generator, discriminator = ClassicEncoder(), ClassicGenerator(),
1352     ClassicDiscriminator()
1353 else:
1354     ValueError("Generator/Disriminator Architecture not Available. Choose between '"
1355     "PreRes' or 'Classic'!")
1356 encoder.compile()
1357 generator.compile()
1358 discriminator.compile()

1359 #####
1360 ##### LOSS FUNCTION #####
1361 #####
1362 @tf.function
1363 def encgen_loss(real_objects):
1364
1365     real_objects_t0 = tf.dtypes.cast(real_objects[:,0], tf.float32)
1366     real_objects_t1 = tf.dtypes.cast(real_objects[:,1], tf.float32)
1367     real_objects_t2 = tf.dtypes.cast(real_objects[:,2], tf.float32)
1368     real_objects_t3 = tf.dtypes.cast(real_objects[:,3], tf.float32)

1369
1370     # Encode
1371     latent_code_t0 = encoder(real_objects_t0)
1372     latent_code_t1 = encoder(real_objects_t1)
1373     latent_code_t2 = encoder(real_objects_t2)

```

```

latent_code = tf.concat([latent_code_t0, latent_code_t1, latent_code_t2], axis=1)

1374
1375 pred_objects_t3 = generator(latent_code, training=True)
1376 fake_logits = discriminator([pred_objects_t3, real_objects_t3], training=True)
1377 loss = wgan_gp_generator_loss(fake_logits)

1378 mse = tf.keras.metrics.mean_squared_error(real_objects_t3, pred_objects_t3)
1379 MSE_loss = tf.reduce_mean(mse)

1380
1381 return loss, MSE_loss

1382
1383 @tf.function
1384 def encdisc_loss(real_objects):
1385
1386     real_objects_t0 = tf.dtypes.cast(real_objects[:,0], tf.float32)
1387     real_objects_t1 = tf.dtypes.cast(real_objects[:,1], tf.float32)
1388     real_objects_t2 = tf.dtypes.cast(real_objects[:,2], tf.float32)
1389     real_objects_t3 = tf.dtypes.cast(real_objects[:,3], tf.float32)

1390
1391     # Encode
1392     latent_code_t0 = encoder(real_objects_t0)
1393     latent_code_t1 = encoder(real_objects_t1)
1394     latent_code_t2 = encoder(real_objects_t2)

1395
1396     latent_code = tf.concat([latent_code_t0, latent_code_t1, latent_code_t2], axis=1)

1397
1398     pred_objects_t3 = generator(latent_code, training=True)
1399     fake_logits = discriminator([pred_objects_t3, real_objects_t3], training=True)
1400     real_logits = discriminator([real_objects_t3, real_objects_t3], training=True)
1401     D_loss, fake_loss, real_loss, gp, ep = wgan_gp_discriminator_loss(real_logits,
1402     fake_logits, real_objects_t3, pred_objects_t3, real_objects_t3, GP_WEIGHT,
1403     EPS_WEIGHT)

1404
1405     return D_loss, fake_loss, real_loss, gp, ep

1406
1407 @tf.function
1408 def val_encdisc_loss(val_real_objects):
1409
1410     val_real_objects_t0 = tf.dtypes.cast(val_real_objects[:,0], tf.float32)
1411     val_real_objects_t1 = tf.dtypes.cast(val_real_objects[:,1], tf.float32)
1412     val_real_objects_t2 = tf.dtypes.cast(val_real_objects[:,2], tf.float32)
1413     val_real_objects_t3 = tf.dtypes.cast(val_real_objects[:,3], tf.float32)

1414
1415     # Encode
1416     val_latent_code_t0 = encoder(val_real_objects_t0)
1417     val_latent_code_t1 = encoder(val_real_objects_t1)
1418     val_latent_code_t2 = encoder(val_real_objects_t2)

```

```

    val_latent_code = tf.concat([val_latent_code_t0, val_latent_code_t1,
                                val_latent_code_t2], axis=1)

1420
1421     val_pred_objects_t3 = generator(val_latent_code, training=False)
1422     val_fake_logits = discriminator([val_pred_objects_t3, val_real_objects_t3],
1423                                     training=False)
1424     val_real_logits = discriminator([val_real_objects_t3, val_real_objects_t3],
1425                                     training=False)
1426     D_loss, fake_loss, real_loss, gp, ep = wgan_gp_discriminator_loss(
1427         val_real_logits, val_fake_logits, val_real_objects_t3, val_pred_objects_t3,
1428         val_real_objects_t3, GP_WEIGHT, EPS_WEIGHT)

1429
1430     return D_loss, fake_loss, real_loss, gp, ep

1431 # WGAN LOSSES
1432 @tf.function
1433 def wgan_gp_generator_loss(fake_logits):
1434     G_loss = -tf.math.reduce_mean(fake_logits)
1435     return G_loss
1436 @tf.function
1437 def wgan_gp_discriminator_loss(real_logits, fake_logits, real_objects, fake_objects,
1438                                 cond_objects, GP_WEIGHT, EPS_WEIGHT):
1439     """ Calculates the gradient penalty.
1440
1441     This loss is calculated on an interpolated image
1442     and added to the discriminator loss.
1443     """
1444
1445     # 1. Get the interpolated object
1446     alpha = tf.random.uniform([real_objects.shape[0], 1, 1, 1, 1], minval=0.0,
1447                               maxval=1.0, seed=None)
1448     differences = fake_objects - real_objects
1449     interpolates = real_objects + (alpha*differences)
1450
1451     # 2. Calculate the gradients w.r.t to this interpolated object.
1452     gradients = tf.gradients(discriminator([interpolates, cond_objects], training=
1453                               False), [interpolates])[0]
1454     norm = tf.math.sqrt(tf.math.reduce_sum(tf.square(gradients), axis=[1,2,3,4])) # norm over all gradients of 3D Grid
1455
1456     # 3. Calculate the norm of the gradients.
1457     gradient_penalty = tf.reduce_mean((norm-1.0)**2)
1458
1459     # Add the gradient penalty to the original discriminator loss and MEAN over
1460     # BATCH_SIZE + epsilon penalty
1461     fake_loss = tf.math.reduce_mean(fake_logits)
1462     real_loss = tf.math.reduce_mean(real_logits)
1463     gp = GP_WEIGHT * gradient_penalty
1464     ep = EPS_WEIGHT * tf.math.reduce_mean((real_logits)**2)
1465
1466     D_loss = fake_loss - real_loss + gp + ep

```

```

1460     return D_loss, fake_loss, real_loss, gp, ep
1462 ##### TRAINING STEP #####
1464
1465 @tf.function
1466 def train_step(obj_batch):
1468
1468     real_objects = obj_batch[..., np.newaxis]
1470
1470     for i in range(D_STEPS):
1471         with tf.GradientTape() as disc_tape:
1472             # discriminator loss
1473             EncDisc_loss = encdisc_loss(real_objects)
1474
1474             gradients_of_discriminator = disc_tape.gradient(EncDisc_loss[0],
1475 discriminator.trainable_variables)
1476             discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
1477 discriminator.trainable_variables))
1478
1478     with tf.GradientTape() as gen_tape, tf.GradientTape() as enc_tape:
1479         # generator loss
1480         EncGen_loss = encgen_loss(real_objects)
1482
1482         gradients_of_encoder = enc_tape.gradient(EncGen_loss[0], encoder.
1483 trainable_variables)
1484         encoder_optimizer.apply_gradients(zip(gradients_of_encoder, encoder.
1485 trainable_variables))
1486
1486         gradients_of_generator = gen_tape.gradient(EncGen_loss[0], generator.
1487 trainable_variables)
1488         generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.
1489 trainable_variables))
1488
1488     return EncGen_loss, EncDisc_loss
1490
1490 @tf.function
1491 def validation_step(val_obj_batch):
1492     val_real_objects = val_obj_batch[..., np.newaxis]
1494
1494     Val_EncDisc_Loss = val_encdisc_loss(val_real_objects)
1496
1496     Val_MSE_reconstruction_loss = MSE_reconstruction_loss(val_real_objects)
1498
1498     return Val_EncDisc_Loss, Val_MSE_reconstruction_loss
1500
1500 @tf.function
1501 def MSE_reconstruction_loss(real_objects):

```

```

1504     real_objects_t0 = tf.dtypes.cast(real_objects[:,0], tf.float32)
1505     real_objects_t1 = tf.dtypes.cast(real_objects[:,1], tf.float32)
1506     real_objects_t2 = tf.dtypes.cast(real_objects[:,2], tf.float32)
1507     real_objects_t3 = tf.dtypes.cast(real_objects[:,3], tf.float32)

1508     # Encode
1509     latent_code_t0 = encoder(real_objects_t0)
1510     latent_code_t1 = encoder(real_objects_t1)
1511     latent_code_t2 = encoder(real_objects_t2)

1512     latent_code = tf.concat([latent_code_t0, latent_code_t1, latent_code_t2], axis=1)

1514     pred_objects_t3 = generator(latent_code, training=False)

1516     mse = tf.keras.metrics.mean_squared_error(real_objects_t3, pred_objects_t3)
1517     MSE_loss = tf.reduce_mean(mse)

