FH University of Applied Sciences
TECHNIKUM WIEN

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 produced 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

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.

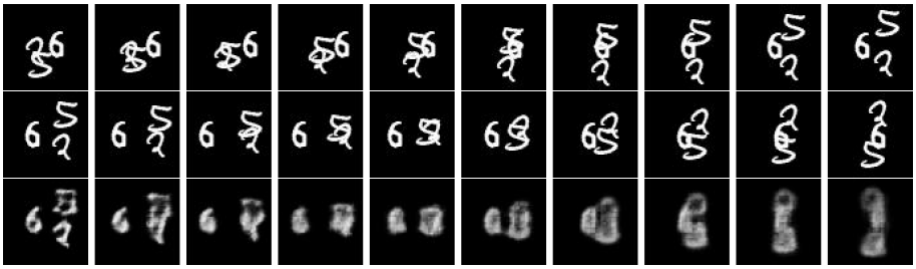

Figure 1.2: 10 next frame predictions with ConvLSTM; top: input, middle: ground truth, bottom: prediction. Source: [26]

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

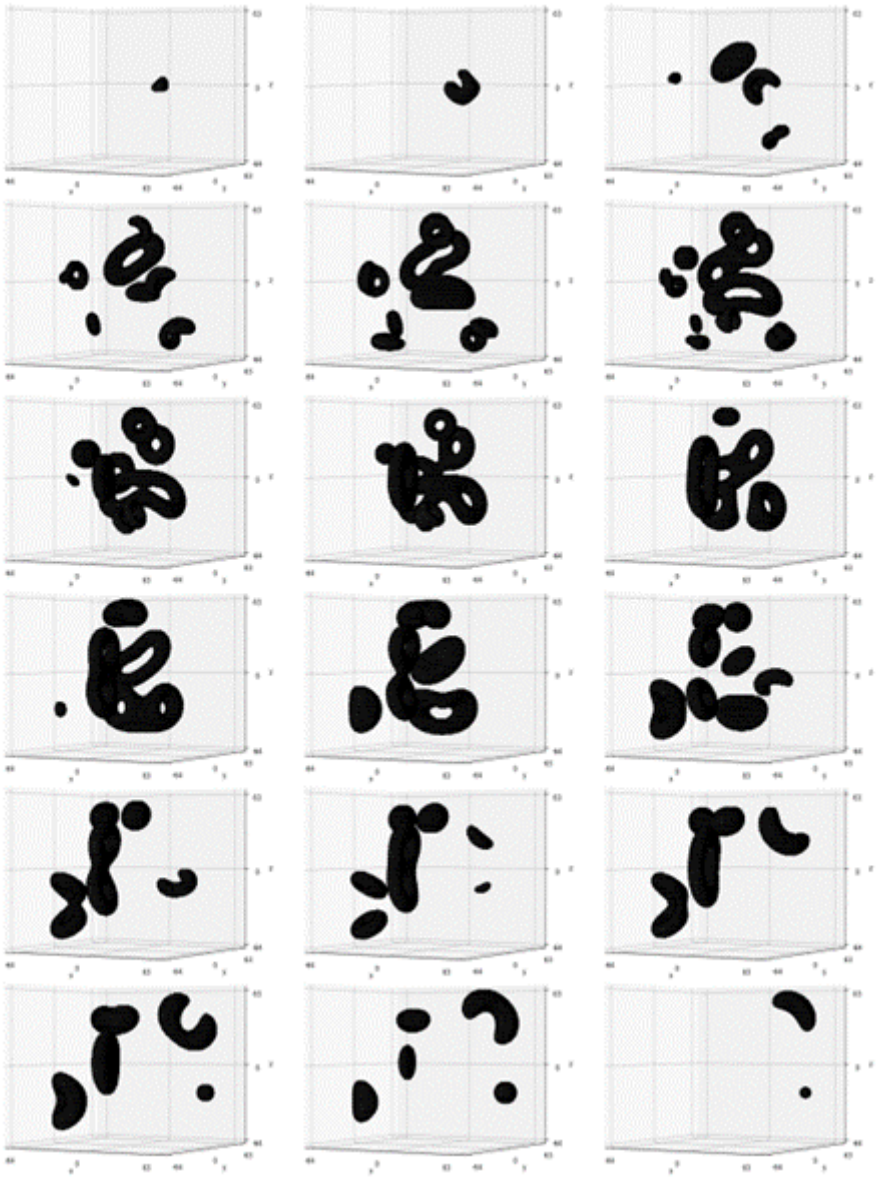

Figure 2.1: Cavities within a 128x128x128x128 sample, 18 equally-spaced 3D slices along the time-axis from left to right and top to bottom. Source: [11]

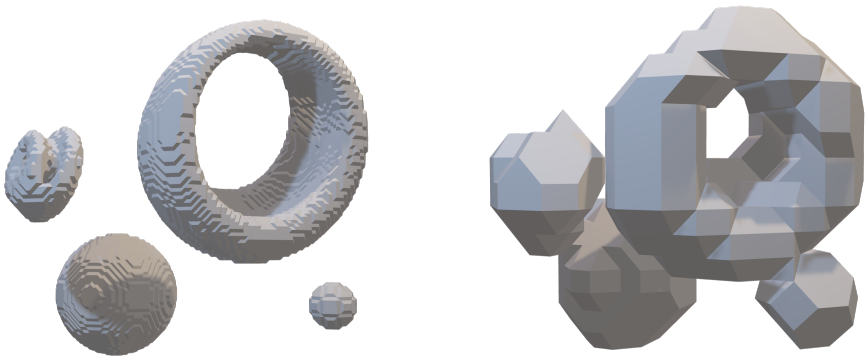

Figure 2.2: Resolution comparison of a random sample at a random time-point. Left: Original Size of 128x128x128; Right: Max-Pooled Size of 16x16x16

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),

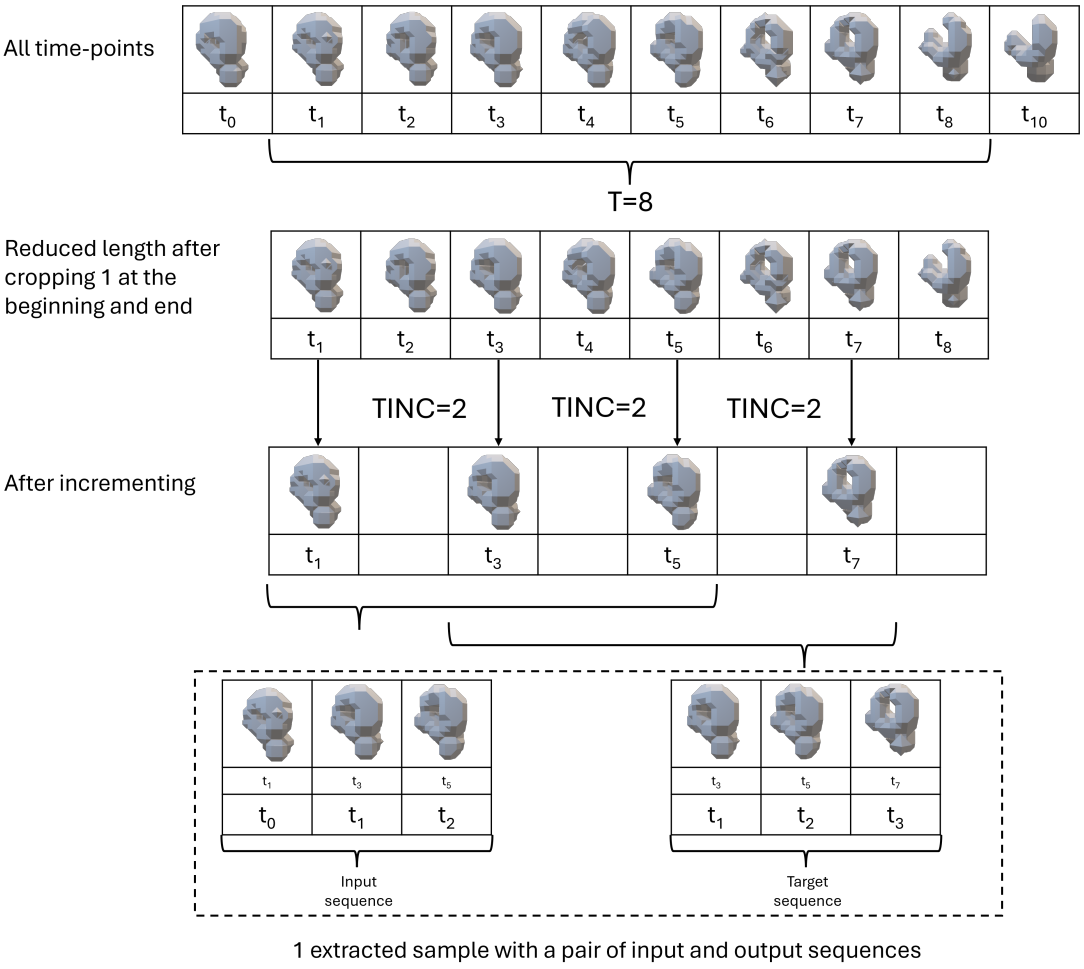

Figure 2.3: Illustration of how parameters T and TINC influence the sample extraction for the ConvLSTM models.

which is then trained on the next time instant t_1 (target). In this case the data format is 2x16x16x16, which consist of the 1x16x16x16 input object and 1x16x16x16 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 4x16x16x16, which consist of the 3x16x16x16 past input objects and 1x16x16x16 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 16x16x16x1 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.

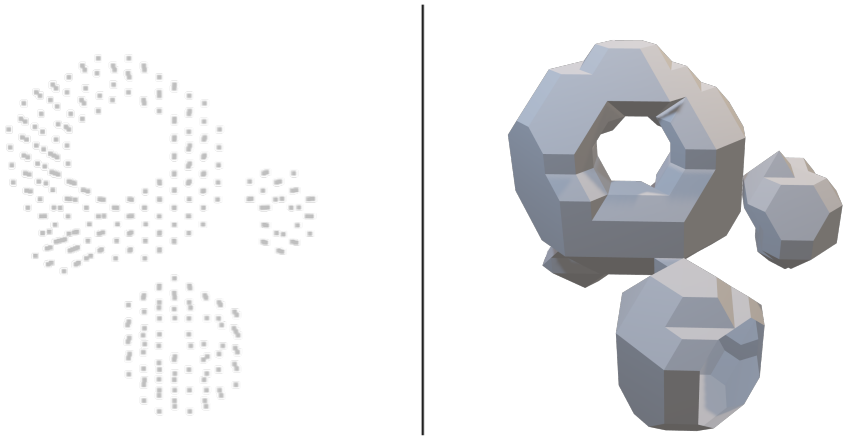

Figure 3.1: Left: Point cloud of an object; Right: Reconstructed object from points with Marching Cubes algorithm.

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.

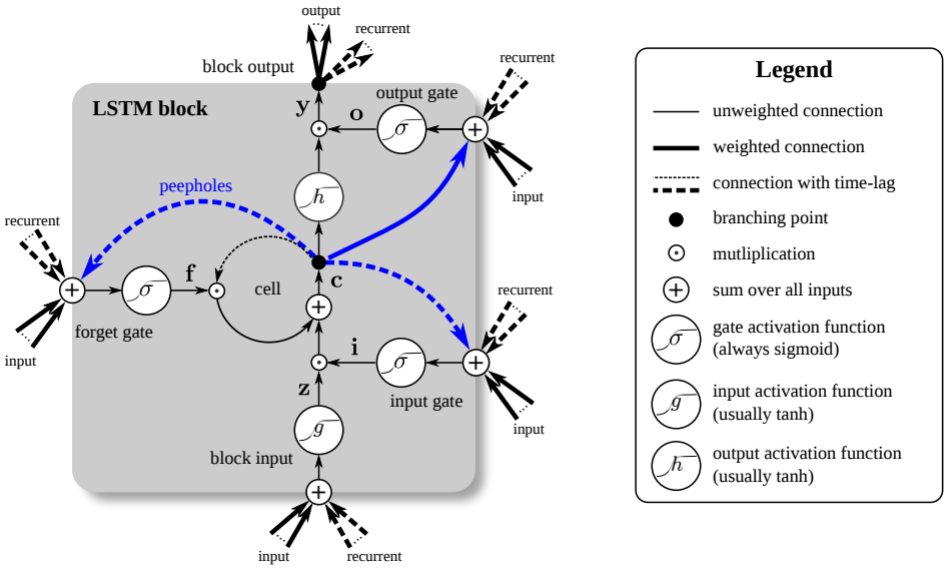

Figure 4.1: LSTM block. Source: [9]

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.

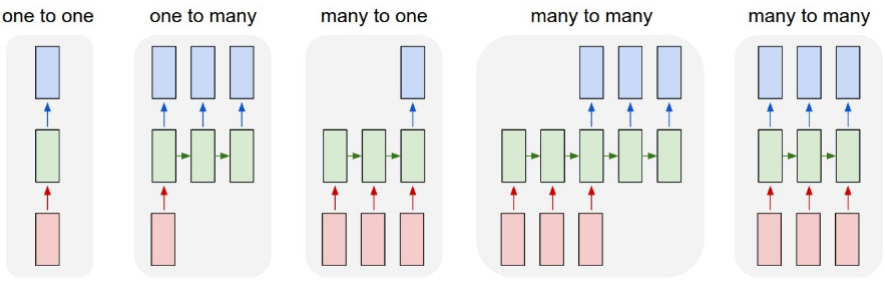

Figure 4.2: LSTM modes. Source: [18]

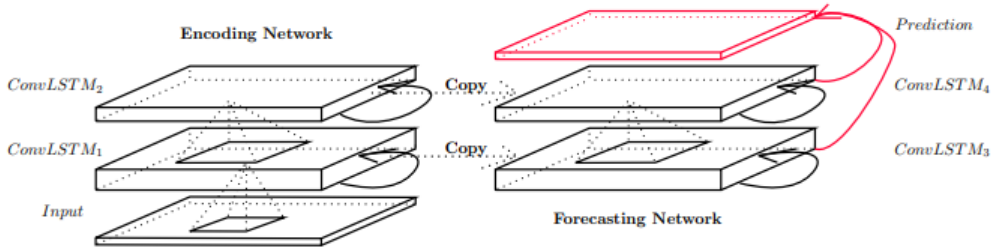

Figure 4.3: Illustration of the ConvLSTM model. Source: [26]

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.

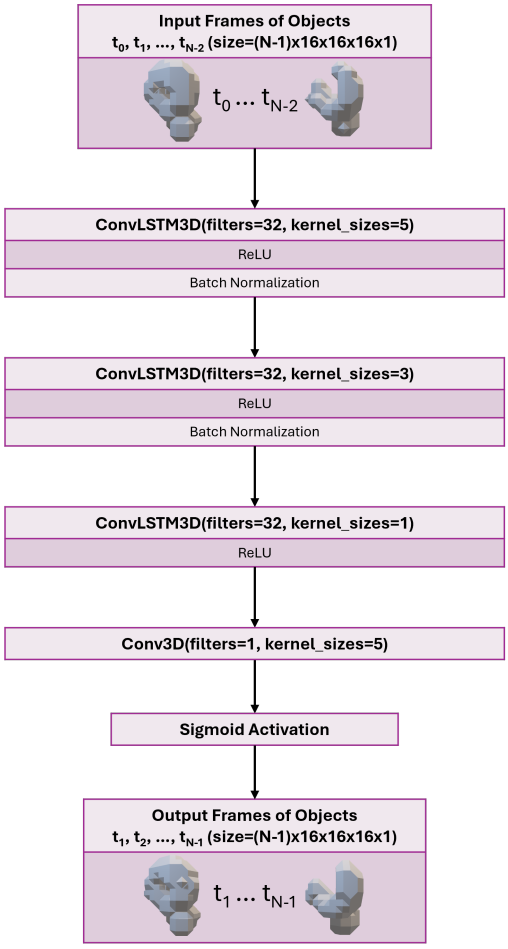

Figure 4.4: ConvLSTM network.

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

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

$\boldsymbol{x}_{t_1}$ and $\tilde{\boldsymbol{x}}_{t_1}$ 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 one convolution is used, see figure 5.1 (weight = convolutional operation).

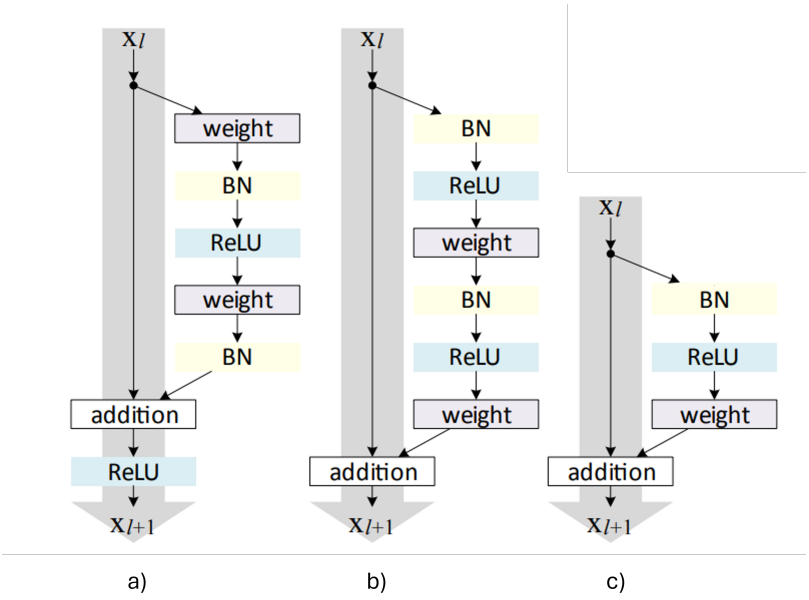

Figure 5.1: a) Residual Unit; b) Pre-Residual Unit; c) Pre-Residual Unit used in the GAN-based models in this work. BN=Batch Normalization; ReLU=Rectified Linear Unit Activation; weight=weights of the layer, e.g. weights of the convolutional layer. Source: [12]

- 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.

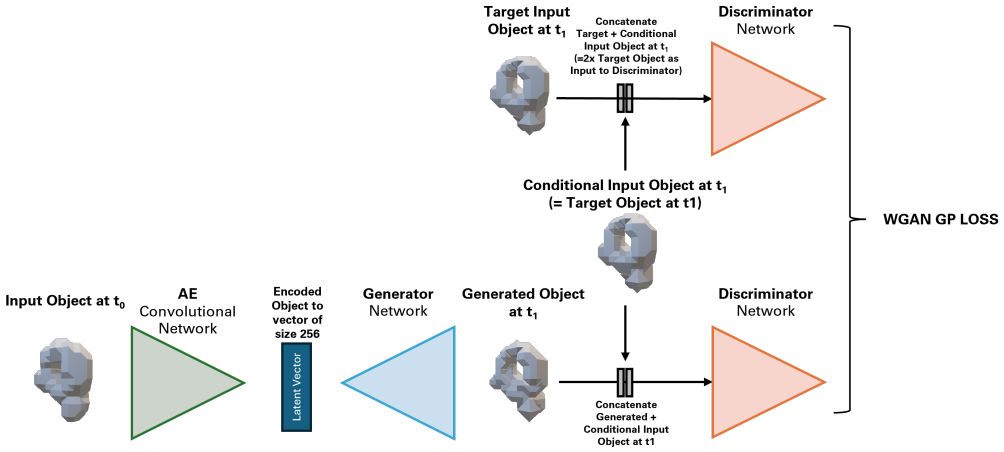

Figure 5.2: Single Frame Input Model.

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 alse 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.

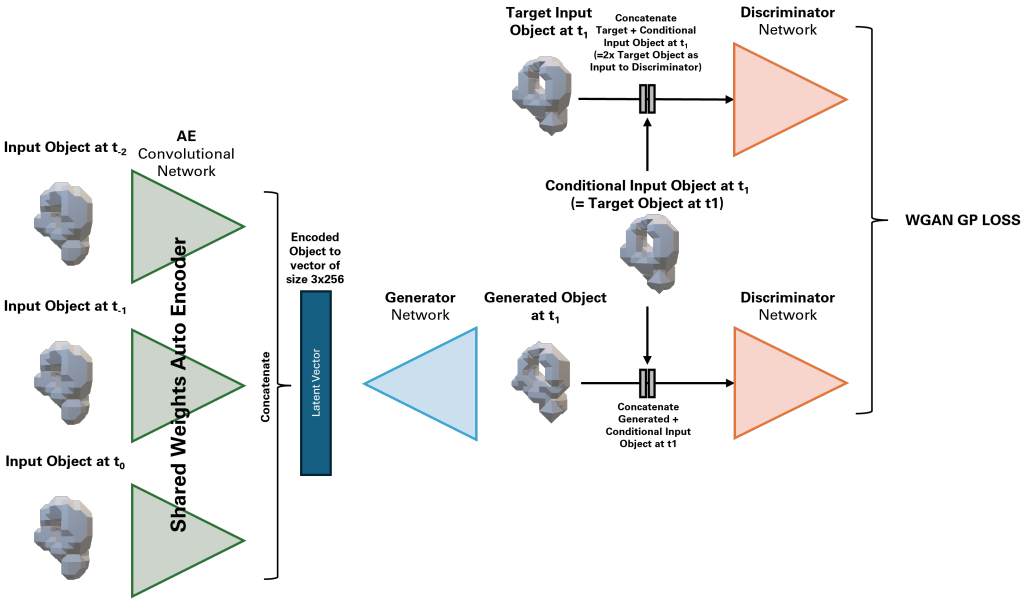

Figure 5.3: Multiple Frame Input Model.

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 to 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 take a single object of size 16x16x16x1 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.

