Human motion convolutional autoencoders using different rotation representations

Vladimir de la Cruz

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Computer Science (Computer Science) at

Concordia University

Montréal, Québec, Canada

January 2020

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Vladimir de la Cruz**

Entitled: **Human motion convolutional autoencoders using different rotation representations**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Sudhir Mudur

_____ Examiner
Dr. Thomas Fevens

_____ Supervisor
Dr. Tiberiu Popa

Approved by _____
Sudhir Mudur, Chair
Department of Computer Science and Software Engineering

_____ 2020

_____
Amir Asif, Dean
Faculty of Engineering and Computer Science

Abstract

Human motion convolutional autoencoders using different rotation representations

Vladimir de la Cruz

This research proposes the application of four different techniques of animation storage (Axis Angle, Quaternions, Rotation Matrices and Euler Angles), in order to determine the advantages and disadvantages of each method through the training and evaluation of autoencoders for reconstructing and denoising parsed data, when passing through a convolutional neural network.

The designed autoencoders provide a novel insight into the comparative performance of these animation representation methods in an analog architecture, making them measurable in the same conditions, and thus possible to evaluate with quantitative metrics such as Minimum Square Error (MSE), and Root Mean Square Error (RMSE), as well as qualitatively through close observation of the naturality, its real-time performance after being decoded in full output sequences.

My results show that the most accurate method for this purpose qualitatively is Quaternions, followed by Rotation Matrices, Euler Angles and finally with the least accurate results:e Axis Angles. These results persist in decoding and in simple encoding-decoding. Consistent denoising results were achieved in the representations, up until sequences with 25% of added gaussian noise.

Acknowledgments

My greatest appreciation goes to my parents for bringing me into this world, and all of my family for their constant support and love throughout all of my life's endeavors.

I am grateful to my supervisor, Dr. Tiberiu Popa, for initiating this research study as well as for his continued technical orientation during the completion of my thesis work. Special thanks to the crew over at Ubisoft La Forge for guiding me towards developing an interest in the complex and vibrant world of ML & DL, while helping me kickstart this research.

I also wish to acknowledge all my friends, especially Jean-François F Fortin, who have volunteered their help during the development of my thesis and provided technical and infrastructural support without which I would not have been able to complete my project with today's results and time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Kinematic human motion data is essential for different types of industries, ranging from videogames and film making to more crucial domains such as army simulations and health care diagnosis. As time goes on this data is becoming increasingly important and integrated into our everyday lives. Kitagawa and Windsor (2012)

The way we represent motion data, becomes relevant in a data driven world, where information is so important, computational advances and the adoption of Machine Learning becomes more and more widespread. By understanding something as rich and complex as human motion, and how to format it for machines to grasp it in a more efficient, or accurate way, we are able to address an interesting problem.

In this thesis I study the effectiveness of rotation representation for kinematics in neural networks, under non-euclidean rotational spaces, in which object motion is best represented. The data is parsed by a convolutional neural network, one of the more studied and effective technique of ML as of late, that provides a reconstruction using encoder-decoder architecture (autoencoder), in a relatively simple data structure. This architecture allows for the learning of a motion manifold, which could be useful for purposes such as denoising, retargeting, or style transfer in future developments. The rotation representation techniques covered in this research are: Axis Angles, Rotation Matrices, Euler Angles and Quaternions.

Neural Networks and Deep Learning use has been growing substantially throughout the years. The possibilities of its applications, adaptability and flexibility to support different types of problems

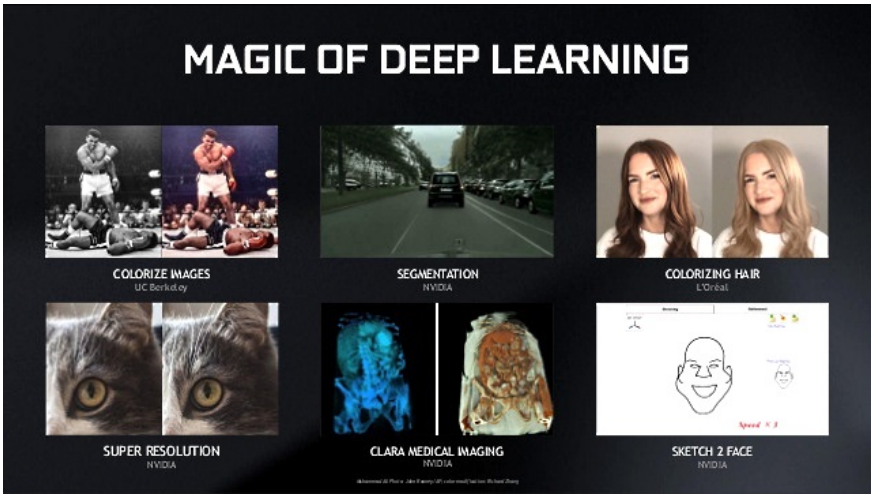

Figure 1.1: Advancements in deep learning presented by Jensen Huang from Nvidia (2018)

provides a versatility that has hardly been seen with other types of algorithms or techniques in the area of Machine Learning in recent years.

In comparison with other techniques such as Reinforcement Learning; Neural Networks and Deep Neural Networks, are a source of novel problem solving that was previously difficult to achieve or apply to problems less mechanical and more often left for the human ability to solve, such as pattern recognition, categorization, or prediction. The choice of Neural Networks as a problem solving algorithm becomes especially apparent when there are vast amounts of data available to train them. Consider this in relation to the possibility of defining a multi variable model, they provide excellent flexibility to develop applications that range a wide range of applications, from colorizing black and white images, to audio lip synching, to image generation or even antialiasing, the range of its applications have been keeping and steady growth. (Fig 1.2).

According to Schmidhuber (2015), a standard Neural Network (NN) consists of many simple, connected processors called neurons, each producing a sequence of real-valued activations. Input neurons get activated through sensors perceiving the environment; other neurons get activated through weighted connections from previously active neurons, thus forming through ordering; a process is known as 'layers'; Actual neuron observations suggest that they do not react promptly, but suppress the input until it has grown so large that it triggers an output; in NN's this kind of threshold is defined as an activation function Rashid (2016). Combining these concepts and computational

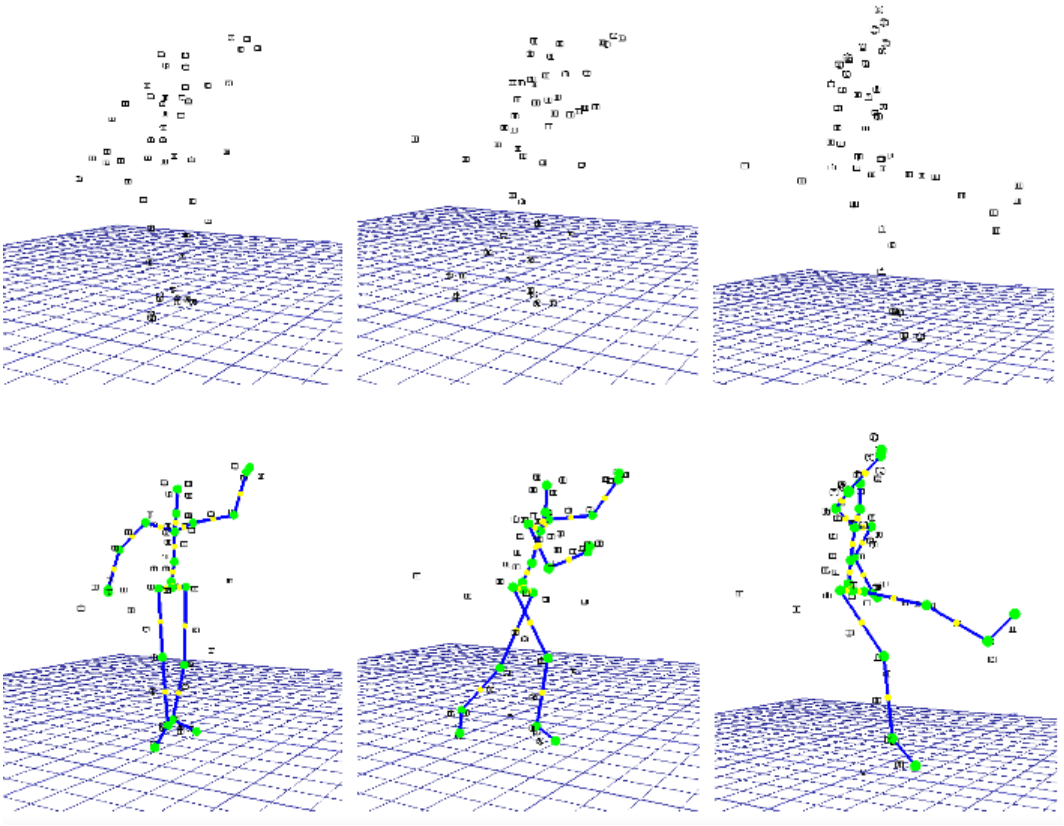

Figure 1.2: MOCAP example. Top row is the raw optical motion data based on each individual marker set (black square shows the marker position) captured from Reactor2 and Vicon system respectively. Bottom row is the constructed skeleton (green sphere displays joint, blue bar is the rigid body segment, and yellow dot shows the centre of mass of each rigid body) Z. Xiao et al. (2008)

structures is how the computing system of Artificial Neural Networks was established.

In the entertainment, medical, and sports industries, a popular technique called Motion Capture (MOCAP) has been growing in popularity in the last quarter of the 20th century. MOCAP is in use all around companies and studios due to its flexibility to record objects or people, reliable accuracy in the capture, and accessibility to the hardware equipment necessary for its implementation. The possibility of capturing through this technique implies that the data obtained may present errors coming from the nature of the original capture Z. Xiao et al. (2008). During the recording and after every shot, the raw markers data has to go through a process of cleaning, that can lead to errors in its execution, when doing manual or automatic processing, or when attempting to handle mismatched

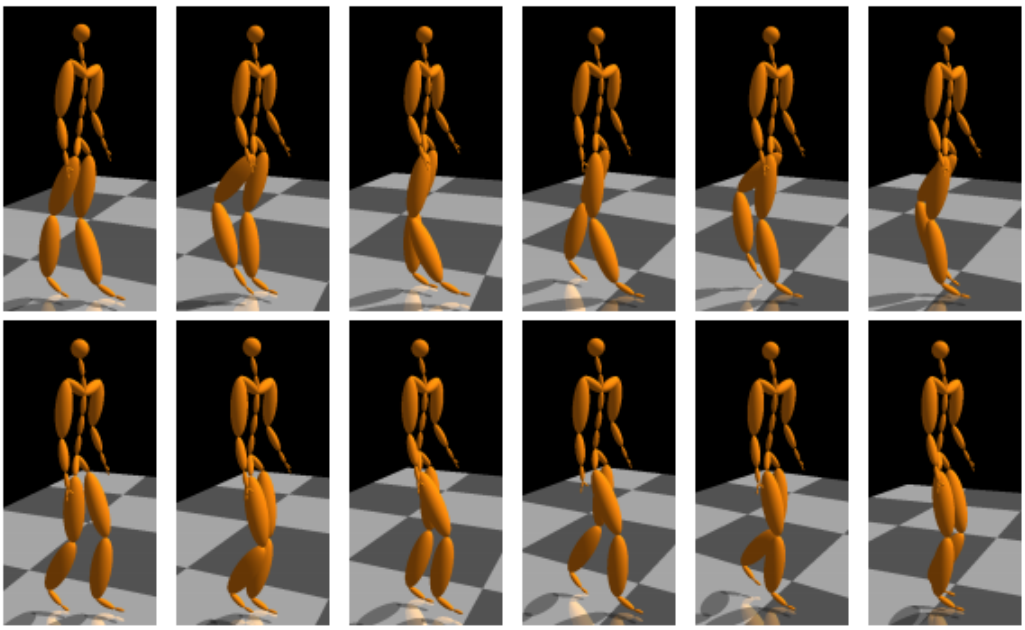

Figure 1.3: Example-based human motion denoising with corrupted knee joint, Lou and Chai (2010)

or occluded markers. This divergence can produce an essential difference in respect to the ground truth, which can be interpreted as 'noise'. In handling the reduction of noise, the 'denoising' process thus becomes an interesting problem to solve.

It is understood that the scope of defining a human motion manifold, and applying denoising while keeping this manifold as a frame of reference, is a challenging process because human motion involves a set of different, highly coordinated movement and these movements among different degrees of freedom are not independent from each other, Lou and Chai (2010). In this domain, it can then be understood that denoising human motion fix input motion data corrupted by outliers, and convert it to filtered motion data.

Holden (2014) worked on a project with a similar motion data definition, which specified that motion is typically represented as a time-series where each frame represents a pose of a character. Poses of a character are parametrized by the character joint angles, or joint positions. Holden also, he believes that this representation is excellent for data processing, and that valid human motion only exists in a small subspace of this representation.

In Butepage et al. (2017)'s paper, 'Deep Representation Learning for human motion prediction

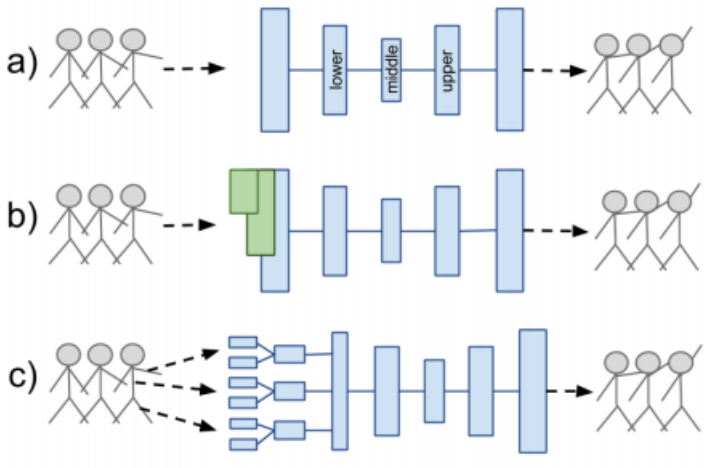

Figure 1.4: Experimental models for human motion prediction, Butepage et al. (2017)

and classification', it becomes clear that possibilities other than this technology can be of use. This study expands on the possibilities of learning motion manifolds and generative models to predict future 3D poses given a set of frames defined (Fig. 1.4).

As is common in the investigative field, during this research a series of challenges were presented across its development. One of the most crucial obstacles was exhibited during its initial phases of development, the detection of erroneous sequences, as available in the original datasets in the works of Holden et al. (2016). These noisy clips made it difficult for the comprehension of a human motion manifold in the designed models of neural networks, which made me reconsider their design.

Other challenges included the optimization of the model, testing across different designs and architectures (more of this can be observed in Chapter 4). The testing of different types of architectures and their results each required extensive hours to evaluate.

A comprehensive literature review on the topic of Neural Networks and Autoencoders is presented in Chapter 2. They are showcased first as introductory concepts and then as more advanced techniques with distinct applications. Following a more technical emphasis on applications and tools in Chapter 3, Chapter 4 goes on to define and explain the structure of the Neural Network, whereas Chapter 5 displays the quantitative and qualitative results of this research, along with the

indicators of comparison between all the methods. Finally, Chapter 6 unpacks the Conclusions and Future work in the field.

