Bachelor of Science Thesis

HOW TO DESCRIBE THE CONTENT OF MOTIONS? MOTION SUMMARIZATION USING GENERATIVE ADVERSARIAL MODELS (GANS)

Pieris Panagi

University of Cyprus

Computer Science Department

May 2022

University of Cyprus

Computer Science Department

How to describe the content of motions? Motion summarization using generative adversarial models (GANs)

Pieris Panagi

Research Supervisor

Andreas Aristidou

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science at the University of Cyprus

May 2022

ACKNOWLEDGEMENTS

First of all, I would like to express my gratitude to my advisor Assistant Professor Andreas Aristidou for the guidance he provided me during the conduction of this project. His support and knowledge have been essential for me during the course of this study, and he was always available when I needed his assistance.

Additionally, I would like to thank Andreas Lazarou, for being there to help me every time I needed his assistance. The discussions I had with both of them and the exchange of ideas we had helped me to better understand and develop the project at hand.

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

Abstract

Motion capture has proven itself as the most effective technology to capture and digitalize robust movements. Despite that, there are only a few ways to modify a captured motion sequence, which are either ineffective or time consuming. Creating a method able to summarize or expand a captured motion sequence was not possible before the introduction of Generative Adversarial Networks (GANs). GANs are a machine learning method which has proven very effective in creating realistic generated data. Inspired by the image processing community and their success in creating networks able to modify the size of an image using GANs we proceeded to design and implement the first method able to modify the length of a motion sequence. Our network is able to create motion sequences in various sizes, based on the original motion given, without the need of a large dataset. Using the ideas introduced in image processing we proceeded with the idea of progressive training to create our network. Instead of using samples of data, we use patches of our input itself we train our network and create coherent and realistic motion sequences. In addition, our work opens up the opportunity to create additional applications in the field of Character Animation with the use of GANs. At the same time we have provided a method which can be used when motion sequences of specific duration are needed or when we need to create highlight of the said motion sequence,

Contents

Chapter 1

Introduction

1.1 Motivation

Motion capture is used in a number of different industries nowadays, including video games and movies to capture and digitalize dynamic movements. However, when a motion capture sequence needs to fit in a bigger sequence, the duration needs to be perfect. If not, only a few options with major weaknesses are left. One option is to divide the sequence in a set of smaller motion clips and then proceed to reunite these movements, whilst removing small parts of them if there is a need to reduce the duration of the sequence. In order to blend these clips together, small movements are used in between them to make the result coherent. However, this method cannot give decent results if we need to make big changes in the duration of the sequence or if the motion is such that big changes in the rotation of the skeleton occur. A better way to understand this is to think of an image we want to reduce in size. In that scenario, we would divide the image in a number of smaller ones, remove parts of their edges and try to reunite them by blending them using new pixels in between them. If there is a big difference between the two images we try to blend, the result will not be coherent. The other option we have is to just modify the number of frames shown per second. In the example with image, the respective method would be to just resize the image. The problem here is that this would change the speed of the motion, thus alternating the distribution of our motion sequence, in the same manner an image would change the ratio of its object when resized.

Our aim was to find a solution which would be able to expand or summarize a movement, while keeping its distribution intact and coherent. Whilst until now these methods were

the only ones available to change the duration of a motion sequence, with the evolution of machine learning and the introduction of Generative Adversarial Networks (GANs) [1], we aim to present a network able to summarize or expand a motion whilst keeping its distribution intact and coherent. This would also allow us to preview the highlights of a dance or sport sequence, whilst keeping its important parts intact, in a smaller clip. In addition, with the rapid growth of GANs, especially in the field of Image Processing [2] [3] [4], able to be trained with Progressive training and thus without the need of a large dataset, we are able to combat the issue created by the lack of big dataset for specific types of movements.

Figure 1.1 Motion Capture is used in the Video Game Industry to capture robust and realistic movements. ©Ubisoft Montreal

1.2 Contributions

During the conduction of this dissertation, we succeeded in creating the first network able to summarize or expand a motion, without the need of a dataset, whilst keeping its distribution intact and the resulted motion coherent. This solves the problems created when using motion capture sequences, thus proving our major technical contribution, as parts of larger sequences in movies and video games as it allows:

- The expansion of the captured sequences to meet certain duration constraints.
- The summarization of the captured sequences when it is required to keep the content of a large sequence in shorter motion clips.

Adding to the above, our network makes an important application as well, as it can create highlights of a motion sequence, such as dance or sports, in shorter meaningful clips, without the need of manually choosing the important parts of a motion and disturbing the coherence of the motion.

Finally, we make an important scientific contribution as we prove that it is possible to create a GAN able to learn through progressive training, by collecting enough information about the input motion by just using overlapping patches of the motion itself.

Chapter 2

Literature

The introduction of Generative Adversarial Networks has opened the opportunity for the implementation of outstanding applications, which could not be created in the past. With their ability to create realistic fake data based on real samples, GANs have been used in a number of fields with many different applications. Our inspiration was their applications in the field of Image Processing and especially their capabilities in image summarization. GANs have shown major potential when used to modify the size of images or textures, without the use of datasets. Instead, many works, that have come out recently, use the idea of progressive training, by either using overlapping patches of an image in different sizes or overlapping crops of the image in the original size. We will get in more detail about these applications in this chapter.

2.1 Generative Adversarial Networks

A number of works have come out in the recent years, which aim to summarize an image with the use of GANs. Using their work as a foundation we proceeded to the implementation of a network able to summarize a motion sequence. Before getting in detail about these works, we have to understand what GANs are and how they work.

GANs [1] is an architecture of unsupervised machine learning, able to generate realistic fake output. The main idea behind GANs is that we have two networks competing against each other. Specifically, we have a model trained to generate fake output, called Generator and a model called Discriminator, which takes samples of real data and data generated by the Generator and tries to classify them as real or fake. The Generator wants the Discriminator to fail as much as possible, thus creating realistic fakes.

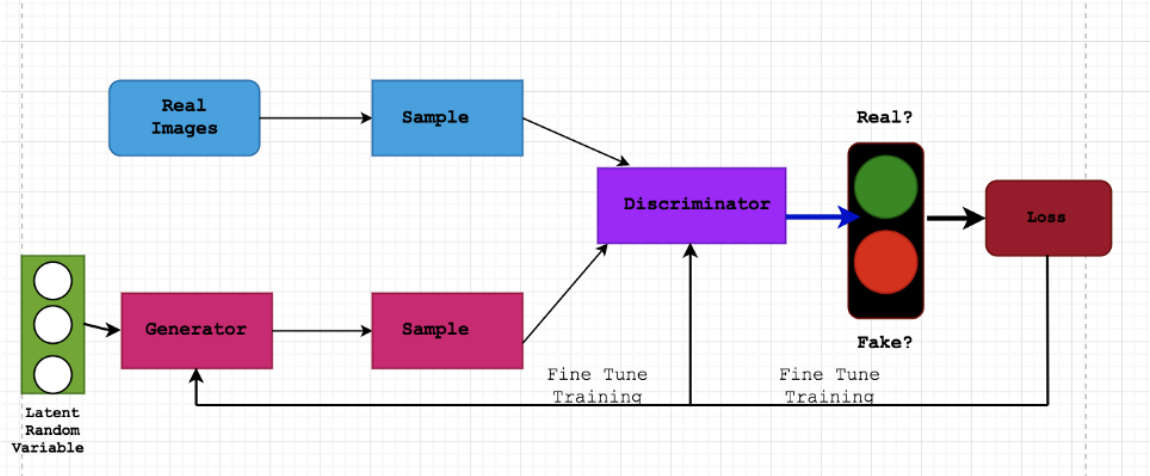

Figure 2.1 The main architecture behind GANs. The Generator is trained to create data, using a dataset and a random noise. Fake and real data are then feed to the Discriminator, which tries to classify which are real and which are fake. Both networks use this game between them to become better at their tasks which leads the Generator to create realistic fake data.

This architecture has found a lot of application in recent years, especially in the fields of image processing, computer vision and Animation.

Figure 2.2 The first examples of the GANs architecture in Image Processing as presented in [1]. The images in the yellow boxes are Generated and the rightmost sample of each row shows the nearest training sample.

2.2 Training GANs with a single image

Two works [2] [3] have come out in the recent years which attempt to introduce models based on the GANs architecture, using progressive training and patches as input for their networks. This has inspired us to proceed and use this idea to create a network able to achieve similar result but for motion sequences instead of images.

SinGAN is one such work. It can create fake versions of the input image in different sizes. To do so it uses a random noise as a starting input. This noise is feed to the Generator, which creates the fake image for the first scale. As a result, many different output images of the same size can be created which depend on the random noise generated by the network.

Figure 2.3 Examples of SinGAN. The images on the first column are the inputs used for each row. The second and third columns consist of random samples of the same size. The last two columns demonstrate examples of the summarization and expansion capabilities of the network.

The model consists of a pyramid of Generators with different scales of the input image. Specifically, we start by resizing the image to a smaller size and using a noise to begin generating fakes using the Generator. We feed the discriminator with patches of the fake and real image and the discriminator attempts to classify if each patch comes from the one image or the other. We repeat this progress until the patches become realistic enough to fool the discriminator. We continue by progressively making the input image larger until it reaches its original size. During this progress we keep the size of the patches given to the Discriminator the same in all scales, in order to start with bigger parts of the image as input in the first couple of scales and then continue with the finer details in the later ones.

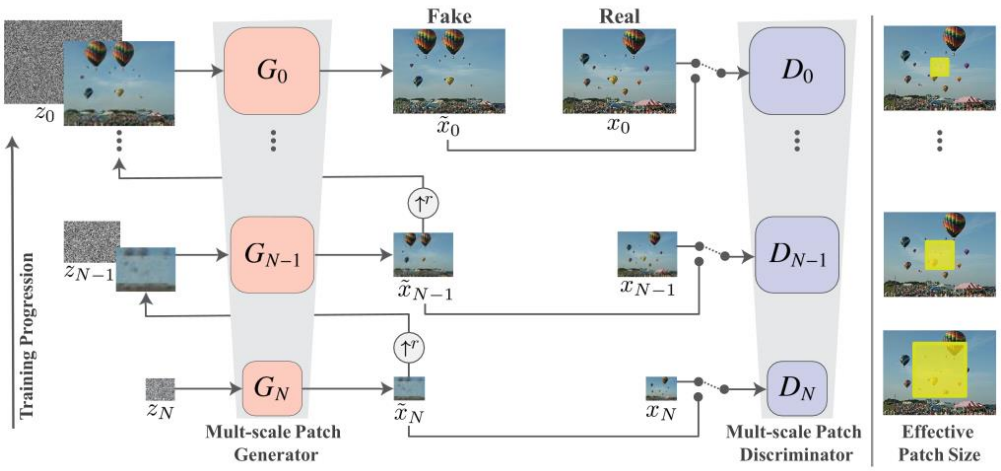

Figure 2.4 The main structure of SinGAN. It starts with a random noise to generate the first fake samples. It then feeds D_N with patches of the real and fake image. After the first scale is completed, the image generated by G_N is used in the next scale along a random noise to train G_{N-1}. This process is repeated for all the scales of our pyramid.

