Learning human motion with reduced marker sets

Emma GOUNÉ, School of Computer Science

McGill University, Montreal

DECEMBER, 2020

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Master of Computer Science

December 4, 2020

Abstract

We address the problem of human body motion prediction in the context of a reduced marker set, which is a case often encountered in productions where markers are occluded, mislabeled or noisy. We aim to find a subset of motion-capture markers that are able to provide fast and high-quality prediction for joint transform. The difficulty of capturing the motion of a performer in this context comes from the harsh environment in which it takes place. Indeed, strong and varying lighting, smoke generators, and other visual pollution is often part of the show, leading to motion capture systems that are cumbersome and expensive. Nevertheless, there has been a greater demand for integrating these motion capture effects in shows, but none of the state-of-the-art technique can handle such environments with few markers.

Our main contributions are as follows. First, we select subsets of markers that provide complete information on the motion by comparing different automatic approaches to computing optimized marker layouts. Then, we develop a robust motion prediction method for reduced marker sets. We do so by training a deep denoising network, a deep learning algorithm that compensate for noise and system latency. In addition, we provide a complete pipeline for data cleaning, prediction and motion reconstruction.

Abrégé

Dans cette thèse, nous appliquons une technique d'apprentissage automatique à la prediction de mouvements humains. L'objectif est d'utiliser un nombre réduit de marqueurs qui captent les mouvements. Cette problematique est souvent rencontrée dans les productions où les marqueurs sont cachés, mal étiquetés ou bruités. Nous cherchons un sous-ensemble de marqueurs de capture de mouvements capables de fournir une prédiction rapide et de haute qualité pour le mouvement des articulations humaines. La difficulté d'enregistrer et de suivre le mouvement d'un acteur dans ce contexte provient souvent de l'environnement dans lequel la capture se déroule. En effet, un éclairage puissant et varié, des générateurs de fumée et d'autres pollutions visuelles font souvent partie des spectacles, conduisant à des systèmes de capture de mouvement encombrants et coûteux. Néanmoins, l'integration de ces effets de capture de mouvements a suscité un engouement important dans les productions, mais aucune des techniques de pointe ne peut gérer de tels environnements avec peu de marqueurs.

Nos principales contributions sont les suivantes. Tout d'abord, nous sélectionnons des sous-ensembles de marqueurs qui fournissent des informations complètes sur le mouvement en comparant différentes approches automatiques. Ensuite, nous développons une méthode de prédiction de mouvements robuste pour des ensembles de marqueurs réduits. De la sorte, nous implémentons un réseau de débruitage profond, un algorithme d'apprentissage en profondeur qui compense le bruit et la latence de certains systèmes. Enfin, nous fournissons une procédure complète pour le nettoyage des données, l'algorithme de prédiction et la reconstruction des mouvements.

Acknowledgements

I would like to thank my supervisor, Dr Paul Kry for his guidance, support and advice.

I would also like to thank Gilles-Philippe Paillé, co-advisor of this project, and the VYV team for their assistance.

I gratefully thank Mitacs and VYV for their financial support.

Lastly, I would like to thank the School of Computer Science, as well as all my friends and family for their support and encouragements.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer-generated imagery and computer animation are largely integrated in our everyday lives. It is used in video games, movies and special effects in the entertainment industry and advertisement. Significant contributions on techniques for digitizing the movement of people, animals or objects appeared since the late 90's. This is essentially linked to the democratization of optical motion capture systems. This area is specifically focused on capturing human motion in order to synthesise smooth animation.

The first step to animate a 3D character is to capture the motion from an actor in a studio. Then, the raw marker data captured by the optical system require to be "cleaned". This step is done by softwares and animators and consists of fixing erroneous markers: they can be occluded, swapped, mislabeled, or the signal can be noisy. This step is usually time consuming and expensive. Moreover, once the data is cleaned, further steps are necessary including "solving", where rigid bodies are fitted to markers, and "reconstruction", where inverse kinematics is used to recover the character's skeleton.

Data-driven techniques, such as machine learning, offer an attractive set of tools for modeling the patterns of data and further predict new ones. Hence, we want to design a robust algorithm that is capable of both handling corrupt data and accurately predicting the motion. For this use, we have implemented a residual network (ResNet). According to [Hol18], this deep denoising feed-forward network can learn from corrupted data and

replace the ”solving” stage of the optical motion pipeline. For this prediction task, we train the network to map from marker positions to joint transforms directly.

Additionally, we evaluate the relevance of different data model evaluation techniques for motion prediction. Our goal is to set up a validation model that will test the network's ability to predict poses from new data. For a given task, models' details such as validation techniques can vary in the literature, making it sometimes hard to navigate and improve. Thus, we compare two different validation models in this thesis and try to establish a a complete pre-processing pipeline when dealing with motion capture data. The first one is k-fold cross-validation, a non-exhaustive cross validation method that involves partitioning the data into subsets, performing the validation analysis on one subset, and the training on the remaining ones. This is a common method for classification tasks. The second approach is a modified cross-validation inspired from time-series forecasting. We adapt our previous method to take into account the temporal order of the motion data. Finally, we evaluate the impact of both validation methods on the model's performance.

There has been a greater demand for integrating motion capture effects in live shows, concerts, and theatrical events. The difficulty of capturing the motion of a performer in this context comes from the harsh environment in which it takes place. Indeed, strong and varying lighting, smoke generators, and other visual pollution is often part of the show, and traditional optical systems often fail in these environments. One of the solution is to reduce the number of body markers that we use and track. This helps reducing the noise, and the impact of swaps and mislabelling. In other words, we aim at finding a subset of motion-capture markers that are able to provide fast and high-quality prediction for a complete human motion. This is still an area of active research in facial or hand motion simulation where the motion and marker placement is very intricate. We will consider previous work from Liu et al [LZWM06] and on the symmetry of human gait Ha et al. [HWPR+10] to chose combinations of only 6 markers to learn from.

The alliance of a fast and robust neural network with a reduced marker set constitutes a new domain with largely unstudied potential for full body motion. In this thesis, we

successfully evaluated the best validation techniques for motion capture data, and build a complete pipeline for pose estimation with only 6 markers, i.e. we were able to do the "solving" step for the raw reduced data as well as the retargetting step. The experimental results demonstrate that our method can quickly generate plausible human motions on a frame-by-frame basis.

Chapter 2

Literature review

This thesis bridges the gap between traditional computer animation techniques and flourishing artificial intelligence techniques. This section contextualizes our research in these two area and provide a brief literature survey on previous papers that used machine learning approaches to learning animated motion from motion capture.

2.1 Motion capture

A significant part of full body motion estimation is to register the motion, a process known as human motion capture. The main aspect is to capture large scale body movements, which are the movements of the head, arms, torso, and legs. The systems used to capture human motion consist of subsystems for sensing and processing, most of them rely on placing devices on the subject and in the surroundings which transmit or receive generated signals. These sensor technologies can be either active sensing such as cameras equipped with markers in a MoCap system, or passive sensing such as force sensors, gyroscope, reflective markers etc. We are mostly interested in building a pose estimation system from a MoCap system, that is by placing markers on a studio actor and record the motion with cameras. The markers information is then used as input data for our learning algorithm.

Human motion capture data usually needs to be ”cleaned”, ”solved” and ”reconstructed” due to interference that occur during a capture. Occlusions can cause mislabelling, too many markers may inhibit natural movement. Moreover, markers mislabelling or disappearance is amplified as the number of markers increases. Park et al. [PH06] have studied the impact of using a very large set of markers (approximately 350) to capture skin deformation, and found that dynamic motions tend to have more disconnected marker trajectories. They developed a cleaning method to counter the numerous marker swaps and occlusions. However, this approach requires a high computation power and is time-consuming. Extensive research is conducted on finding good body landmarks (see Steimle et al. [SBLW+17]). To the extent possible, the markers are placed on body landmarks so that they can more easily be used to approximate the motion of the skeleton. Recent work by Koźbiał et al. [KMS20] uses 3D scans and mid-surface projection to evaluate and locate points of interests for body landmarks. These points of interest can provide valuable information on where to position markers, especially in a setting where we are reducing their number.

In addition, an automatic approach to computing optimized marker layouts is crucial for a precise study of movements, as presented in Dagnes et al. [DMV+19] and Schröder et al. [SMB15]. It was mainly applied to hand motion reconstruction, which can be complex due to the hand anatomy. The use of a reduced marker set does not change the hand movement perception. This result was presented in Hoyet et al. [HRMO12], showing that for the majority of cases 8 markers produces sufficiently high quality motions. This observation is supported by other studies on hand motion reconstruction. In Wheatland et. al [WJZ13] using Principal Component Analysis coupled with a linear regression model, they were able to reconstruct hand motion close to the original with a sparse marker set of three to six markers. Other selective automatic approaches are derived from PCA, in order to isolate and select markers that carry the most information about the full body movement. In their paper, Liu et al. [LZWM06] present a selective approach for motion learning and reconstruction based on principal feature analysis (PFA) (see Cohen et

al. [CXZ+02]). Their results show that a set of six markers is consistent and stable under different initial settings. However, this linear technique is not optimal to predict complex motion and requires many local basis for interpolating large data. In order to efficiently train a neural network, we want to reduce the data dimensionality and select markers that maintain the most information to accurately predict a pose.

Techniques that infer human motion from partial information are part of a larger story, especially when it comes to commercial devices. One of the first popularized game that uses partial information was the Wii Balance Board. It is capable to provide reliable information about the full body pose from ground reaction forces. Such applications were further studied by Yin et al. [YP03] for their FootSee device, and it becomes a very powerful and accurate tool when coupled with hand tracking devices that provide arm movement information (see Ha et al. [HBL11]). Another user interface that uses force sensor is the Tango ball by Kry et al. [KP08], that measures the contact pressures and acceleration for hand movement reconstruction. Whereas these techniques work well for VR headsets or the Wii and provide a solid point of departure for real-time application, they are not highly accurate and usually need to be coupled with other non-intrusive sensing techniques like cameras, which was explored in Aladdin et al. [AK12].

2.2 Machine learning

Machine learning techniques are attracting the interest of the computer animation community from speech animation in Taylor et al. [TKY+17] to hand pose estimation in Oberweger et al. [OWL]. Properties of ML systems, such as artificial neural networks (ANN), include high scalability and automatic features learning. These networks have been evaluated by Du et al. [DWW15], among others, to be very efficient for data-driven motion prediction.

Recently, more methods employing convolutional neural network (CNN) based learning approaches led to important progress for single-person 3D pose estimation (see [LC14,

LZC15, TKS+16]). For full body mocap data, a family of methods based on deep recurrent neural networks (RNNs) and encoder-recurrent-decoder (ERD) have shown good performance on this task while also trying to predict poses. For instance, Martinez et al. [MBR17] focus on learning time-dependant representation that perform task such as long-term human motion synthesis and short-term prediction. Whereas [FLFM15] propose an ERD approach to motion recognition and prediction, where the model is a recurrent neural network that incorporates nonlinear encoder and decoder networks before and after recurrent layers.

We are particularly interested in networks able to handle corrupted or partial data. Denoising neural networks usually outperforms other models for this task. It was first introduced by Vincent et al. [VLL+10] as an unsupervised-learning method to train stacked denoising autoencoders in order to guide the learning of higher representation. Denoising autoencoders achieve state of the art results for classification, regression or data-recovery tasks (see [AVT+17, LTMH13]). Recent work on denoising autoencoders from Mall et al. [MLCC] and Holden et al. [HSKJ15, HSK16] introduce custom noise functions to synthesize and edit motion. In continuity with these results, Holden et al. [Hol18] present a deep neural network approach to denoising corrupted motion capture data. The latter implement a multi-layer residual neural network (ResNet) coupled with a custom noise function for motion prediction. We take inspiration from deep artificial neural networks and denoising networks to predict the joint transforms of reduced marker sets.

Chapter 3

Motion capture concepts

Traditional character animation includes manually positioning the 3D characters, it involves skilled artists and takes large amount of time. Motion capture (or MoCap) was introduced as a digital rotoscoping[1] technique to produce faster and more realistic motion. Nowadays, this method works by recording actors performing a desired motion in a studio, capturing and tracking their movements (see Figure . The techniques won its spurs in the entertainment industry. For instance, in Peter Jackson's movie "Lord of the Rings: The Two Towers", the character Gollum played by Andy Serkis was revealed to be animated with motion capture. It was during the process of creating Gollum that the technology developed by WETA [2] was created to allow motion capture actors to act in real time on set. Then, motion capture was popularized and improved to a point of creating believable characters and environments. James Cameron's "Avatar" (2008) is an example of hyper realistic capture of movement and facial expressions.

3.1 Optical systems and markers

We are particularly interested in application where the captured motion is used to provide controlling functionalities; meaning that it could be used as an interface to games, virtual

[1] https://archive.org/details/themakingofbrilliance1985
[2] https://www.wetafx.co.nz/

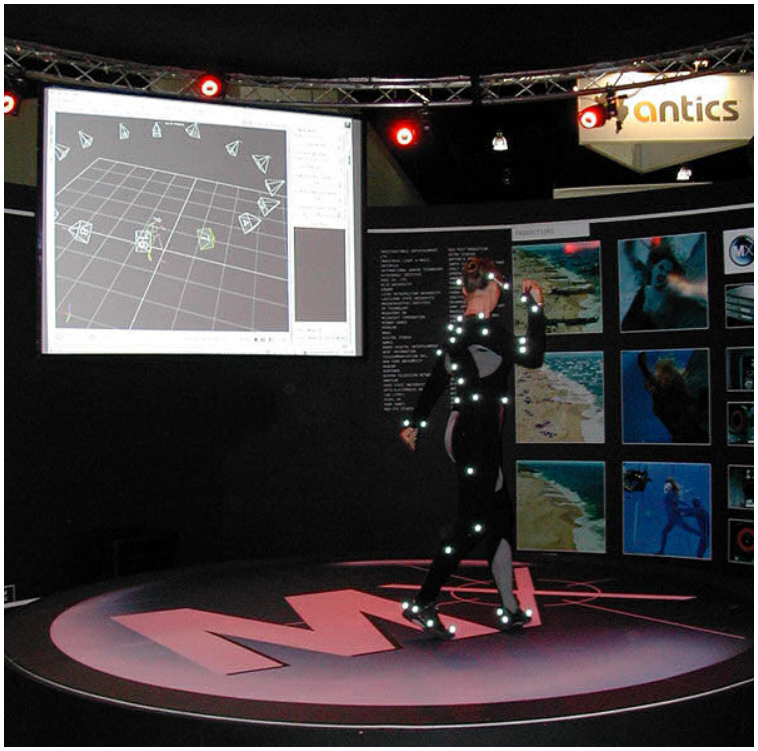


Figure 3.1: A dancer wearing a suit used in an optical motion capture system.

environments, animation or real time motion detection and prediction. As previously seen on Figure 3.1, data acquisition is implemented using special markers attached to an actor. Only a black and white images (2 bits) of these points are captured, and not the images of the actor's body. This is called an optical system. Traditionally, performers wear markers near each joint to identify the motion by the positions or angles between the markers.

We are particularly interested in two types of markers that provide different information about the position and movement: the passive markers and the active markers. In passive optical systems, the cameras emit infrared radiation, reflected by the markers whose surface is coated with reflective tape, then sent back to these same cameras, which have IR filters to cut out all other illumination. The positions in the spatial frame of each marker is deduced by triangulation from the image processing of a minimum of 2 cameras. Generally, markers need to be tracked by a software engine, that is to say: register them in an identification tracking list from the T-pose (initial T position of the actor); and

sometimes manage occlusions and swappings. In active optical systems, instead of reflecting the light, the markers emit their own signal. The active markers directly emit an infrared signal captured by photosensitive cells. The cells are sensitive to a type of wavelength and can identify active markers in real time. These markers are therefore identified automatically and immediately in real time dynamic points with 3D coordinates in virtual space.