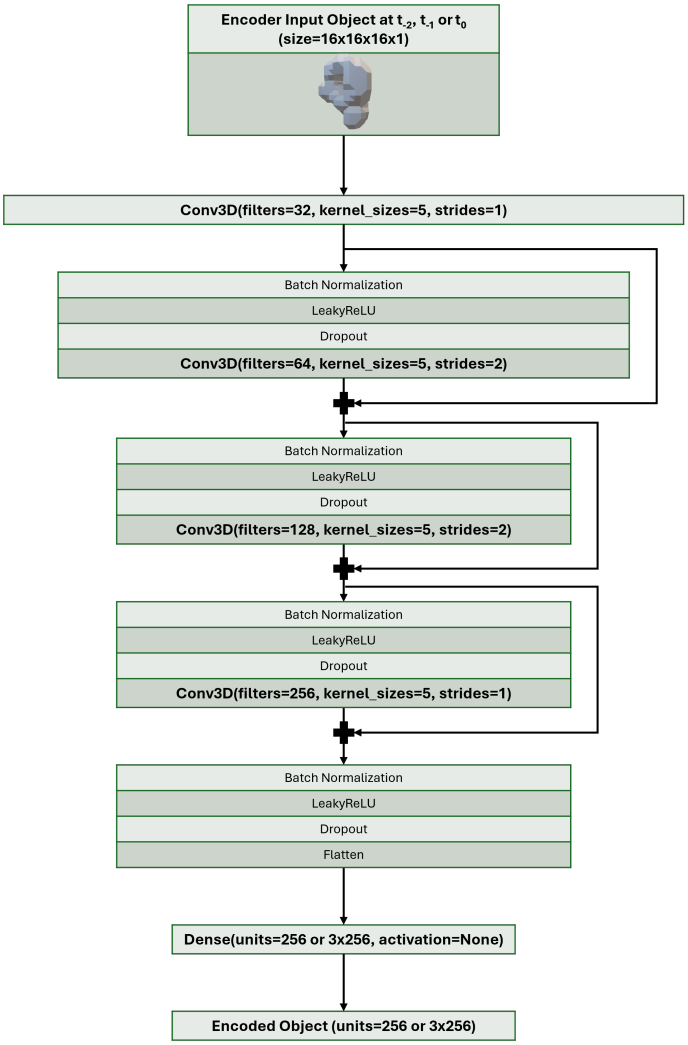

Figure 5.4: Encoder network.

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 16x16x16x1 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.

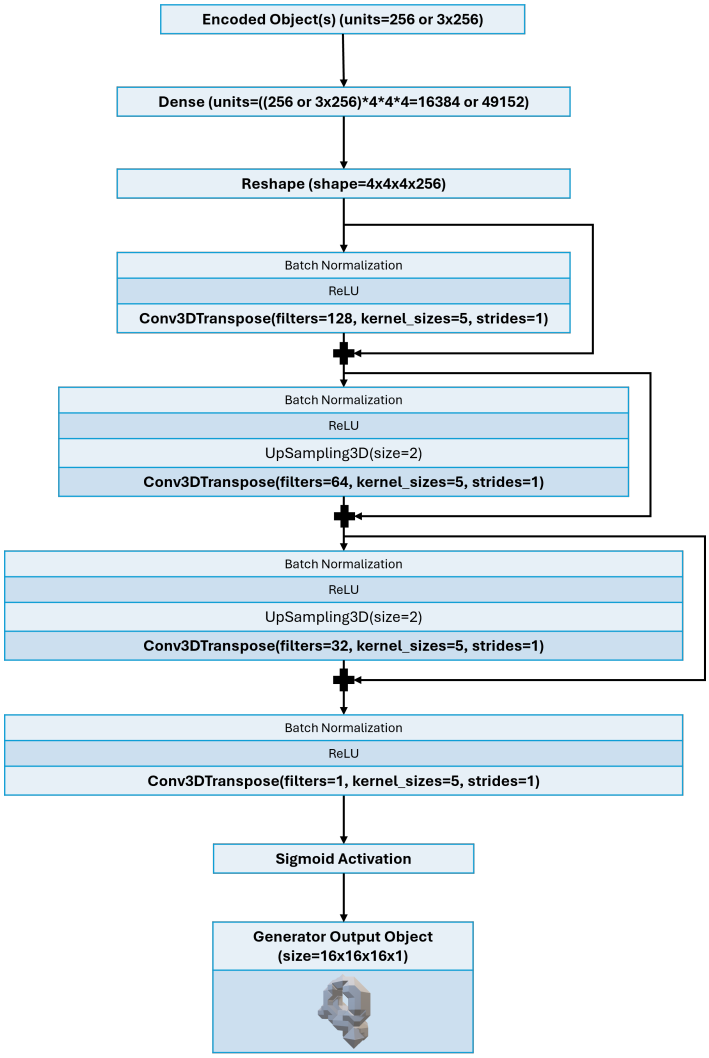

Figure 5.5: Generator network.

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.

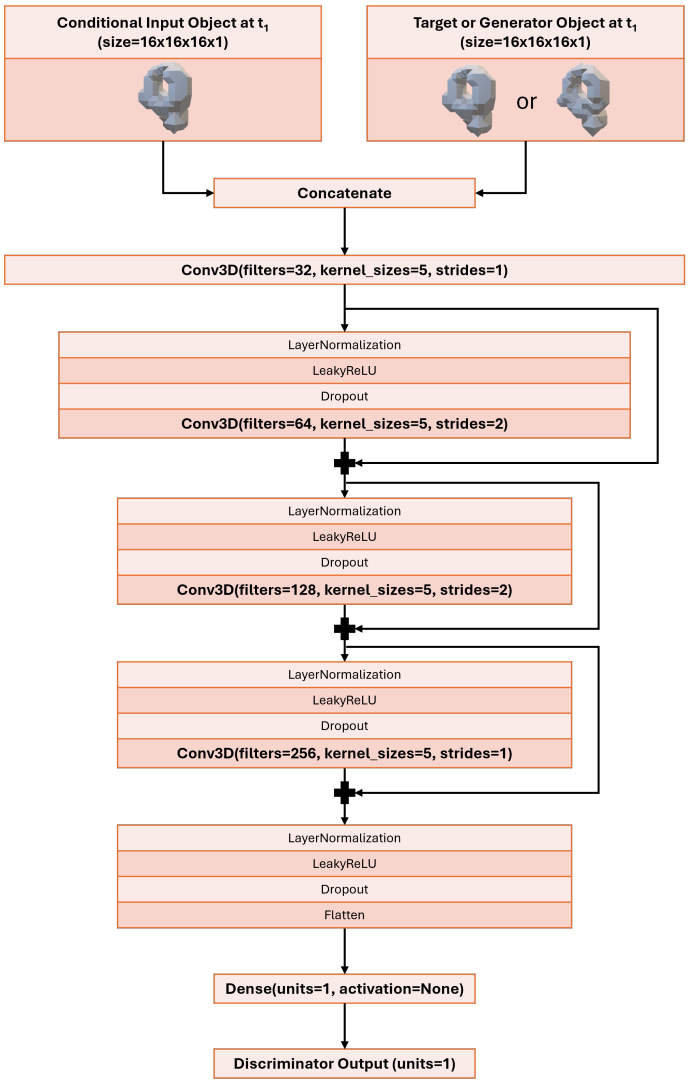

Figure 5.6: Discriminator network.

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 though of a score) looks as follows, including the epsilon penalty term from [3]

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

where $\mathbb{P}_r$ is the distribution of the data, $\mathbb{P}_g$ is the distribution of the generated data implicitly defined by $\tilde{\boldsymbol{x}} = G(\boldsymbol{z})$, $\tilde{\boldsymbol{x}} \sim p(\tilde{\boldsymbol{x}})$, λ is the gradient penalty term, $\|...\|_2$ 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, $\boldsymbol{z}$ isn't drawn from a distribution anymore, like in standard GANs, but is now

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

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

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

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 $\boldsymbol{x}_{t_1}$ and $\tilde{\boldsymbol{x}}_{t_1}$, 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.

$$\mathrm{MSE}(\boldsymbol{x}_{t_1}, \tilde{\boldsymbol{x}}_{t_1}) = \frac{1}{n} \sum_{i=1}^{n} \left(x_{i,t_1} - \tilde{x}_{i,t_1}\right)^2 . \tag{5}$$

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

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

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

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

A Jaccard Distance of zero would mean a perfect match between the next frame 3D tensors of target object and the generated object $\boldsymbol{x}_{t_1}$ and $\tilde{\boldsymbol{x}}_{t_1}$, 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 $\boldsymbol{x}_{t_1}$

$$\mathrm{MSE}(\boldsymbol{x}_{t_1}, \tilde{\boldsymbol{x}}_{t_1}) \tag{8}$$

$$d_J(\boldsymbol{x}_{t_1}, \tilde{\boldsymbol{x}}_{t_1}) \tag{9}$$

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 $\boldsymbol{x}_{t_0}$

$$\mathrm{MSE}(\boldsymbol{x}_{t_0}, \tilde{\boldsymbol{x}}_{t_1}) \tag{10}$$

$$d_J(\boldsymbol{x}_{t_0}, \tilde{\boldsymbol{x}}_{t_1}) \tag{11}$$

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 $\tilde{\boldsymbol{x}}_{t_0} = \boldsymbol{x}_{t_0}$. This last frame is now evaluated against the target frame (next frame):

- Null Model

$$\mathrm{MSE}(\boldsymbol{x}_{t_1}, \boldsymbol{x}_{t_0}) \tag{12}$$

$$d_J(\boldsymbol{x}_{t_1}, \boldsymbol{x}_{t_0}) \tag{13}$$

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.

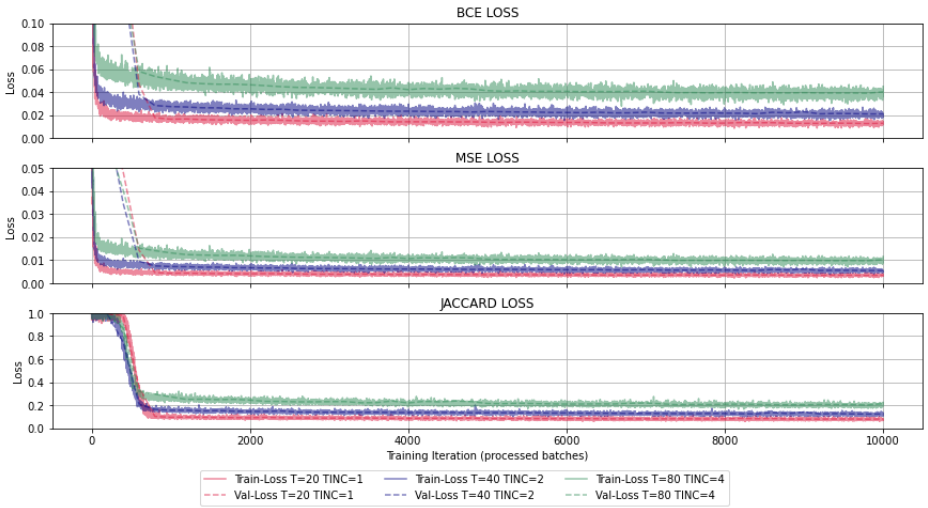

Figure 7.1: Training and validation loss of the BCE, MSE and Jaccard Distance.

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 $\boldsymbol{x}_{t_0}...\boldsymbol{x}_{t_{19}}$ to predict the next sequence $\boldsymbol{x}_{t_1}...\boldsymbol{x}_{t_{20}}$, 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:

$$\frac{1}{n}\sum_{i=1}^{n}\mathbf{MSE}(\boldsymbol{x}_{t_1}...\boldsymbol{x}_{t_{20}},\tilde{\boldsymbol{x}}_{t_1}...\tilde{\boldsymbol{x}}_{t_{20}})_i \qquad and \qquad \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{d_J}(\boldsymbol{x}_{t_1}...\boldsymbol{x}_{t_{20}},\tilde{\boldsymbol{x}}_{t_1}...\tilde{\boldsymbol{x}}_{t_{20}})_i \tag{14}$$

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 $\boldsymbol{x}_{t_0}...\boldsymbol{x}_{t_{19}}$

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

- Null Model

$$\frac{1}{n}\sum_{i=1}^{n}\mathbf{MSE}(\boldsymbol{x}_{t_1}...\boldsymbol{x}_{t_{20}},\boldsymbol{x}_{t_0}...\boldsymbol{x}_{t_{19}})_i \qquad and \qquad \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{d_J}(\boldsymbol{x}_{t_1}...\boldsymbol{x}_{t_{20}},\boldsymbol{x}_{t_0}...\boldsymbol{x}_{t_{19}})_i \tag{16}$$

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.

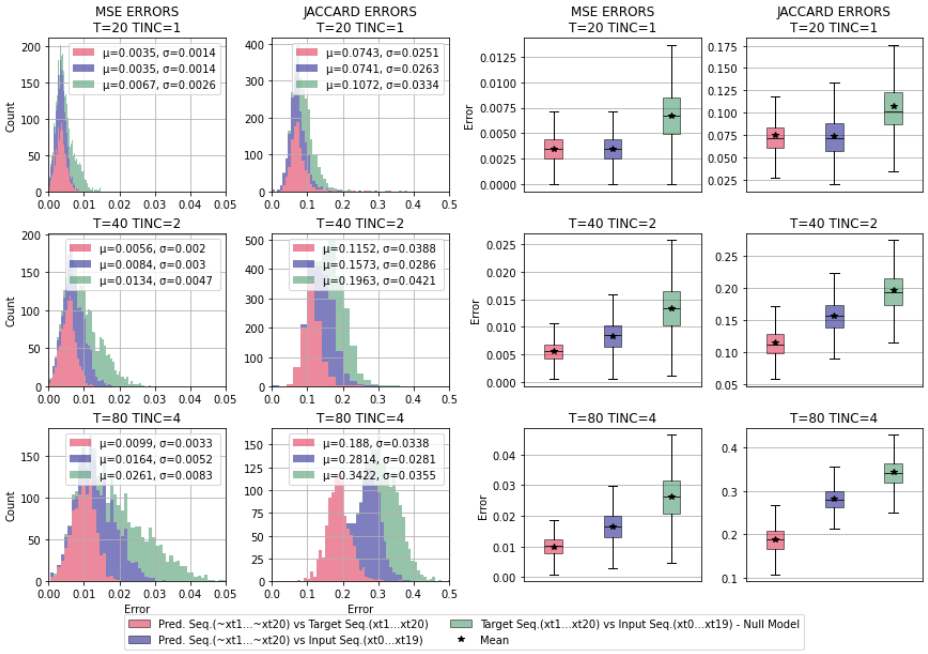

Figure 7.2: ConvLSTM Model - Histogram and boxplot of the MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T=20-INC=1, T=40-INC=2 and T=80-INC=4.

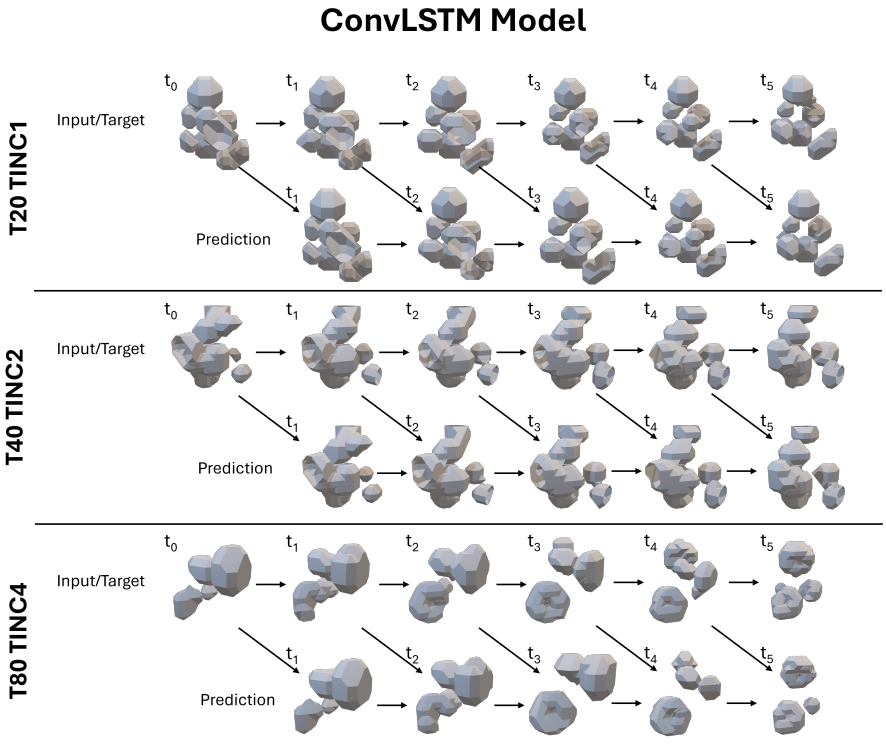

Figure 7.3: ConvLSTM Model - Exemplary generated predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

7.3 Long-Term Prediction Quality

The first 5 frames are used as input sequence $\boldsymbol{x}_{t_0}...\boldsymbol{x}_{t_5}$ to predict the one-step ahead sequence $\tilde{\boldsymbol{x}}_{t_1}...\tilde{\boldsymbol{x}}_{t_6}$. From this prediction the last frame $\tilde{\boldsymbol{x}}_{t_6}$ is concatenated to the former input frames to predict the next frame with the newly updated input sequence $\boldsymbol{x}_{t_1}...\boldsymbol{x}_{t_5}, \tilde{\boldsymbol{x}}_{t_6}$ 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.

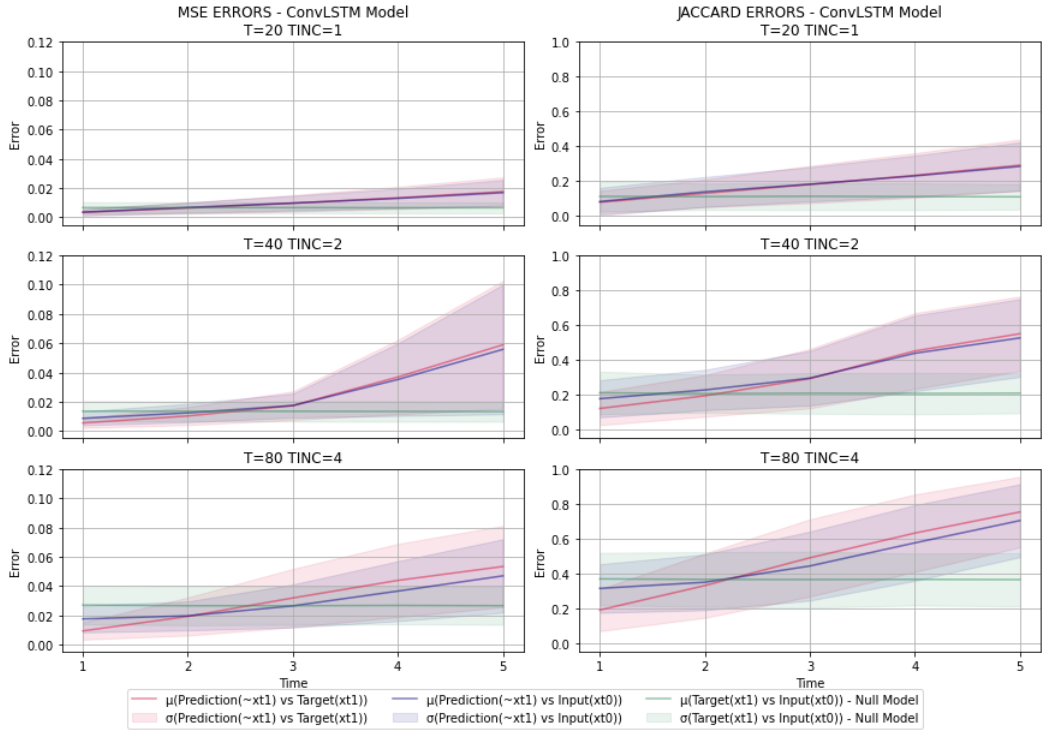

Figure 7.4: ConvLSTM Model - Long-term MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T=20-INC=1, T=40-INC=2 and T=80-INC=4.

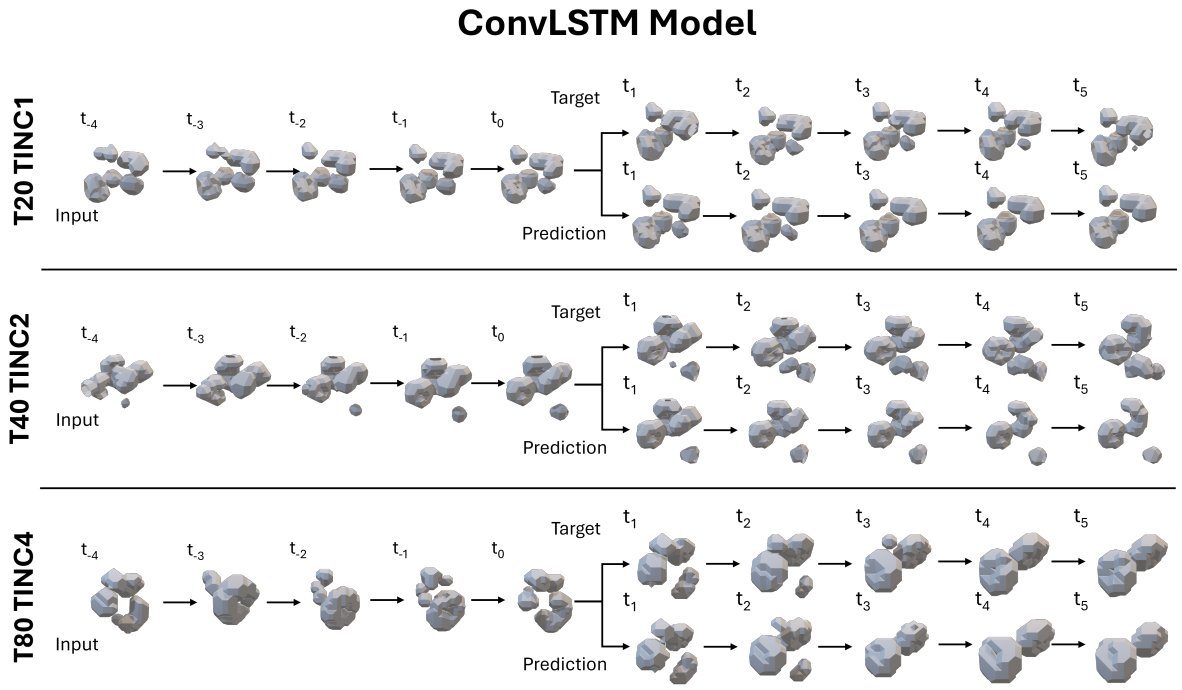

Figure 7.5: ConvLSTM Model - Exemplary generated long-term predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

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 noisey 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<Error<0.025; Jaccard: 0.2<Error<0.4) than the multiple frame input model (MSE: 0.01<Error<0.015; Jaccard: 0.18<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.

