Thesis Dissertation

SINGLE-MOTION GENERATIVE MODELS FOR MOTION SUMMARIZATION

Christoforos Nicolaou

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

May 2023

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

Single-Motion Generative Models for Motion Summarization

Christoforos Nicolaou

Supervisor

Dr. Andreas Aristidou

Thesis submitted in partial fulfilment of the requirements for the award of degree of Bachelor in Computer Science at University of Cyprus

May 2023

Acknowledgements

I would like to express my sincere gratitude to my advisor Assistant Professor Andreas Aristidou, for his invaluable guidance and support throughout this project. His expertise and availability were crucial in helping me conduct this study and develop my skills.

I would also like to thank Pieris Panagi, for his guidance at the beginning of the project and his willingness to help me every other time I needed his assistance.

Finally, I would like to thank my family and friends for their support and patience during my studies and particularly during the conduction of this dissertation.

Abstract

Motion capture has been shown to be an effective means of digitizing dynamic movements. However, modifying the motion sequence to achieve a different duration is challenging, time-consuming, and inefficient. To solve this challenge, this dissertation proposes the application of image processing techniques in the field of computer animation and utilizes two recently developed machine learning techniques, Generative Adversarial Networks (GANs) and Denoising Diffusion Models (DDMs), to develop two unsupervised generative models for motion summarization and expansion. A GAN-based model that can modify the duration of a motion sequence and can be expanded to maintain its contextual structure and coherence, has been developed, along with the foundations of a DDM-based model that can summarize and expand motion sequences. Although both models have limitations, this study shows that they are a promising direction for further research in motion summarization. The proposed models have significant implications for a range of applications, from video games to movies, and could greatly enhance the efficiency and effectiveness of motion capture.

Contents

Chapter 1

Introduction

1.1 The Problem

Nowadays, motion capture has been proven to be a very effective way to capture and digitize dynamic movements. This technology has been used as means to produce some of the best and critically acclaimed works, with a wide range of applications spanning from the video game industry to the movie industry. Motion capture can be used to bring digital characters and creatures to life, and even create autonomous characters with realistic movements. It can even be used to populate digital cities in virtual reality environments and create digital crowds. One useful feature of this technology would be the ability to summarize a captured motion - that is, expanding a small motion to reach a larger temporal size, and the opposite, fitting a larger sequence into a smaller one. This feature has numerous use cases, such as extracting and replaying the most important parts in a sport or dance sequence. Our goal was to find a solution for summarizing or expanding a given motion while maintaining its temporal distribution, content, and coherence intact. However, manually editing the captured raw data to achieve this is extremely challenging, while capturing a new animation is costly. Additionally, the limited existing ways of modifying a motion sequence are both time-consuming and inefficient. Finally, it is difficult to achieve consistent temporality and content without changing the speed of the animation.

1.2 Motivation

That was the case until recent developments in content analysis and the evolution of the machine learning field, where the ability to summarize or expand a captured motion in post processing was almost impossible. With the introduction of generative models, particularly Generative Adversarial Networks (GANs) and Denoising Diffusion Models (DDMs), this is now possible, as these models can be trained to learn enough information about the distribution of a dataset and further produce realistic samples that follow that distribution. Therefore, we aim to present a generative model that can summarize or expand a given motion while maintaining its distribution intact and coherent. Furthermore, by leveraging the extensive use of these models in the image processing field and their ability to train on a single image, we can expand them in the field of character animation, addressing the issue of the lack of large datasets for specific types of movements. Specifically, we aim to create a model that can achieve the following:

1. Temporally alter the duration of a given animation file, by either increasing or decreasing its length to a specified duration, while keeping the contextual structure intact. In other words, the frequency of occurrence of specific movements should remain the same. It should be noted that simply cropping the animation itself is not a viable solution, as no significant movement should be cut out. For example, if the end of a dance performance is truncated, the movement would appear to end abruptly, and the dance will remain unfinished.

2. Identify the appropriate parts of the motion, which we can refer to as motion words, that we would like to add or remove. Specifically, a motion consists of both unique and common parts. Common parts typically occur in any arbitrary movement, whereas unique parts characterize a specific movement and distinguish it from others. For instance, in a dance routine where the dancer performs several intricate figures, those figures constitute the unique parts of the movement, while the simpler parts are considered to be the common ones. If we intended to shorten the dance duration, it would not be ideal to remove the complex figures. However, if the intention was to extend the duration of the dance, it would not be realistic for the dancer to continuously execute the complex figures throughout the routine. In fact, during summarization (shortening of the animation duration) we would want to remove the common parts and retain the unique ones, while during expansion (widening the

animation duration) we would still want to retain the unique parts but increase the duration of the common ones.

In conclusion, our goal is to develop a model that can identify both the unique and common motion words, and then modify their duration while preserving their coherence. By keeping the unique parts intact and adjusting the duration of the common parts, we can both shorten and expand the animation without compromising its artistic and expressive qualities.

1.3 Related Work

There exist several methods that try to achieve our goal, most of which are inspired by image processing techniques. The simplest method is the frame removal technique, in which random frames from the animation are removed. This method results in a shorter output motion, which however, is played faster since the speed has changed. Another method is inspired by the seam carving technique for image processing, in which the animation is treated as an image where “seams” of pixels are selected and removed. These “seams” are in fact horizontal or vertical paths of pixels spanning across the whole image, which when removed provide unnoticeable results [4]. However, as this method results again in a shorter motion, the movement is faster and not coherent. Both aforementioned methods do not allow control over which parts of the motion are removed. Another approach is to manually divide the motion sequence into a set of smaller motion sequences. These small sequences are then reunited together, by blending nearby ones with smaller movements to make the result coherent and removing small parts if needed to shorten the duration. However, this approach does not provide smooth and temporally consistent results, as it cannot handle big changes in the animation duration or in joint rotations of the skeleton. More recent methods utilize generative models for image scaling. GANimator (2022), for example, inspired by the image scaling model SinGAN, is one of the first methods that use this approach [10]. GANimator succeeds in generating high level animations from a single motion sequence, but has two major weaknesses. Firstly, there is no way to control the contextual structure of the generated motion. Secondly, there is no control over the translation of the generated motion. Therefore, while there are several methods proposed for motion summarization, generative model-based approaches like GANimator show promising results and offer a new direction for further research in the field.

1.4 Our Approach

Taking these approaches into account and inspired by image processing techniques, we adapted two well-known image processing techniques for image scaling into the field of computer animation, ensuring high level control of the distribution of motion words and applying constraints to the creation of movement translation. We ended up utilizing generative networks with progressive training to develop two solutions for tackling the motion summarization problem. The first method, inspired by InGAN, an Internal Generative Adversarial Network model for image retargeting [12], was initiated by Pieris Panagi during the continuation of his dissertation over the summer of 2022. This approach succeeded in generating high-quality movements with minor inconsistencies. The second method, inspired by SinDDM, a Single Image Denoising Diffusion Model for image synthesis and manipulation [9], was developed during the conduction of this dissertation, with the intention of testing the abilities of diffusion models in the field of character animation and obtaining better results than our first approach. In the end, we managed to lay the foundations for the development of a diffusion model for motion summarization and expansion.

1.5 Contributions

In the context of the research we conducted, we laid the foundations for the development of a network with the ability to summarize or expand a motion based on a single training file, while maintaining its temporal distribution, content, and coherence intact. More importantly, we were able to methodologically extend single-generative models in the field of computer animation so that movement content can be controlled at a higher level, thus making an important scientific contribution. Our major contribution is therefore the creation of a generative model able to learn enough information from a single motion, and then summarize it or expand it to match a desired duration, providing a way to control the generated content. Once finalized, our network can be used in the movie and video game industries for creating shorter versions of captured animations, in order to fit them to a certain script or scenario, or even generate expansions from an animation of limited duration, thus making a major technical contribution. In addition, our network has a significant application in that it can emphasize motion sequences and create highlights, for example in sports or dance, without the need to manually select the motion's key components and risking its coherence.

Chapter 2

Related Work

2.1 Traditional Image Scaling Techniques

Image scaling significantly contributed to our approach for tackling the motion summarization problem. It has been used for years since the size or aspect ratio of an image must often be changed for it to fit to different displays or to generally meet the requirements of a specific use. Before the introduction of deep learning, methods used for image scaling consisted of various interpolation techniques and cropping. Standard image scaling methods, though, do not take the image content into account, while cropping limits the result by only removing peripheral pixels. A method known as seam carving (Figure 2.1) was introduced as a more mature approach, supporting content-aware image resizing for both reduction and expansion. Seam carving introduces the concept of seams, described as “optimal 8-connected path of pixels on a single image from top to bottom, or left to right, where optimality is defined by an image energy function”. An image's aspect ratio can be altered by continuously carving seams out of it or inserting them in a certain direction. The importance of pixels is

defined using an energy function, and low energy pixels are used for connecting the pixels in a seam, spanning from top to bottom or left to right. In order to reduce the image size, more low energy pixels are removed, preserving the image structure. To enlarge the image, the original image content and the artificially inserted pixels must be balanced by the order of seam insertion. The selection of seams based on the energy function protects the image content from being distorted or removed, resulting in a content-aware resizing of images [4].

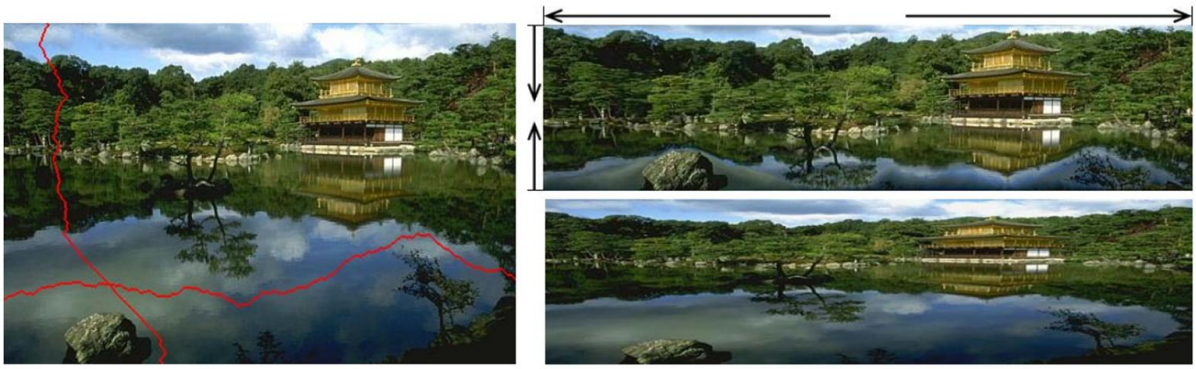

Figure 2.1 *Left: Selection of a horizontal and a vertical seam using the seam carving technique. Right: The results of scaling the image in both axes using seam carving (top) and basic interpolation techniques (bottom). Source: [4].*

This technique inspired our first two approaches. For the first approach, being framerate reduction/frame removal, we randomly selected different frames from the input motion and removed them to reduce the motion's duration to a specified length. To increase the motion's duration, the selected frames were duplicated instead of being removed. For the second approach, we treated the animation file as an image and parsed it into a 2-dimensional matrix of size [3J + 3, N], where J is the number of joints in the skeleton and N is the number of frames in the motion. In this matrix, the y-dimension represents the joint rotations in Euler angles and root translation of the skeleton, and the x-dimension represents the frames of the animation. The length of the image in the x-axis is none other than the actual duration of the animation. Therefore, by reducing or enlarging the image length in the x-axis, the animation duration is altered as intended. By applying the seam carving technique and removing vertical seams from the matrix, the length in the x-dimension was reduced, thus resulting in shorter duration. By inserting vertical seams, the length in the x-dimension was increased, resulting in a longer duration. The results, though, were far from satisfying. While both framerate reduction and seam carving keep the content intact and maintain correct temporal order, they cause the output motion to move much faster during summarization and much slower during expansion, providing no control over the removed motion parts. Additionally, the animation produced from the seam carving technique loses its coherence and sometimes produces abnormal movements. Therefore, turning to a generative model-based approach was an essential next step for overcoming these limitations.

2.2 Generative Models

In recent years, generative models have made impressive progress in their ability to generate realistic fake data based on real samples, with numerous applications spanning across a variety of fields, such as image, audio, and video synthesis and manipulation [7, 9]. In the image processing field, after typically learning a distribution of images in a large dataset, generative models are then able to generate new images from that distribution [12]. Modern generative models such as Generative Adversarial Networks (GANs) and Denoising Diffusion Models (DDMs) are now able to treat arbitrary image dimensions, achieve high levels of photorealism, train on sophisticated text prompts, and work with a wide range of image restoration and alteration tasks. The latter have shown astounding improvements in image generation, editing and reproduction tasks [9]. Nevertheless, both GANs and DDMs have demonstrated incredible potential regarding image resizing, especially when it comes to training on a single image file [9, 11, 12]. The potential integration of these capabilities in the field of computer animation is where our inspiration comes from. Both methods will now be explained in further detail.

2.2.1 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are a type of unsupervised machine learning model that falls under the category of deep generative models. They consist of two neural networks that compete as an adversary: a generator and a discriminator. The generator network learns to produce images from random noise, while the discriminator network learns to determine whether a sample image belongs to the dataset of real images or is generated by the generator (Figure 2.2). The two networks are simultaneously trained in an adversarial fashion, with the generator trying to fool the discriminator as much as possible. As the networks train, the generator gradually learns to produce increasingly realistic images, while the discriminator becomes better at distinguishing real images from fake ones [5].