Our goal is to infer joint positions from markers, we need to have identified markers, or at least a set of them. Hence, we will deal with passive markers that have been identified or labelled in a pre-processing step. We didn't have access to an active optical system, however, our algorithm is also compatible with active markers or a mix of active and passive markers with small changes in nomenclature and labelling.

3.2 Motion capture file format

In the animation jargon, the whole character that is animated is called a skeleton. It is composed of bones that are linked in a hierarchical structure. They are the smallest segment subject to individual rotation and translation changes during the motion. To represent the local position, orientation or scale changes of a bone we introduce a parameter called a channel. Finally, the animation is comprised of a number of frames, where for each frame the channel data for each bone is defined.

We use the bioVision hierarchical data (BVH) data format from the Carnegie Mellon University Motion Capture Database, or CMU. A BVH file consists of two parts: the first section details the hierarchy and initial pose (T-pose) of the skeleton, and the second section, or motion section, provides the channel data for each frame (see Figure 3.2). The structure of the files is similar to a hierarchical tree structure, where a node contains a bone. The root of the hierarchical tree is the hips and all the other bones derive from it. For instance in Figure 3.3, we can see that the Left Elbow has the Left Shoulder as parent node and the Left Wrist as child node, the left Shoulder has the Left Collar as parent node

```
HIERARCHY
ROOT Hips
{
        OFFSET 0.00 0.00 0.00
        CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation
Yrotation
        JOINT Chest
        {
                OFFSET 5.00 0.00 0.00
                CHANNELS 3 Zrotation Xrotation Yrotation
                End Site
                {
                        OFFSET 0.00 5.00 0.00
                }
        }
        JOINT Leg
        {
                OFFSET -5.0 0.0 0.0
                CHANNELS 3 Zrotation Xrotation Yrotation
                End Site
                {
                        OFFSET 0.0 5.0 0.0
                }
        }
}
MOTION
Frames:         2
Frame Time: 0.033333
 0.00 0.00 0.00 0.00 0.00 0.00  0.00 0.00 0.00  0.00 0.00 0.00
 0.00 0.00 0.00 0.00 0.00 0.00  0.00 45.00 0.00  0.00 0.00 0.00
```

Figure 3.2: An example BVH file with two joints attached to the root (hips).

etc. The translation of the origin of the bone is defined with respect to its parent's origin, meaning that every information is given in local coordinates. The root is the only node in the tree that does not have a parent node, and the nodes at the extremities that do not have a child node are called End Sites. The bvh files from the Carnegie Mellon University Motion Capture Database that we use in this project are hierarchical trees with 32 nodes.

3.3 Kinematics chain

The motion of an individual bone consists of translation, rotation and scale, which can be merged together to give an overall transform using homogeneous coordinates. The combination order of these different transforms give the full transform matrix of a given bone and follow

$$M = TRS, \tag{3.1}$$

where S, R and T are the separate scale, rotation and translation matrices respectfully. The construction of the rotation matrix R, can be easily done by multiplying together the

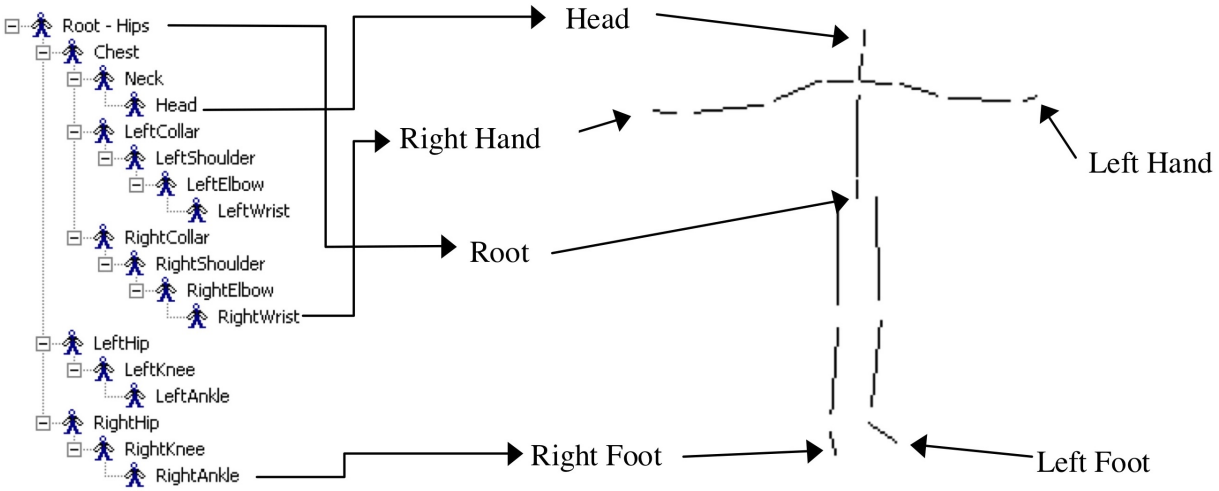


Figure 3.3: Hierarchical Structure of a sample BVH file with the sample T-pose of the skeleton.

rotation matrices for each of the different channel, this gives for our case the new rotation matrix:

$$R = R_z R_x R_y. \tag{3.2}$$

We can rewrite the transform matrix M into homogeneous coordinates simply by adding a last row such that:

$$M = \begin{pmatrix} & & & T_x \\ & R & & T_y \\ & & & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.3}$$

Normally, the root is the only bone that has per-frame translation data, however each bone has a local offset that needs to be added to the local matrix stack. Therefore, T_x, T_y and T_z represent the summation of a bone's local position and frame translation data.

Now, to obtain a global matrix transform for a given bone, the local transform needs to be pre-multiplied by its parent's global transform. The following equation exemplifies this process for the LeftAnkle in Figure 3.3:

$$
\begin{aligned}
v'_0 &= M_{Hips} M_{LeftHip} M_{LeftKnee} M_{LeftAnkle} [0,0,0,1]^T \\
v'_1 &= M_{Hips} M_{LeftHip} M_{LeftKnee} M_{LeftAnkle} v,
\end{aligned}
\tag{3.4}
$$

where v'_0 and v'_1 are the endpoints of the bone with local orientation v, M_i are the local transforms of the bones involved in the hierarchical chain, and $[0,0,0,1]^T$ is the local origin of the LeftAnkle.

Chapter 4

Machine learning background

Machine learning models are typically well adapted to learn from incomplete or noisy datasets, since many models are designed to capture the inherent noise and data complexity. In our approach, we are trying to skip the solving stage of the optical motion pipeline and directly learn from the raw motion data. In this chapter, we provide a description of our neural network, as well as the metrics and the hyperparameters that we tune.

4.1 Artificial neural network

Artificial neural networks are one of the main tools used in machine learning. As the "neural" part of their name suggests, their systems are inspired by the neuronal connections in brains. Neural networks are known to be universal function approximators, most of the time the functions are nonlinear. The ANN task is to optimize weight values through a learning method.

Such networks can be broken down into layers. The inputs to the neural network constitute the input layer. This layer consists of passive nodes, they only transmit the signal to the following layers: the hidden layers. There is an arbitrary number of these layers with arbitrary number of neurons. The nodes are modifying the signal (and changing the

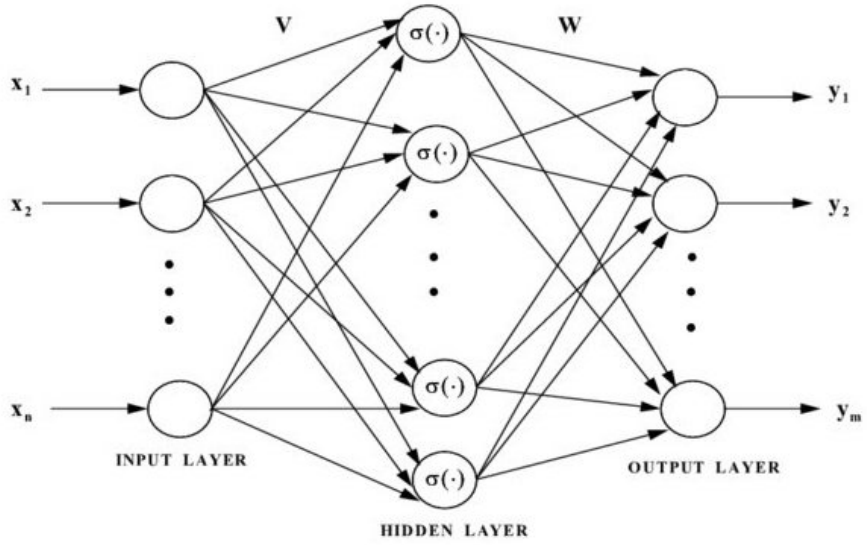


Figure 4.1: Diagram of an artificial neural network. V: input weight matrix, W: output weight matrix, σ: activation function (function that is used to optimize the weight). In our task, the input layer is the marker position and the output layer is the joint transform.

weights), they are active. The number of neurons in the output layer corresponds to the number of the output values of the neural network. The nodes in this layer are also active.

The network is essentially learning how to produce the output within desired accuracy corresponding to an input pattern. The final result is the updated optimal weights. Our aim is to predict the class of an input vector, see Figure 4.1. This is called a classification task. To do this we implement a Supervised Learning method since such methods require a training set which consists of input vectors and a target vector associated with each input vector.

4.1.1 Activation functions

We previously learned that ANNs consist of various layers of interconnected artificial neurons. The neurons are powered by activation functions which help in switching them On/Off. Each neuron receives an input and weights, which is then added with static bias value (unique to each neuron layer). The sum is then passed to an appropriate activation function that determines the final output of the neuron, see Figure 4.2.

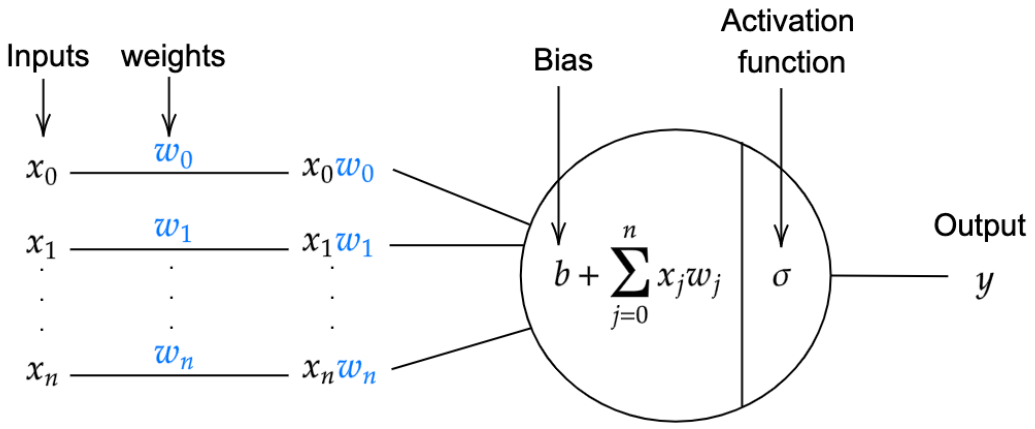


Figure 4.2: Example of a neuron with its inputs, corresponding weights, a bias, and the activation function applied to the weighted sum of the inputs.

The activation function transforms its inputs into outputs that have a certain range. It is essentially the switch that turns the neuron On or Off. Without an activation function, the output signal becomes a simple linear function, and the network will act as a linear regression model with limited learning power.

We are particularly interested in rectified linear unit (ReLu) activation functions. A ReLU function is simply,

$$\sigma(x) = max(0, x). \tag{4.1}$$

The function returns 0 for negative inputs, but returns positive values back. Thus, such activation functions' main property is sparsity.

Moreover, this function and its derivative are both monotonic. This is important because neural networks are trained using the process gradient descent. It consists of a backward propagation step (chain rule) to get the change in weights in order to reduce the loss after every epoch. It is important to note that the derivatives play an important role in updating of weights.

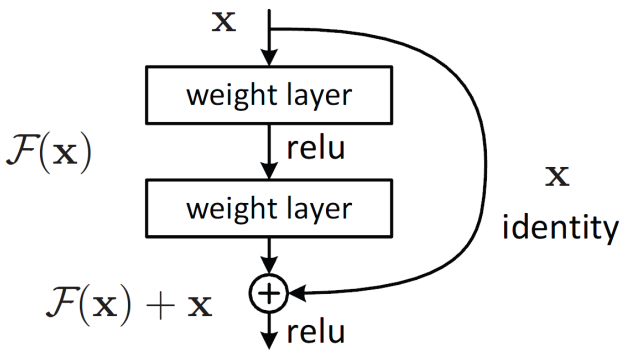


Figure 4.3: A skipped connection in Residual learning.

4.1.2 Residual networks

Now that we have defined the components of a network and the activation function, we are interested in the network architecture. We take inspiration from Holden et al. [Hol18] and opt to build a inception deep residual network (ResNet). This model has already won its spurs in pattern recognition in Szegedy et al. [SIVA17]. The ResNet model, with ReLu activation function, solves a major problem with deep neural networks. When deeper networks start converging, a degradation problem occurs: with the network depth increasing, accuracy gets saturated and then degrades rapidly. This is called the vanishing gradient problem and was a limitation on previous application of deep action recognition.

In a residual network, the objective is to fit a residual mapping and skip connections. Instead of learning a direct mapping $x \rightarrow y$ with underlying function $H(x)$, we define a residual function using $F(x) = H(x) - x$ (called a "shortcut connection"), which can be re-framed into $H(x) = F(x) + x$, where $F(x)$ and x represent the stacked non-linear layers and the identity function (input=output) respectively (see Figure 4.3). In this way, we resolve the degradation problem.

4.1.3 Metrics

Our neural network solves a classification task, hence we need a supervised learning metric to evaluate its performance. In our model, and for each frame, the observations

are the marker positions around joints and the class they are matched to is the associated joint transform. An observation belongs to a unique class, hence we choose to use the classification accuracy as metric. The accuracy can be defined as the ratio of number of correct predictions to the total number of input samples. For the validation set, the metric is computed at the end of each epoch and on the complete validation dataset. The train set metric is computed at the end of each batch and the average is constantly updated until the epochs end.

4.2 Hyperparameters tuning

In order to optimally solve our machine learning problem we will have to tune several hyperparameters, scaling them will change the performance of our metric. Since there is no predefined way of choosing them, we started from the information in Holden et al. [Hol18] and refined them from there.

Mini batch

We know that an epoch is one forward pass and one backward pass of all the training examples. However, this step can be very long and memory expensive. Especially in our case, our machines could not fit the whole dataset. The first step for boosting our algorithm when dealing with a lot of samples is to cut an epoch into multiple mini batches iteration. The batch size defines the number of samples that will be propagated through the network for this iteration. For example if you have 1000 training samples and a batch size is 500, then two iteration will be needed to complete one epoch. This technique has a lot of advantages: it reduces the cost of memory and it trains faster, because we update the weights after each propagation. We used mini batches of size 256, as presented in Holden et al. [Hol18].

Regularization

When we want to reduce the model high variance, in the case of overfitting, we can increase the data or add a regularization term, this reduces the complexity of the model. In our case we added l_2 regularization. Increasing the regularization parameter results in a greater weights decay during the gradient descent update. Hence, the weights of most of the hidden units will be close to zero and the network will be simpler.

Learning decay rate

An other way to speed up the process is to slowly reduce the learning rate over time. This is called learning rate decay. If you start with a relatively large learning rate, the training will be faster. In order to have a higher chance to reach a global minima we slowly reduce it over time.