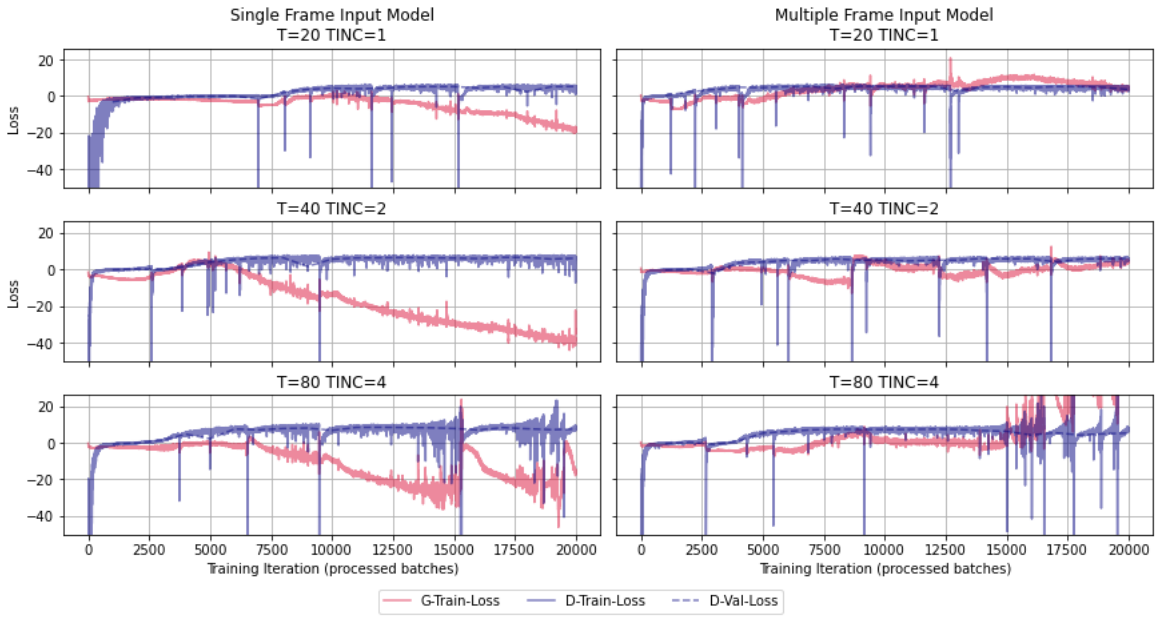

Figure 8.1: Training and validation loss of the Generator and Discriminator.

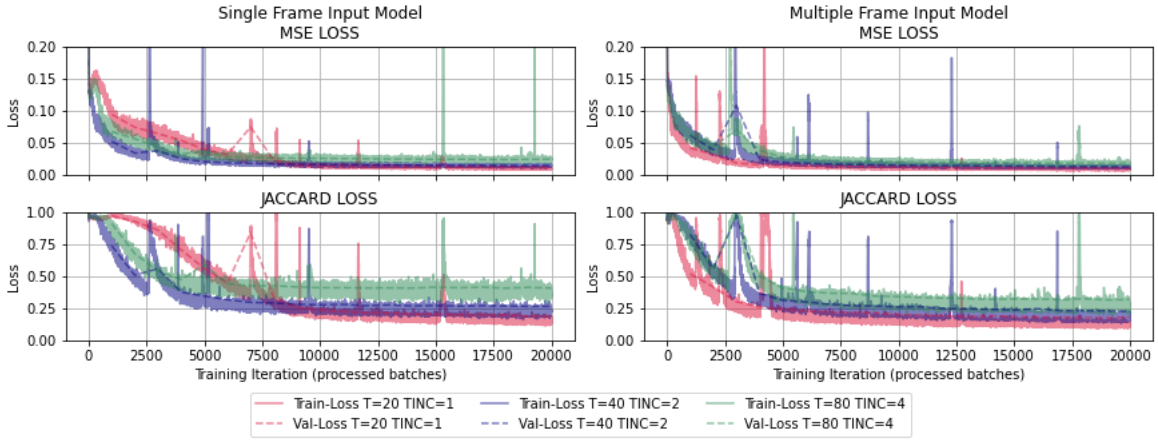

Figure 8.2: Training and validation loss of the MSE and Jaccard Distance.

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.

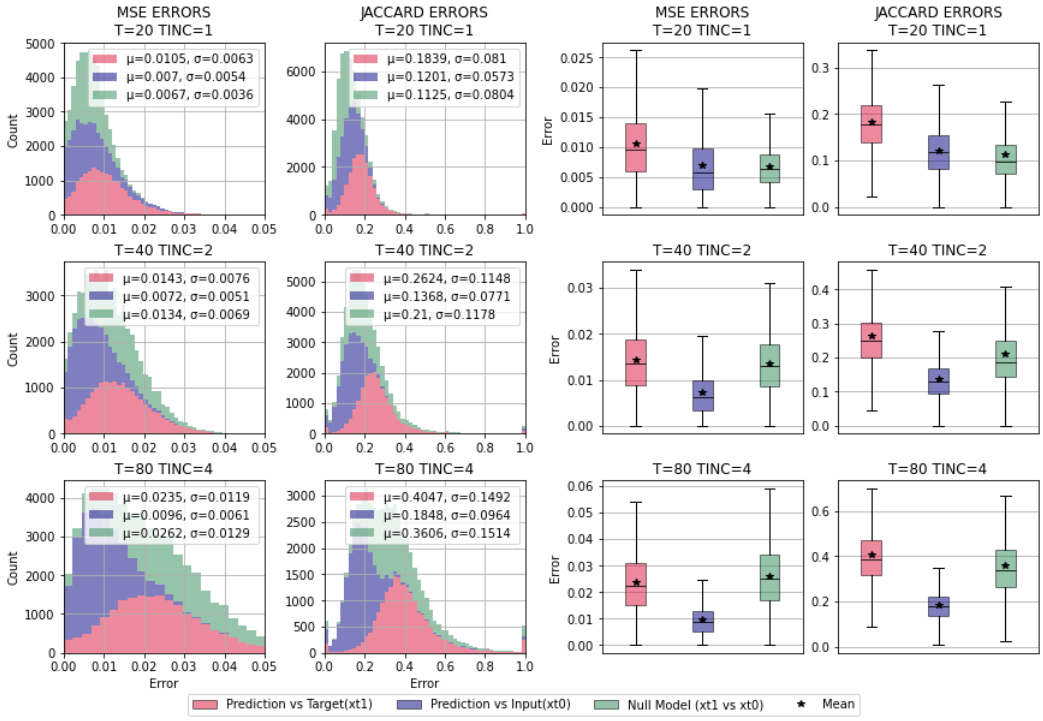

Figure 8.3: Single Frame Input Model - Histogram and Boxplot of the MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

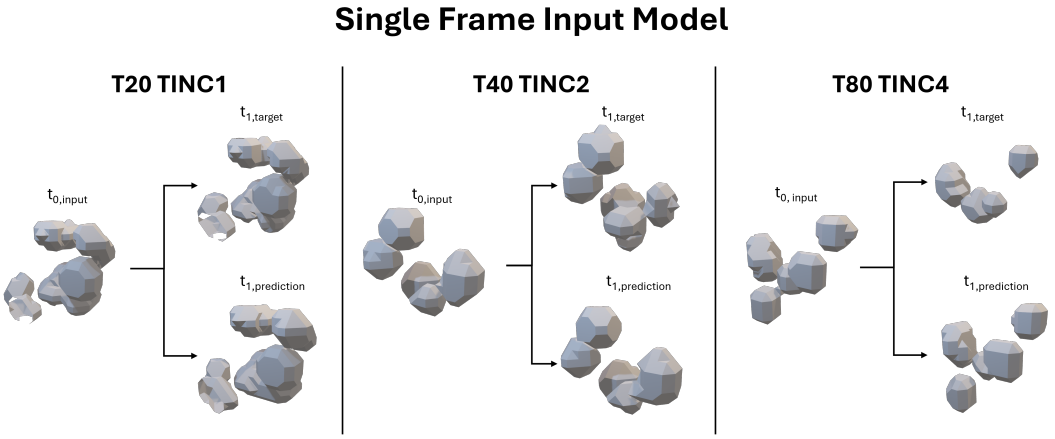

Figure 8.4: Single Frame Input Model - Exemplary generated predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

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.

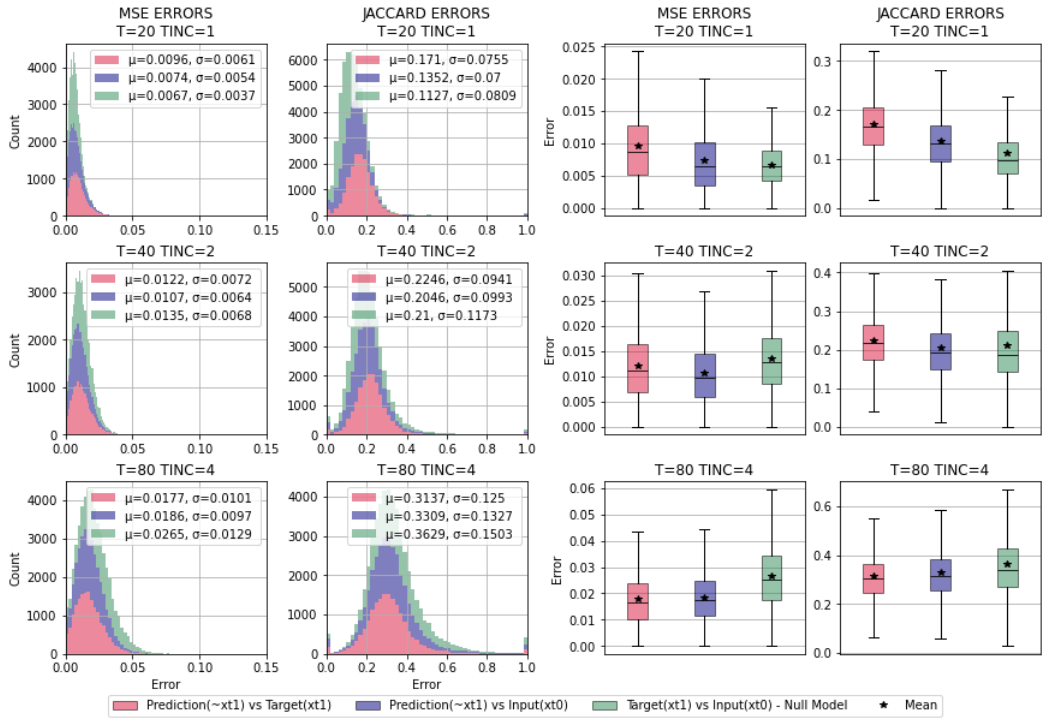

Figure 8.5: Multiple Frame Input Model - Histogram and Boxplot of the MSE and Jaccard Distance between prediction-target, prediction-input and the null model for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

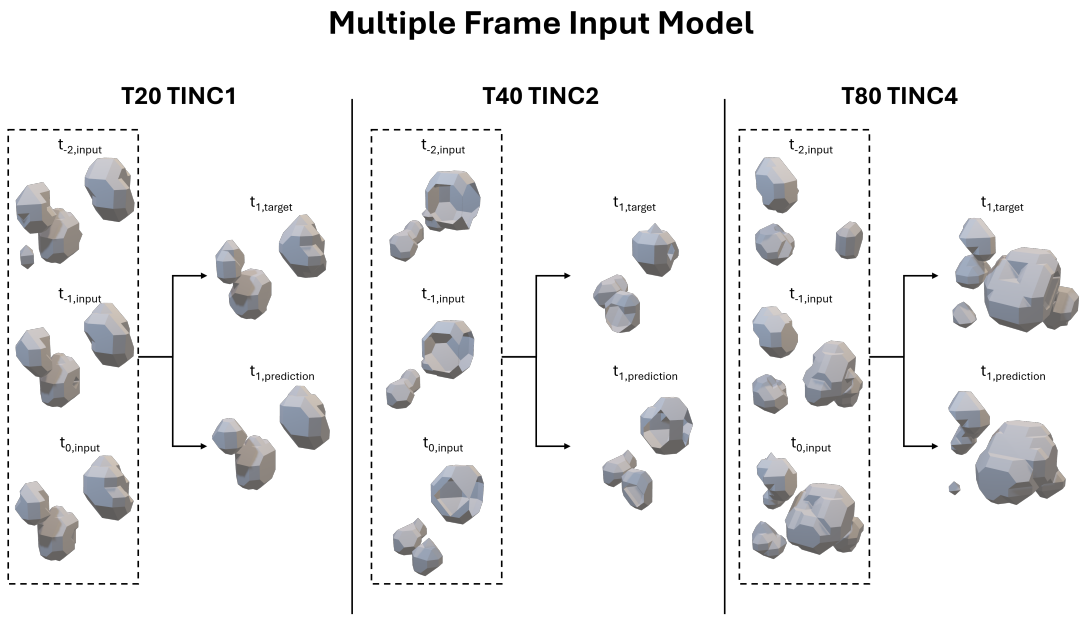

Figure 8.6: Multiple Frame Input Model - Exemplary generated predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

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 $\boldsymbol{x}_{t_0}$ to predict $\tilde{\boldsymbol{x}}_{t_1}$, followed by a loop subsequently setting $\tilde{\boldsymbol{x}}_{t_1} = \tilde{\boldsymbol{x}}_{t_0} = \boldsymbol{x}_{t_0}$ 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.

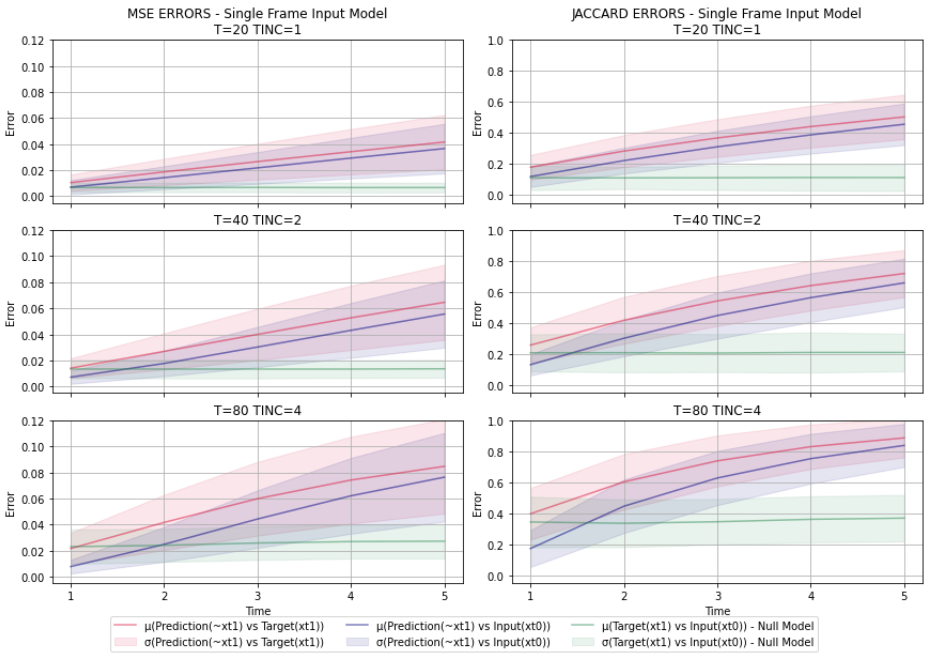

Figure 8.7: Single Frame Input Model - Long-term MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

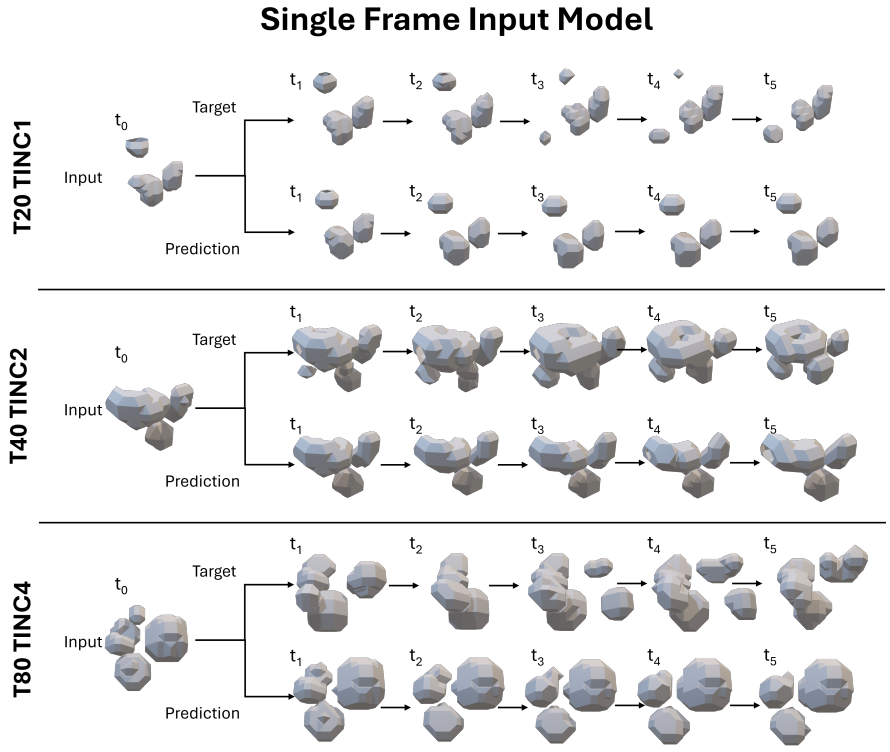

Figure 8.8: Single Frame Input Model - Exemplary generated long-term predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

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.

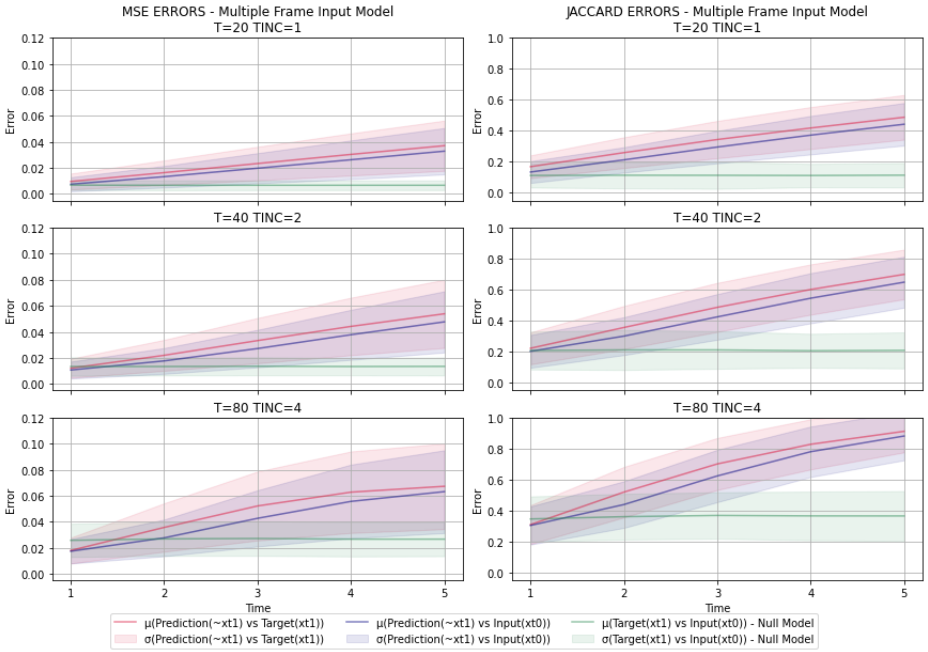

Figure 8.9: Multiple Frame Input Model - Long-term MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

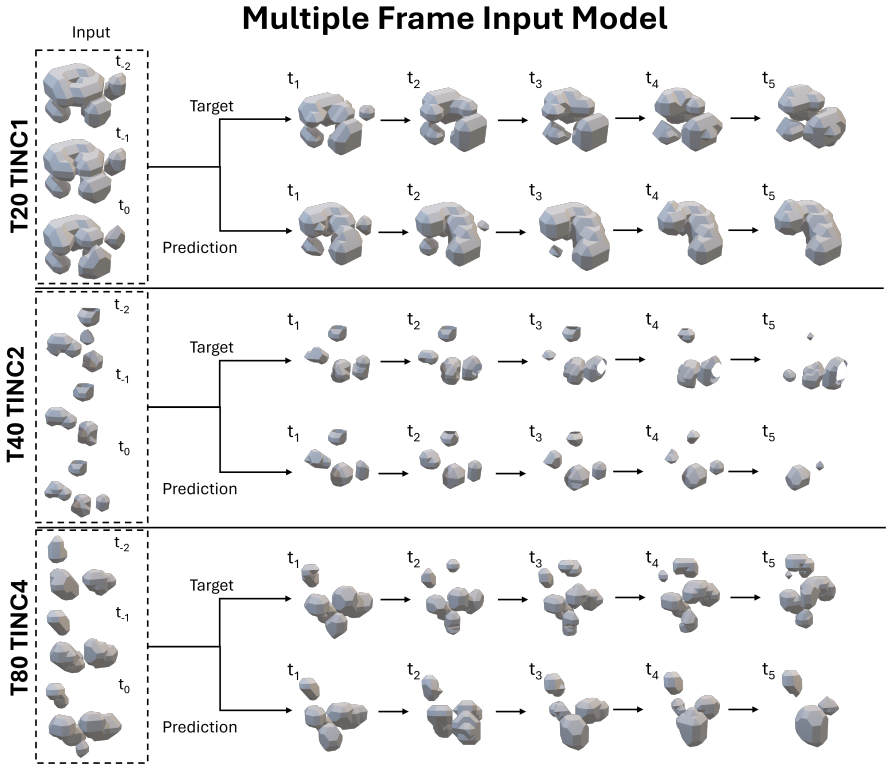

Figure 8.10: Multiple Frame Input Model - Exemplary generated long-term predictions for the three data formats T20-TINC1, T40-TINC2 and T80-TINC4.