The other work introducing a similar idea in image summarization using GANs is called InGAN. The authors of this work use different constraints to keep the distribution and the localization of the original image intact in the output. To achieve that, they take advantage of the fact that InGAN is automorphism. They invert the Generator of their network and try to reconstruct the original image. They then proceed to calculate a loss which corresponds to how similar these two images are, and this results their network to retain the distribution of the original image. To also preserve the position of each object in the image they designed their network in such a way that each output pixel only depends on specific pixels of the input image.

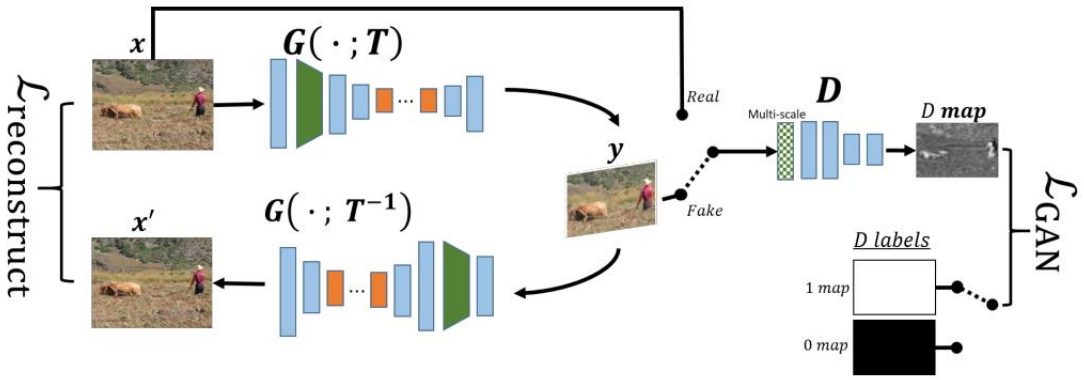

Figure 2.5 The main Architecture of InGAN. A Generator attempts to recreate the input image to the requested size. A multi-scale discriminator tries to classify patches of the two images as real or fake. At the same time an inverted Generator attempts to reconstruct the input image.

Simillary to SinGAN the Discriminator is trained using pacthes of the original image and the fake one as input.

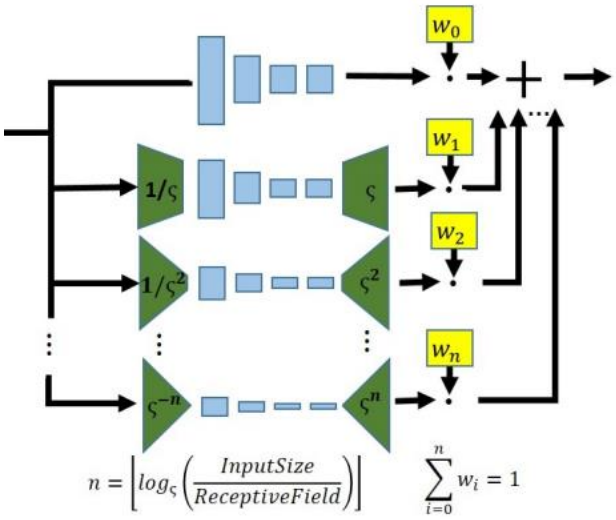

Figure 2.6 The Architecture of the multi-scale discriminator.

Figure 2.7 Examples from InGAN. The image in the red box is the input. Both examples of summarized and expanded images are shown

The major difference between these two works is that, whilst both of them use patches in downsampled versions of the image, the first uses a random noise, along the input image, while the latter uses the fact that its Generator can be inverted in order to guarantee that the distribution and localization of the original image is kept intact. We created our network based on the network introduced in SinGAN, as we will show in more details in the later chapters. On the other hand, the idea introduced in InGAN can be used to make

sure that we can create summarized and expanded Animations which satisfy these requirments. We aim to explore this possibility in the future.

2.3 Deep Motifs and Motion Signatures

In order to use the idea of progressive training in animation we need divide our motion sequence in smaller motion sets. This work [5] introduces the idea that motion sequences can be broken down to smaller movements which can be used to describe the motion and to find its distribution.

This work proposes that large motion sequences can be divided to smaller movement and thus represented by the distribution of these movements, A network is trained using motion words, which is "a narrow temporal-window of all joint rotations around a given frame" [5]. A motion sequence is basically described as a number of overlapping motion words. The network then maps each motion word in a latent space R^d based on the similarity between the motion words. All motion words are then grouped in clusters. Each cluster has a motif motion word as it's centroid. As a result, any new motion words can be classified in a cluster by finding the motif with shortest distance.

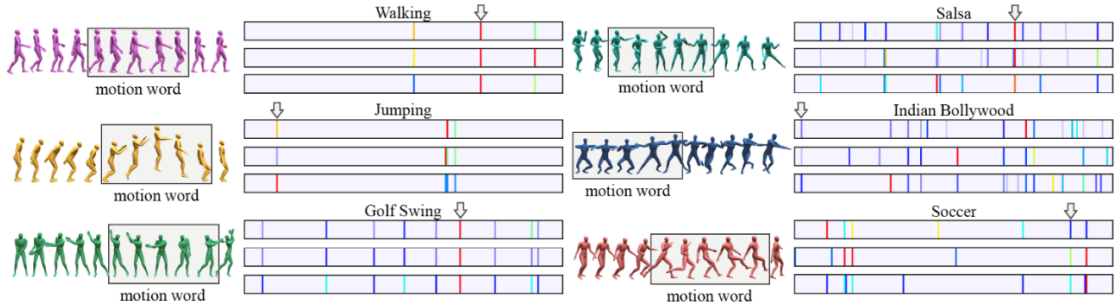

Figure 2.8 Each horizontal bar shows the frequency of a motion motif. Specifically, the frequency is color coded from red (high) through blue (low) to gray (zero). It is important to note that these signatures represent the frequency and not the time-evolution. Three signatures are shown for each type of motion. The motif pointed by the arrow above the signature are associated with the motion words illustrated in the respective rectangle on the left,

In addition, this work introduces motion signatures which are the normalized histogram of the words in all k cluster for a motion sequence. We can also calculate the distance

between two motion sequences by just calculating the distance between their motion signatures.

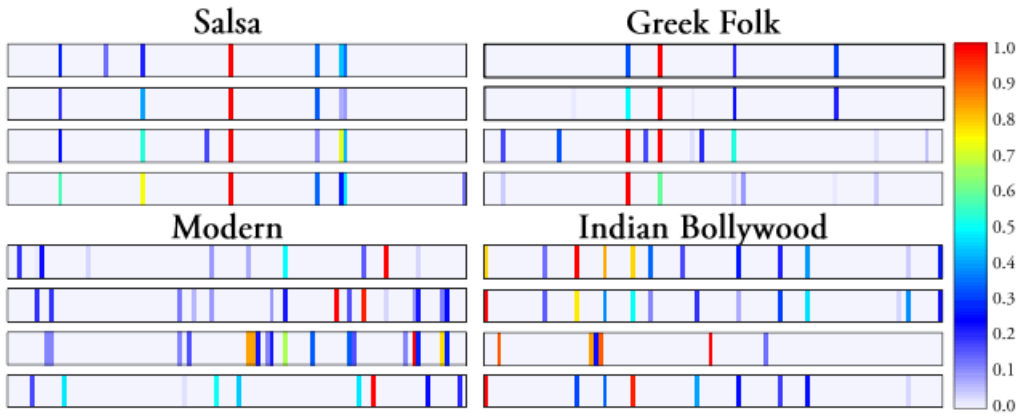

Figure 2.9 Four motion signatures are shown here. We notice that different dances have different distribution, Modern Dancing has a larger distribution of motif as it is more dynamic, compared to the more structured Greek Folk dancing.

2.4 Non-Stationary Texture Synthesis by Adversarial Expansion

This was another network [4] used to modify the size of images that picked our interest. The major difference of this network, in comparison to the two we saw before, is that it uses overlapping crops of the given image instead of patches of downsampled versions of the image. Since this network is only able to expand an image, we did not proceed with any experiments using this network.

Figure 2.10 Examples of four non-stationary in the middle with their respective result on the left and right.

This study aimed to create a network able to get an image of a small texture as input, and create an expanded version of it, whilst keeping the main characteristics of the original input intact. To manage this the network uses k*k crops of the original texture and use a Generator which aims to create expanded networks of the given image of 2k*2k size. It then continues by giving a Discriminator the generated images and cropped images from the original image of the same size. The discriminator aims to classify each image as fake or real, respectively. This process is repeated multiple times with overlapping crops of the original image. It is important for the value chosen for the crop sizes (k) to be large enough to capture enough information about the texture, but small enough to generate enough overlapping patches of the original image. Once the network is train it can generate expanded versions of the original texture in different sizes.

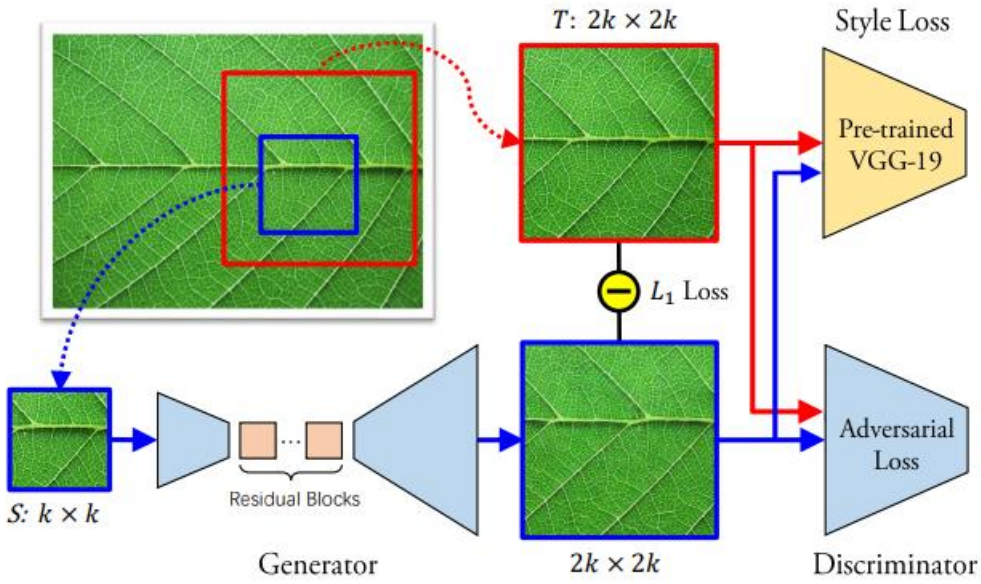

Figure 2.11 The main architecture of the network. The Generator learn to expand cropped images of the original. The Discriminator tries to classify if an image is generated or if it derived from the input.