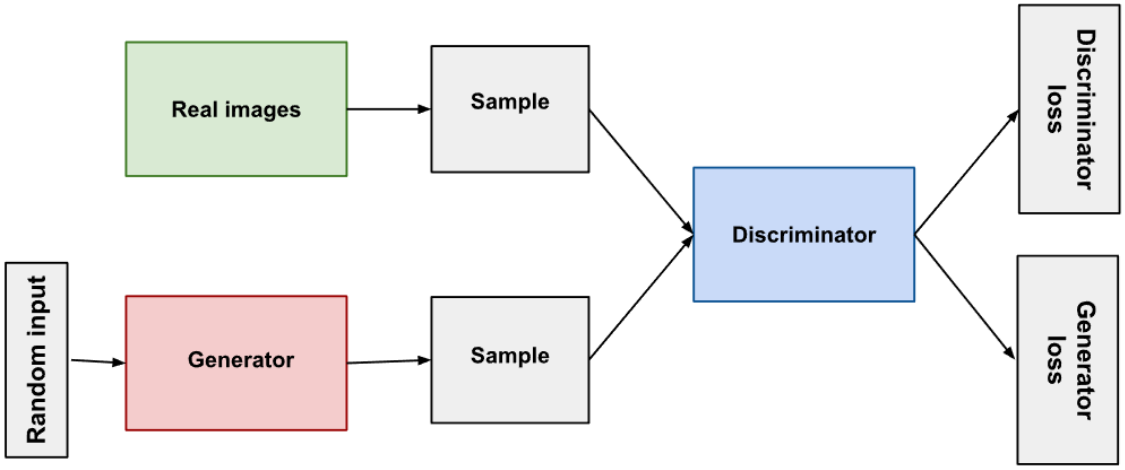

Figure 2.2 *A basic GAN architecture. The generator's output is directly connected to the discriminator's input. The discriminator tries to classify the input as real or fake and produces two losses that are used to train both itself and the generator. Source: https://developers.google.com/machine-learning/gan/gan_structure.*

GANs have gained popularity for their ability to generate high-quality realistic images of various types, including realistic faces, landscapes and even artwork. More recently, they have been used for retargeting/resizing images in numerous works, as they are capable of training using only a single image's internal statistics, captured by its distribution of patches, unlocking many new potential applications [11, 12].

2.2.2 Diffusion Models (DMs)

Denoising Diffusion Probabilistic Models (DDPMs) or Diffusion Models (DMs) are a class of generative models, introduced as an alternative to GANs, for generating high-quality images with less distortion [7, 9]. Inspired by nonequilibrium thermodynamics, DMs learn a distribution over images by modeling the diffusion process, which is the process of iteratively adding noise to an image until the final image is obtained (Figure 2.3). At each step of the diffusion process, the image is perturbed with a noise of increasing variance from the conditional probability distribution q, which models the amount of noise to be added to the image at each step. This is known as the forward diffusion process. After adding the noise, the image is then transformed to a new image by applying a diffusion operator p, which models how the image transforms over time and is often designed as a Markov chain, a type of stochastic process that models the probability of transitioning from one state to another. This is known as the reverse diffusion process. At the end of the diffusion process, the resulting image is a sample from the distribution p at the final step. Using a dataset of images, the conditional probability distributions q and p are learned during the DM's training, whose goal is to maximize the likelihood that the training data will be distributed in the same way as

the learned distributions, in other words, the produced images are as similar to the training images as possible [7].

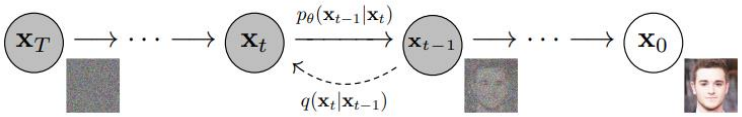

Figure 2.3 *The forward and reverse diffusion processes. The image sample at the first timestep is pure noise whereas the final sample at the last timestep is the denoised, generated image. Source: [7].*

DMs have demonstrated incredible performance improvements in image generation, editing and restoration, with their results even outperforming GANs [7, 9]. While most DMs use very large datasets for training, a recent method introduces a framework for training a DM on a single image [9]. In this dissertation, we aim to utilize this framework for motion summarization, in an attempt to develop a model able to obtain better results than those of GANs.

2.3 Single-generative Models for Image Scaling

In the past few years, several single-generative models have been developed for image synthesis and manipulation. These models can capture the internal distribution of patches or deep features in a single image using progressive training, a technique that is often used in deep neural networks for image generation that gradually increases the complexity of the model during training [9]. In progressive training, the model is first trained on lower-resolution versions of the input data and then gradually trained on higher-resolution versions, allowing the model to learn coarse features first before moving on to more fine-grained details [9, 11]. We studied three such models developed for image resizing in our aim to solve the motion summarization problem.

2.3.1 SinGAN

The first model, SinGAN, is an unconditional, hierarchical GAN model that can generate high quality, diverse samples of arbitrary size based on a single training image. Its core idea is to use patches of a single image for training samples, rather than whole image samples from a database. SinGAN's multi-scale architecture consists of a pyramid of generators of different scales, each associated with a discriminator with the same scale and receptive field.

Each generator trains on a down-sampled version of the input image and is responsible for generating realistic image samples with respect to the patch distribution in its corresponding image scale. During training, a pure white noise image at the smallest scale is given as input to the first generator, which maps it to an image sample. The image sample then passes sequentially through each generator after being up-sampled and injected with noise at every scale. The generators then try to fool their associated discriminator with their generated input sample, who in turn tries to classify if the sample comes from the real or fake image. This results in the generator at the finest scale being able to produce a realistic sample of any desired scale from the input image [11]. The process is illustrated in Figure 2.4.

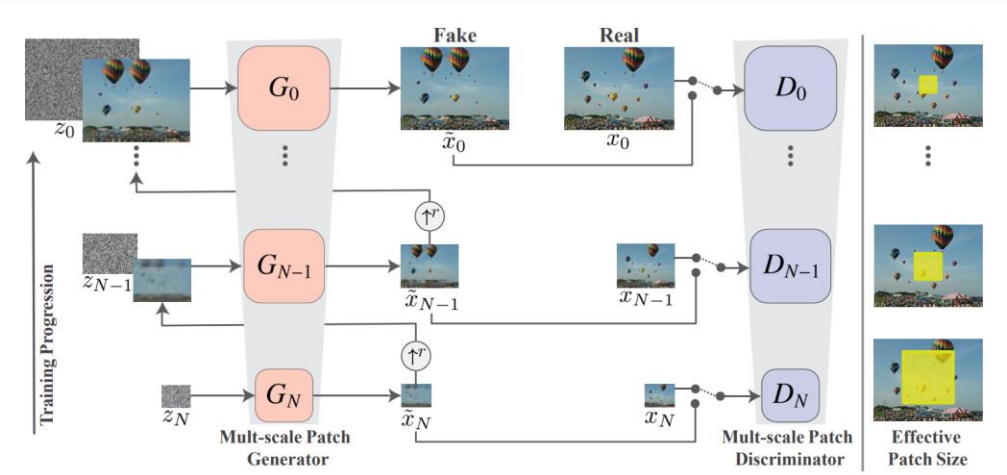

Figure 2.4 *SinGAN's multi-scale pipeline. The model is a pyramid of GANs that trains and infers in a coarse-to-fine manner. Each scale learns to generate images where overlapping patches cannot be distinguished from the down-sampled training image by its discriminator, with decreasing patch size as it goes up the pyramid. The input to each level's generator is a random noise image and the generated image from the previous scale. Source: [11].*

2.3.2 InGAN

The second model, InGAN, is a single image conditional GAN model developed for image retargeting that offers similar results as SinGAN. This model is structured differently from SinGAN, aiming to ensure that both the distribution of patches and the image elements' relative locations match those of the original image. InGAN, unlike most GANs that map between two different distributions, is an automorphism. By taking advantage of this characteristic, the generator (Figure 2.6 Left) can be inverted to serve as a decoder and reconstruct the original image, in an encoder-encoder architecture. With a reconstruction loss between the initial image and the decoded generated image, the patch distribution of the original image can be retained. In addition, InGAN has a multi-scale discriminator (Figure 2.6 Right), in which each scale has several convolution layers and weights that are updated from the coarsest to the finest scale and are not shared between them. The multi-scale

discriminator uses matrices of real/fake labels of same size as the desired output, thus grading how well each patch matches the patch distribution, rather than grading the entire synthesized image. This preserves the relative location of each element in the image. The whole training process is illustrated in Figure 2.5. Once the network is trained on the input image, it can be used to remap the input to any size or shape in a single feedforward pass, without losing the internal patch distribution [12].

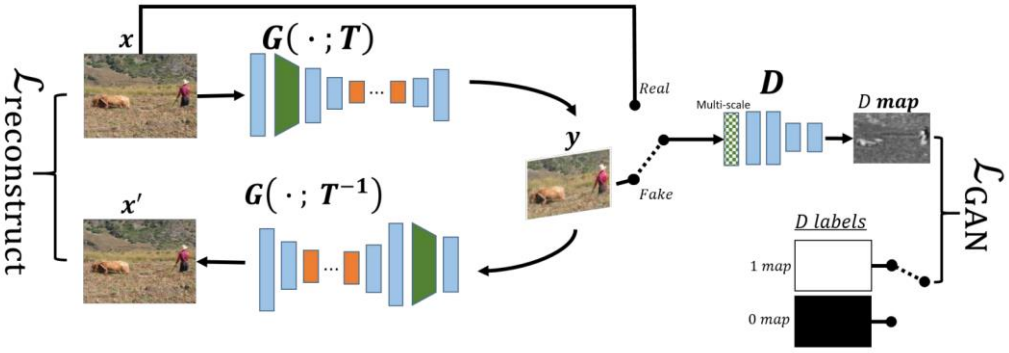

Figure 2.5 *InGAN's architecture. It includes a generator that transforms input x to output y based on a geometric transformation T. A multi-scale discriminator is used to distinguish between fake and true patch statistics of the output y and input x, respectively. The generator's automorphism is used to reconstruct the input back from the output using the inverse transformation* T^{-1}*. Source: [12].*

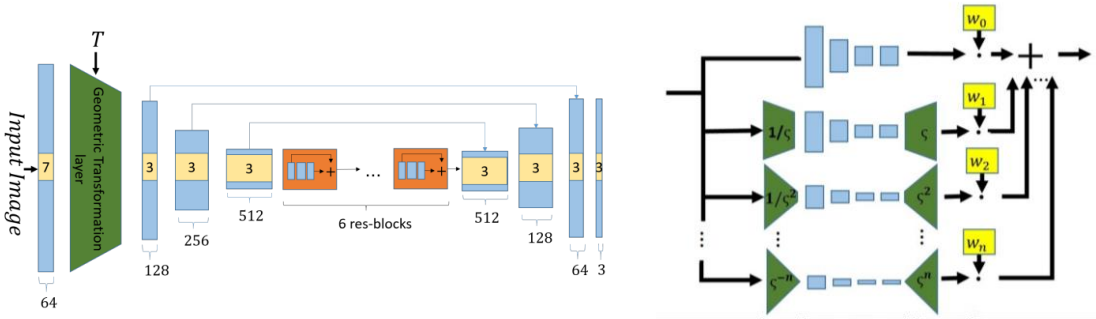

Figure 2.6 *Left: InGAN's hourglass-shaped invertible generator. The output size and shape of the generator are determined by a geometric transformation T applied to the input image. Right: InGAN's multi-scale discriminator. It captures both fine-grained details and coarse structures of the image by matching the patch distribution over a range of patch sizes. Source: [12].*

2.3.3 SinDDM

The third model, SinDDM, follows the hierarchical approach of SinGAN, but uses DDPMs instead of GANs. This enables it to both generate high quality images and support guided image generation. SinDDM learns the internal statistics of the training image through a multi-scale diffusion process, which gradually turns the image into white Gaussian noise in a hierarchical manner that combines both blur and noise. For the reverse diffusion process, the model has a fully convolutional light-weight denoiser conditioned on both the noise level and the scale. Before training, a pyramid of different scales is created by interpolating the input

image several times. During the forward diffusion process (Figure 2.7), each scaled sample in the pyramid is blended with random noise and the down-sampled (blurry) version of its previous scale, in an attempt to train the denoiser for reversing the process. For sampling, i.e. the reverse diffusion process (Figure 2.8), an image of pure white noise is generated at the smallest scale and is then passed through the denoiser for a number of timesteps. For each following scale in the pyramid, after the resulting image of the previous scale denoising is up-scaled, it is denoised and deblurred again for a number of timesteps until it reaches the finest scale, in which the final sample is obtained. With this architecture, image samples of arbitrary size can be generated from the input image in a coarse-to-fine manner [9].