Chapter 5

Methodology

In this chapter we will detail our technical approach. We will give a detail description of the experimental data, followed by how the markers are positioned and the features extracted. Then, we will describe the the different pre-processing approaches that we explored as well as the hyperparameters of our network. A brief description on the reasoning behind the hyperparameters for this specific model will be given. Finally, we will cover the inverse kinematic and smoothing methods used for motion reconstruction. An overview of our pipeline for predicting a motion with a reduced marker set is displayed in Figure 5.1.

5.1 Pre-processing

Data pre-processing is an essential step to enhance the network efficiency. This step includes several techniques like cleaning, normalization, feature extraction, transformation etc. We will go through each of them in this section. In our case the type of data are motion capture files, with different motion types and character sizes. We aim at creating a pre-processing pipeline that scales and standardizes the input files as well as sorting and splitting them into training and validation sets to prepare for training. To this end, we will explore two validation techniques applicable to our classification task.

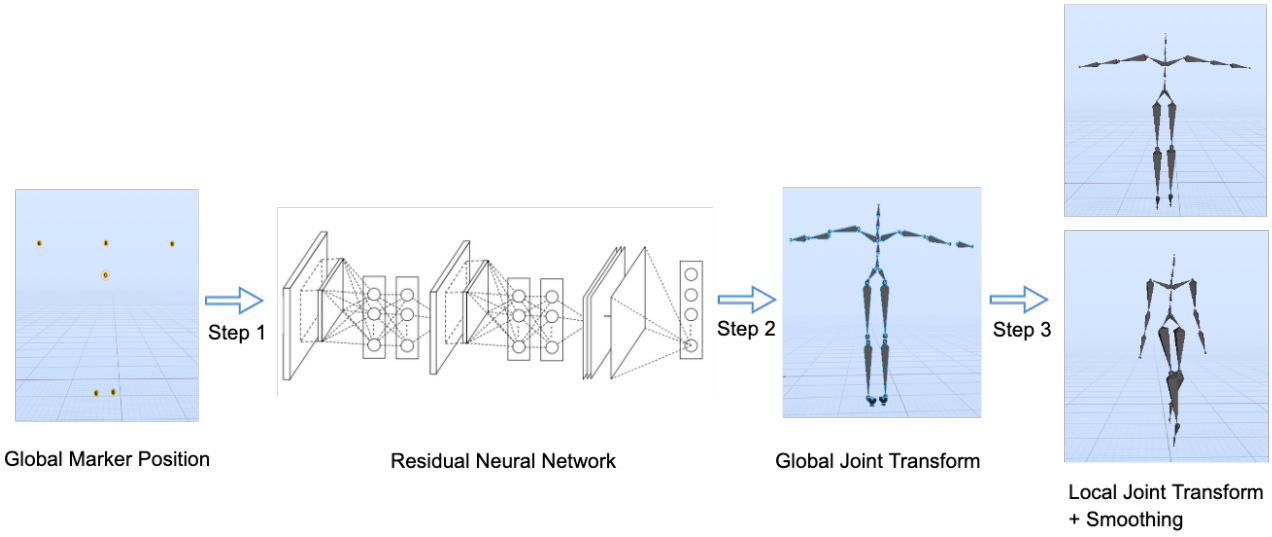


Figure 5.1: Outline of the pipeline for motion prediction. The first step includes pre-processing the data and feeding the reduced markers' global position to the trained ResNet. The second step consists of retrieving the global joint transform predicted by the network. The last step is post-processing, which involves building the kinematics chain back and smoothing the motion.

5.1.1 Input data

The first step for data cleaning is to make a selection from the available motion capture data. We are interested in single human motion only from the CMU dataset[1]. In particular we selected the following motions:

- **Locomotion**: running, walking, jumping.
- **Interaction with environment**: climbing, swing, walking on uneven terrain, walking on a path with obstacles.
- **Physical activities**: solo dance, acrobatics, martial arts, kicks, boxing, stretching.
- **Situations and Scenarios**: common behaviors, communication gestures and signals.

[1]http://mocap.cs.cmu.edu/

This collection consists of 2332 different files, which represents 3.9 million frames. These files are in a biovision hierarchy character animation format [2].

5.1.2 Marker positioning

A crucial step in the methodology is the selection and position of the markers. This section encompasses some pre-processing techniques such as normalization, feature extraction and transformation. We overview the marker positioning algorithm in the first place, then we provide further details about the marker sets and parameters.

Technical approach for markers placement

We reproduced the framework from the pre-processing section of Holden et al. [Hol18] to our subject matter. Before generating markers, we need to scale the motion data. Our framework deals with character motion of uniform height and potentially different proportions. Hence, we compute a scaling factor s from the T-pose, using the average lengths of the character's joints. We also keep information about the joint hierarchy. These parameters are kept in a metadata file that is saved and kept for motion reconstruction.

In our dataset, a motion data file defines the motion of a single character with j joints and n poses. Using the rotation and translation component of the joints stored in the motion files, we compute the global homogeneous transformation matrices of all joints for every pose and store it in $Y \in \mathbb{R}^{n\times j\times 3\times 4}$. Then we generate m markers around the *selected* joints for each frame following a normal probability density function $\mathcal{N}(\mu, \sigma^2)$. This local marker configuration is represented by a matrix $Z \in \mathbb{R}^{n\times j\times m\times 3}$. We also set the skinning weights for the marker offsets $w \in \mathbb{R}^{m\times j}$. These weights define the marker-to-joint association. In principle, a marker can be place perfectly on a joint (trivial skinning weight), or show movements of more than one joint if placed further away. Altogether, we can now compute the weighted local offset $\hat{Z} \in \mathbb{R}^{n\times m\times 3}$ for markers, such that we only carry information about the offset of the markers and their skinned joints. It is simply

[2] https://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html

computed,

$$\hat{Z} = \sum_{i=0}^{j} w_i \odot Z_i, \tag{5.1}$$

where $\odot$ represents a component-wise multiplication.

In order to train our network, we must compute the set of global marker positions $X \in \mathbb{R}^{n \times m \times 3}$ using the linear blend skinning function defined as

$$LBS(Y, Z) = \sum_{i=0}^{j} w_i \odot (Y_i \otimes Z_i), \tag{5.2}$$

where $\otimes$ represents the homogeneous transformation matrix multiplication of each of the marker offset in Z_i by Y_i for each of the m markers.

Moreover, we transform the poses into a local reference space to standardize the data for training. We take the root position (hips) as reference for each frame and compute the local frames as

$$Y^* = \frac{F^{-1}}{s} \otimes Y, \tag{5.3}$$

where s is the scaling factor and $F \in \mathbb{R}^{n \times 3 \times 4}$ is the reference frames matrix.

Reduced Marker sets

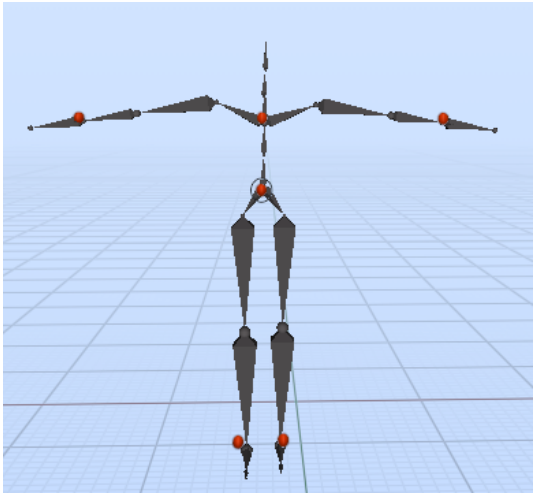
We conduct four sets of experiments, each with a different configuration of markers. For each of these experiments we decide to set trivial skinning weights for the markers' offset and generate our own marker positions according to a normal distribution $\mathcal{N}(0, 0.1)$ around the selected joints. These parameters deal with data corruption. In this configuration, the markers will be positioned randomly and close to their assigned joints, in a 3D sphere of small radius for minimal mismatch and corruption. The four sets of markers are display in Table 5.1 below.

The baseline of our algorithm is computed on the full set with 32 markers (Set A). This is the standard setting for motion capture. The Set B from the above table is motivated by Liu et al. [LZWM06], the 6 markers are the most stable under PCA and thus, carry the

Set ID	Number of markers m	Configuration
Set A	32	On all the joints.
Set B	6 from Liu et al. [LZWM06]	On the head, the right forearm, the left arm, the right leg and on the left and right toes.
Set C	6 Symmetrical	On the left and right hands, the left and right feet, on the hips and on the neck.
Set D	2	On the head and the hips.

Table 5.1: Marker set configurations.

Figure 5.2: Marker configuration of Set C: 6 markers are placed on the left and right hands, the left and right feet, on the hips and on the neck.



most information about the other ones. The hypothesis is that this configuration should score the highest on our machine learning model. We try to challenge this result by adding Set C with 6 markers positioned symmetrically on the body. This is motivated by the human gait that is often assumed to be symmetrical with right and left sides performing identical motions, and was studied in Hsiao-Wecksler et al. [HWPR+10]. According to this plane of symmetry, we place the markers in the upper and lower body where the contribution of each limb to propulsion and control tasks is predominant (legs, arms, hips, feet), from Saghedi et al. [SAPL00]. This setting is displayed in Figure 5.2. Finally, we push the limits of our algorithm by placing only two central markers (Set D).

5.2 Model evaluation and training

In this section we will detail the structure of the network as well as two different validation techniques. For the training process, we use the Cedar[3] cluster, available through Compute Canada, and trained on 2 GPUs with 6 CPU cores per GPU and 64GB per node.

5.2.1 Model validation techniques

For data-driven prediction tasks, One wants to estimate how accurately a predictive model will perform in practice. To this end, we often split our data into a train and a test set: the training set is used to prepare or train the model and the test set used to evaluate the final model. In our experiments, we choose a 80:20 split for the training/testing ratio. In order to tune parameters and add regularization constraints during the training, we also implement a validation set. It provides an unbiased evaluation of the model fit on the training dataset. We explore different ways of partitioning our dataset. In this way we want to evaluate the performance of validation model for classification tasks on MoCap data.

We first explore k-fold cross-validation to partition our data. This technique systematically splits the data into k groups, each given a chance to be a held out model. The goal of cross-validation is to test the model's ability to predict new data that was not used in the estimation. It gives an insight on how well the model generalizes to an independent dataset for instance.

First, we shuffle randomly the observations, and split the dataset into k groups (we used $k = 10$). For each iteration, a group is kept as a hold out or test data set, and the remaining groups compose the training set. The evaluation score of the iteration is kept and we move on the next iteration. At the end of all the iterations, we estimate the accuracy of your machine learning model by averaging the scores derived in all the k cases of cross validation. This procedure is detailed in Figure 5.3. We can see that each

[3]https://docs.computecanada.ca/wiki/Cedar

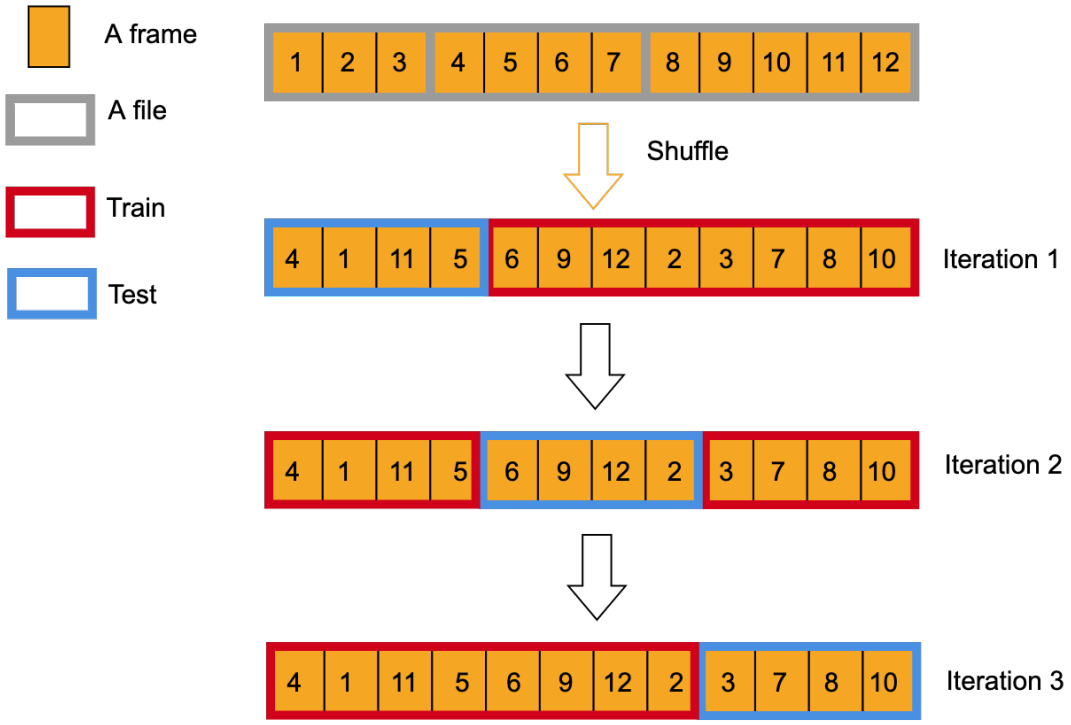


Figure 5.3: Diagram of k-fold cross-validation when n = 12 observations and k = 3. After shuffling, a total of 3 models will be trained and tested.

sample is used in the hold out set 1 time and used to train the model $k - 1$ times. In our first model architecture we used $k = 10$.

A main observation is that k-fold cross validation only yields meaningful results when the structure of the system being studied does not evolve over time. The order of the data is important in this case, and ignoring it can introduce systematic differences between the training and validation sets. Hence, we introduce a second approach inspired from backtesting for time series forecasting. Its goal is to make future predictions from previous time-related data. In the case of time series data, fast methods like exhaustive k-fold cross validation do not work. The time dimension of observations means that we cannot randomly split them into groups.

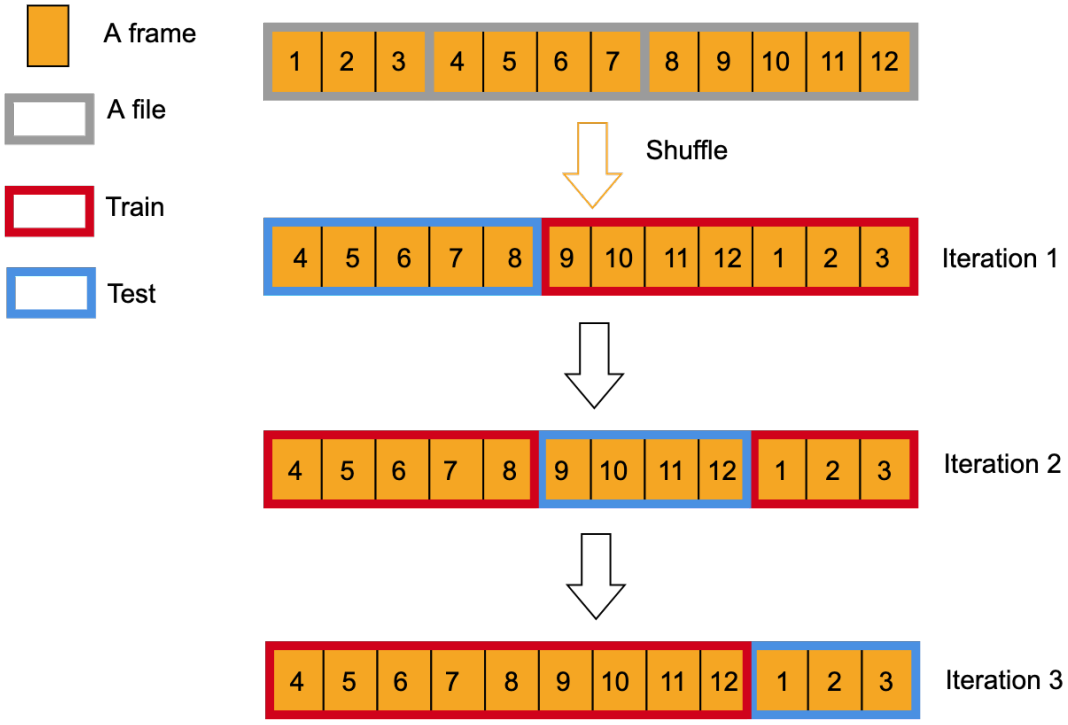


Figure 5.4: A Train-Test split that respect temporal order of observations, when n = 3 observations and k = 3. After shuffling, a total of 3 models will be trained and tested.