2.5 Seam Carving

The last work tackling image summarization, we investigated is Seam Carving [6]. Seam Carving uses the distribution of the pixels of the given image to decide which to remove or add, in order to summarize or expand the image respectively. When we used this idea in motion, we received a coherent result. The main weakness of seam carving was that the resulted motion sequence did not maintain the distribution of the original, making it

faster or slower accordingly. These results were used to compare the results of the different methods as they did achieve the general target of expanding or summarizing the motion sequence. Our aim was to achieve an even better result, which would not modify the speed of the motion sequence.

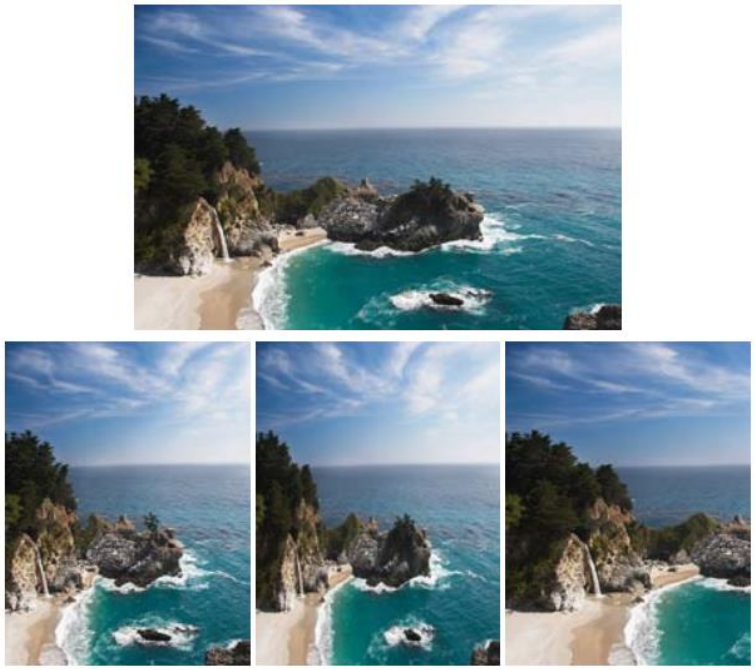

Figure 2.12 Comparison between seam carving and other methods when the aspect ratio changes. The image at the top is the input. On the bottom we have from left to right: Seam Carving, scaling and cropping.

The main idea of this method is that we can remove pixels that blend with their surrounding and are as a result unnoticeable. To keep the shape, coherence and content of the original image they proceed to remove the seams ranked with the lowest energy and are as a result less important. Seams are paths, either horizontal or vertical, of pixels connected from left to right or from top to bottom. Similarly, to enlarge an image this method simply adds seams by averaging their left and right neighbors, for vertical seams or top and bottom neighbors for horizontal seams. The number of horizontal and vertical seams removed or added depend on the difference in the aspect ratio of the output image in comparison to the input.

Chapter 3

Network

3.1 Overview

We will start by talking about the general idea behind our network. Our application starts with a BVH file as input. It then to change the pose representation of our motion from Euler Angles, which is the way motion is represented in the motion section of our BVH file, to a representation where each rotation is represented by 6 values (6D), instead of 3 [7]. Our input then goes through a multi-scale Generative Adversarial Network and the output motion which consists of 6 values for each rotation is transformed back to Euler angles to create a new BVH file as output of our application. Our multi-scale Generative Adversarial Network uses the idea described before in SinGAN but with several modifications in order to generate descent results with motion, but we will discuss these modifications in depth later. The general design of our network is that we create a pyramid of Generators of different sizes. Each Generator gets a downsampled version of our original motion and the Generator attempts to create fake motions of the same size. Then the Discriminator is given parts, or patches, of the real and fake motion and tries to classify them as real or fake. This process is repeated multiple times with motions of different sizes.

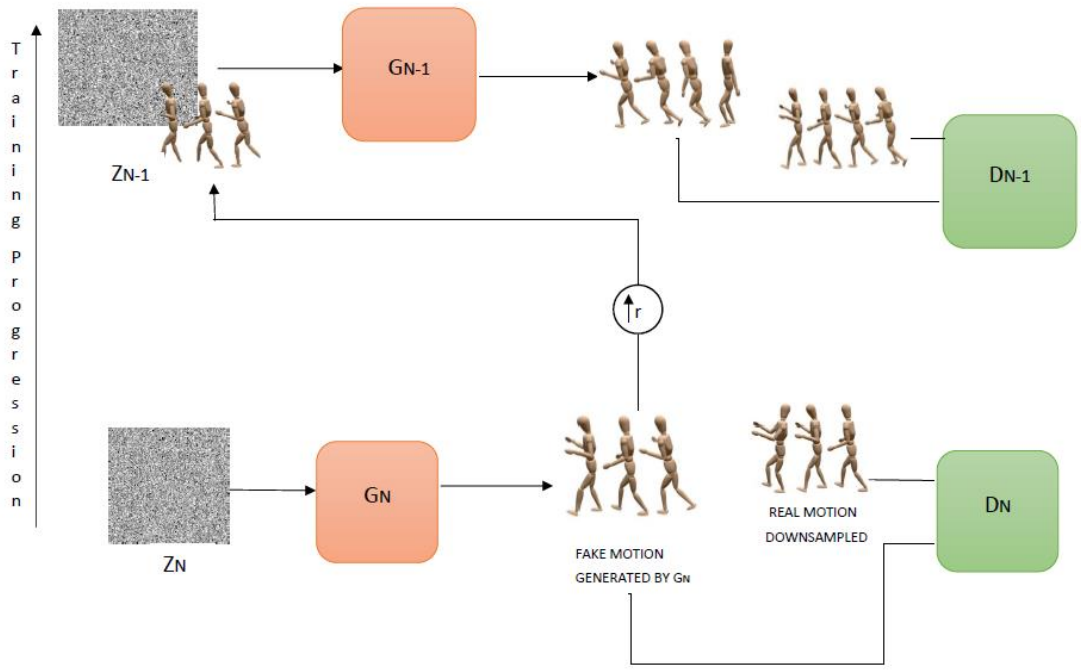

Figure 3.1 The main architecture of our network. Similarly, to SinGAN the first stage of our network is purely generative using a random noise. Patches are then given to the discriminator which tries to classify them. After the first scale the Generators also receive the output of the previous scale, along the random noise. The process shown in the figure is repeated for all scales of our pyramid.

3.2 Model Architecture

3.2.1 Pose Representation

The first step in creating our network is to choose the Pose Representation we will use as input for it. Our application receives a BVH file as input from the user. A BVH file consist of two main sections. The Hierarchy section and the Data Section. The first consists of the joints and the initial pose of the skeleton, whilst the latter consists with the rotation of each joint in each frame in Euler Angles.

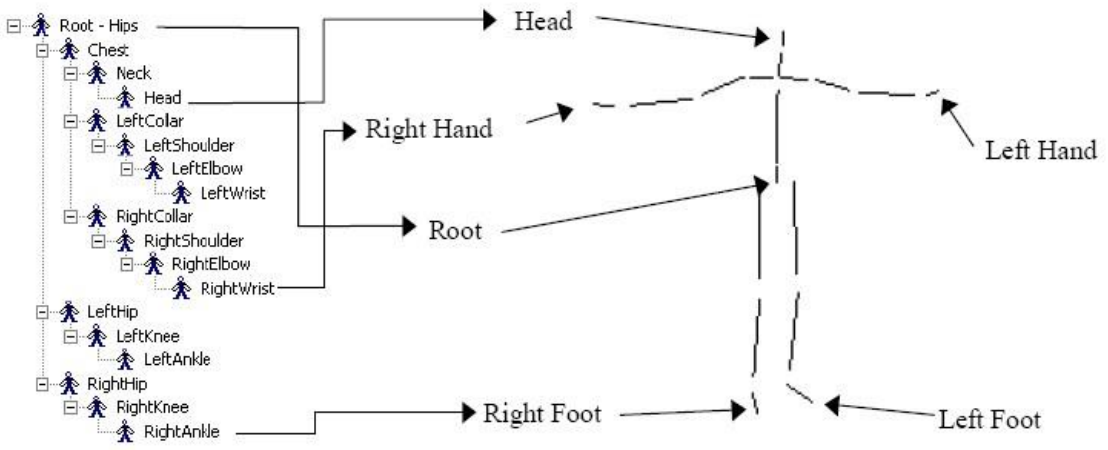

Figure 3.2 The structure of a BVH skeleton. On the left we can inspect the hierarchy of out skeleton. We can also see the corresponding points on the skeleton as show by the arrows.