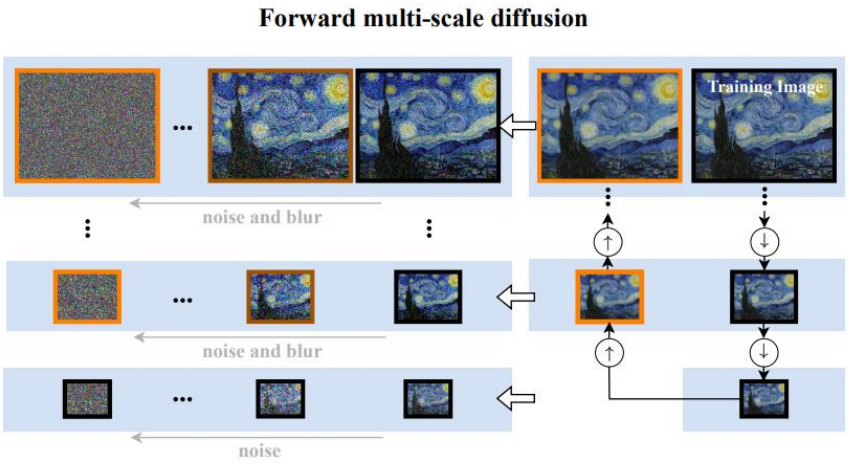

Figure 2.7 *SinDDM's forward multi-scale diffusion process. It is constructed from down-sampled versions of the training image (black frames) and their blurry versions (orange frames). Each scale produces a series of images that are linear combinations of the scale's original image, its blurred version, and noise. Source: [9].*

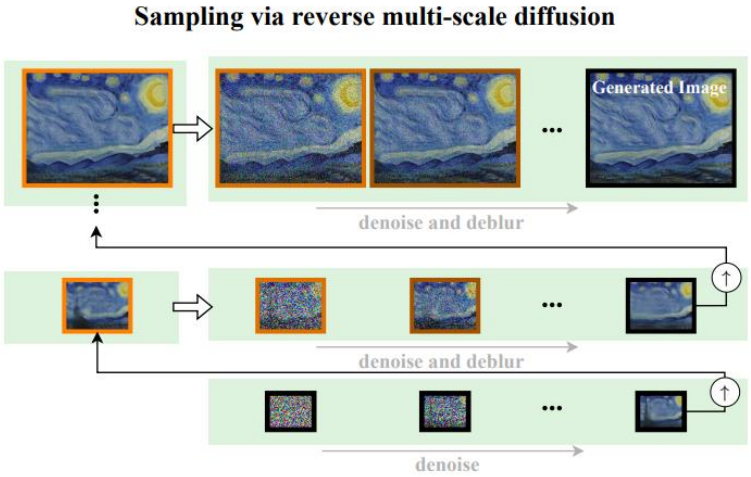

Figure 2.8 *SinDDM's reverse multi-scale diffusion sampling process. It starts with pure noise at the coarsest scale, and as it progresses through each scale, noise is gradually removed until a clean image is obtained. This image is then up-sampled and combined with noise to begin the process again at the next scale. Source: [9].*

All three models are fully unsupervised and require no additional data other than the input image itself [9, 11, 12]. SinGAN was the inspiration behind GANimator, a generative model for motion synthesis which will be further explained in the next section [10]. After testing the

capabilities of these models on images, we created our first network based on the network introduced in InGAN and laid the foundations for our second network based on the network introduced in SinDDM.

2.4 GANimator

GANimator is a generative model inspired by SinGAN that learns to generate new and diverse motions while simultaneously resembling the core elements of the original motion. It can support a variety of skeletal structures like bipeds, quadropeds, hexapeds, and more while only training on a single, short motion sequence. GANimator's framework contains a series of generative and adversarial neural networks generating motions in different frame rates. By progressively learning to synthesize motion from random noise, the framework allows for hierarchical control over the generated content at varying levels of detail. To achieve this, SinGAN's multi-scale generative architecture is employed to learn the distribution of temporal patches at different resolutions in a motion sequence. The generative network starts with random noise and synthesizes a coarse motion sequence which is progressively up-sampled to the finest temporal resolution. In each level, the input sequence goes through an up-sample operation, followed by a random noise to control the generated result's variation. A skip connection is used in each level to focus on the missing high-frequency details and relieve the network from regenerating the input. During training, the high-resolution result is fed into a patch-style discriminator that decides if each patch is fake or real. This process is illustrated in Figure 2.9. Adversarial loss and reconstruction loss are used to provide better quality and stabilize the training. Furthermore, contact consistency loss is introduced to enforce the generated foot velocity to be close to zero when the predicted contact label is enabled, thus avoiding sudden change artifacts and increasing the perceptive quality of the result [10].

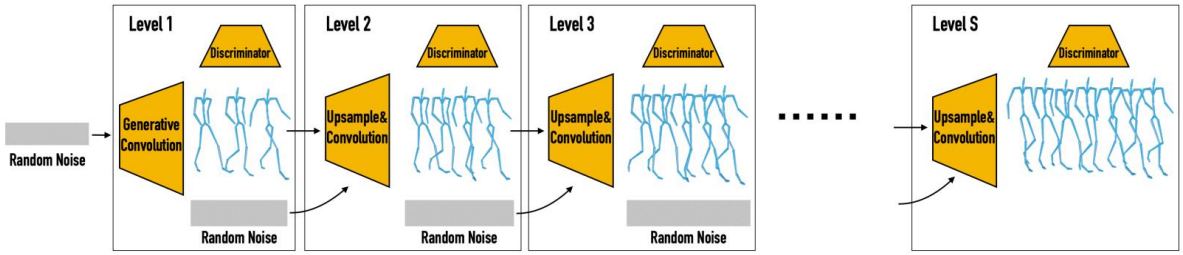

Figure 2.9 *GANimator's architecture. It begins with random noise and generates a coarse motion sequence using a generative network. The motion is progressively up-sampled until the finest temporal resolution is achieved. Each level, except the first, receives the output from the previous level and a random noise as input to generate an up-sampled version of the input sequence. Adversarial training is used by feeding each generated result into a discriminator at the corresponding level. Source: [10].*

As part of the work we conducted, we evaluated the proposed network's ability in the context of motion summarization and concluded that despite GANimator's astounding results in motion generation, it appears to have notable limitations. Specifically, it has no control over the generated content, does not perform any highlighting of significant elements of the motion, and more importantly, lacks global translation control. Our networks were developed in an effort to overcome these limitations.

2.5 Deep Motion Motifs and Motion Signatures

In the following work [2], it has been shown that motion sequences can be divided into a smaller set of overlapping movements whose distribution can characterize the sequence itself. These smaller movements are described as motion words, i.e. narrow temporal windows of all joint transformations in each frame, representing the local evolution of pose. Motion words are gathered from a large dataset of various motion-captured activities to define a d-dimensional universal motion word feature-space R^d, in which motion words can be clustered by the K-means clustering algorithm based on semantic similarity. Each cluster is represented by a motion motif, which is defined as "a small movement that is common inside a motion sequence and reveals its characteristics" and is essentially the motion word at its centroid. An example of a motion motif is a simple dance figure in a dance routine. Thus, new motion words can be clustered by finding the most similar motion motif. Another term used in the paper is a motion signature, that is, "a global, semantically meaningful representation" which can be used to describe a motion sequence and compare it with other motion sequences. To provide a motion signature for a specific motion sequence, one has to extract all its motion words, map them to the R^d space, assign each word to the most similar motif and calculate the normalized histogram of the words in all K clusters. Motion signatures are re-weighted using tf-idf (term frequency – inverse document frequency) to combat naive frequency counting problems. The similarity of two motion sequences can then be determined by calculating the distance between their motion signatures [2]. An example of different motion signatures for various motion sequences, along with their matching motifs between motions of the same type, is displayed in Figure 2.10.

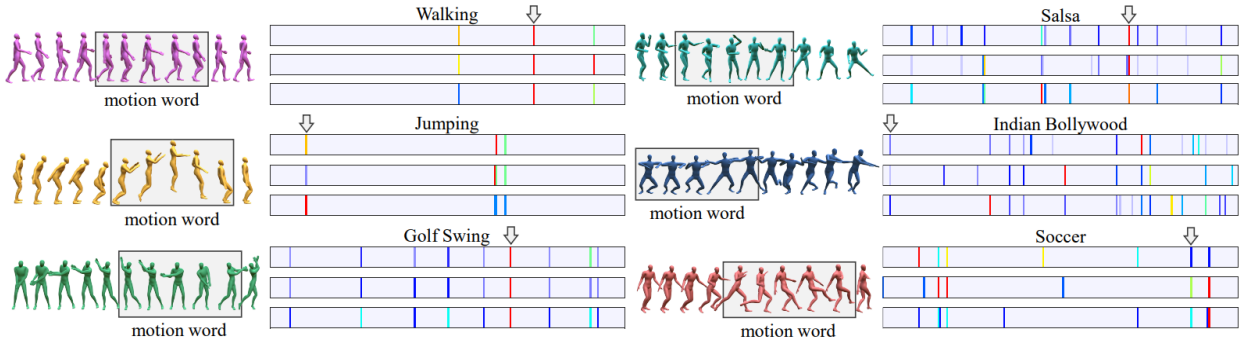

Figure 2.10 *Motion signatures. They are displayed as horizontal bars that use different colors to represent the frequency of motion-motifs, with red indicating a high frequency and gray indicating zero. These signatures show distributions rather than time evolution, and similar motions produce similar signatures where many motifs align. For each motion, three signatures of sequences are shown, and the rectangles in the motion sequence on the left of the signatures represent the motion words associated with the motifs shown by the corresponding arrow above the signature. Source: [2].*

When it comes to matching, clustering, segmenting, and indexing motions, motion motifs and signatures effectively capture both the structure and the characteristics of human motion. They also support sophisticated applications such as motion synthesis [2]. Our goal is to incorporate this idea into progressive training in animation, to identify the unique and common parts of a motion sequence. We can then train a generative model to keep the unique parts intact and in turn, shorten or increase the duration of the common parts, to summarize or expand a motion respectively.

Chapter 3

Implementation

In our general effort to tackle the motion summarization problem, we developed a framework with two networks following a single-motion generative model approach, with the first using a multi-scale discriminator and the second using multi-scale diffusion to train on a single motion file. We will refer to our two networks as InGAN-based and SinDDM-based respectively, as each of them was inspired by the respective pre-discussed model.

3.1 Database

The single-motion generative model approach eliminated the need for a large dataset of motion sequences. The data used to train and test our models were dance animations taken from DanceDB, a database of motion captured dance performances from University of

Cyprus's VRLab [14], in BVH (BioVision Hierarchy) format. The database's performances consist of skeletons of 31 joints captured at 120 frames per second. Using Autodesk's MotionBuilder software we reduced the input files' frame rate to 24 frames per second, the minimum framerate that could retain smooth and coherent animations, to minimize computational resources during the training process. The main motion sequence used to train both our networks was the "Third Antikristos" dance sequence, which resulted in a length of 1219 frames after modification. However, due to GPU memory limitations, we had to halve the file size for the SinDDM-based model, resulting in a length of 609 frames.

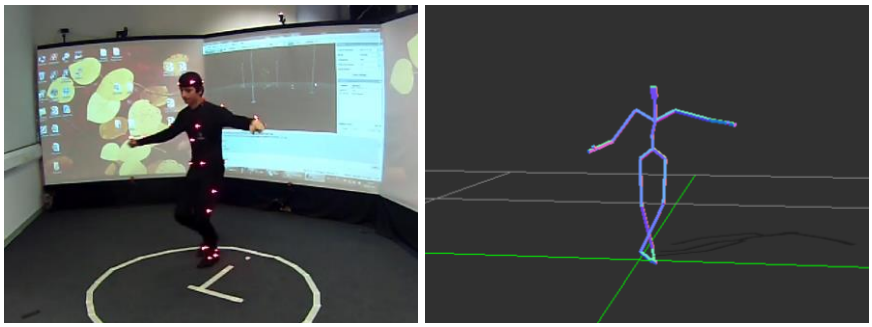

Figure 3.1 *The "Third Antikristos" dance sequence. Left: The real performance by the dancer at UCY's VRLab. Right: The same dance sequence in BVH format. Source: [14].*

3.2 Motion Representation

As our networks were inspired by the aforementioned single-generative models for images, our main idea was to create a representation for the motion sequence that is similar to that of an image. The BVH data must be parsed to this representation before being fed to the network. A BVH file is composed of two main sections: the Hierarchy section (Figure 3.2 and Figure 3.3 Left) and the Data section (Figure 3.3 Right). The Hierarchy section includes information about the joints and the initial pose of the skeleton, while the Data section contains the rotation of each joint in each frame in Euler angles. By considering the joint hierarchy in the Hierarchy section we must extract the respective data from the Data section for each frame and transform it into a matrix-like format, similar to an image. We achieved this by using a slightly modified version of the parser from GANimator, which follows the same idea.