In our method, we will consider each motion file as a time-related observation and we will not split. Hence, each motion file will belong to one set only: training, validation or testing (see Figure 5.4). We shuffle the test files without changing the temporal order of the frames inside them. Then, we apply the previous method with the number of observations being the number of files instead of the number of frames ($k = \#\text{files}$), and the width of the training/testing sets is the number of frames in the files. In this way, the train-test split respects temporal order of observations.

Algorithm 1: One iteration of training. It takes as input a mini-batch of n poses Y and updates the network weights Θ .

Input: $Y \in \mathbb{R}^{n \times j \times 3 \times 4}$, $F \in \mathbb{R}^{n \times 3 \times 4}$
Output: $\hat{Y} \in \mathbb{R}^{n \times j \times 3 \times 4}$
Function `Train(`Y, F`)`**:**

$Y^* \longleftarrow \frac{F^{-1}}{s} \otimes Y$

$\hat{Z} \longleftarrow \sum_{i=0}^{j} w_i \odot Z_i$

$X \longleftarrow LBS(Y, Z)$

$\hat{X} \longleftarrow \frac{(X - x^{\mu})}{x^{\sigma}}$ `// Normalization`

$\hat{Z} \longleftarrow \frac{(Z - z^{\mu})}{z^{\sigma}}$ `// Normalization`

$\hat{Y} \longleftarrow Resnet([\hat{X}\hat{Z}]; \Delta)$ `// concatenation then input to the Residual Network`

$\hat{Y} \longleftarrow (\hat{Y} \odot y^{\sigma}) + y^{\mu}$ `// Denormalization`

$loss \longleftarrow l_1(\hat{Y}, Y)$ `// Loss`

$\Theta \longleftarrow Nesterov(\Theta, \nabla loss)$ `// Weight update`

return $\hat{Y}$

End Function

5.2.2 Network structure

The model is trained using a supervised learning method, we implemented a residual network with 6 dense ReLU layers (see Figure 5.5). The network takes as an input a batch of n frames of marker positions $m \in \mathbb{R}^{n \times m \times 3}$ and the associated pre-weighted marker configuration $\hat{Z} \in \mathbb{R}^{n \times m \times 3}$. The output of the network is the corresponding batch of n joint transform $\hat{Y}$. We used the Keras (Tensorflow) backend with its build in weighted accuracy metric. We also use the l_1 norm as loss metric,

$$\sum_{n=1}^{j} |y_n - \hat{y}_n| w_n, \tag{5.4}$$

where w is the skinning weight of the marker at that joint position (in our case it is either 1 if there is a marker associated to that joint or 0), $\hat{y}_n \in \hat{Y}$ is the prediction and $y_n \in Y$ is the

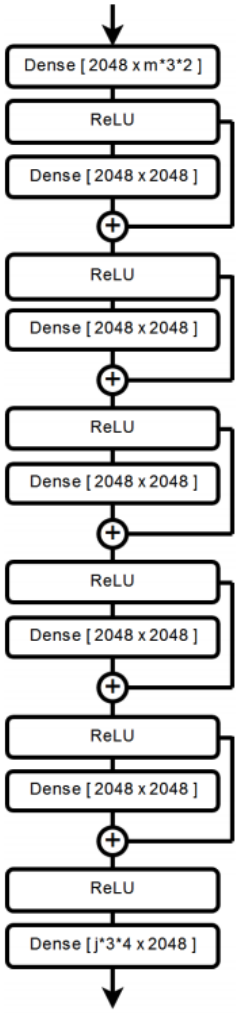


Figure 5.5: Diagram of the ResNet layers, from Holden et al. [Hol18].

true value. The full network pipeline is adapted from Holden et al. [Hol18] is described in Algorithm 1.

We trained the network for 200 epochs. Recent studies form Keskar et al. [KS17] and Fetterman et al. [FKA19] show that adaptive optimization methods tend to perform well in the initial portion of training but are outperformed by stochastic gradient descent (often abbreviated SGD) at later stages. Hence, we use the Amsgrad version of the Adam optimizer (see Reddi et al. [RKK19]) for the first 150 epochs, and SGD for the remaining of the training. Since we use SGD in the last steps of the training, we update the weights of the network using the Nesterov momentum method from Sutskever et al. [SMDH13]. Fi-

nally, since our data is of high-dimenstionality, we decide to use L_2 regularization for the Kernel and Bias regularizer. Once the training is over, we store the weights for prediction.

5.3 Post-processing

Once we have trained our network on the 3.9 million frames and saved the network training weights, we can finally do motion estimation, that is test the network for new data. This is essentially the pipeline previously shown in Figure 5.1. The first step is to estimate the global joint transform from new global marker positions. We get the global marker positions by either recording a motion in a studio or by automatically generating them using the previous marker positioning method. We opt for the latter and we generate markers on .bvh files from the CMU Dataset that are not part of the training nor validation sets. When this estimation step is done we are left with the predicted global joint transform for the new markers.

5.3.1 Motion reconstruction

The important step in our process is to rebuild the local kinematic chain, and have final brand new BVH file. The joint positions and rotations are in the world (global) frame after training, however we are interested in their local position an rotation to reconstruct a full skeleton. In this section we detail the algorithm that helps us rebuild the joint hierarchy from the global joint transforms.

First, we orthogonalize the rotational parts of the joint transforms using singular value decomposition. Moreover, recalling the markers positioning method from Section 5.1.2, we keep the metadata information about the joint hierarchy of the .bvh file, and can easily map the joints to their position in the hierarchy, see Figure 5.6. This step helps us define the joints hierarchy. In this way we know the joints parents and position in the skeleton. We then rebuild the kinematics chain from the end of the skeleton (toes and hands) to

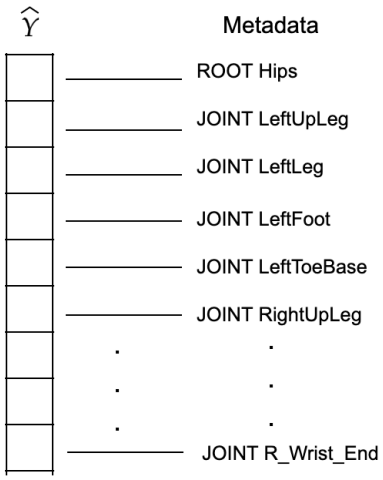


Figure 5.6: Simplified diagram of the joint mapping.

the root. We pass the orthogonalized global data to a Jacobian inverse-kinematics based retargeting solution to extract the local joint transforms.

5.3.2 Temporal smoothing

The last step is essentially smoothing the motion in order to remove any residual high frequency noise or jitter from the output. We used the Stravinsky-Golay algorithm from Press et al. [PT90] to improve the precision of the data without distorting the signal tendency. The filter fits a degree three polynomial. Thus, we assume locally that the data have a jerk of zero, which is a good prior for our motion capture data Flash et al. [FH85]. The filter is applied separately to each dimension of the joints rotation. Finally, we use Blender[4] as software to visualize the final motion. It is a free open source software.

[4]https://www.blender.org/

Chapter 6

Results and discussion

This chapter highlights our most important findings on how efficient both pre-processing methods are on a large dataset and on how robust our method is to reconstruct full body motion from reduced marker sets. We first introduce the results for the k-fold cross-validation technique then we compare them with the second validation method. On all experiments, the preliminary step is to tune the hyper-parameters then to train the ResNet on 4 marker configurations: 32 markers, 6 symmetrical markers, 6 markers from Liu et al. [LZWM06] and 2 markers (in Table 5.1). At the end of this section we will have estimated the adequate validation technique for training motion data on a ResNet and the optimal reduced marker sets.

6.1 K-fold cross-validation

We conduct three experiments with k-fold cross-validation. We display the training results using loss and accuracy graphs and we summarize the fine-tuning process in Table 6.1. Then, we predict a new motion with the trained weights and compare the movement accuracy with the training accuracy.

Table 6.1: Hyper-parameter fine-tuning for the CMU dataset with k-fold cross-validation.

Experiment	Regularizer	Optimizer Learning Rate (Adam)	Optimizer Decay Rate (Adam)
1	10^{-4}	0.001	10^{-3}
2	10^{-4}	0.005	10^{-3}
3	10^{-4}	0.005	10^{-6}
4	10^{-6}	0.001	10^{-4}

6.1.1 Hyperparameters tuning

The best hyper-parameter combination is found using a grid-search algorithm. We will mainly focus on the regularization term and the Adam optimizer learning rate and decay. We do not change the SGD learning rate and decay, they are respectively 0.001 and 10^{-6}. We start fine-tuning with simple parameters used in Holden et al. [Hol18] and with a regularization parameter of 10^{-4}; from this point we first tune the regularization then the optimizer learning rate and decay. We show a subset of the exhaustive search in Table 6.1 and the optimization curves Figure 6.1.

The errors are plotted in Figure 6.1 where training is done until 200 epochs. We can quickly discard Experiment 2; with this learning rate and decay combination for the Adam optimizer, the network fails to reach a global minima at that stage. Hence, when switching to SGD at 150 epochs, the loss curve drastically decreases and continues to decrease at the end of the plot. This indicates that the training is not done by the end of the 200 epochs.

In contrast, Experiment 1 seems to be stuck at a global minima from the start. There is a slight improvement in Experiment 3. However, the loss curve indicates that the training loss essentially converges after just ten epochs. In order to understand the underlying potential problems, we also plot the loss and accuracy curves of the other marker sets, see Figures 1, 2, 3 in the Appendix. The network does not generalize well to new data, and this happens for all marker subsets. Whereas both the validation and Training accuracy increase over time, the validation accuracy curve is unstable and jumps. This result describes a random guess for the output. Moreover, the accuracy and training loss curves

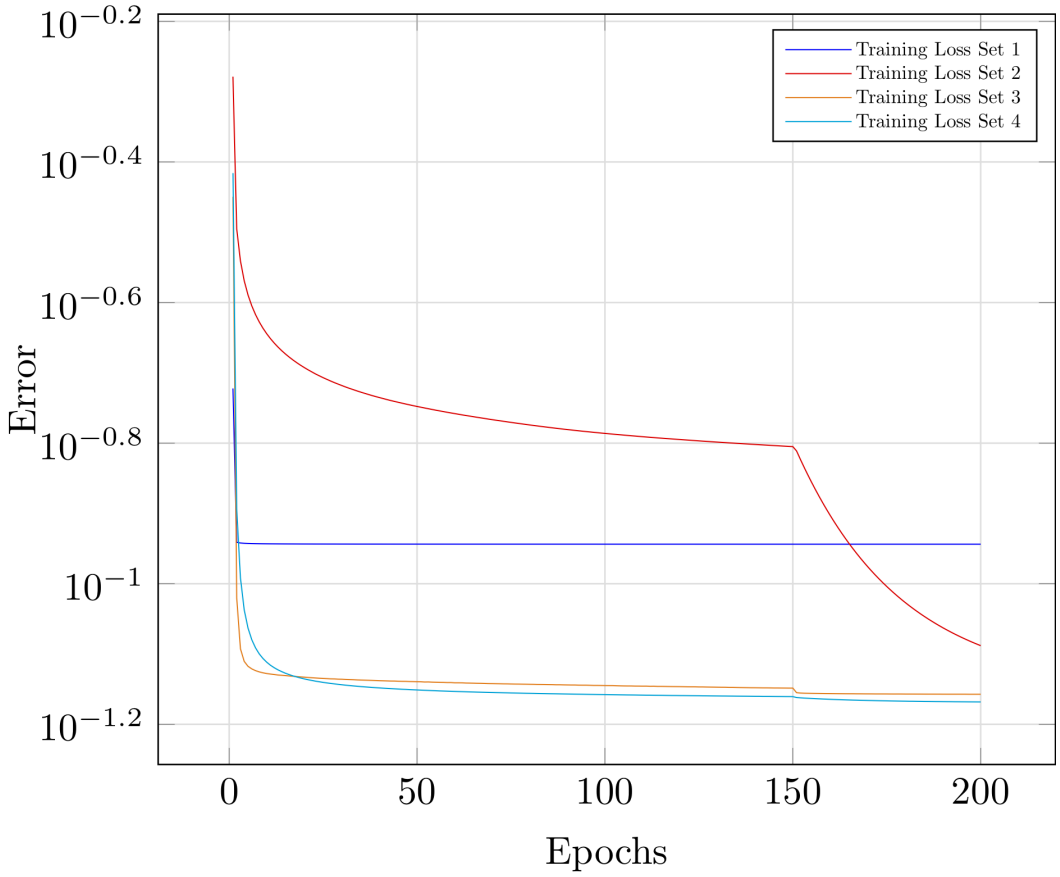


Figure 6.1: K-fold cross-validaiton: residual network optimization curves for the baseline marker set (32 markers).

overlap. In fact, the validation loss is slightly lower than the training loss. This suggests that the model remains incapable of learning the distribution or that the model learns the underlying distribution extremely fast. The latter is potentially due to the sampling procedures that is inaccurate and that duplicate observations exist in the training and validation datasets.

Whereas the other learning curves, the training loss plot of Experiment 4 decreases to a point of stability after 180 epochs, this is the optimal fit learning curve. Hence, the optimized model and hyperparameters are picked from this last experiment. For k-fold cross-validation best settings are a L2 regularization of 10^{-6} and a learning rate of 0.001 for the Adam with 10^{-4} decay steps. The SGD optimizer's optimal parameter are a learning rate of 0.001 with decay steps of 10^{-6} and a 0.9 momentum.

Table 6.2: Training and Validation scores for the model trained with k-fold cross-validation.

Set ID	Number of markers	Test Score	Validation Score
A	32 markers	93.24%	93.21%
B	6 markers from	89.31%	89.31%
C	6 symmetrical markers	92.67%	92.75%
D	2 markers	69.07%	69.04%

6.1.2 Training result

We now have the best hyperparameters configuration for training our network. Under these settings we are interested in knowing how well each marker set performs. The validation and training scores are displays in Table 6.2. Our Resnet achieves the highest score on the complete set of markers, with a 93.21% accuracy. This comes close to the Set C with a 92.75% accuracy.

Nonetheless, one may notice from Figure 6.2 that the model tends to overfit on all marker sets. Even if the jumps in the accuracy curves could be due to the mini batch iteration, the curves tend to overlap and the validation curve scores higher than the training curve on average. Furthermore, the network completely fails to learn the underlying distribution and patterns when we only use two markers, suggesting that two markers are not enough to generalize a full body motion.

6.1.3 Motion simulation

In this section we aim at challenging the previous results by evaluating the network performance on new data and how this transcribes on a 3D motion. We decide to predict a new motion with the same marker set configuration than the one scoring the highest during the training step. Hence we place 32 markers and pick a simple walk motion with a $90°$ turn to the right.