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
                                        }
                                }
                        }
```

Figure 3.3 The hierarchy of the skeleton as shown on the hierarchy section of the BVH file. We have the root, which usually are the hips, and then we proceed with the rest of the joints along their starting rotations.

Source: https://www.cs.cityu.edu.hk/~howard/Teaching/CS4185-5185-2007-SemA/Group12/BVH.html

```
MOTION
Frames: 2
Frame Time: 0.04166667
-9.533684       4.447926        -0.566564       -7.757381       -1.735414       89.207932       9.763572
                6.289016        -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        -23.069189
                -18.352753      15.051558       -7.514462       8.397663        2.953842        -7.213992
                2.494318        -1.543435       2.970936        -25.086460      -4.195537       -1.752307
                7.093068        -1.507532       -2.633332       3.858087        0.256802        7.892136
                12.803010       -28.692566      2.151862        -9.164188       8.006427        -5.641034
                -12.596124      4.366460
-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        -4.378940       22.399654
                2.244878        -29.421873      -6.918557       6.131992        4.521327        -18.013180
                3.059388        -3.768287       8.079588        10.124812       5.808083        -22.417845
                -15.736264      18.827469       -8.070700       9.689109        2.417364        -7.600582
                2.505005        -1.625679       2.430162        -27.579708      -3.852241       -1.830524
                12.520144       -1.653632       -2.688550       4.545600        0.296320        8.031574
                13.837914       -28.922058      2.077955        -9.176716       7.166249        -5.170825
```

Figure 3.4 The rotations of the skeleton as shown on the data section of the BVH file. Except the rotation of each joint, through the frames, we can also learn the number of frames of this motion sequence, 2 in this case, and the frame time.

Source: https://www.cs.cityu.edu.hk/~howard/Teaching/CS4185-5185-2007-SemA/Group12/BVH.html

To train our network to proceed to represent the motion by the 6D Rotation features [7], instead of the Euler angles contained in the BVH file. We also add foot contact labels in our representation in order to combat any foot sliding artifacts.

3.2.2 Generator and Discriminator

As mentioned before our network consists of a pyramid of Generators where each generator receives downsampled versions of the input motion. To downsample the input in the appropriate sizes for our different scales we used linear interpolation. The main idea is that we use two already known data to estimate an unknown value in some point between them. For example, if we have two coordinates, the linear interpolant is the straight line between them, and we can estimate the values on that straight line as a result.

By using this method, we managed to create the input for the different scales of our pyramid. For each scale we estimated the values that are between the values we need to remove to get the wanted size for each scale. In a fashion similar to SinGAN we manage to capture the bigger motion words in the coarser scales and the smaller motion words, the “details” of our motion, as we go to the finer ones.

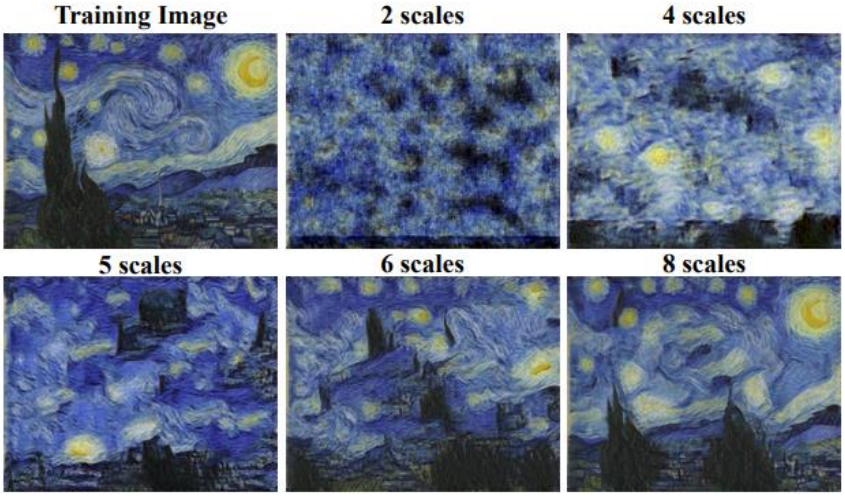

Figure 3.5 To better understand the effect of progressive training at each scale we will use this figure from SinGAN. As shown, until scale 4 the network mostly learns about the background of the image. In scales 5 and 6 smaller objects, such as parts of the tower are used. After the training is completed, the network has learned about the finer details, such as the circles as well.

We proceed by feeding patches, which are of the same size for all scales as explained before, to the discriminator. The Discriminator is competing with the Generator and the latter creates realistic patches, and thus realistic animations as a result, to "beat" the Discriminator. We repeat this process from the coarser to the finer scales to capture the bigger motion words at first and the "details" of the motion in the later stages.

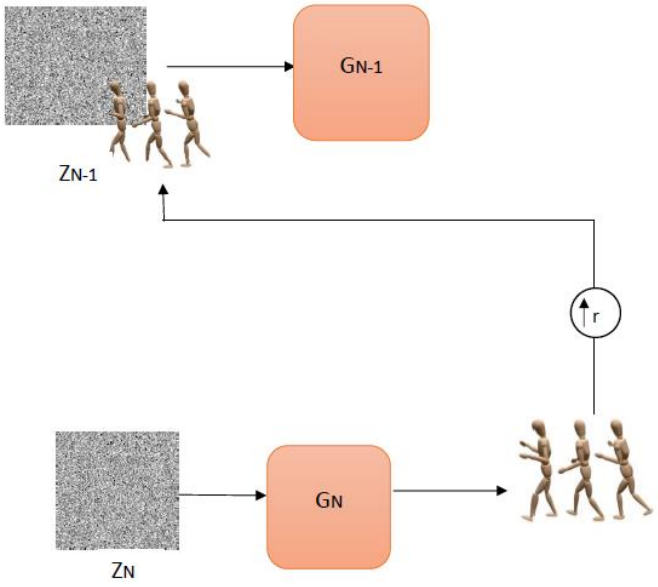

Figure 3.6 The first scale is purely generative. The fake motion sequence is created, only by using a random noise as input. In the other scales the output of the previous scale is upsampled and used as input, along the random noise.

In the coarsest scale of our network the generation is purely generative, starting by only a random Gaussian noise. Each Generator in the finer scales of our network gets a upsampled version of the previous scale as input, as well as the Gaussian Noise of the appropriate size. The result of the upsampled motion from the coarser level and the Gaussian Noise is then fed into 5 convolution layers. Then the output is added again to the upsampled image to get the final output.

3.2.3 Losses

Adding to the above, a number of losses, is also applied to our network to optimize its output. Specifically, we apply an Adversarial Loss (L_{adv}), which aims to reduce the distance between the distribution of patches, in the real animation and the generated animation of each scale. We also apply a reconstruction loss (L_{rec}) to ensure the existence of noise maps able to reproduce the real animation. Finally, we apply a temporal coherence loss (L_{coh}) [8] to ensure that the generated motion is smooth and cohered.

We use WGAN-GP loss as an Adversarial Loss. This score is calculated for the whole animation, instead of the patches to allow our model to learn boundary conditions. This score shows us the difference in the average score succeeded by the Generator in the real and fake animation.

