Universitat Politècnica de Catalunya-Institut de Robòtica i Informàtica Industrial

A Human Shape-Motion Predictor with Deep Learning

Master in Artificial Intelligence

Antonio Romero Merida

April 24, 2019

Director: Francesc Moreno-Noguer
Codirector: Alejandro José Hernández Ruiz
Advisor: René Alquézar Mancho

Facultat d'Informàtica de Barcelona (FIB)
Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Facultat de Matematiques
Universitat de Barcelona (UB)

Escola Tècnica Superior d'Enginyeria
Universitat Rovira i Virgili (URV)

Abstract

The objective of this master thesis is to obtain an end-to-end solution which predicts human motion and shape from a given video by extracting its 3D pose in the Wild. Given incomplete human motion sequences our goal is to predict the following frames of those sequences. The output of the model would allow us to visualize the predicted motion with a realistic human shape.

To do this, we will make use of a realistic learned model of body shape and pose for the movement representation. For the human motion prediction several deep learning approaches which use RNN networks will be explored and determine which are more efficient for our problem.

Finally, given a video of a person movement in the Wild, we will extract the pose and shape of the person and predict the movement. This will allow us to obtain a end-to-end RGB video to 3D pose and shape prediction.

Acknowledgements

I would like to thank my codirector and director Alejandro Hernández and Francesc Moreno-Noguer for helping me with the development of this master thesis. Without their help in the knowledge of the human-motion prediction and deep learning this would not have been possible.

I also would like to thank Alejandro for the extensive reviews of the process of writing the master thesis and for helping me with the doubts about the implementation of the models and theorical concepts of the framework. Also want to thank Francesc and René Alquézar for the reviews of the master thesis which have improved the quality of this document and make it more scientific. I want to thank my friends from the master in artificial intelligence for helping me resolving some sporadic doubts, as well as my colleagues from the IRI.

Finally, I would like to thank my family and for encouraging me during the realization of this master thesis. Also thanks to my father which has also helped me during the writing and review of this document to improve my style of writing.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The field of the human motion prediction is a relatively little explored field. Several papers have been published during the last few years related with this field. It should be noted that this is not an easy problem because the high dimensionality of the prediction. As stated in [1]: "Learning statistical models of human motion is a difficult task due to the high-dimensionality, non-linear dynamics and stochastic nature of human movement."

Nowadays, artificial intelligence is experiencing great advances. New AI related papers are published and patents are registered [2]. Human motion prediction is a field of AI which can be used in several AI fields.

One example of a field in which it could be used is in the field of autonomous driving. This field although with good future prospects, it has its dangers as several fatal accidents have been reported while doing car tests [3].

In cities, this accidents could have been avoided if the cars had the ability to predict the movement of the pedestrians who are walking on the sidewalk. If a pedestrian were going to make the sudden move of crossing the street without looking for incoming car, an accident with the autonomous vehicle could happen. In the hypothetical case where the autonomous car is equipped with a good human motion predictor system, the accident could have been avoided, as the car would slow down or try to dodge the pedestrian instead of colliding.

In order to make that possible, several human motion prediction models have been published in the lasts years [4], [5],[1], obtaining pretty good results and in each of them in a particular approach.

However, we would like to test whether this results can be improved so we will try a different approach of our own for facing the problem on the human motion prediction. And we are not talking only about improving performance wise. In the previously mentioned papers, the human pose is represented with a skeleton of joints. Such as the representation in figure 1.

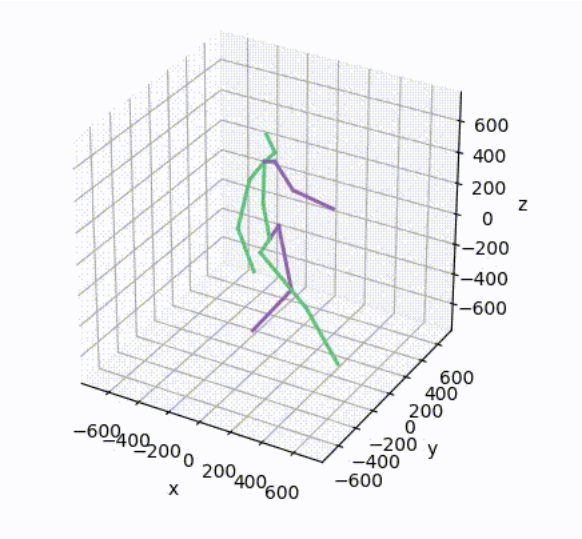


Figure 1: Representation of the motion of a human in [1]

The data used in those papers comes from the Human3.6m dataset [6], composed of 3.6 Million accurate 3D Human poses. The approach that we will follow in this master thesis will be similar to the approach of the one mentioned above. The main difference, is that our data will not exactly be the same as the data from the human3.6m. Our dataset will be in SMPL [7] format, a learned model of human body shape and pose-dependent shape variation that is more accurate than previous models

The main change of the SMPL model with respect to the previous works is that it adds, as quoted from [7]: "a simple formulation that enables training the entire model from a relatively large number of aligned 3D meshes of different people in different poses." It also provides a realistic human shape for the pose representation.

On the contrary, the previous papers that we have mentioned before, do not add this realistic human shape.

That is one of the **main differences** between our approach and theirs. Hence one of our main motivations is to compare different deep learning approaches applied to the field of the human motion prediction by following the SMPL model approach instead of the existing approaches, and compare how good it performs with respect to the existing approaches. Basically, in this master thesis, we ask ourselves the question of whether the SMPL representation data approach obtains better results than following the approach of the Human3.6m dataset.

At this point, all of the papers we have mentioned extract their data from the Human3.6m dataset. If we were following the approaches from those papers, we would be limiting ourselves to only work with the videos from the Human3.6m dataset. However, if we could find an approach that could predict human-motion for any given video which features a person doing some movement, by extracting the needed parameters for our models to predict human-motion, this would be even better than limiting ourselves to the videos from the Human3.6m dataset.

1.2 Goals

One of the goals of this master thesis is to observe how different deep learning architectures work when applied to the human shape motion predictor, assessing its parameters and inputs. Several deep learning architectures in state of the art papers will be reviewed and implemented by using the Pytorch framework of the Python programming language.

To do this, we will start with the paper [1] as a baseline. This paper focuses on the use of RNNs deep learning model to predict human motion for the short-term prediction.

Regarding the representation of the pose and shape of the persons, we will use the SMPL model [7]. With this model of representation we will predict human movement through a short period of time of no more than 2 seconds. The SMPL model has two parameters as an input, the pose and the shape parameters. The combination of the shape and pose parameters will determine the representation of the movement of the subject in each frame of the video of the movement. We will also implement two networks of our own that are inspired by the previous work related to human motion prediction with deep learning.

Another important goal is to be able to make an end-to-end human-motion-prediction, from a given video featuring a human performing some movement. To do this we will use the method from [8], an end-to-end framework for reconstructing a full 3D mesh of a human body from a single RGB image. With this method, we will be able to extract the SMPL input parameters for a given video and predict the human-motion. If done successfully, this would make our approach gain the ability to generalize for doing the human-motion prediction, as long as the video is compatible with the SMPL model.

In the next few sections, we will try implement our differences with respect to the previous work and make experiments obtaining facts and results. Moreover, we will give a theoretical explanation of the methods and publications used during the experiments performed, our proposed methods, as well as a description of the practical framework we have used to prepare and execute the different experiments.

2 Previous Work

In this section we will introduce the previous work on human motion prediction and its different approaches.

First we will review the **SMPL** model [7] which allows a realistic representation of the shape and the pose. Then we will check how to transform from Exponential Map angles to Euler angles, as it is necessary for the metrics of the experiments. After that, we will take a look at the baseline chosen for the experiments [1].