1519     return MSE_loss

1522 def JACCARD_reconstruction_loss(val_obj_batch):
1523     val_real_objects = val_obj_batch[..., np.newaxis]

1524     val_real_objects_t0 = tf.dtypes.cast(val_real_objects[:,0], tf.float32)
1525     val_real_objects_t1 = tf.dtypes.cast(val_real_objects[:,1], tf.float32)
1526     val_real_objects_t2 = tf.dtypes.cast(val_real_objects[:,2], tf.float32)
1527     val_real_objects_t3 = tf.dtypes.cast(val_real_objects[:,3], tf.float32)

1529     # Encode
1530     val_latent_code_t0 = encoder(val_real_objects_t0)
1531     val_latent_code_t1 = encoder(val_real_objects_t1)
1532     val_latent_code_t2 = encoder(val_real_objects_t2)

1534     val_latent_code = tf.concat([val_latent_code_t0, val_latent_code_t1,
1535                                 val_latent_code_t2], axis=1)
1536     #pred_objects_t1 = generator([latent_code, real_labels, real_objects_t0],
1537     #                           training=False)
1538     val_pred_objects_t3 = generator(val_latent_code, training=False)

1540     if NORMALIZE:
1541         real = np.asarray(val_real_objects_t3 > 0.0, bool)
1542         pred = np.asarray(val_pred_objects_t3 > 0.0, bool)
1543     else:
1544         real = np.asarray(val_real_objects_t3 > 0.5, bool)
1545         pred = np.asarray(val_pred_objects_t3 > 0.5, bool)

1546     j_and = tf.math.reduce_sum(np.double(np.bitwise_and(real, pred)), axis=[1,2,3,4])
1547     j_or = tf.math.reduce_sum(np.double(np.bitwise_or(real, pred)), axis=[1,2,3,4])

```

```

JACCARD_loss = tf.reduce_mean(np.nan_to_num(1.0 - j_and / j_or, nan=0.0))

1550
1551     return JACCARD_loss

1552 #####
1553 ##### SOME UTILITY FUNCTIONS #####
1554 #####
1555

1556 def generate_sample_objects(objects, total_iterations, tag="pred"):
    # Random Sample at every VAL_SAVE_FREQ
    i=0
    generated_objects = []
    for X in objects:
        X = tf.dtypes.cast(X, tf.float32)
        # Encode
        latent_code_t0 = encoder(X[0][0][np.newaxis,...])
        latent_code_t1 = encoder(X[0][1][np.newaxis,...])
        latent_code_t2 = encoder(X[0][2][np.newaxis,...])

1556

        latent_code = tf.concat([latent_code_t0, latent_code_t1, latent_code_t2], axis=1)

1558

        sample_object = generator(latent_code, training=False)
        if NORMALIZE:
            object_out = np.squeeze(sample_object>0.0)
        else:
            object_out = np.squeeze(sample_object>0.5)

1574

        try:
            d.plotMeshFromVoxels(object_out, obj=model_directory+tag+"_"+str(i)+"_next"+"_iter_"+str(total_iterations))
        except:
            print(f"Cannot generate STL, Marching Cubes Algo failed for sample {i}")
            # print("""Cannot generate STL, Marching Cubes Algo failed: Surface
            level must be within volume data range! \n
            # This may happen at the beginning of the training, if it still
            happens at later stages epoch>10 --> Check Object and try to change Marching
            Cube Threshold."""")

1582

        generated_objects.append(object_out)

1584

        i+=1

1586     print()

1588     return generated_objects

1590 def IOhousekeeping(model_directory, KEEP_LAST_N, VAL_SAVE_FREQ, RESTART_TRAINING,
total_iterations, KEEP_BEST_ONLY, best_iteration):
    if total_iterations > KEEP_LAST_N*VAL_SAVE_FREQ and total_iterations % VAL_SAVE_FREQ == 0:

```

```

1592     if KEEP_BEST_ONLY:
1593         fileList_all = glob.glob(model_directory + "*_iter_*")
1594         fileList_best = glob.glob(model_directory + "*_iter_" + str(int(
1595             best_iteration)) + "*")
1596         fileList_del = [ele for ele in fileList_all if ele not in fileList_best]
1597
1598     else:
1599         fileList_del = glob.glob(model_directory + "*_iter_" + str(int(
1600             total_iterations -KEEP_LAST_N*VAL_SAVE_FREQ)) + "*")
1601
1602         # Iterate over the list of filepaths & remove each file .
1603         for filePath in fileList_del:
1604             os.remove(filePath)
1605
1606 ###### LAST PREPARATION STEPS BEFORE TRAINING STARTS #####
1607 #####
1608 #generate folders:
1609 if RESTART_TRAINING:
1610     model_directory = os.getcwd() + "/" + RESTART_FOLDER_NAME + "/"
1611 else:
1612     named_tuple = time.localtime() # get struct_time
1613     time_string = time.strftime("%Y%m%d_%H%M%S", named_tuple)
1614     model_directory = os.getcwd() + "/" + time_string + "_" + os.path.splitext(os.path.
1615     basename(__file__))[0] + "/"
1616     if not os.path.exists(model_directory):
1617         os.makedirs(model_directory)
1618
1619 # Save running python file to run directory
1620 if RESTART_TRAINING:
1621     named_tuple = time.localtime() # get struct_time
1622     time_string = time.strftime("%Y%m%d_%H%M%S", named_tuple)
1623     copyfile(__file__, model_directory+time_string +"_restart_" +os.path.basename(
1624     __file__))
1625 else:
1626     copyfile(__file__, model_directory+os.path.basename(__file__))
1627
1628 # Activate Console Logging:
1629 if LOGGING:
1630     sys.stdout = Logger(filename=model_directory+"log_out.txt")
1631
1632 # PRINT NETWORK SUMMARY
1633 print(encoder.summary())
1634 print(generator.summary())
1635 print(discriminator.summary())
1636
1637 ##### GET DATA/OBJECTS #####
1638 #####
1639 TRAINING_FILERAMES = tf.io.gfile.glob(TRAIN_DATASET_LOC+"/*.tfrecords")[:N_TRAINING]

```

```

1638 VALIDATION_Filenames = tf.io.gfile.glob(VALID_DATASET_LOC+"/*.tfrecords")[:N_VALIDATION]

1640 training_set_size = len(TRAINING_Filenames)
validation_set_size = len(VALIDATION_Filenames)
1642 training_samples = training_set_size * (T//TINC-3) // BATCH_SIZE * BATCH_SIZE
validation_samples = validation_set_size * (T//TINC-3) // BATCH_SIZE * BATCH_SIZE
1644 print()
print("Total Number of TFRecord Files to Train : ", training_set_size)
1646 print("Total Number of Samples to Train: ", training_samples)
print()
1648 print("Total Number of TFRecord Files for Validation:", validation_set_size)
print("Total Number of Samples for Validation:", validation_samples)
1650 print()

1652 # Convert/Decode dataset from tfrecords file for training
train_objects = p4d.get_dataset(TRAINING_Filenames, shuffle=True, batch_size=BATCH_SIZE, t=T, tinc=TINC, normalize=NORMALIZE )
1654 val_objects = p4d.get_dataset(VALIDATION_Filenames, shuffle=False, batch_size=BATCH_SIZE, t=T, tinc=TINC, normalize=NORMALIZE )

1656 # save some real examples
i=0
1658 test_objects = []
for X in train_objects.take(N_SAMPLES):
    Xt0_2 = X[0][0:3]
    Xt3 = X[0][3]
1662
    try:
        sample_name = model_directory+"train_"+str(i)+"_init"+str(t)
        d.plotMeshFromVoxels(np.array(Xt0_2[0]), obj=sample_name)
        sample_name = model_directory+"train_"+str(i)+"_next"+str(t)
        d.plotMeshFromVoxels(np.array(Xt3), obj=sample_name)
    except:
        print("Example Object couldnt be generated with Marching Cube Algorithm.
Probable Reason: Empty Input Tensor, All Zeros")
1670
        test_objects.append([Xt0_2[...,np.newaxis]]) # for validation of reconstruction
        during training , see generate_sample_objects():
    i+=1

1674 # save some validation examples
i=0
1676 valid_objects = []
for X in val_objects.take(N_SAMPLES):
    Xt0_2 = X[0][0:3]
    Xt3 = X[0][3]
1680
    try:
        sample_name = model_directory+"valid_"+str(i)+"_init"+str(t)

```