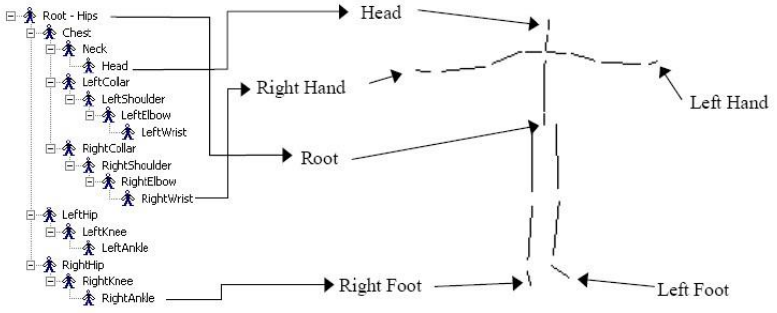

Figure 3.2 *A sample BVH skeleton hierarchy with several indicated joints.*
Source: https://www.cs.cityu.edu.hk/~howard/Teaching/CS4185-5185-2007-SemA/Group12/BVH.html.

```
HIERARCHY
ROOT Hips
        {
                OFFSET 0.00 0.00 0.00
                CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
                JOINT Chest
                {
                        OFFSET 0.000000 6.275751 0.000000
                        CHANNELS 3 Zrotation Xrotation Yrotation
                        JOINT Neck
                        {
                                OFFSET 0.000000 14.296947 0.000000
                                CHANNELS 3 Zrotation Xrotation Yrotation
                                JOINT Head
                                {
                                        OFFSET 0.000000 2.637461 0.000000
                                        CHANNELS 3 Zrotation Xrotation Yrotation
                                        End Site
                                        {
                                                OFFSET 0.000000 4.499004 0.000000
                                        }
                                }
                        }
                }
```

```
MOTION
Frames: 2
Frame Time: 0.04166667
-9.533684   4.447926   -0.566564   -7.757381   -1.735414   89.287932   9.763572
            6.289816   -1.825344   -6.106647   3.973667    -3.706973   -6.474916
            -14.391472 -3.461282   -16.504230  3.973544    -3.805107   22.204674
            2.533497   -28.283911  -6.862538   6.191492    4.448771    -16.292816
            2.951538   -3.418231   7.634442    11.325822   5.149696    -23.869189
            -18.352753 15.051558   -7.514462   8.397663    2.953842    -7.213992
            2.494318   -1.543435   2.970936    -25.086468  -4.195537   -1.752307
            7.093068   -1.587532   -2.633332   3.858087    0.256802    7.892136
            12.883010  -28.692566  2.151862    -9.164188   8.086427    -5.641034
            -12.596124 4.366468
-8.489557   4.285263   -0.621559   -8.244940   -1.784412   90.041962   8.849357
            5.557910   -1.926571   -5.487280   4.119726    -4.714622   -5.790586
            -15.218462 -3.167648   -15.823254  3.871795    -4.378948   22.399654
            2.244878   -29.421873  -6.918557   6.131992    4.521327    -18.813188
            3.059388   -3.768287   8.079588    10.124812   5.888083    -22.417845
            -15.736264 18.827469   -8.878788   9.689189    2.417364    -7.680582
            2.505805   -1.625679   2.430162    -27.579788  -3.852241   -1.830524
            12.528144  -1.653632   -2.688550   4.545600    0.296328    8.031574
            13.837914  -28.922058  2.077955    -9.176716   7.166249    -5.178825
```

Figure 3.3 *Left: sample BVH Hierarchy section. Right: sample BVH Data section.*
Source: https://www.cs.cityu.edu.hk/~howard/Teaching/CS4185-5185-2007-SemA/Group12/BVH.html.

Based on the desired rotation representation, our parser converts this data into a 2D matrix that contains the rotations of each joint and the translation of the root joint for each frame in its columns. The number of columns is the number of frames in the motion whereas the number of rows in the matrix is determined by the rotation representation. For example, if we desired to represent a motion sequence of N frames and J joints in Euler angles, the number of columns would be N and the number of rows would be J times 3 (x, y, z rotations for each joint) plus 3 (x, y, z translation of the root joint). Columns 0 and N-1 would then represent the first and last frames respectively.

For our representation, there are two important factors that we should consider: translation and rotation. To represent translation, we use relative distances *dx*, *dy*, and *dz* between frames instead of using the absolute x, y, and z values for each frame, which helps avoid uncontrolled and abnormal translation problems in the resulting motion [10]. Regarding rotation, there are various types of representations available for the 3D rotations in our data, such as Euler angles, quaternions, 6D rotation etc. Euler angles are a basic representation using a set of three angles, one for each axis (x, y, and z). Quaternions are another representation for rotations in the 3D space using a vector of four values instead of three, consisting of a scalar value and three imaginary numbers. This representation has several advantages over Euler angles, such as avoiding the gimbal-lock problem, where two axes

align and the system loses a degree of freedom (Figure 3.4 Left). However, as demonstrated in [13] all representations in the real Euclidean spaces of four or fewer dimensions are discontinuous for 3D rotations. For example, as shown in the right panel of Figure 3.4, angles that are close to 0 and 2π (original space) are represented far from each other (representation space), leading to discontinuity errors. Thus, the above widely used representations (Euler angles and quaternions) are difficult for neural networks to learn. On the other hand, it has been shown that 3D rotations have continuous representations in 5D and 6D, where the rotation is represented with five or six dimensions respectively, are more suitable for deep learning applications [13].

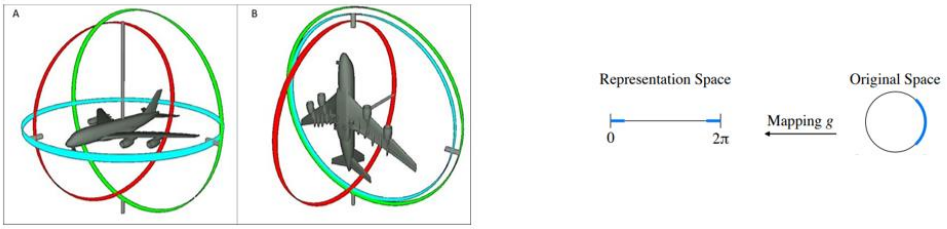

Figure 3.4 *Left: Gimbal-lock. A – normal case, B - two axes align. Right: Discontinuity of Euler angles. Sources: https://www.researchgate.net/figure/Gimbal-lock-problem-for-Euler-angles-A-no-gimbal-lock-B-yaw-and-roll-angles-are_fig14_332394380, [13]*

Therefore, we used the 6D representation for our 3D rotations, which was shown to be continuous and more effective for our network. A more recent representation proposed in [1], dual quaternions, has also shown promise in producing better results than 5D/6D, but we left this for future exploration. Additionally, to further improve our results, we use foot contact labels to ensure that there is correct foot contact and no foot sliding artifacts. For a 6D representation on the modified “Third Antikristos” motion sequence, our resulting input data format is a 2D matrix of size [192, 1219] (and [192, 609] for our second network).

3.3 InGAN-based Model

Our first network, which we refer to as InGAN-based, was developed based on the InGAN model for image retargeting. To create the network, we started with the InGAN code as a foundation and made modifications to allow it to generate high-quality results for motion sequences. This involved implementing network layers, forward and backward propagation, and optimization techniques using PyTorch. Through these modifications, we were able to adapt InGAN's framework to suit the requirements of motion synthesis and summarization. The resulting network was able to produce promising results when creating smaller motion sequences in our experiments.

3.3.1 Network Architecture

The network is structured similarly to InGAN, featuring an invertible generator and a multi-scale discriminator, as shown in [Fig]. To produce a new fake motion, the generator takes in the input motion and a transformation that specifies the desired duration for the generated motion. We follow the approach of InGAN by using the generator's ability to reconstruct the input motion as a measure of whether the generated motion captures all the relevant content. Specifically, we invert the generator by feeding it the generated motion and the negative of the transformation used to create it, and then compute a reconstruction loss between the input motion and the motion generated from the inverted generator. Meanwhile, our multi-scale fully-convolutional patch discriminator includes convolutional layers at different scales, with weights updated from coarse to fine scales. By using overlapping patches and down-sampling factor $\varsigma = \sqrt{2}$, the discriminator can effectively capture both fine-grained details and coarse structures in the motion. This architecture helps the discriminator match the patch distribution over a range of sizes.

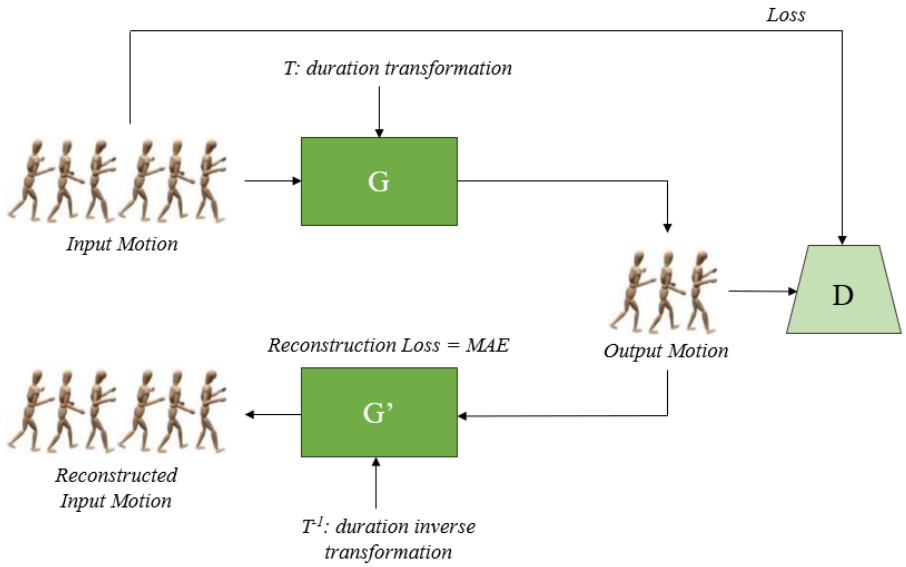

Figure 3.5 *Our InGAN-based model's architecture. It includes a generator that transforms the input motion to an output motion based on a duration transformation T. A multi-scale discriminator is used to distinguish between fake and true patch statistics of the output and input motions, respectively. Like InGAN, the generator's automorphism is used to reconstruct the input back from the output using the inverse transformation* T^{-1}.

For both the generator's and discriminator's convolutional networks, we maintain InGAN's architecture. As InGAN works with images, it expects a tensor of size [3, M, N], where M and N are the height and width of the image, and 3 represents the image's R, G, and B channels. However, as our input is a 2D matrix of size [192, N], where N is the number of frames in the motion, we had to modify the number of channels in the convolution blocks and

change it from 3 to 1. Moreover, we changed all occurrences of the input tensor's height to 192 since motion rescaling would only occur in the x-axis, and the height of the tensor would remain fixed. This modification ensures that the generator does not produce samples with different heights, which would have no meaning in our representation. Furthermore, since each row of our 6D representation corresponds to specific joint rotations, we did not modify them in any way to maintain the accuracy of the motion data distribution.

To be more specific, the generator architecture contains several blocks as follows:

1. **Entry block**: A convolution block with kernel size of 7 × 7, stride of 1, input channel of 1 (modified), and padding of 3. The output channels are equal to the G_base_channels, followed by normalization and LeakyReLU activation function.
2. **Geometric transformation block**: A block that applies the geometric transformation to the input motion.
3. **Downscaling block**: A sequence of rescale blocks with three down-sampling layers that perform convolution with kernel size of 3 × 3 and stride of 2. The output channels are doubled (2 × G_base_channels) compared to the input channels (G_base_channels).
4. **Bottleneck block**: A sequence of six ResNet blocks that maintain the same motion size and output channels as the previous block.
5. **Upscaling block**: A sequence of rescale blocks that perform transposed convolution with kernel size of 3 × 3 and stride of 2. The output channels are halved compared to the input channels.
6. **Final block**: A convolution block with kernel size of 7 × 7, stride of 1 and padding of 3. The input channels are equal to the G_base_channels and the output channels are 1, with Tanh activation function that returns a range of -1 to 1.

For each scale, the discriminator has a four block fully-convolutional network as follows:

1. **Entry block**: A convolution layer with kernel size of 3 × 3, stride of 1 and input channel of 1 (modified), followed by Batch Normalization and LeakyReLU activation function with slope of 0.2.
2. **Downscaling block**: Another convolution layer with kernel size of 3 × 3 and stride of 2. The output channels are doubled (2 × D_base_channels) compared to the input channels (D_base_channels), followed by Batch Normalization and LeakyReLU activation function.
3. **Regular conv-block**: A convolution layer with kernel size of 3 × 3 and input and output channels of 2 × D_base_channels. This block does not change the image size

as the stride is 1, followed by Batch Normalization and LeakyReLU activation function.

4. **Final conv-block**: A convolution layer with kernel size of 1 × 1, input channels of 2 × D_base_channels and output channels of 1. This block outputs a single scalar value representing the probability of the input image being real or fake in a GAN setting. It ends with a Sigmoid activation function to get a range of 0-1.

All convolutional layers use spectral normalization to stabilize training and avoid the mode collapse problem. The output of each block is concatenated to the output of the previous block and passed to the next block. The final output of the generator's architecture is a motion with the desired size, and tensor values ranging from -1 to 1. The final output of the discriminator's architecture for each scale is a single scalar value representing the probability of the input image being real or fake. A forward pass of the discriminator takes an input tensor and a set of scale weights and aggregates the interpolated results from all scales to generate the output tensor.