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 weer 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.

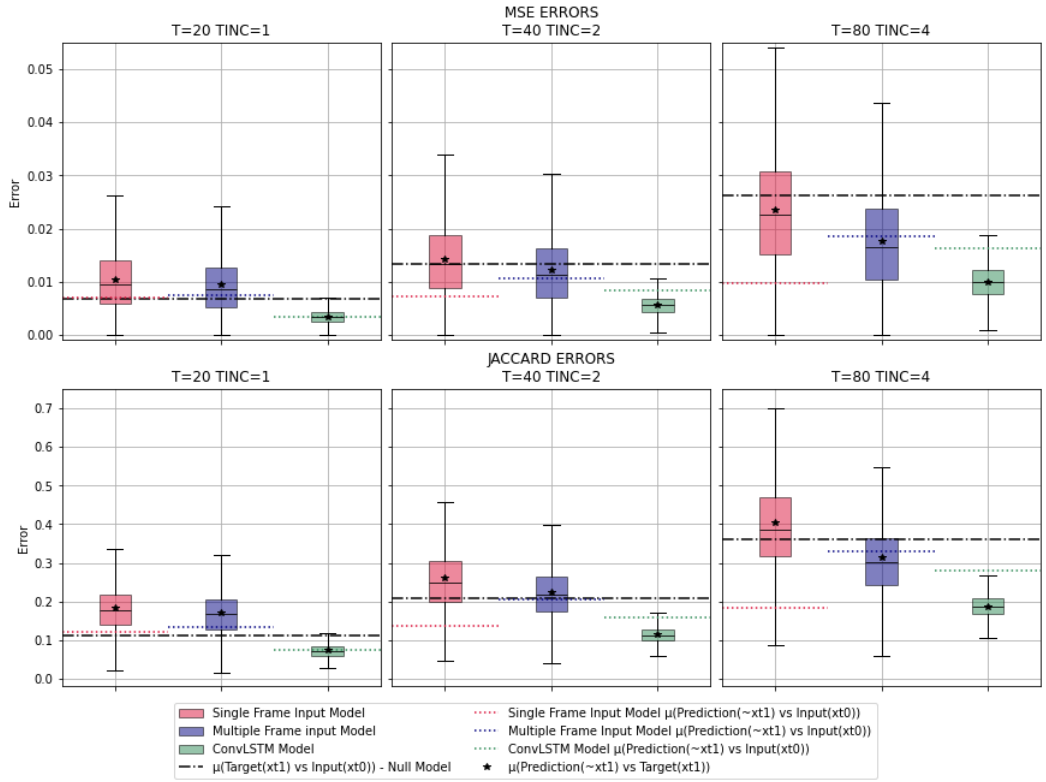

Figure 9.1: Boxplot comparison of all models for the MSE and Jaccard Distance between prediction-target, prediction-input and the null model for the three data formats T=20-TINC=1, T=40-TINC=2 and T=80-TINC=4.

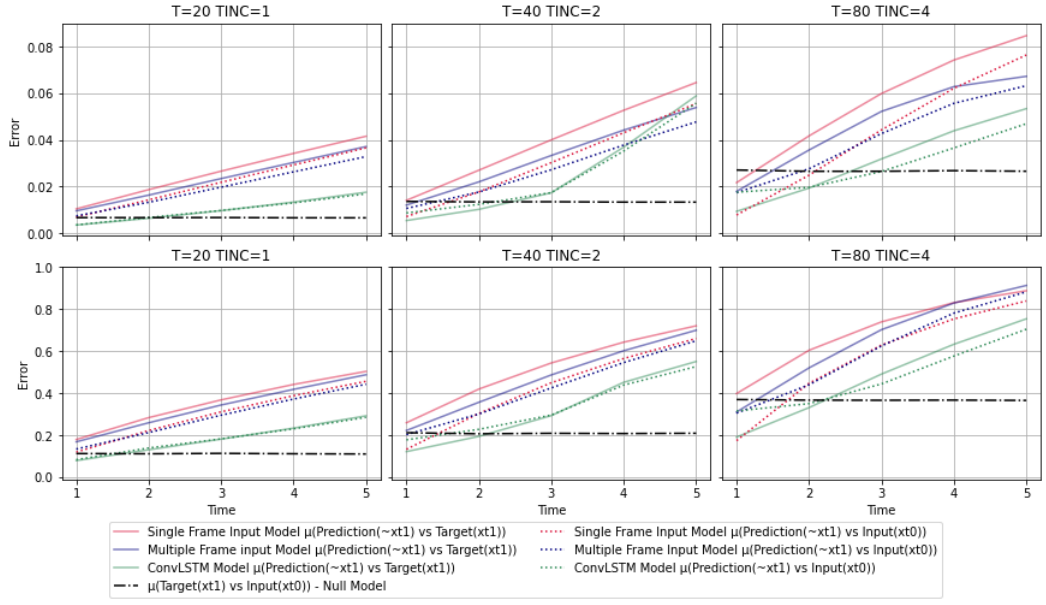

Figure 9.2: Long-term prediction comparison of all models for the MSE and Jaccard Distance between prediction-target, prediction-input and the Null Model for the three data formats T=20-INC=1, T=40-INC=2 and T=80-INC=4.

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 produced 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 128x128x128x128 (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 process-able 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].

[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].

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[5] Emad Barsoum, John Kender, and Zicheng Liu. *HP-GAN: Probabilistic 3D human motion prediction via GAN*. 2017. arXiv: 1711.09561 [cs.CV].

[6] Andrew Brock et al. *Generative and Discriminative Voxel Modeling with Convolutional Neural Networks*. 2016. arXiv: 1608.04236 [cs.CV].

[7] Baptiste Chopin et al. *Human Motion Prediction Using Manifold-Aware Wasserstein GAN*. 2021. arXiv: 2105.08715 [cs.CV].

[8] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[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. 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].

[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].

[12] Kaiming He et al. *Identity Mappings in Deep Residual Networks*. 2016. arXiv: 1603.05027 [cs.CV].

[13] Justin Hellermann and Stefan Lessmann. *Leveraging Image-based Generative Adversarial Networks for Time Series Generation*. 2023. arXiv: 2112.08060 [cs.LG].

[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. 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].

[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].

[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].

[20] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[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].

[24] Masaki Saito, Eiichi Matsumoto, and Shunta Saito. *Temporal Generative Adversarial Nets with Singular Value Clipping*. 2017. arXiv: 1611.06624 [cs.LG].

[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].

[28] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[29] Satya Pratheek Tata and Subhankar Mishra. *3D GANs and Latent Space: A comprehensive survey*. 2023. arXiv: 2304.03932 [cs.CV].

[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].

[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.

[34] Jiajun Wu et al. *Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling*. 2017. arXiv: 1610.07584 [cs.CV].

[35] Wenxuan Wu, Zhongang Qi, and Li Fuxin. *PointConv: Deep Convolutional Networks on 3D Point Clouds*. 2020. arXiv: 1811.07246 [cs.CV].

[36] Xiaohui Zeng et al. *LION: Latent Point Diffusion Models for 3D Shape Generation*. 2022. arXiv: 2210.06978 [cs.CV].

List of Figures

List of Tables

List of Source Codes

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

```
import os
from shutil import copyfile
import glob
import time
import sys
import numpy as np

from keras import layers
import tensorflow as tf
from tensorflow.keras import Model
from tensorflow.keras.optimizers import Adam

# OWN LIBRARIES
from utils.loss_plotter import render_graphs
from utils import dataIO as d
from utils import parse_tfrecords_4D_16_for_training_ConvLSTM as p4d
from utils.logger import Logger

################################################################################
########################################
########## inputs  ###################
########################################
N_EPOCHS            = 10000             # Default: 10000 - Max. training epochs
MAX_ITERATIONS      = 10000             # Default: -1 --> Max. Training Iterations (Max. Number of processed Batches) before training stops
TRAIN_DATASET_LOC   = "Z:/Master_Thesis/data/00_4D_training_datasets"
VALID_DATASET_LOC   = "Z:/Master_Thesis/data/01_4D_validation_datasets"
N_TRAINING          = 10000             # Number of Files for Training: -1 == USE ALL
N_VALIDATION        = 1000              # Number of Files for Validation: -1 == USE ALL
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.
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]
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?
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
RESTART_FOLDER_NAME = '20240221_012216_aencgan_3dnext_16_t3'
##########################################################################
# Output Settings
VAL_SAVE_FREQ       = 200               # Default: N_TRAINING//BATCH_SIZE - OFF: -1, every n training iterations (n batches) check validation loss
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)
BEST_METRIC         = "JACCARD"         # Default: JACCARD - "JACCARD" or "MSE"
# if KEEP_BEST_ONLY = False define max. Number of last models to keep:
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
##########################################################################
########## HYPERPARAMETERS ############
#######################################
BATCH_SIZE          = 16                # Default: 64 - Number of Examples per Batch / Probably Increase for better Generalization
FEATUREMAPS         = 32                # Default: 5 - Max Featuremap size (Maybe increase for learning larger/more diverse datasets...)
##########################################################################
# Optimizer Settings
LR                  = 0.001             # Default: 0.001
##########################################################################
# DEFAULT - DONT TOUCH SETTINGS
SIZE                = 16                # Default: 16 - Voxel Cube Size
##########################################################################
############# GET DATA/OBJECTS ###########################################
##########################################################################
TRAINING_FILENAMES = tf.io.gfile.glob(TRAIN_DATASET_LOC+"/*.tfrecords")[:N_TRAINING]
VALIDATION_FILENAMES = tf.io.gfile.glob(VALID_DATASET_LOC+"/*.tfrecords")[:N_VALIDATION]
training_set_size = len(TRAINING_FILENAMES)
validation_set_size = len(VALIDATION_FILENAMES)
print()
print("Total Number of Samples to Train:", training_set_size)
print("Total Number of Samples for Validation:", validation_set_size)
```

```
print()

# 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 )
val_objects = p4d.get_dataset(VALIDATION_FILENAMES, shuffle=False, batch_size=
    BATCH_SIZE, t=T, tinc=TINC, normalize=NORMALIZE )

################################################################################
################### CONVOLUTIONAL LSTM NETWORK #################################
################################################################################
# DEFINE CONVLSTM NETWORK
def ConvLSTM3D():
    inp = layers.Input(shape=(None, *(16,16,16,1)))

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

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

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

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

    model = Model(inp, x)

    return model

################################################################################
########## COMPILE NETWORK #####################################################
################################################################################
model = ConvLSTM3D()
#Compile Model
model.compile()
# PRINT NETWORK SUMMARY
print(model.summary())

################################################################################
############## Optimizer #######################################################
################################################################################
optimizer = Adam(learning_rate=LR)

################################################################################
############## LOSS ############################################################
```

```
#############################################################################
@tf.function
def BCE_LOSS(y_true, y_pred):
    bce=tf.keras.metrics.binary_crossentropy(y_true, y_pred, from_logits=False,
    label_smoothing=0.0, axis=-1)
    BCE_loss = tf.math.reduce_mean(bce)

    return BCE_loss

@tf.function
def MSE_LOSS(y_true, y_pred):
    mse = tf.keras.metrics.mean_squared_error(y_true, y_pred)
    MSE_loss = tf.reduce_mean(mse)

    return MSE_loss

def JACCARD_LOSS(X, y):
    y_true = tf.dtypes.cast(y, tf.float32)
    y_pred = np.squeeze(model(X, training=False))

    y_real = np.asarray(y_true>0.5, bool)
    y_pred = np.asarray(y_pred>0.5, bool)

    j_and = tf.math.reduce_sum(np.double(np.bitwise_and(y_real, y_pred)), axis
    =[2,3,4])
    j_or = tf.math.reduce_sum(np.double(np.bitwise_or(y_real, y_pred)), axis
    =[2,3,4])

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

    return JACCARD_loss

#############################################################################
############## TRAINING STEP ################################################
#############################################################################
@tf.function
def train_step(X, y):

    X = X[...,np.newaxis]
    y_true = y[...,np.newaxis]

    with tf.GradientTape() as tape:
        y_pred = model(X, training=True)
        BCE_loss = BCE_LOSS(y_true, y_pred)
        MSE_loss = MSE_LOSS(y_true, y_pred)

    gradients = tape.gradient(BCE_loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    return BCE_loss, MSE_loss
```

```
1158
@tf.function
1160 def validation_step(X, y):
    X = X[...,np.newaxis]
1162     y_true = y[...,np.newaxis]

1164     y_pred = model(X, training=False)
    BCE_loss = BCE_LOSS(y_true, y_pred)
1166     MSE_loss = MSE_LOSS(y_true, y_pred)

1168     return BCE_loss, MSE_loss

1170 ##############################################################################
######### SOME UTILITY FUNCTIONS ###############################################
1172 ##############################################################################
def generate_sample_objects(objects, total_iterations, tag="pred"):
1174     # Random Sample at every VAL_SAVE_FREQ
    i=0
1176
    for X, y in objects:
1178         X = tf.dtypes.cast(X, tf.float32)
        y_pred = np.squeeze(model(X, training=False) > 0.5)
1180         y_pred_last = y_pred[-1]
        try:
1182             d.plotMeshFromVoxels(y_pred_last, obj=model_directory+tag+'_'+str(i)+'
    _next'+'_iter_'+str(total_iterations))
        except:
1184             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
1186                 # 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.''')

1188         i+=1

1190 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:
1192         if KEEP_BEST_ONLY:
            fileList_all = glob.glob(model_directory + '*_iter_*')
1194             fileList_best = glob.glob(model_directory + '*_iter_' + str(int(
    best_iteration)) +'*')
            fileList_del = [ele for ele in fileList_all if ele not in fileList_best]
1196
        else:
1198             fileList_del = glob.glob(model_directory + '*_iter_' + str(int(
    total_iterations-KEEP_LAST_N*VAL_SAVE_FREQ)) +'*')
```

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

################################################################################
########### LAST PREPARATION STEPS BEFORE TRAINING STARTS ######################
################################################################################
#generate folders:
if RESTART_TRAINING:
    model_directory = os.getcwd() + '/' + RESTART_FOLDER_NAME + '/'
else:
    named_tuple = time.localtime() # get struct_time
    time_string = time.strftime("%Y%m%d_%H%M%S", named_tuple)
    model_directory = os.getcwd()+'/'+time_string+'_' + os.path.splitext(os.path.
    basename(__file__))[0] + '/'
    if not os.path.exists(model_directory):
        os.makedirs(model_directory)

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

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

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

    try:
        sample_name = model_directory+'train_'+str(i)+'_init'
        d.plotMeshFromVoxels(np.array(X_last), obj=sample_name)
        sample_name = model_directory+'train_'+str(i)+'_next'
        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")

    train_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

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

    try:
        sample_name = model_directory+'valid_'+str(i)+'_init'
        d.plotMeshFromVoxels(np.array(X_last), obj=sample_name)
        sample_name = model_directory+'valid_'+str(i)+'_next'
        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")

    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

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

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

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_set_size // 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  = [], []
BCE_losses_train = []
MSE_losses_train = []
JACCARD_losses_train = []
BCE_losses_val_mean = []
MSE_losses_val_mean = []
JACCARD_losses_val_mean = []
n_objects = 0
total_iterations = 0
best_iteration = VAL_SAVE_FREQ
best_loss = 1e10
for epoch in range(1, N_EPOCHS+1):
    ############################################################################
    # START OF EPOCH
    t_epoch_start = time.time()
    n_batch=0
    ###################################

    for X, y in train_objects:
        # START OF BATCH
        batch_time = time.time()
        BCE_loss_train, MSE_loss_train = train_step(X, y)
        #Append arrays
        BCE_losses_train.append(BCE_loss_train)
        MSE_losses_train.append(MSE_loss_train)

        JACCARD_loss_train = JACCARD_LOSS(X, y)
        JACCARD_losses_train.append(JACCARD_loss_train)

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

        print("E: %3d/%5d | B: %3d/%3d | I: %5d | T: %5.1f | dt: %2.2f || LOSSES:
    BCE %.4f | MSE %.4f | JACCARD %.4f | " \
            % (epoch, N_EPOCHS, n_batch, training_set_size//BATCH_SIZE,
    total_iterations, time.time() - start_time, time.time() - batch_time,
    BCE_loss_train, MSE_loss_train, JACCARD_loss_train))

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

            BCE_losses_val = []
```

```
            MSE_losses_val = []
            JACCARD_losses_val = []
            for X, y in val_objects:
                BCE_loss_val, MSE_loss_val = validation_step(X, y)
                BCE_losses_val.append(BCE_loss_val) # Collect Val loss per Batch
                MSE_losses_val.append(MSE_loss_val) # Collect Val loss per Batch

                JACCARD_loss_val = JACCARD_LOSS(X, y)
                JACCARD_losses_val.append(JACCARD_loss_val) # Collect Val loss per
    Batch

            BCE_losses_val_mean.append(np.mean(BCE_losses_val)) # Mean over batches
            MSE_losses_val_mean.append(np.mean(MSE_losses_val)) # Mean over batches
            JACCARD_losses_val_mean.append(np.mean(JACCARD_losses_val)) # Mean over
    batches

            val_iter_arr.append(total_iterations)
        ###########################################################################
            # Output generated objects
            generate_sample_objects(train_objects_samples, total_iterations, 'train'
    )
            generate_sample_objects(val_objects_samples, total_iterations, 'valid')
            # # Plot and Save Loss Curves
            render_graphs(model_directory, BCE_losses_train, MSE_losses_train,
    JACCARD_losses_train, BCE_losses_val_mean, MSE_losses_val_mean,
    JACCARD_losses_val_mean, iter_arr, val_iter_arr, RESTART_TRAINING) #this will
    only work after a 50 iterations to allow for proper averaging

            if KEEP_BEST_ONLY:
                if BEST_METRIC == "JACCARD":
                    BEST_LOSS_METRIC = JACCARD_losses_val_mean
                    KEEP_LAST_N = 1
                elif BEST_METRIC == "MSE":
                    BEST_LOSS_METRIC = MSE_losses_val_mean
                    KEEP_LAST_N = 1
                if len(BEST_LOSS_METRIC) > 1 and (BEST_LOSS_METRIC[-1] <= best_loss)
    :
                    best_loss = BEST_LOSS_METRIC[-1]
                    best_iteration = total_iterations
                    # Save models
                    model.save(model_directory+'_trained_ConvLSTM3D_'+'iter_'+str(
    total_iterations)+'.h5')
            else:
                # Save models
                model.save(model_directory+'_trained_ConvLSTM3D_'+'iter_'+str(
    total_iterations)+'.h5')

            # # Delete Model, keep best or last N models
            IOhousekeeping(model_directory, KEEP_LAST_N, VAL_SAVE_FREQ,
    RESTART_TRAINING, total_iterations, KEEP_BEST_ONLY, best_iteration)
```

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

1386            ################################
                # STOP TRAINING AFTER MAX DEFINED ITERATIONS ARE REACHED
1388            if total_iterations == MAX_ITERATIONS :
                    break
1390        if total_iterations == MAX_ITERATIONS:
                break
1392
        # Print Status at the end of the epoch
1394      dt_epoch =  time.time() - t_epoch_start
        n_objects = n_objects + training_set_size
1396      print("\n----------------------------------------------------------------------")
        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")
1398      print("----------------------------------------------------------------------\n\n"
        )

1400    # END OF EPOCH
        ###########################################################################
1402


1404 print('\n TRAINING DONE! \n')
    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
import numpy as np
1004
AUTOTUNE = tf.data.AUTOTUNE
1006
def decode_object(example):
1008     object4d = tf.io.decode_raw(example['object4d'], tf.uint8)
    labelBetti = tf.io.decode_raw(example['labelBetti'], tf.uint8)
1010     labelObjs = tf.io.decode_raw(example['labelObjs'], tf.uint8)

1012     object4d = tf.reshape(object4d, [128, 16, 16, 16])
    labelBetti = tf.reshape(labelBetti, [4])
1014     labelObjs = tf.reshape(labelObjs, [4])

1016     return object4d, labelBetti, labelObjs


1018
def read_tfrecord(example):
1020     global NORMALIZE
    global T
1022     global TINC

1024     object_feature_description = {
        'object4d': tf.io.FixedLenFeature([], tf.string),
1026         'labelBetti': tf.io.FixedLenFeature([], tf.string),
        'labelObjs': tf.io.FixedLenFeature([], tf.string),
1028     }

1030     example = tf.io.parse_single_example(example, object_feature_description)

1032     object4d, labelBetti, labelObjs = decode_object(example)

1034     CROP=(128-T)//2
    # Reshape into time slices of n time steps (axis=1)
1036     object4d = object4d[CROP:128-CROP:TINC]
    object4d = tf.reshape(object4d, [T//TINC, 16, 16, 16])
1038     x_object4d = object4d[0 : object4d.shape[0] - 1, :, :]
    y_object4d = object4d[1 : object4d.shape[0], :, :]
1040
    if NORMALIZE:
1042         object4d = tf.cast(object4d, tf.float32) - 0.5

1044     return x_object4d, y_object4d #, tf.ones([object4d.shape[0]-1,4], tf.uint8)*
    labelObjs
```

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

def load_dataset(filenames):
    options = tf.data.Options()
    options.deterministic = False  # disable order, increase speed
    dataset = tf.data.TFRecordDataset(filenames, compression_type="GZIP",
    num_parallel_reads=AUTOTUNE)  # automatically interleaves reads from multiple
    files
    dataset = dataset.with_options(options)  # uses data as soon as it streams in,
    rather than in its original order
    dataset = dataset.map(read_tfrecord, num_parallel_calls=AUTOTUNE)
    return dataset

def get_dataset(filenames: list, shuffle: bool, batch_size: int, t: int, tinc: int,
    normalize: bool):
    global NORMALIZE
    global T
    global TINC
    NORMALIZE = normalize
    TINC = tinc
    T = t
    dataset = load_dataset(filenames)

    if shuffle:
        dataset = dataset.shuffle(len(filenames)*(T//TINC)//2) # ALL: len(filenames)
    *(T//TINC)
    if batch_size is not None:
        dataset = dataset.batch(batch_size, drop_remainder=True)
    #dataset = dataset.prefetch(buffer_size=AUTOTUNE)
    #dataset = dataset.cache()
    return dataset