```

1684     d.plotMeshFromVoxels(np.array(Xt0_2[0]), obj=sample_name)
1685     sample_name = model_directory+"valid_"+str(i)+"_next"+str(t)
1686     d.plotMeshFromVoxels(np.array(Xt3), obj=sample_name)
1687 except:
1688     print("Example Object couldnt be generated with Marching Cube Algorithm.
1689 Probable Reason: Empty Input Tensor, All Zeros")
1690
1691
1692 #####
1693 ##### OPTIMIZER #####
1694 #####
1695 if CLR:
1696     steps_per_epoch = len(TRAINING_FILERAMES) // BATCH_SIZE
1697     clr_g = tfa.optimizers.CyclicalLearningRate(initial_learning_rate=G_LR,
1698         maximal_learning_rate=G_LR*10,
1699         scale_fn=lambda x: 1/(2.**x-1)),
1700         step_size=2 * steps_per_epoch
1701     )
1702     clr_d = tfa.optimizers.CyclicalLearningRate(initial_learning_rate=D_LR,
1703         maximal_learning_rate=D_LR*10,
1704         scale_fn=lambda x: 1/(2.**x-1)),
1705         step_size=2 * steps_per_epoch
1706     )
1707
1708     encoder_optimizer = Adam(learning_rate=clr_g, beta_1=0.5, beta_2=0.999, epsilon
1709     =1e-07)
1710     generator_optimizer = Adam(learning_rate=clr_g, beta_1=0.5, beta_2=0.999,
1711     epsilon=1e-07)
1712     discriminator_optimizer = Adam(learning_rate=clr_d, beta_1=0.5, beta_2=0.999,
1713     epsilon=1e-07)
1714
1715 else:
1716     encoder_optimizer = Adam(learning_rate=G_LR, beta_1=0.5, beta_2=0.999, epsilon=1
1717     e-07)
1718     generator_optimizer = Adam(learning_rate=G_LR, beta_1=0.5, beta_2=0.999, epsilon
1719     =1e-07)
1720     discriminator_optimizer = Adam(learning_rate=D_LR, beta_1=0.5, beta_2=0.999,
1721     epsilon=1e-07)
1722
1723 #####
1724 # Define Checkpoint
1725 checkpoint_path = model_directory+"checkpoints /"
1726 checkpoint_dir = os.path.dirname(checkpoint_path)
1727
1728 ckpt = tf.train.Checkpoint(encoder_optimizer=encoder_optimizer,
1729                         generator_optimizer=generator_optimizer,

```

```

1726
1727         discriminator_optimizer=discriminator_optimizer ,
1728         encoder=encoder ,
1729         generator=generator ,
1730         discriminator=discriminator)
1731
1732 manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=1)
1733
1734 def save_checkpoints(ckpt, model_directory):
1735     if not os.path.exists(model_directory+"checkpoints/"):
1736         os.makedirs(model_directory+"checkpoints/")
1737     manager.save()
1738
1739 if RESTART_TRAINING:
1740     total_iterations = RESTART_ITERATION
1741     RESTART_EPOCH = training_samples // BATCH_SIZE
1742     if manager.latest_checkpoint:
1743         ckpt.restore(manager.latest_checkpoint)
1744         print("\nRestored from {} \n".format(manager.latest_checkpoint))
1745     else:
1746         print("No Checkpoint found --> Initializing from scratch.")
1747
1748 ##### TRAINING #####
1749
1750 print("-----")
1751 print("RUN TRAINING:\n")
1752 start_time = time.time()
1753 iter_arr, val_iter_arr = [], []
1754 G_losses = []
1755 MSE_losses = []
1756 JACCARD_losses = []
1757 D_losses = []
1758 D_val_loss, MSE_val_loss, JACCARD_val_loss = [], [], []
1759 n_objects = 0
1760 total_iterations = 0
1761 best_iteration = VAL_SAVE_FREQ
1762 best_loss = 1e10
1763 for epoch in range(1, N_EPOCHS+1):
1764     ##### # START OF EPOCH
1765     t_epoch_start = time.time()
1766     n_batch=0
1767     #####
1768     for obj_batch in train_objects:
1769         # START OF BATCH
1770         batch_time = time.time()
1771         G_loss, D_loss = train_step(obj_batch)
1772         #Append arrays
1773         G_losses.append(G_loss[0])
1774

```

```

MSE_losses.append(G_loss[1])
D_losses.append(-D_loss[0])

1778 n_batch+=1
1780 total_iterations+=1
iter_arr.append(total_iterations)

1782 JACCARD_loss = JACCARD_reconstruction_loss(obj_batch)
1784 JACCARD_losses.append(JACCARD_loss)
print("E: %3d/%5d | B: %3d/%3d | I: %5d | T: %5.1f | dt: %2.2f || LOSSES: D
%.4f | G %.4f | MSE %.4f | JACCARD %.4f" \
      % (epoch, N_EPOCHS, n_batch, np.ceil(training_set_size*(T//TINC-3)//
1786 BATCH_SIZE), total_iterations, time.time() - start_time, time.time() -
batch_time, -D_loss[0], G_loss[0], G_loss[1], JACCARD_loss))

1788 if PRINT_D_LOSSES:
    print("\nfake_loss: ", D_loss[1].numpy(), "real_loss: ", D_loss[2].numpy()
(), "gp: ", D_loss[3].numpy(), "ep: ", D_loss[4].numpy(), "\n\n")

1790 ##### # AFTER N TOTAL ITERATIONS GET VALIDATION LOSSES, SAVE GENERATED OBJECTS AND
1792 MODEL:
1794 if (total_iterations % VAL_SAVE_FREQ == 0) and VAL_SAVE_FREQ != -1:
    print("\nRUN VALIDATION, SAVE OBJECTS, MODEL AND PLOT LOSSES... ")

1796 Val_losses = []
1798 Val_MSE_losses = []
Val_JACCARD_losses = []
for val_obj_batch in val_objects:
    Val_loss, MSE_recon = validation_step(val_obj_batch)
    Val_losses.append(-Val_loss[0]) # Collect Val loss per Batch
    Val_MSE_losses.append(MSE_recon) # Collect Val loss per Batch
    JACCARD_recon = JACCARD_reconstruction_loss(val_obj_batch)
    Val_JACCARD_losses.append(JACCARD_recon) # Collect Val loss per
1800 Batch

1804 D_val_loss.append(np.mean(Val_losses)) # Mean over batches
1806 MSE_val_loss.append(np.mean(Val_MSE_losses)) # Mean over batches
1808 JACCARD_val_loss.append(np.mean(Val_JACCARD_losses)) # Mean over batches
val_iter_arr.append(total_iterations)

1810 ##### # Output generated objects
1812 generate_sample_objects(test_objects, total_iterations, "train")
generate_sample_objects(valid_objects, total_iterations, "valid")
1814 # # Plot and Save Loss Curves
render_graphs(model_directory, G_losses, D_losses, D_val_loss,
MSE_losses, MSE_val_loss, JACCARD_losses, JACCARD_val_loss, iter_arr,
val_iter_arr, RESTART_TRAINING) #this will only work after a 50 iterations to
allow for proper averaging

```

```

1816     if KEEP_BEST_ONLY:
1817         if BEST_METRIC == "JACCARD":
1818             BEST_LOSS_METRIC = JACCARD_val_loss
1819             KEEP_LAST_N = 1
1820         elif BEST_METRIC == "MSE":
1821             BEST_LOSS_METRIC = MSE_val_loss
1822             KEEP_LAST_N = 1
1823         if len(BEST_LOSS_METRIC) > 1 and (BEST_LOSS_METRIC[-1] <= best_loss):
1824             :
1825             best_loss = BEST_LOSS_METRIC[-1]
1826             best_iteration = total_iterations
1827             # Save models
1828             encoder.save(model_directory+"_trained_encoder_"+str(
1829             total_iterations)+".h5")
1830             generator.save(model_directory+"_trained_generator_"+str(
1831             total_iterations)+".h5")
1832         else:
1833             # Save models
1834             encoder.save(model_directory+"_trained_encoder_"+str(
1835             total_iterations)+".h5")
1836             generator.save(model_directory+"_trained_generator_"+str(
1837             total_iterations)+".h5")
1838
1839             # # Delete Model, keep best or last N models
1840             IOhousekeeping(model_directory, KEEP_LAST_N, VAL_SAVE_FREQ,
1841             RESTART_TRAINING, total_iterations, KEEP_BEST_ONLY, best_iteration)
1842
1843
1844             ######
1845             # SAVE CHECKPOINT FOR RESTART:
1846             if total_iterations % WRITE_RESTART_FREQ == 0:
1847                 print("WRITE RESTART FILE...\n")
1848                 # Save Checkpoint after every 100 epoch
1849                 save_checkpoints(ckpt, model_directory)
1850
1851
1852             # END OF BATCH
1853             #####
1854             # STOP TRAINING AFTER MAX DEFINED ITERATIONS ARE REACHED
1855             if total_iterations == MAX_ITERATIONS :
1856                 break
1857             if total_iterations == MAX_ITERATIONS:
1858                 break
1859
1860             # Print Status at the end of the epoch
1861             dt_epoch = time.time() - t_epoch_start
1862             n_objects = n_objects + np.ceil(training_set_size*(T//TINC-3)//BATCH_SIZE)*
1863             BATCH_SIZE
1864             print("\n-----")
1865             print("END OF EPOCH ", epoch, " | Total Training Iterations: ", int(
1866             total_iterations), " | Total Number of objects trained: ", int(n_objects/1000),
1867             "K", " | time elapsed:", str(int((time.time() - start_time) / 60.0 )), "min")

```

```
    print("-----\n\n")
)
# END OF EPOCH
#####
#####
#
print("\n TRAINING DONE! \n")
print("Total Training Time: ", str((time.time() - start_time) / 60.0), "min" )
```

Listing B.5: Multiple Frame Input Training Algorithm

B.6 Multiple Frame Input Training Algorithm - Data Parser