The results show that the network does not generalize to new data. From Figure 6.3 we can see that the original motion is a smooth $90°$ turn to the right, while the predicted

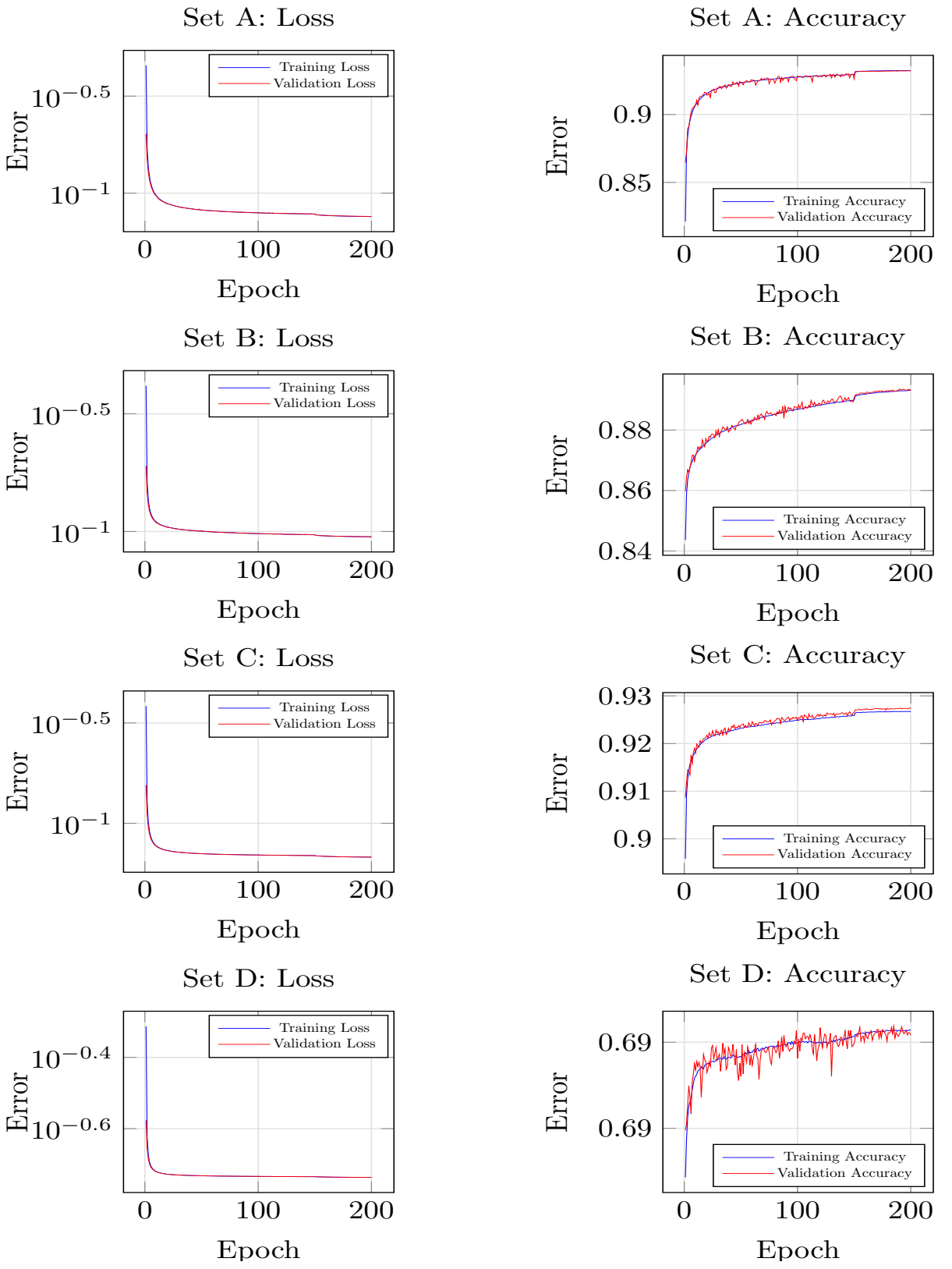


Figure 6.2: Results for the CMU dataset with k-fold cross-validation. Set A: 32 markers, Set B: 6 markers from Liu et al. [LZWM06], Set C: 6 symmetrical markers, Set D: 2 markers.

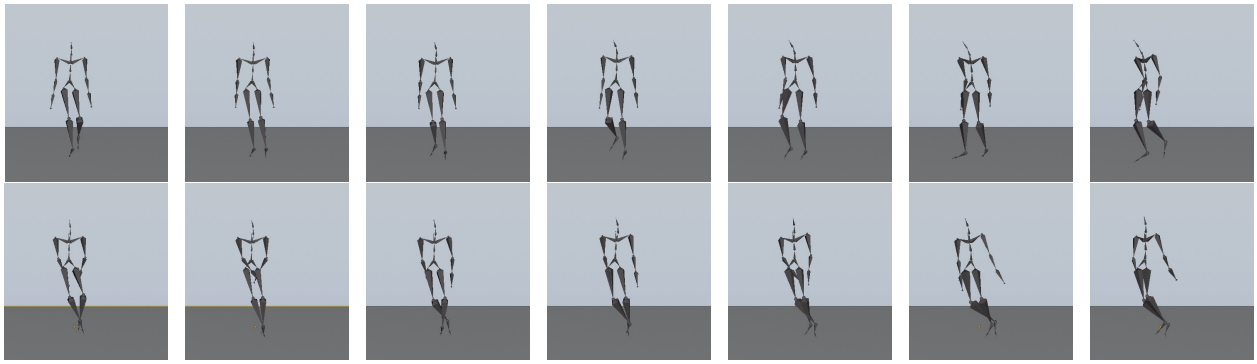


Figure 6.3: Comparison of the prediction and ground truth motion. The 7 matching frames are taken 2 s apart. Top: Ground truth motion. Bottom: Motion predicted by our network on the marker set that scores the highest (32 markers) trained with k-fold cross-validation.

motion is a stiff turn without leg motion. The character fails to move its knees and only the hip's position and rotation is learned and reconstructed. The positions and rotations of both arms are also inaccurate. This observation is also verified by the rotation curves of the character's left knees in Figure 6.4.

Additionally, the motion curve of the left knee rotation jumps and is not smooth, even after the Savitzy-Golay filter. In this way, We can also infer that the predicted joint rotates at the same time as the ground truth motion but at different angles. With the previous 93.21% accuracy score on the complete marker set, we would expect the curve to have roughly the same distribution and be smoother. In this case we have two options: either the motion reconstructed step is erroneous or the model is overfitting. Since the inverse kinematic chain has been tested for different test .bvh motions without errors, we are almost certain that the problem comes from the model. More precisely, k-fold cross-validation may not be suited for our dataset. This method underestimates the relation between frames of the same motion file. This assumption is supported by the joint rotation curve in Figure 6.4: the jumps indicates that the model does not learn the relationship between the frames for the same joint and tends to take guesses at each frame. Hence, even if the loss and accuracy curves of our previous results looked correct, the test was inconclusive. In fact, we are far from the potential 93.21% score on the complete

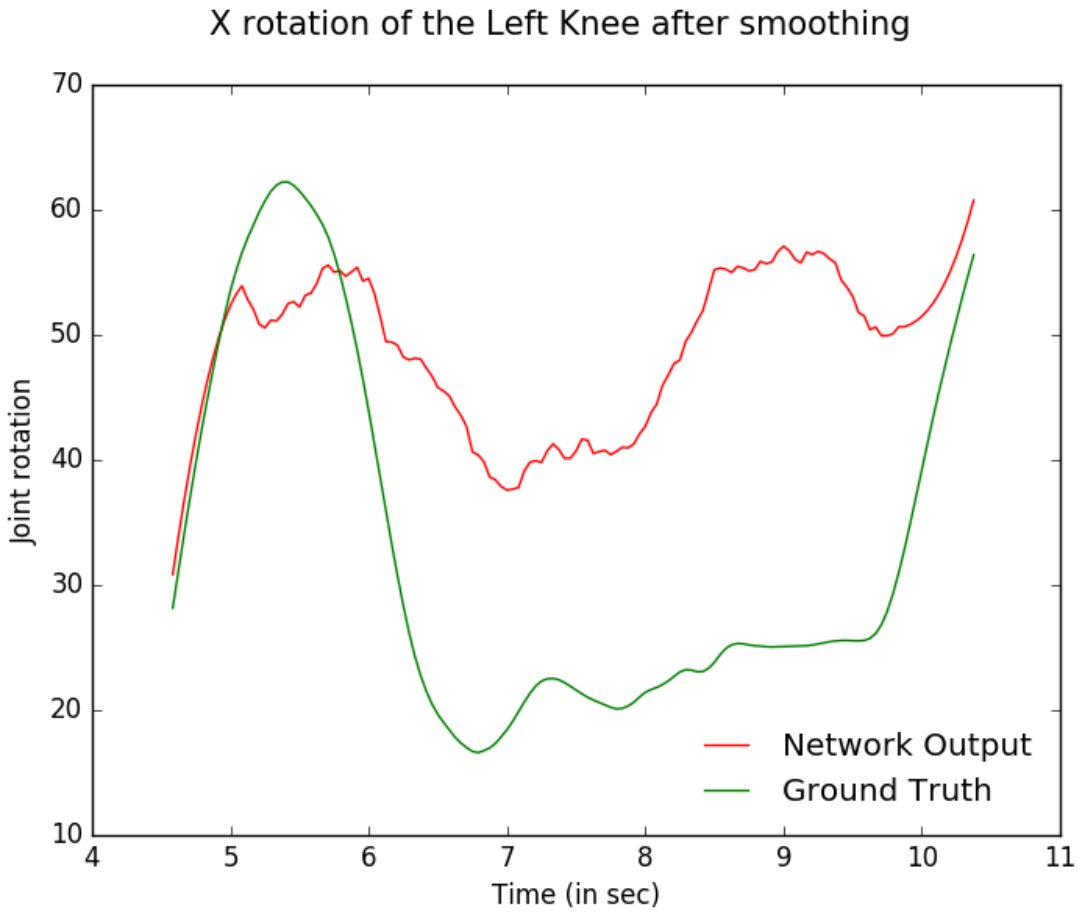


Figure 6.4: X-axis rotation of the left knee joint after Savitzky-Golay smoothing from frame 110 to frame 150. This corresponds to a $90°$ right turn.

marker set. This last observation also question our loss metric, thus we will conduct more experiment in the next seciton.

6.2 Time-related observation

We previously stated that k-fold cross-validation was not a suitable technique for splitting our the frames. In this section, we fix this issue by training our network on a data split that respect the temporal order of the observations, that is when using the individual files as observation. In this way, we aim at predicting smoother joint transforms.

Table 6.3: Hyper-parameter fine-tuning for the CMU dataset with the time-related observation method.

Experiment	Regularizer	Optimizer Learning Rate (Adam)	Optimizer Decay Rate (Adam)
1	10^{-6}	0.001	10^{-4}
2	10^{-4}	0.001	10^{-4}
3	10^{-4}	0.005	10^{-4}

6.2.1 Hyperparameters tuning

We compile the hyper-parameters in Table 6.3 for the time-related data split. The SGD learning and decay rates remain the same, and are respectively 0.001 and 10^{-6}. We also look for the best hyperparameters using a grid-search algorithm. However, we take as starting point the set that performed the highest with k-fold cross validation. We show a subset of the exhaustive search in Table 6.3.

From the optimization curves in Figure 6.5 we can see that the first two experiments have approximately the same plots. The slopes are reaching a local minima after 5 epochs then remain steady. The first two models fail to explore relevant weights in hidden units. This is highlighted in the Figures 4 and 5 in the Appendix.

The training and loss curves of the first experiment are identical, implying that the model overfit. Whereas in Experiment 2, the validation losses start to increase again after few epochs and the generalization gap is large. We simplify the network and reach a global minima by: decreasing the regularization from 10^{-6} to 10^{-4} and starting with a larger learning rate of 0.005. Although the loss error is higher with these hyper-parameters, the plot of training loss for Experient 3 decreases smoothly to a point of stability. We will therefore use these parameters for the training.

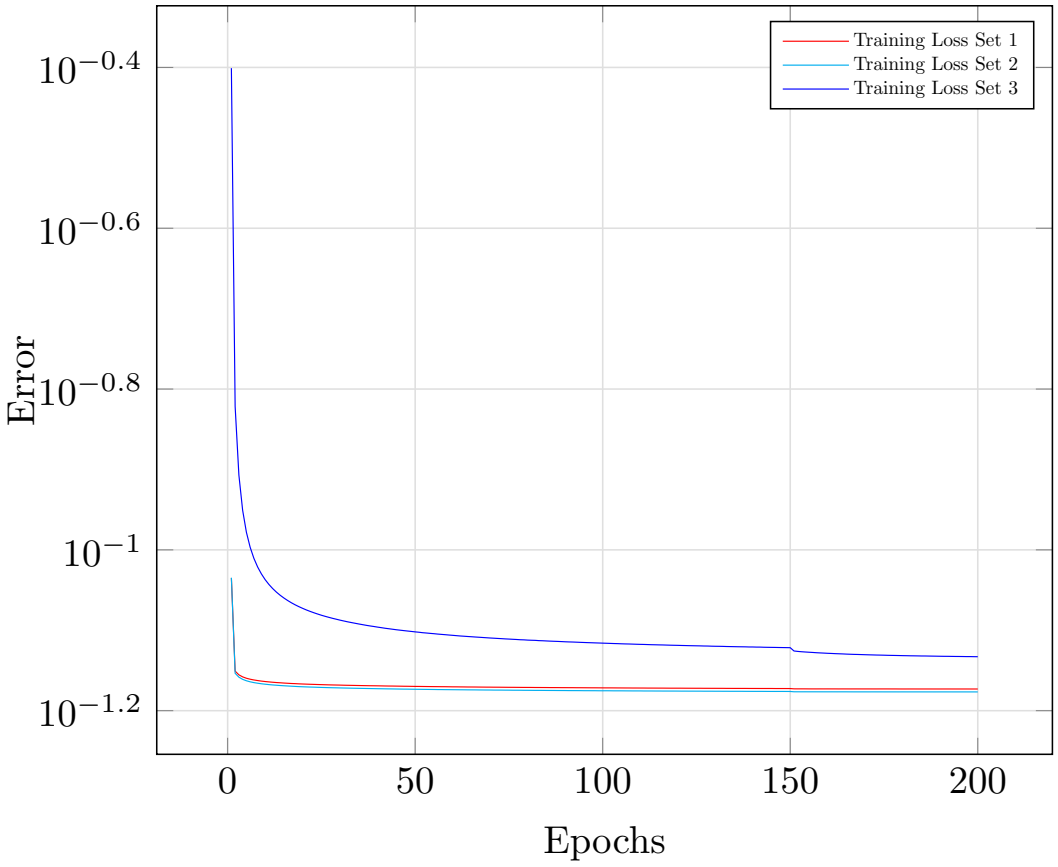


Figure 6.5: Time-Related Observations: residual network optimization curves for the baseline marker set (32 markers).

6.2.2 Training result

The training results are shown in Figure 6.6 on Experiment 3. This time, the network scores the lowest when trained on two markers and the highest when trained on 6 markers placed symmetrically. Nonetheless, all networks tend to converge quickly and remain steady after less than 100 epochs, especially on the validation set. A simple fix would be to add an early stopping callback to stop training once the validation curve has stopped improving.