# FOR TESTING:
'''
folder_name="Z:/Master_Thesis/data/4D_128_32_topological_dataset_0-31_16_label8only"

FILENAMES = tf.io.gfile.glob(folder_name+"/*.tfrecords")[:7]
split_ind = int(1 * len(FILENAMES))
TRAINING_FILENAMES, VALID_FILENAMES = FILENAMES[:split_ind], FILENAMES[split_ind:]

#TEST_FILENAMES = tf.io.gfile.glob(GCS_PATH + "/tfrecords/test*.tfrec")
print("Train TFRecord Files:", len(TRAINING_FILENAMES))
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)

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

Listing A.2: ConvLSTM3D - Data Parser

A.3 ConvLSTM3D Prediction Algorithm

```
import os
import tensorflow as tf
from keras.models import load_model
import numpy as np
# OWN LIBRARIES
from utils import parse_tfrecords_4D_16_for_training_ConvLSTM as p4d
from utils import dataIO as d


base_dir = r"Z:\Master_Thesis\code\99_FINAL_MODELS_FOR_THESIS"

MODEL_LOCATION       = os.path.join(base_dir, "convlstm3d_16_T20_TINC1")

TEST_DATASET_LOC    = "Z:/Master_Thesis/data/02_4D_test_datasets"

T                    = int(MODEL_LOCATION[-8:-6])
TINC                 = int(MODEL_LOCATION[-1])
N_PREDICT            = 5

if not os.path.exists(os.path.join(base_dir,f'objects/T{T}_TINC{TINC}')):
    os.makedirs(os.path.join(base_dir,f'objects/T{T}_TINC{TINC}'))
################################################################################
############# GET Data/Objects ################################################
################################################################################
TEST_FILENAMES = tf.io.gfile.glob(os.path.join(TEST_DATASET_LOC,"*.tfrecords"))
test_set_size = len(TEST_FILENAMES)

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

# Convert/Decode dataset from tfrecords file for training
test_objects = p4d.get_dataset(TEST_FILENAMES, shuffle=True, batch_size=1, t=T, tinc
    =TINC, normalize=False)

convlstm3d_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*ConvLSTM3D*.h5"))
convlstm3d = load_model(convlstm3d_model[-1])

def predictions(objects):
    n=0
    for X, y in objects:
        d.plotFromVoxels(np.squeeze(X[0,0]), f"{n}_T{T}_TINC{TINC}_t-1")
        d.plotMeshFromVoxels(np.squeeze(X[0,0]), obj=f"objects/T{T}_TINC{TINC}/
    convlstm_{n}_T{T}_TINC{TINC}_t0")
        d.plotFromVoxels(np.squeeze(X[0,1]), f"{n}_T{T}_TINC{TINC}_t-2")
        d.plotMeshFromVoxels(np.squeeze(X[0,1]), obj=f"objects/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/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/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/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/T{T}_TINC{TINC}/convlstm_{n}_T{T}_TINC{TINC}_t5")

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

        d.plotFromVoxels(y_pred[0], f"{n}_T{T}_TINC{TINC}_t1_pred")
        d.plotMeshFromVoxels(y_pred[0], obj=f"objects/T{T}_TINC{TINC}/convlstm_{n}_T{T}_TINC{TINC}_t1_pred")
        d.plotFromVoxels(y_pred[1], f"{n}_T{T}_TINC{TINC}_t1_pred")
        d.plotMeshFromVoxels(y_pred[1], obj=f"objects/T{T}_TINC{TINC}/convlstm_{n}_T{T}_TINC{TINC}_t2_pred")
        d.plotFromVoxels(y_pred[2], f"{n}_T{T}_TINC{TINC}_t1_pred")
        d.plotMeshFromVoxels(y_pred[2], obj=f"objects/T{T}_TINC{TINC}/convlstm_{n}_T{T}_TINC{TINC}_t3_pred")
        d.plotFromVoxels(y_pred[3], f"{n}_T{T}_TINC{TINC}_t1_pred")
        d.plotMeshFromVoxels(y_pred[3], obj=f"objects/T{T}_TINC{TINC}/convlstm_{n}_T{T}_TINC{TINC}_t4_pred")
        d.plotFromVoxels(y_pred[4], f"{n}_T{T}_TINC{TINC}_t1_pred")
        d.plotMeshFromVoxels(y_pred[4], obj=f"objects/T{T}_TINC{TINC}/convlstm_{n}_T{T}_TINC{TINC}_t5_pred")
        n+=1
        if n==N_PREDICT:
            break


predictions(test_objects)
```

Listing A.3: ConvLSTM3D Prediction Algorithm

A.4 ConvLSTM3D Long-Term Prediction Algorithm

```
import os
import tensorflow as tf
from keras.models import load_model
import numpy as np
# OWN LIBRARIES
from utils import parse_tfrecords_4D_16_for_training_ConvLSTM as p4d
from utils import dataIO as d


base_dir = r"Z:\Master_Thesis\code\99_FINAL_MODELS_FOR_THESIS"

MODEL_LOCATION      = os.path.join(base_dir, "convlstm3d_16_T20_TINC1")

TEST_DATASET_LOC    = "Z:/Master_Thesis/data/02_4D_test_datasets"

T                   = int(MODEL_LOCATION[-8:-6])
TINC                = int(MODEL_LOCATION[-1])
N_PREDICT           = 5
MAX_TIME            = 5

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

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

# Convert/Decode dataset from tfrecords file for training
test_objects = p4d.get_dataset(TEST_FILENAMES, shuffle=True, batch_size=1, t=T, tinc
    =TINC, normalize=False)

convlstm3d_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*ConvLSTM3D*.h5"))
convlstm3d = load_model(convlstm3d_model[-1])

def predictions(objects):
    n=0
    for X, y in objects:
        d.plotFromVoxels(np.squeeze(X[0,0]), f"{n}_T{T}_TINC{TINC}_t-1")
        d.plotMeshFromVoxels(np.squeeze(X[0,0]), obj=f"objects_longterm/T{T}_TINC{
    TINC}/convlstm_{n}_T{T}_TINC{TINC}_t0")
        d.plotFromVoxels(np.squeeze(X[0,1]), f"{n}_T{T}_TINC{TINC}_t-2")
```

```
        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")

        X = tf.dtypes.cast(X[:,0:5], tf.float32)
        for t in range(4,y.shape[1]-1):
            y_pred = np.squeeze(convlstm3d(X[...,np.newaxis], training=False))
            y_pred = np.asarray(y_pred>0.5, bool)[-1]

            d.plotFromVoxels(y_pred, f"{n}_T{T}_TINC{TINC}_t{t+1}_pred")
            d.plotMeshFromVoxels(y_pred, obj=f"objects_longterm/T{T}_TINC{TINC}/convlstm_{n}_T{T}_TINC{TINC}_t{t+1}_pred")

            if t==MAX_TIME+4-1:
                break

            X = np.concatenate((X, np.expand_dims(y_pred[np.newaxis,...], axis=1)), axis=1)[0,1:6][np.newaxis,...]

        n+=1
        if n==N_PREDICT:
            break


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

```
import os
from shutil import copyfile
import glob
import sys
import numpy as np
#np.set_printoptions(threshold=sys.maxsize)
import time
import tensorflow as tf
from tensorflow.keras import Model
from tensorflow.keras.layers import Dense, BatchNormalization, LayerNormalization,
    LeakyReLU, Reshape, Conv3DTranspose, Conv3D, Flatten, ReLU, Dropout, Concatenate
    , Input, UpSampling3D, AveragePooling3D
from tensorflow.keras.activations import sigmoid, tanh
from tensorflow.keras.optimizers import Adam
import tensorflow_addons as tfa

# OWN LIBRARIES
from utils.loss_plotter import render_graphs
from utils import dataIO as d
from utils import parse_tfrecords_4D_16_for_training as p4d
from utils.logger import Logger

################################################################################
########### INPUTS  ###################
########################################
N_EPOCHS            = 10000             # Default: 10000 - Max. training epochs OR
    Define Max. Iterations
MAX_ITERATIONS      = 20000             # Default: -1 - Max. Training Iterations (
    Max. Number of processed Batches) before training stops, if "-1" Training stops
    after reaching N_EPOCHS
TRAIN_DATASET_LOC   = "Z:/Master_Thesis/data/00_4D_training_datasets"
VALID_DATASET_LOC   = "Z:/Master_Thesis/data/01_4D_validation_datasets"
N_TRAINING          = 10000             # Number of Files for Training: -1 == USE
    ALL
N_VALIDATION        = 1000              # Number of Files for Validation: -1 == USE
    ALL
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.
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]
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?
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
RESTART_FOLDER_NAME = "20240221_012216_aencgan3d_16"
################################################################################
# Output Settings
VAL_SAVE_FREQ       = 1000              # Default: 1000 - OFF: -1, every n training iterations (n batches) check validation loss
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)
BEST_METRIC         = "JACCARD"         # Default: JACCARD - "JACCARD" or "MSE"
# if KEEP_BEST_ONLY = False define max. Number of last models to keep:
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
################################################################################
########## HYPERPARAMETERS  ###########
########################################
GD_ARCHITECTURE     = "PreRes"          # Default: "PreRes" - "PreRes" or "Classic" Convolutions
BATCH_SIZE          = 32                # Default: 32 - Number of Examples per Batch / Probably Increase for better Generalization
Z_SIZE              = 256               # Default: 256 - Latent Random Vector Size
FEATUREMAP_BASE_DIM = 256               # Default: 256 - Max Featuremap size (Maybe increase for learning larger/more diverse datasets...)
################################################################################
# WGAN-GP Settings
GP_WEIGHT           = 100               # Default: 10 - WGAN-GP gradient penalty weight
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 ################################################################################
     # 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
     ################################################################################
1066 # Dropout
     DROPOUT_RATE        = 0.2             # Default: 0.2
1068 ################################################################################
     # weight_initializer & regularizer
1070 weight_initializer  = "glorot_uniform"  # "glorot_uniform" "glorot_normal" or "
         he_uniform"
     weight_regularizer  = "L1"            # tf.keras.regularizers.L1(l1=0.01)  #
         Default l1=0.01
1072 ################################################################################
     # USE BIAS IN GENERATOR
1074 USE_BIAS            = True
     bias_initializer    = "zeros" # DEFAULT: "zeros" - "ones" or tf.keras.initializers.
         Constant(0.5)
1076 ################################################################################
     # DEFAULT - DONT TOUCH SETTINGS
1078 SIZE                = 16              # Default: 16 - Object Grid Size
     BASE_SIZE           = 4               # Default: 4 - Base Resolution to start from
          for generator and to end with for discriminator
1080 KERNEL_SIZE         = 5               # Default: 5 - Lower values learn pretty bad
         , higher than 5 doesnt change training much
     ################################################################################
1082 ################### NETWORKS ###################################################
     ################################################################################
1084 # latent vector input
     latent_vector = Input(shape=(Z_SIZE,))
1086 #-------------------------------------------------------------------------------
     ################### ENCODER ####################################################
1088 #-------------------------------------------------------------------------------
     def obj_input(object_dim=(SIZE, SIZE, SIZE, 1)):
1090     obj_in = Input(shape=object_dim)
         return obj_in
1092
     ################################################################################
1094 # Encoder PreResidual Block
     def EncoderPreResBlock(x, in_filters, out_filters, kernel_size, strides,
         kernel_initializer, kernel_regularizer, padding):
1096     # Define Shortcut
         if strides == 1 and in_filters == out_filters: # Identity Skip Connection
1098         shortcut = x
         elif strides == 2 and in_filters == out_filters: # Identity Skip Connection
1100         shortcut = AveragePooling3D(pool_size=(strides, strides, strides), strides=(
         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
```

```
    model = Model(obj_in, out_layer)
    return model

def PreResEncoder():
    obj_in = obj_input()

    # 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)

    # PreRes Blocks
    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")
    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")

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

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

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

    # define model
    model = Model(obj_in, out_layer)
    return model

##########################################################################

#--------------------------------------------------------------------------
################### GENERATOR ############################################
#--------------------------------------------------------------------------

def latent_input(latent_vector=latent_vector, BASE_SIZE=BASE_SIZE):
    base = BASE_SIZE * BASE_SIZE * BASE_SIZE

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

##########################################################################
```

```
# 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
    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)
    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)
    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)
        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)

    # Define Activation on Input
    x = BatchNormalization()(x)
    x = ReLU()(x)
    if up_size != 1:
        x = UpSampling3D(size=(up_size, up_size, up_size))(x)
    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)

    return shortcut + x
################################################################################
# DEFINE GENERATOR NETWORK
def ClassicGenerator():
    latent_in = latent_input()
    x = BatchNormalization()(latent_in)
    x = ReLU()(x)

    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)

    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)
```

```
    x = BatchNormalization()(x)
    x = ReLU()(x)

    x = UpSampling3D(size=(2, 2, 2))(x)
    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 =
    bias_initializer)(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    x = Conv3DTranspose(filters=1, 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)

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

    model = Model(latent_vector, out_layer)

    return model

def PreResGenerator():
    x = latent_input()

    # PreRes Blocks
    x = GPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM    , out_filters=
    FEATUREMAP_BASE_DIM//2, kernel_size=KERNEL_SIZE, up_size=1, kernel_initializer=
    weight_initializer, kernel_regularizer=weight_regularizer, padding="same",
    use_bias=USE_BIAS, bias_initializer = bias_initializer)
    x = GPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//2, out_filters=
    FEATUREMAP_BASE_DIM//4, kernel_size=KERNEL_SIZE, up_size=2, kernel_initializer=
    weight_initializer, kernel_regularizer=weight_regularizer, padding="same",
    use_bias=USE_BIAS, bias_initializer = bias_initializer)
    x = GPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//4, out_filters=
    FEATUREMAP_BASE_DIM//8, kernel_size=KERNEL_SIZE, up_size=2, kernel_initializer=
    weight_initializer, kernel_regularizer=weight_regularizer, padding="same",
    use_bias=USE_BIAS, bias_initializer = bias_initializer)

    x = BatchNormalization()(x)
    x = ReLU()(x)
    # Final Convolution
    x = Conv3DTranspose(filters=1, 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)

    # End Activation
    if NORMALIZE:
```

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

    model = Model(latent_vector, out_layer)

    return model

#============================================================================

#-----------------------------------------------------------------------
################### DISCRIMINATOR ###########################################
#-----------------------------------------------------------------------

def real_obj_input_disriminator(object_dim=(SIZE, SIZE, SIZE, 1)):
    real_obj_in = Input(shape=object_dim)
    return real_obj_in
def cond_obj_input_disriminator(object_dim=(SIZE, SIZE, SIZE, 1)):
    cond_obj_in = Input(shape=object_dim)
    return cond_obj_in

#============================================================================
# Discriminator PreResidual Block
def DPreResBlock(x, in_filters, out_filters, kernel_size, strides,
    kernel_initializer, kernel_regularizer, padding):
    # Define Shortcut
    if strides == 1 and in_filters == out_filters: # Identity Skip Connection
        shortcut = x
    elif strides == 2 and in_filters == out_filters: # Identity Skip Connection
        shortcut = AveragePooling3D(pool_size=(strides, strides, strides), strides=(
    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 = LayerNormalization()(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 DISCRIMINATOR NETWORK
def ClassicDiscriminator():
    real_obj_in = real_obj_input_disriminator()
    cond_obj_in = cond_obj_input_disriminator()

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

    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)
    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 = LayerNormalization()(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(1, activation=None)(x)

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

def PreResDiscriminator():
    real_obj_in = real_obj_input_disriminator()
    cond_obj_in = cond_obj_input_disriminator()

    # merge label_conditioned_disriminator and latent_input output
    x = Concatenate()([real_obj_in, cond_obj_in])

    # 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")(x)
```

```
    # PreRes Blocks
    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")

    x = LayerNormalization()(x)
    x = LeakyReLU()(x)
    x = Dropout(DROPOUT_RATE)(x)

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

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

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

################################################################################
########## COMPILE NETWORKS ####################################################
################################################################################
if GD_ARCHITECTURE == "PreRes":
    encoder, generator, discriminator = PreResEncoder(), PreResGenerator(),
    PreResDiscriminator()
elif GD_ARCHITECTURE == "Classic":
    encoder, generator, discriminator = ClassicEncoder(), ClassicGenerator(),
    ClassicDiscriminator()
else:
    ValueError("Generator/Disriminator Architecture not Available. Choose between '
    PreRes' or 'Classic'!")
encoder.compile()
generator.compile()
discriminator.compile()

################################################################################
############## LOSS FUNCTION ###################################################
################################################################################
@tf.function
def encgen_loss(real_objects):

    real_objects_t0 = tf.dtypes.cast(real_objects[:,0], tf.float32)
    real_objects_t1 = tf.dtypes.cast(real_objects[:,1], tf.float32)

    # Latent Encoding
```

```
    latent_code = encoder(real_objects_t0, training=True)

    pred_objects_t1 = generator(latent_code, training=True)
    fake_logits = discriminator([pred_objects_t1, real_objects_t1], training=True)
    loss = wgan_gp_generator_loss(fake_logits)

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

    return loss, MSE_loss

@tf.function
def encdisc_loss(real_objects):

    real_objects_t0 = tf.dtypes.cast(real_objects[:,0], tf.float32)
    real_objects_t1 = tf.dtypes.cast(real_objects[:,1], tf.float32)

    # Latent Encoding
    latent_code = encoder(real_objects_t0, training=True)

    pred_objects_t1 = generator(latent_code, training=True)
    fake_logits = discriminator([pred_objects_t1, real_objects_t1], training=True)
    real_logits = discriminator([real_objects_t1, real_objects_t1], training=True)
    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)

    return D_loss, fake_loss, real_loss, gp, ep

@tf.function
def val_encdisc_loss(val_real_objects):

    val_real_objects_t0 = tf.dtypes.cast(val_real_objects[:,0], tf.float32)
    val_real_objects_t1 = tf.dtypes.cast(val_real_objects[:,1], tf.float32)

    # Latent Encoding
    val_latent_code = encoder(val_real_objects_t0, training=False)

    val_pred_objects_t1 = generator(val_latent_code, training=False)
    val_fake_logits = discriminator([val_pred_objects_t1, val_real_objects_t1],
    training=False)
    val_real_logits = discriminator([val_real_objects_t1, val_real_objects_t1],
    training=False)
    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)

    return D_loss, fake_loss, real_loss, gp, ep

# WGAN LOSSES
```

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

    This loss is calculated on an interpolated image
    and added to the discriminator loss.
    """
    # 1. Get the interpolated onject
    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)

    # 2. Calculate the gradients w.r.t to this interpolated object.
    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

    # 3. Calculate the norm of the gradients.
    gradient_penalty = tf.reduce_mean((norm-1.0)**2)

    # Add the gradient penalty to the original discriminator loss and MEAN over
    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)

    D_loss =  fake_loss - real_loss + gp + ep

    return D_loss, fake_loss, real_loss, gp, ep

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

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

    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:
        # generator loss
1462        EncGen_loss = encgen_loss(real_objects)

1464    gradients_of_encoder = enc_tape.gradient(EncGen_loss[0], encoder.
    trainable_variables)
    encoder_optimizer.apply_gradients(zip(gradients_of_encoder, encoder.
    trainable_variables))
1466
    gradients_of_generator = gen_tape.gradient(EncGen_loss[0], generator.
    trainable_variables)
1468    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.
    trainable_variables))

1470    return EncGen_loss, EncDisc_loss

1472@tf.function
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
@tf.function
1484def MSE_reconstruction_loss(real_objects):
    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)
    pred_objects_t1 = generator(latent_code, training=False)
1490
    mse = tf.keras.metrics.mean_squared_error(real_objects_t1, pred_objects_t1)
1492    MSE_loss = tf.reduce_mean(mse)

1494    return MSE_loss

1496def JACCARD_reconstruction_loss(val_obj_batch):
    val_real_objects = val_obj_batch[...,np.newaxis]
1498
    val_real_objects_t0 = tf.dtypes.cast(val_real_objects[:,0], tf.float32)
```

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

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

    if NORMALIZE:
        x = np.asarray(val_real_objects_t1>0.0, bool)
        y = np.asarray(val_pred_objects_t1>0.0, bool)
    else:
        x = np.asarray(val_real_objects_t1>0.5, bool)
        y = np.asarray(val_pred_objects_t1>0.5, bool)

    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])

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

    return JACCARD_loss

################################################################################
######### SOME UTILITY FUNCTIONS ###############################################
################################################################################
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 = encoder(X, training=False)

        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)

        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.""")

        generated_objects.append(object_out)
```

```
1546            i+=1

1548        print()

1550        return generated_objects

1552    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:
1554            if KEEP_BEST_ONLY:
                    fileList_all = glob.glob(model_directory + "*_iter_*")
1556                fileList_best = glob.glob(model_directory + "*_iter_" + str(int(
        best_iteration)) +"*")
                    fileList_del = [ele for ele in fileList_all if ele not in fileList_best]
1558
            else:
1560                fileList_del = glob.glob(model_directory + "*_iter_" + str(int(
        total_iterations-KEEP_LAST_N*VAL_SAVE_FREQ)) +"*")

1562            # Iterate over the list of filepaths & remove each file.
            for filePath in fileList_del:
1564                os.remove(filePath)

1566    ##############################################################################
        ########### LAST PREPARATION STEPS BEFORE TRAINING STARTS ####################
1568    ##############################################################################
        #generate folders:
1570    if RESTART_TRAINING:
        model_directory = os.getcwd() + "/" + RESTART_FOLDER_NAME + "/"
1572    else:
        named_tuple = time.localtime() # get struct_time
1574    time_string = time.strftime("%Y%m%d_%H%M%S", named_tuple)
        model_directory = os.getcwd()+"/"+time_string+"_" + os.path.splitext(os.path.
        basename(__file__))[0] + "/"
1576    if not os.path.exists(model_directory):
            os.makedirs(model_directory)
1578
    # Save running python file to run directory
1580    if RESTART_TRAINING:
        named_tuple = time.localtime() # get struct_time
1582    time_string = time.strftime("%Y%m%d_%H%M%S", named_tuple)
        copyfile(__file__, model_directory+time_string +"_restart_"+os.path.basename(
        __file__))
1584    else:
        copyfile(__file__, model_directory+os.path.basename(__file__))
1586
    # Activate Console Logging:
1588    if LOGGING:
        sys.stdout = Logger(filename=model_directory+"log_out.txt")
```