Another significant modification was switching the GAN's optimization variant from LS-GAN to WGAN-GP, or Wasserstein GAN with Gradient Penalty. In both variants, the discriminator tries to predict the distance between the real and fake samples, instead of classifying them as real or fake, while the generator tries to minimize the distance of the predicted value of the fake sample and a target value. However, in WGAN-GP the discriminator becomes a Lipschitz function with a Wasserstein loss, which measures the distance between the distribution of the generated motion and the distribution of the real motion. A gradient penalty term is added to this loss function to enforce small gradients. WGAN-GP is more stable than LS-GAN, especially when it comes to high-dimensional data. This is because WGAN-GP's Wasserstein distance is a more meaningful metric for measuring the distance between distributions than LS-GAN's least-squares loss. Using this approach provides more stable gradients during training, better convergence, and overcomes problems like mode collapse [3, 6].

Finally, we encapsulated our network in two parsers used for converting the input and output of the network from and to BVH files respectively, by integrating and modifying some code from GANimator. The first parser is responsible for converting the input BVH file into a tensor with the pre-described motion representation, which is then provided as the true input for our network. The second parser converts the output tensor of our generator from our

motion representation to a BVH file, containing the generated motion sequence. This BVH file is the final output of our network.

3.3.2 Losses

To ensure that the generated motion contains all the content of the original motion, we use a reconstruction loss between the input motion and the motion generated from the inverse generator. The reconstruction loss is based on the Mean Absolute Error (MAE) loss, which is the same as InGAN's reconstruction loss and provided satisfactory results. As mentioned earlier, we have adopted a Wasserstein loss for our discriminator to address issues that arise from using InGAN's LS-GAN. The generator's total GAN loss is a weighted average of the reconstruction loss and the discriminator's Wasserstein loss. In the future, we plan to introduce a Global Pelvis Translation loss to address pelvis translation issues, as demonstrated in the next chapter. Additionally, we may include a loss to ensure that the generator considers the pre-processing step for unique and common motions.

3.3.3 Parameters

Our network consists of several hyperparameters. The most significant are shown below along with their previous value as in InGAN, where modified:

- Number of base channels in generator convolutional network: **G_base_channels** = 64
- Number of base channels in discriminator convolutional network: **D_base_channels** = 64
- Total number of training epochs: **max_iters** = 30,000 (changed from 75,000)
- Momentum term for Adam optimizer: **beta1** = 0.9 (changed from 0.1)
- Initial learning rate for generator and discriminator: **g_lr** = **d_lr** = 0.000005 (changed from 0.00005)
- Learning rate starts linearly decaying after epoch: **lr_start_decay_iter** = 20,000
- Inverse generator starts training after epoch: **G_extra_inverse_train_start_iter** = 10,000

All network parameters can be set from command line arguments during execution.

3.3.4 Training and Testing

Training is done using a single motion file and the number of epochs is given as a program argument. In each epoch, both the generator and the discriminator are trained with the Adam optimizer [8]. First, the generator is trained for one step to generate an image of random size from noise, which in turn is fed to the discriminator. To calculate the reconstruction loss, the generated image is passed through the inverse generator, which is essentially the generator given the reversed input size. At this point, after a certain number of epochs, the inverse generator is trained for an additional step for all remaining epochs. The final loss for the generator is a weighted average of the reconstruction loss and the loss which was based on the discriminator's prediction. Next, the discriminator is trained for two steps, in which it first makes a prediction for the real motion, and then for the generated motion. This was another modification of the initial InGAN network, which trained the discriminator for one step only. The final discriminator loss is the Wasserstein loss. As the training progresses, the learning rates for both the generator and discriminator start linearly decaying after a certain number of epochs. Even though this technique is used to improve the stability of the training process and ensure better convergence, it did not have an impact on our results. For testing, we load the checkpoint of the model at its best state (before losses and gradient penalty start increasing), according to the GAN losses and gradient penalty graphs. After thorough testing, we concluded that the most accurate results are generated at about 15,000 epochs, half the number of max epochs we configured. We then input the motion sequence into the generator with our desired duration to generate the summarized or expanded version.

3.4 SinDDM-based Model

Our second network, which we refer to as SinDDM-based, was built uppon the SinDDM model for image synthesis and manipulation. Similar to our previous network, we started with the SinDDM code as a foundation and made modifications by implementing network layers, forward and backward propagation, and optimization techniques again using PyTorch. Through these modifications, we established the foundations for a diffusion model capable of performing motion summarization and expansion. However, we encountered some problems during training and are currently researching ways to overcome them and improve the quality of our results.

3.4.1 Network Architecture

The network is structured similarly to SinDDM, with a denoiser that learns the internal statistics of the training motion through a multi-scale diffusion process. The denoiser is fully-convolutional, light-weight and is conditioned on both the noise level and the scale. Prior to training, a pyramid of different motion scales is created by interpolating the input motion several times. During the forward diffusion process, each scaled sample in the pyramid is blended with random noise and the down-sampled version of its previous scale to train the denoiser for reversing the process, as shown in Figure 3.6. In the reverse diffusion process, a motion matrix of pure white noise is generated at the smallest scale and is then passed through the denoiser for a certain number of timesteps. For each subsequent scale in the pyramid, after the resulting motion of the previous scale denoising is up-scaled, it is denoised and deblurred again for a number of timesteps until it reaches the finest scale, in which the final sample is obtained. This is illustrated in Figure 3.7. With this architecture, motions with arbitrary duration can be generated from the input motion in a coarse-to-fine manner.

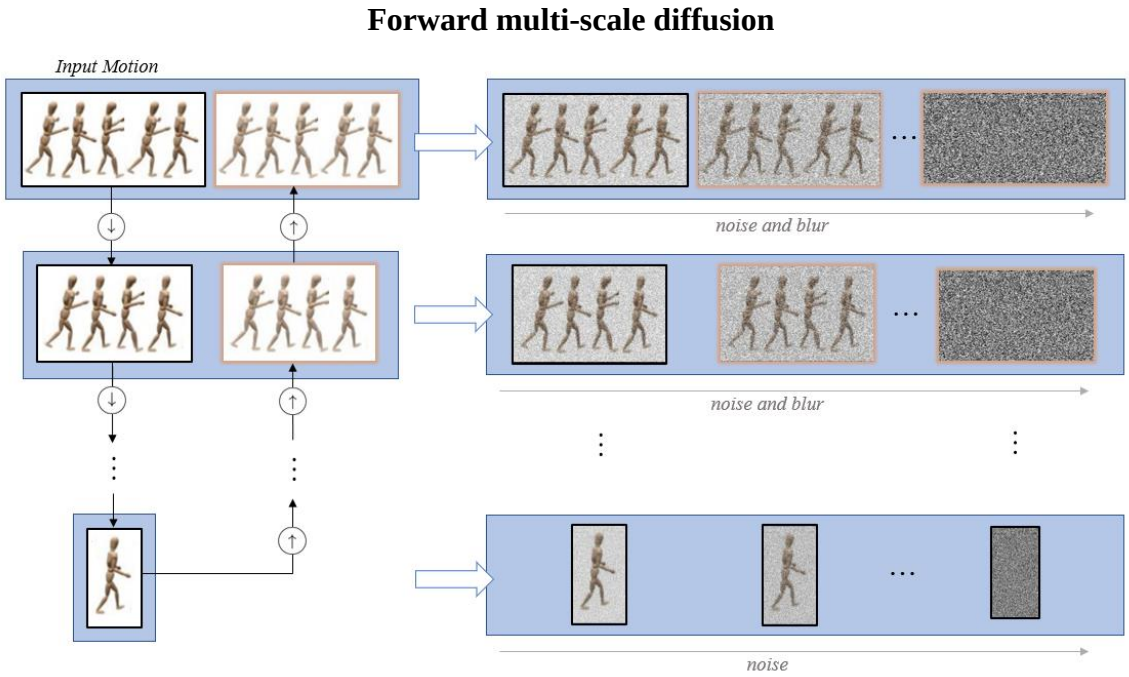

Figure 3.6 *Our SinDDM-based model's forward multi-scale diffusion process. It is constructed from down-sampled versions of the training motion (black frames) and their "blurry" versions (orange frames). In a fashion similar to SinDDM, each scale produces a series of motions that are linear combinations of the scale's original motion, its blurred version, and noise.*

Sampling via reverse multi-scale diffusion

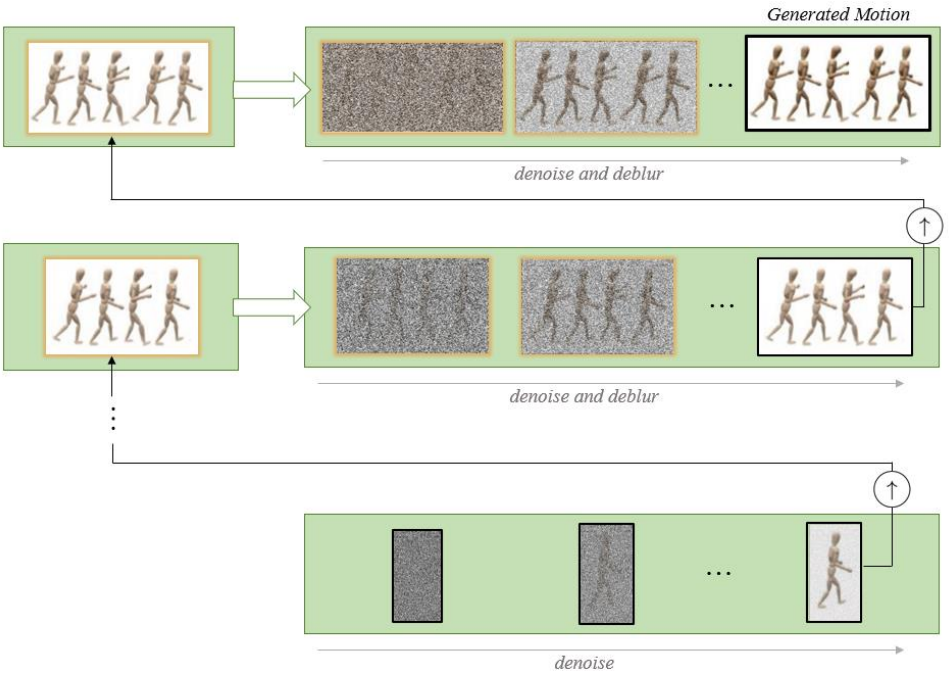

Figure 3.7 *Our SinDDM-based model's reverse multi-scale diffusion sampling process. It starts with pure noise at the coarsest scale, and as it progresses through each scale, noise is gradually removed until a clean motion is obtained. This motion is then up-sampled and combined with noise to begin the process again at the next scale.*

As previously mentioned, our network consists of a pyramid of scales with down-sampled versions of the input motion, which is a 2D matrix in the pre-described representation. To achieve the appropriate sizes for our different scales, we utilized linear interpolation with a down-scaling factor of $\varsigma = \sqrt{2}$ to down-scale the input, thus generating inputs for all the various scales of our pyramid. Similar to SinDDM, this allowed us to capture the larger motion patterns in the coarser scales, while the finer scales captured the smaller motion patterns or "details" of our motion. In addition, a reconstructed motion is created at each scale greater than the smallest one, by up-scaling the down-scaled motion of the previous scale, as the SinDDM model works on both noise and blur. Note that, like our first network, there was no meaning in scaling the motion height, which for 6D representation has a fixed value of 192. Therefore, linear interpolation was performed only in the x-axis for all scales.

The denoiser model takes an input tensor (motion representation), a time step and a scale. It consists of an embedding block and a pipeline of four connected convolution blocks. The timestep and scale are first converted to two vectors and pass through the embedding block, resulting in a time-scale embedding vector:

- **Embedding Block:** A block that calculates the Sinusoidal Positional Embedding (SPE) of a time tensor with a size of 32 and a scale tensor with a size of 32 using two **SPE blocks***, concatenates the resulting tensors and feeds them into a multi-layer perceptron (MLP). The MLP has a linear layer with 64 input features and 128 output

features, the output of this layer is then passed through a GeLU activation function, and then another linear layer with 128 input features and 64 output features.

- ***Sinusoidal Positional Embedding (SPE) Block:** A block that applies SPE to the input tensor.

The input motion, along with the time-scale embedding vector then pass through the pipeline of connected convolution blocks. Since we train on a single motion, we take measures to avoid memorization of the motion by limiting the receptive field of the model. Specifically, in our fully convolutional pipeline, each convolution block has a receptive field of 9 × 9, yielding a total receptive field of 35 × 35, and is structured as follows:

- **Convolution Block:** A block that consists of two parts: an upper part that processes the time-scale embedding vector by passing it through a fully-connected layer and a GeLU activation function, and a lower part that processes the output tensor of the previous block with a residual connection. The lower part consists of a 2D convolutional layer with a kernel size of 5 × 5, followed by an element-wise sum operation with the processed time-scale embedding vector, another 2D convolutional layer with a kernel size of 3 × 3, a GeLU activation function, and another 2D convolutional layer with a kernel size of 3 × 3. The resulting tensor is then passed through another element-wise sum operation with the input tensor, which has been processed by a 2D convolutional layer with a kernel size of 1 × 1 (residual connection).