The next step will be to review the network in [9]. In this paper they introduce another LSTM deep learning model for short-term human motion prediction. We will also check the the network in [4]. In this paper, they use an Encoder Recurrent Decoder (ERD) architecture for human motion prediction.

Finally, we will introduce the network in [8]. This model extracts the pose and shape parameters in the same format as the **SMPL** model. With this model, given a video in which a human performs a movement, we can extract its pose and shape parameters for each frame of the video.

2.1 SMPL: A Skinned Multi-Person Linear Model

SMPL [7] is a learned model of human body shape and pose-dependent shape variation that is more accurate than previous models and is compatible with existing graphics pipelines. The Skinned Multi-Person Linear model (SMPL) presented in that paper is a skinned vertex-based model that accurately represents a wide

variety of body shapes in natural human poses. The parameters of the model are learned from data including the rest pose template, blend weights, pose-dependent blend shapes, identity-dependent blend shapes, and a regressor from vertices to joint locations. Unlike previous models, the pose-dependent blend shapes are a linear function of the elements of the pose rotation matrices. This simple formulation enables training the entire model from a relatively large number of aligned 3D meshes of different people in different postures.

The SMPL model has two parameters: pose and shape. These parameters were trained with the use of the multipose dataset [10] and the CAESAR dataset [11] respectively. In the figure 2 we can see examples of the multipose dataset with different poses, with examples for both men and women.

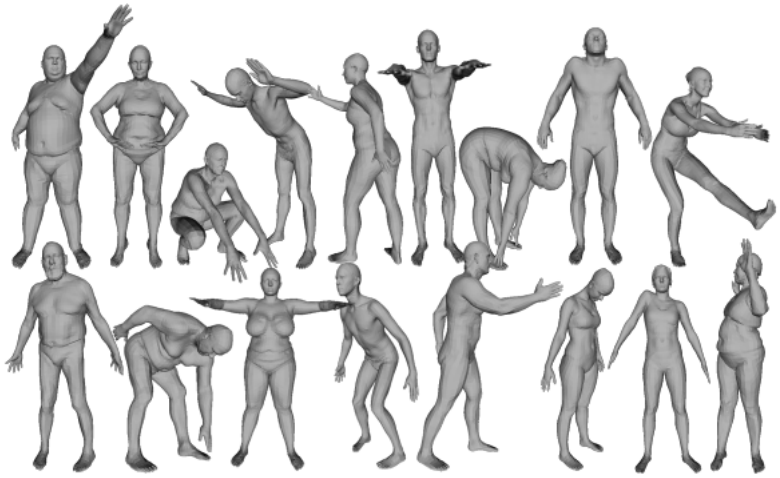


Figure 2: Sample registrations from the multipose dataset [7]

The subjects in the datasets were trained according to their biological sex, either male or female. This training resulted in two different models, one for females and one for males.

Now we will take a closer look at the SMPL parameters.

2.1.1 SMPL Parameters

For pose representation, the SMPL model makes use of two parameters: shape and pose.

- The **pose** parameter of 72 elements. The first three elements represent the root orientation of the pose in the space. The other elements denotes the axis-angle representation of the relative rotation of part k with respect to its parent in the kinematic tree. It has 23 joints. This kinematic tree is the same as in [12].
- The **shape** parameter of 10 elements. This parameter defines the shape of a person, such as height, weight and so on.

2.1.1.1 Root Orientation Parameters

In the sample model, the first three parameters correspond to the root orientation of the model. The first, second and third parameters corresponds to the rotation over the first, second and third axis respectively.

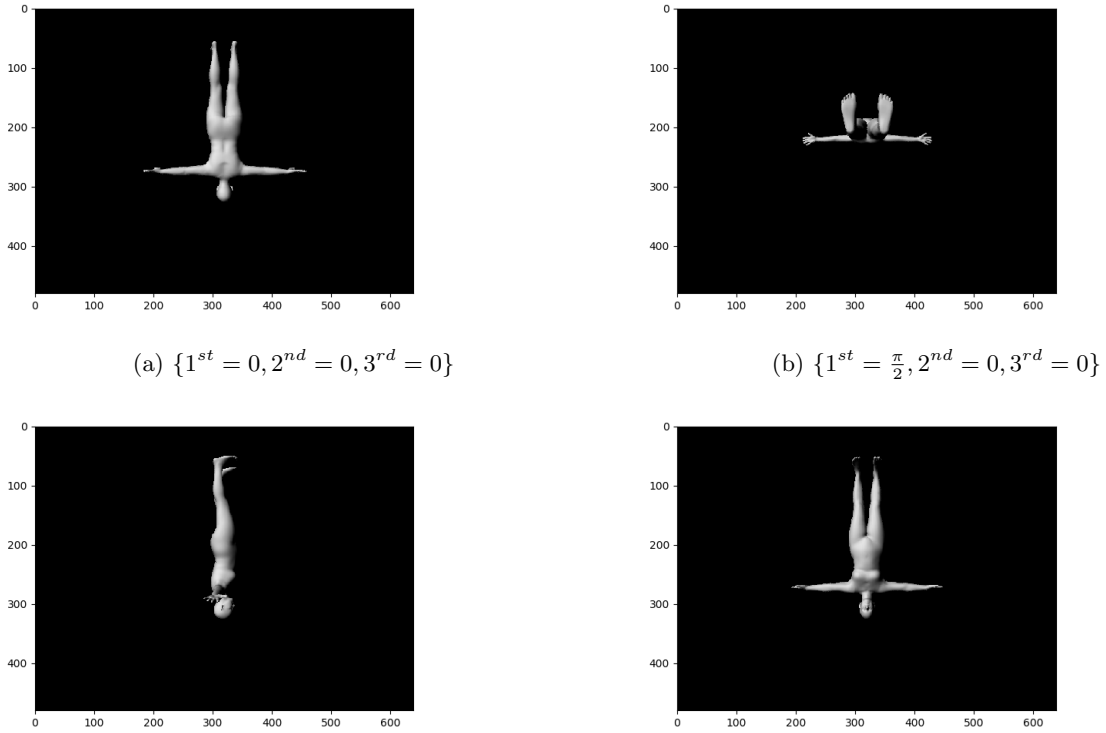
To show this we have set all the elements of the pose parameter vector to 0. Then we assign each of the three first parameters a value in the list $\{0, \pi/2, \pi, 3\pi/2\}$ in radians, which corresponds to the values $\{0, 90, 180, 270\}$ in degrees.

At the figure 3, we can observe the rotation over the first and second axis of the 3D space. Moreover in figure 4 (a) and (b) we can see the rotation over the third axis with the third parameter.

This is achieved by modifying the first three parameters of the SMPL model. Moreover in parts (c) and (d) of the figure 4 we can observe an example of how different parameters affects to the orientation of different joints. For instance, the 6^{th} parameter refers to the joint between the waist and the left leg, whereas the 11^{th} parameter refers to the waist joint.

Please note that, when performing the experiments, the global translation is set to 0 and will not be predicted, as to comply with the preprocessing & protocol in [1] and [4], in order to have comparable results.

This means that only the joints regulation parameters will be predicted.



(a) $\{1^{st} = 0, 2^{nd} = 0, 3^{rd} = 0\}$ (b) $\{1^{st} = \frac{\pi}{2}, 2^{nd} = 0, 3^{rd} = 0\}$

(c) $\{1^{st} = 0, 2^{nd} = \frac{\pi}{2}, 3^{rd} = 0\}$ (d) $\{1^{st} = 0, 2^{nd} = \pi, 3^{rd} = 0\}$

Figure 3: Woman model pose generated with first and second parameters of SMPL [7]

2.1.1.2 Joints Regulation Parameter

While the 3 first parameters of the pose correspond to root orientation, the next 69 parameters of the SMPL model refers to each of the 23 joints in which humans are represented.