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

1596 ##########################################################################
############ GET DATA/OBJECTS ############################################
1598 ##########################################################################
TRAINING_FILENAMES = tf.io.gfile.glob(TRAIN_DATASET_LOC+"/*.tfrecords")[:N_TRAINING]
1600 VALIDATION_FILENAMES = tf.io.gfile.glob(VALID_DATASET_LOC+"/*.tfrecords")[:
    N_VALIDATION]

1602 training_set_size = len(TRAINING_FILENAMES)
validation_set_size = len(VALIDATION_FILENAMES)
1604 training_samples = training_set_size * (T//TINC-1) // BATCH_SIZE * BATCH_SIZE
validation_samples = validation_set_size * (T//TINC-1) // BATCH_SIZE * BATCH_SIZE
1606 print()
print("Total Number of TFRecord Files to Train :", training_set_size)
1608 print("Total Number of Samples to Train:", training_samples)
print()
1610 print("Total Number of TFRecord Files for Validation:", validation_set_size)
print("Total Number of Samples for Validation:", validation_samples)
1612 print()

1614 # 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 )
1616 val_objects = p4d.get_dataset(VALIDATION_FILENAMES, shuffle=False, batch_size=
    BATCH_SIZE, t=T, tinc=TINC, normalize=NORMALIZE )

1618 # save some real examples
i=0
1620 test_objects = []
for X in train_objects.take(N_SAMPLES):
1622     Xt0 = X[0][0]
    Xt1 = X[0][1]
1624
    try:
1626         sample_name = model_directory+"train_"+str(i)+"_init"#+str(t)
        d.plotMeshFromVoxels(np.array(Xt0), obj=sample_name)
1628         sample_name = model_directory+"train_"+str(i)+"_next"#+str(t)
        d.plotMeshFromVoxels(np.array(Xt1), obj=sample_name)
1630     except:
        print("Example Object couldnt be generated with Marching Cube Algorithm.
    Probable Reason: Empty Input Tensor, All Zeros")
1632
    test_objects.append([Xt0[...,np.newaxis]]) # for validation of reconstruction
    during training, see generate_sample_objects():
1634     i+=1
```

```
# save some validation examples
i=0
valid_objects = []
for X in val_objects.take(N_SAMPLES):
    Xt0 = X[0][0]
    Xt1 = X[0][1]

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

    valid_objects.append([Xt0[...,np.newaxis]]) # for validation of reconstruction during training, see generate_sample_objects():
    i+=1

################################################################################
############## OPTIMIZER ########################################################
################################################################################
if CLR:
    steps_per_epoch = len(TRAINING_FILENAMES) // BATCH_SIZE
    clr_g = tfa.optimizers.CyclicalLearningRate(initial_learning_rate=G_LR,
        maximal_learning_rate=G_LR*10,
        scale_fn=lambda x: 1/(2.**(x-1)),
        step_size=2 * steps_per_epoch
    )
    clr_d = tfa.optimizers.CyclicalLearningRate(initial_learning_rate=D_LR,
        maximal_learning_rate=D_LR*10,
        scale_fn=lambda x: 1/(2.**(x-1)),
        step_size=2 * steps_per_epoch
    )

    encoder_optimizer = Adam(learning_rate=clr_g, beta_1=0.5, beta_2=0.999, epsilon=1e-07)
    generator_optimizer = Adam(learning_rate=clr_g, beta_1=0.5, beta_2=0.999, epsilon=1e-07)
    discriminator_optimizer = Adam(learning_rate=clr_d, beta_1=0.5, beta_2=0.999, epsilon=1e-07)

else:
    encoder_optimizer = Adam(learning_rate=G_LR, beta_1=0.5, beta_2=0.999, epsilon=1e-07)
    generator_optimizer = Adam(learning_rate=G_LR, beta_1=0.5, beta_2=0.999, epsilon=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):
    ############################################################################
```

```
    # START OF EPOCH
    t_epoch_start = time.time()
    n_batch=0
    ###################################
    for obj_batch in train_objects:
        # START OF BATCH
        batch_time = time.time()
        G_loss, D_loss = train_step(obj_batch)
        #Append arrays
        G_losses.append(G_loss[0])
        MSE_losses.append(G_loss[1])
        D_losses.append(-D_loss[0])

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

        JACCARD_loss = JACCARD_reconstruction_loss(obj_batch)
        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-1)//
BATCH_SIZE), total_iterations, time.time() - start_time, time.time() -
batch_time, -D_loss[0], G_loss[0], G_loss[1], JACCARD_loss))

        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")

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

            Val_losses = []
            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
Batch

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

```
        ###########################################################################
            # Output generated objects
            generate_sample_objects(test_objects, total_iterations, "train")
            generate_sample_objects(valid_objects, total_iterations, "valid")
            # # 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

            if KEEP_BEST_ONLY:
                if BEST_METRIC == "JACCARD":
                    BEST_LOSS_METRIC = JACCARD_val_loss
                    KEEP_LAST_N = 1
                elif BEST_METRIC == "MSE":
                    BEST_LOSS_METRIC = MSE_val_loss
                    KEEP_LAST_N = 1
                if len(BEST_LOSS_METRIC) > 1 and (BEST_LOSS_METRIC[-1] <= best_loss)
    :
                    best_loss = BEST_LOSS_METRIC[-1]
                    best_iteration = total_iterations
                    # Save models
                    encoder.save(model_directory+"_trained_encoder_"+"iter_"+str(
    total_iterations)+".h5")
                    generator.save(model_directory+"_trained_generator_"+"iter_"+str
    (total_iterations)+".h5")
            else:
                # Save models
                encoder.save(model_directory+"_trained_encoder_"+"iter_"+str(
    total_iterations)+".h5")
                generator.save(model_directory+"_trained_generator_"+"iter_"+str(
    total_iterations)+".h5")

            # # Delete Model, keep best or last N models
            IOhousekeeping(model_directory, KEEP_LAST_N, VAL_SAVE_FREQ,
    RESTART_TRAINING, total_iterations, KEEP_BEST_ONLY, best_iteration)

        ###########################################################################
        # SAVE CHECKPOINT FOR RESTART:
        if total_iterations % WRITE_RESTART_FREQ == 0:
            print("WRITE RESTART FILE...\n")
            # Save Checkpoint after every 100 epoch
            save_checkpoints(ckpt, model_directory)

        # END OF BATCH
        ################################
        # STOP TRAINING AFTER MAX DEFINED ITERATIONS ARE REACHED
        if total_iterations == MAX_ITERATIONS :
            break
    if total_iterations == MAX_ITERATIONS:
```

```
            break

        # Print Status at the end of the epoch
        dt_epoch =  time.time() - t_epoch_start
        n_objects = n_objects + np.ceil(training_set_size*(T//TINC-1)//BATCH_SIZE)*
        BATCH_SIZE
        print("\n----------------------------------------------------------------------")
        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")
        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.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
import numpy as np
1004
AUTOTUNE = tf.data.AUTOTUNE
1006
def decode_object(example):
1008     object4d = tf.io.decode_raw(example['object4d'], tf.uint8)
    labelBetti = tf.io.decode_raw(example['labelBetti'], tf.uint8)
1010     labelObjs = tf.io.decode_raw(example['labelObjs'], tf.uint8)

1012     object4d = tf.reshape(object4d, [128, 16, 16, 16])
    labelBetti = tf.reshape(labelBetti, [4])
1014     labelObjs = tf.reshape(labelObjs, [4])

1016     return object4d, labelBetti, labelObjs


1018
def read_tfrecord(example):
1020     global NORMALIZE
    global T
1022     global TINC

1024     object_feature_description = {
        'object4d': tf.io.FixedLenFeature([], tf.string),
1026         'labelBetti': tf.io.FixedLenFeature([], tf.string),
        'labelObjs': tf.io.FixedLenFeature([], tf.string),
1028     }

1030     example = tf.io.parse_single_example(example, object_feature_description)

1032     object4d, labelBetti, labelObjs = decode_object(example)

1034     CROP=(128-T)//2
    # Reshape into time slices of n time steps (axis=1)
1036     object4d = object4d[CROP:128-CROP:TINC]
    object4d = tf.reshape(object4d, [T//TINC, 16, 16, 16])
1038     rearranged_objs4d = []
    for t in range(object4d.shape[0]):
1040         if t == object4d.shape[0] - 1:
            break
1042         rearranged_objs4d.append(object4d[t])
        rearranged_objs4d.append(object4d[t+1])
1044
    object4d = tf.reshape(tf.stack(rearranged_objs4d), [T//TINC-1, 2, 16, 16, 16])
1046
```

```
    if NORMALIZE:
        object4d = tf.cast(object4d, tf.float32) - 0.5

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

def load_dataset(filenames):
    ignore_order = tf.data.Options()
    ignore_order.deterministic = False  # disable order, increase speed
    dataset = tf.data.TFRecordDataset(filenames, compression_type="GZIP", num_parallel_reads=AUTOTUNE)  # automatically interleaves reads from multiple files
    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)
    return dataset

def get_dataset(filenames: list, shuffle: bool, batch_size: int, t: int, tinc: int, normalize: bool):
    global NORMALIZE
    global T
    global TINC
    NORMALIZE = normalize
    TINC = tinc
    T = t
    dataset = load_dataset(filenames)
    dataset = dataset.unbatch()
    if shuffle:
        dataset = dataset.shuffle(len(filenames)*(T//TINC-1)//2) # ALL: len(filenames)*(T//TINC)
    if batch_size is not None:
        dataset = dataset.batch(batch_size, drop_remainder=True)
    dataset = dataset.prefetch(buffer_size=AUTOTUNE)
    #dataset = dataset.cache()
    return dataset


# FOR TESTING:
'''
folder_name="Z:/Master_Thesis/data/4D_128_32_topological_dataset_0-31_16_label8only"

FILENAMES = tf.io.gfile.glob(folder_name+"/*.tfrecords")[:7]
split_ind = int(1 * len(FILENAMES))
TRAINING_FILENAMES, VALID_FILENAMES = FILENAMES[:split_ind], FILENAMES[split_ind:]

#TEST_FILENAMES = tf.io.gfile.glob(GCS_PATH + "/tfrecords/test*.tfrec")
print("Train TFRecord Files:", len(TRAINING_FILENAMES))
print("Validation TFRecord Files:", len(VALID_FILENAMES))
#print("Test TFRecord Files:", len(TEST_FILENAMES))
```

```
BATCH_SIZE = 128

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)
        #print(object4d_i.shape, labelBetti)

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

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

B.3 Single Frame Input Prediction Algorithm

```
import os
import tensorflow as tf
from keras.models import load_model
import numpy as np
# OWN LIBRARIES
from utils import parse_tfrecords_4D_16_for_training as p4d
from utils import dataIO as d


base_dir = r"Z:\Master_Thesis\code\99_FINAL_MODELS_FOR_THESIS"

MODEL_LOCATION       = os.path.join(base_dir, "aencgan3d_16_T20_TINC1")

TEST_DATASET_LOC    = "Z:/Master_Thesis/data/02_4D_test_datasets"

T                    = int(MODEL_LOCATION[-8:-6])
TINC                 = int(MODEL_LOCATION[-1])
N_PREDICT            = 5

if not os.path.exists(os.path.join(base_dir,f'objects/T{T}_TINC{TINC}')):
    os.makedirs(os.path.join(base_dir,f'objects/T{T}_TINC{TINC}'))
################################################################################
############# GET Data/Objects ################################################
################################################################################
TEST_FILENAMES = tf.io.gfile.glob(os.path.join(TEST_DATASET_LOC,"*.tfrecords"))
test_set_size = len(TEST_FILENAMES)

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

# Convert/Decode dataset from tfrecords file for training
test_objects = p4d.get_dataset(TEST_FILENAMES, shuffle=True, batch_size=1, t=T, tinc
    =TINC, normalize=False)

encoder_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*encoder*.h5"))
encoder = load_model(encoder_model[-1])

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

def predictions(objects):
    n=0
    for X in objects:

        d.plotFromVoxels(np.squeeze(X[0,0]), f"{n}_T{T}_TINC{TINC}_t0")
```

```
        d.plotMeshFromVoxels(np.squeeze(X[0,0]), obj=f"objects/T{T}_TINC{TINC}/single_{n}_T{T}_TINC{TINC}_t0")
        d.plotFromVoxels(np.squeeze(X[0,1]), f"{n}_T{T}_TINC{TINC}_t1")
        d.plotMeshFromVoxels(np.squeeze(X[0,1]), obj=f"objects/T{T}_TINC{TINC}/single_{n}_T{T}_TINC{TINC}_t1")

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

        d.plotFromVoxels(pred_object, f"{n}_T{T}_TINC{TINC}_t1_pred")
        d.plotMeshFromVoxels(pred_object, obj=f"objects/T{T}_TINC{TINC}/single_{n}_T{T}_TINC{TINC}_t1_pred")
        if n==N_PREDICT:
            break
        n+=1


predictions(test_objects)
```

Listing B.3: Single Frame Input Prediction Algorithm

B.4 Single Frame Input Long-Term Prediction Algorithm

```
import os
import tensorflow as tf
from keras.models import load_model
import numpy as np
# OWN LIBRARIES
from utils import parse_tfrecords_4D_16_for_LongTerm_prediction as p4d
from utils import dataIO as d


base_dir = r"Z:\Master_Thesis\code\99_FINAL_MODELS_FOR_THESIS"

MODEL_LOCATION       = os.path.join(base_dir, "aencgan3d_16_T20_TINC1")

TEST_DATASET_LOC    = "Z:/Master_Thesis/data/02_4D_test_datasets"

T                    = int(MODEL_LOCATION[-8:-6])
TINC                 = int(MODEL_LOCATION[-1])
N_PREDICT            = 5
MAX_TIME             = 5

if not os.path.exists(os.path.join(base_dir,f'objects_longterm/T{T}_TINC{TINC}')):
    os.makedirs(os.path.join(base_dir,f'objects_longterm/T{T}_TINC{TINC}'))
################################################################################
############# GET Data/Objects ################################################
################################################################################
TEST_FILENAMES = tf.io.gfile.glob(os.path.join(TEST_DATASET_LOC,"*.tfrecords"))
test_set_size = len(TEST_FILENAMES)

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

# Convert/Decode dataset from tfrecords file for training
test_objects = p4d.get_dataset(TEST_FILENAMES, shuffle=False, batch_size=None, t=T,
    tinc=TINC, normalize=False)

encoder_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*encoder*.h5"))
encoder = load_model(encoder_model[-1])

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

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

```
            d.plotMeshFromVoxels(np.squeeze(X[0]), obj=f"objects_longterm/T{T}_TINC{TINC}/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}/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}/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}/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}/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}/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,...]


                # 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)

                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}/single_{n}_T{T}_TINC{TINC}_t{t+1}_pred")


                if t==MAX_TIME-1:
                    break

                X0 = pred_object[np.newaxis,...]


            n+=1
            if n==N_PREDICT:
                break


predictions(test_objects)
```

Listing B.4: Single Frame Input Prediction Algorithm

B.5 Multiple Frame Input Training Algorithm

```
import os
from shutil import copyfile
import glob
import sys
import numpy as np
#np.set_printoptions(threshold=sys.maxsize)
import time
import tensorflow as tf
from tensorflow.keras import Model
from tensorflow.keras.layers import Dense, BatchNormalization, LayerNormalization,
    LeakyReLU, Reshape, Conv3DTranspose, Conv3D, Flatten, ReLU, Dropout, Concatenate
    , Input, UpSampling3D, AveragePooling3D
from tensorflow.keras.activations import sigmoid, tanh
from tensorflow.keras.optimizers import Adam
import tensorflow_addons as tfa

# OWN LIBRARIES
from utils.loss_plotter import render_graphs
from utils import dataIO as d
from utils import parse_tfrecords_4D_16_t3_for_training as p4d
from utils.logger import Logger

################################################################################
########### INPUTS  ###################
########################################
N_EPOCHS            = 10000             # Default: 10000 - Max. training epochs OR
    Define Max. Iterations
MAX_ITERATIONS      = 20000             # Default: -1 - Max. Training Iterations (
    Max. Number of processed Batches) before training stops, if "-1" Training stops
    after reaching N_EPOCHS
TRAIN_DATASET_LOC   = "Z:/Master_Thesis/data/00_4D_training_datasets"
VALID_DATASET_LOC   = "Z:/Master_Thesis/data/01_4D_validation_datasets"
N_TRAINING          = 10000             # Number of Files for Training: -1 == USE
    ALL
N_VALIDATION        = 1000              # Number of Files for Validation: -1 == USE
    ALL
T                   = 20                # 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.
TINC                = 1                 # 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!)
1034 ##############################################################################
     # 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 #######################################
     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 ##############################################################################
     # 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
DROPOUT_RATE        = 0.2               # Default: 0.2
################################################################################
# weight_initializer & regularizer
weight_initializer  = "glorot_uniform"  # "glorot_uniform" "glorot_normal" or "
    he_uniform"
weight_regularizer  = "L1"              # tf.keras.regularizers.L1(l1=0.01)  #
    Default l1=0.01
################################################################################
# USE BIAS IN GENERATOR
USE_BIAS            = True
bias_initializer    = "zeros" # DEFAULT: "zeros" - "ones" or tf.keras.initializers.
    Constant(0.5)
################################################################################
# DEFAULT - DONT TOUCH SETTINGS
SIZE                = 16                # Default: 16 - Object Grid Size
BASE_SIZE           = 4                 # Default: 4 - Base Resolution to start from
     for generator and to end with for discriminator
KERNEL_SIZE         = 5                 # Default: 5 - Lower values learn pretty bad
    , higher than 5 doesnt change training much
################################################################################
################### NETWORKS ###################################################
################################################################################
# latent vector input
latent_vector = Input(shape=(Z_SIZE*3,))
#-------------------------------------------------------------------------------
################### ENCODER ####################################################
#-------------------------------------------------------------------------------
def obj_input(object_dim=(SIZE, SIZE, SIZE, 1)):
    obj_in = Input(shape=object_dim)
    return obj_in

################################################################################
# Encoder PreResidual Block
def EncoderPreResBlock(x, in_filters, out_filters, kernel_size, strides,
    kernel_initializer, kernel_regularizer, padding):
    # Define Shortcut
    if strides == 1 and in_filters == out_filters: # Identity Skip Connection
        shortcut = x
    elif strides == 2 and in_filters == out_filters: # Identity Skip Connection
        shortcut = AveragePooling3D(pool_size=(strides, strides, strides), strides=(
    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
```

```
1104            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)
1106
        # Define Activation on Input
1108    x = BatchNormalization()(x)
        x = LeakyReLU()(x)
1110    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)
1112
        return shortcut + x
1114 ################################################################################
    # DEFINE ENCODER NETWORK
1116 def ClassicEncoder():
        obj_in = obj_input()
1118
        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)
1120    x = BatchNormalization()(x)
        x = LeakyReLU()(x)
1122    x = Dropout(DROPOUT_RATE)(x)

1124    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)
1126    x = LeakyReLU()(x)
        x = Dropout(DROPOUT_RATE)(x)
1128
        x = Conv3D(filters=FEATUREMAP_BASE_DIM, kernel_size=KERNEL_SIZE, strides=1,
        kernel_initializer=weight_initializer, kernel_regularizer=weight_regularizer,
        padding="same")(x)
1130
        x = Flatten()(x)
1132
        out_layer = Dense(Z_SIZE, activation=None)(x)
1134
        # define model
1136    model = Model(obj_in, out_layer)
        return model
1138
    def PreResEncoder():
1140    obj_in = obj_input()

1142    # 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)

    # PreRes Blocks
    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")
    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")

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

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

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

    # define model
    model = Model(obj_in, out_layer)
    return model

##########################################################################################

#--------------------------------------------------------------------------
################### GENERATOR ############################################
#--------------------------------------------------------------------------

def latent_input(latent_vector=latent_vector, BASE_SIZE=BASE_SIZE):
    base = BASE_SIZE * BASE_SIZE * BASE_SIZE

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

##########################################################################################
# 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
    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)
    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)
    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)
        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)

    # Define Activation on Input
    x = BatchNormalization()(x)
    x = ReLU()(x)
    if up_size != 1:
        x = UpSampling3D(size=(up_size, up_size, up_size))(x)
    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)

    return shortcut + x
################################################################################
# DEFINE GENERATOR NETWORK
def ClassicGenerator():
    latent_in = latent_input()
    x = BatchNormalization()(latent_in)
    x = ReLU()(x)

    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)

    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)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    x = UpSampling3D(size=(2, 2, 2))(x)
    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 =
```

```
    bias_initializer)(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    x = Conv3DTranspose(filters=1, 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)

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

    model = Model(latent_vector, out_layer)

    return model

def PreResGenerator():
    x = latent_input()

    # PreRes Blocks
    x = GPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM    , out_filters=
    FEATUREMAP_BASE_DIM//2, kernel_size=KERNEL_SIZE, up_size=1, kernel_initializer=
    weight_initializer, kernel_regularizer=weight_regularizer, padding="same",
    use_bias=USE_BIAS, bias_initializer = bias_initializer)
    x = GPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//2, out_filters=
    FEATUREMAP_BASE_DIM//4, kernel_size=KERNEL_SIZE, up_size=2, kernel_initializer=
    weight_initializer, kernel_regularizer=weight_regularizer, padding="same",
    use_bias=USE_BIAS, bias_initializer = bias_initializer)
    x = GPreResBlock(x, in_filters=FEATUREMAP_BASE_DIM//4, out_filters=
    FEATUREMAP_BASE_DIM//8, kernel_size=KERNEL_SIZE, up_size=2, kernel_initializer=
    weight_initializer, kernel_regularizer=weight_regularizer, padding="same",
    use_bias=USE_BIAS, bias_initializer = bias_initializer)

    x = BatchNormalization()(x)
    x = ReLU()(x)
    # Final Convolution
    x = Conv3DTranspose(filters=1, 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)

    # End Activation
    if NORMALIZE:
        out_layer = tanh(x)
    else:
        out_layer = sigmoid(x)

    model = Model(latent_vector, out_layer)

    return model
```

```
1252
################################################################################
1254
#-------------------------------------------------------------------------------
1256 ################### DISCRIMINATOR ##############################################
#-------------------------------------------------------------------------------
1258
def real_obj_input_disriminator(object_dim=(SIZE, SIZE, SIZE, 1)):
1260     real_obj_in = Input(shape=object_dim)
    return real_obj_in
1262 def cond_obj_input_disriminator(object_dim=(SIZE, SIZE, SIZE, 1)):
    cond_obj_in = Input(shape=object_dim)
1264     return cond_obj_in

1266 ################################################################################
# Discriminator PreResidual Block
1268 def DPreResBlock(x, in_filters, out_filters, kernel_size, strides,
    kernel_initializer, kernel_regularizer, padding):
    # Define Shortcut
1270     if strides == 1 and in_filters == out_filters: # Identity Skip Connection
        shortcut = x
1272     elif strides == 2 and in_filters == out_filters: # Identity Skip Connection
        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
    Skip
        shortcut = Conv3D(filters=out_filters, kernel_size=1, strides=1,
    kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
    padding="same")(x)
1276     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)
1278         shortcut = Conv3D(filters=out_filters, kernel_size=1, strides=1,
    kernel_initializer=kernel_initializer, kernel_regularizer=weight_regularizer,
    padding="same")(shortcut)

1280     # Define Activation on Input
    x = LayerNormalization()(x)
1282     x = LeakyReLU()(x)
    x = Dropout(DROPOUT_RATE)(x)
1284     x = Conv3D(filters=out_filters, kernel_size=kernel_size, strides=strides,
    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_disriminator()
```

```
    cond_obj_in = cond_obj_input_disriminator()

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

    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)
    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 = LayerNormalization()(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(1, activation=None)(x)

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

def PreResDiscriminator():
    real_obj_in = real_obj_input_disriminator()
    cond_obj_in = cond_obj_input_disriminator()

    # merge label_conditioned_disriminator and latent_input output
    x = Concatenate()([real_obj_in, cond_obj_in])

    # 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")(x)

    # PreRes Blocks
    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")

    x = LayerNormalization()(x)
    x = LeakyReLU()(x)
    x = Dropout(DROPOUT_RATE)(x)

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

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

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

################################################################################
########## COMPILE NETWORKS ####################################################
################################################################################
if GD_ARCHITECTURE == "PreRes":
    encoder, generator, discriminator = PreResEncoder(), PreResGenerator(),
    PreResDiscriminator()
elif GD_ARCHITECTURE == "Classic":
    encoder, generator, discriminator = ClassicEncoder(), ClassicGenerator(),
    ClassicDiscriminator()
else:
    ValueError("Generator/Disriminator Architecture not Available. Choose between '
    PreRes' or 'Classic'!")
encoder.compile()
generator.compile()
discriminator.compile()

################################################################################
############## LOSS FUNCTION ###################################################
################################################################################
@tf.function
def encgen_loss(real_objects):

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

    # Encode
    latent_code_t0 = encoder(real_objects_t0)
    latent_code_t1 = encoder(real_objects_t1)
    latent_code_t2 = encoder(real_objects_t2)