The input and output tensor sizes vary between the convolution blocks in the pipeline, and are configured by input_dim and output_dim as follows:

- **1st Convolution Block:** input_dim of size 1 (modified) and output_dim of size 80
- **2nd Convolution Block:** input_dim of size 80 and output_dim of size 160
- **3rd Convolution Block:** input_dim of size 160 and output_dim of size 160
- **4th Convolution Block:** input_dim of size 160 and output_dim of size 80

The final output tensor is passed through a 2D convolutional layer with kernel size of 1 × 1, that reduces the number of channels to the initial number of channels in the motion tensor (1, modified), producing the denoised output.

We encapsulated our network in two parsers used for converting the input and output of the network from and to BVH files respectively, with the same code as our previous network. Similar to our first network, the first parser converts the input BVH file into a tensor with the pre-described motion representation, which is then provided as the true input for our network,

and the second parser converts the output tensor of the reverse diffusion process from our motion representation to a BVH file, containing the final generated motion sequence.

3.4.2 Losses

After creating the pyramid of scaled motions, we calculate a rescale loss using the Mean Square Error (MSE) loss between each rescaled motion and the down-sampled motion of its previous scale. This helps determine the most effective number of timesteps to be used in the sampling process for each scale. However, since our representation is different than SinDDM, which uses images that have pixel values ranging from 0 to 255, we multiply the rescale losses with a loss factor that is set as a network parameter. For training the denoiser model, we use the Mean Absolute Error (MAE) loss between the initial white noise and the denoised motion. Both losses are the same as the ones used in the SinDDM model.

3.4.3 Parameters

Our network consists of several hyperparameters, with the most significant shown below:

- Widest number of channels in convolutional network: **dim** = 160
- Down-scaling factor for motion scales pyramid: **scale_factor** = 1.414
- Batch size during training: **train_batch_size** = 8 (changed from 32)
- Total number of training epochs: **train_num_steps** = 480,000 (changed from 120,000)
- Initial learning rate: **train_lr** = 0.001
- The epochs (x1000) at which the learning rate is halved: **sched_k_milestones** = [20, 40, 70, 80, 90, 110]
- Exponential moving average decay: **ema_decay** = 0.995
- Ratio between MSE loss and starting diffusion step for each scale: **loss_factor**

Like our first network, all network parameters can be set from command line arguments during execution. However, for the "loss_factor" parameter, we have not yet determined the optimal value. We had to experiment with different values for each modification and approach we took to address the issues we encountered.

3.4.4 Training and Testing

The network is trained using a single motion file and the number of training epochs is provided as a program argument. As our previous network, the denoiser model is trained with the Adam optimizer [8]. During each training epoch, the network performs a forward diffusion process, training to predict the noise at a specific scale and time. This involves randomly selecting a scale from the available scales in the pyramid, which represents the different levels of abstraction in the motion. The motion sample at that scale is fetched, along with its reconstruction (up-scaled sample of previous scale) and a random timestep in the range of 0 to 100. A forward pass is then performed based on the selected scale and timestep. During the forward pass, the network performs denoising and deblurring, depending on the scale. If the scale is the smallest in the pyramid, only denoising is performed. Otherwise, both denoising and deblurring are performed. The network generates a Gaussian white noise motion of the same scale and averages it with the fetched sample. If the selected scale is not the smallest, the fetched sample is first averaged with the reconstructed motion at that scale, and then with the noise. The denoiser model then performs a forward pass on this distorted motion, generating the denoised version. The network calculates the Mean Absolute Error (MAE) loss between this version and the initial white noise, which is used to update the model parameters. The learning rate is halved each time a scheduled milestone is reached. Additionally, an Exponential Moving Average (EMA) is applied on the model's weights with a decay factor of 0.995 to reduce noise effects and make the learning process more stable [9]. The testing of the model is based on the reverse diffusion process, also known as the sampling process. First, the best checkpoint of the model is loaded based on the running loss graph. Then, the desired motion duration is input to generate either a summarized or an expanded version of the motion. The sampling process starts by generating a motion matrix of pure Gaussian white noise at the smallest scale, which is then denoised with the denoiser for a certain number of timesteps. The sample is then up-scaled, denoised, and deblurred again for a certain number of timesteps, and this process is repeated for each subsequent scale in the motion pyramid. At the finest scale, the final sample is obtained. The number of timesteps for the sampling process at each scale is based on the previously described MSE loss, scaled by a loss factor.

Chapter 4

Results

4.1 System Specifications

Our networks were trained and tested on hardware with the following specifications:

- **Processor**: Intel Core i9-9900KF 3.60GHz
- **Processor Cores**: 8 physical, 16 logical
- **RAM**: 32GB
- **Graphics Card**: NVIDIA GeForce RTX 2080Ti
- **GPU Memory**: 10GB

4.2 Data

The motion data used to produce all the following results is the “Third Antikristos” dance sequence from the University of Cyprus’s DanceDB [14], which was previously described in section 3.1.

Screenshots of our results are displayed in each following section, while the animated versions will be presented in the thesis presentation.

4.3 Frames Removal - Seam Carving

Frame removal/insertion keeps the content of the motion intact and maintains correct temporal order of movements. However, it alters the speed of the motion, causing it to become faster during summarization and slower during expansion. This can be seen in Figure 4.1, where the summarized version (red) starts dancing before the initial (blue) and expanded (green) versions in the left panel, and completes the dance sequence while the others are still dancing in the right panel. In other words, this technique does not produce any new movements or remove core movements. Rather, it simply changes the motion's speed.

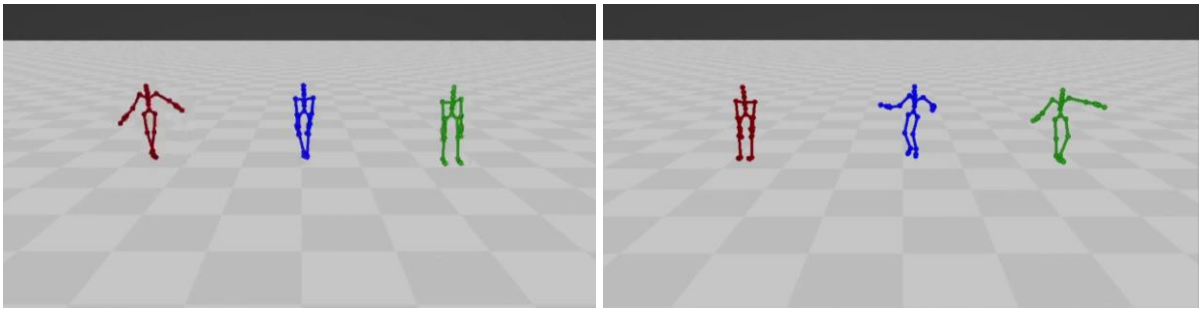

Figure 4.1 *Snapshots of the generated motions from using the frame removal technique. Three motions are shown at the same time: the summarized motion (red), the initial motion (blue), and the expanded motion (green). Left: a snapshot of the initial frames of the motion sequence, in which the summarized motion starts dancing before the initial and expanded motions. Right: a snapshot of the final frames of the motion sequence, in which the summarized motion has ended and the initial and expanded motions are still performing the dance sequence at different points in time.*

Similar to the above, seam carving maintains the motion content and temporal order, but alters the animation speed. As shown in the left panel of Figure 4.2, the summarized version (red) completes the dance sequence before the other two, which is similar to the previous approach. On the other hand, the animation produced from the seam carving technique loses its coherence and sometimes produces abnormal movements. It appears that, in the summarized version, several continuous frames are skipped, making the motion "jump" to the next movement, probably due to the removal of seams that are close to each other. In the expanded version, the insertion of seams produces abnormal rotations and unnatural movements, as displayed in all panels in Figure 4.2, where in some frames the dance appears to be performed upside down.

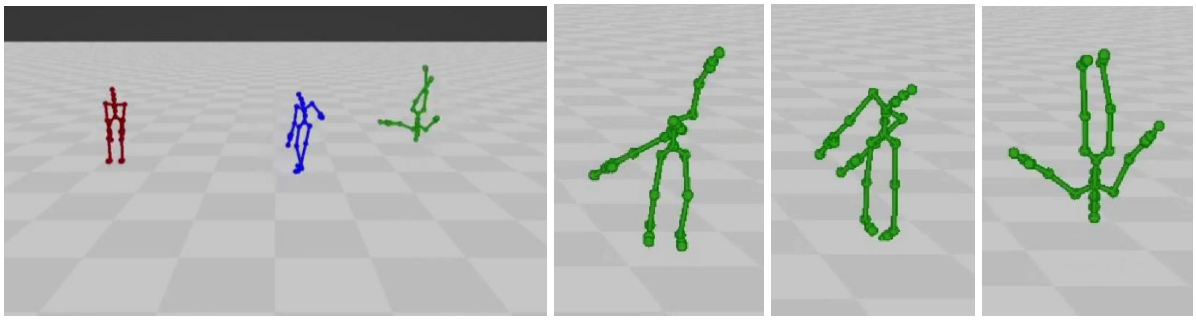

Figure 4.2 *Snapshots of the generated motions from using the seam carving technique. Three motions are shown at the same time: the summarized motion (red), the initial motion (blue), and the expanded motion (green). Left: a snapshot of the final frames of the motion sequence, in which the summarized motion has ended and the initial and expanded motions are still performing the dance sequence at different points in time. Right: a sequence of abnormal poses of the expanded motion skeleton.*

4.4 GANimator

Overall, GANimator is capable of producing high-quality animations with only minor inconsistencies. For example, at some frames the skeleton appears slightly tilted, as shown in the left panel of Figure 4.3. However, GANimator's major drawback is its inability to control the motion translation, as seen in the right panel of Figure 4.3. Here, the first and last frames of the generated motion are merged to show the total translation of the skeleton from its starting position. During the generated dance animation, the skeleton started moving slowly to the left while dancing, getting further and further away from its initial starting position. The significant difference between the final frames of the generated version (red) and the initial motion (blue) is evident in Figure 4.4. Despite the initial motion being constrained to a small range from the starting position, GANimator's result reaches far beyond that range.

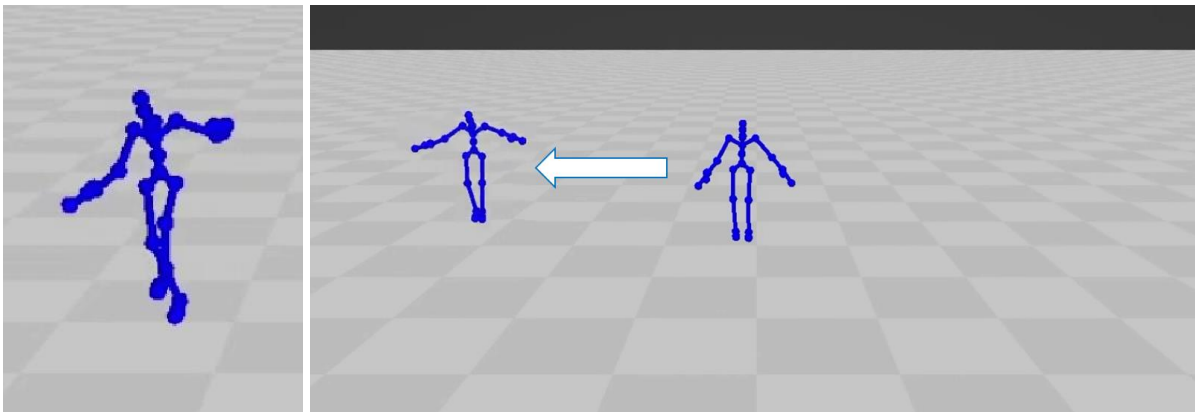

Figure 4.3 *Snapshots of the generated motion from using GANimator. Left: a slightly tilted pose of the skeleton during the generated motion. Right: two different frames are merged into one picture. The skeleton in the middle belongs to one of the initial frames and the skeleton to the left belongs to one of the final frames. It is evident that the skeleton had been gradually moving to the left during the entire dance sequence.*

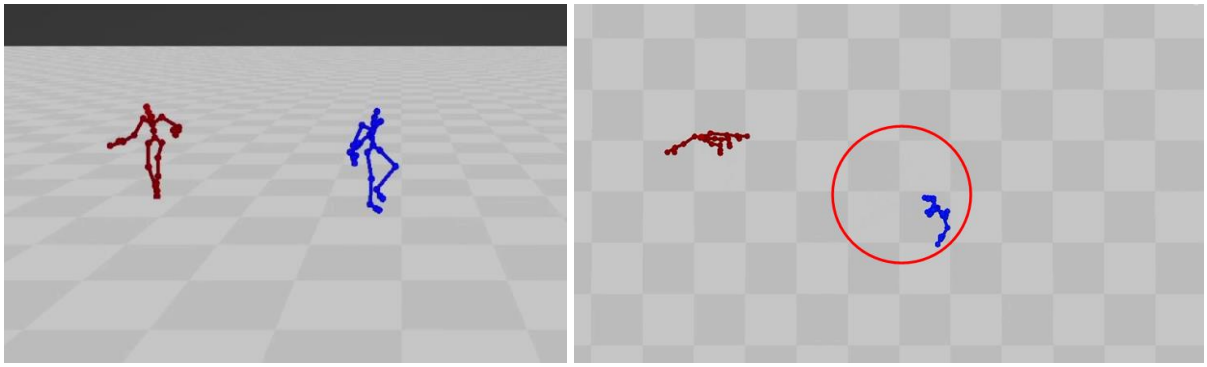

Figure 4.4 *Snapshots of the generated motion from using GANimator. Two motions are shown at the same time: the motion generated from GANimator (red) and the initial motion (blue). Left: one of the final frames, where the distance between the generated motion and the initial motion at the same time is evident. Right: the top view of one of the final frames. The initial motion's constrained range of movement is shown as a red circle.*

4.5 InGAN-based Model

As illustrated in the bottom panel of Figure 4.5, the reconstruction loss for the motion generated by the inverted generator decreases gradually as the number of epochs increases. However, both the total GAN loss (Figure 4.5 Top) and the gradient penalty of the Wasserstein loss (Figure 4.6) start increasing rapidly after approximately 15,000 epochs. Based on these findings and extensive testing, we determined that the optimal results were obtained using the model checkpoint at around 15,000 epochs.