1.1 Contributions

This thesis presents a number of theoretical and practical contributions, including:

- The analysis of four different techniques of animation storage, Axis Angle, Quaternions, Rotation Matrices and Euler Angles, in the problem of encoding-decoding a temporal motion manifold and the denoising human motion animation sequences through a CNN.
- An objective measurement of training and validation performance for the animation representation formats
- The estimation of the robustness of trained motion manifolds generated by each model
- A reference for processing human motion in neural networks under relative angles' joint positioning
- A framework for training differently represented animation data, including validation and testing, with pre-trained models released and made publicly available at Vladimir de la Cruz (2019b)

Chapter 2

Literature Review

In this chapter, I will be referencing current knowledge and substantial findings in related literature and research, as well as theoretical and methodological contributions to the topics covered in this thesis.

2.1 Human Motion Data Denoising

Human motion denoising is the process of removing noise and outliers while keeping the intrinsic information, such as the structural information of a human body, and the spatial–temporal patterns embedded in the motion data. J. Xiao et al. (2015) specifies the existing human motion denoising methods can be classified into three categories: signal-based methods, data-driven methods, and low-rank matrix based methods.

Due the nature of our work and dataset training that uses several clips, our study would be considered data-driven under this classification. See figure [2.1] as an example.

2.2 Neural Networks

A Neural Network (NN) is a function mapping data, such as an image to an output vector. The function $g = f_L...f_1$ is the compendium of a sequence of simpler functions f_L, which are called computational blocks or layers. Let $x_1, x_2, ..., x_L$ be the outputs of each layer in the network, and let $x_0 = x$ denote the network input. Vedaldi and Lenc (2015)

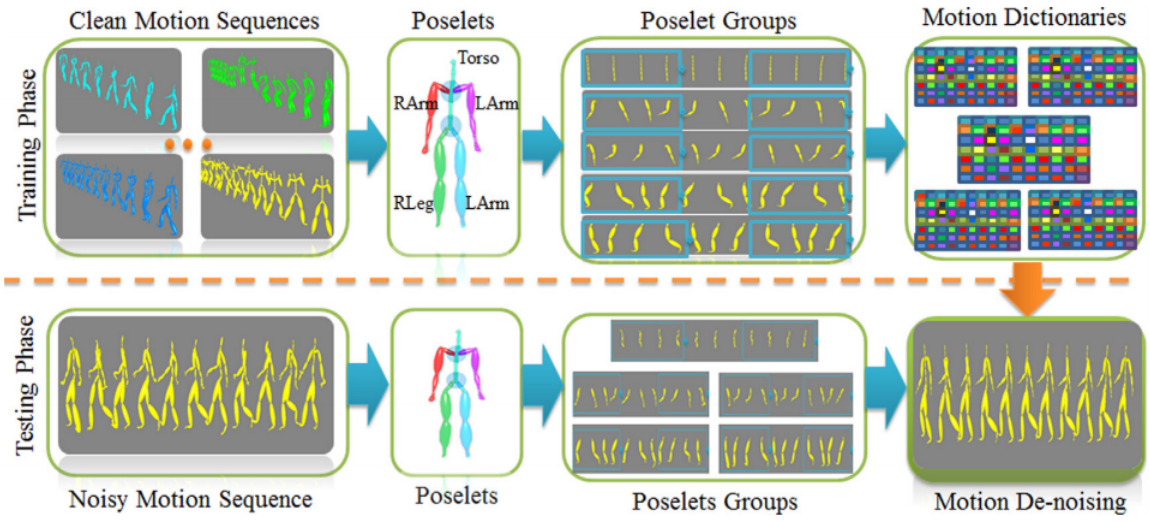

Figure 2.1: Human motion data denoising framework, J. Xiao et al. (2015)

Each intermediate output $x_l = f_l(x_l - 1; w_l)$ is computed from the previous output $x_l - 1$ by applying the function f_l with parameters w_l

As Rashid (2016) states NN's emerged from a drive for biologically inspired computers that attempt to emulate how human neurons works.

2.3 Convolutional Neural Networks

Specifically Ian Goodfellow and Courville (2016) mention that Convolutional Neural Network (CNN) are the more classic configuration for neural networks, that is specialized for processing data that has a known grid-like topology. Examples can be time-series, prediction series, and image data.

The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution, that consist of a linear operation over the nodes with a rectifier function on the output of each operation that works as an input to another neuron (see fig. 2.2).

While most CNN are obtained by composing simple linear and non-linear filtering operations such as convolution and rectification, their implementation is far from trivial. The reason is that CNN need to be learned from vast amounts of data, often millions of samples. It is also possible to conceive CNN with more than two spatial dimensions. In these cases, the additional dimensions, may represent volume or time. Vedaldi and Lenc (2015)

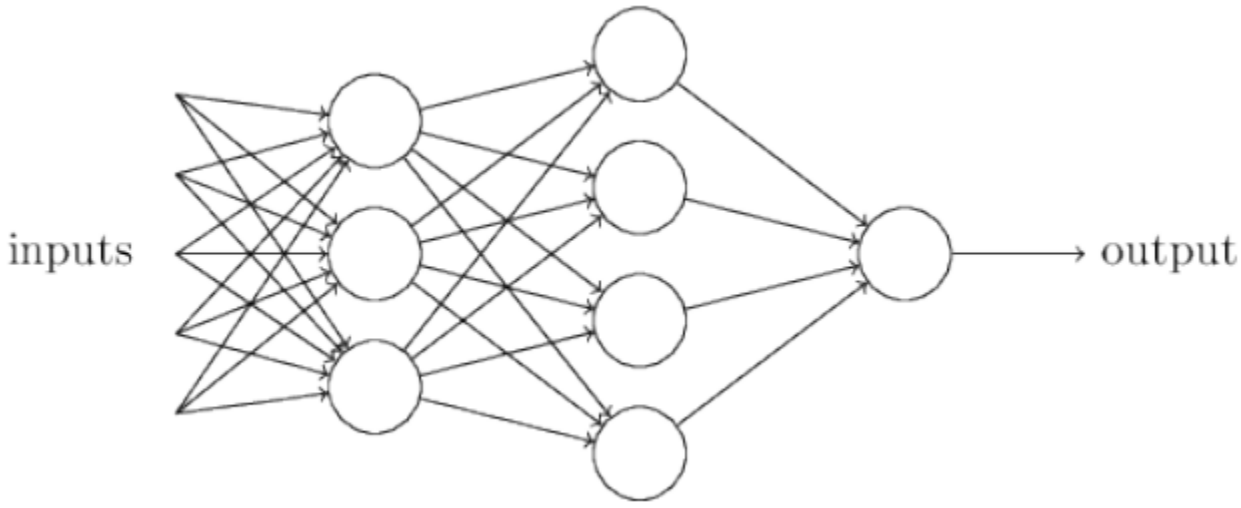

Figure 2.2: Example of a simple neural network, Nielsen (2015)

2.4 Autoencoders

One of the several applications of neural networks is the definition of an architecture defined as Autoencoders. Autoencoders are structures that comprise of at least an encoding layer, a number of hidden layers, and a decoder, where the hidden space is decoded as output.

These types of algorithms have several applications such as denoising, reconstruction, or definition of manifolds, depending on the input of the raw data provided, see [2.3]

2.5 Convolutional Autoencoders (CANN)

Convolutional Autoencoder Neural Networks are the implementation of a CNN in the internal layers of an autoencoder. According to Chen, Shi, Zhang, Wu, and Guizani (2017) a CANN is usually proposed for learning features from large amounts of data to avoid the uncertainty of hand-crafted features. It has the advantages of both unsupervised learning and unlabeled data learning, a CANN efficiently addresses the issue of insufficient training data caused during its difficult obtention and serves to demonstrate the intrinsic space in which the inputted data operates.

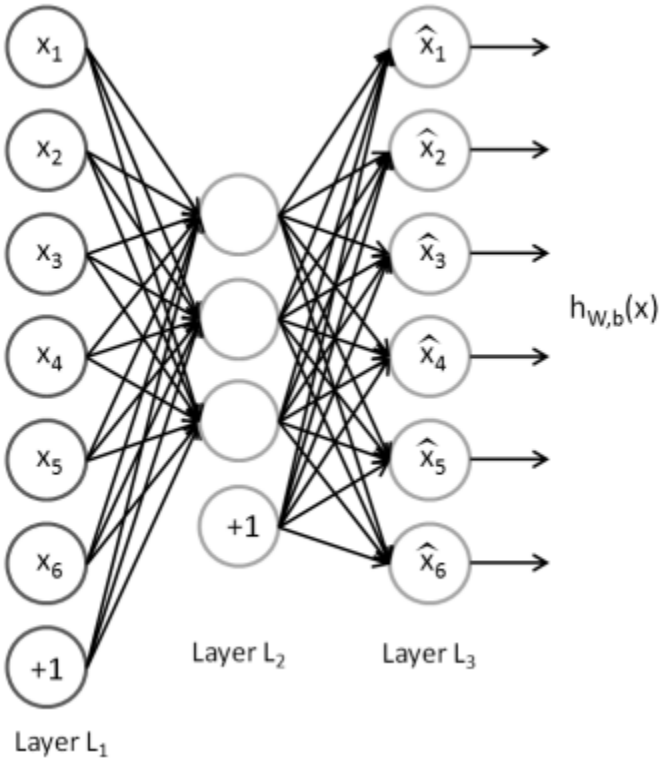

Figure 2.3: A basic autoencoder, Gondara (2016)

2.6 Denoising Autoencoders (DAE)

The machine learning concept of denoising has been used to achieve state of the art results on a large number of problems in computer graphics, such as the production of disentangled representations of data or data recovery Holden (2018).

Denoising, auto-encoders or DAE, are neural networks with at least an encoding and decoder layer. Their aim is to reconstruct the ideal data from a corrupted version of it Rifai, Vincent, Muller, Glorot, and Bengio (2011). More specifically, it receives corrupted input X before sending it through the autoencoder, which is trained to reconstruct the clean version (to denoise) Vincent, Larochelle, Lajoie, Bengio, and Manzagol (2010).

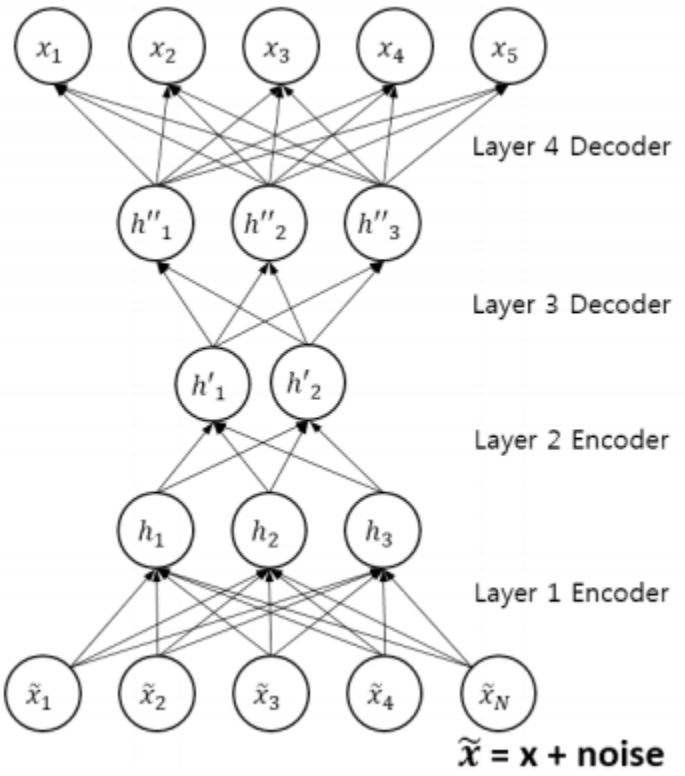

Figure 2.4: A stacked denoising autoencoder, Gondara (2016)

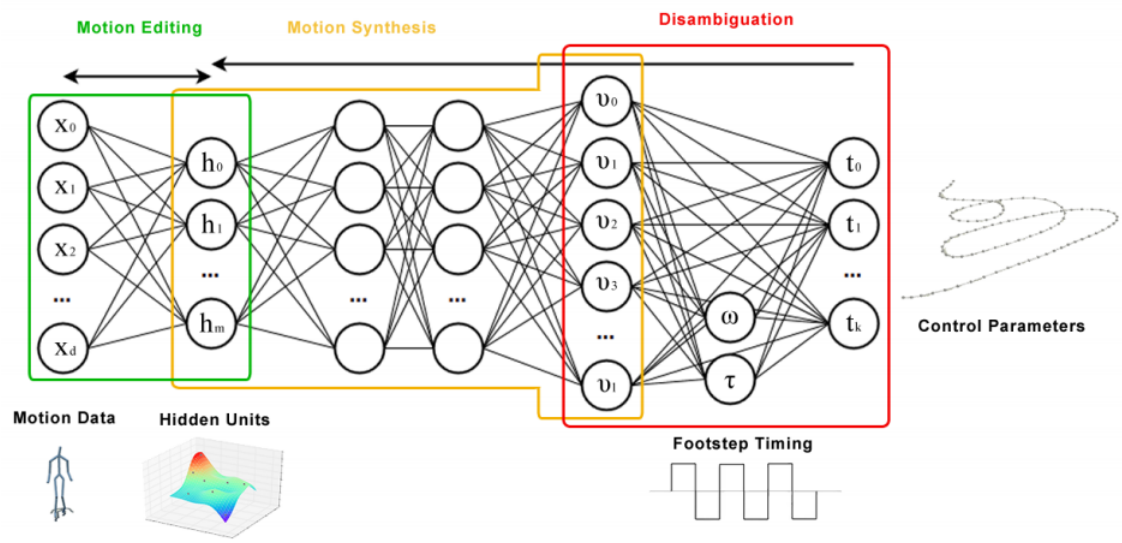

Figure 2.5: High level parameterizations are disambiguated and used as input to feed forward neural networks that produce motion in the space of the hidden units of a convolutional autoencoder, which can be further used to edit the generated motion, Holden et al. (2016)

2.7 State of the art Human Motion Studies

There are some studies and algorithms created regarding human motion, and more specifically, its denoising and encoding, that are relevant to my research. One of the most significant references is the work done by Holden, such as 'A Deep Learning Framework For Character Motion Synthesis and Editing' Holden et al. (2016), which uses a feedforward neural network stacked on top of the convolutional autoencoder to showcase the possibilities of learning this motion manifolds, such as very fast motion synthesis, natural motion editing, and style transfer (Fig 2.5).

In another one of his more recent works, Holden (2018) worked on 'Robust Solving of Optical Motion Capture Data by denoising', In his work, he presents a method for computing the locations of a character's joints from raw optical mocap data, which is extremely robust to errors in the input. His algorithm of choice used to solve this problem is a deep denoising feed-forward Neural Network, which is trained from a large database where skeletal motion capture data is first reconstructed, after corrupted, the inputs are passed through a noise function in an attempt to emulate real-life errors that can be presented in a typical setup.