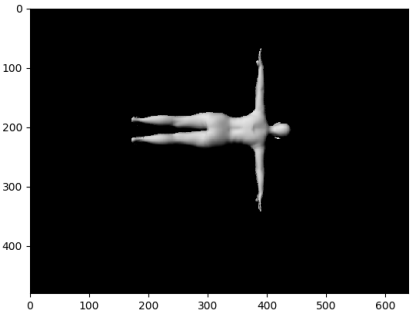
The 69 parameters are organized into tuples of 3 elements. Each one of these tuples refers to a specific joint of the kinematic tree. Because the SMPL model uses 23 joints in total, their rig has K = 23 joints, hence a pose $\overrightarrow{\theta} = [\overrightarrow{w}_0^T, \ldots, \overrightarrow{w}_K^T]^T$ is defined by $\left|\overrightarrow{\theta}\right| = 3 \times 23 + 3 = 72$ parameters. The three parameters added are the root orientation parameters.

The pose of the body is defined by a standard skeletal rig, where $\overrightarrow{w}_k \in \mathbb{R}^3$ denotes the axis representation of the relative rotation of part k with respect to its parent in the kinematic tree in **exponential map** format. The transformation between axis-angle representation and exponential map is done by applying the Rodrigues formula:

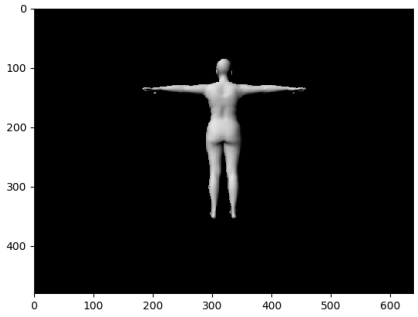
$$\exp(\overrightarrow{w}_j) = I + \hat{\overrightarrow{w}}_j \sin(\|\overrightarrow{w}_j\|) + \hat{\overrightarrow{w}}_j^2 \cos(\|\overrightarrow{w}_j\|)$$

Since the publication of the paper [13], the use of exponential maps in human motion prediction has increased. This paper shows the advantages of this formulation such as being numerically stable when encountering with precision issues, dealing with the inherent singularities of the exponential maps and dealing with the accuracy of the shoulder and hip joints modelling in articulated figures.

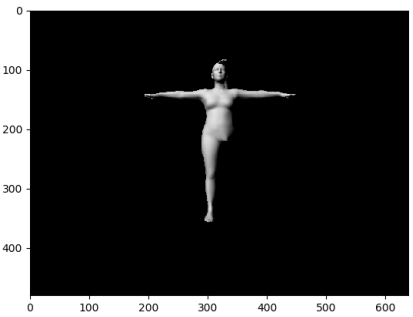
Moreover, the same preprocessing from [1],[5] and [14] is adopted, where the pose is given in exponential map format. If we want to adopt the same protocol from the distances as in [1] when evaluating the model, we have to transform the angles in exponential map format to Euler angles format. This will be explained in the section 4.



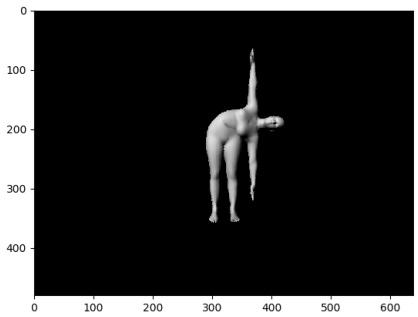
(a) $\{1^{st} = \pi, 2^{nd} = 0, 3^{rd} = \frac{3\pi}{2}\}$



(b) $\{1^{st} = \pi, 2^{nd} = 0, 3^{rd} = \pi\}$



(c) $\{1^{st} = \pi, 2^{nd} = 0, 3^{rd} = 0, 6^{th} = \pi\}$



(d) $\{1^{st} = \pi, 2^{nd} = 0, 3^{rd} = 0, 11^{th} = \frac{3\pi}{2}\}$

Figure 4: Woman model pose generated with different parameters of SMPL [7]

2.1.1.3 Shape Parameter Within the SMPL model, the body shapes of different people are represented by a linear function B_S

$$B_S(\overrightarrow{\beta}; S) = \sum_{n=1}^{\left|\overrightarrow{\beta}\right|} \beta_n S_n$$

where $\overrightarrow{\beta} = [\beta_1, \ldots, \beta_{\left|\overrightarrow{\beta}\right|}]$ and $\left|\overrightarrow{\beta}\right|$ is the number of linear shape coefficients. In our case $\left|\overrightarrow{\beta}\right| = 10$ elements. $S_n \in \mathbb{R}^{3n}$ represent represent orthonormal principal components of shape displacements that were found during the training of the shape from the subjects, with $S = [S_1, \ldots, S_{\left|\overrightarrow{\beta}\right|}] \in \mathbb{R}^{3N \times \left|\overrightarrow{\beta}\right|}$

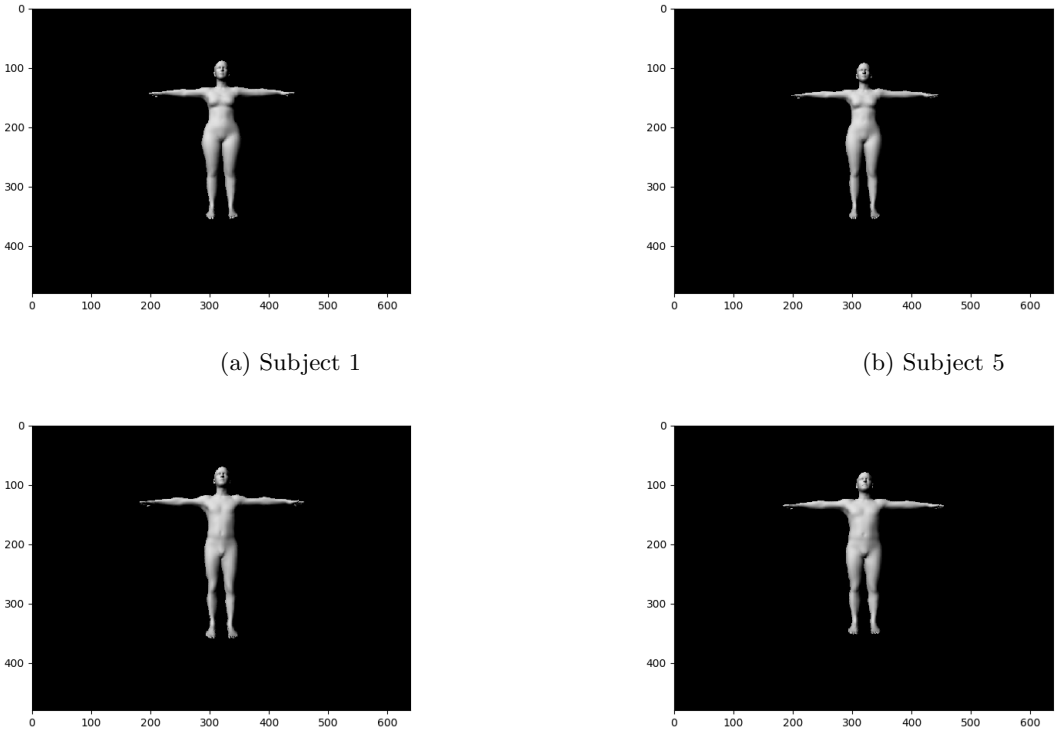
In $B_S(\overrightarrow{\beta}; S)$, the S values a parameter learned from the training of the shape, whereas the parameter $\overrightarrow{\beta}$ was manually set by the SMPL animators as a way to represent the characteristic of each subject. This $\overrightarrow{\beta}$ parameter will be unique to each subject, and it will not change during the human-motion prediction experiments.