```
1000 import os
os.environ[ 'TF_CPP_MIN_LOG_LEVEL ' ] = '1'
1002 import tensorflow as tf
import numpy as np
1004
AUTOTUNE = tf.data.AUTOTUNE
1006
1007 def decode_object(example):
1008     object4d = tf.io.decode_raw(example[ 'object4d '], tf.uint8)
1009     labelBetti = tf.io.decode_raw(example[ 'labelBetti '], tf.uint8)
1010     labelObjs = tf.io.decode_raw(example[ 'labelObjs '], tf.uint8)
1011
1012     object4d = tf.reshape(object4d, [128, 16, 16, 16])
1013     labelBetti = tf.reshape(labelBetti, [4])
1014     labelObjs = tf.reshape(labelObjs, [4])
1015
1016     return object4d, labelBetti, labelObjs
1017
1018 def read_tfrecord(example):
1019     global NORMALIZE
1020     global T
1021     global TINC
1022
1023     object_feature_description = {
1024         'object4d ': tf.io.FixedLenFeature([], tf.string),
1025         'labelBetti ': tf.io.FixedLenFeature([], tf.string),
1026         'labelObjs ': tf.io.FixedLenFeature([], tf.string),
1027     }
1028
1029     example = tf.io.parse_single_example(example, object_feature_description)
1030
1031     object4d, labelBetti, labelObjs = decode_object(example)
1032
1033     CROP=(128-T)//2
1034     # Reshape into time slices of n time steps (axis=1)
1035     object4d = object4d[CROP:128-CROP:TINC]
1036     object4d = tf.reshape(object4d, [T//TINC, 16, 16, 16])
1037     rearranged_objs4d = []
1038     for t in range(object4d.shape[0]):
1039         if t == object4d.shape[0] - 3:
1040             break
1041         rearranged_objs4d.append(object4d[t])
1042         rearranged_objs4d.append(object4d[t+1])
1043         rearranged_objs4d.append(object4d[t+2])
1044         rearranged_objs4d.append(object4d[t+3])
1045
1046
```

```

object4d = tf.reshape(tf.stack(rearranged_objs4d), [T//TINC-3, 4, 16, 16, 16])
1048
1049 if NORMALIZE:
1050     object4d = tf.cast(object4d, tf.float32) - 0.5
1051
1052 return object4d #, tf.ones([object4d.shape[0],4], tf.uint8)*labelObjs
1053 #return object4d, tf.ones(object4d.shape[0], tf.uint8)*labelBetti[-1]#, tf.ones
1054 ([object4d.shape[0],4], tf.uint8)*labelObjs
1055
1056 def load_dataset(filenames):
1057     options = tf.data.Options()
1058     options.deterministic = False # disable order, increase speed
1059     dataset = tf.data.TFRecordDataset(filenames, compression_type="GZIP",
1060     num_parallel_reads=AUTOTUNE) # automatically interleaves reads from multiple
1061     files
1062     dataset = dataset.with_options(options) # uses data as soon as it streams in,
1063     rather than in its original order
1064     dataset = dataset.map(read_tfrecord, num_parallel_calls=AUTOTUNE)
1065     return dataset
1066
1067 def get_dataset(filenames: list, shuffle: bool, batch_size: int, t: int, tinc: int,
1068 normalize: bool):
1069     global NORMALIZE
1070     global T
1071     global TINC
1072     NORMALIZE = normalize
1073     TINC = tinc
1074     T = t
1075     dataset = load_dataset(filenames)
1076     dataset = dataset.unbatch()
1077     if shuffle:
1078         dataset = dataset.shuffle(len(filenames)*(T//TINC-3)//2) # ALL: len(
1079         filenames)*(T//TINC)
1080     if batch_size is not None:
1081         dataset = dataset.batch(batch_size, drop_remainder=True)
1082     dataset = dataset.prefetch(buffer_size=AUTOTUNE)
1083
1084     return dataset
1085
1086 # FOR TESTING:
1087 ''
1088 folder_name="Z:/Master_Thesis/data/4D_128_32_topological_dataset_0-31_16_label8only"
1089
1090 FILENAMES = tf.io.gfile.glob(folder_name + "/*.tfrecords")[:7]
1091 split_ind = int(1 * len(FILENAMES))
1092 TRAINING_FILENOAMES, VALID_FILENOAMES = FILENAMES[:split_ind], FILENAMES[split_ind:]
1093
1094 #TEST_FILENOAMES = tf.io.gfile.glob(GCS_PATH + "/tfrecords/test*.tfrec")
1095 print("Train TFRecord Files:", len(TRAINING_FILENOAMES))

```

```

print("Validation TFRecord Files:", len(VALID_Filenames))
# print("Test TFRecord Files:", len(TEST_Filenames))

BATCH_SIZE = 64

train_dataset = get_dataset(TRAINING_Filenames, shuffle=True, batch_size=BATCH_SIZE,
                           t=80, tinc=4, normalize=False)
#print(len(list(train_dataset)) * BATCH_SIZE)
#object4d, labelBetti = next(iter(train_dataset))
#print(object4d.shape, labelBetti.shape)
#print(object4d[0], labelBetti)

#print(tf.data.experimental.cardinality(train_dataset))

def jaccard(x,y):
    x = np.asarray(x, bool) # Not necessary, if you keep your data
    y = np.asarray(y, bool) # in a boolean array already!

    #j = tf.math.reduce_sum(tf.square(differences), axis=[1,2,3,4])
    j_and = tf.math.reduce_sum(np.double(np.bitwise_and(x, y)), axis=[1,2,3])
    j_or = tf.math.reduce_sum(np.double(np.bitwise_or(x, y)), axis=[1,2,3])

    return tf.reduce_mean(1 - j_and / j_or)

import dataIO
for object4d, labelBetti in train_dataset.take(1):
    print(object4d.shape, labelBetti.shape)
    jaccard_ = jaccard(object4d[:,0], object4d[:,1])

    i=0
    for object4d_i, labelBetti_i in zip(object4d, labelBetti):
        print(object4d_i.shape, labelBetti_i.shape)
        #print(object4d_i.shape, labelBetti_i)

        for object4d_ii in object4d_i:
            dataIO.plotFromVoxels(object4d_ii.numpy(), [i, labelBetti_i])
            i+=1
    ...

```

Listing B.6: Multiple Frame Input Training Algorithm - Data Parser

B.7 Multiple Frame Input Prediction Algorithm

```
1000 import os
1001 import tensorflow as tf
1002 from keras.models import load_model
1003 import numpy as np
1004 # OWN LIBRARIES
1005 from utils import parse_tfrecords_4D_16_t3_for_training as p4d
1006 from utils import dataIO as d
1007
1008 base_dir = r"Z:\Master_Thesis\code\99_FINAL_MODELS_FOR_THESIS"
1009
1010 MODEL_LOCATION      = os.path.join(base_dir, "aengan3d_16_t3_T80_TINC4")
1011
1012 TEST_DATASET_LOC    = "Z:/Master_Thesis/data/02_4D_test_datasets"
1013
1014 T                  = int(MODEL_LOCATION[-8:-6])
1015 TINC                = int(MODEL_LOCATION[-1])
1016 N_PREDICT          = 5
1017
1018 if not os.path.exists(os.path.join(base_dir, f'objects/T{T}_TINC{TINC}')):
1019     os.makedirs(os.path.join(base_dir, f'objects/T{T}_TINC{TINC}'))
1020 #####
1021 ##### GET Data/Objects #####
1022 #####
1023 #####
1024 TEST_Filenames = tf.io.gfile.glob(os.path.join(TEST_DATASET_LOC, "*tfrecords"))
test_set_size = len(TEST_Filenames)
1026
1027 print()
1028 print("Total Number of TFRecord Files to Test : ", test_set_size)
print()
1029
1030 # Convert/Decode dataset from tfrecords file for training
1031 test_objects = p4d.get_dataset(TEST_Filenames, shuffle=True, batch_size=1, t=T, tinc=TINC, normalize=False)
1032
1033 encoder_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*encoder*.h5"))
encoder = load_model(encoder_model[-1])
1035
1036 generator_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*generator*.h5"))
generator = load_model(generator_model[-1])
1038
1039
1040 def predictions(objects):
1041     n=0
1042     for X in objects:
1043         d.plotFromVoxels(np.squeeze(X[0,0]), f"{n}_T{T}_TINC{TINC}_t-1")
1044         d.plotMeshFromVoxels(np.squeeze(X[0,0]), obj=f"objects/T{T}_TINC{TINC}/multiple_{n}_T{T}_TINC{TINC}_t0")
```

```

1046     d.plotFromVoxels(np.squeeze(X[0,1]), f"{{n}}_T{T}_TINC{TINC}_t-2")
1047     d.plotMeshFromVoxels(np.squeeze(X[0,1]), obj=f"objects/T{T}_TINC{TINC}/
multiple_{n}_T{T}_TINC{TINC}_t1")
1048     d.plotFromVoxels(np.squeeze(X[0,2]), f"{{n}}_T{T}_TINC{TINC}_t0")
1049     d.plotMeshFromVoxels(np.squeeze(X[0,2]), obj=f"objects/T{T}_TINC{TINC}/
multiple_{n}_T{T}_TINC{TINC}_t2")
1050     d.plotFromVoxels(np.squeeze(X[0,3]), f"{{n}}_T{T}_TINC{TINC}_t1")
1051     d.plotMeshFromVoxels(np.squeeze(X[0,3]), obj=f"objects/T{T}_TINC{TINC}/
multiple_{n}_T{T}_TINC{TINC}_t3")