In 'Modeling Human Motion with Quaternion-based Neural Networks', Pavllo, Feichtenhofer,

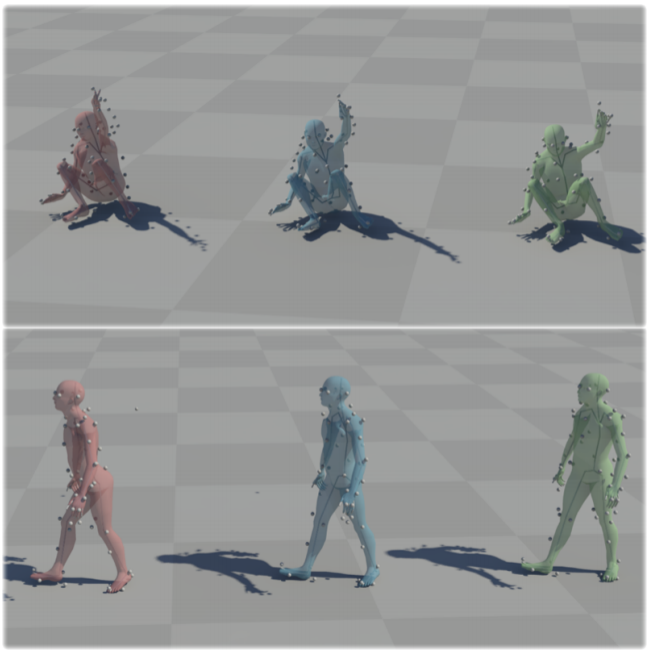

Figure 2.6: Comparison of results, Left: Raw uncleaned data. Middle: Holden (2018) method. Right: Hand cleaned data.

Auli, and Grangier (2019), elaborate on the research of predicting or generating 3D human poses sequences, while developing a method in a quaternion based NN (QuaterNet) that is more accurate than previous research when reducing the error along a kinematic chain. Their neural network represents rotations with quaternions and their loss function applies forward kinematics that penalize absolute position instead of angle errors. In their work, they compare Quaternions to Euler angles and exponential maps for prediction purposes, concluding that quaternions offer a more accurate prediction. On short-term predictions, QuaterNet improves the state-of-the-art quantitatively. For long-term generation, their approach is qualitatively judged as realistic as recent neural strategies.

Chapter 3

Tools and Technical Overview

The current chapter will expand on the tools used in this research, as well as the details in their implementation and structure.

3.1 Deep Learning Frameworks

Deep learning frameworks offer building blocks for designing, training, and validating Neural Networks and Deep Neural Networks, using a high-level programming interface. Among the most popular frameworks are Theano, Keras, Tensorflow and PyTorch which often rely on GPU-accelerated libraries such as cuDNN and NCCL to deliver high-performance multi-GPU accelerated training. Nvidia (2019)

3.1.1 Keras

Keras (2018) is a framework for easy and fast prototyping of Neural Networks that can run in CPU or GPUs, and is excellent for deep learning. It runs using Python and works as an abstraction layer for another backend such as Tensorflow and Theano.

Keras contains numerous functions for building blocks such as layers, objective functions, activation functions, optimizers, and a set of tools to facilitate the work with image and text.

3.1.2 Tensorflow

As defined in their Github repository, Tensorflow (2018) TensorFlow is ‘an open-source software library for numerical computation using data flow graphs. The graph nodes represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them. This flexible architecture enables the developer to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device without rewriting code. TensorFlow also includes TensorBoard, a data visualization toolkit.’

3.2 Animation Storing Formats and Tools

3.2.1 Acclaim Skeleton Format (ASF) and Acclaim Motion Capture file (AMC)

The Acclaim Skeleton Format is one of the most supported formats in 3D animation packages. It is comprised of two files, the .ASF that defines the skeleton joints and hierarchy, and the AMC file that contains the motion data.

ASF is separated by sections specified by a label and a colon, with descriptive sections such as :version, :name and :documentation, the values used in the measure, with the axis and order elements to describe the initial transformation of every joint or bone. The AMC files define the actual channel of animation and each frame joint or bone transformation. Lander (1998).

3.2.2 The Biovision Hierarchy (BVH)

The Biovision Hierarchy (BVH) character animation file format was developed by Biovision, to provide motion capture data to their customers. BVH seemed perfect for the task, as the format barely has any extra features, besides storing animation. The BVH file consists of two parts where the first section details a hierarchical data structure representing the bones of the skeleton [3.2, 3.4], and the following depicts a set of values or ”motion” section, defining the joint position values in euler angles, per every single frame in the animation. Razzaq, Wu, Zhou, Ali, and Iqbal (2015)

Due to the simplicity of its structure It is a widely accepted format that can be utilized in several 3D animation softwares that support it. This method of storing the animation served as the main

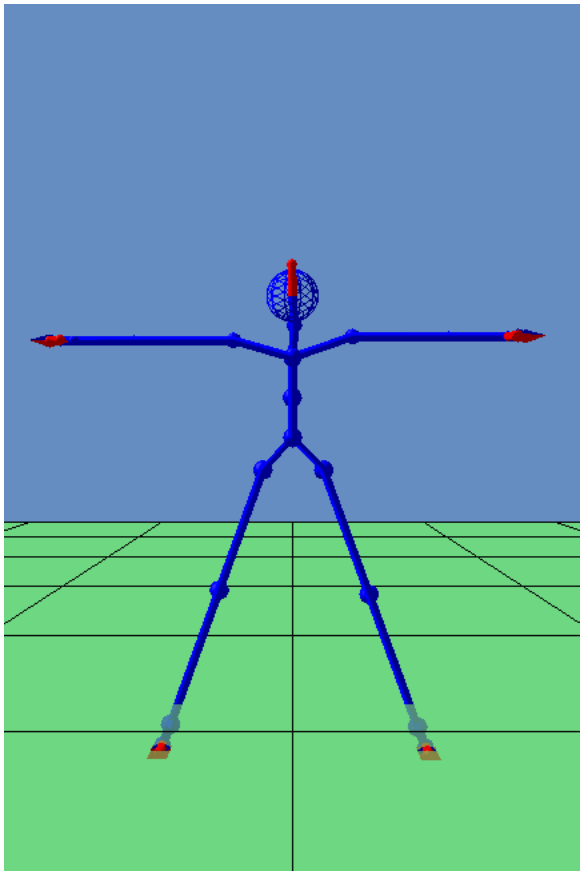

Figure 3.1: Retargeted skeleton with 21 joints in T pose

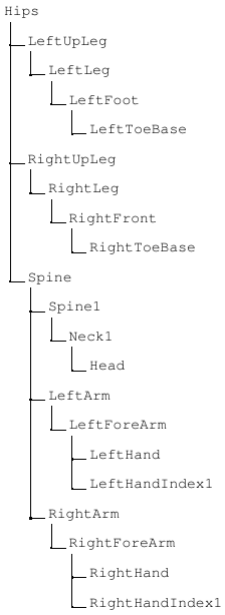

Figure 3.2: Named bones hierarchy of skeleton with 21 joints

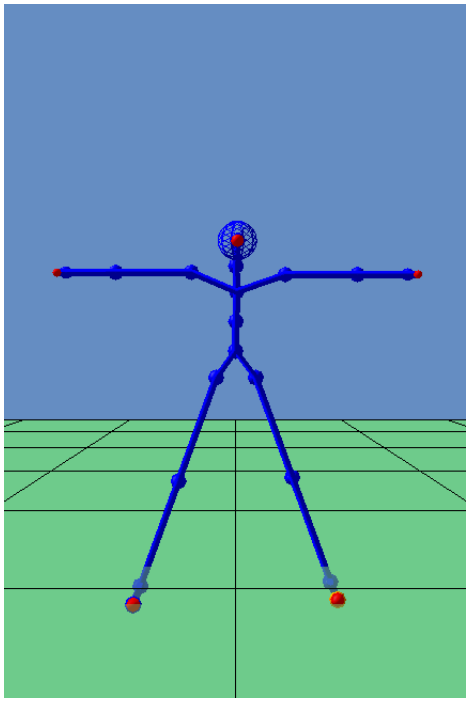

Figure 3.3: CMU skeleton with 30 joints in T pose

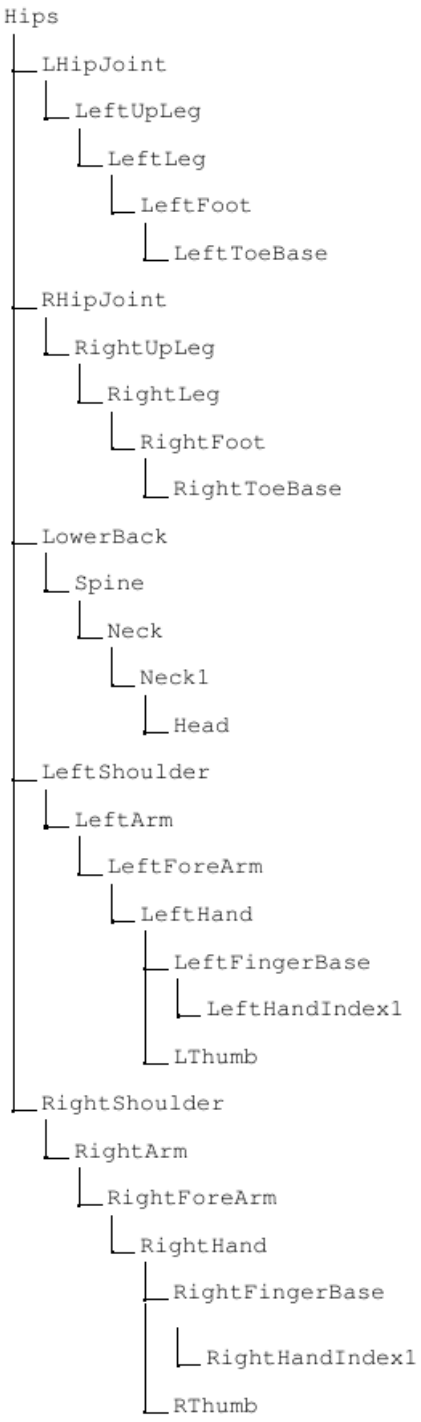

Figure 3.4: CMU named bones hierarchy of skeleton with 30 joints

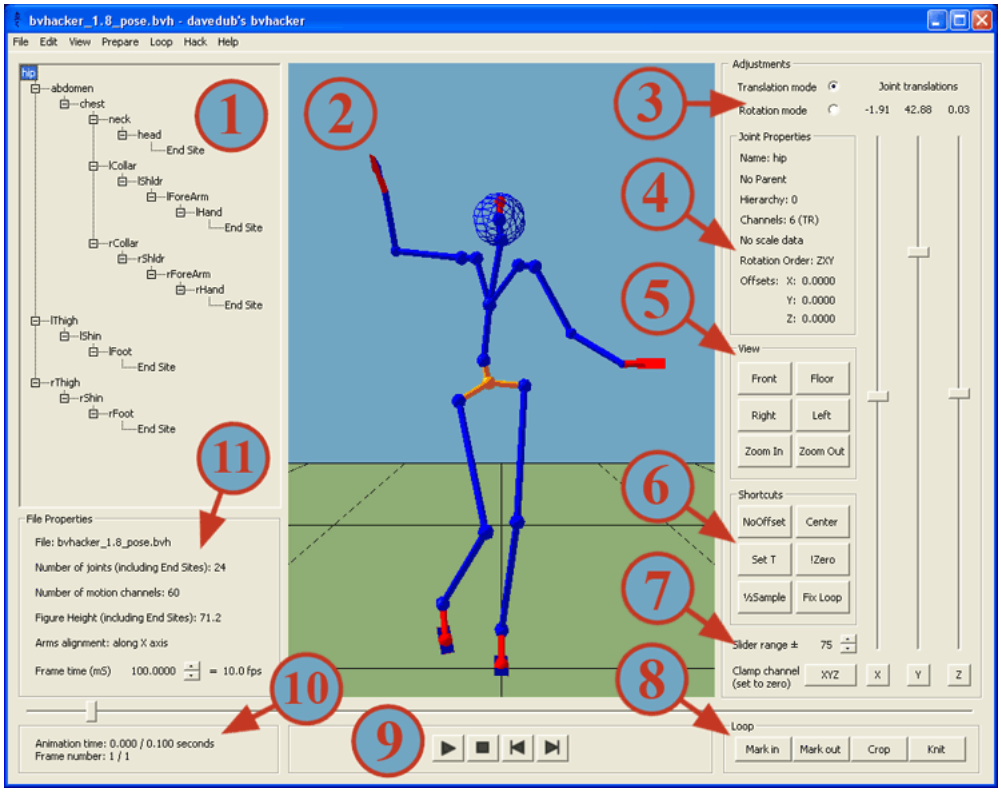

Figure 3.5: BVH Hacker Interface for version 1.8, Wooldridge (2019)

format across all 4 datasets due its simplicity, parallel to Holden, Komura, and Saito (2017), including their implementation of retargeting. The original base skeleton comprised of a hierarchy with 30 different joints [3.3], while the retargeting version is instead defined by 21 joints [3.1].

3.2.3 BVH Hacker

Bvhacker is a quick loading tool that is well-suited for the preparation and reproduction of BVH files Wooldridge (2019). It was initially developed for the conversion of files to be included in the videogame, *Second Life.*. Bvhacker is great for viewing, analyzing, converting, fault finding and preparing BVH files.

A customized version of this tool was used during the comparison of the results phase, as its source code is freely available for consultation online. An example of its user interface can be seen in fig. [3.5].

Chapter 4

Convolutional Autoencoder for Computer Animation

In order to achieve the best results, several iterations of the neural network architecture were considered. In the current chapter, this structure is defined, alongside the training and testing dataset sources to show how they were splitted and utilized for each method.

4.1 Rotation Representations

Rotations can be stored in different representation formats, each with different characteristics such as length of representation, format, or complexity of the data stored.

The represented rotations representations are as follows:

4.1.1 Axis Angle

Any finite rotation may be achieved by a single rotation around an appropriately chosen axis Diebel (2006). This representation system is compromised of four(4) values, three (3) of which represent the vector and another additional one representing the angle θ that defines the number of degrees to which the vector rotates.

4.1.2 Rotation Matrices

A rotation matrix is a matrix whose multiplication with a vector rotates the vector while preserving its length. The special orthogonal group of all 3x3 rotation matrices is denoted by SO(3) Diebel (2006). Thus, if R $\in$ SO(3), we can consider the following set of matrices, for a given euler angle (yaw, pitch, and roll), where each matrix represents a different rotation over a different axis.

$$\text{Rz}(\alpha) = \begin{pmatrix} cos\alpha & -sin\alpha & 0 \\ sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\text{Ry}(\beta) = \begin{pmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{pmatrix}$$

$$\text{Rx}(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\gamma & -sin\gamma \\ 0 & sin\gamma & cos\gamma \end{pmatrix}$$

Therefore the 3x3 combination of

$$R(\alpha, \beta, \gamma) = Rz(\alpha) * Ry(\beta)Rz(\gamma) \tag{1}$$

represents a rotation under a 3D space.

4.1.3 Euler Angles

The most popular way to represent the attitude of an object or rigidbody is using a set of three Euler angles. Euler angles are so popular due to their accessibility (to understand and use them) Diebel (2006). Using Euler's definition of any rotation or sequence of rotations of a rigidbody or coordinate system (ϕ, θ, ψ ; x, y, z ; roll, pitch, yaw) these angles describe a fixed point with a forward direction along the positive, body-fixed x-axis, with the body-fixed y-axis to starboard, and the body-fixed z-axis downward. [4.1]

The main disadvantage of Euler angles are the gimbal lock singularities, when two axes are

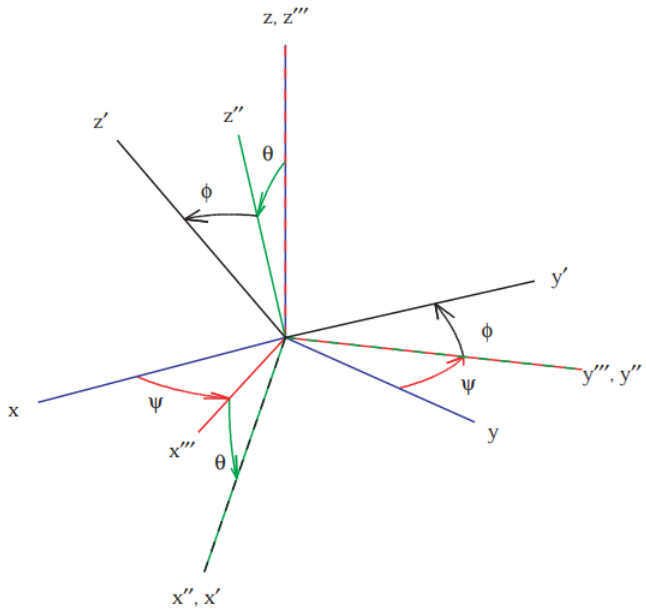

Figure 4.1: Euler Angle Sequence (1,2,3), Diebel (2006)

aligned. Applying a rotation over a third axis may yield the same transformation as if it was applied over one of the angles previously aligned.