The shape parameter defines the body of a subject. In the human3.6m dataset there are 7 subjects available. The rest of the subjects are used for an evaluation benchmark for competitions.

Each of these 7 subjects have different values for the shape parameter. This can be observed at the figure 5, where several value for the beta parameter have been tested.

As we can see the subjects of the human3.6m dataset are pretty similar in shape. The subjects 1, 5 and 7 are female subjects hence the female model of SMPL is used in those cases. We can observe no significant differences between the shapes of the females.

On the other hand, the subjects 6, 8, 9 and 11 are male hence the male model of SMPL is used in those cases. We have also observed that the subject 9 is considerably shorter than the other male subjects.



(a) Subject 1

(b) Subject 5

(c) Subject 8

(d) Subject 9

Figure 5: SMPL Example Shape Parameter Models 1 [7]

2.2 Exponential map angles to Euler angles

We will see in this section how to transform between exponential map angles and Euler angles using the formulas from [13]. In order to transform between exponential map angles and Euler angles, we have to do the following steps:

1. Transform from exponential map angles to rotation matrix angles
2. Transform from rotation matrix angles to Euler angles

Exponential map angles to rotation matrix angles. To transform between exponential map angles and rotation matrix angles is a relatively simple step. Given an exponential map angle r, we can transform it to a rotation matrix R with the following formula:

$$\theta = \|r\|$$
$$r_0 = r/\theta$$
$$R = I + sin(\theta)r_0 + (1 - cos(\theta))r_0^2$$

where, θ is the angle of rotation and r_0 is a unit vector which denotes the axis of rotation [15].
However the transformation between rotation matrix and Euler angles is considerably more complex.

Rotation Matrix to Euler angles. To transform between rotation matrix angles and Euler angles is not that simple. However, with the formulas in [16], this is a much easier step. Briefly, this paper states as follows:
Given a generalized case for a rotation matrix R:

$$R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

Where the angles ψ, θ, and ϕ are the Euler angles associated with the rotation matrix. All the possible solutions from the angles of R can be expressed depending of the value of R_{31}.

1. $R_{31} = \pm 1$. In this case, ϕ can be any value, so we set it to 0 to make calculations easier. There are other two possible cases depending of the value of R_{31}

   (a) $R_{31} = -1$ In this case:

   $$\theta = \frac{\pi}{2}$$
   $$\psi = \phi + \arctan 2(R_{12}, R_{13})$$

   (b) $R_{31} = 1$ In this case:

   $$\theta = -\frac{\pi}{2}$$
   $$\psi = -\phi + \arctan 2(R_{12}, R_{13})$$

2. $R_{31} \neq \pm 1$ In this case:

   $$\theta_1 = -\arcsin(R_{31})$$
   $$\theta_2 = \pi - \theta_1$$
   $$\psi_1 = \arctan 2(\frac{R_{32}}{\cos\theta_1}, \frac{R_{33}}{\cos\theta_1})$$
   $$\psi_2 = \arctan 2(\frac{R_{32}}{\cos\theta_2}, \frac{R_{33}}{\cos\theta_2})$$
   $$\phi_1 = \arctan 2(\frac{R_{21}}{\cos\theta_1}, \frac{R_{11}}{\cos\theta_1})$$
   $$\phi_2 = \arctan 2(\frac{R_{21}}{\cos\theta_2}, \frac{R_{11}}{\cos\theta_2})$$

With the use of the formulas in [16], we can always obtain a solution (there can be more than one solution). But for the sake of the experiments performed in this master thesis, it does not matter which Euler angle we obtain as long as it is equivalent to the rotation matrix angle. So we will only consider ψ_1 and ϕ_1 in the case where $R_{31} \neq \pm 1$.

Within the code, the functions for this transformations are in *src/utils/pytorchangles.py*.

2.3 On human motion prediction using recurrent neural networks (RNNs)

The paper [1], is a paper focused on the use of deep recurrent neural networks (RNNs) to model human motion, with the goal of learning time-dependent representations that perform tasks such as short-term motion prediction and long-term human motion synthesis.

The deep learning model discussed in the paper is shown in figure 6. The figure represents an unrolled GRU network. Each GRU box represents a cell state of the GRU network. This model is composed by two subnetworks. The first subnetwork is an encoder network, where the ground truth of the movement is fed as an input.

The second subnetwork of the model is a decoder. In this decoder, being a Recurrent Neural network (RNN), for each frame of time t, the output of the network is given as an input of the $t+1$ frame. Moreover, the input of the network is added to the output, giving the decoder a **residual connection** (represented in the image with the circle with the + sign).

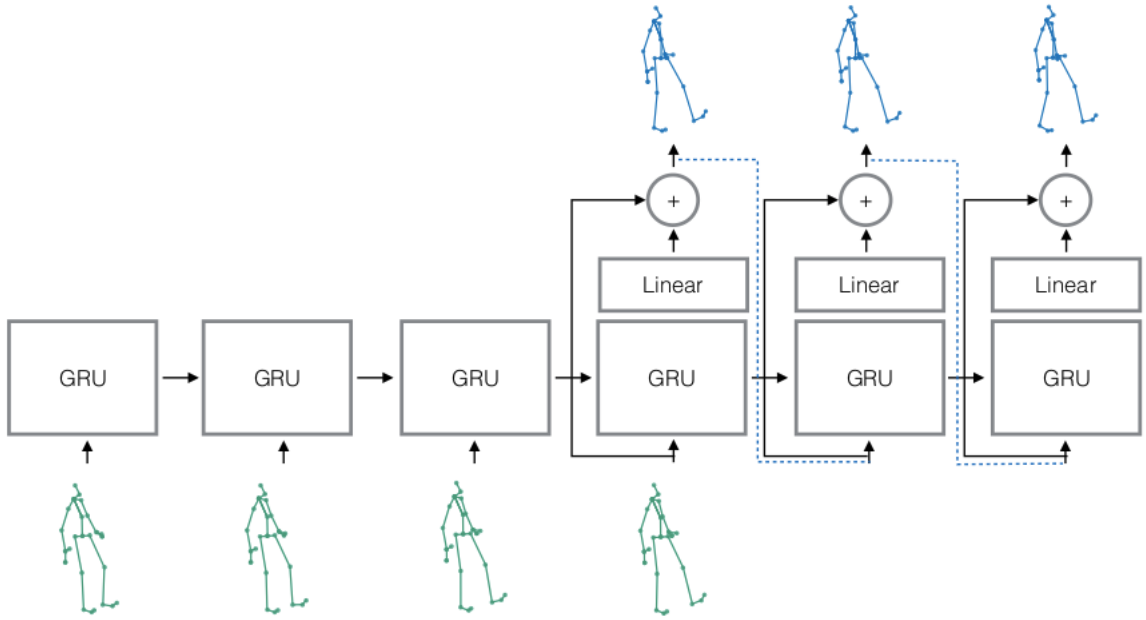


Figure 6: The sequence-to-sequence architecture in the [1] paper. The figure represents an unrolled GRU network. Each GRU box represents a cell state of the GRU network. During training, the ground truth is fed to an encoder network, and the error is computed on a decoder network that feeds its own predictions. The decoder also has a residual connection, which effectively forces the RNN to internally model angle velocities.

The more interesting part for us about this network is that a skipping connection to the next frame is added between frames (represented with the blue dotted line in the figure 6). They perform experiments with different versions of this network such as making the decoder having a residual connection by feeding his own input, or having a supervised or unsupervised training.

2.4 Unsupervised Learning of Video Representations using LSTMs

In this section we will give the theoretical basis of the paper [9].