1052     # Encode & Generate
1053     X0 = tf.dtypes.cast(X[:,0], tf.float32)

1054
1055     Xminus2 = tf.dtypes.cast(X[:,0], tf.float32)
1056     Xminus1 = tf.dtypes.cast(X[:,1], tf.float32)
1057     X0 = tf.dtypes.cast(X[:,2], tf.float32)
1058     latent_code_tminus2 = encoder(Xminus2)
1059     latent_code_tminus1 = encoder(Xminus1)
1060     latent_code_t0 = encoder(X0)
1061     latent_code = tf.concat([latent_code_tminus2, latent_code_tminus1,
1062                             latent_code_t0], axis=1)
1063     pred_object = np.squeeze(generator(latent_code, training=False))
1064     pred_object = np.asarray(pred_object>0.5, bool)

1065
1066     d.plotFromVoxels(pred_object, f"{{n}}_T{T}_TINC{TINC}_t1_pred")
1067     d.plotMeshFromVoxels(pred_object, obj=f"objects/T{T}_TINC{TINC}/multiple_{n}
_T{T}_TINC{TINC}_t3_pred")
1068     n+=1
1069     if n==N_PREDICT:
1070         break
1071
1072 predictions(test_objects)

```

Listing B.7: Multiple Frame Input Prediction Algorithm

B.8 Multiple Frame Input Long-Term Prediction Algorithm

```
1000 import os
1001 import tensorflow as tf
1002 from keras.models import load_model
1003 import numpy as np
1004 # OWN LIBRARIES
1005 from utils import parse_tfrecords_4D_16_for_LongTerm_prediction as p4d
1006 from utils import dataIO as d

1008 base_dir = r"Z:\Master_Thesis\code\99_FINAL_MODELS_FOR_THESIS"
1009
1010 MODEL_LOCATION      = os.path.join(base_dir, "aengan3d_16_t3_T20_TINC1")
1011
1012 TEST_DATASET_LOC    = "Z:/Master_Thesis/data/02_4D_test_datasets"
1013
1014 T                  = int(MODEL_LOCATION[-8:-6])
1015 TINC                = int(MODEL_LOCATION[-1])
1016 N_PREDICT          = 5
1017 MAX_TIME           = 5

1018 if not os.path.exists(os.path.join(base_dir, f'objects_longterm/T{T}_TINC{TINC}')):
1019     os.makedirs(os.path.join(base_dir, f'objects_longterm/T{T}_TINC{TINC}'))
1020 #####
1021 ##### GET Data/ Objects #####
1022 #####
1023 #####
1024 #####
1025 TEST_Filenames = tf.io.gfile.glob(os.path.join(TEST_DATASET_LOC, "*.tfrecords"))
1026 test_set_size = len(TEST_Filenames)

1027 print()
1028 print("Total Number of TFRecord Files to Test : ", test_set_size)
1029 print()

1030 #####
1031 #####
1032 # Convert/Decode dataset from tfrecords file for training
1033 test_objects = p4d.get_dataset(TEST_Filenames, shuffle=False, batch_size=None, t=T,
1034                                 tinc=TINC, normalize=False)
1035
1036 encoder_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*encoder*.h5"))
1037 encoder = load_model(encoder_model[-1])

1038 generator_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*generator*.h5"))
1039 generator = load_model(generator_model[-1])

1040 #####
1041 #####
1042 def predictions(objects):
1043     n=0
1044     for X in objects:
1045         d.plotFromVoxels(np.squeeze(X[0]), f"{n}_T{T}_TINC{TINC}_t0")
```

```

1046     d.plotMeshFromVoxels(np.squeeze(X[0]), obj=f"objects_longterm/T{T}_TINC{TINC}"
1047     }/multiple_{n}_T{T}_TINC{TINC}_t0")
1048     d.plotFromVoxels(np.squeeze(X[1]), f"{n}_T{T}_TINC{TINC}_t1")
1049     d.plotMeshFromVoxels(np.squeeze(X[1]), obj=f"objects_longterm/T{T}_TINC{TINC}"
1050     }/multiple_{n}_T{T}_TINC{TINC}_t1")
1051     d.plotFromVoxels(np.squeeze(X[2]), f"{n}_T{T}_TINC{TINC}_t2")
1052     d.plotMeshFromVoxels(np.squeeze(X[2]), obj=f"objects_longterm/T{T}_TINC{TINC}"
1053     }/multiple_{n}_T{T}_TINC{TINC}_t2")
1054     d.plotFromVoxels(np.squeeze(X[3]), f"{n}_T{T}_TINC{TINC}_t3")
1055     d.plotMeshFromVoxels(np.squeeze(X[3]), obj=f"objects_longterm/T{T}_TINC{TINC}"
1056     }/multiple_{n}_T{T}_TINC{TINC}_t3")
1057     d.plotFromVoxels(np.squeeze(X[4]), f"{n}_T{T}_TINC{TINC}_t4")
1058     d.plotMeshFromVoxels(np.squeeze(X[4]), obj=f"objects_longterm/T{T}_TINC{TINC}"
1059     }/multiple_{n}_T{T}_TINC{TINC}_t4")
1060     d.plotFromVoxels(np.squeeze(X[5]), f"{n}_T{T}_TINC{TINC}_t5")
1061     d.plotMeshFromVoxels(np.squeeze(X[5]), obj=f"objects_longterm/T{T}_TINC{TINC}"
1062     }/multiple_{n}_T{T}_TINC{TINC}_t5")
1063     d.plotFromVoxels(np.squeeze(X[6]), f"{n}_T{T}_TINC{TINC}_t5")
1064     d.plotMeshFromVoxels(np.squeeze(X[6]), obj=f"objects_longterm/T{T}_TINC{TINC}"
1065     }/multiple_{n}_T{T}_TINC{TINC}_t6")
1066     d.plotFromVoxels(np.squeeze(X[7]), f"{n}_T{T}_TINC{TINC}_t5")
1067     d.plotMeshFromVoxels(np.squeeze(X[7]), obj=f"objects_longterm/T{T}_TINC{TINC}"
1068     }/multiple_{n}_T{T}_TINC{TINC}_t7")
1069     #Loop over time per sample
1070     X = tf.dtypes.cast(X, tf.float32)
1071     for t in range(2,X.shape[1]-3):
1072         if t==2:
1073             X_minus2 = X[0][np.newaxis ,...]
1074             X_minus1 = X[1][np.newaxis ,...]
1075             X0 = X[2][np.newaxis ,...]
1076
1077         # Encode & Generate
1078         latent_code_tminus2 = encoder(X_minus2)
1079         latent_code_tminus1 = encoder(X_minus1)
1080         latent_code_t0 = encoder(X0)
1081         latent_code = tf.concat([latent_code_tminus2, latent_code_tminus1,
1082         latent_code_t0], axis=1)
1083         pred_object = np.squeeze(generator(latent_code, training=False))
1084         pred_object = np.asarray(pred_object>0.5, bool)
1085
1086         d.plotFromVoxels(pred_object, f"{n}_T{T}_TINC{TINC}_t{t+1}_pred")
1087         d.plotMeshFromVoxels(pred_object, obj=f"objects_longterm/T{T}_TINC{TINC}"
1088         }/multiple_{n}_T{T}_TINC{TINC}_t{t+1}_pred")
1089
1090         if t==MAX_TIME+2-1:
1091             break
1092
1093         X_minus2 = X_minus1
1094         X_minus1 = X0
1095         X0 = pred_object[np.newaxis ,...]

```

```
1086
1087     n+=1
1088     if n==N_PREDICT:
1089         break
1090
1091 predictions(test_objects)
```

Listing B.8: Multiple Frame Input Prediction Algorithm

B.9 Long-Term Prediction - Data Parser