We also notice a large gap between the validation and training curves for Set A. A large generalization gap is usually a sign of unrepresentative training data, meaning that the validation data may have features not represented in your training data. However, this trend disappears when we reduce the number of markers. Thus, a more plausible

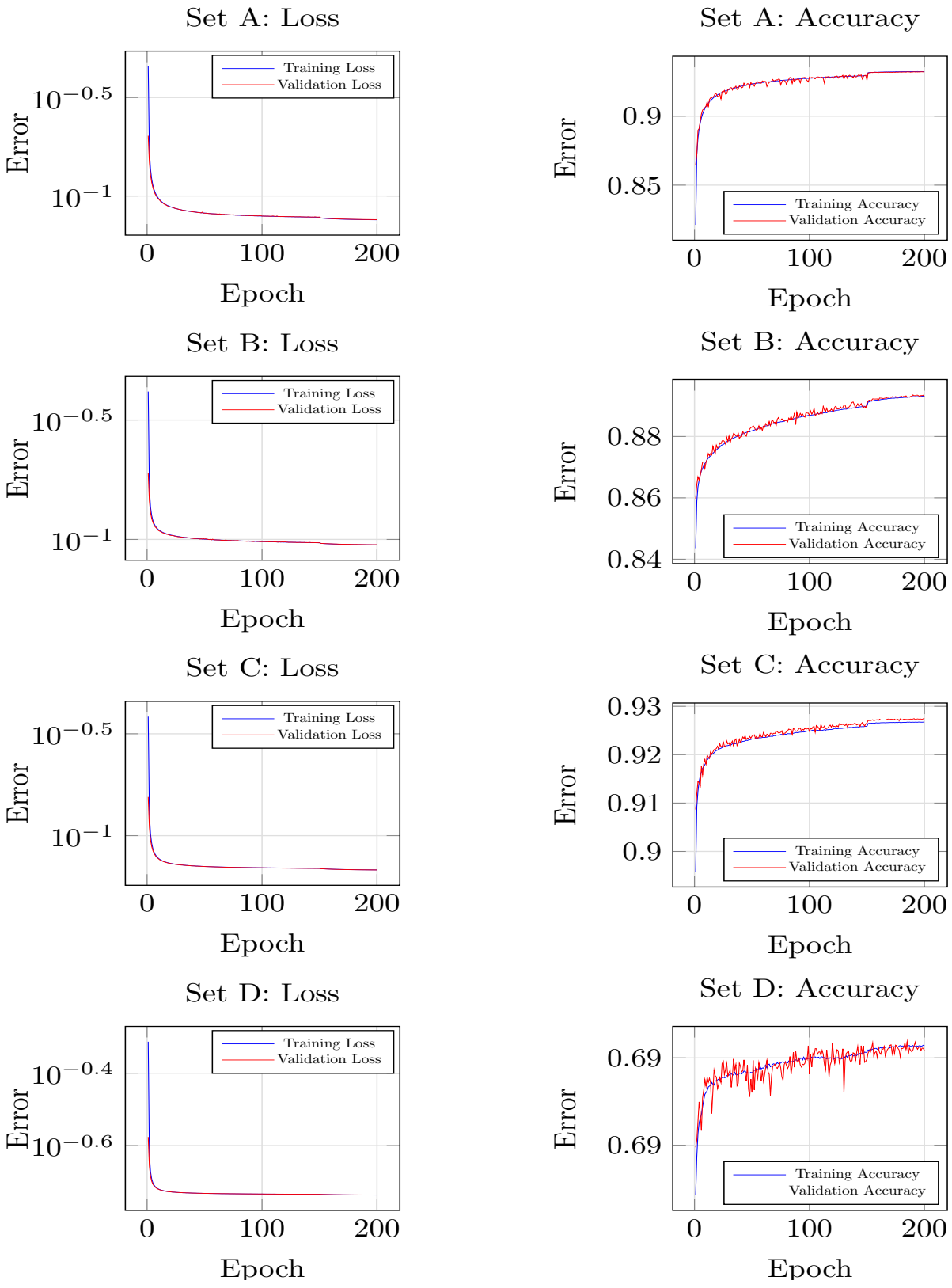


Figure 6.6: Experiment 3: Results for the CMU dataset trained on time related observations (with motion files as observations).

Table 6.4: Training and Validation scores for the model trained with time-related observations.

Set	Number of markers	Test Score	Validation Score
A	32 markers	93.63%	82.29%
B	6 markers from Liu et al. [LZWM06]	89.63%	86.72%
C	6 Symmetrical markers	92.83%	**91.60**%
D	2 markers	69.07%	68.75%

explanation would be that we tend to mis-classify the joints more often when we give a marker per joint. Finally, from Table 6.4 we can see that the set with two markers positioned symmetrically (Set C) on the body scores the highest.

6.2.3 Motion simulation

We previously saw a disconnection between the network output and the motion simulation and reconstruction. Hence, we aim at evaluating how robust this network architecture is. To this end, we are going to predict a new motion for the marker configuration that scored the highest. We select two new motions to predict: the first one is the same walk as used previously in Section 6.1.3, and the second one is a stair climb. From the .bvh motion files we place 6 markers symmetrically, since they are the markers that scored the highest during the training step, and predict the new joint transforms.

The first motion is the same simple walk with $90°$ right turn. This time, the motion is smooth and the joints' transforms match the ground truth motion almost everywhere. We can see from the Figure 6.8 that the two rotation curves of the left knee have the same distribution and almost overlay. The smoothness of the rotation curve indicates that the network successfully learns the time-dependency of a joint through different frames. However, the network still fails at learning movement subtleties: on Figure 6.8 there are small rotations between 7 s and 9 s on the ground truth motion that are not appearing on the network output. This is not coming from the Savitzky-Golay filter, as the raw motion was also not capturing the jumps.

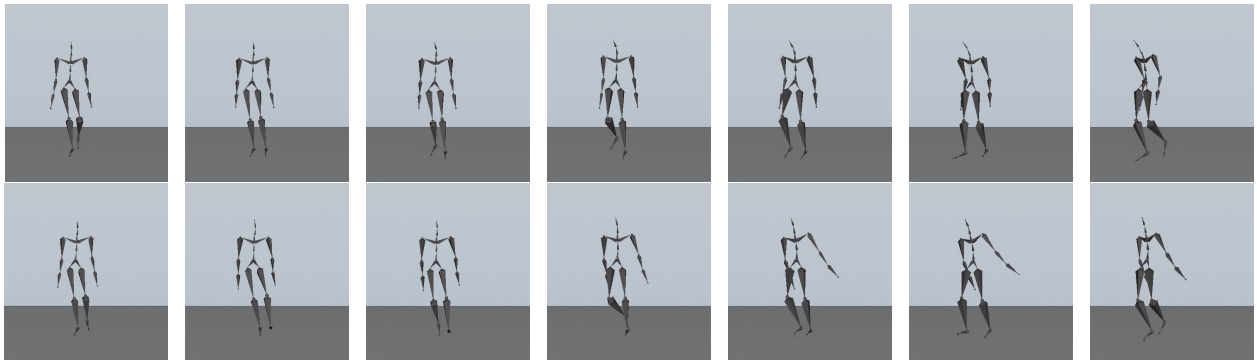


Figure 6.7: Comparison of the prediction and ground truth motion. The 7 matching frames are taken 2 s apart. Top: Ground truth motion. Bottom: Motion predicted by our network on the marker set that scores the highest (32 markers) trained with Time related observaiton.

Finally, from the Figure 6.7 only the character's arms are not matching. This is also observable on the second predicted motion on Figure 6.9 and the associated left arm rotation in Figure 6.10. This result highlights the model limitation. Out of all the limbs, the arms are harder to predict. This can be explained by the size of the bones: the arms, forearms and hands are shorter bones than the legs. Hence, the joint location are closer and the network has a higher chance to mislabel the joint and give erroneous joint transform. Since this model also takes into account the previous frames, the error is propagated through the motion.

6.3 Discussion

In this section we summarize our prominent results and expose the model limitations. Originally, k-fold cross-validation is a reference basis for classification tasks, however it does is not work for learning full body motions. By shuffling the frames we skew the learning process since frames from the same file happen in the training and validation sets. This leads to an immediate overfit and incorrect accuracy curves. We found a solution and remedied the problem by implementing a time related approach to cross-

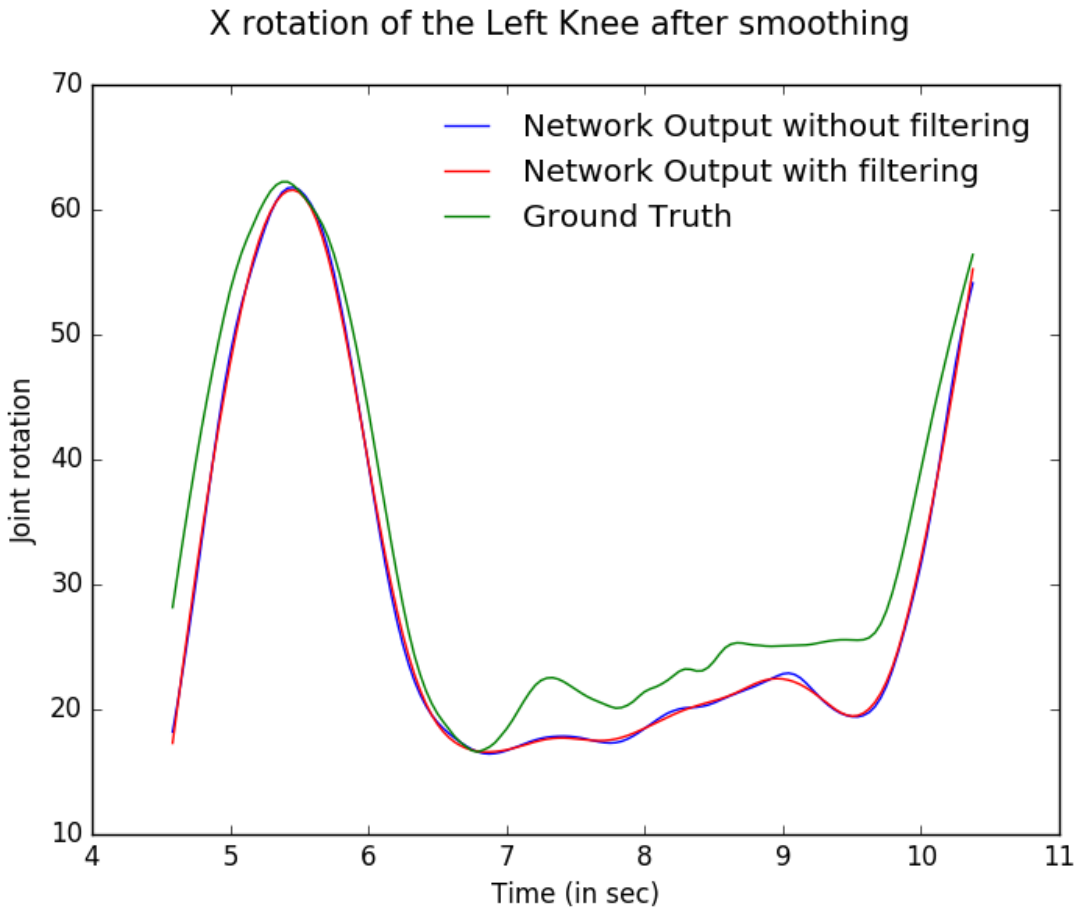


Figure 6.8: Plots showing the x-axis rotation of the left knee joint for the first 200 frames after Savitzky-Golay smoothing.

validation. Training the data with this technique and the residual network showed valuable improvement in the output.

First, since we aim at estimating human motions from a small set of the most informative markers we are interesting in the marker subset with the highest accuracy. The neural network is scoring the highest with 6 markers symmetrically placed and the lowest with two markers. Whereas the latter result was anticipated from the few of information carried with two markers, we find that the hypothesis from Liu et al. [LZWM06] does not hold. In fact, the subset found with Principal Component Analysis scores 86.72%, which is barely higher than our baseline with 32 markers and is significantly lower than the symmetrical marker set that scores 91.60%. Hence, our experimental results consolidate

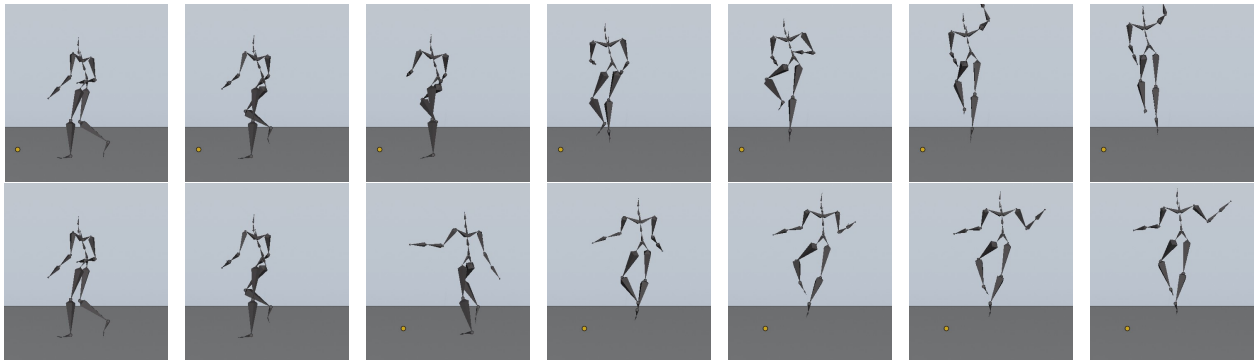


Figure 6.9: Comparison of the prediction and ground truth motion. The 7 matching frames are taken 2 s apart. Top: Stair climb ground truth motion. Bottom: Motion predicted by our network on 6 markers placed symmetrically trained on time related observation.

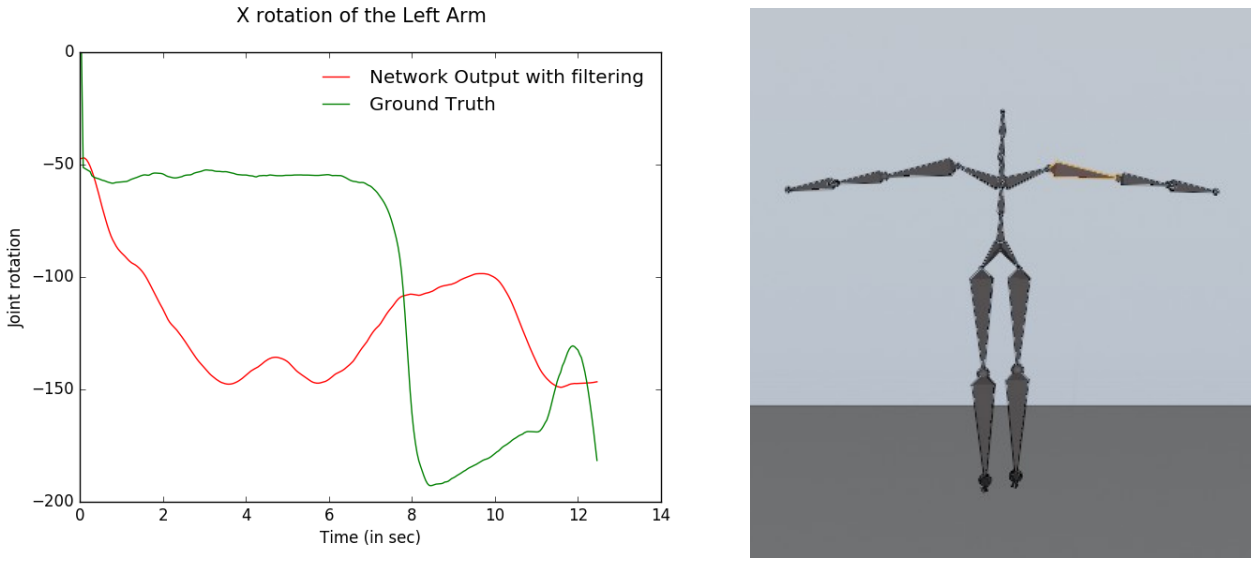


Figure 6.10: Left: X-axis rotation of the left arm joint after Savitzky-Golay. This corresponds to the stairs climb motion. Right: Left Arm position (in orange).

the hypothesis stating that the human gait is symmetrical on average, making the 6 symmetrical markers carrying more information in that setting and when trained on a large dataset. In order to second this result, we have to consider that our dataset is 23.1 times bigger than the dataset used to test PCA and train their model, making our method more robust. We also notice that our model has an unrepresentative training dataset when attempting to learn on 32 markers. This situation was identified by the large gap between

the training and loss curves, despite some improvement over time. Since we are training on time-related observations that has been shuffled and cross validated for this network, a way to fix this would be to simply add more observations. They could be generated in a motion capture lab for example. Additionally, the other loss plots decrease to a point of stability before 200 epochs. A way to improve the model would be to add an early stopping callback to stop training once the validation curve has stopped improving. At this stage, the network requires more fine tuning to reduce the generalization gap. The plots still show a potential overfit.