In this work, they use Long Short Term Memory (LSTM) networks to learn representations of video sequences. The model uses an encoder LSTM to map an input sequence into a fixed length representation. This representation is decoded using single or multiple decoder LSTMs to perform different tasks, such as reconstructing the input sequence, or predicting the future sequence.

Within this paper, they try the 3 different networks, each one with its advantages and disadvantages.

The first network is a **LSTM Autoencoder Model**, which uses both the Learned Representation and the Input Reconstruction networks from the figure 7.

It consists on an encoder LSTM and a decoder LSTM. The encoder LSTM reads the movement sequence. After the last input has been read, the cell state and output state of the encoder are copied over to the decoder LSTM. The decoder outputs a prediction for the target sequence, where the target sequence is same as the input sequence, but in reverse order. The inspiration behind this reasoning is the stack representation of certain programming languages such as LISP.

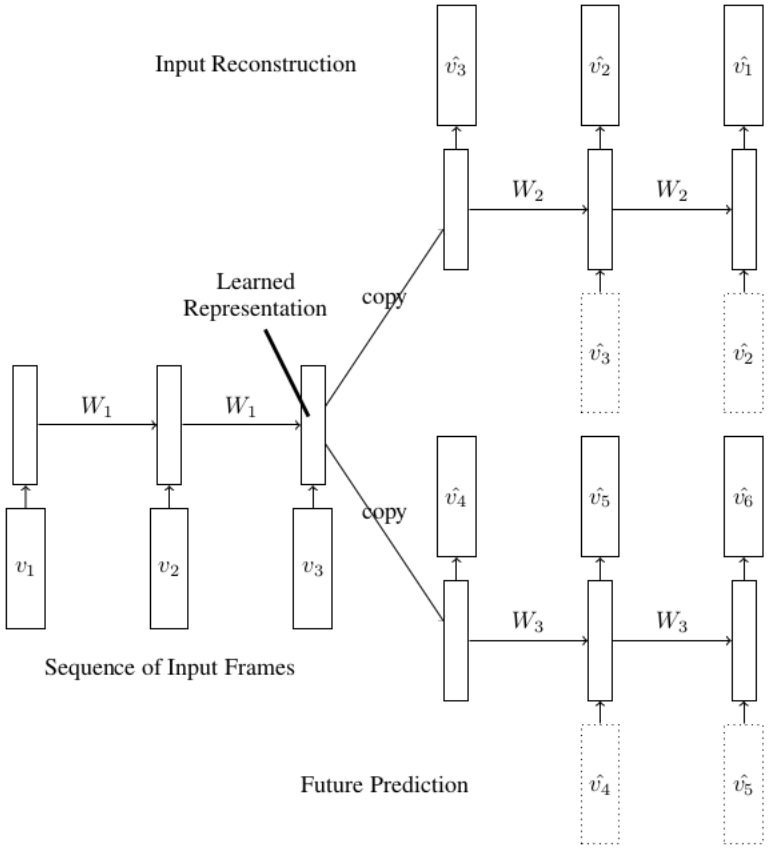


Figure 7: The composite model: The LSTM predicts the future as well as the input sequence [9]

The second network is a **LSTM Future Predictor Model**. It is similar to the LSTM Autoencoder Model. The difference is that, whereas the LSTM Autoencoder predicts frames of the movement that come just after the frames passed as an input, the LSTM Future Predictor Model predicts frames into the future. That is why, in figure 7 the LSTM Autoencoder starts with the frames v_3,v_2,v_1..., whereas the Future Predictor starts with the frames v_4,v_5,v_6...

Finally, the third network, is a combination of the previous two networks attempting to reduce their shortcomings and strengthen their advantages. It is the **composite model** and is shown in figure 7.

On the one hand, a good autoencoder would learn trivial movements which just memorize the inputs, not predicting the future really well.

On the other hand, a future predictor would tend to store only the information about the last frames as they are more useful for predicting the future than the previous frames.

In theory, as the Composite Model combines both of the autoencoder and the future predictor, the results of this network will be better than the results of the two previous networks. In practice, this is corroborated by the results shown in [9].

2.5 Recurrent Network Models for Human Dynamics

The paper [4] is a paper in which a Encoder-Recurrent-Decoder (ERD) network for human motion prediction is used. While being a Recurrent Neural Network, the ERD model incorporates nonlinear encoder and decoder networks before and after recurrent layers.

The architecture of the ERD model can be seen at figure 8. On the left, we can observe that at each time step t, given an input vector x_t of a sequence $x = (x_1, \ldots, x_T)$, it passes through the encoder, the recurrent layers, and the decoder network, producing the output y_t. In this paper, they are interested in estimating some function $f(x)$ of the input x at the current time step, or at some time in the future.

On the right of the figure, we can see the specific configuration that they use in this paper for human motion

prediction. They apply some noise to the input sequence. After that they apply a non-linearity functionwith two fully connected networks where they apply the relu activator.

After that, they apply 2 LSTM networks for the Recurrent part of the ERD. Finally, for the decoder, they apply 3 fully connected layers with several relu activators.

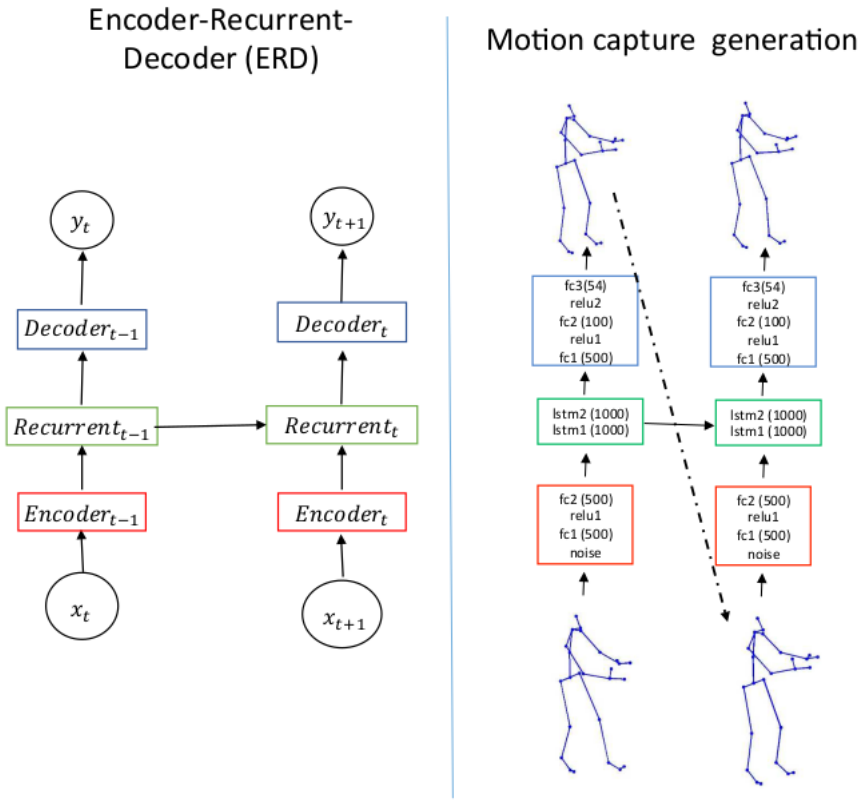


Figure 8: Encoder Recurrent Decoder (ERDs) for human dynamics in video and motion capture [4]

2.6 End-to-end Recovery of Human Shape and Pose

The models we have seen until now are used for the pose estimation or the pose and shape representation in the case of the SMPL model. When using any of those methods we already know the shape and the pose for each of the frames in the video representing the human movement. However, what if we wanted to, from any given video in which there is a person doing a movement of some kind, extract his/her shape and pose parameters?.