4.1.4 Quaternions

Quaternions is a very stable number system that extends complex numbers. The fundamental algebra of quaternions indicate that:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{2}$$

A common way of defining quaternion orientation is in conjunction to Euler's Theorem, which states that the orientation of a rigid body can be described as a rotation about axis v by rotation angle θ, constraining the vector part to be unit magnitude [Cooke, Zyda, Pratt, and McGhee (1992), Goldstein and Poole (1980)]. They also have an advantage over Euler angles as they avoid the risk of gimbal lock. By definition, they are normalized, and can be represented computationally as follows:

Letting q be a unit quaternion, i.e. $|q| = 1$.

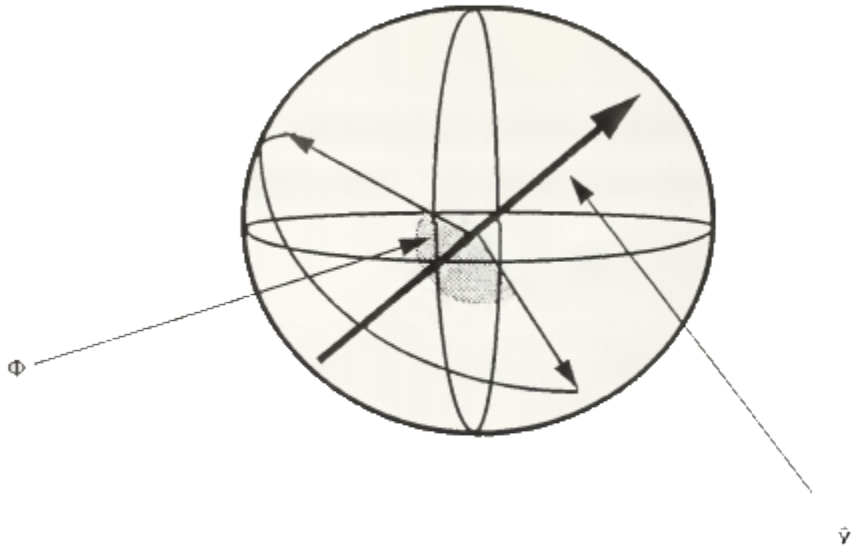

Figure 4.2: Representation of a Quaternion Orientation, Goldstein and Poole (1980)

It can be expressed

$$q = \cos\frac{\theta}{2}, v\sin\frac{\theta}{2} \tag{3}$$

$$q = w + z + y + x \tag{4}$$

$$w = cos(\theta/2) \tag{5}$$

Where w is the θ rotation angle around the axis of the quaternion.

4.2 Neural Network Architecture

This section describes some of the concepts and specific architectural models used in current research.

4.2.1 Shallow Network

The complexity of a Neural Network, in relation to the number of hidden layers, can help to discover or decompress a problem in different ways. A shallow network is therefore defined as a network with a single layer of hidden units.

Nowadays, deep neural networks have been established empirically and are becoming more frequently favored over shallow networks. Nonetheless, the theory of architectures of Neural Networks still poses many questions, and determining the number of layers depends on several factors, so each problem may pose a different complexity for selecting the appropiate type. Montufar, Pascanu, Cho, and Bengio (2014)

In the current thesis, the obtained results were trained under a shallow model to show the features that were lost when working with a deep model, as denoted by a much higher loss.

4.2.2 Network Models

Due to the variable, multi-dimensional nature of the training input, it was designed three-dimensionally with different architectures that consist of the same number and type of layers. Seeing as the dimensionality is the same, and the virtual data type is different in the case of Quaternions and Axis Angle (4 values), these two architectures can be modeled and referenced with the same diagram as in figure 4.5, or in a simplified 3D representation as shown in 4.8

During the design phase of the network models, and in order to avoid overfitting, two layers of dropout were added, with a rate of 15%, meaning 3 in 20 inputs were randomly excluded from each update cycle.

Meanwhile, the case of the convolution layer used a unidimensional (1D) convolutional layer with a kernel size, k = 25. As an activation function, it utilized ReLu, as optimizer it used a carefully parametrized Adam as defined in 4.2.2

Parameter	Value
Learning rate	0.0001
$\beta 1$	0.9
$\beta 2$	0.999
Epsilon (ϵ)	$1e^{-08}$
Decay	0.0

Table 4.1: Training values for Adam optimizer

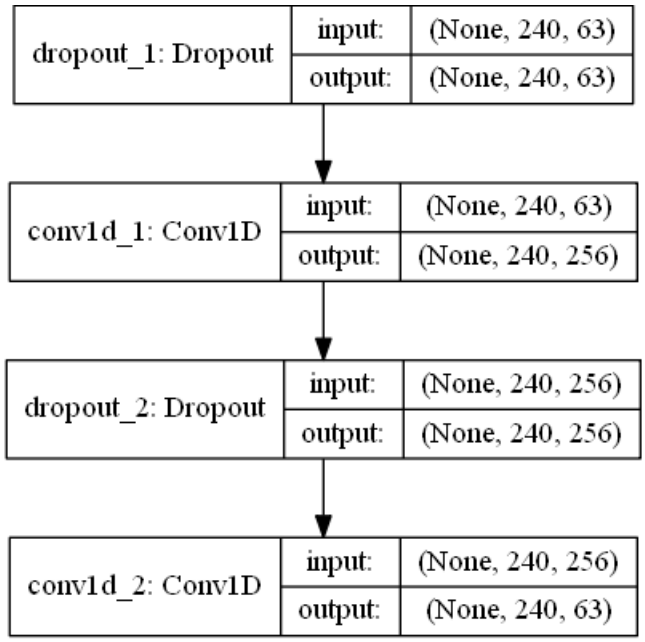

Figure 4.3: NN model for Euler

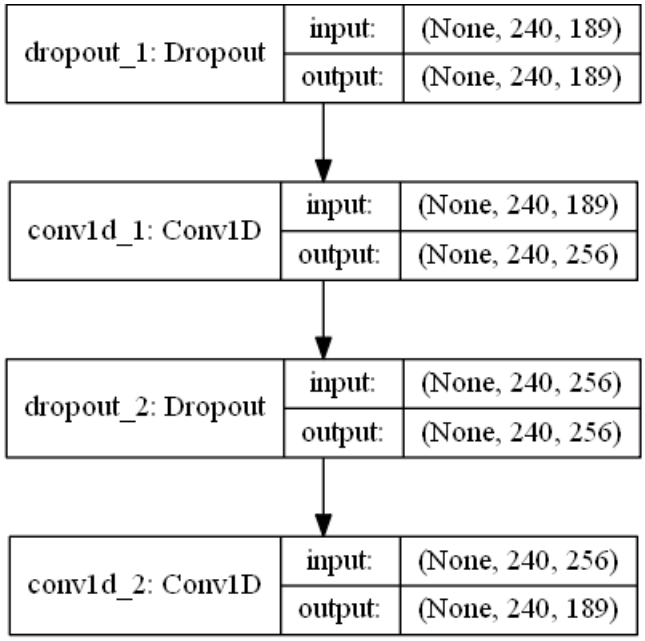

Figure 4.4: NN model for Rotation Matrix

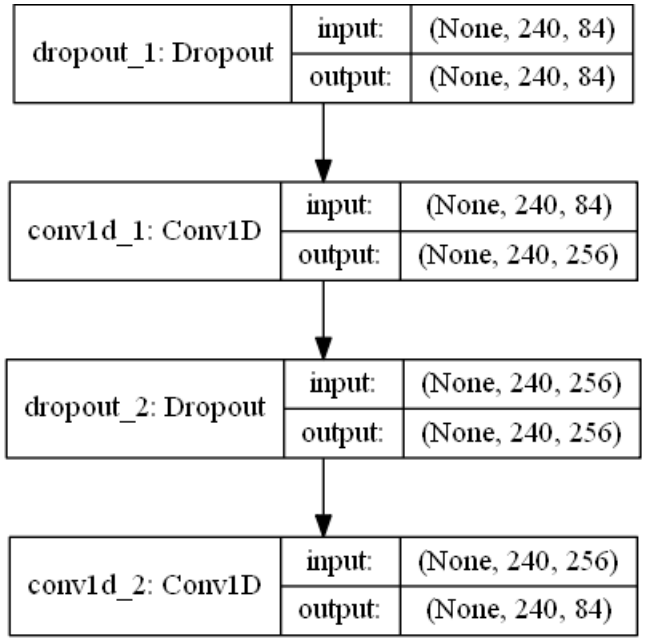

Figure 4.5: NN model for Quaternions and Axis Angle

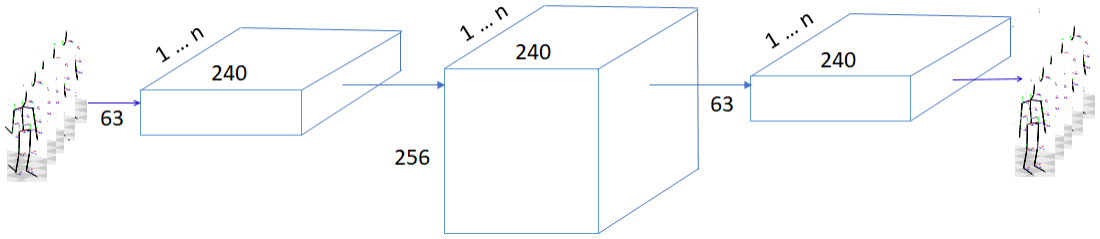

Figure 4.6: Simplified representation of architecture for Euler

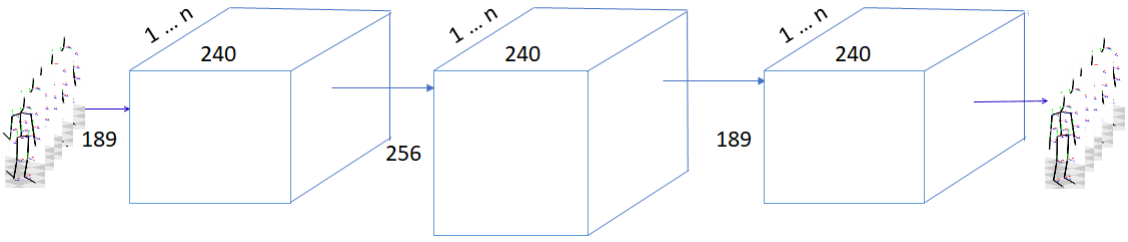

Figure 4.7: Simplified representation of architecture for Rotation Matrix

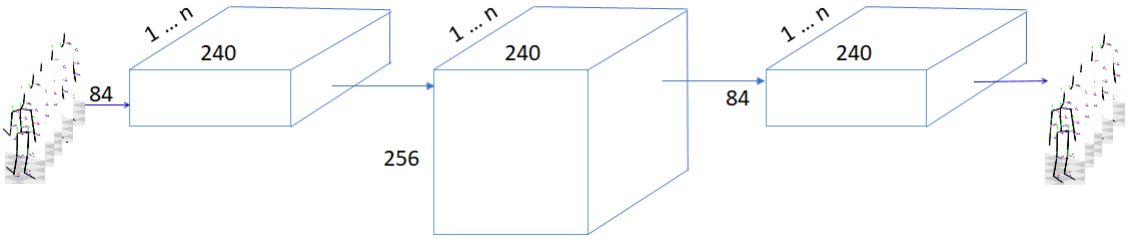

Figure 4.8: Simplified representation of architecture for Quaternions and Axis Angle

Chapter 5

Results and Experiments

In the present chapter, I describe several observations, methods, and analyses of the obtained results in the current research. The methods are split mainly between two categories, quantitative results and qualitative.

The quantitative metrics in this study are the Minimum Square Error (MSE) across each type of training, and the Root Mean Square Error (RMSE) for testing the denoising capabilities of the autoencoder, through specific animation files, in and out of the original training dataset.

The qualitative indicators are a result of the analysis of the animation fluidity, quality, and both the visual and physical consistency with an eye-estimated ground truth. In this chapter, I also elaborate on the generation of the datasets necessary for the training of the network, and expand on the intricacies and specific features of every single animation storage method covered.

5.1 Motion capture Databases

5.1.1 Carnegie Mellon University Motion Capture Database (CMU)

The Carnegie Mellon University Motion Capture Database (CMU) is a dataset of motions entirely free for all users. It consists of different subject movements and motion categories. There are 2605 trials in 6 categories and 23 subcategories. Carnegie Mellon University (2019). I am using a validated and retargeted version of the character with uniform joint lengths, retargeted in the work of Holden (2014).

The process of retargeting is first achieved by copying any joint angles in the original skeleton structure to the target skeleton, then scaling the source skeleton to the same size as the target skeleton, and finally performing IK to match the joint positions.

A subset of the dataset comprehended by 2434 data-clips from the original 2552 data clips is utilized in this research. The filtering of the dataset is a consequence of sampling and separating the correctly retargeted clips from the incorrect ones, meaning extremely noisy or incorrectly retargeted.

5.1.2 Edinburgh University Database

This collection comprises of existing databases and internal captures made at Edinburgh University, retargeted to a skeleton with common structure and joint lengths. This is decomposed by Holden (2014) in the following:

- **edinlocomotion**: This is a database containing long clips of locomotion data, including running, walking, jogging, and various sidestepping motions. It contains around 20 minutes of raw data and is not segmented into individual strides.

- **edinkinect**: This is a database containing a large variety of motions captured by standing in a small area using the kinect motion capture system. Because this was captured with the kinect, it contains many errors and artifacts, and so, should not be used as training data, but could be useful to researchers for other research purposes.

- **edinxsens**: This is a database containing the exact same motions as in the 'edinkinect' database, but they are instead captured through the use of an xsens inertia based motion capture system. Seeing as there is a frame-by-frame correspondence between the motion in this database and 'edinkinect' this database may be of interest to researchers trying to improve the output of the kinect.

- **edinmisc**: This is a small database of various miscellaneous captures made at the university, including some different walking styles.

- **edinpunching**: This is a small database of punching, kicking, and fighting motions segmented into many small sections.

- **edinterrain**: This is a database of walking and jumping on platforms of different heights.

For this research purpose, different clips were sampled from the edin database, such as in chapter 4, to showcase the results from decoding and denoising using this dataset during tests.