As far as the reconstruction loss is concerned, we choose a set of input noise maps for which the following correlation is achieved $\{z_N^{\text{rec}}, z_{N-1}^{\text{rec}} \cdots, z_0^{\text{rec}}\} = \{z^*, 0, \ldots, 0\}$ where z* is a fixed noise map used in training. The reconstructed, by the set of noise map, animation is also used in training to get an idea of how many details have to be added to the animation of that scale, by taking the standard deviation of the noise map of the said scale (Zn) to be proportional to the root mean square error between the reconstructed animation of the next scale and the real animation used in that scale.

Finally, we use a temporal coherence loss to make sure that the motion is smooth and stable. We calculate the difference between two consecutive poses for all degrees of freedoms during the whole animation and we aim to reduce the angle difference between consecutive frames in order to prevent sudden changes in our output motion.

Chapter 4

Implementation

4.1 Pose Transformation

The foundation of our implementation is the code executing SinGAN. In the process of building our implementation to function with animation we faced a number of challenges, which we will discuss in detail later in this chapter. The first step in building our network was to modify its input. Specifically, we modify our network in order to receive a BVH file as input. We then isolate the data section of our file and proceed to transform it to the representation described in the previous chapter. The user also has the option to reduce the joints of their skeleton if needed, by declaring the appropriate hierarchy in the respective file. This feature is especially useful when dealing with skeletons with a large number of joints in order to reduce the resources needed to proceed with the training.

4.2 Scales

We then proceeded to calculate the number of scales needed to train our network for the input animation and the size of each scale. The number of scales is calculated by the following formula.

$$\text{First Scale} = \lceil \log_{\text{scale\_factor}}(\min(\text{max\_size}, \text{joints})/\text{joints}) \rceil$$

$$\text{Last Scale} = \lceil \log_{\text{scale\_factor}}(\text{min\_size} / \text{joints})+1 \rceil - \lceil \log_{\text{scale\_factor}}(\text{min\_size} / \text{joints})+1 \rceil$$

Number of scales = Last Scale – First Scale

The default values for the parameters of the formula shown above are as follows:

- min_size = 35
- max_size = 20000
- scale_factor = 0.75

We then proceed to calculate the scale factor which is going to be used to calculate the size of the downsampled animations using the following formula:
Scale Factor = $(\text{min\_size}/\text{joints}^{1/\text{Number of Scales}})$

We continued by creating an array with downsampled versions of our input. The size of each input is equal to the scale factor calculated before to the power of the Last Scale – 1 and is generated using linear interpolation as implemented by PyTorch.

4.3 Training our model

We are now ready to proceed with the training of our model. To train our model we used methods provided by PyTorch. We begin by generating our random noise in the size of the input animation of the given scale. After that we start by updating our Discriminator by training it with patches of the real animation. We then proceed to train our Discriminator with patches of the fake animation. Specifically, for each step of the Discriminator we generate a fake animation, using our Generator as trained thus far and the Discriminator tries to classify the patches as it was explained before. The patches of our model have a size which is equal to 32 frames. Our Discriminator aims to minimize the mistakes it makes when classifying the patches origin. We also calculate and apply the appropriate losses as described previously. The next step in training our network is to train our Generator, based on the new things our Discriminator has learned. Our Generator creates new fakes and aims to maximize the mistakes made by the Discriminator as trained thus far. This procedure is repeated for 2000 epochs for each scale. Consequently, the two networks keep learning from each other, inducing the Generator to create realistic fakes, able to fool the Discriminator, which also becomes smarter and harder to fool as

we proceed with the training. This workflow is repeated for all our scales, as calculated from the coarser, to the finer ones.

4.3 Output

The final step to finish our implementation is to use the array to get a BVH file. We first fix any foot sliding artifact by using the foot contact labels and go back to the Euler angle representation. We then create the new BVH file which consists of the initial pose of the skeleton in the hierarchy section, along with the joints and the rotations, in Euler angles, of the skeleton on each frame in the data section of our file.

Chapter 5

Results and Discussion

5.1 Data

In this chapter we will discuss the results of the model implemented as described in the previous chapters. To create our results, we use dance animations captured by the VRLab of the university of Cyprus [9] as found in the DanceDB database (http://www.dancedb.eu/).

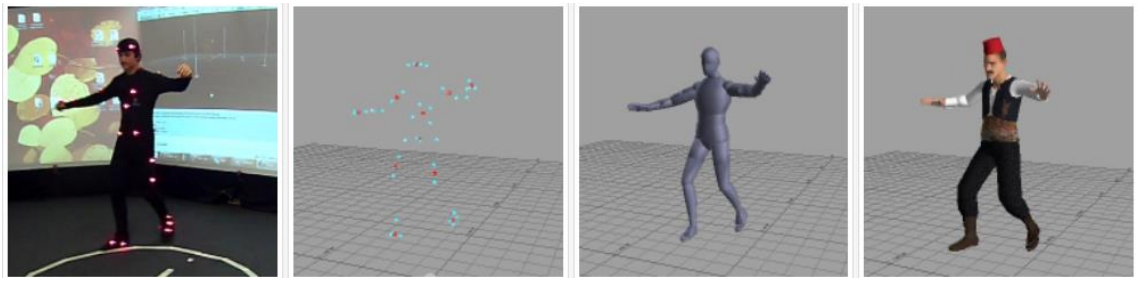

Figure 7 The process of motion capture.

All the input files used are in BVH format and consist of skeletons of 31 joints and various lengths in frames. We have reduced the frame rate of the input files, from 60 frames per second to 24 frames per second, by utilizing the MotionBuilder software, in order to reduce the size of our input data and minimize the resources needed as a consequence. At the same time the 24 frames per second are enough to keep our animation smooth and coherent. The file used to present our result consists of 1219 frames. We will represent some images of our results and input data below. The animated versions of our results and input will be shown in the thesis presentation.

5.2 Results and discussion

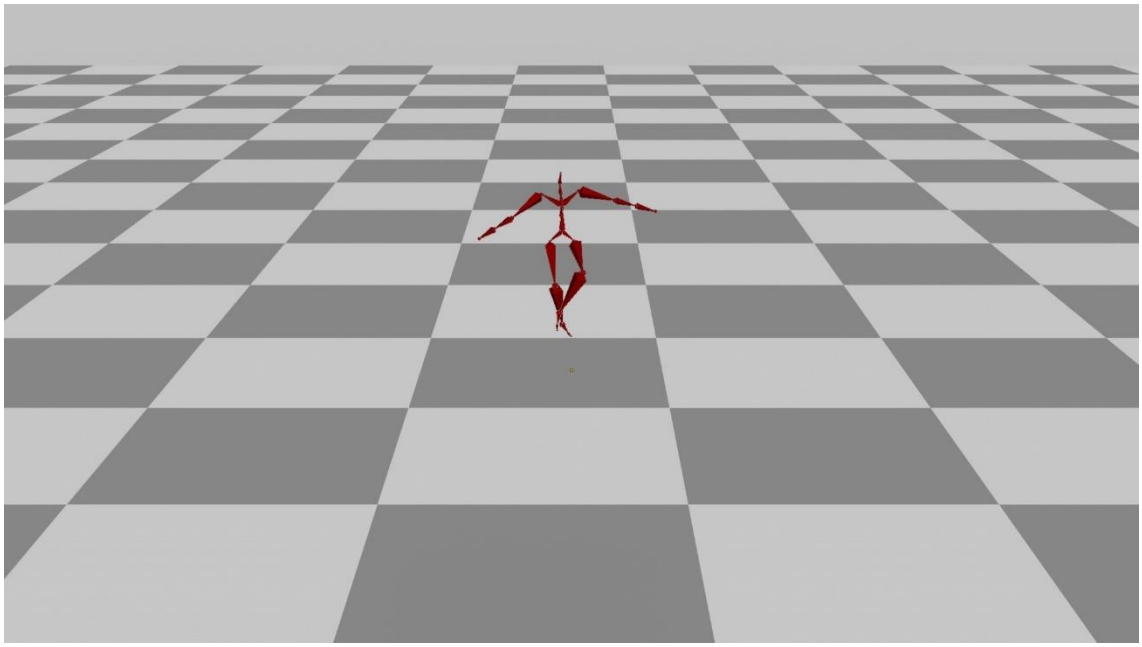

Figure 5.2 The frame 244 of our input. This is the frame in the 1/5 of our input. To present our results we will use the frame representing the 1/5 and 4/5 of our motion sequence.