This is where we introduce the paper [8]. With the network from this paper, they make use of the Human Mesh Recovery (HMR), an end-to-end framework for reconstructing a full 3D mesh of a human body from a single RGB image.

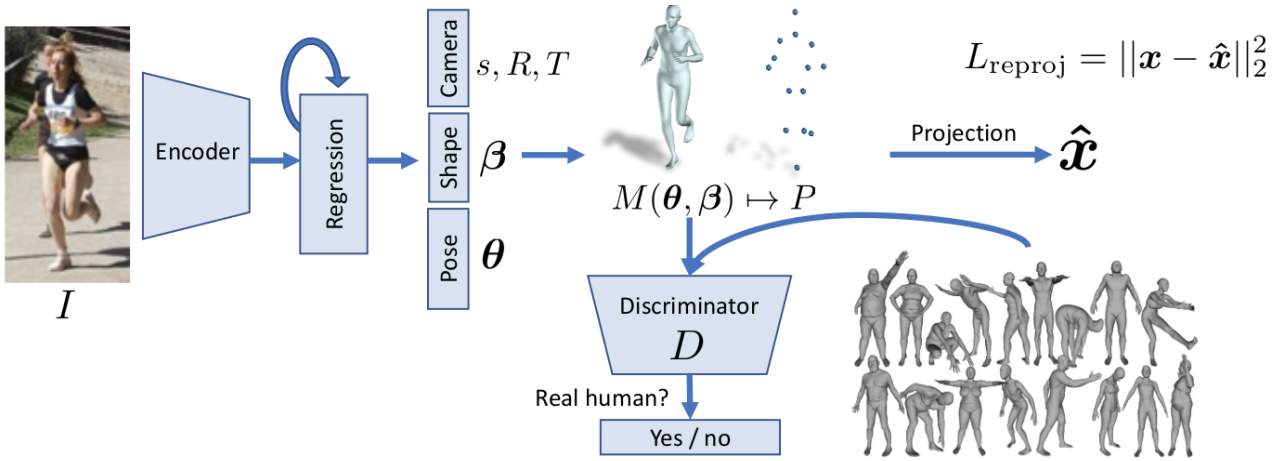


Figure 9: Overview of the proposed framework. An image I is passed through a convolutional encoder. This is sent to an iterative 3D regression module that infers the latent 3D representation of the human that minimizes the joint reprojection error. The 3D parameters are also sent to the discriminator D, whose goal is to tell if these parameters come from a real human shape and pose [8].

As we can observe from the figure 9, the main idea of the model of this paper is to use a convolutional encoder, which is then sent to an iterative 3D regression module. This module infers the latent 3D representation of the human which minimizes the joint reprojection error. Then the 3D parameters are sent to a discriminator, in order to tell whether those parameters are from a real human or not.

For our experiments, we will only need the shape (β) and the pose (θ) parameters. The rest of the parameters will not be necessary as they are not used in the **SMPL** model.

An example of the application of this method can be found at the figure 10 using an image from the COCO dataset. As we can observe in the 3D Mesh overlay part of the figure, this method is not perfect although it gives pretty good results.

Please note that the dataset in which we base the experiments from this master thesis was extracted by applying this very same method to the original human3.6m dataset. This method is not perfect so extracting the pose and shape parameters for each frame of the video movements has some noise and the data obtained is not as good as extracting the data from the original human 3.6m dataset.

However, if the SMPL parameters are extracted successfully and with no a big amount of noise, then the movement prediction will be possible.

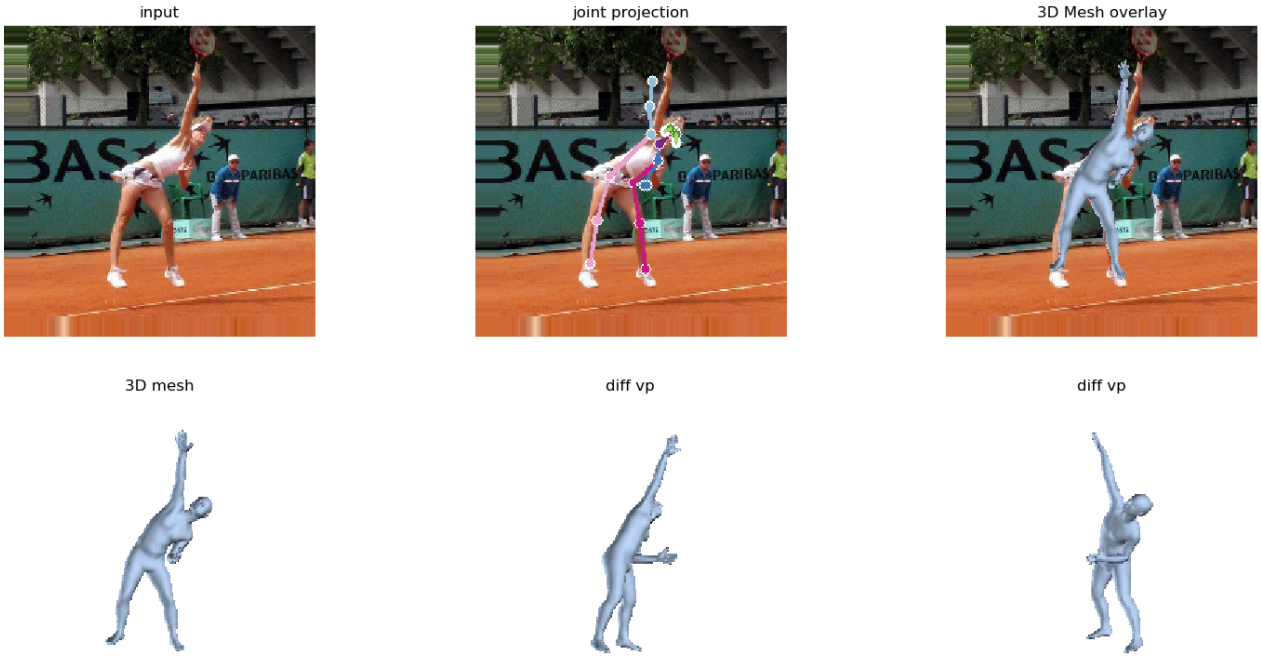


Figure 10: Example of the End-to-end Recovery of Human Shape and Pose. [8]

In the following section, we will comment the theoretical framework in which the experiments performed during this master thesis will be based.

3 New Models

Now that we have introduced the previous related work with the purpose of the human motion prediction, let's introduce the theoretical concepts of the networks that we will use on the experiments.

Apart from the networks from the Previous Work, we have implemented two more networks of our own which are based on the papers from the Previous Work. The first network is based on the network from [1] whereas the second network is based on the ERD network from [4]. We will refer to this first network as *Simple Residual*, because we have added a residual connection and is a relatively simple network to implement. The second network will be called *ERD Residual* (Encoder Recurrent Decoder), because is based on the ERD network from [4] and because a residual connection was added.

3.1 Simple Residual

The **Simple Residual** network is based on the ideas of the [1] paper. The main reason why we thought that this network could be a good idea is because it has a similar but simpler architecture to the baseline paper, which is the one which gives the best results of all that we have tried.

Please also note that this network was the first one to be implemented during the development of the master thesis so we decided for the experiments to leave a simple deep learning architecture to see how well it compared with the other implementations.

In figure 11 we can observe the differences between the architecture from the baseline paper and our network. As we can observe our network does not use the decoder that they use in the network paper.