5.2 Joints Positioning and orientation

During the development of this study, I considered two different approaches: absolute positioning versus relative positioning. Positionings define how to use and locate joints in the space. Absolute positioning means transforming the values of the rotations under a location, where the joint positions are defined under the body local coordinate system, and where the root position is projected onto the ground, such as in the reference of Holden et al. (2016). Alternatively 'relative positioning' means that, instead of defining a global joint positioning, I would work with the raw rotational input. For this reason any joint positioning would be product of their space in the skeleton hierarchy, under its orientation, plus the previous rotations of its parent joints (taking into consideration that every joint would have its length, as defined in the input animation file).

In the initial prototype modeling following both approaches, it was clear that working directly with the relative joint angles provided more accurate results, even if they were not the most optimal. This scenario occurs, I suspect, due to the Neural Network's ability to interpret the data in a more uniform scope, whereas the distance between different samples would be closer. Nonetheless, this approach was still not as accurate for the reconstruction of inputs. To optimize this model and get results that approached the state of the art, I worked on simplifying this model by removing the length for each joint, and adjusting the global root joint positioning.

Simplifying the model, ultimately, provides more accurate results, as shown in the other sections of this chapter, such as in 5.6 and 5.7 for comparison. The global positioning removal from the model implies that the original global rotation defined and the original position, in each decoding (or denoising), is restored. This means that the inputs to the Neural Network keep a closer abstract relationship between each other, which I believe could imply a simplified manifold to learn. This simplification of the model would then augment the accuracy during training, validation, and testing.

When deciding on which variables to remove, to simplify the model, discarding the global

Method	Windows (n)	Frames	Joints	Values	Degrees of Freedom (d)	Represents
Axis Angle	35342	240	21	4	84	θ, z, y, x
Euler	35342	240	21	3	63	z, y, x
Quaternions	35342	240	21	4	84	w, x, y, z
Rotation Matrices	35342	240	21	9	189	3x3 Matrix

Table 5.1: Custom datasets per animation method used in training and validation

rotation was a choice of special consideration during the testing of the denoising capabilities 5.5. Discarding the positioning was the best option because this variable's inclusion during the training would otherwise provide a less accurate reconstruction.

5.3 Customized datasets

Due the different nature of storing the data, meaning the different quantity of values to store as well as field representation, four (4) distinct datasets had to be defined. This can be seen in Table 5.3. It is worth noting that the model works with a fixed window size during training to optimize and simplify the conceptual data structure. The fixed window size, n = 240 frame windows overlapped by n/2 frames, results in a final input vector defined as:

$$X \in \mathbb{R}^{n \times d} \tag{6}$$

where d is the degrees of freedom of the model. To the input values X, standardization is applied, which subtracts the mean of the whole training dataset and divides it over the standard deviation, while applying the inverse operation to decode. In our case, the degrees of freedom vary depending on the datatype, which keeps the number of joints consistent at 21.

The training and validation is based on clips from the Carnegie Mellon University (2019), and stored in the .npz file format, alongside the standard deviation σ and the mean $\overline{x}$ utilized during the normalization.

Method	Loss (MSE)	Val Loss (MSE)
Axis Angle	0.008776	0.014880
Euler	0.011501	0.018716
Quaternions	0.003029	0.005513
Rotation Matrices	0.006036	0.004004

Table 5.2: Obtained Loss (MSE) on training and testing sets

File	Noise	Axis Angle	Euler	Quaternions	Rotation Matrix
Punch Sequence	0	3.364	2.3599	3.1665	1.2904
Punch Sequence	0.027 (10°)	3.4226	2.5304	3.1585	1.3477
Punch Sequence	0.055 (20°)	4.4155	3.1185	3.314	1.5202
Punch Sequence	0.125 (45°)	12.9981	6.4097	5.2769	2.1485
Punch Sequence	0.25 (90°)	30.2013	13.3739	6.8463	3.8254
Kicking Sequence	0	6.3168	4.608	6.9974	5.7742
Kicking Sequence	0.027 (10°)	6.6096	4.9106	6.5948	6.1387
Kicking Sequence	0.055 (20°)	7.6796	6.5306	7.3961	6.5948
Kicking Sequence	0.125 (45°)	14.7836	11.2852	8.4578	7.8075
Kicking Sequence	0.25 (90°)	30.8443	21.4401	10.5774	10.27
Gorilla Run	0	5.0798	4.7953	5.0349	4.7114
Gorilla Run	0.027 (10°)	5.0694	5.323	5.1615	4.8808
Gorilla Run	0.055 (20°)	6.1374	6.5342	5.3273	5.1056
Gorilla Run	0.125 (45°)	14.563	11.0896	5.9515	5.7848
Gorilla Run	0.25 (90°)	29.0411	21.2824	7.7702	7.3231
Gorilla Run Asymmetric	0	5.0798	4.7953	5.0349	4.7114

Table 5.3: Obtained RMSE with noise per method over individual animations

5.4 Root Mean Square Error (RMSE)

The RMSE is used as a standard estimator to know how much, each storing method differs from the original, on average, after being parsed and decoded by the neural network,

The closer the result is to zero, the closer it is to the original representation. Comparing the results of the different methods of representation demonstrates that there is significant quantitative variation between them when changing from decoding to denoising.

5.5 Added Noise and Denoising

It was determined that noise should have been added to the original animations to evaluate the denoising capabilities, meaning that a uniform gaussian noise of 2.7%, 5.5%, 12.5%, and 25% (see Table 5.4), translating to $\pm 10°$, $\pm 20°$, $\pm 22.5°$, and $\pm 45°$ degrees, was added.

As a tabulated example, three sets of files were selected, a first one, 'Punch Sequence' from the CMU, 'Kicking Sequence' from the Edinburgh dataset, and finally 'Gorilla Run' also from the Edingburgh database. It is worth noticing that the added input noise was generated by adding gaussian noise in a normal distribution.

Finally, a modified version of the "Gorilla Run" animation it was also added to exemplify a modified skeletal version of the gorilla data in the animation. This served to demonstrate that any change in the length of each bone in our base skeleton, would keep the same values as another analog one, with different bone lengths.

5.6 Training and validation loss (MSE)

As a measure of the accuracy of the training and validation sets, the loss function indicated that the performance of these were designed as in the Minimum Square Error (MSE). A lost function or objective function is one of the parameters used to measure the training success.

Considering the results tabulated in Table 5.4, and the results represented in figures 5.3 and 5.4, there does not seem to be notable differences regarding the loss and validation loss of both methods. It is interesting to notice, however, that in the case of Quaternions, the validation loss is superior to

the training loss, whereas the opposite is true in the case of Rotation Matrices.

In light of these findings, it is reasonable to conclude that the quaternions training tends to be more fitted towards the training set, as well as adapting and smoothing more closely towards temporal motion manifold, while the rotation matrices tend to adapt better to cases farther than the ones in the training set, which preserves a more generalist inference or approximation. However if the goal is to tailor and correct a learned motion towards a learned manifold, this is a disadvantage. It is important to notice this distinction, as it can mean that the more data is made available for training, the general case of quaternions should be qualitatively more pleasant and natural, and quantitatively more accurate on average.

5.7 Training and validation

For the training of the Neural Network, a dataset of n times n, with a window size of 240 frames was utilized, as further explained in section 5.3

The dataset was split between two sets, a training set with 80% of the original data and a validation set with the 20% of remaining data. This is a standard practice in Neural Network's performance evaluation that helps validate the results in the reconstruction are satisfactory given that the encoded-decoded results are quite similar to each other.

In order to evaluate the reconstruction of the encoding-decoding of the input sequences, the different methods defined in this research - Axis Angle, Rotation Matrices, Quaternions, and Euler Angles - were evaluated.

To see the obtained results, refer to table 5.4. It is interesting to notice that Quaternions and Rotation Matrices seem to be the closer methods in relation to the ground truth considering the error obtained; nonetheless, even if such values were consistent when decoding the validation set without alterations (in the case of added noise), or when using the neural network as denoising architecture, the performance would not differ significantly.

When testing the denoising of values on individual files, for example, the ones listed in 5.4 , are consistent with my initial hypothesis that the most stable method is Quaternions and Rotation Matrices, followed closely by Euler Angles passed the 5.5% of added noise, and diverging dramatically

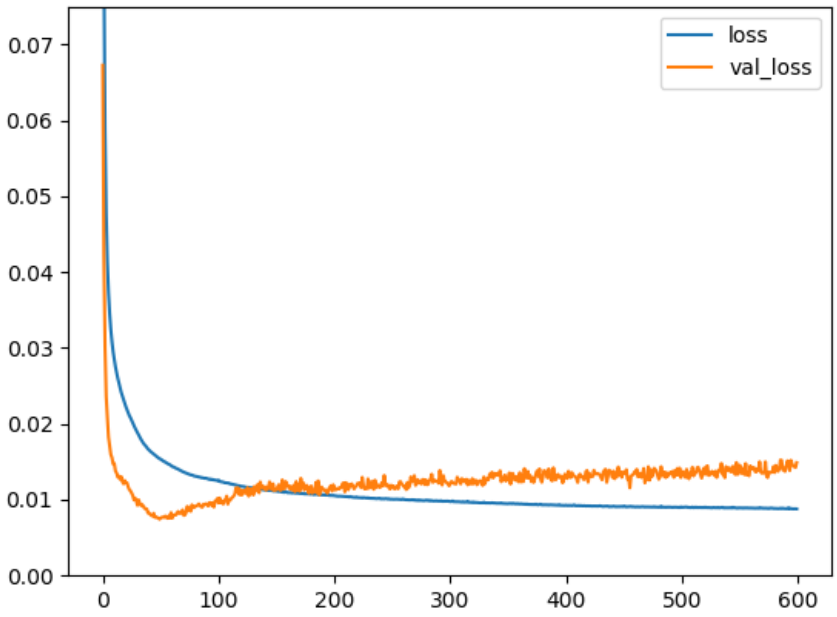

Figure 5.1: Axis Angle training vs. testing loss

passed the 12.5% of added noise.

It is worth observing that the tolerance of error for the most notable methods, Rotation Matrices and Quaternions, can go up to 25% of added noise. Even if the features in the space are complex to discern by human eye perception, the abstract relationship between the values over a temporal window is possible to be inferred and translated into the temporal motion manifold with this Neural Network. It can therefore generate a representation that is not far from the original input, denoised, and quantitatively and qualitatively accurate.

The video recording with comparisons can be viewed online via: Vladimir de la Cruz (2019a).