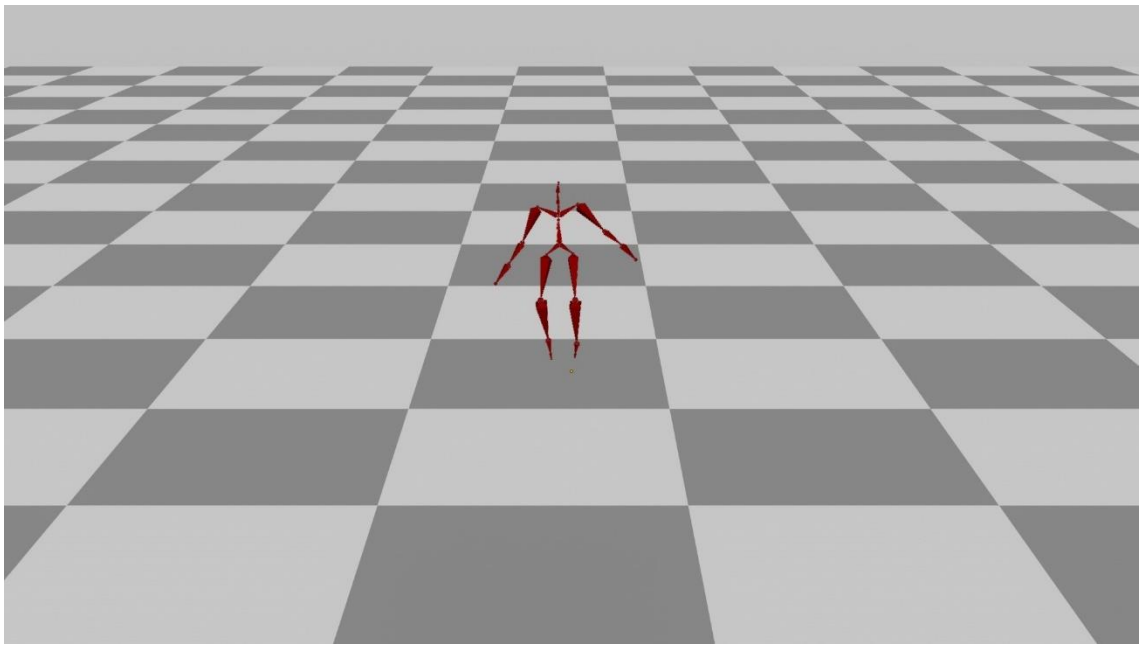

Figure 5.3 The frame 975 of our input. This is the frame in the 4/5 of our input. To present our results we will use the frame representing the 1/5 and 4/5 of our motion sequence.

In this chapter we will talk about the final results of our model. We started by trying some simpler solutions before proceeding to using GANs.

5.2.1 Frames Removal

5.2.1.1 Results

Our first effort was to try the simplest solution which was to just remove random frames in order to make the output motion shorter than the original.

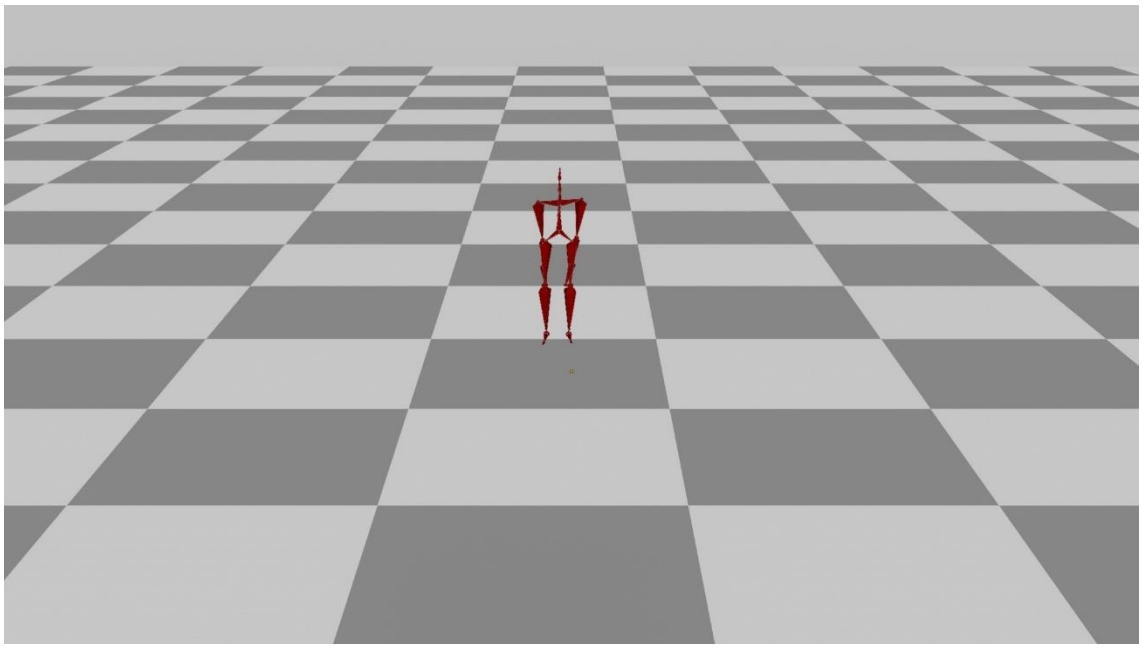

Figure 5.4 We generated a summarized motion-sequence of 600 frames by randomly removing frames. The resulted animation is not coherent. In this picture we have the frame 120 of our output, which is the frame on the 1/5 of our sequence. The distribution and localization of the sequence is not preserved either as this frame is completely different when compared to the corresponding frame of the input

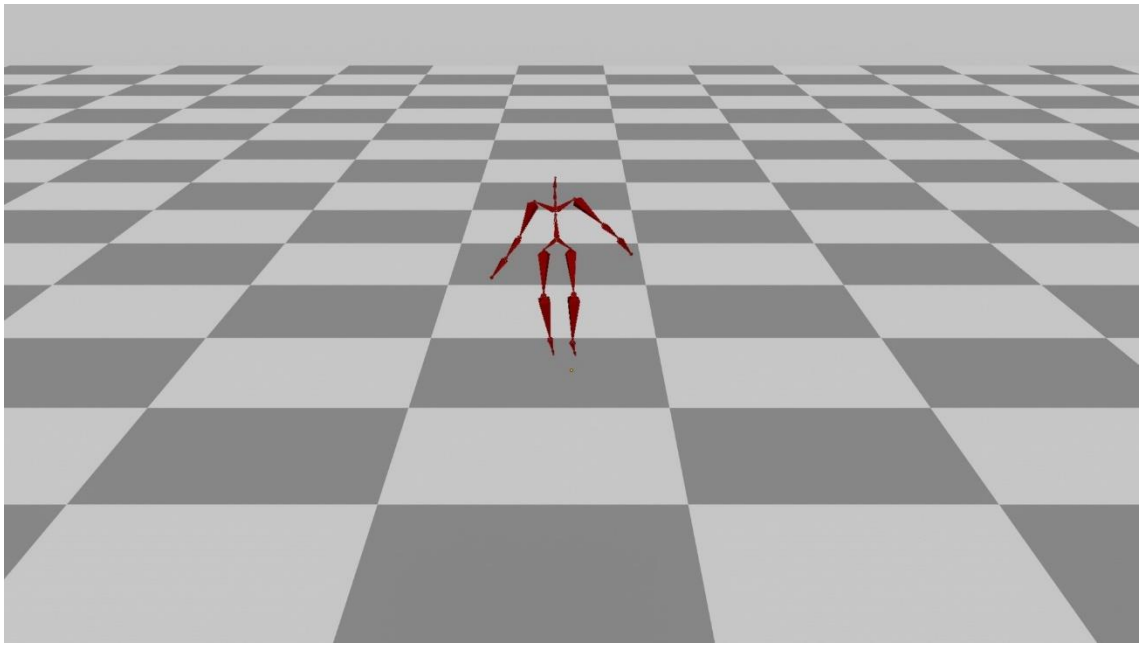

Figure 5.5 We generated a summarized motion-sequence of 600 frames by randomly removing frames. The resulted animation is not coherent. In this picture we have the frame 480 of our output, which is the frame on the 4/5 of our sequence. The distribution and localization of the sequence. In this case the captured frame looks similar to the input, but as is illustrated by our results as a whole this happened by random chance.

5.2.1.2 Discussion

As could be expected the output of this process was not coherent or smooth and a lot of the content was lost, because we couldn't control which frames would be removed and which frames would be included. As a result, the output animation was a series of different parts of the input motion sequence.

5.2.2 Seam Carving

5.2.2.1 Results

We then proceeded to try some "Smarter" solutions. The next method we tried was Seam Carving. To use models designed for images as input instead of motion, we proceeded to transform our BVH files to motion textures [10].

The idea is that we create an "image" consisting of the rotations of each joint in the three dimensions, as found in the data section of our BVH file. Each Dimension is represented by a channel of the image and then the image is imported to the model.

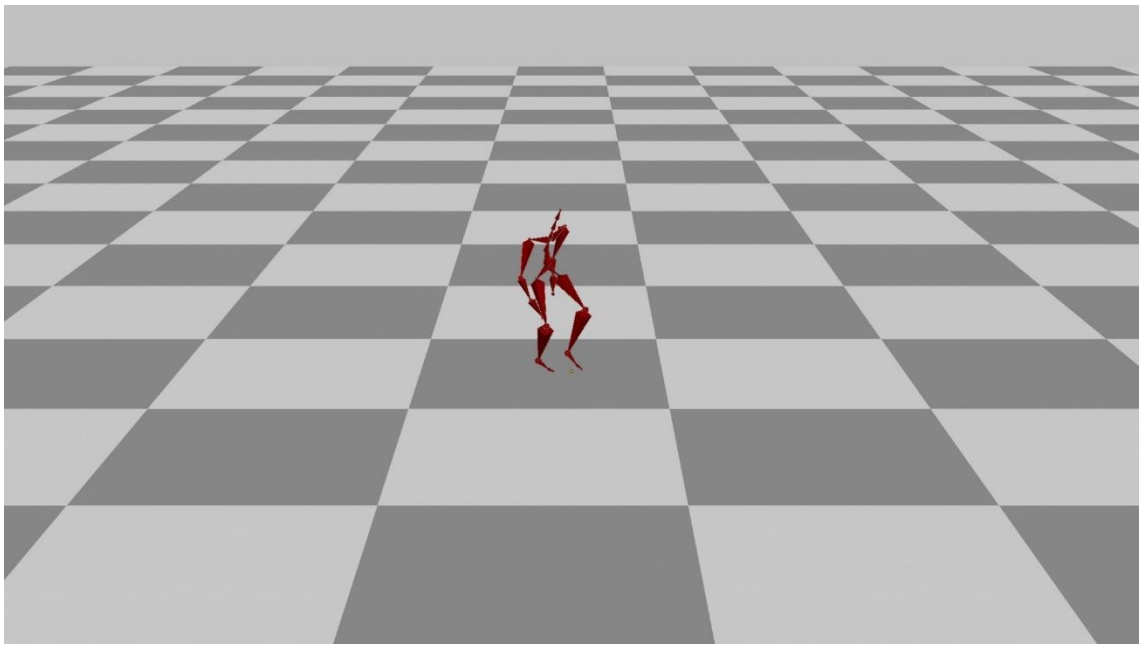

Figure 5.6 We generated a summarized motion-sequence of 600 frames by using seam carving as described above. The resulted animation is coherent. In this picture we have the frame 120 of our output, which is the frame on the 1/5 of our sequence. As we can see the frame capture is similar to the corresponding frame of the input, though not the same, showing us that at least the localization of the sequence is kept intact. This is supported by our whole sequence as well. On the other hand the speed of the sequence is not preserved and thus not all of our requirements are met. Despite that, we can use seam carving as a point of reference for our method.

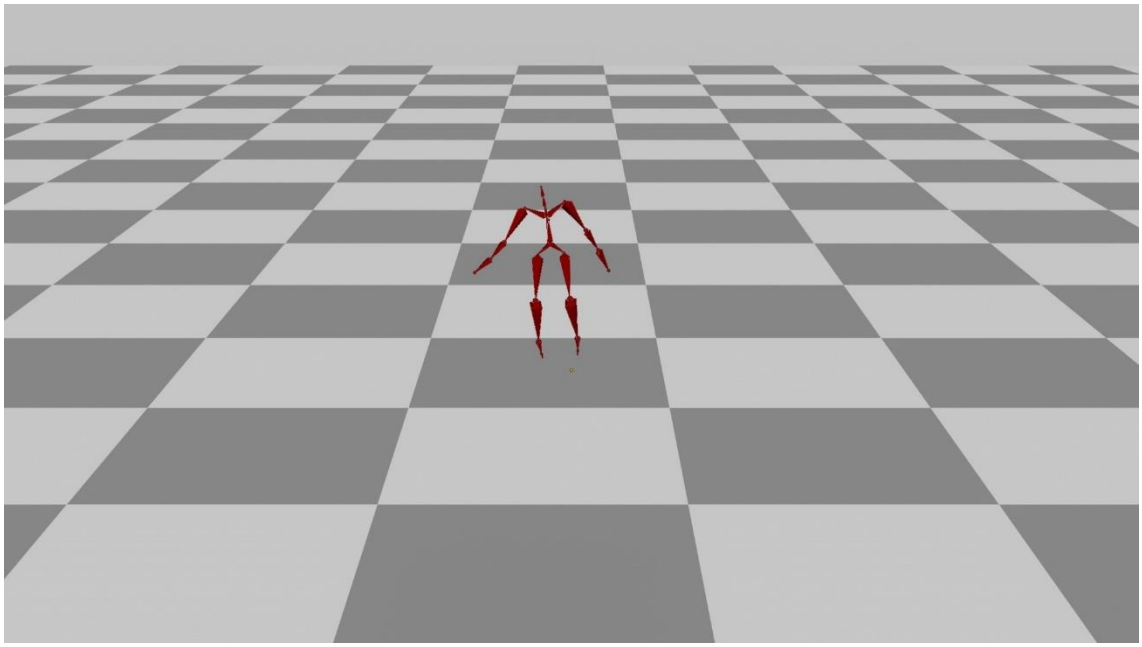

Figure 5.7 We generated a summarized motion-sequence of 600 frames by using seam carving as described above. The resulted animation is coherent. In this picture we have the frame 480 of our output, which is the frame on the 4/5 of our sequence. The captured frame mostly reinforces the conclusions presented in the previous figure.

5.2.2.2 Discussion

Whilst Seam Carving was able to make our motion shorter and mostly keep the output coherent and smooth, it had a completely different issue. Despite keeping the content of the input motion, it didn't maintain its speed. Our output motion was just a faster version of the input. That being the case, we were bound to continue by using GANs.

5.2.3 Original SinGAN

5.2.3.1 Results

As mentioned above with simpler solution having failed, we proceed to the use of GANs, with SinGAN as a foundation to work on.

Figure 5.8 The image used as an input to train SinGAN.

Before starting to work with modifying the network, we had to get familiar with the original network first. To achieve this, we proceeded to make a few experiments in order to understand better how it functions with different inputs. We started by generating random samples of the original size. Each time we generated results we got 50 random samples of the selected size. We can see the results generated by the original network below.

Figure 5.9 Example of a random sample generated by SinGAN in the same size as the input.

We then proceeded to create samples of different sizes to see how the summarization and enlargement functions in the original network.

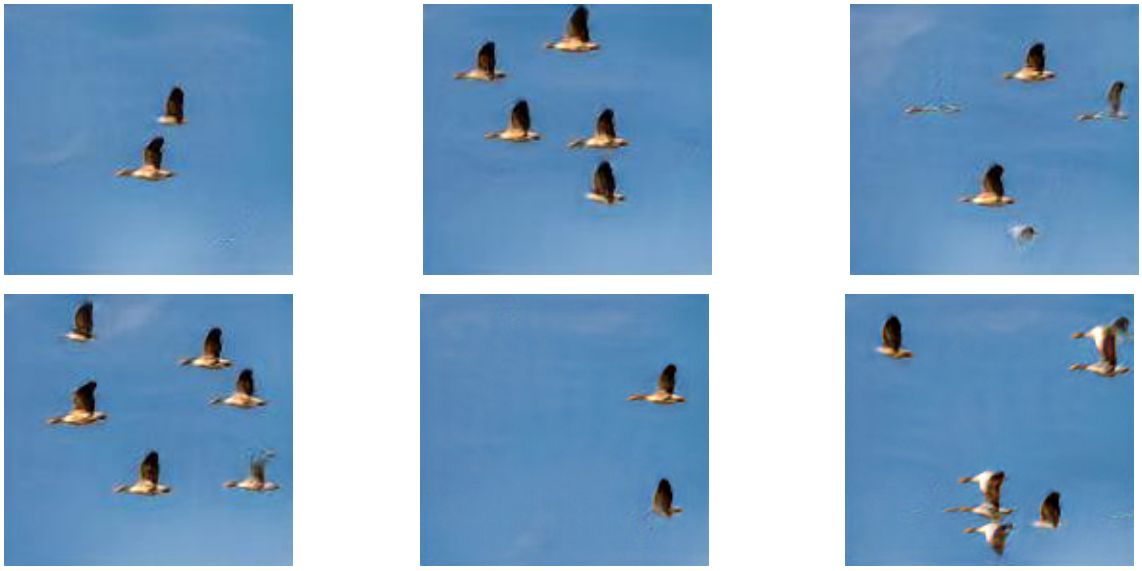

Figure 5.10 Example of different images summarizing the input shown before. The network can generate many different images from the same input, as a result of the random noise. As we can see some results are more successful than others.

After getting familiar with SinGAN we began experimenting with motion. Specifically, our next step was to use motion texture as input for the original SinGAN, in a similar manner to our Seam Carving tests. We also tried to modify the patch size to 32*32 to make sure that the patches given to our networks consist of all joints.

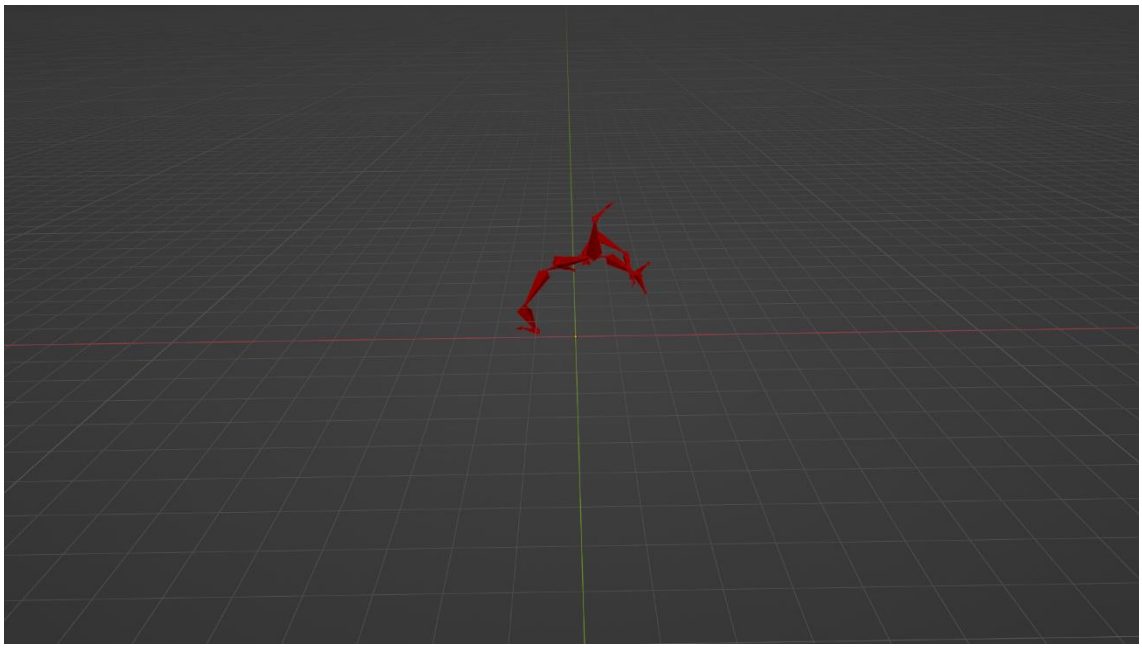

Figure 5.11 The generated result of this method resulted to a poor output. Even the general position of the joints is not preserved and as a result this method is completely unusable.

5.2.3.2 Discussion

As shown in the results above, the use of Euler Angles did not give us the expected results. The 3 values utilized to give information to our network are not enough for its training, while the absence of the appropriate losses lead to an output that is not coherent and does not represent the given content enough to consider the given result as successful. Therefore, we had to continue our efforts and modify the network to get the desired results.

5.2.4 Our Network

5.2.4.1 Results