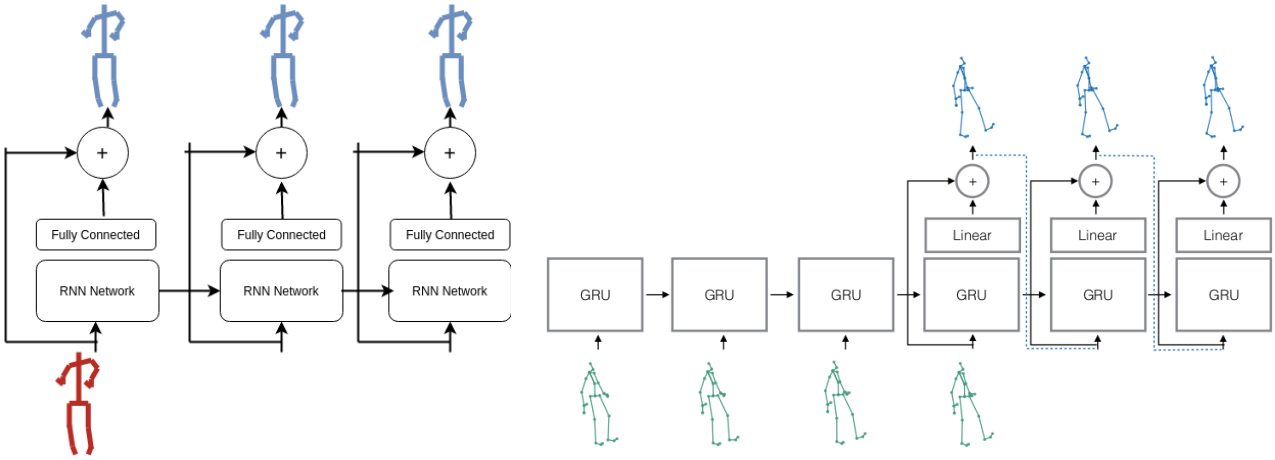


Figure 11: Comparison of the Simple Residual vs Martinez. [8] architectures. Left:Our Simple Residual Network with a RNN network divided into 3 RNN Cells, a linear fully connected function and a Residual Connection. No encoder or decoder networks. Right: Network in [1] with an encoder, decoder and residual networks. The figure represents an unrolled GRU network. Each GRU box represents a cell state of the GRU network

Keeping a simple architecture by using shallow RNNs can achieve great results in tasks such as learning of sentence-level embeddings [20], whenever a long as a large corpus of data is available. It also has the advantage of being computationally less expensive that other deep models. This allow us to keep a relatively short amount of time when performing experiments in the dataset.

Moreover, we decided to not apply the skip frames connection, but put the residual connection. The reason why we decided to apply a residual connection in the network is to allow the network to learn only the prediction part of the problem, copying the known information. This way the network is able to train faster and reach a better solution. Regarding why we decided to not include the encoder and decoder networks, we wanted to test if the encoding and decoding are better modeled explicitly or simply let the RNN to learn the transformation for us. Most of the papers cited in this master thesis [1], [4], [9] use an encoder somewhere in their network. We want to check what happens with the results when we keep the network simple. That is, there is no non-linearity operations or encoders and decoders.

3.2 ERD Residual

The **ERD Residual** network is based on the network in the paper [4]. This network, while being a Recurrent Neural Network, incorporates a nonlinear encoder and decoder networks before and after recurrent layers

To make a variant of this network, our idea is to try how a **residual connection** would alter the results of the original network. This is because residuals connections are present is some relatively new models [1] and we would like to see the performance of uniting a encoder RNN decoder architecture with a residual connection. This can be seen at the figure 12.

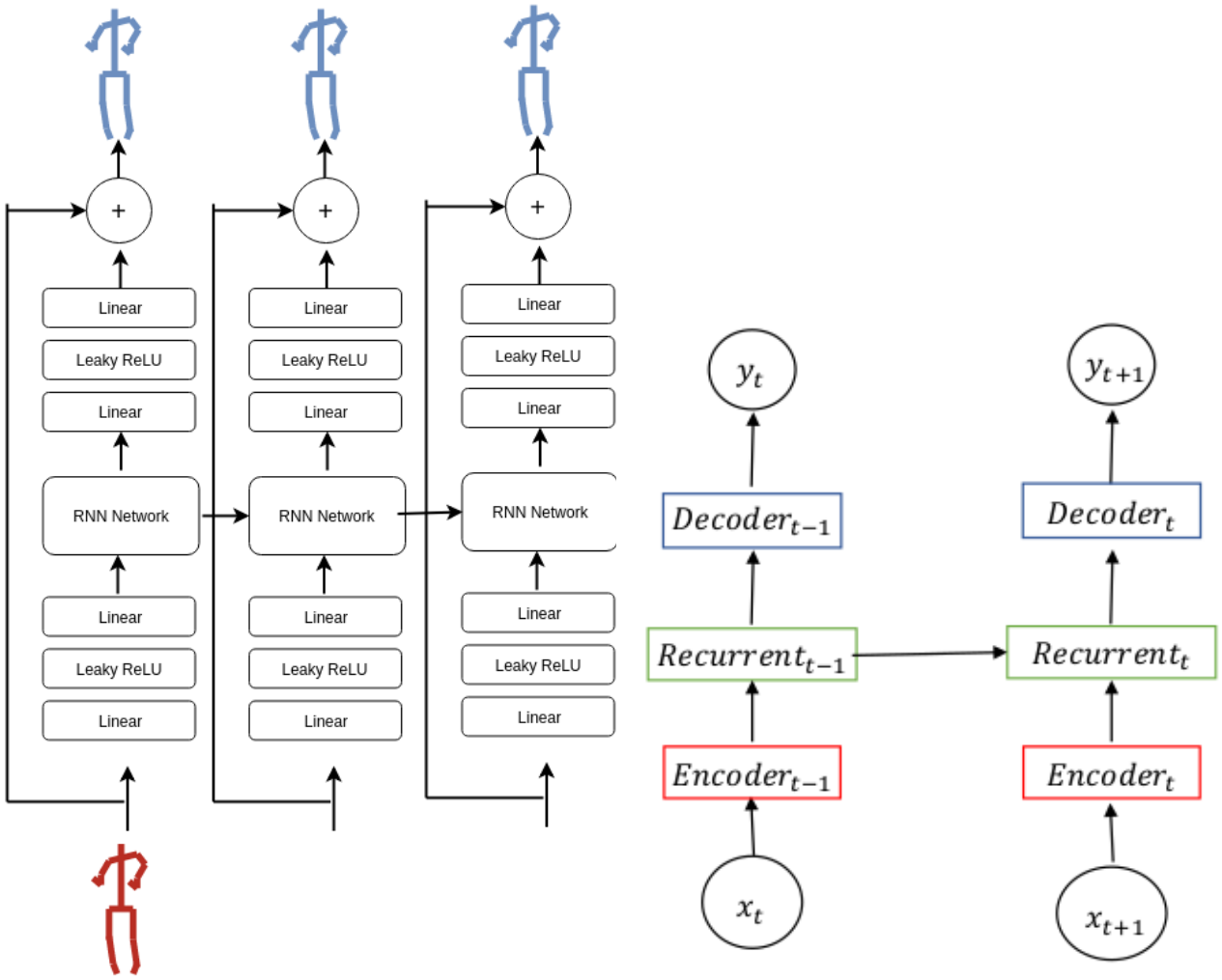


Figure 12: Comparison of the ERD Residual vs ERD original architectures. Left:Our ERD Residual Network. Right: Network ERD in [4] without a residual connection

As we can see observe on the figure, each input frame passes through an encoder which is composed of 2 linear functions plus a Leaky ReLU activator. The reasoning behind why we decided to use a ReLU type activation function instead of a sigmoidal one is because it has a better gradient propagation, with fewer vanishing gradient problems compared to sigmoidal activation functions that can saturate in both directions [17].