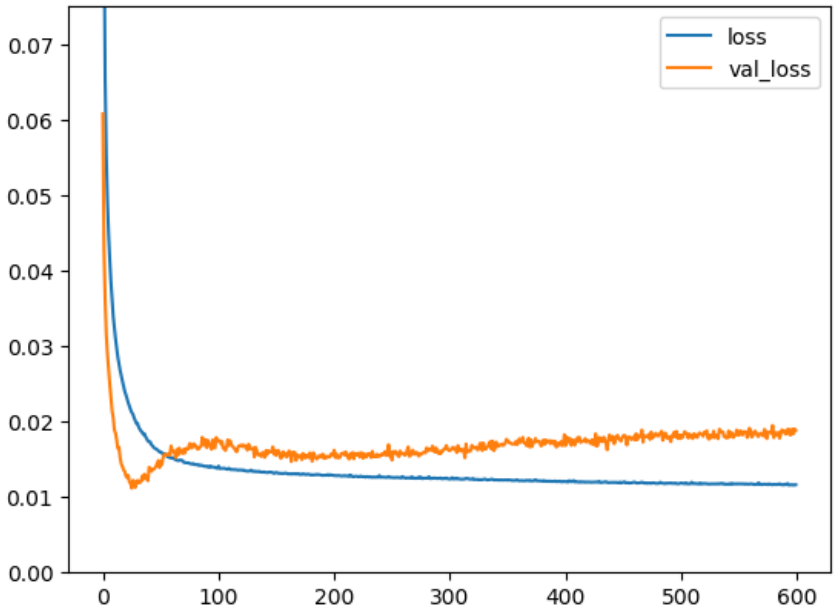

Figure 5.2: Euler angles training vs. testing loss

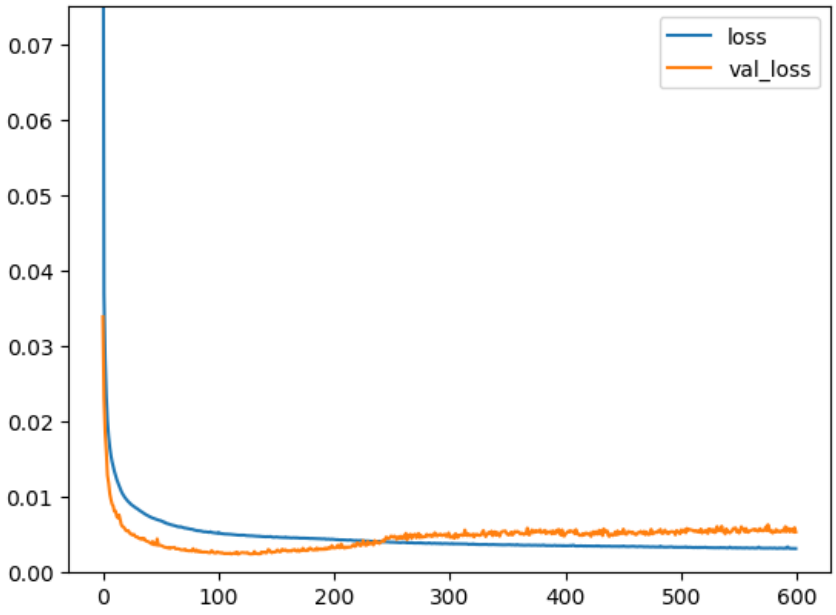

Figure 5.3: Quaternions training vs. testing loss

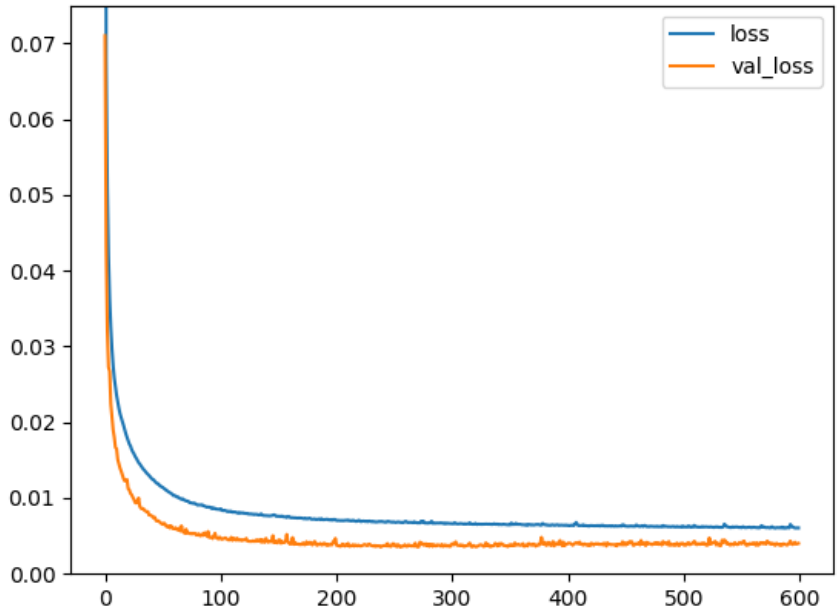

Figure 5.4: Rotation Matrices training vs. testing loss

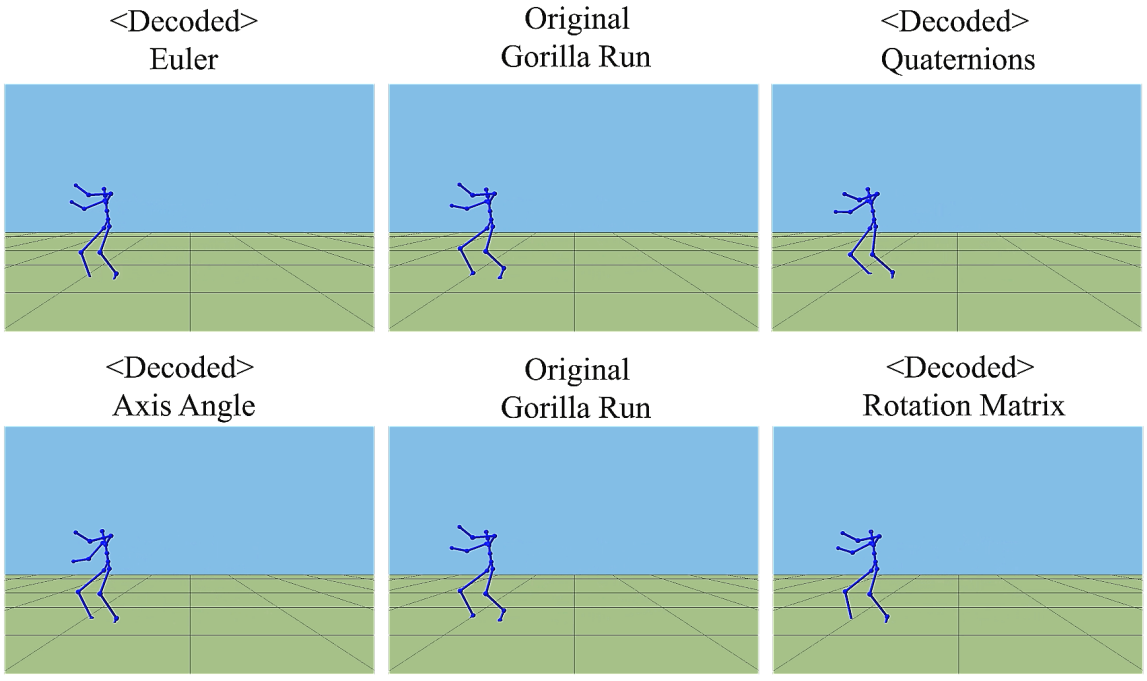

Figure 5.5: Decoded gorilla run comparison frame

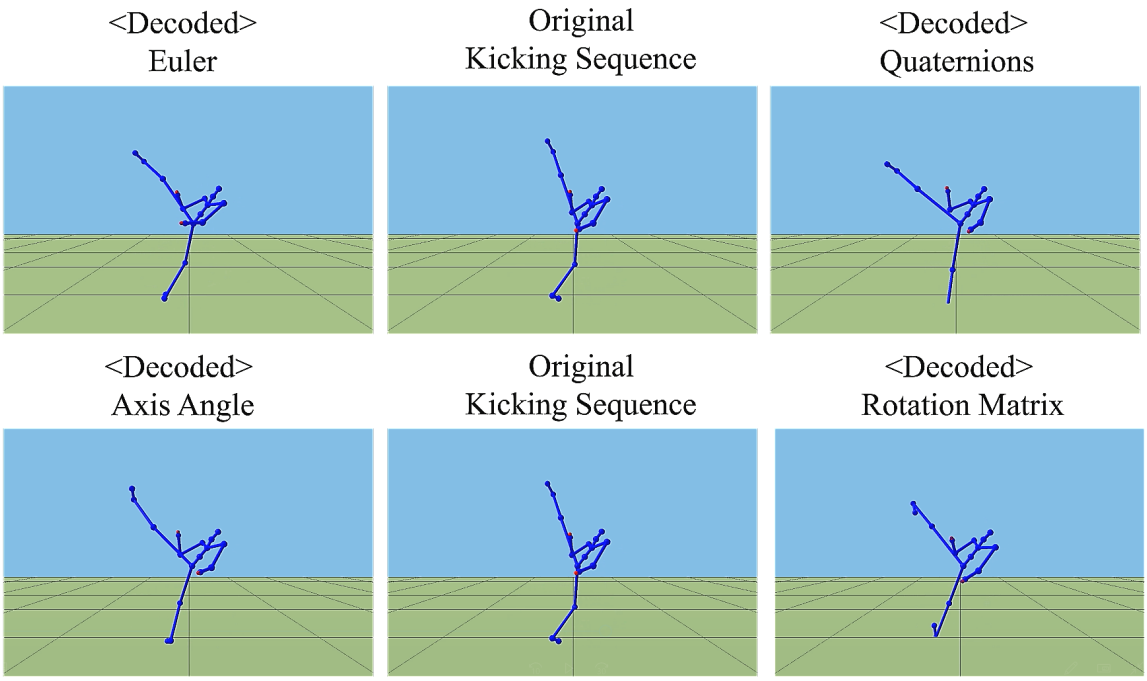

Figure 5.6: Decoded kicking animation comparison in a complex frame

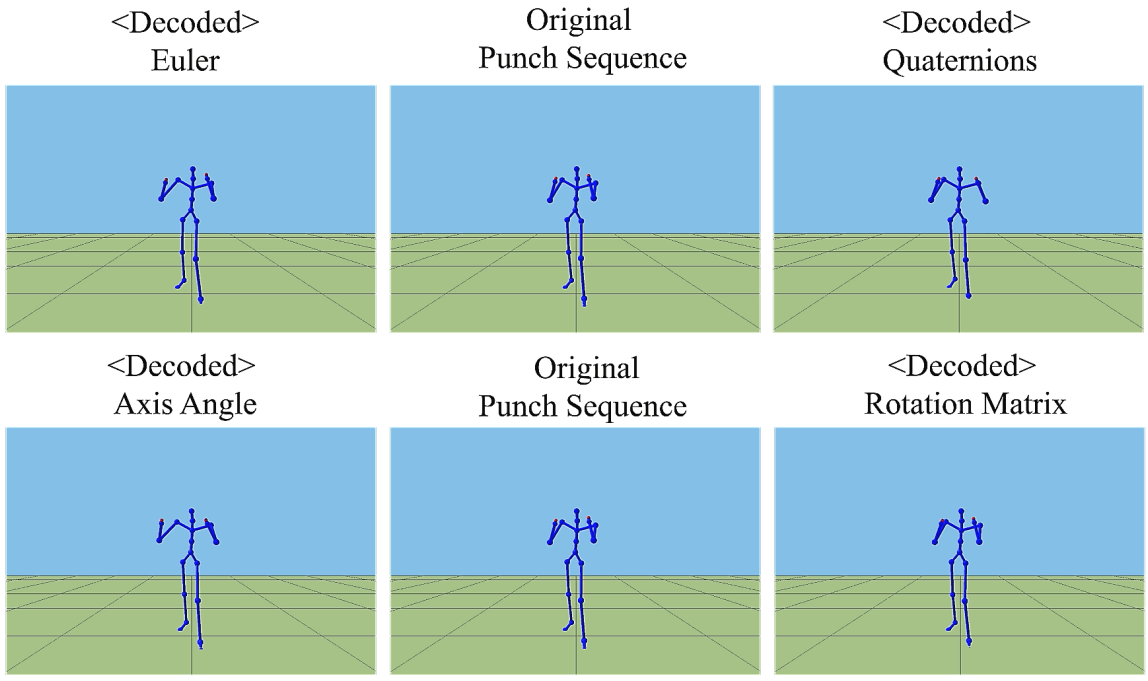

Figure 5.7: Decoded punch sequence comparison frame

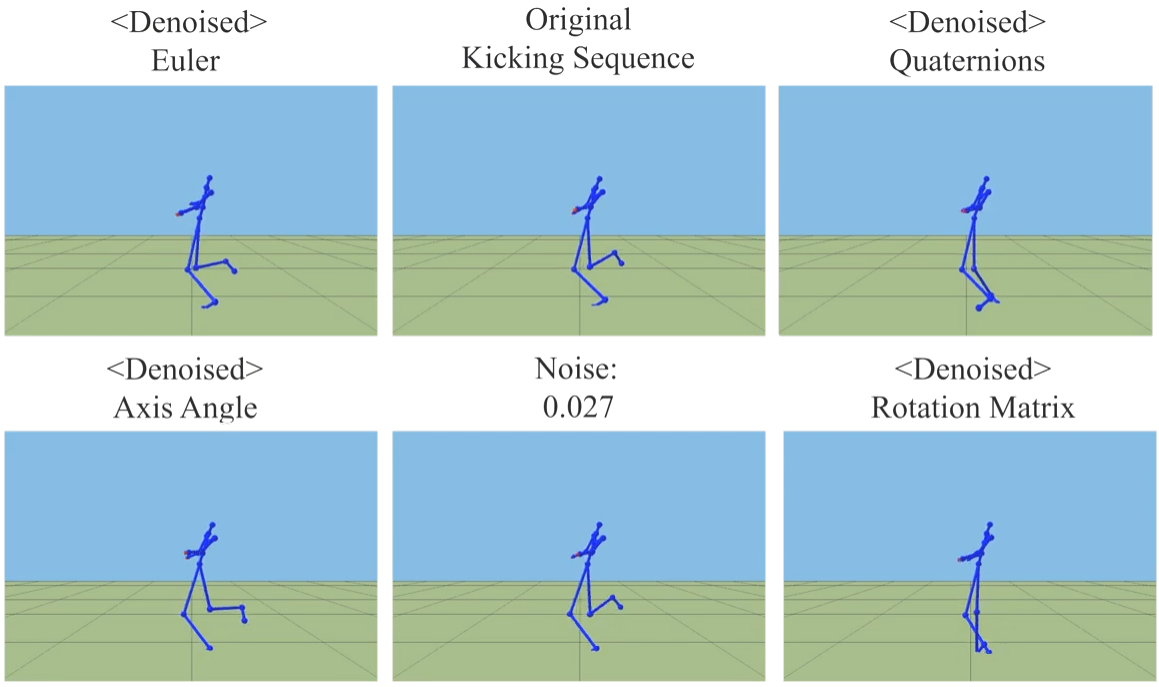

Figure 5.8: Denoised kicking sequence frame with n = 0.027

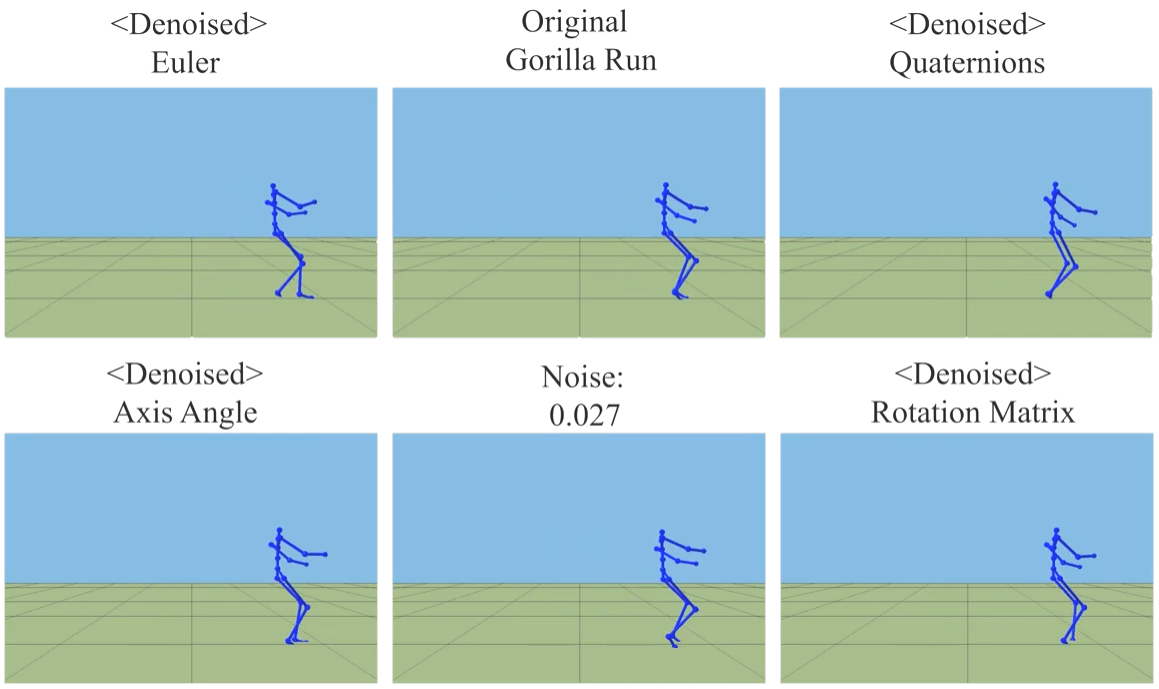

Figure 5.9: Denoised gorilla run frame with n = 0.027

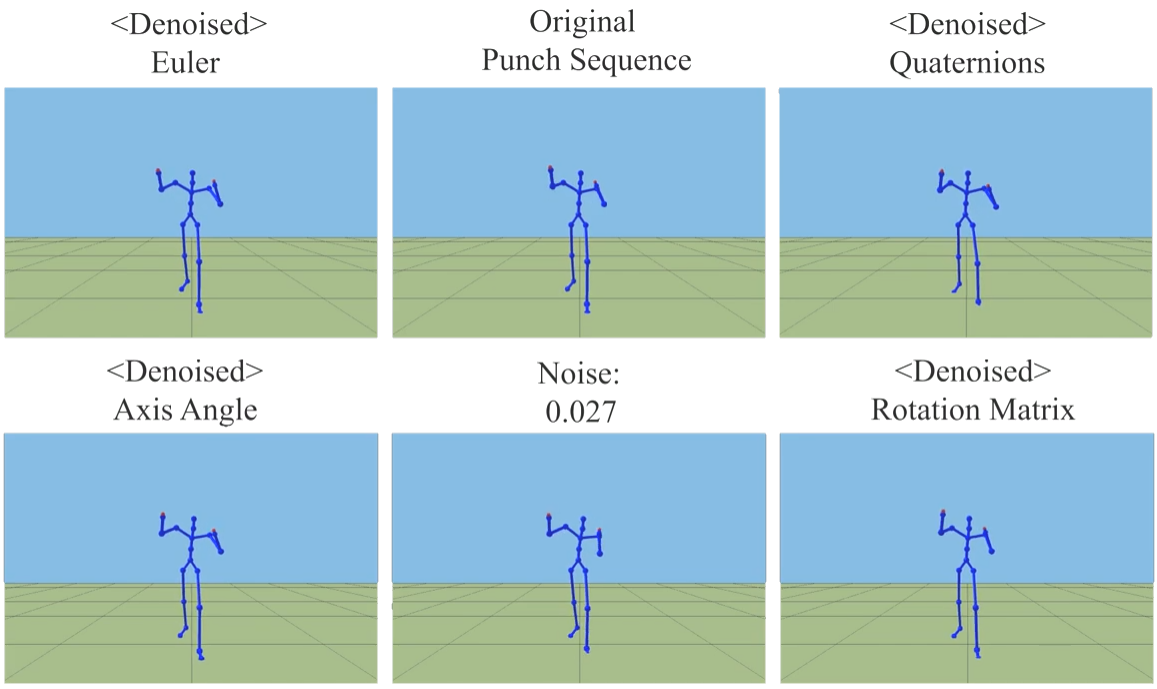

Figure 5.10: Denoised punch sequence frame with n = 0.027

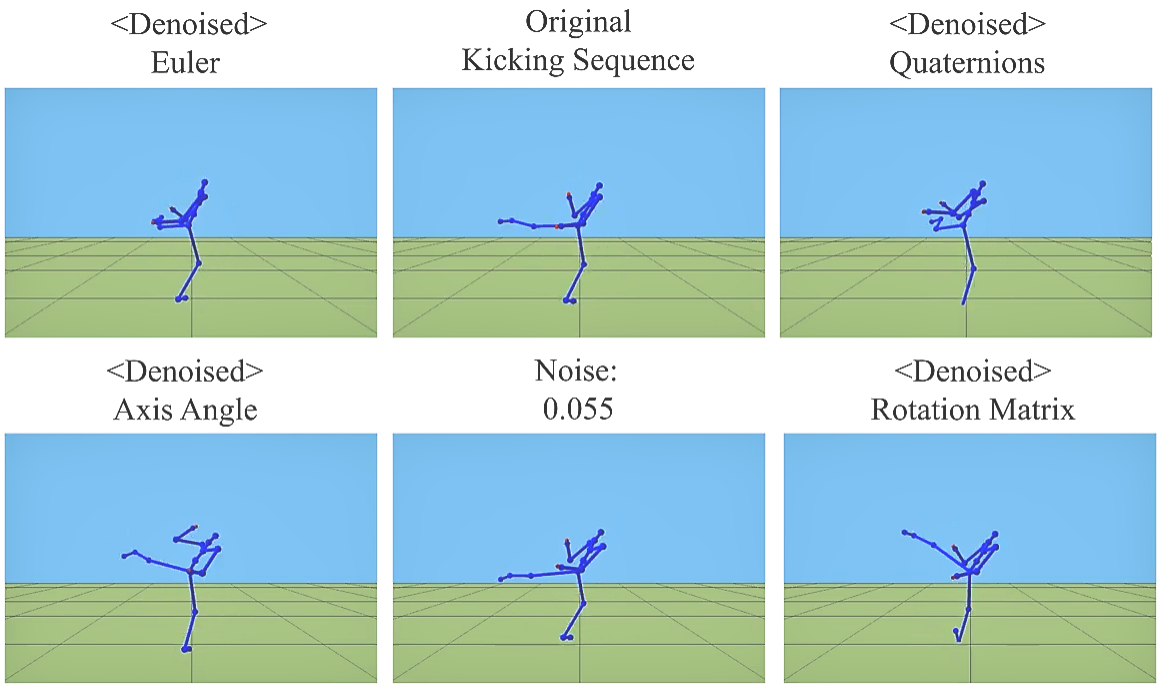

Figure 5.11: Denoised kicking sequence frame with n = 0.055

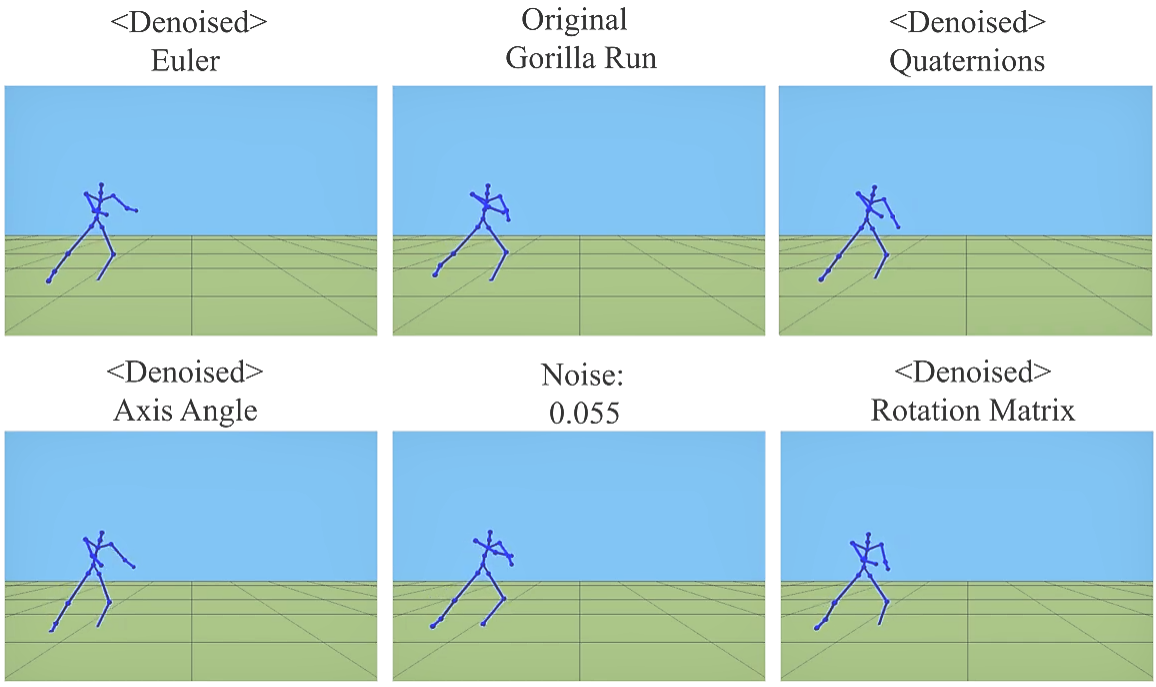

Figure 5.12: Denoised gorilla run frame with n = 0.055

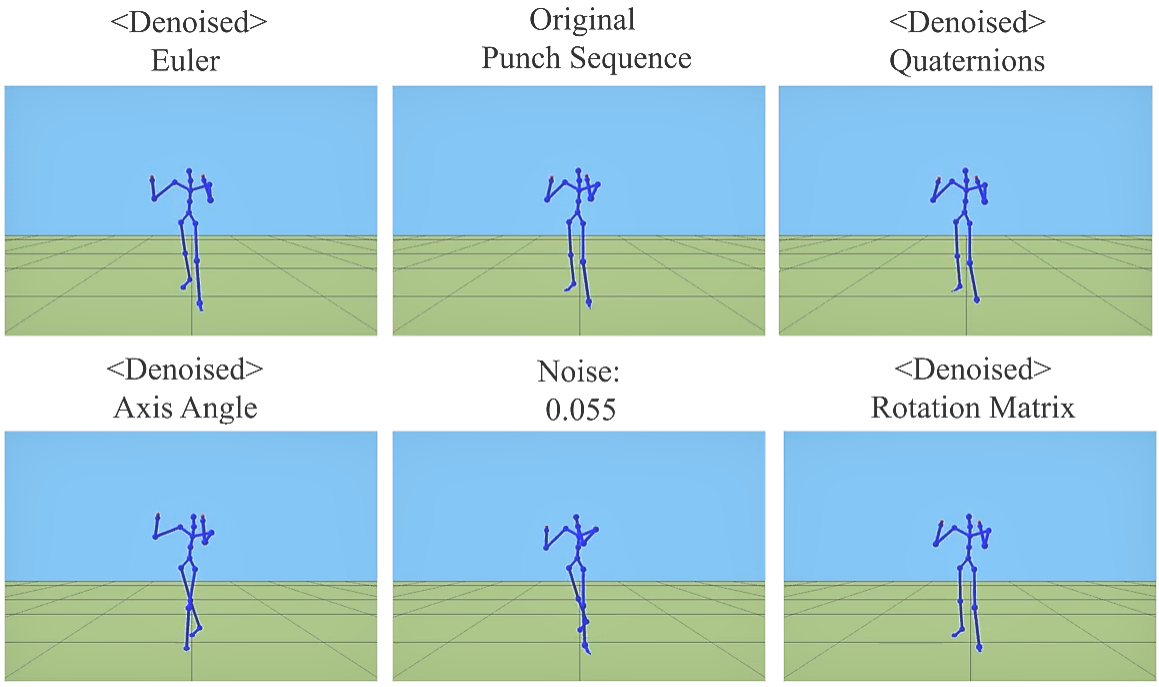

Figure 5.13: Denoised punch sequence frame with n = 0.055

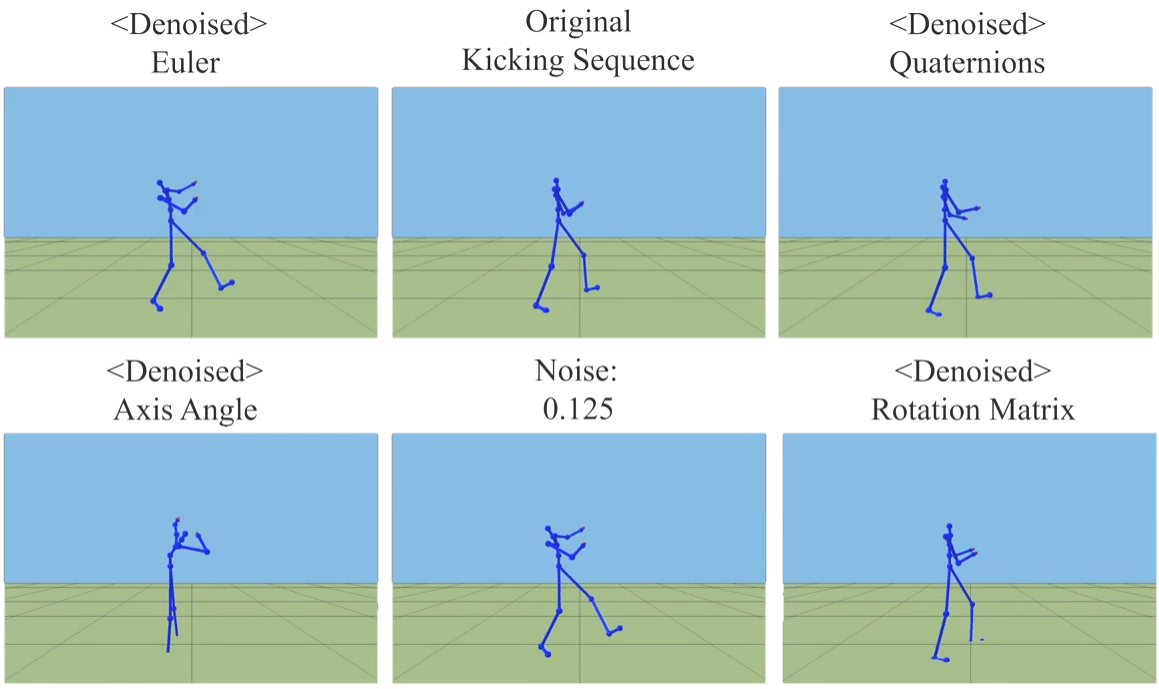

Figure 5.14: Denoising kicking animation frame with n = 0.125

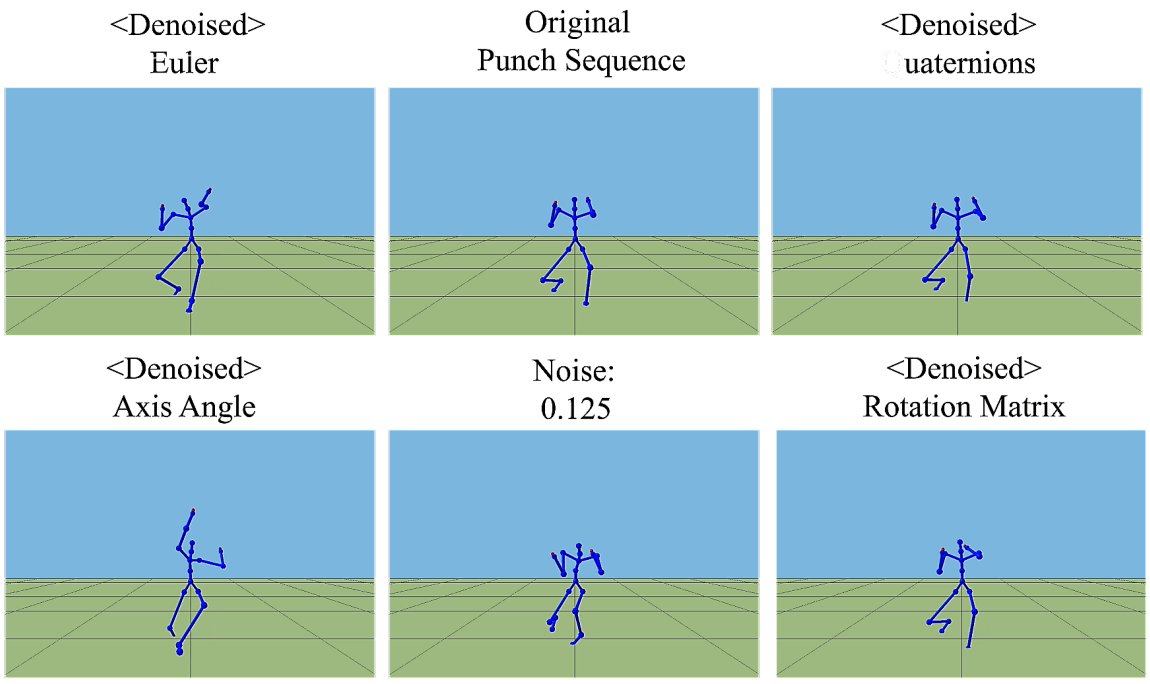

Figure 5.15: Denoised punch sequence comparison frame with n = 0.125

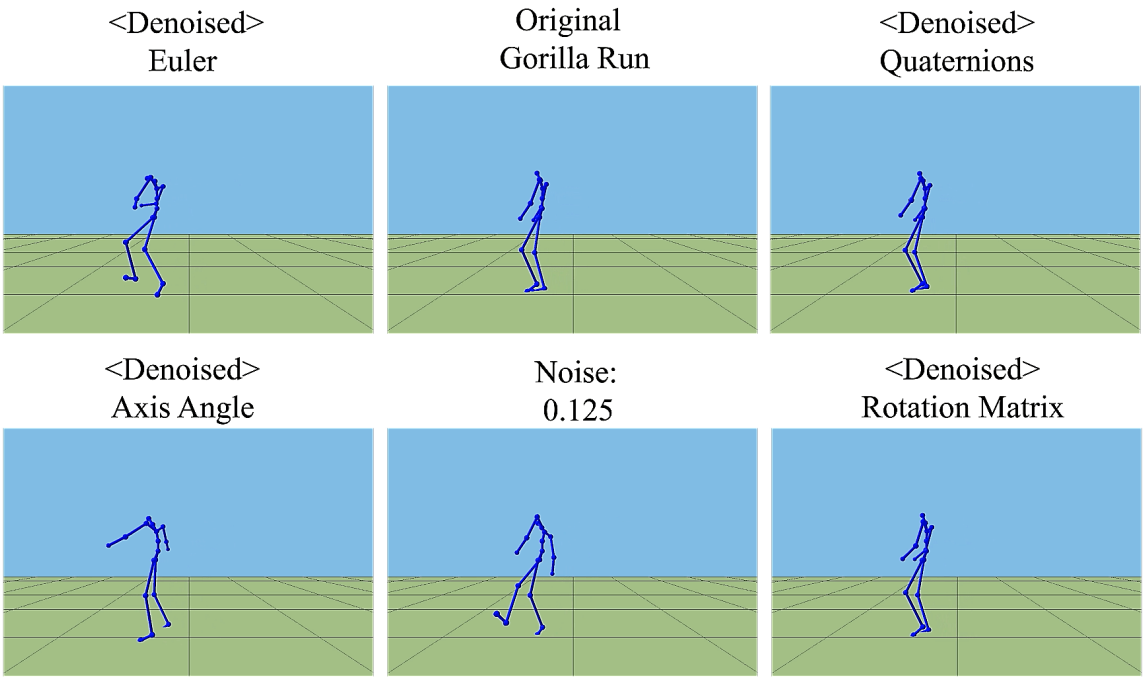

Figure 5.16: Denoised gorilla run comparison frame with n = 0.125

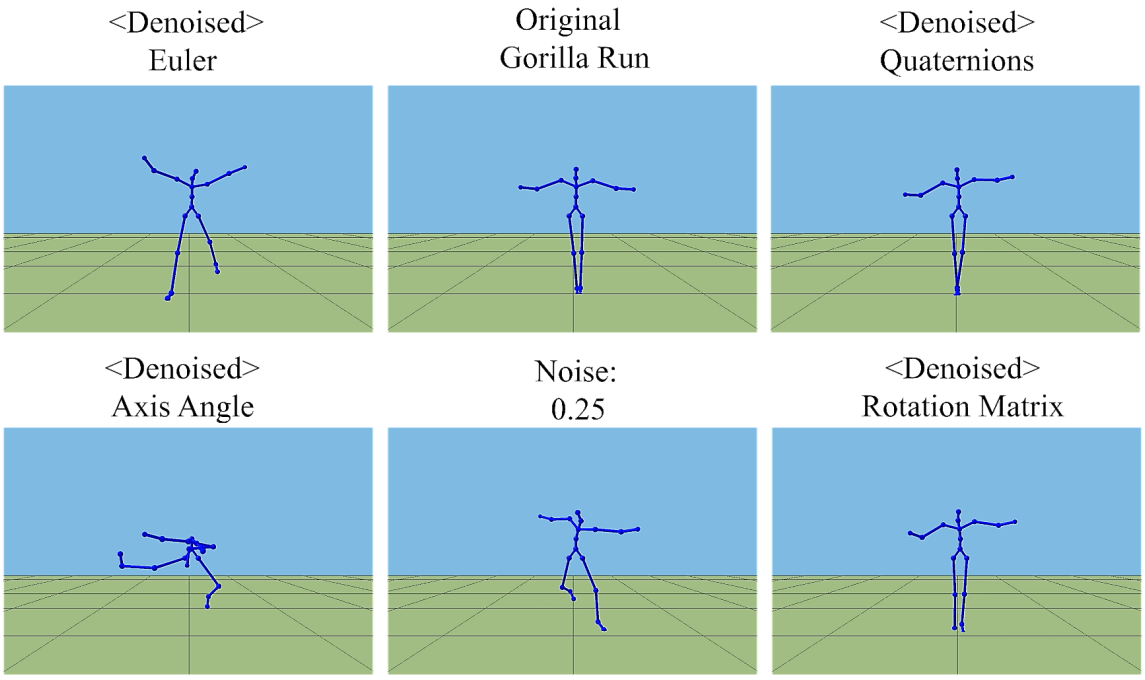

Figure 5.17: Denoised gorilla run comparison frame with n = 0.25

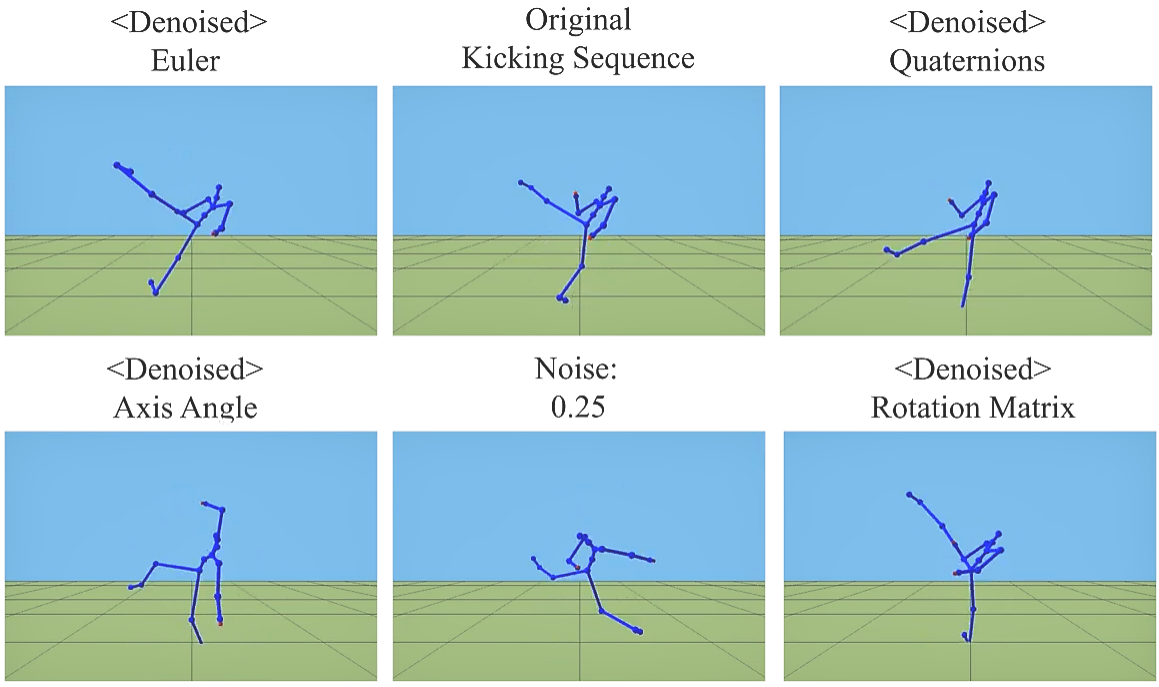

Figure 5.18: Denoised kicking sequence comparison frame with n = 0.25

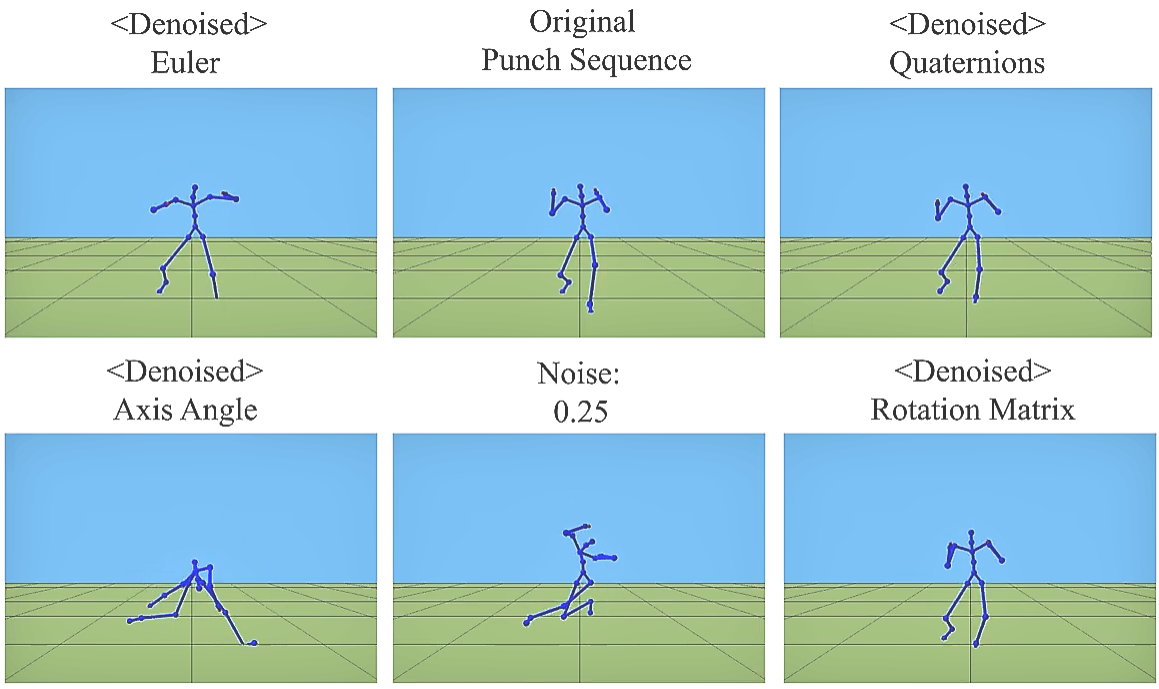

Figure 5.19: Denoised punch sequence comparison frame with n = 0.25

Chapter 6

Conclusion and Future Work

While conducting this research, I have experimented with various motion representations and asserted their effectiveness in Convolutional Autoencoders to parse human motion captures under four different rotation data models, Rotation Matrices, Axis Angles, Euler Angles, and Quaternions.

The designed autoencoders provide a novel insight into the comparative performance of these animation representation methods in an analog architecture, making them measurable in the same conditions, and thus possible to evaluate with quantitative metrics such as Minimum Square Error (MSE), and Root Mean Square Error (RMSE), as well as qualitatively through close observation of the naturality, its real-time performance after being decoded in full output sequences.

The trained subspace of human motion was tested under denoising properties of the learned manifold, with input clips presenting additive Gaussian noise 2.7%, 5.5%, 12.5%, and 25%. Under these conditions, the stability of Rotation Matrices and Quaternions was established at over 12.5% of added noise, with a threshold at which point Euler Angles and Axis Angles cease to be stable.

In the case of encoding-decoding, the method with a more accurate representation, on average, is Quaternions; it is also the smoother qualitatively. Even if it was slightly outperformed by Rotation Matrices during some validation RMSE tests, it is clear that in terms of robustness, the best method of representation remains be Quaternions, followed by Rotations Matrices, then Euler Angles, and finally Axis Angles.