```

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

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

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

    return loss, MSE_loss

@tf.function
def encdisc_loss(real_objects):

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

    # Encode
    latent_code_t0 = encoder(real_objects_t0)
    latent_code_t1 = encoder(real_objects_t1)
    latent_code_t2 = encoder(real_objects_t2)

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

    pred_objects_t3 = generator(latent_code, training=True)
    fake_logits = discriminator([pred_objects_t3, real_objects_t3], training=True)
    real_logits = discriminator([real_objects_t3, real_objects_t3], training=True)
    D_loss, fake_loss, real_loss, gp, ep = wgan_gp_discriminator_loss(real_logits,
    fake_logits, real_objects_t3, pred_objects_t3, real_objects_t3, GP_WEIGHT,
    EPS_WEIGHT)

    return D_loss, fake_loss, real_loss, gp, ep

@tf.function
def val_encdisc_loss(val_real_objects):

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

    # Encode
    val_latent_code_t0 = encoder(val_real_objects_t0)
    val_latent_code_t1 = encoder(val_real_objects_t1)
    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)

    val_pred_objects_t3 = generator(val_latent_code, training=False)
    val_fake_logits = discriminator([val_pred_objects_t3, val_real_objects_t3],
    training=False)
    val_real_logits = discriminator([val_real_objects_t3, val_real_objects_t3],
    training=False)
    D_loss, fake_loss, real_loss, gp, ep = wgan_gp_discriminator_loss(
    val_real_logits, val_fake_logits, val_real_objects_t3, val_pred_objects_t3,
    val_real_objects_t3, GP_WEIGHT, EPS_WEIGHT)

    return D_loss, fake_loss, real_loss, gp, ep

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

    This loss is calculated on an interpolated image
    and added to the discriminator loss.
    """
    # 1. Get the interpolated onject
    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)

    # 2. Calculate the gradients w.r.t to this interpolated object.
    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

    # 3. Calculate the norm of the gradients.
    gradient_penalty = tf.reduce_mean((norm-1.0)**2)

    # Add the gradient penalty to the original discriminator loss and MEAN over
    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)

    D_loss =  fake_loss - real_loss + gp + ep
```

```
1460        return D_loss, fake_loss, real_loss, gp, ep

1462    ################################################################################
        ############ TRAINING STEP ######################################################
1464    ################################################################################
        @tf.function
1466    def train_step(obj_batch):

1468        real_objects = obj_batch[...,np.newaxis]

1470        for i in range(D_STEPS):
                with tf.GradientTape() as disc_tape:
1472                # discriminator loss
                    EncDisc_loss = encdisc_loss(real_objects)
1474
                gradients_of_discriminator = disc_tape.gradient(EncDisc_loss[0],
            discriminator.trainable_variables)
1476            discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
            discriminator.trainable_variables))

1478        with tf.GradientTape() as gen_tape, tf.GradientTape() as enc_tape:
                # generator loss
1480            EncGen_loss = encgen_loss(real_objects)

1482        gradients_of_encoder = enc_tape.gradient(EncGen_loss[0], encoder.
            trainable_variables)
            encoder_optimizer.apply_gradients(zip(gradients_of_encoder, encoder.
            trainable_variables))
1484
            gradients_of_generator = gen_tape.gradient(EncGen_loss[0], generator.
            trainable_variables)
1486        generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.
            trainable_variables))

1488        return EncGen_loss, EncDisc_loss

1490    @tf.function
        def validation_step(val_obj_batch):
1492        val_real_objects = val_obj_batch[...,np.newaxis]

1494        Val_EncDisc_Loss = val_encdisc_loss(val_real_objects)

1496        Val_MSE_reconstruction_loss = MSE_reconstruction_loss(val_real_objects)

1498        return Val_EncDisc_Loss, Val_MSE_reconstruction_loss


1500
        @tf.function
1502    def MSE_reconstruction_loss(real_objects):
```

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

    # Encode
    latent_code_t0 = encoder(real_objects_t0)
    latent_code_t1 = encoder(real_objects_t1)
    latent_code_t2 = encoder(real_objects_t2)

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

    pred_objects_t3 = generator(latent_code, training=False)

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

    return MSE_loss

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

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

    # Encode
    val_latent_code_t0 = encoder(val_real_objects_t0)
    val_latent_code_t1 = encoder(val_real_objects_t1)
    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)
    #pred_objects_t1 = generator([latent_code, real_labels, real_objects_t0],
    training=False)
    val_pred_objects_t3 = generator(val_latent_code, training=False)

    if NORMALIZE:
        real = np.asarray(val_real_objects_t3>0.0, bool)
        pred = np.asarray(val_pred_objects_t3>0.0, bool)
    else:
        real = np.asarray(val_real_objects_t3>0.5, bool)
        pred = np.asarray(val_pred_objects_t3>0.5, bool)

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

    return JACCARD_loss

################################################################################
######### SOME UTILITY FUNCTIONS ###############################################
################################################################################
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,...])

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

        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)

        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.""")

        generated_objects.append(object_out)

        i+=1

    print()

    return generated_objects

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:
```

```
        if KEEP_BEST_ONLY:
            fileList_all = glob.glob(model_directory + "*_iter_*")
            fileList_best = glob.glob(model_directory + "*_iter_" + str(int(
    best_iteration)) +"*")
            fileList_del = [ele for ele in fileList_all if ele not in fileList_best]

        else:
            fileList_del = glob.glob(model_directory + "*_iter_" + str(int(
    total_iterations-KEEP_LAST_N*VAL_SAVE_FREQ)) +"*")

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

################################################################################
########### LAST PREPARATION STEPS BEFORE TRAINING STARTS ######################
################################################################################
#generate folders:
if RESTART_TRAINING:
    model_directory = os.getcwd() + "/" + RESTART_FOLDER_NAME + "/"
else:
    named_tuple = time.localtime() # get struct_time
    time_string = time.strftime("%Y%m%d_%H%M%S", named_tuple)
    model_directory = os.getcwd()+"/"+time_string+"_" + os.path.splitext(os.path.
    basename(__file__))[0] + "/"
    if not os.path.exists(model_directory):
        os.makedirs(model_directory)

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

# Activate Console Logging:
if LOGGING:
    sys.stdout = Logger(filename=model_directory+"log_out.txt")

# PRINT NETWORK SUMMARY
print(encoder.summary())
print(generator.summary())
print(discriminator.summary())

################################################################################
############# GET DATA/OBJECTS #################################################
################################################################################
TRAINING_FILENAMES = tf.io.gfile.glob(TRAIN_DATASET_LOC+"/*.tfrecords")[:N_TRAINING]
```

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

training_set_size = len(TRAINING_FILENAMES)
validation_set_size = len(VALIDATION_FILENAMES)
training_samples = training_set_size * (T//TINC-3) // BATCH_SIZE * BATCH_SIZE
validation_samples = validation_set_size * (T//TINC-3) // BATCH_SIZE * BATCH_SIZE
print()
print("Total Number of TFRecord Files to Train :", training_set_size)
print("Total Number of Samples to Train:", training_samples)
print()
print("Total Number of TFRecord Files for Validation:", validation_set_size)
print("Total Number of Samples for Validation:", validation_samples)
print()

# 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 )
val_objects = p4d.get_dataset(VALIDATION_FILENAMES, shuffle=False, batch_size=
    BATCH_SIZE, t=T, tinc=TINC, normalize=NORMALIZE )

# save some real examples
i=0
test_objects = []
for X in train_objects.take(N_SAMPLES):
    Xt0_2 = X[0][0:3]
    Xt3 = X[0][3]

    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")

    test_objects.append([Xt0_2[...,np.newaxis]]) # for validation of reconstruction
    during training, see generate_sample_objects():
    i+=1

# save some validation examples
i=0
valid_objects = []
for X in val_objects.take(N_SAMPLES):
    Xt0_2 = X[0][0:3]
    Xt3 = X[0][3]

    try:
        sample_name = model_directory+"valid_"+str(i)+"_init"#+str(t)
```

```
            d.plotMeshFromVoxels(np.array(Xt0_2[0]), obj=sample_name)
            sample_name = model_directory+"valid_"+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")

        valid_objects.append([Xt0_2[...,np.newaxis]]) # for validation of reconstruction
         during training, see generate_sample_objects():
        i+=1


################################################################################
############## OPTIMIZER ########################################################
################################################################################
if CLR:
    steps_per_epoch = len(TRAINING_FILENAMES) // BATCH_SIZE
    clr_g = tfa.optimizers.CyclicalLearningRate(initial_learning_rate=G_LR,
        maximal_learning_rate=G_LR*10,
        scale_fn=lambda x: 1/(2.**(x-1)),
        step_size=2 * steps_per_epoch
    )
    clr_d = tfa.optimizers.CyclicalLearningRate(initial_learning_rate=D_LR,
        maximal_learning_rate=D_LR*10,
        scale_fn=lambda x: 1/(2.**(x-1)),
        step_size=2 * steps_per_epoch
    )

    encoder_optimizer = Adam(learning_rate=clr_g, beta_1=0.5, beta_2=0.999, epsilon
    =1e-07)
    generator_optimizer = Adam(learning_rate=clr_g, beta_1=0.5, beta_2=0.999,
    epsilon=1e-07)
    discriminator_optimizer = Adam(learning_rate=clr_d, beta_1=0.5, beta_2=0.999,
    epsilon=1e-07)

else:
    encoder_optimizer = Adam(learning_rate=G_LR, beta_1=0.5, beta_2=0.999, epsilon=1
    e-07)
    generator_optimizer = Adam(learning_rate=G_LR, beta_1=0.5, beta_2=0.999, epsilon
    =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):
    ############################################################################
    # START OF EPOCH
    t_epoch_start = time.time()
    n_batch=0
    ###################################
    for obj_batch in train_objects:
        # START OF BATCH
        batch_time = time.time()
        G_loss, D_loss = train_step(obj_batch)
        #Append arrays
        G_losses.append(G_loss[0])
```

```
        MSE_losses.append(G_loss[1])
1776    D_losses.append(-D_loss[0])

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

1782    JACCARD_loss = JACCARD_reconstruction_loss(obj_batch)
        JACCARD_losses.append(JACCARD_loss)
1784    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)//
BATCH_SIZE), total_iterations, time.time() - start_time, time.time() -
batch_time,  -D_loss[0], G_loss[0], G_loss[1], JACCARD_loss))
1786
        if PRINT_D_LOSSES:
1788        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
 MODEL:
1792    if (total_iterations % VAL_SAVE_FREQ == 0) and VAL_SAVE_FREQ != -1:
            print("\nRUN VALIDATION, SAVE OBJECTS, MODEL AND PLOT LOSSES...")
1794
            Val_losses = []
1796        Val_MSE_losses = []
            Val_JACCARD_losses = []
1798        for val_obj_batch in val_objects:
                Val_loss, MSE_recon = validation_step(val_obj_batch)
1800            Val_losses.append(-Val_loss[0]) # Collect Val loss per Batch
                Val_MSE_losses.append(MSE_recon) # Collect Val loss per Batch
1802            JACCARD_recon = JACCARD_reconstruction_loss(val_obj_batch)
                Val_JACCARD_losses.append(JACCARD_recon) # Collect Val loss per
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
            JACCARD_val_loss.append(np.mean(Val_JACCARD_losses)) # Mean over batches
1808        val_iter_arr.append(total_iterations)
        ###########################################################################
1810        # Output generated objects
            generate_sample_objects(test_objects, total_iterations, "train")
1812        generate_sample_objects(valid_objects, total_iterations, "valid")
            # # Plot and Save Loss Curves
1814        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
```

```
            if KEEP_BEST_ONLY:
                if BEST_METRIC == "JACCARD":
                    BEST_LOSS_METRIC = JACCARD_val_loss
                    KEEP_LAST_N = 1
                elif BEST_METRIC == "MSE":
                    BEST_LOSS_METRIC = MSE_val_loss
                    KEEP_LAST_N = 1
                if len(BEST_LOSS_METRIC) > 1 and (BEST_LOSS_METRIC[-1] <= best_loss)
:
                    best_loss = BEST_LOSS_METRIC[-1]
                    best_iteration = total_iterations
                    # Save models
                    encoder.save(model_directory+"_trained_encoder_"+"iter_"+str(
total_iterations)+".h5")
                    generator.save(model_directory+"_trained_generator_"+"iter_"+str
(total_iterations)+".h5")
            else:
                # Save models
                encoder.save(model_directory+"_trained_encoder_"+"iter_"+str(
total_iterations)+".h5")
                generator.save(model_directory+"_trained_generator_"+"iter_"+str(
total_iterations)+".h5")

            # # Delete Model, keep best or last N models
            IOhousekeeping(model_directory, KEEP_LAST_N, VAL_SAVE_FREQ,
RESTART_TRAINING, total_iterations, KEEP_BEST_ONLY, best_iteration)

        ##########################################################################
        # SAVE CHECKPOINT FOR RESTART:
        if total_iterations % WRITE_RESTART_FREQ == 0:
            print("WRITE RESTART FILE...\n")
            # Save Checkpoint after every 100 epoch
            save_checkpoints(ckpt, model_directory)

        # END OF BATCH
        ################################
        # STOP TRAINING AFTER MAX DEFINED ITERATIONS ARE REACHED
        if total_iterations == MAX_ITERATIONS :
            break
    if total_iterations == MAX_ITERATIONS:
        break

    # Print Status at the end of the epoch
    dt_epoch =  time.time() - t_epoch_start
    n_objects = n_objects + np.ceil(training_set_size*(T//TINC-3)//BATCH_SIZE)*
BATCH_SIZE
    print("\n----------------------------------------------------------------")
    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")
```

```
        print("------------------------------------------------------------------\n\n"
        )
1858
        # END OF EPOCH
1860    ##########################################################################


1862
print("\n TRAINING DONE! \n")
1864 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

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '1'
import tensorflow as tf
import numpy as np

AUTOTUNE = tf.data.AUTOTUNE

def decode_object(example):
    object4d = tf.io.decode_raw(example['object4d'], tf.uint8)
    labelBetti = tf.io.decode_raw(example['labelBetti'], tf.uint8)
    labelObjs = tf.io.decode_raw(example['labelObjs'], tf.uint8)

    object4d = tf.reshape(object4d, [128, 16, 16, 16])
    labelBetti = tf.reshape(labelBetti, [4])
    labelObjs = tf.reshape(labelObjs, [4])

    return object4d, labelBetti, labelObjs


def read_tfrecord(example):
    global NORMALIZE
    global T
    global TINC

    object_feature_description = {
        'object4d': tf.io.FixedLenFeature([], tf.string),
        'labelBetti': tf.io.FixedLenFeature([], tf.string),
        'labelObjs': tf.io.FixedLenFeature([], tf.string),
    }

    example = tf.io.parse_single_example(example, object_feature_description)

    object4d, labelBetti, labelObjs = decode_object(example)

    CROP=(128-T)//2
    # Reshape into time slices of n time steps (axis=1)
    object4d = object4d[CROP:128-CROP:TINC]
    object4d = tf.reshape(object4d, [T//TINC, 16, 16, 16])
    rearranged_objs4d = []
    for t in range(object4d.shape[0]):
        if t == object4d.shape[0] - 3:
            break
        rearranged_objs4d.append(object4d[t])
        rearranged_objs4d.append(object4d[t+1])
        rearranged_objs4d.append(object4d[t+2])
        rearranged_objs4d.append(object4d[t+3])

```

```
    object4d = tf.reshape(tf.stack(rearranged_objs4d), [T//TINC-3, 4, 16, 16, 16])

    if NORMALIZE:
        object4d = tf.cast(object4d, tf.float32) - 0.5

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

def load_dataset(filenames):
    options = tf.data.Options()
    options.deterministic = False  # disable order, increase speed
    dataset = tf.data.TFRecordDataset(filenames, compression_type="GZIP",
    num_parallel_reads=AUTOTUNE)  # automatically interleaves reads from multiple
    files
    dataset = dataset.with_options(options)  # uses data as soon as it streams in,
    rather than in its original order
    dataset = dataset.map(read_tfrecord, num_parallel_calls=AUTOTUNE)
    return dataset

def get_dataset(filenames: list, shuffle: bool, batch_size: int, t: int, tinc: int,
    normalize: bool):
    global NORMALIZE
    global T
    global TINC
    NORMALIZE = normalize
    TINC = tinc
    T = t
    dataset = load_dataset(filenames)
    dataset = dataset.unbatch()
    if shuffle:
        dataset = dataset.shuffle(len(filenames)*(T//TINC-3)//2) # ALL: len(
    filenames)*(T//TINC)
    if batch_size is not None:
        dataset = dataset.batch(batch_size, drop_remainder=True)
    dataset = dataset.prefetch(buffer_size=AUTOTUNE)

    return dataset


# FOR TESTING:
'''
folder_name="Z:/Master_Thesis/data/4D_128_32_topological_dataset_0-31_16_label8only"

FILENAMES = tf.io.gfile.glob(folder_name+"/*.tfrecords")[:7]
split_ind = int(1 * len(FILENAMES))
TRAINING_FILENAMES, VALID_FILENAMES = FILENAMES[:split_ind], FILENAMES[split_ind:]

#TEST_FILENAMES = tf.io.gfile.glob(GCS_PATH + "/tfrecords/test*.tfrec")
print("Train TFRecord Files:", len(TRAINING_FILENAMES))
```

```
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)