We then proceed to see the output of our modified network with animations of different sizes. Our network is able to generate summarized version of the input animation, fake animations of the same size and enlarged animations based on our input. Looking at our results we can notice that our network is able to identify the different motion words of motion in our input file and use them to create new generated animations.

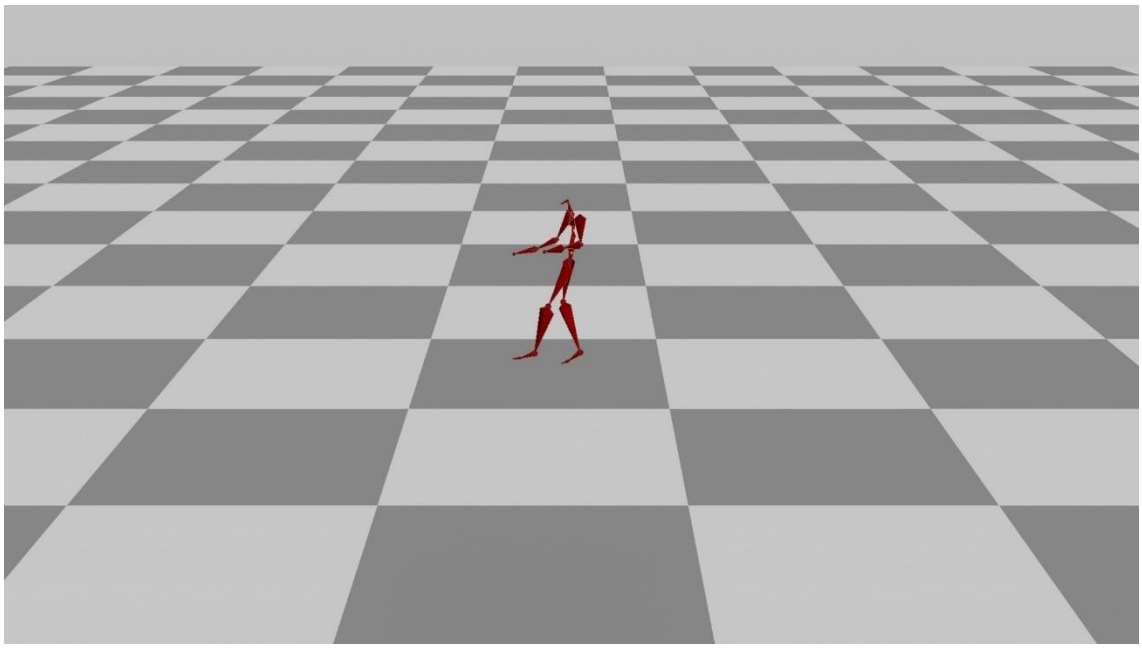

Figure 5.12 We generated a summarized motion-sequence of 600 frames by using our network. In this picture we have the frame 120 of our output, which is the frame on the 1/5 of our sequence. The resulted animation is coherent, and the speed is the same as that of the input. Despite that because random parts of the motion sequence are chosen not all the content of the input is present.

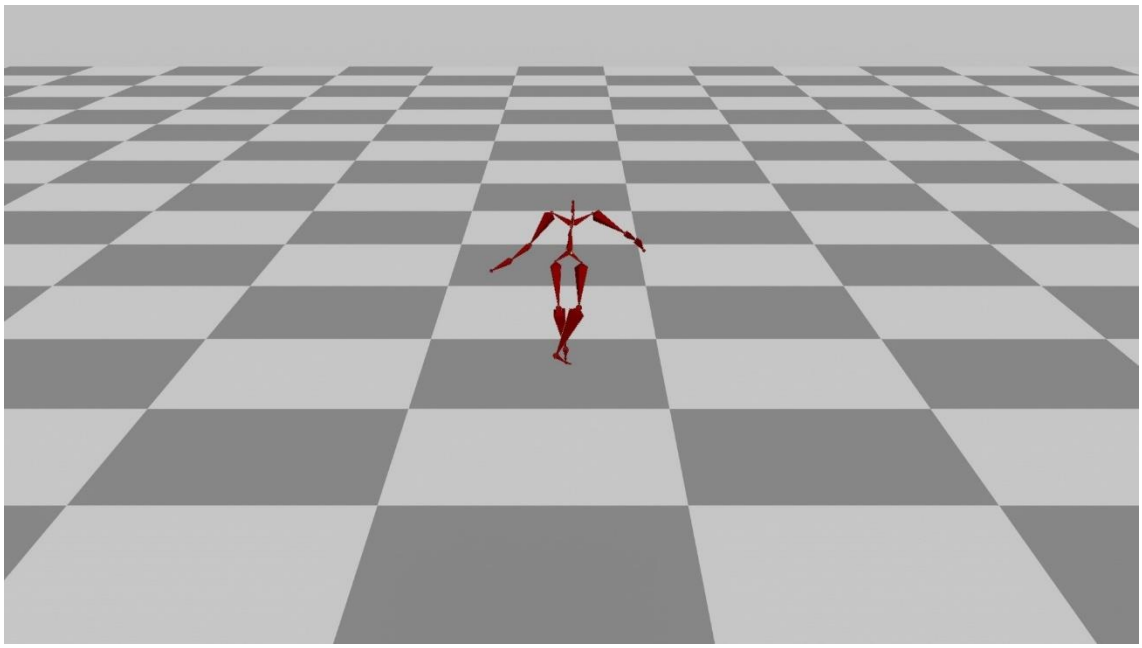

Figure 5.13 We generated a summarized motion-sequence of 600 frames by using our network. In this picture we have the frame 480 of our output, which is the frame on the 4/5 of our sequence. The resulted animation is coherent, and the speed is the same as that of the input. Despite that because random parts of the motion sequence are chosen not all the content of the input is present.

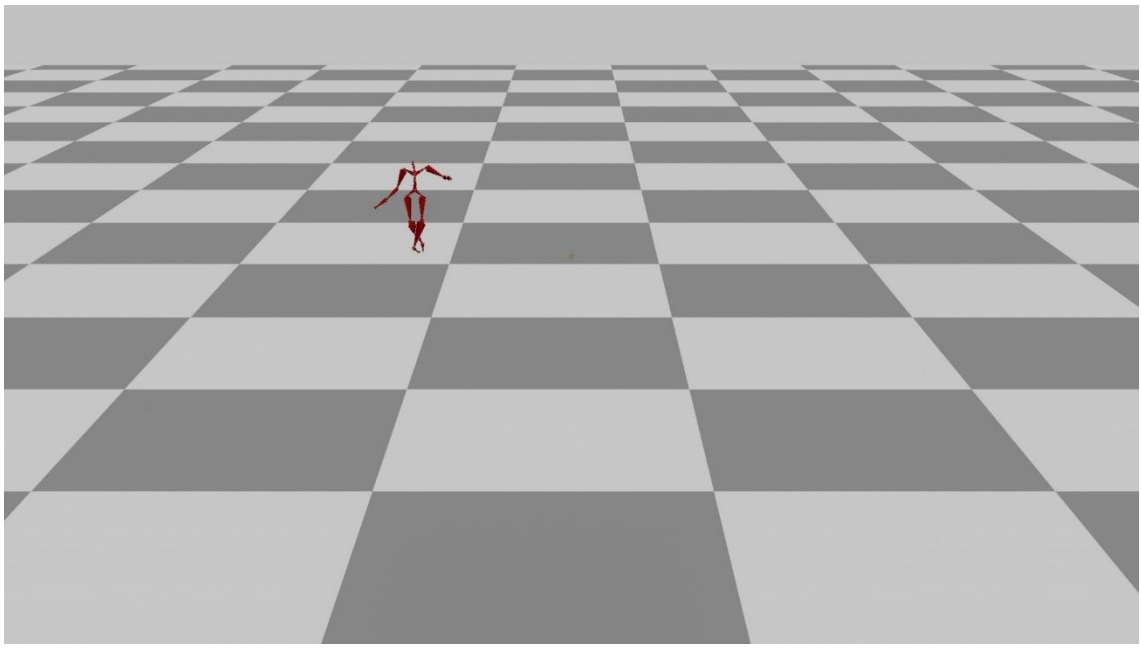

Figure 5.14 We generated an expanded motion-sequence of 2500 frames by using our network. In this picture we have the frame 500 of our output, which is the frame on the 1/5 of our sequence. The resulted animation is coherent, and the speed is the same as that of the input. As we can see the character starts to move to the left side more than the original.

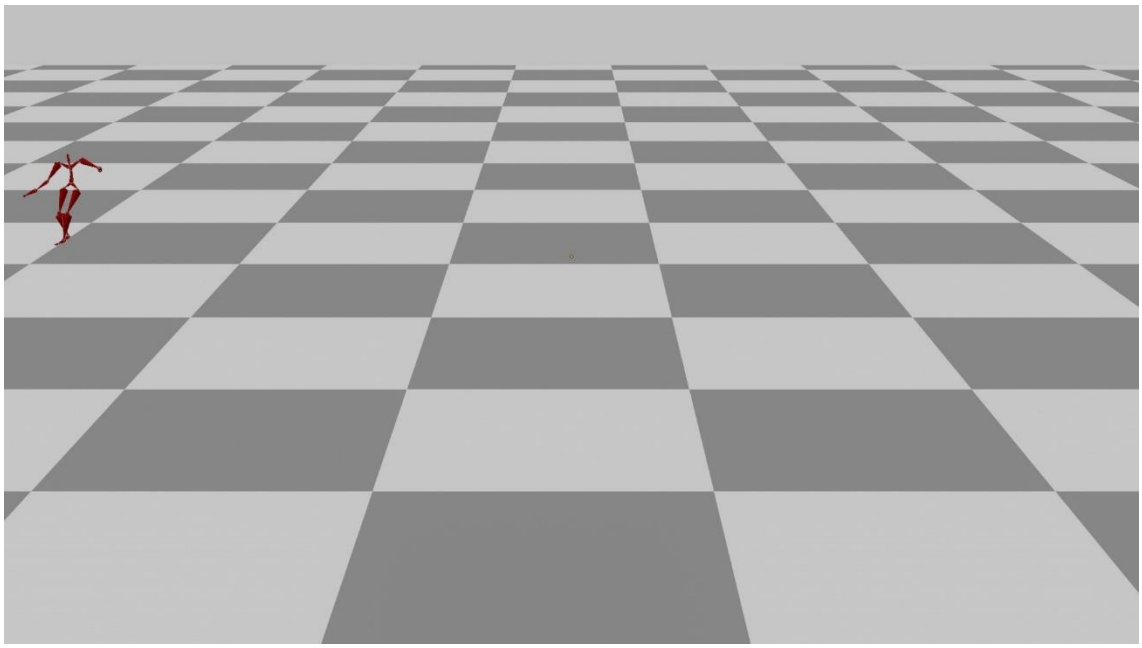

Figure 5.15 We generated an expanded motion-sequence of 2500 frames by using our network. In this picture we have the frame 2000 of our output, which is the frame on the 4/5 of our sequence. The resulted animation is coherent, and the speed is the same as that of the input. As we can see the character starts to move to the left side more than the original. This effect is more extreme in this figure than the previous one as more time has passed and more parts of the sequence where chosen that move the character to that side.

5.2.4.2 Discussion

It is important to note that the motion words chosen are random each time and for this reason we get a different each time we run our model for a certain animation depending on the random noise inserted to our Generator. That being the case, our current network cannot guarantee that it keeps all the content of the input motion in the generated fake animation. We can also notice that due to the randomness of the noise inserted to our Generator, the distribution of the original animation is not kept in the output. This creates a new issue for us. The skeleton in the output animation usually moves more to one direction in space than our original animation. This happens because our model randomly chooses to use more sequences of motion with the skeleton moving to a certain direction than moving to the opposite one, thus creating this effect. This is more apparent as we increase the size of the output. This happens because as we increase the size of our output and the model chooses more motion sequences in order to get to the targeted size, it is more likely to choose a disproportional number of sequences to each direction.

On the other hand, when we choose to summarize our input, by choosing an output size smaller than the original, we are more likely to have an animation where not all the content is present. It is also important to note that even when all the content is present, in various lengths, there is no guarantee that the content will be in the same order as the original one.

All in all, we successfully managed to create the first model able to create realistic looking animation of various size, whilst the output animation is still coherent and smooth. On account of this, the user is able to create diverse animations animation based on an input of the same or different size. This can find uses in situations where we need different characters performing similar, but different actions of a certain length.

Chapter 6

Conclusions

6.1. Conclusion

Whilst numerous studies have attempted to create model able to summarize or expand an image without the use of large dataset with decent success, motion summarization, using progressive training, remains a major challenge in the fields of Character Animation and Machine Learning. Through our study we manage to understand how some of the latest methods on image summarization function. In addition, we manage to learn about Generative Adversarial Networks and their major potential. During the course of this study, we also had the opportunity to learn about the different motion representations used in character animation with their advantages and disadvantages and work with different network losses in order to get the expected results. Of course, the most important breakthrough of this study is the fact that we prove that it is possible to create a network able to get enough information about an animation with the overlapping patches as input. By expanding the idea introduced in previous works, like SinGAN and InGAN, we managed to create an innovative network able to summarize and expand motion using GANs. Throughout the conduction of this dissertation, we proceeded to implement the mentioned network by using the code of the SinGAN project as foundation for our project. We had to do several modifications to the said network to transform it to a network able to work with character animation. First and foremost, we had to change the input of the network. The original network expects an image as output. Our network expects a BVH file, which in turn, transform this motion from the Euler angle representation to 6D representation. We also had to modify the patch size to equal the number of joints in order to give patches with whole frames to our network. Adding to the above, we had to apply