```
1000 import os
os.environ[ 'TF_CPP_MIN_LOG_LEVEL ' ] = '1'
1002 import tensorflow as tf
1003 import numpy as np
1004
AUTOTUNE = tf.data.AUTOTUNE
1006
1007 def decode_object(example):
1008     object4d = tf.io.decode_raw(example[ 'object4d '], tf.uint8)
1009     labelBetti = tf.io.decode_raw(example[ 'labelBetti '], tf.uint8)
1010     labelObjs = tf.io.decode_raw(example[ 'labelObjs '], tf.uint8)
1011
1012     object4d = tf.reshape(object4d, [128, 16, 16, 16])
1013     labelBetti = tf.reshape(labelBetti, [4])
1014     labelObjs = tf.reshape(labelObjs, [4])
1015
1016     return object4d, labelBetti, labelObjs
1017
1018 def read_tfrecord(example):
1019     global NORMALIZE
1020     global T
1021     global TINC
1022
1023     object_feature_description = {
1024         'object4d ': tf.io.FixedLenFeature([], tf.string),
1025         'labelBetti ': tf.io.FixedLenFeature([], tf.string),
1026         'labelObjs ': tf.io.FixedLenFeature([], tf.string),
1027     }
1028
1029     example = tf.io.parse_single_example(example, object_feature_description)
1030
1031     object4d, labelBetti, labelObjs = decode_object(example)
1032
1033     CROP=(128-T)//2
1034     # Reshape into time slices of n time steps (axis=1)
1035     object4d = object4d[CROP:128-CROP:TINC]
1036     object4d = tf.reshape(object4d, [T//TINC, 16, 16, 16])
1037     #object4d = tf.reshape(object4d, [128, 16, 16, 16])
1038     #print("Time_Points: ", object4d.shape[0])
1039     if NORMALIZE:
1040         object4d = tf.cast(object4d, tf.float32) - 0.5
1041
1042     return object4d #, tf.ones([ object4d.shape[0],4 ], tf.uint8)*labelObjs
1043
1044 def load_dataset(filenames):
1045     ignore_order = tf.data.Options()
1046     ignore_order.experimental_deterministic = False # disable order, increase speed
```

```

dataset = tf.data.TFRecordDataset(filenames, compression_type="GZIP",
num_parallel_reads=AUTOTUNE) # automatically interleaves reads from multiple
files
1048 dataset = dataset.with_options(ignore_order) # uses data as soon as it streams
in, rather than in its original order
dataset = dataset.map(read_tfrecord, num_parallel_calls=AUTOTUNE)
1050 return dataset

1052 def get_dataset(filenames: list, shuffle: bool, batch_size: int, t: int, tinc: int,
normalize: bool):
    global NORMALIZE
    global T
    global TINC
1054    NORMALIZE = normalize
1056    TINC = tinc
1058    T = t
1059    dataset = load_dataset(filenames)
1060    dataset = dataset.prefetch(buffer_size=AUTOTUNE)
#dataset = dataset.unbatch()
1062    if shuffle:
        dataset = dataset.shuffle(len(filenames)*(T//TINC))
1064    if batch_size is not None:
        dataset = dataset.batch(batch_size, drop_remainder=True)
1066    return dataset

1068 # FOR TESTING:
1069 """
1070 folder_name="Z:/Master_Thesis/data/4D_128_32_topological_dataset_0-31_16_label8only"
1072 FILENAMES = tf.io.gfile.glob(folder_name+"/*.tfrecords")[:1]
1074 print("Train TFRecord Files:", len(FILENAMES))

1076 train_dataset = get_dataset(FILENAMES, shuffle=False, batch_size=None, t=80, tinc=4,
normalize=False)
1078 import dataIO
1080 for object4d, labelBetti in train_dataset:
    print(object4d.shape, labelBetti)

1082 i=0
1084 for object4d_i, labelBetti_i in zip(object4d, labelBetti):
    print(object4d_i.shape, labelBetti_i.shape)
1086 #print(object4d_i.shape, labelBetti_i)

1088     dataIO.plotFromVoxels(object4d_i.numpy(), [i, labelBetti_i])
    i+=1
1090 """

```

Listing B.9: Long-Term Prediction - Data Parser

C Utility Functions

C.1 Function to Plot Losses and Save them to a File

```
1000 import os
1001 import numpy as np
1002 import matplotlib.pyplot as plt
1003
1004 def render_graphs(save_dir, track_g_loss, track_d_loss, track_d_val_loss,
1005     track_MSE_train_loss, track_MSE_val_loss, track_JACCARD_train_loss,
1006     track_JACCARD_val_loss, iterations_arr, val_iterations_arr, restart_training):
1007     # try:
1008         if not os.path.exists(save_dir + '/plots/'):
1009             os.makedirs(save_dir + '/plots/')
1010
1011         # SAVE HISTORY TO FILE:
1012         if restart_training:
1013             LOSSES = np.concatenate((np.asarray(iterations_arr)[:, np.newaxis], np.
1014                 asarray(track_g_loss)[:, np.newaxis], np.asarray(track_d_loss)[:, np.newaxis],
1015                 np.asarray(track_MSE_train_loss)[:, np.newaxis], np.asarray(
1016                     track_JACCARD_train_loss)[:, np.newaxis]), axis=1)
1017             header = "TRAINING ITERATION,           G_LOSS,          D_LOSS,
1018             MSE LOSS,      JACCARD LOSS"
1019             np.savetxt(save_dir + '/plots/_train_losses_restart.txt', LOSSES, fmt='
1020 %20d, %16.5f, %16.5f, %16.5f', delimiter=',', header=header)
1021
1022             VAL_LOSSES = np.concatenate((np.asarray(val_iterations_arr)[:, np.
1023                 newaxis], np.asarray(track_d_val_loss)[:, np.newaxis], np.asarray(
1024                     track_MSE_val_loss)[:, np.newaxis], np.asarray(track_JACCARD_val_loss)[:, np.
1025                     newaxis]), axis=1)
1026             header = "TRAINING ITERATION,           D_VAL LOSS,        MSE_VAL LOSS,
1027             JACCARD_VAL LOSS"
1028             np.savetxt(save_dir + '/plots/_val_losses_restart.txt', VAL_LOSSES, fmt='
1029 %20d, %16.5f, %16.5f, %16.5f', delimiter=',', header=header)
1030         else:
1031             LOSSES = np.concatenate((np.asarray(iterations_arr)[:, np.newaxis], np.
1032                 asarray(track_g_loss)[:, np.newaxis], np.asarray(track_d_loss)[:, np.newaxis],
1033                 np.asarray(track_MSE_train_loss)[:, np.newaxis], np.asarray(
1034                     track_JACCARD_train_loss)[:, np.newaxis]), axis=1)
1035             header = "TRAINING ITERATION,           G_LOSS,          D_LOSS,
1036             MSE LOSS,      JACCARD LOSS"
1037             np.savetxt(save_dir + '/plots/_train_losses.txt', LOSSES, fmt='
1038 %20d, %16.5f, %16.5f, %16.5f', delimiter=',', header=header)
```

```

1022
    VAL_LOSSES = np.concatenate((np.asarray(val_iterations_arr)[...,np.
newaxis], np.asarray(track_d_val_loss)[...,np.newaxis], np.asarray(
track_MSE_val_loss)[...,np.newaxis], np.asarray(track_JACCARD_val_loss)[...,np.
newaxis]), axis=1)
1024     header = "TRAINING ITERATION,           D_VAL LOSS,      MSE_VAL LOSS,
JACCARD_VAL LOSS"
        np.savetxt(save_dir+ '/plots/_val_losses.txt' , VAL_LOSSES, fmt='%.20d',
%.16.5f , %.16.5f , %.16.5f' , delimiter=',', header=header)
1026
1028
1028     track_d_loss = np.ma.masked_invalid(track_d_loss)#.compressed()
1029     track_g_loss = np.ma.masked_invalid(track_g_loss)#.compressed()
1030     high_d = np.percentile(track_d_loss , 99)
1031     high_g = np.percentile(track_g_loss , 99)
1032     low_d = np.percentile(track_d_loss , 1)
1033     low_g = np.percentile(track_g_loss , 1)
1034     if (len(track_d_val_loss) > 0) and (len(val_iterations_arr) > 0):
1035         track_d_val_loss = np.ma.masked_invalid(track_d_val_loss)#.compressed()
1036         high_d_val = np.percentile(track_d_val_loss , 95)
1037         low_d_val = np.percentile(track_d_val_loss , 5)
1038         high_y = max([high_d , high_g , high_d_val])
1039         low_y = min([low_d , low_g , low_d_val]) #- 0.5*min([high_d , high_g])
1040     else :
1041         high_y = max([high_d , high_g])
1042         low_y = min([low_d , low_g]) #- 0.5*min([high_d , high_g])
1044
1044     fig = plt.figure()
1046     plt.plot(iterations_arr , track_g_loss , color='crimson' , alpha=0.5, label='G-
loss')
1047     plt.plot(iterations_arr , track_d_loss , color='navy' , alpha=0.5, label='D-
loss')
1048     #plt.plot(iterations_arr , track_MSE_train_loss , color='seagreen' , alpha=0.5,
label='MSE-Train-loss')

1050     if (len(track_d_val_loss) > 0) and (len(val_iterations_arr) > 0):
1051         plt.plot(val_iterations_arr , track_d_val_loss , color='navy' , alpha=0.5,
linestyle='dashed', label='D-Val-loss')
1052         #plt.plot(val_iterations_arr , track_MSE_val_loss , color='seagreen' ,
alpha=0.5, linestyle='dashed', label='MSE-Val-loss')

1054     plt.legend()
1055     plt.title("LOSS")
1056     plt.xlabel('Training Iteration (processed batches)')
1057     plt.ylabel('Loss')
1058     plt.ylim([low_y - np.abs(0.5*low_y) , high_y + np.abs(0.5*high_y)])
1059     plt.grid(True)
1060     if restart_training:
1061         plt.savefig(save_dir+ '/plots/DISCGEN LOSS_restart.png' )

```