        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

```
import os
import tensorflow as tf
from keras.models import load_model
import numpy as np
# OWN LIBRARIES
from utils import parse_tfrecords_4D_16_t3_for_training as p4d
from utils import dataIO as d


base_dir = r"Z:\Master_Thesis\code\99_FINAL_MODELS_FOR_THESIS"

MODEL_LOCATION       = os.path.join(base_dir, "aencgan3d_16_t3_T80_TINC4")

TEST_DATASET_LOC    = "Z:/Master_Thesis/data/02_4D_test_datasets"

T                    = int(MODEL_LOCATION[-8:-6])
TINC                 = int(MODEL_LOCATION[-1])
N_PREDICT            = 5

if not os.path.exists(os.path.join(base_dir,f'objects/T{T}_TINC{TINC}')):
    os.makedirs(os.path.join(base_dir,f'objects/T{T}_TINC{TINC}'))
################################################################################
############# GET Data/Objects ################################################
################################################################################
TEST_FILENAMES = tf.io.gfile.glob(os.path.join(TEST_DATASET_LOC,"*.tfrecords"))
test_set_size = len(TEST_FILENAMES)

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

# Convert/Decode dataset from tfrecords file for training
test_objects = p4d.get_dataset(TEST_FILENAMES, shuffle=True, batch_size=1, t=T, tinc
    =TINC, normalize=False)

encoder_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*encoder*.h5"))
encoder = load_model(encoder_model[-1])

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

def predictions(objects):
    n=0
    for X in objects:
        d.plotFromVoxels(np.squeeze(X[0,0]), f"{n}_T{T}_TINC{TINC}_t-1")
        d.plotMeshFromVoxels(np.squeeze(X[0,0]), obj=f"objects/T{T}_TINC{TINC}/
    multiple_{n}_T{T}_TINC{TINC}_t0")
```

```
            d.plotFromVoxels(np.squeeze(X[0,1]), f"{n}_T{T}_TINC{TINC}_t-2")
            d.plotMeshFromVoxels(np.squeeze(X[0,1]), obj=f"objects/T{T}_TINC{TINC}/multiple_{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/T{T}_TINC{TINC}/multiple_{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/T{T}_TINC{TINC}/multiple_{n}_T{T}_TINC{TINC}_t3")

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

            Xminus2 = tf.dtypes.cast(X[:,0], tf.float32)
            Xminus1 = tf.dtypes.cast(X[:,1], tf.float32)
            X0 = tf.dtypes.cast(X[:,2], tf.float32)
            latent_code_tminus2 = encoder(Xminus2)
            latent_code_tminus1 = encoder(Xminus1)
            latent_code_t0 = encoder(X0)
            latent_code = tf.concat([latent_code_tminus2, latent_code_tminus1, latent_code_t0], axis=1)
            pred_object = np.squeeze(generator(latent_code, training=False))
            pred_object = np.asarray(pred_object>0.5, bool)

            d.plotFromVoxels(pred_object, f"{n}_T{T}_TINC{TINC}_t1_pred")
            d.plotMeshFromVoxels(pred_object, obj=f"objects/T{T}_TINC{TINC}/multiple_{n}_T{T}_TINC{TINC}_t3_pred")
            n+=1
            if n==N_PREDICT:
                break


predictions(test_objects)
```

Listing B.7: Multiple Frame Input Prediction Algorithm

B.8 Multiple Frame Input Long-Term Prediction Algorithm

```
import os
import tensorflow as tf
from keras.models import load_model
import numpy as np
# OWN LIBRARIES
from utils import parse_tfrecords_4D_16_for_LongTerm_prediction as p4d
from utils import dataIO as d


base_dir = r"Z:\Master_Thesis\code\99_FINAL_MODELS_FOR_THESIS"

MODEL_LOCATION       = os.path.join(base_dir, "aencgan3d_16_t3_T20_TINC1")

TEST_DATASET_LOC    = "Z:/Master_Thesis/data/02_4D_test_datasets"

T                    = int(MODEL_LOCATION[-8:-6])
TINC                 = int(MODEL_LOCATION[-1])
N_PREDICT            = 5
MAX_TIME             = 5

if not os.path.exists(os.path.join(base_dir,f'objects_longterm/T{T}_TINC{TINC}')):
    os.makedirs(os.path.join(base_dir,f'objects_longterm/T{T}_TINC{TINC}'))
################################################################################
############# GET Data/Objects ################################################
################################################################################
TEST_FILENAMES = tf.io.gfile.glob(os.path.join(TEST_DATASET_LOC,"*.tfrecords"))
test_set_size = len(TEST_FILENAMES)

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

# Convert/Decode dataset from tfrecords file for training
test_objects = p4d.get_dataset(TEST_FILENAMES, shuffle=False, batch_size=None, t=T,
    tinc=TINC, normalize=False)

encoder_model = tf.io.gfile.glob(os.path.join(MODEL_LOCATION, "*encoder*.h5"))
encoder = load_model(encoder_model[-1])

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

def predictions(objects):
    n=0
    for X in objects:

        d.plotFromVoxels(np.squeeze(X[0]), f"{n}_T{T}_TINC{TINC}_t0")
```

```
        d.plotMeshFromVoxels(np.squeeze(X[0]), obj=f"objects_longterm/T{T}_TINC{TINC}/multiple_{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}/multiple_{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}/multiple_{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}/multiple_{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}/multiple_{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}/multiple_{n}_T{T}_TINC{TINC}_t5")
        d.plotFromVoxels(np.squeeze(X[6]), f"{n}_T{T}_TINC{TINC}_t5")
        d.plotMeshFromVoxels(np.squeeze(X[6]), obj=f"objects_longterm/T{T}_TINC{TINC}/multiple_{n}_T{T}_TINC{TINC}_t6")
        d.plotFromVoxels(np.squeeze(X[7]), f"{n}_T{T}_TINC{TINC}_t5")
        d.plotMeshFromVoxels(np.squeeze(X[7]), obj=f"objects_longterm/T{T}_TINC{TINC}/multiple_{n}_T{T}_TINC{TINC}_t7")
        #Loop over time per sample
        X = tf.dtypes.cast(X, tf.float32)
        for t in range(2,X.shape[1]-3):
            if t==2:
                X_minus2 = X[0][np.newaxis,...]
                X_minus1 = X[1][np.newaxis,...]
                X0 = X[2][np.newaxis,...]

            # Encode & Generate
            latent_code_tminus2 = encoder(X_minus2)
            latent_code_tminus1 = encoder(X_minus1)
            latent_code_t0 = encoder(X0)
            latent_code = tf.concat([latent_code_tminus2, latent_code_tminus1, latent_code_t0], axis=1)
            pred_object = np.squeeze(generator(latent_code, training=False))
            pred_object = np.asarray(pred_object>0.5, bool)

            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}/multiple_{n}_T{T}_TINC{TINC}_t{t+1}_pred")

            if t==MAX_TIME+2-1:
                break

            X_minus2 = X_minus1
            X_minus1 = X0
            X0 = pred_object[np.newaxis,...]
```

```
1086
            n+=1
1088        if n==N_PREDICT:
                break
1090

1092 predictions(test_objects)
```

Listing B.8: Multiple Frame Input Prediction Algorithm

B.9 Long-Term Prediction - Data Parser

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '1'
import tensorflow as tf
import numpy as np

AUTOTUNE = tf.data.AUTOTUNE

def decode_object(example):
    object4d = tf.io.decode_raw(example['object4d'], tf.uint8)
    labelBetti = tf.io.decode_raw(example['labelBetti'], tf.uint8)
    labelObjs = tf.io.decode_raw(example['labelObjs'], tf.uint8)

    object4d = tf.reshape(object4d, [128, 16, 16, 16])
    labelBetti = tf.reshape(labelBetti, [4])
    labelObjs = tf.reshape(labelObjs, [4])

    return object4d, labelBetti, labelObjs

def read_tfrecord(example):
    global NORMALIZE
    global T
    global TINC

    object_feature_description = {
        'object4d': tf.io.FixedLenFeature([], tf.string),
        'labelBetti': tf.io.FixedLenFeature([], tf.string),
        'labelObjs': tf.io.FixedLenFeature([], tf.string),
    }

    example = tf.io.parse_single_example(example, object_feature_description)

    object4d, labelBetti, labelObjs = decode_object(example)

    CROP=(128-T)//2
    # Reshape into time slices of n time steps (axis=1)
    object4d = object4d[CROP:128-CROP:TINC]
    object4d = tf.reshape(object4d, [T//TINC, 16, 16, 16])
    #object4d = tf.reshape(object4d, [128, 16, 16, 16])
    #print("Time_Points: ", object4d.shape[0])
    if NORMALIZE:
        object4d = tf.cast(object4d, tf.float32) - 0.5

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

def load_dataset(filenames):
    ignore_order = tf.data.Options()
    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
    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)
    return dataset

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


# FOR TESTING:
'''
folder_name="Z:/Master_Thesis/data/4D_128_32_topological_dataset_0-31_16_label8only"

FILENAMES = tf.io.gfile.glob(folder_name+"/*.tfrecords")[:1]

print("Train TFRecord Files:", len(FILENAMES))

train_dataset = get_dataset(FILENAMES, shuffle=False, batch_size=None, t=80, tinc=4,
     normalize=False)

import dataIO
for object4d, labelBetti in train_dataset:
    print(object4d.shape, labelBetti)

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

        dataIO.plotFromVoxels(object4d_i.numpy(), [i, labelBetti_i])
        i+=1
'''
```

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

C Utility Functions

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

```
import os
import numpy as np
import matplotlib.pyplot as plt

def render_graphs(save_dir, track_g_loss, track_d_loss, track_d_val_loss,
    track_MSE_train_loss, track_MSE_val_loss, track_JACCARD_train_loss,
    track_JACCARD_val_loss, iterations_arr, val_iterations_arr, restart_training):
    # try:
        if not os.path.exists(save_dir+'/plots/'):
            os.makedirs(save_dir+'/plots/')

        # SAVE HISTORY TO FILE:
        if restart_training:
            LOSSES = np.concatenate((np.asarray(iterations_arr)[...,np.newaxis], np.
    asarray(track_g_loss)[...,np.newaxis], np.asarray(track_d_loss)[...,np.newaxis],
     np.asarray(track_MSE_train_loss)[...,np.newaxis], np.asarray(
    track_JACCARD_train_loss)[...,np.newaxis]), axis=1)
            header = "TRAINING ITERATION,          G_LOSS,          D_LOSS,
      MSE_LOSS,      JACCARD_LOSS"
            np.savetxt(save_dir+'/plots/_train_losses_restart.txt', LOSSES, fmt='
    %20d, %16.5f, %16.5f, %16.5f, %16.5f', delimiter=',', header=header)

            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)
            header = "TRAINING ITERATION,        D_VAL_LOSS,      MSE_VAL_LOSS,
    JACCARD_VAL_LOSS"
            np.savetxt(save_dir+'/plots/_val_losses_restart.txt', VAL_LOSSES, fmt='
    %20d, %16.5f, %16.5f, %16.5f', delimiter=',', header=header)
        else:
            LOSSES = np.concatenate((np.asarray(iterations_arr)[...,np.newaxis], np.
    asarray(track_g_loss)[...,np.newaxis], np.asarray(track_d_loss)[...,np.newaxis],
     np.asarray(track_MSE_train_loss)[...,np.newaxis], np.asarray(
    track_JACCARD_train_loss)[...,np.newaxis]), axis=1)
            header = "TRAINING ITERATION,          G_LOSS,          D_LOSS,
      MSE_LOSS,      JACCARD_LOSS"
            np.savetxt(save_dir+'/plots/_train_losses.txt', LOSSES, fmt='%20d,
    %16.5f, %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    track_d_loss = np.ma.masked_invalid(track_d_loss)#.compressed()
    track_g_loss = np.ma.masked_invalid(track_g_loss)#.compressed()
1030    high_d = np.percentile(track_d_loss, 99)
    high_g = np.percentile(track_g_loss, 99)
1032    low_d = np.percentile(track_d_loss, 1)
    low_g = np.percentile(track_g_loss, 1)
1034    if (len(track_d_val_loss) > 0) and (len(val_iterations_arr) > 0):
        track_d_val_loss = np.ma.masked_invalid(track_d_val_loss)#.compressed()
1036        high_d_val = np.percentile(track_d_val_loss, 95)
        low_d_val = np.percentile(track_d_val_loss, 5)
1038        high_y = max([high_d, high_g, high_d_val])
        low_y = min([low_d, low_g, low_d_val]) #- 0.5*min([high_d, high_g])
1040    else:
        high_y = max([high_d, high_g])
1042        low_y = min([low_d, low_g]) #- 0.5*min([high_d, high_g])


1044

    fig = plt.figure()
1046    plt.plot(iterations_arr, track_g_loss, color='crimson', alpha=0.5, label='G-
    loss')
    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):
        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()
    plt.title("LOSS")
1056    plt.xlabel('Training Iteration (processed batches)')
    plt.ylabel('Loss')
1058    plt.ylim([low_y - np.abs(0.5*low_y), high_y + np.abs(0.5*high_y)])
    plt.grid(True)
1060    if restart_training:
        plt.savefig(save_dir+'/plots/DISCGEN_LOSS_restart.png' )
```

```
        else:
            plt.savefig(save_dir+'/plots/DISCGEN_LOSS.png')
        plt.clf()
        plt.close(fig)

        if (len(track_MSE_val_loss) > 0) and (len(val_iterations_arr) > 0):
            fig = plt.figure()
            plt.plot(iterations_arr, track_MSE_train_loss, color='seagreen', alpha=0.5, label='MSE-Train-loss')
            plt.plot(val_iterations_arr, track_MSE_val_loss, color='navy', alpha=0.5, label='MSE-Val-loss')
            high_MSE_train = np.percentile(track_MSE_train_loss, 95)
            low_MSE_train = np.percentile(track_MSE_train_loss, 5)
            high_MSE_val = np.percentile(track_MSE_val_loss, 95)
            low_MSE_val = np.percentile(track_MSE_val_loss, 5)

            high_y = max([high_MSE_train, high_MSE_val])
            low_y = min([low_MSE_train, low_MSE_val])

            plt.legend()
            plt.title("MSE")
            plt.xlabel('Training Iteration (processed batches)')
            plt.ylabel('Loss')
            plt.ylim([low_y - np.abs(0.5*low_y), high_y + np.abs(0.5*high_y)])
            plt.grid(True)
            if restart_training:
                plt.savefig(save_dir+'/plots/MSE_LOSS_restart.png')
            else:
                plt.savefig(save_dir+'/plots/MSE_LOSS.png')
            plt.clf()
            plt.close(fig)

        if (len(track_JACCARD_val_loss) > 0) and (len(val_iterations_arr) > 0):
            fig = plt.figure()
            plt.plot(iterations_arr, track_JACCARD_train_loss, color='seagreen', alpha=0.5, label='JACCARD-Train-loss')
            plt.plot(val_iterations_arr, track_JACCARD_val_loss, color='navy', alpha=0.5, label='JACCARD-Val-loss')
            high_JACCARD_train = np.percentile(track_JACCARD_train_loss, 95)
            low_JACCARD_train = np.percentile(track_JACCARD_train_loss, 5)
            high_JACCARD_val = np.percentile(track_JACCARD_val_loss, 95)
            low_JACCARD_val = np.percentile(track_JACCARD_val_loss, 5)

            high_y = max([high_JACCARD_train, high_JACCARD_val])
            low_y = min([low_JACCARD_train, low_JACCARD_val])

            plt.legend()
            plt.title("JACCARD")
            plt.xlabel('Training Iteration (processed batches)')
            plt.ylabel('Loss')
```

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

```
"""
code from: https://github.com/meetps/tf-3dgan/tree/master
"""

import sys
import os

import scipy.ndimage as nd
import scipy.io as io
import numpy as np
import matplotlib.pyplot as plt
import skimage.measure as sk

from mpl_toolkits import mplot3d


import trimesh
from stl import mesh


def getVF(path):
    raw_data = tuple(open(path, 'r'))
    header = raw_data[1].split()
    n_vertices = int(header[0])
    n_faces = int(header[1])
    vertices = np.asarray([map(float,raw_data[i+2].split()) for i in range(
    n_vertices)])
    faces = np.asarray([map(int,raw_data[i+2+n_vertices].split()) for i in range(
    n_faces)])
    return vertices, faces

def plotFromVF(vertices, faces, obj):
    input_vec = mesh.Mesh(np.zeros(faces.shape[0], dtype=mesh.Mesh.dtype))
    for i, f in enumerate(faces):
        for j in range(3):
            input_vec.vectors[i][j] = vertices[f[j],:]
    '''
    figure = plt.figure()
    axes = mplot3d.Axes3D(figure)
    axes.add_collection3d(mplot3d.art3d.Poly3DCollection(input_vec.vectors))
    scale = input_vec.points.flatten('C')
    axes.auto_scale_xyz(scale, scale, scale)
    plt.show()
    '''
    input_vec.save(obj+'.stl')
```

```
1044 def plotFromVoxels(voxels, i=None):
     #print(voxels.nonzero().shape)
1046     z,x,y = voxels.nonzero()
     fig = plt.figure()
1048     ax = fig.add_subplot(111, projection='3d')
     ax.scatter(x, y, z, zdir='z', c= 'black', marker='s')
1050     #ax.voxels(voxels, facecolors="black", edgecolor='k')
     ax.set_xlim(0, voxels.shape[0])
1052     ax.set_ylim(0, voxels.shape[1])
     ax.set_zlim(0, voxels.shape[2])
1054     if i is not None:
         ax.set_title(i)
1056     plt.show()

1058 def getVFByMarchingCubes(voxels):
     v, f, normals, values =  sk.marching_cubes(voxels,
1060                                                level=0.01,
                                                   spacing=(1, 1, 1),
1062                                                gradient_direction='descent',
                                                   step_size=1,
1064                                                allow_degenerate=False,
                                                   method='lewiner', mask=None)
1066     return v, f

1068 def plotMeshFromVoxels(voxels, obj='toilet'):
     v,f = getVFByMarchingCubes(voxels)
1070     plotFromVF(v,f,obj)

1072 def plotVoxelVisdom(voxels, visdom, title):
     v, f = getVFByMarchingCubes(voxels)
1074     visdom.mesh(X=v, Y=f, opts=dict(opacity=0.5, title=title))

1076 def plotFromVertices(vertices):
     figure = plt.figure()
1078     axes = mplot3d.Axes3D(figure)
     axes.scatter(vertices.T[0,:],vertices.T[1,:],vertices.T[2,:])
1080     plt.show()

1082 def getVolumeFromOFF(path, sideLen=32):
     mesh = trimesh.load(path)
1084     volume = trimesh.voxel.Voxel(mesh, 0.5).raw
     (x, y, z) = map(float, volume.shape)
1086     volume = nd.zoom(volume.astype(float),
                      (sideLen/x, sideLen/y, sideLen/z),
1088                   order=1,
                      mode='nearest')
1090     volume[np.nonzero(volume)] = 1.0
     return volume.astype(np.bool)
1092
```

```
def getVoxelFromMat(path, cube_len=64):
    voxels = io.loadmat(path)['instance']
    voxels = np.pad(voxels,(1,1),'constant',constant_values=(0,0))
    if cube_len != 32 and cube_len == 64:
        voxels = nd.zoom(voxels, (2,2,2), mode='constant', order=0)
    if cube_len != 32 and cube_len == 128:
        voxels = nd.zoom(voxels, (4,4,4), mode='constant', order=0)
    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):
    def __init__(self, filename="Default.log"):
1004         self.terminal = sys.stdout
             self.log = open(filename, "w")
1006
    def write(self, message):
1008         self.terminal.write(message)
             self.log.write(message)
1010
    def flush(self):
1012         # this flush method is needed for python 3 compatibility.
             # this handles the flush command by doing nothing.
1014         # you might want to specify some extra behavior here.
             pass
```

Listing C.3: Logger Class

C.4 *.npz to *.tfrecords Converter

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

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

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

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]))

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

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))

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")
     # Expandd 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 Interpretes 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)
    return tf.cast(pooled_object4d, tf.uint8).numpy()

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

    # tbegin = time.time()
    ######################################################################

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

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

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

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

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

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

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

```

```
    ########################################################################
    # 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/4D_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

```
import os
import shutil
import numpy as np
import tensorflow as tf

################################################################################
########### INPUTS  ############################################################
################################################################################
DATASET_LOCATION              = "Z:/Master_Thesis/data/4D_128_ALL_16/*"   # 4D_128_32_topological_dataset_0-31_16_label8only  OR 4D_128_ALL_16/*
TRAIN_VAL_TEST_RATIO          = 0.8                                       # Default: 0.8 - Fraction of objects of full dataset to be used for training, rest is equaly split into Validation and Test Set
TRAINING_DATASET_FOLDER       = "00_4D_training_datasets"                 # FOLDER TO SAVE THE TRAINING FILES TO
VALIDATION_DATASET_FOLDER     = "01_4D_validation_datasets"               # FOLDER TO SAVE THE VALIDATION FILES TO
TEST_DATASET_FOLDER           = "02_4D_test_datasets"                     # FOLDER TO SAVE THE TEST FILES TO
################################################################################
############# GET DATA/OBJECTS #################################################
################################################################################
#read all files from folder
FILENAMES = tf.io.gfile.glob(DATASET_LOCATION+"/*.tfrecords")
FILENAMES_SHUFFLED = np.random.shuffle(FILENAMES)

# Split into Train, Validation and Test Sets
split_int_train = int(TRAIN_VAL_TEST_RATIO * len(FILENAMES))
split_int_valid_test = split_int_train + int(np.round((1-TRAIN_VAL_TEST_RATIO) / 2 * len(FILENAMES)))

TRAINING_FILENAMES = FILENAMES[:split_int_train]
VALIDATION_FILENAMES = FILENAMES[split_int_train:split_int_valid_test]
TEST_FILENAMES = FILENAMES[split_int_valid_test:]

training_set_size = len(TRAINING_FILENAMES)
validation_set_size = len(VALIDATION_FILENAMES)
test_set_size = len(TEST_FILENAMES)
print("Total Number of TFRecord Files to Train :", training_set_size)
print("Total Number of TFRecord Files for Validation:", validation_set_size)
print("Total Number of TFRecord Files for Testing:", test_set_size)


# Generate Folder with Training Files
if not os.path.exists(TRAINING_DATASET_FOLDER):
    os.makedirs(TRAINING_DATASET_FOLDER)
else:
```

```
1040        shutil.rmtree(TRAINING_DATASET_FOLDER)
            os.makedirs(TRAINING_DATASET_FOLDER)
1042
    for file_name in TRAINING_FILENAMES:
1044        if os.path.isfile(file_name):
                shutil.copy(file_name, TRAINING_DATASET_FOLDER)
1046
1048# Generate Folder with Validation Files
    if not os.path.exists(VALIDATION_DATASET_FOLDER):
1050        os.makedirs(VALIDATION_DATASET_FOLDER)
    else:
1052        shutil.rmtree(VALIDATION_DATASET_FOLDER)
            os.makedirs(VALIDATION_DATASET_FOLDER)
1054
    for file_name in VALIDATION_FILENAMES:
1056        if os.path.isfile(file_name):
                shutil.copy(file_name, VALIDATION_DATASET_FOLDER)
1058
1060# Generate Folder with Test Files
    if not os.path.exists(TEST_DATASET_FOLDER):
1062        os.makedirs(TEST_DATASET_FOLDER)
    else:
1064        shutil.rmtree(TEST_DATASET_FOLDER)
            os.makedirs(TEST_DATASET_FOLDER)
1066
    for file_name in TEST_FILENAMES:
1068        if os.path.isfile(file_name):
                shutil.copy(file_name, TEST_DATASET_FOLDER)
```

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