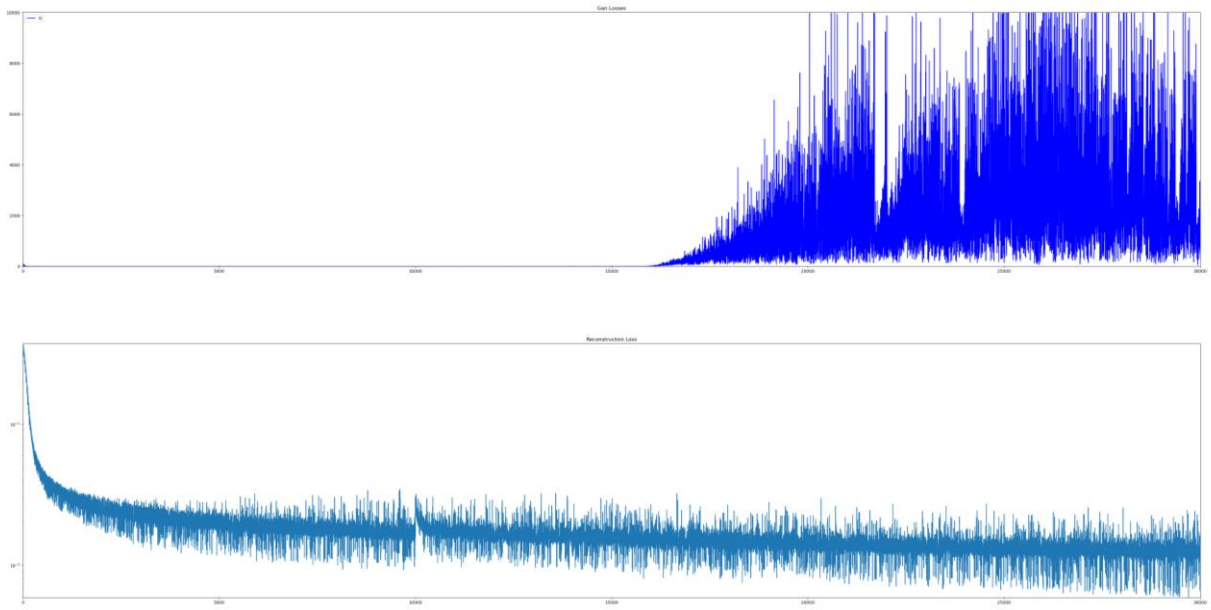

Figure 4.5 *Top: Our InGAN-based network's total GAN loss per epoch graph. Bottom: reconstruction loss per epoch graph of the same network. (both losses are taken from the same training example of "Third Antikristos")*

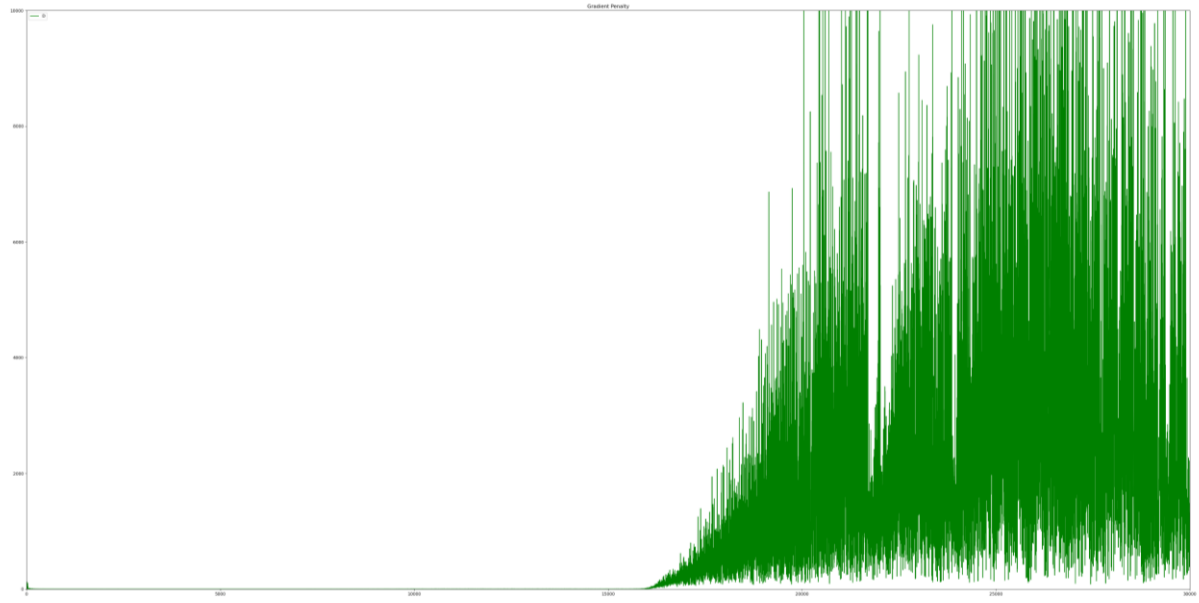

Figure 4.6 *Our InGAN-based network's Gradient Penalty per epoch graph (taken from the same training example of "Third Antikristos" as Figure 4.5).*

The animation produced by our InGAN-based model was a summarized version of the initial dance sequence, taken at 15,000 epochs. The model successfully generated a shorter version of the dance sequence while maintaining the basic dance figures such as "sitting" and "slapping the foot". The result is of high-quality as displayed in the left panel of Figure 4.7, with a minor issue: the occurrence of several abnormal pelvis translations. As illustrated in the "sit" figure example in the right panel sequence of Figure 4.7, instead of the pelvis moving downwards, the knees bend, and the legs move upwards. This error was consistently present in every "sitting" motion generated by the model. Unfortunately, similar to GANimator, the skeleton also slowly moves away from its starting position as time passes.

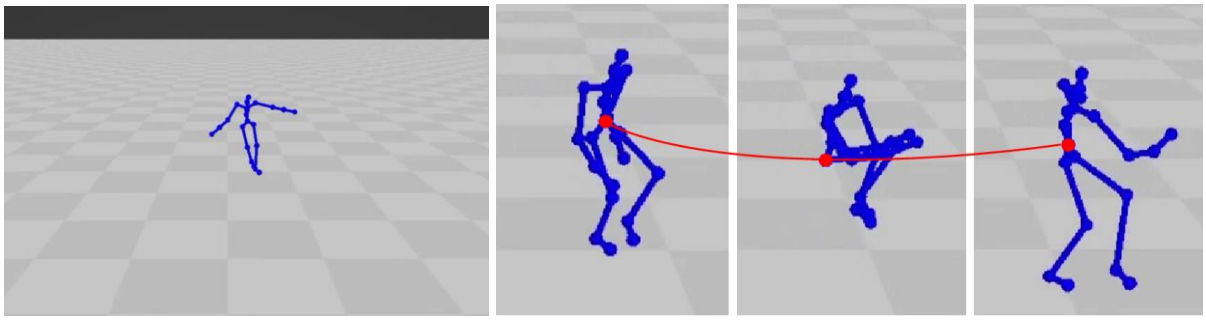

Figure 4.7 *Snapshots of the generated motion from using our InGAN-based network. Left: a sample frame of the result. Right: the abnormal pelvis translation. Instead of moving downwards during a "sit" figure, the pelvis stays around the same height, while the legs move upwards.*

4.6 SinDDM-based Model

In Figure 4.8, the total running loss for our SinDDM-based model, which was the MAE loss

between the original motion and the denoised motion at each epoch, appears to gradually decrease as the number of epochs increases, similar to our InGAN-based model. As there was no other loss or gradient penalty, we had to test different checkpoints of the model to see where we had better results. Our best results were taken at the final checkpoint, which was at 480,000 epochs.

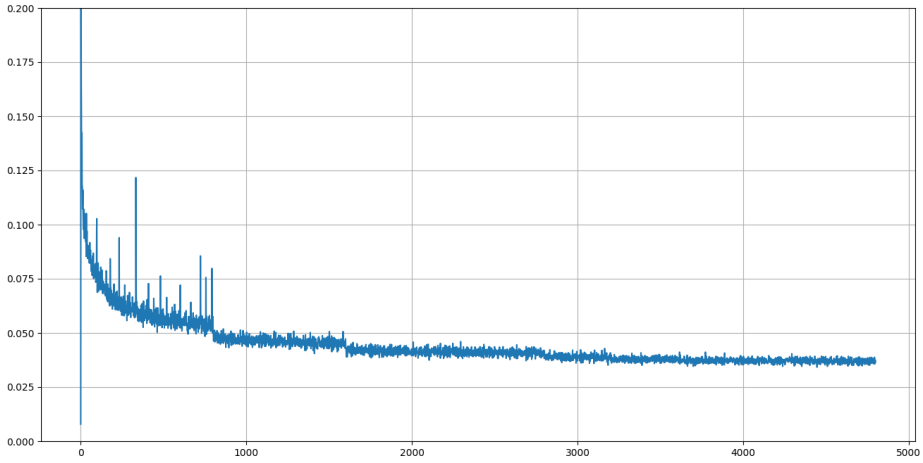

Figure 4.8 *Our SinDDM-based network's running loss (MAE) per epoch graph.*

However, as shown in Figure 4.9, the results are unsatisfactory. Despite the generated animation being shorter or longer as desired and keeping the "rhythm" of the dance (noticeable from the pelvis translations), it fails to maintain a correct pose for the dancer. For many frames, both the arms and legs of the skeleton perform fast and noisy movements, while the pelvis moves up and down, sometimes going below the floor plane. Only for very few frames can a more normal pose be seen.

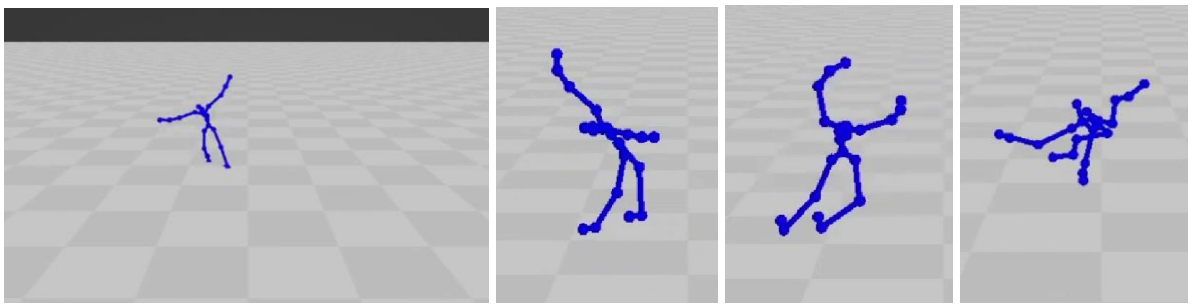

Figure 4.9 *Snapshots of the generated motion from using our SinDDM-based network. Left: a relatively normal sample frame of the result. Right: abnormal and noisy poses of the skeleton.*

The reason behind these abnormal poses is the distortions in the motion matrix representation. To understand what was going on, we tried visualizing the matrix as an image by normalizing the matrix values, multiplying them by 255, and casting them to int values to resemble a grayscale image. The image in Figure 4.10 is the original "Third Antikristos" dance sequence in its image format. We then gave this image as input to the original

SinDDM to see the results. After training on this image, SinDDM produced the results shown in the top left corner of Figure 4.11 (same size as original) and the top of Figure 4.12 (double the length of the original). We noticed that the horizontal lines, most of which represented rotations for specific joints in 6D format, were not straight in the resulting images. Moreover, the blurring removed certain pixels that had a significant impact on our motion. Therefore, training should not have been done on an image but on the matrix representation itself, which was our initial approach.