It is important to highlight that all of the representation methods parsed in this study are working under relative angle orientations. During experimentation, the use of absolute positioning of

joints was detrimental in the sense that the models had much more difficulty in understanding the manifold. So unless the learning of absolute positioning in joints is treated as a necessity, I would recommend the use of relative joint orientation instead, for further study. It was observed that this approach allows the data to work with a more standard scope, under which the neural network has more facilities to learn the motion-temporal features.

6.1 Future Work

For further developments beyond the scope of this research I would recommend exploring and evaluating other representation methods, such as exponential numbers, and as well as exploiting the capabilities of the learned motion manifolds while evaluating other possible applications of it, such as style transfer or animation generation, under local space angle transformations.

It would also be interesting to expand the training and test datasets with further clips from databases such as Edin [5.1.2] and the Concordia University (2018) Action and Motion Repository (CAMREP).

References

Butepage, J., Black, M. J., Kragic, D., & Kjellstrom, H. (2017). Deep representation learning for human motion prediction and classification. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 6158–6166).

Carnegie Mellon University. (2019). *Cmu motion capture database.* Retrieved 2019-02-02, from http://mocap.cs.cmu.edu/info.php

Chen, M., Shi, X., Zhang, Y., Wu, D., & Guizani, M. (2017). Deep features learning for medical image analysis with convolutional autoencoder neural network. *IEEE Transactions on Big Data*.

Concordia University. (2018). *Concordia action and motion repository.* Retrieved 2019-05-04, from https://github.com/vladimirdlc/CAMREP

Cooke, J. M., Zyda, M. J., Pratt, D. R., & McGhee, R. B. (1992). Npsnet: Flight simulation dynamic modeling using quaternions. *Presence: Teleoperators & Virtual Environments*, *1*(4), 404–420.

Diebel, J. (2006). Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, *58*(15-16), 1–35.

Goldstein, H., & Poole, C. (1980). *Classical mechanics*. Addison-Wesley Publishing Company. Retrieved from https://books.google.ca/books?id=9M8QAQAAIAAJ

Gondara, L. (2016). Medical image denoising using convolutional denoising autoencoders. In *2016 ieee 16th international conference on data mining workshops (icdmw)* (pp. 241–246).

Holden, D. (2014). Deep learning in neural networks: An overview. , 75. Retrieved 2018-06-12, from https://arxiv.org/abs/1404.7828

Holden, D. (2018). Robust solving of optical motion capture data by denoising. *ACM Transactions on Graphics (TOG)*, *37*(4), 165.

Holden, D., Komura, T., & Saito, J. (2017). Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)*, *36*(4), 42.

Holden, D., Saito, J., & Komura, T. (2016). A deep learning framework for character motion synthesis and editing. *ACM Transactions on Graphics (TOG)*, *35*(4), 138.