```

1062     else :
1063         plt.savefig(save_dir+ '/plots /DISCGEN_LOSS.png' )
1064         plt.clf()
1065         plt.close(fig)
1066
1067         if (len(track_MSE_val_loss) > 0) and (len(val_iterations_arr) > 0):
1068             fig = plt.figure()
1069             plt.plot(iterations_arr , track_MSE_train_loss , color='seagreen' , alpha
1070 =0.5, label='MSE-Train-loss')
1071             plt.plot(val_iterations_arr , track_MSE_val_loss , color='navy' , alpha
1072 =0.5, label='MSE-Val-loss')
1073             high_MSE_train = np.percentile(track_MSE_train_loss , 95)
1074             low_MSE_train = np.percentile(track_MSE_train_loss , 5)
1075             high_MSE_val = np.percentile(track_MSE_val_loss , 95)
1076             low_MSE_val = np.percentile(track_MSE_val_loss , 5)
1077
1078             high_y = max([high_MSE_train , high_MSE_val])
1079             low_y = min([low_MSE_train , low_MSE_val])
1080
1081             plt.legend()
1082             plt.title("MSE")
1083             plt.xlabel('Training Iteration (processed batches)')
1084             plt.ylabel('Loss')
1085             plt.ylim([low_y - np.abs(0.5*low_y) , high_y + np.abs(0.5*high_y)])
1086             plt.grid(True)
1087             if restart_training:
1088                 plt.savefig(save_dir+ '/plots /MSE LOSS_restart.png' )
1089             else:
1090                 plt.savefig(save_dir+ '/plots /MSE LOSS.png' )
1091                 plt.clf()
1092                 plt.close(fig)
1093
1094         if (len(track_JACCARD_val_loss) > 0) and (len(val_iterations_arr) > 0):
1095             fig = plt.figure()
1096             plt.plot(iterations_arr , track_JACCARD_train_loss , color='seagreen' ,
1097 alpha=0.5, label='JACCARD-Train-loss')
1098             plt.plot(val_iterations_arr , track_JACCARD_val_loss , color='navy' , alpha
1099 =0.5, label='JACCARD-Val-loss')
1100             high_JACCARD_train = np.percentile(track_JACCARD_train_loss , 95)
1101             low_JACCARD_train = np.percentile(track_JACCARD_train_loss , 5)
1102             high_JACCARD_val = np.percentile(track_JACCARD_val_loss , 95)
1103             low_JACCARD_val = np.percentile(track_JACCARD_val_loss , 5)
1104
1105             high_y = max([high_JACCARD_train , high_JACCARD_val])
1106             low_y = min([low_JACCARD_train , low_JACCARD_val])
1107
1108             plt.legend()
1109             plt.title("JACCARD")
1110             plt.xlabel('Training Iteration (processed batches)')
1111             plt.ylabel('Loss')

```

```
1108     plt.ylim([low_y - np.abs(0.5*low_y), high_y + np.abs(0.5*high_y)])
1109     plt.grid(True)
1110     if restart_training:
1111         plt.savefig(save_dir+ '/plots/JACCARD_LOSS_restart.png' )
1112     else:
1113         plt.savefig(save_dir+ '/plots/JACCARD_LOSS.png' )
1114     plt.clf()
1115     plt.close(fig)
```

Listing C.1: Function to Plot Losses and Save them to a File

C.2 Functions to Generate 3D Objects from Grid-based data

```
1000 """
1001 code from: https://github.com/meetps/tf-3dgan/tree/master
1002 """
1003
1004 import sys
1005 import os
1006
1007 import scipy.ndimage as nd
1008 import scipy.io as io
1009 import numpy as np
1010 import matplotlib.pyplot as plt
1011 import skimage.measure as sk
1012
1013 from mpl_toolkits import mplot3d
1014
1015
1016 import trimesh
1017 from stl import mesh
1018
1019
1020 def getVF(path):
1021     raw_data = tuple(open(path, 'r'))
1022     header = raw_data[1].split()
1023     n_vertices = int(header[0])
1024     n_faces = int(header[1])
1025     vertices = np.asarray([map(float, raw_data[i+2].split()) for i in range(n_vertices)])
1026     faces = np.asarray([map(int, raw_data[i+2+n_vertices].split()) for i in range(n_faces)])
1027     return vertices, faces
1028
1029 def plotFromVF(vertices, faces, obj):
1030     input_vec = mesh.Mesh(np.zeros(faces.shape[0], dtype=mesh.Mesh.dtype))
1031     for i, f in enumerate(faces):
1032         for j in range(3):
1033             input_vec.vectors[i][j] = vertices[f[j],:]
1034     ...
1035
1036     figure = plt.figure()
1037     axes = mplot3d.Axes3D(figure)
1038     axes.add_collection3d(mplot3d.art3d.Poly3DCollection(input_vec.vectors))
1039     scale = input_vec.points.flatten('C')
1040     axes.auto_scale_xyz(scale, scale, scale)
1041     plt.show()
1042     ...
1043
1044     input_vec.save(obj+'.stl')
```

```

1044 def plotFromVoxels(voxels, i=None):
1045     #print(voxels.nonzero().shape)
1046     z,x,y = voxels.nonzero()
1047     fig = plt.figure()
1048     ax = fig.add_subplot(111, projection='3d')
1049     ax.scatter(x, y, z, zdir='z', c= 'black', marker='s')
1050     #ax.voxels(voxels, facecolors="black", edgecolor='k')
1051     ax.set_xlim(0, voxels.shape[0])
1052     ax.set_ylim(0, voxels.shape[1])
1053     ax.set_zlim(0, voxels.shape[2])
1054     if i is not None:
1055         ax.set_title(i)
1056     plt.show()
1057
1058 def getVFByMarchingCubes(voxels):
1059     v, f, normals, values = sk.marching_cubes(voxels,
1060                                                 level=0.01,
1061                                                 spacing=(1, 1, 1),
1062                                                 gradient_direction='descent',
1063                                                 step_size=1,
1064                                                 allow_degenerate=False,
1065                                                 method='lewiner', mask=None)
1066     return v, f
1067
1068 def plotMeshFromVoxels(voxels, obj='toilet'):
1069     v,f = getVFByMarchingCubes(voxels)
1070     plotFromVF(v,f,obj)
1071
1072 def plotVoxelVisdom(voxels, visdom, title):
1073     v, f = getVFByMarchingCubes(voxels)
1074     visdom.mesh(X=v, Y=f, opts=dict(opacity=0.5, title=title))
1075
1076 def plotFromVertices(vertices):
1077     figure = plt.figure()
1078     axes = mplot3d.Axes3D(figure)
1079     axes.scatter(vertices.T[0,:], vertices.T[1,:], vertices.T[2,:])
1080     plt.show()
1081
1082 def getVolumeFromOFF(path, sideLen=32):
1083     mesh = trimesh.load(path)
1084     volume = trimesh.voxel.Voxel(mesh, 0.5).raw
1085     (x, y, z) = map(float, volume.shape)
1086     volume = nd.zoom(volume.astype(float),
1087                      (sideLen/x, sideLen/y, sideLen/z),
1088                      order=1,
1089                      mode='nearest')
1090     volume[np.nonzero(volume)] = 1.0
1091     return volume.astype(np.bool)
1092

```

```
1094     def getVoxelFromMat(path, cube_len=64):
1095         voxels = io.loadmat(path)[ 'instance' ]
1096         voxels = np.pad(voxels,(1,1), 'constant', constant_values=(0,0))
1097         if cube_len != 32 and cube_len == 64:
1098             voxels = nd.zoom(voxels, (2,2,2), mode='constant', order=0)
1099         if cube_len != 32 and cube_len == 128:
1100             voxels = nd.zoom(voxels, (4,4,4), mode='constant', order=0)
1101         return voxels
```

Listing C.2: Functions to Generate 3D Objects from Grid-based data

C.3 Logger Class

```
1000 import sys  
  
1002 class Logger(object):  
1003     def __init__(self, filename="Default.log"):  
1004         self.terminal = sys.stdout  
1005         self.log = open(filename, "w")  
  
1006     def write(self, message):  
1007         self.terminal.write(message)  
1008         self.log.write(message)  
  
1009     def flush(self):  
1010         # this flush method is needed for python 3 compatibility.  
1011         # this handles the flush command by doing nothing.  
1012         # you might want to specify some extra behavior here.  
1013         pass
```

Listing C.3: Logger Class

C.4 *.npz to *.tfrecords Converter

```
1000 import os
os.environ[ 'TF_CPP_MIN_LOG_LEVEL' ] = '1'
1002 import tensorflow as tf
gpus = tf.config.list_physical_devices( 'GPU' )
tf.config.experimental.set_memory_growth(gpus[0], True)

1006 #print(tf.config.list_physical_devices('GPU'))

1008 import numpy as np
from glob import glob
1010 import time
from multiprocessing import Pool, Manager
1012 import gc

1014 def _bytes_feature(value):
    """Returns a bytes_list from a string / byte."""
    if isinstance(value, type(tf.constant(0))):
        value = value.numpy() # BytesList won't unpack a string from an EagerTensor.
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))

1020 def _float_feature(value):
    """Returns a float_list from a float / double."""
    return tf.train.Feature(float_list=tf.train.FloatList(value=value))