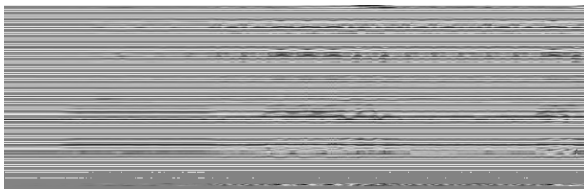

Figure 4.10 *The original "Third Antikristos" dance sequence in our proposed motion representation as an image.*

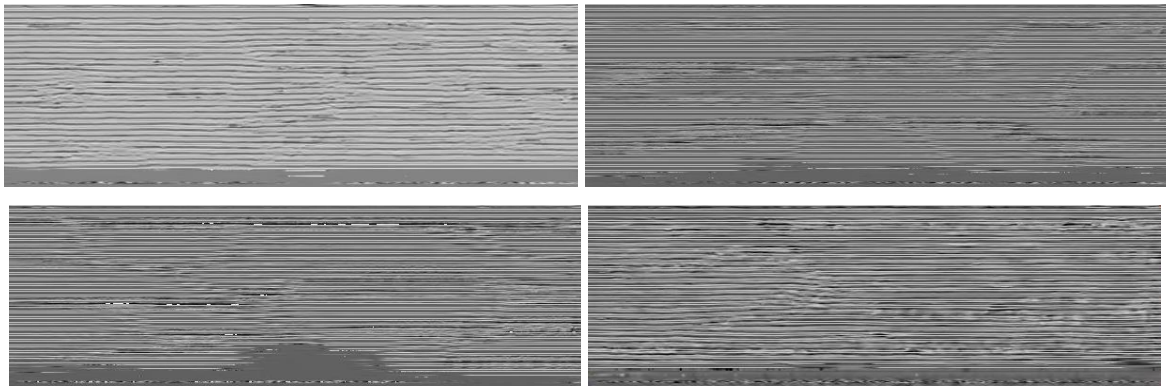

Figure 4.11 *Four samples of the "Third Antikristos" dance sequence in our proposed motion representation as images. Top Left: result of original SinDDM trained on the image of Figure 4.10. Top Right: result of our SinDDM-based model trained with 1D scaling and normalization. Bottom Left: another sample result of our SinDDM-based model trained with 1D scaling. Bottom Right: result of our SinDDM-based model trained with 1D scaling but for more epochs.*

We tried several methods to improve the results, including normalizing the motion matrix values, increasing the total number of epochs, modifying the loss factor to change the number of sampling timesteps, and performing 1D scaling, i.e., modifying the image length only in the pyramid. 1D scaling managed to fix the non-straight lines, but there was still significant noise and distortions. For some reason, even though scaling was only performed horizontally, the model copied values to upper or lower lines, as shown in the mid-lower area of the low left corner image in Figure 4.11. These distortions resulted in abnormal movements in the generated motions. A higher loss factor for sampling resulted in even noisier results, as seen in the bottom of Figure 4.12. More epochs resulted in worse results, as illustrated in the bottom right of Figure 4.11, as the model overtrained. Normalizing the matrix values to be between 0 and 1 did not provide significant changes.

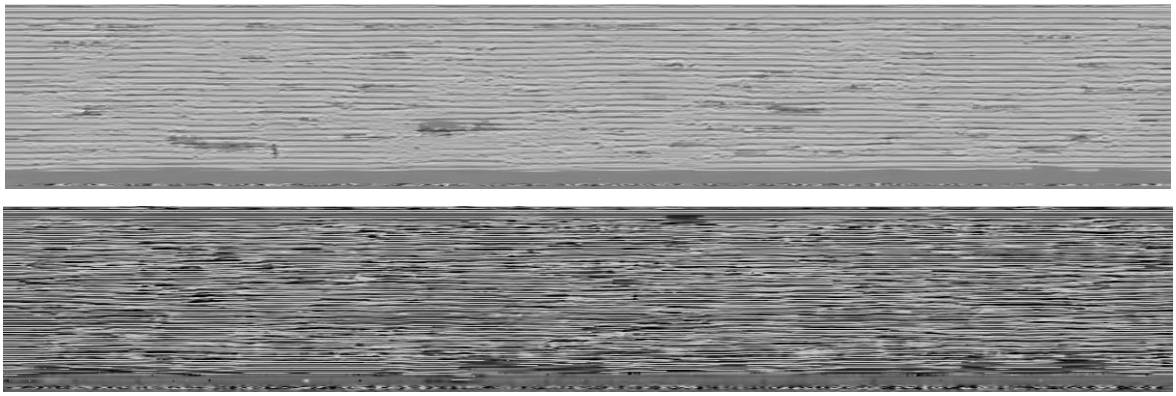

Figure 4.12 *Two scaled samples (x2 length) of the "Third Antikristos" dance sequence in our proposed motion representation as images. Top: result of original SinDDM trained on the image of Figure 4.10. Bottom: result of our SinDDM-based model trained with 1D scaling but sampled with a very high loss factor.*

Chapter 5

Discussion

5.1 Implementation Difficulties

During the implementation of our networks, we faced some difficulties that slowed down the implementation process. The main problem we had was the GPU's memory limitations. Due to limited memory in the GPU, less data could be transferred from the CPU and processed in the GPU in parallel, resulting in smaller data batch sizes. However, reducing the batch size meant that the models had to train on more batches over a longer period of time for the learning process to take effect. Taking into account training details in [9], the diffusion model needed four times the initial number of epochs, resulting in training times up to 20+ hours, given that no other process was using the GPU in during the training period. GPU memory limitations also resulted in the InGAN-based model crashing every time there were insufficient resources (mostly due to another process running in parallel). This could happen at any point during the training process, which took approximately 10+ hours. These long training times also worsened the fact that to see the results of something as simple as a parameter change, we had to wait until the next day, as the model had to complete its training. Another challenge that remains unsolved were the distortions of the input motion through SinDDM's sampling process, as shown previously in the results section. The source of these distortions could not be identified and as the distortions remained, they prevented us from being able to determine whether certain parameter values improved the model's performance.

5.2 Limitations

Despite our efforts, both the InGAN-based and SinDDM-based models have limitations when it comes to achieving the general goal of this study. Firstly, since both models are unsupervised and require no additional information other than the input motion, they lack semantic understanding of the motion's content, leading to unnatural results. In our first network, we encountered minor issues with animation quality, and as shown in the result section, abnormal pelvis translation problems occurred, where the pelvis does not move in the y-axis as it should. For example, during a "sit" motion, the legs bend and move upwards instead of the pelvis moving downwards. We also encountered problems with the translation of the skeleton, whose position cannot be controlled to remain within a designated range from its initial position. In our second network, we have not yet identified the cause of distortions, which results in very low animation quality and abnormal movements. Furthermore, we currently lack a way to control the generated motion content of the second model, meaning we cannot identify the unique and common parts of the motion sequence and modify their durations accordingly.

5.3 Future Work

Although we encountered implementation difficulties and limitations with our models, we successfully developed a single-motion generative GAN-based model for motion generation and laid the foundations for a single-motion generative diffusion model capable of summarizing and expanding motion sequences. In the future, we intend to search for ways in which our networks can achieve a semantic understanding of the motion content in order to eliminate unnatural generated movements. Moving forward, we plan to address issues with animation quality in our InGAN-based model, while also fixing the distortion issues and expanding our SinDDM-based model. Our major goal is for our networks to selectively summarize and expand only specific parts of motion while retaining the unique sequences intact. We believe this can be achieved by improving the content discriminator in our InGAN-based model using tf-idf processing for calculating the total loss, as shown in Figure 5.1. This implies the use of a larger database of different motions, where we can apply the ideas introduced in [2]. We plan to work over the summer to refine our first network as described and explore ways to implement these improvements in our second network after addressing its distortion issues. Additionally, we plan to tackle translation problems in the

first network by incorporating a global pelvis translation loss. Finally, we aim to rearrange the motion representation for both networks to use dual quaternions instead of 6D rotations, as suggested in [1]. Our ultimate goal is to present our work at a major conference in character animation, such as the EuroGraphics 2023 conference.

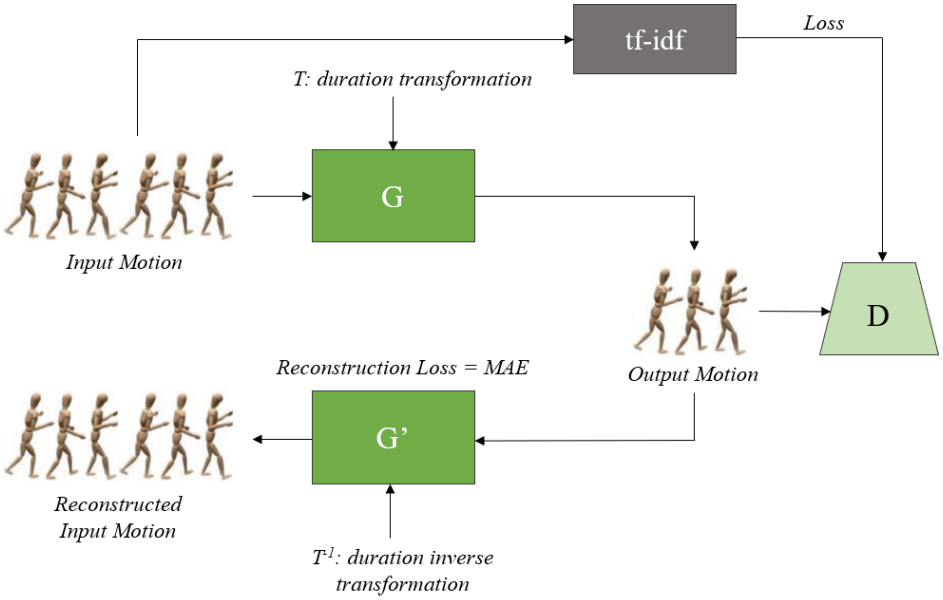

Figure 5.1 *Our InGAN-based model's future architecture. Tf-idf processing will be performed to the input motion in order for the discriminator to recognize unique and common motion words.*

Chapter 6

Conclusions

6.1 Conclusions

Summarizing or expanding a motion sequence has a wide range of practical applications, as it can reduce the cost of motion capture and enable animators to modify existing motions to fit different durations or contexts, in a more efficient manner. Therefore, developing a solution for this problem is crucial. In this dissertation, we show the potential of Generative Adversarial Networks (GANs) and Denoising Diffusion Models (DDMs) in character animation, as they have enabled the creation of a model that can summarize or expand a given motion while keeping its temporal distribution intact and coherent. This approach has several advantages over other methods, such as the ability to handle big changes in the animation duration and joint rotations of the skeleton, without altering the speed of the animation. While other methods exist, such as frame removal and seam carving, they do not offer satisfactory results, as they alter the motion speed and lack control over the parts of the motion being modified. More recent techniques such as GANimator offer high-level results, but are unable to control the translation of the generated motion. However, the use of generative models shows promising results and points to a new direction for further research in the field of character animation. In this dissertation, we test all these approaches and prove that it is possible to create a network able to learn enough information from a single motion by using overlapping patches and then summarize and expand it. This was done by modifying two models used for image processing: InGAN and SinDDM. Our modified networks can work with BVH files and 6D representation, can maintain temporal coherence, and avoid foot sliding in the output animation using foot contact labels. Our InGAN-based model can further be extended to alter the duration of an animation by adjusting the frequency of specific

movements and identify the appropriate parts of the motion (unique and common motion words) to alter. Overall, our approach shows promising results in achieving our goal of summarizing or expanding a given motion while maintaining its temporal distribution, content, and coherence intact. It offers a new direction for further research in the field of computer animation and has potential applications in various industries, such as film, video games, and virtual reality.

References

[1] N. Andreou, A. Aristidou, and Y. Chrysanthou, "Pose Representations for Deep Skeletal Animation," in ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA), 2022

[2] A. Aristidou, D. Cohen-Or, J. K. Hodgins, Y. Chrysanthou and A. Shamir, "Deep Motifs and Motion Signatures," ACM Trans. Graph., vol. 37, pp. 187:1-187:13, 2018.

[3] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein Generative Adversarial Networks," in Proceedings of the 34th International Conference on Machine Learning, D. Precup and Y. W. Teh, Eds., vol. 70, pp. 214-223, PMLR, 2017.

[4] S. Avidan and A. Shamir, "Seam carving for content-aware image resizing," ACM Trans. Graph, vol. 26, no. 3, p. 10, 2007

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative Adversarial Nets," Advances in Neural Information Processing Systems, vol. 27, 2014.

[6] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs," arXiv:1704.00028 [cs.LG], 2017.

[7] J. Ho, A. Jain, and P. Abbeel, "Denoising Diffusion Probabilistic Models," arXiv preprint arXiv:2006.11239, 2020.

[8] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," arXiv:1412.6980 [cs.LG], 2017.

[9] V. Kulikov, S. Yadin, M. Kleiner, and T. Michaeli, "SinDDM: A Single Image Denoising Diffusion Model," arXiv preprint arXiv:2211.16582, 2022.

[10] P. Li, K. Aberman, Z. Zhang, R. Hanocka, and O. Sorkine-Hornung, "GANimator: Neural Motion Synthesis from a Single Sequence," ACM Transactions on Graphics (TOG), vol. 41, no. 4, pp. 138, 2022.

[11] T. R. Shaham, T. Dekel and T. Michaeli, "SinGAN: Learning a Generative Model from a Single Natural Image," in IEEE International Conference on Computer Vision (ICCV), 2019.

[12] A. Shocher, S. Bagon, P. Isola and M. Irani, "InGAN: Capturing and a the "DNA" of a Natural Image," in IEEE/CVF International Conference on Computer Vision (ICCV), 2019.

[13] Y. Zhou, C. Barnes, J. Lu, J. Yang and H. Li, "On the Continuity of Rotation Representations in Neural Networks," in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019

[14] “Dance Motion Capture Database,” University of Cyprus, 2013. [Online]. Available: http://dancedb.eu/.