Secondly, the reconstructed motion was smooth and our model was able to accurately learn joint rotations when trained on 6 markers. For instance, this did not hold with regular cross-validation. Our model is highly effective for walking and leg motions, showing little to no difference with the ground truth motion after smoothing. The reconstructed motions were visually credible and realistic. Nonetheless, the model had difficulty learning arm motion. This issue could potentially arise from two factors: the inverse kinematics or the classification. The joint positions in the output might be too close to each other, leading to mismatches sometimes.

Finally, we are tackling the performance of the model. The pre-processing step takes the most time with an average of 15 h. The training task for time-related observation takes on average 10 h, while the prediction is done in 0.15 ms and the motion reconstruction in 1.9 ms for 500 frames on average. Since we only need to train our model once for every marker set configuration, our model show great potentials for real time motion estimation.

Chapter 7

Conclusion

In this thesis we have presented a complete pipeline for motion prediction with reduced marker sets. We defined proper marker subsets for that task: 6 markers found with PCA, 6 markers placed symmetrically on the body from the symmetrical human gait, a complete marker set as baseline and only 2 markers. Moreover, we have discussed different validation techniques for our machine learning model. The experimental results demonstrate that our method can quickly generate plausible human motions, on a frame-by-frame basis, and scales well with the size of the character and heterogeneity of motions. The ResNet model reaches a 91.60% accuracy when trained on time-related observations and for 6 markers placed symmetrically on the body. This new result challenges the previous literature on reduced markers for motion prediction, especially the work from Liu et al. [LZWM06].

An important take away from this project shows that despite the complexity of motion capture data, deep learning models are nonetheless less vulnerable to perturbations and capable of capturing the inherent temporal relationships of the observations through the frames. It is also observed that, while deep learning is extremely powerful, a lot of work is required for its optimization and adapting it for dataset such as the one used in this project. Below, I propose future directions for this project that I think will build upon the results obtained in this thesis.

7.1 Future work

Both the model and the reconstruction method are areas that would have great potential for future improvements. First, we tackle the limitations of the dataset and the potential network additions or changes. Then, to pinpoint the reason behind the motion reconstruction potential failures, we suggest a different approach to our inverse kinematics algorithm.

7.1.1 Model Improvement

Several model improvements could be made in future extensions. The optimization of the ResNet can be done more thoroughly including using a larger dataset and adding an early stopping callback. As of right now, we have modelled an ideal situation with trivial skinning weight for marker positioning and limited noise, which is still rarely the case. Hyper-parameters fine tuning can also be done more thoroughly and one may add more constraints on the markers' positions. Finally, potential further studies could also evaluate the impact of adding or removing dense layers and implementing this pipeline for real time motion estimation.

One drawback of this approach is the space and computational power required to run the experiments, as the data represent roughly 10 GB of data, and the pre-processing and training steps combined take up to 25 hours. Regarding the dataset, we believe that the CMU dataset provides a complete and diverse training set and is overall very popular in the literature. However, the files are now getting old and use different nomenclatures, coordinate systems and scaling than the new free open-source MoCap datasets like the SFU[1], making it very hard to pre-process together. One may benefit from building a new dataset in a studio or updating the CMU dataset to be compatible to the new motion capture files available on the market.

[1] http://mocap.cs.sfu.ca/

7.1.2 Motion reconstruction

Another limitation of our system is the inverse kinematics pipeline. Our methods is relying on the saved hierarchical information of the BVH files, and we are only exploring the Jacobian inverse-kinematics based retargeting solution. This step is a tedious step in post-processing. This project could highly benefit the use of a more advanced and adapting Inverse Kinematics (IK) solver such as the one found in animation softwares like Maya[2].

7.2 Closing remarks

This project has shown that a machine learning solution offer very promising results for the future of human motion estimation. This project is part of a technological continuity that is already implemented in fields such as biomechanics, medical imaging, the automated car, the film and video games industry, live streamed event etc. The power of the Residual Network reside in its ability to robustly predict human motion. We believe that further studies would be able to predict, create and anticipate future human movements from such reduced marker sets.

[2]https://www.autodesk.ca/en/products/maya

Appendices

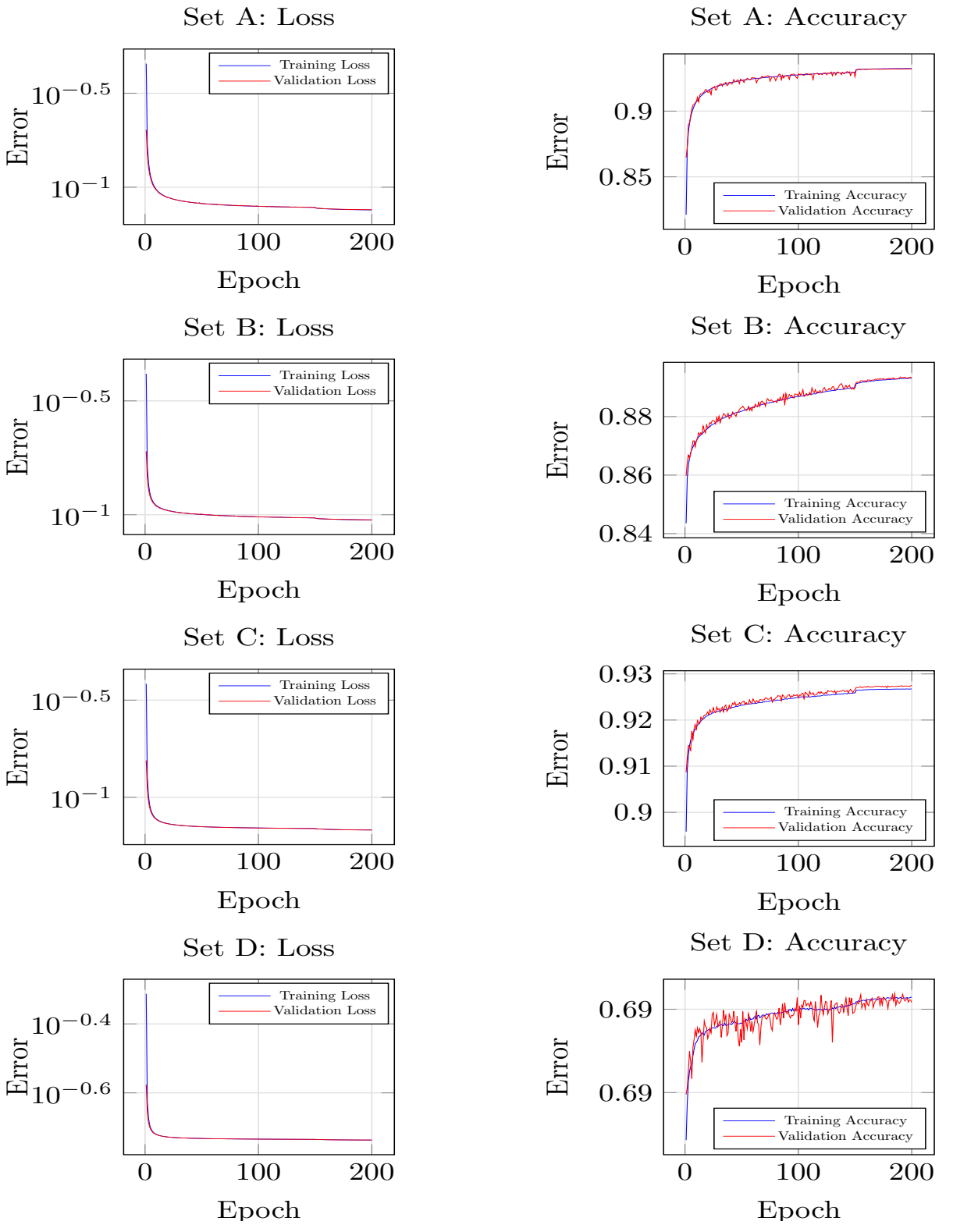


Figure 1: Experiment 1: Results for the CMU dataset with k-fold cross-validation, with motion frames as observations. Set A: 32 markers, Set B: 6 markers from Liu et al. [LZWM06], Set C: 6 symmetrical markers, Set D: 2 markers.

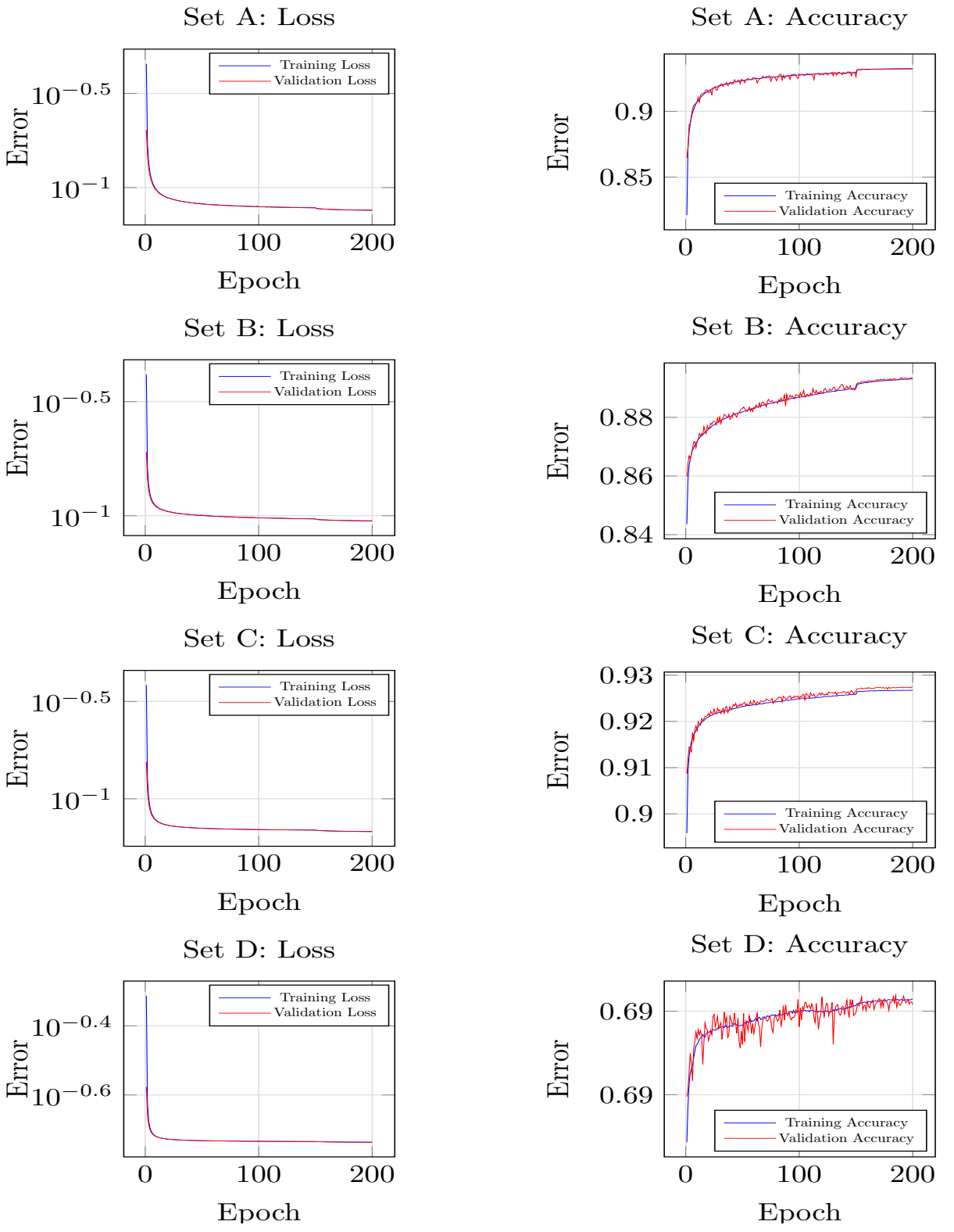


Figure 2: Experiment 2: Results for the CMU dataset with k-fold cross-validation,with motion frames as observations. Set A: 32 markers, Set B: 6 markers from Liu et al. [LZWM06], Set C: 6 symmetrical markers, Set D: 2 markers.

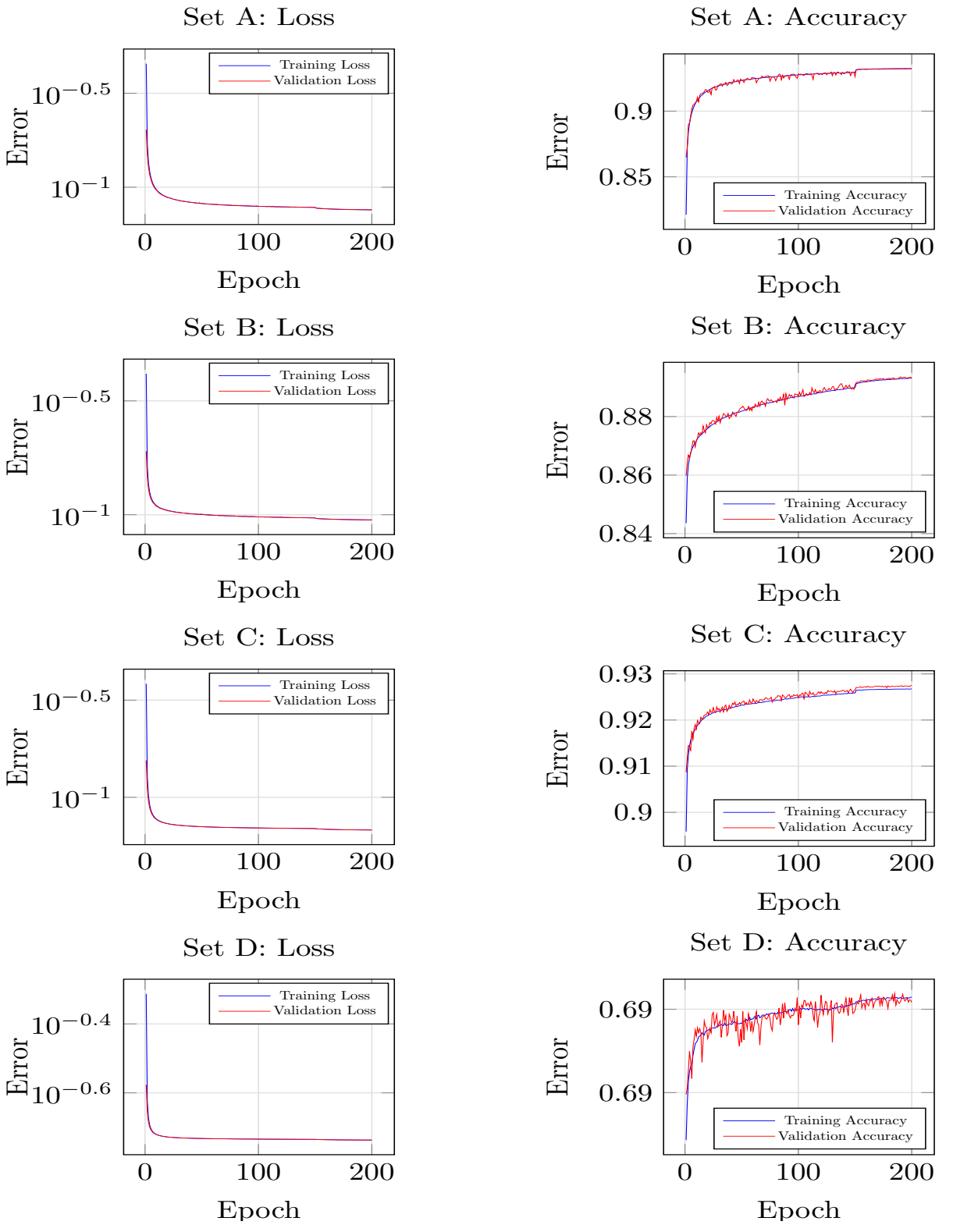


Figure 3: Experiment 3: Results for the CMU dataset with k-fold cross-validation, with motion frames as observations. Set A: 32 markers, Set B: 6 markers from Liu et al. [LZWM06], Set C: 6 symmetrical markers, Set D: 2 markers.