1024 def _int64_feature(value):
    """Returns an int64_list from a bool / enum / int / uint."""
    return tf.train.Feature(int64_list=tf.train.Int64List(value=value))

1028 def max_pool_dataset(object4d: np.ndarray, max_pooling_size: int) -> tf.Tensor:
    tf.keras.backend.clear_session()
    # Convert to tensorflow tensor for MaxPooling 3D Operation
    tf_object4d = tf.convert_to_tensor(object4d, dtype="half")
    # Expands last dim to fit tensorflow MaxPooling 3D Operation which expects
    Channels last (channels=1),
    # since we are only have binary data at each position 1 or 0, like in a black/
    white image)
    expanded_object4d = tf.expand_dims(tf_object4d, axis=-1)
    # Pooling operation
    # NOTE: DIM 0: Time axis; Tensorflow Pooling Layer Interprets DIM 0 generally
    as Batch Size,
    # in our case this is the time axis (here 128 time steps), therefore a full
    batch is Pooling applied over all time instants
    pooled_object4d = tf.nn.max_pool3d(expanded_object4d, ksize=max_pooling_size,
    strides=max_pooling_size, padding="VALID")
    # squeeze (delete) last dim (channel) and add dim at beginning => Shape=(1,
    time, gridx, gridy, gridz)
    pooled_object4d = tf.expand_dims(tf.squeeze(pooled_object4d), axis=0)
```

```

#pooled_object4d = tf.identity(pooled_object4d)
1042 return tf.cast(pooled_object4d, tf.uint8).numpy()

1044 def read_and_serialize(dataset: str, max_pooling_size: int) -> list:

1046     # tbegin = time.time()
######
1048     #print(f"\nLoading and Converting Dataset: {os.path.basename(dataset)}")
1049     # t2 = time.time()
1050     np_dataset = np.load(dataset)

1052     object4d = ((np_dataset["data"]-1)*-1).astype("uint8")
1053     labelBetti = np_dataset["bettiNumbers"].astype("uint8")
1054     labelObjs = np_dataset["objects"][:,1].astype("uint8")

1056     # Max. Pooling of Objects to Reduce Size and Serialize Objects for tfrecords
1057     # file
1058     if max_pooling_size > 1:
1059         object4d = max_pool_dataset(object4d, max_pooling_size).ravel().tobytes()
1060     else:
1061         object4d = object4d.ravel().tobytes()

1062     labelBetti = labelBetti.ravel().tobytes()
1063     labelObjs = labelObjs.ravel().tobytes()
1064     # t3 = time.time()
1065     # print(f"\n get objects and maxpool3d - time = {t3-t2}")

1068     # t2 = time.time()
1069     object4d_ex = tf.train.Example(features=tf.train.Features(feature={
1070         'object4d': _bytes_feature(object4d),
1071         'labelBetti': _bytes_feature(labelBetti),
1072         'labelObjs': _bytes_feature(labelObjs)}))
1073     # t3 = time.time()
1074     # print(f"\n convert object4d to feature - time = {t3-t2}")

1076     ######
1077     # WRITE TO TFrecords file
1078     tfrecords_filename = os.path.splitext(dataset)[0] + f"_{{int(128/max_pooling_size)}}.tfrecords"
1079     # Define tfrecords writer
1080     tfrecords_settings = tf.io.TFRecordOptions(compression_type="GZIP")
1081     writer = tf.io.TFRecordWriter(tfrecords_filename, options=tfrecords_settings)

1082     #t2 = time.time()
1083     with writer as file_writer:
1084         file_writer.write(object4d_ex.SerializeToString())
1085     #t3 = time.time()
1086     #print(f"\n Write to tf records file - time elapsed = {t3-t2}")
1087

```

```

#####
# tend = time.time()
# print(f" total time per object - time = {tend-tbegin}")

if __name__ == '__main__':
#####
# SETTINGS
folder_name="C:/Users/mgold/OneDrive/Dokumente/_FH/Master_Thesis/data/4
D_128_32_topological_dataset_0-31"
datasets = glob(folder_name+"/*.npz")
max_pooling_size = 4

# Split datasets into Chunks Size == MP Pools
n_processes = 8
datasets_chunk = [datasets[i * n_processes:(i + 1) * n_processes] for i in range
((len(datasets) + n_processes - 1) // n_processes )]

manager = Manager()
objects_list = manager.list()

total_chunks = len(datasets_chunk)

convert_tbegin = time.time()
for n_chunk, chunk in enumerate(datasets_chunk):
    iterable_args = []
    for n in range(len(chunk)):
        tup = (chunk[n], max_pooling_size)
        iterable_args.append(tup)

    pool_tbegin = time.time()
    with Pool(n_processes) as p:
        p.starmap(func=read_and_serialize, iterable=iterable_args)

    pool_tend = time.time()
    print(f" Finished Chunk no. {n_chunk+1}/{total_chunks} with Pool of {n_processes} Processes - time elapsed = {pool_tend-pool_tbegin}")

    gc.collect()

convert_tend = time.time()
print(f" Finished Conversion to TFrecords format - time elapsed = {convert_tend-
convert_tbegin}")

```

Listing C.4: *.npz to *.tfrecords Converter

C.5 Split Data into Train-, Validation- and Testsets

```
1000 import os
1001 import shutil
1002 import numpy as np
1003 import tensorflow as tf
1004
1005 ##### INPUTS #####
1006 # DATASET_LOCATION      = "Z:/Master_Thesis/data/4D_128_ALL_16/*"    # 4
1007 # D_128_32_topological_dataset_0-31_16_label8only OR 4D_128_ALL_16/*
1008 DATASET_LOCATION      = "Z:/Master_Thesis/data/4D_128_ALL_16/*"    # 4
1009 TRAIN_VAL_TEST_RATIO   = 0.8                                         # Default:
1010 # 0.8 - Fraction of objects of full dataset to be used for training , rest is
1011 # equaly split into Validation and Test Set
1012 TRAINING_DATASET_FOLDER = "00_4D_training_datasets"                  # FOLDER TO
1013 # SAVE THE TRAINING FILES TO
1014 VALIDATION_DATASET_FOLDER = "01_4D_validation_datasets"                # FOLDER TO
1015 # SAVE THE VALIDATION FILES TO
1016 TEST_DATASET_FOLDER     = "02_4D_test_datasets"                      # FOLDER TO
1017 # SAVE THE TEST FILES TO
1018
1019 ##### GET DATA/OBJECTS #####
1020 #read all files from folder
1021 FILENAMES = tf.io.gfile.glob(DATASET_LOCATION+"/*.tfrecords")
1022 FILENAMES_SHUFFLED = np.random.shuffle(FILENAMES)
1023
1024 # Split into Train, Validation and Test Sets
1025 split_int_train = int(TRAIN_VAL_TEST_RATIO * len(FILENAMES))
1026 split_int_valid_test = split_int_train + int(np.round((1-TRAIN_VAL_TEST_RATIO) / 2 *
1027             len(FILENAMES)))
1028
1029 TRAINING_FILENOES = FILENAMES[:split_int_train]
1030 VALIDATION_FILENOES = FILENAMES[split_int_train:split_int_valid_test]
1031 TEST_FILENOES = FILENAMES[split_int_valid_test:]
1032
1033 training_set_size = len(TRAINING_FILENOES)
1034 validation_set_size = len(VALIDATION_FILENOES)
1035 test_set_size = len(TEST_FILENOES)
1036 print("Total Number of TFRecord Files to Train : ", training_set_size)
1037 print("Total Number of TFRecord Files for Validation: ", validation_set_size)
1038 print("Total Number of TFRecord Files for Testing: ", test_set_size)
1039
1040
1041 # Generate Folder with Training Files
1042 if not os.path.exists(TRAINING_DATASET_FOLDER):
1043     os.makedirs(TRAINING_DATASET_FOLDER)
1044 else:
```

```

1040     shutil.rmtree(TRAINING_DATASET_FOLDER)
1041     os.makedirs(TRAINING_DATASET_FOLDER)

1042     for file_name in TRAINING_Filenames:
1043         if os.path.isfile(file_name):
1044             shutil.copy(file_name, TRAINING_DATASET_FOLDER)

1046

1048 # Generate Folder with Validation Files
1049 if not os.path.exists(VALIDATION_DATASET_FOLDER):
1050     os.makedirs(VALIDATION_DATASET_FOLDER)
1051 else:
1052     shutil.rmtree(VALIDATION_DATASET_FOLDER)
1053     os.makedirs(VALIDATION_DATASET_FOLDER)

1054     for file_name in VALIDATION_Filenames:
1055         if os.path.isfile(file_name):
1056             shutil.copy(file_name, VALIDATION_DATASET_FOLDER)

1058

1060 # Generate Folder with Test Files
1061 if not os.path.exists(TEST_DATASET_FOLDER):
1062     os.makedirs(TEST_DATASET_FOLDER)
1063 else:
1064     shutil.rmtree(TEST_DATASET_FOLDER)
1065     os.makedirs(TEST_DATASET_FOLDER)

1066     for file_name in TEST_Filenames:
1067         if os.path.isfile(file_name):
1068             shutil.copy(file_name, TEST_DATASET_FOLDER)

```

Listing C.5: Split Data into Train-, Validation- and Testsets