additional losses to our network. In order to keep the output motion coherent, we used a temporal coherence loss. We also used foot contact labels to avoid the presence of foot skating in our output animation. By implementing the above, we managed to create the first network able to summarize and expand an animation using GANs without the use of a dataset. This network is extremely useful for the character animation research and industry. As far as, the research side is concerned we were able to prove for the first time that it is possible to create a network able to learn enough information about a motion by just using patches and then in turn summarize and expand it. On the side of the industry this application can reduce the cost of motion capture and reduce the time needed to do motion capture. Specifically, it will give the opportunity to the industry to modify the duration of the motion captured in cases a different duration is needed to fit, said animation in a movie or video game, instead of needing to re-capture from the beginning which would increase the cost and time needed to complete the motion capture.

6.2 Limitations

Despite the success we had in creating our network, it still has several limitations when compared to the target of this study. Firstly, our results don't keep the distribution of the input animation. As mentioned above because our model relies on a random noise to start generating the fake animations, we cannot control that the output motion will keep the distribution of our input. As a result, we noticed that the fake animations produced by our network usually move more in space that the original animation, because our model randomly chooses more motion sequences to one direction, than the opposite one. The other major limitation of our architecture is that we cannot guarantee that all the content of the real animation exists in our generated ones. Similarly, to our first major limitation this comes because of the randomization, that comes with the random noise.

6.3 Future Work

As mentioned before, we manage to create an innovative network able to summarize an animation, without the use of a dataset. In order to expand our network, we aim use a

technique similar to the one used in InGAN in order to erase the limitations of our work. Specifically, we target to use the idea that if we are able to reconstruct our input motion using the fake generated by our network, we will be able to guarantee that the fake produced has all the content of the original. To achieve that we will revert our generator in order to create the original motion using the fake and then use an additional loss, specifically a reconstruction loss, to compare this motion to the input of our network. On the other hand, to combat the limitation of not keeping the distribution of the original image, we will have to make sure that the generated motion also keeps the different sequences of motion of the original in the right order. To combat this limitation, we will use another idea implemented in InGAN. To be exact we will limit the receptive field of each output frame. Basically, each output frame will be able to depend only on specific input frames. The last objective we would like to complete would be for our network to choose only specific frames to summarize and expand, whilst keeping the other intact in duration. The idea here is that we would like to keep the "special" sequences of motion intact and modify the more generic ones. A good example would be that if we have a basketball animation, shooting would be a "special" move as it can only be found in basketball, whilst running would be a generic motion, as it can be found in a number of different categories of motion. To achieve this, we could follow the general idea introduced in [11]. In a similar manner the authors used images to train a model to identify the city an image came from, we could train our model to identify the category our animation belongs to, then identify the motion words associated with the specific category and mark them as special and mark all the others as general.

Figure 6.1 Applications of the algorithm applied to different data sources. As shown the algorithm can successfully identify a room based on pictures of different elements. Also, it can be used to identify the decade different cars come from.

We will proceed to keep the motion words marked as special intact, whilst we would modify the duration of the motion words marked as general, to create an animation of the desired duration. We aim to expand our work as described in order to present this innovative network in one of the major upcoming conferences in character animation in the following months, such as the Motion, Interaction and Games 2022 or the EuroGraphics 2023 conference.

References

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

[2] 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.

[3] 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.

[4] Y. Zhou, Z. Zhu, X. Bai, D. Lischinski, D. Cohen-Or and H. Huang, "Non-stationary Texture Synthesis by Adversarial Expansion," in *ACM Transactions on Graphics 37(4) (Proc. SIGGRAPH 2018)*, 2018.

[5] 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.

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

[7] 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.

[8] Y. Dong, A. Aristidou, A. Shamir, M. Mahler and E. Jain, "Adult2Child: Motion Transfer using CycleGANs," in *ACM SIGRAPH Conference on Motion, Interaction and Games*, Charleston, South Carolina, USA, 2020.

[9] A. Aristidou, E. Stavrakis and Y. Chrysanthou, "Cypriot Intangible Cultural Heritage: Digitizing Folk Dances," *Cyprus Computer Society journal,* vol. 25, pp. 42-49, 2014.

[10] A. Aristidou, D. Cohen-Or, J. K. Hodgins and A. Shamir, "Self-similarity Analysis for Motion Capture Cleaning," *Computer Graphics Forum,* vol. 37, no. 2, pp. 297-309, 2018.

[11] C. Doersch, S. Singh, A. Gupta, J. Sivic and A. A. Efros, "What Makes Paris Look like Paris?," *ACM Transactions on Graphics (SIGGRAPH 2012)*, vol. 31, no. 3, 2012.