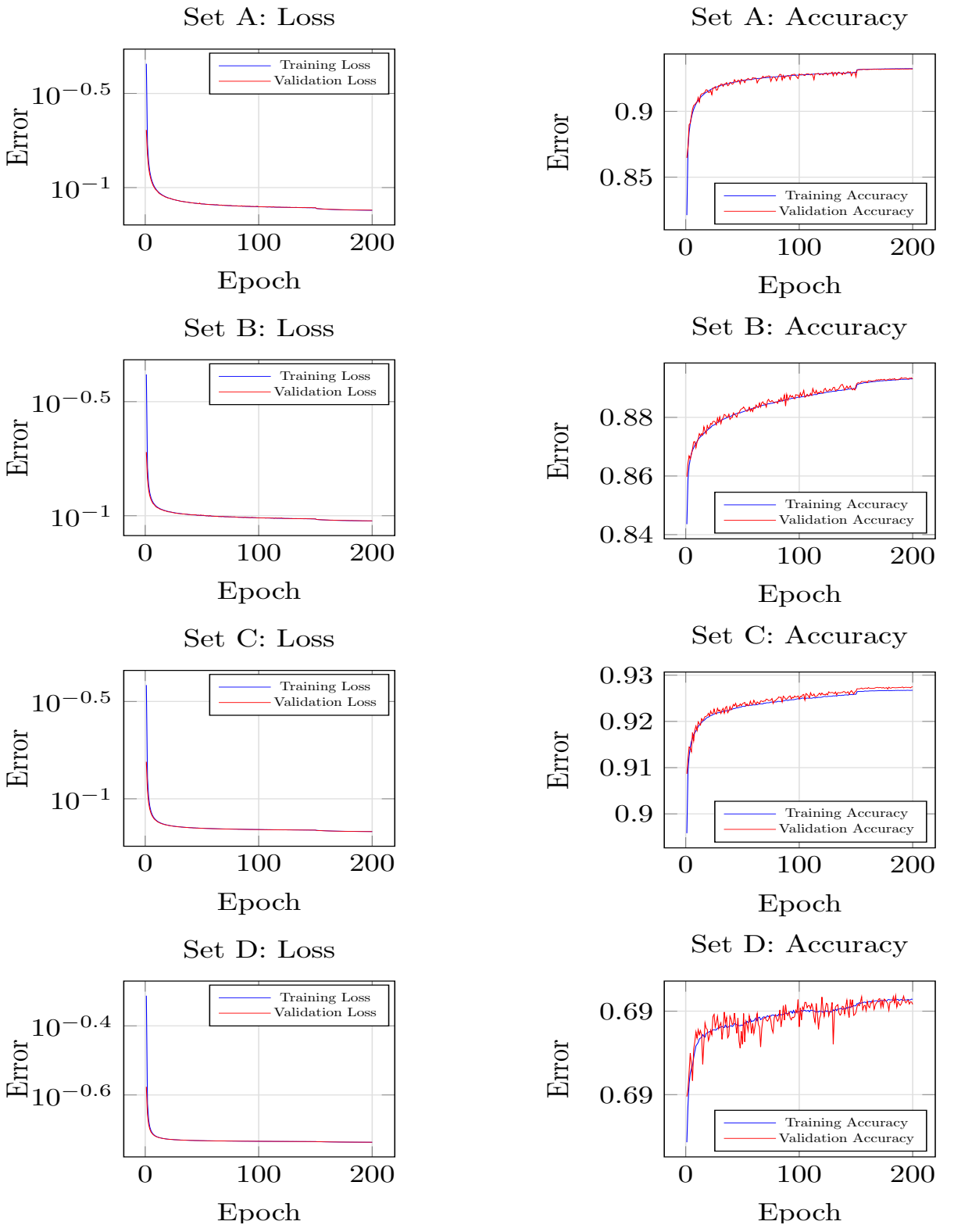


Figure 4: Experiment 1: Results for the CMU dataset trained on time related observations (with files as observations). Set A: 32 markers, Set B: 6 markers from Liu et al. [LZWM06], Set C: 6 symmetrical markers, Set D: 2 markers.

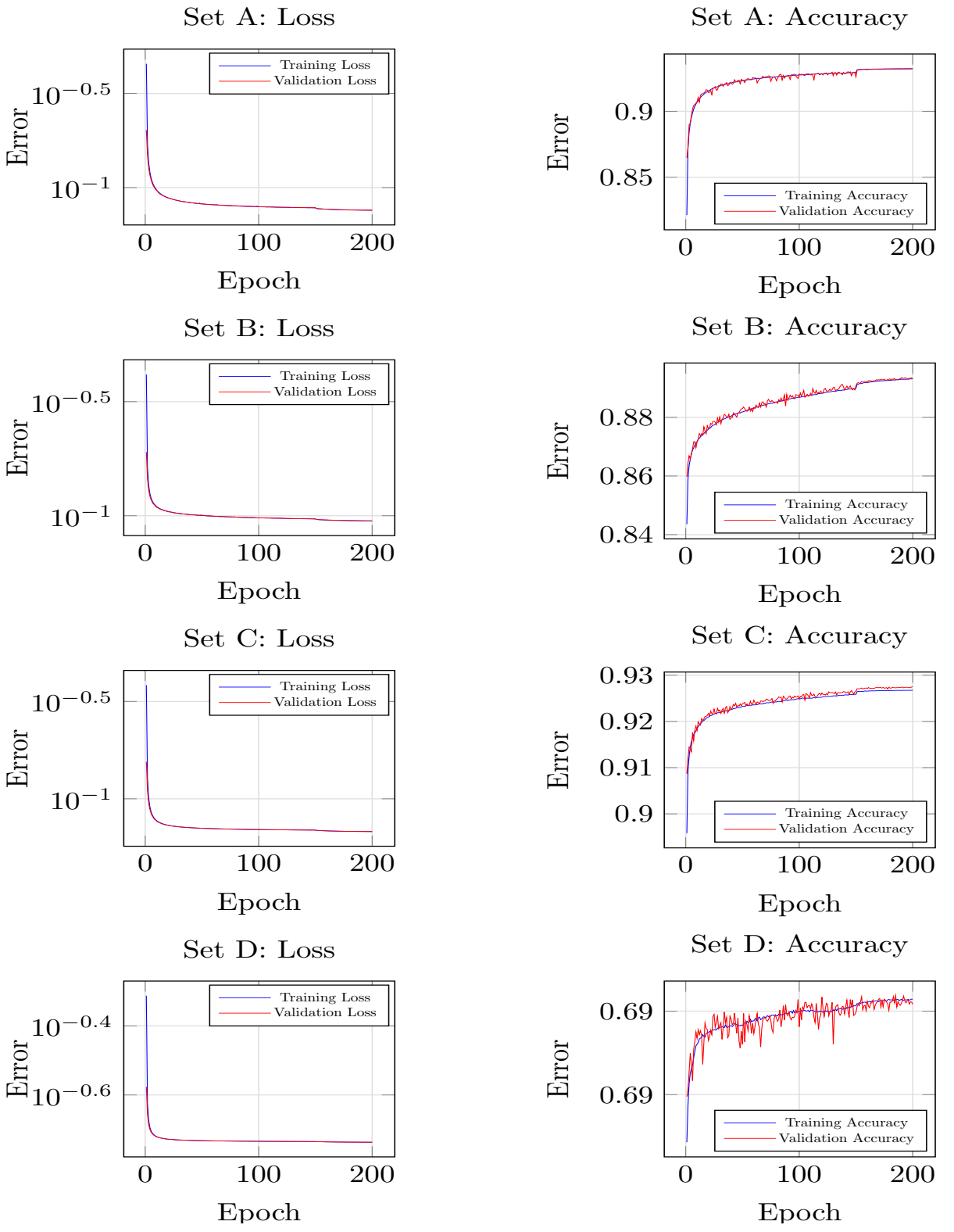


Figure 5: Experiment 2: Results for the CMU dataset trained on time related observations (with files as observations). Set A: 32 markers, Set B: 6 markers from Liu et al. [LZWM06], Set C: 6 symmetrical markers, Set D: 2 markers.

Bibliography

[AK12] Rami Aladdin and Paul Kry. Static pose reconstruction with an instrumented bouldering wall. In *Proceedings of the 18th ACM symposium on Virtual reality software and technology*, pages 177–184, 2012.

[AVT+17] Varghese Alex, Kiran Vaidhya, Subramaniam Thirunavukkarasu, Chandrasekharan Kesavadas, and Ganapathy Krishnamurthi. Semisupervised learning using denoising autoencoders for brain lesion detection and segmentation. *Journal of Medical Imaging*, 4(4):041311, 2017.

[CXZ+02] Ira Cohen, Qi Tian Xiang, Sean Zhou, Xiang Sean, Zhou Thomas, and Thomas S Huang. Feature selection using principal feature analysis. 2002.

[DMV+19] Nicole Dagnes, Federica Marcolin, Enrico Vezzetti, François-Régis Sarhan, Stéphanie Dakpé, Fréderic Marin, Francesca Nonis, and Khalil Ben Mansour. Optimal marker set assessment for motion capture of 3d mimic facial movements. *Journal of biomechanics*, 93:86–93, 2019.

[DWW15] Yong Du, Wei Wang, and Liang Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1110–1118, 2015.

[FH85] Tamar Flash and Neville Hogan. The coordination of arm movements: an experimentally confirmed mathematical model. *Journal of neuroscience*, 5(7):1688–1703, 1985.

[FKA19] Abraham J Fetterman, Christina H Kim, and Joshua Albrecht. Softadam: Unifying sgd and adam for better stochastic gradient descent. 2019.

[FLFM15] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent network models for human dynamics. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4346–4354, 2015.

[HBL11] Sehoon Ha, Yunfei Bai, and C Karen Liu. Human motion reconstruction from force sensors. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 129–138, 2011.

[Hol18] Daniel Holden. Robust solving of optical motion capture data by denoising. *ACM Transactions on Graphics (TOG)*, 37(4):1–12, 2018.

[HRMO12] Ludovic Hoyet, Kenneth Ryall, Rachel McDonnell, and Carol O'Sullivan. Sleight of hand: perception of finger motion from reduced marker sets. In *Proceedings of the ACM SIGGRAPH symposium on interactive 3D graphics and games*, pages 79–86, 2012.

[HSK16] Daniel Holden, Jun Saito, and Taku Komura. A deep learning framework for character motion synthesis and editing. *ACM Transactions on Graphics (TOG)*, 35(4):1–11, 2016.

[HSKJ15] Daniel Holden, Jun Saito, Taku Komura, and Thomas Joyce. Learning motion manifolds with convolutional autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs*, pages 1–4. 2015.

[HWPR+10] Elizabeth T Hsiao-Wecksler, John D Polk, Karl S Rosengren, Jacob J Sosnoff, and Sungjin Hong. A review of new analytic techniques for quantifying symmetry in locomotion. *Symmetry*, 2(2):1135–1155, 2010.

[KMS20] Michał Koźbiał, Łukasz Markiewicz, and Robert Sitnik. Algorithm for detecting characteristic points on a three-dimensional, whole-body human scan. *Applied Sciences*, 10(4):1342, 2020.

[KP08] Paul G Kry and Dinesh K Pai. Grasp recognition and manipulation with the tango. In *Experimental Robotics*, pages 551–559. Springer, 2008.

[KS17] Nitish Shirish Keskar and Richard Socher. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*, 2017.

[LC14] Sijin Li and Antoni B Chan. 3d human pose estimation from monocular images with deep convolutional neural network. In *Asian Conference on Computer Vision*, pages 332–347. Springer, 2014.

[LTMH13] Xugang Lu, Yu Tsao, Shigeki Matsuda, and Chiori Hori. Speech enhancement based on deep denoising autoencoder. In *Interspeech*, pages 436–440, 2013.

[LZC15] Sijin Li, Weichen Zhang, and Antoni B Chan. Maximum-margin structured learning with deep networks for 3d human pose estimation. In *Proceedings of the IEEE international conference on computer vision*, pages 2848–2856, 2015.

[LZWM06] Guodong Liu, Jingdan Zhang, Wei Wang, and Leonard McMillan. Human motion estimation from a reduced marker set. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 35–42, 2006.

[MBR17] Julieta Martinez, Michael J Black, and Javier Romero. On human motion prediction using recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2891–2900, 2017.

[MLCC] Utkarsh Mall, G Roshan Lal, Siddhartha Chaudhuri, and Parag Chaudhuri. A deep recurrent framework for cleaning motion capture data.

[OWL] Markus Oberweger, Paul Wohlhart, and Vincent Lepetit. Hands deep in deep learning for hand pose estimation.

[PH06] Sang Il Park and Jessica K Hodgins. Capturing and animating skin deformation in human motion. *ACM Transactions on Graphics (TOG)*, 25(3):881–889, 2006.

[PT90] William H Press and Saul A Teukolsky. Savitzky-golay smoothing filters. *Computers in Physics*, 4(6):669–672, 1990.

[RKK19] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond, 2019.

[SAPL00] Heydar Sadeghi, Paul Allard, François Prince, and Hubert Labelle. Symmetry and limb dominance in able-bodied gait: a review. *Gait & posture*, 12(1):34–45, 2000.

[SBLW+17] Jürgen Steimle, Joanna Bergstrom-Lehtovirta, Martin Weigel, Aditya Shekhar Nittala, Sebastian Boring, Alex Olwal, and Kasper Hornbæk. On-skin interaction using body landmarks. *Computer*, 50(10):19–27, 2017.

[SIVA17] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[SMB15] Matthias Schröder, Jonathan Maycock, and Mario Botsch. Reduced marker layouts for optical motion capture of hands. In *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, pages 7–16, 2015.

[SMDH13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[TKS+16] Bugra Tekin, Isinsu Katircioglu, Mathieu Salzmann, Vincent Lepetit, and Pascal Fua. Structured prediction of 3d human pose with deep neural networks. *CoRR*, abs/1605.05180, 2016.

[TKY+17] Sarah Taylor, Taehwan Kim, Yisong Yue, Moshe Mahler, James Krahe, Anastasio Garcia Rodriguez, Jessica Hodgins, and Iain Matthews. A deep learning approach for generalized speech animation. *ACM Transactions on Graphics (TOG)*, 36(4):1–11, 2017.

[VLL+10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.

[WJZ13] Nkenge Wheatland, Sophie Jörg, and Victor Zordan. Automatic hand-over animation using principle component analysis. In *Proceedings of motion on games*, pages 197–202. 2013.

[YP03] KangKang Yin and Dinesh K Pai. Footsee: an interactive animation system. In *Symposium on Computer Animation*, pages 329–338, 2003.