Ian Goodfellow, Y. B., & Courville, A. (2016). *Deep learning*. Retrieved from http://www.deeplearningbook.org (Book in preparation for MIT Press)

Keras. (2018). Why use keras? Retrieved 2018-06-29, from https://keras.io/why-use-keras/

Kitagawa, M., & Windsor, B. (2012). *Mocap for artists: workflow and techniques for motion capture*. Focal Press.

Lander, J. (1998). Working with motion capture file formats. *Game Developer*, *5*(1), 30–37.

Lou, H., & Chai, J. (2010). Example-based human motion denoising. *IEEE Transactions on Visualization and Computer Graphics*, *16*(5), 870–879.

Montufar, G. F., Pascanu, R., Cho, K., & Bengio, Y. (2014). On the number of linear regions of deep neural networks. In *Advances in neural information processing systems* (pp. 2924–2932).

Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 25). Determination press USA.

Nvidia. (2018). Nvidia geforce rtx launch event. Retrieved 2019-03-03, from https://www.slideshare.net/NVIDIA/nvidia-geforce-rtx-launch-event/31

Nvidia. (2019). Deep learning frameworks. Retrieved 2018-02-20, from https://developer.nvidia.com/deep-learning-frameworks

Pavllo, D., Feichtenhofer, C., Auli, M., & Grangier, D. (2019). Modeling human motion with quaternion-based neural networks. *CoRR*, *abs/1901.07677*. Retrieved from http://arxiv.org/abs/1901.07677

Rashid, T. (2016). *Make your own neural network*. CreateSpace Independent Publishing Platform.

Razzaq, A., Wu, Z., Zhou, M., Ali, S., & Iqbal, K. (2015). Automatic conversion of human mesh into skeleton animation by using kinect motion. *International Journal of Computer Theory and Engineering*, *7*(6), 482.

Rifai, S., Vincent, P., Muller, X., Glorot, X., & Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th international conference on international conference on machine learning* (pp. 833–840).

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, *61*, 85–117.

Tensorflow. (2018). Tensorflow github. Retrieved 2018-07-05, from http://github.com/tensorflow/tensorflow

Vedaldi, A., & Lenc, K. (2015). Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd acm international conference on multimedia* (pp. 689–692).

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P. (2010). Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res*, *11*, 3371–3408.

Vladimir de la Cruz. (2019a). *Human motion decoding/denoising - results.* Retrieved 2019-03-03, from https://www.youtube.com/watch?v=ZNI6BoclFjA&list=PL4LdhnnKucce6L0DmSZMfOdFPAEaUbf_c

Vladimir de la Cruz. (2019b). *Universal motion repository.* Retrieved 2019-03-04, from https://github.com/vladimirdlc/UniversalMotion

Wooldridge, D. (2019). *Bvh hacker interface.* Retrieved 2019-05-04, from http://www.bvhacker.com/interface.html

Xiao, J., Feng, Y., Ji, M., Yang, X., Zhang, J. J., & Zhuang, Y. (2015). Sparse motion bases selection for human motion denoising. *Signal Processing*, *110*, 108–122.

Xiao, Z., Nait-Charif, H., & Zhang, J. J. (2008). Automatic estimation of skeletal motion from optical motion capture data. In A. Egges, A. Kamphuis, & M. Overmars (Eds.), *Motion in games* (pp. 144–153). Berlin, Heidelberg: Springer Berlin Heidelberg.