Moreover, ReLU activations can suffer from the vanishing fradient point as well. Whenever the learning rate is set too high, ReLU neurons can sometimes be pushed into states in which they become inactive for essentially all inputs. This can be reduced by using Leaky ReLUs, as they allow a small, positive gradient when the unit is not active.

We will also consider two different types of inputs and different recurrent neural networks to see which combination gives better results: copying the last frame and passing zeros. With this network, from a conceptual point of view, because is a residual model, it benefits greatly on copying the last frame. This is because that way it can focus on the problem of the motion, and doesn't need to learn to reconstruct the whole unseen frames.

After introducing the two networks, we will test them in the experiments section. We will compare them with the networks introduced in the papers from the previous work.

4 Implementation

This section is related to the implementation of the deep learning framework in which the experiments were executed.

First, we will talk about the general implementation details of the code and the hardware for the experiments.

After that, we will describe the dataset human3.6m and the preprocessing which was done to obtain the data for our experiments.

Finally, we will enter in more detail about the implementations of the network and models used in this master thesis.

4.1 General Aspects

The first important decision that was taken when starting the implementation of the master thesis code was which deep learning framework to use for the implementation of the deep learning models.

Currently Tensorflow is at the top of the list for the most usage in companies [18]. However, Pytorch is getting more and more popular in research environments. Moreover, for us Pytorch looks more intuitive that Tensorflow for the implementation of deep learning models.

At the end it was decided to do the implementation with **Pytorch**, as it seems more intuitive for us and is quite common to use in research.

This master thesis was done collaborating with the *Institut de Robòtica i Informàtica Industrial of the Universitat Politècnica de Catalunya-CSIC* [19].

As a member of this research group has developed a common pytorch framework for deep learning [20], it was decided that the framework would act as a code baseline for the experiments in this master thesis. The framework is written in Python version 3.6.

The code which has been used to implement the deep learning models, preprocess the data and generate the results of the experiments can be found at `https://github.com/finuxs/Shape-Motion-Predictor` [21].

Please note that, although the framework uses syntax of Python 3.6, it has been adapted to understand Python 2.7 as well. The reason behind this is that the whole SMPL model, is written in Python 2.7, using libraries such as *Chumpy* and *Opendr* which are not compatible with Python 3.6. The adaptation of the code has been done with the use of python **future statement definitions** among other code adapters.

Regarding the **hardware** used for the experiments. They were executed using several Graphical Processing Units (GPUs).

As doing the master thesis at the IRI, I had access to one of its servers. This server had 4 GPUs of type Tesla K40c, with a Ram memory of 12 Gb each.

Moreover a personal laptop was used with a GPU Nvidia GeForce GTX 1050 Ti which was available 100 % of the time.

The practical framework was developed using the Python IDE Pycharm Professional edition.

4.2 Dataset

Here we will give more details about the dataset used to perform the experiments. This dataset is the Human3.6m dataset [6].

The **human3.6m dataset** [6], is composed of 3.6 Million accurate 3D Human poses, acquired by recording the performance of 5 female and 6 male subjects, under 4 different viewpoints, for training realistic human sensing systems and for evaluating the human pose estimation models and algorithms.

Although the human3.6m dataset includes 11 different subjects, we will only have 7 of them available. This is because the rest of the subjects are reserved by the authors as a test set for the Human 3.6m benchmark. So our dataset will consist only of the subjects: 1,5,6,7,8,9 and 11. Three of the subjects in the dataset are females whereas the other four are males.

The data is comprised by 15 training motions. These motions containing the movements that can be seen at the figure 13.

Figure 13: Different movements included in the human3.6m dataset [6]

The actors were given detailed tasks with examples in order to help them plan a stable set of poses between repetitions for the creation of training, validation and test sets. In the execution of these tasks the actors were however given quite a bit of freedom in moving naturally over a strict, rigid interpretation of the tasks.

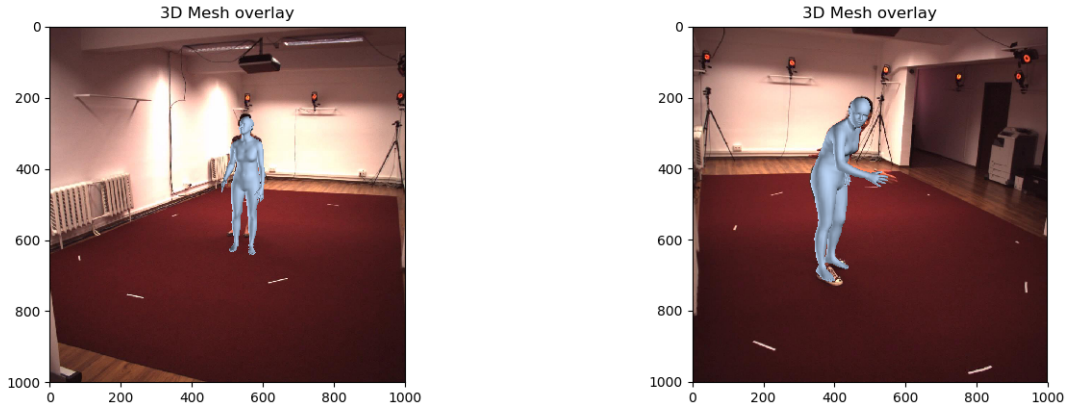
For each movement, there are two submovements. The submovements of one movement differ from one another in the sense that even though the semantic information contained in the images is similar, the two submovements are not exactly the same as they represent the same movement in different ways.

4.2.1 Data preprocessing, protocol & normalization

4.2.1.1 Preprocessing There is an important difference between the data used in the papers [1] [4] and the data used in this master thesis. While most methods obtain their data after processing the original joint data from the Human 3.6 dataset, we obtain ours from processing the RGB videos of the Human 3.6 dataset.

This is because the data from this master thesis was extracted by applying the method in [8]. That is, by applying their convolutional encoder, we manage to extract the pose and shape parameters of the SMPL model [7].

Although good, this extraction of the SMPL beta and pose parameters is not perfect. We can see some examples of this in the figure 14.



(a) Subject 1 Action 5 Subaction 1

(b) Subject 7 Action 5 Subaction 1

Figure 14: Human 3.6m to SMPL model with the use of the End-to-end Recovery of Human Shape and Pose [8]

This produces some noise in the dataset for the experiments. The reason is that some frames can have an small error in the shape and the pose extracted with respect to the original human3.6m dataset.

In fact, some of the movements of the subject 11 could not be extracted with this method as some parameters were missing. Hence, we decided to exclude these movements from our dataset. The affected movements are the Act02 (Give Directions) and the Act06 (Talking on the phone).

As each of the subjects, performs 15 movements each, and each movement is performed 2 times and divided into 2 subacts. we have in total $7 \times 15 \times 2 = 210$ different movements. As we have excluded those 2 movements from subject 11 with 2 submovements each, at the end we have 206 **movements** in our dataset in total.

4.2.1.2 Protocol Our data protocol will be similar to the used in [1]. We will use the subjects 1,6,7,8,9 and 11 for the training and the evaluation with the subject 5.

The division of the data inside the subjects can be found in the figure 15. Each subject performs 15 different acts(or movements). Each movement is divided into 2 subacts (or submovements).

Inside each subact, there are 4 cameras. This cameras recorded each of the movements from different positions of the 3D space. Finally, inside each camera, there are Matlab (.mat) files which contain the pose and beta vectors in SMPL model format of the subjects per each frame of time.

Moreover, there is the X32 representation of the joints of the subject per frame of time, which corresponds to the human3.6m representation of the pose. The X32 is a matrix of dimension of 32x3, where the columns represent the x,y and z 3D coordinates respectively.

The number 32 is the number of joints used for the pose representation in the human3.6m dataset. Please note that in the SMPL model, the number of joints is only 23, as some of them were not used at all for the shape representation so they were not included in the data.

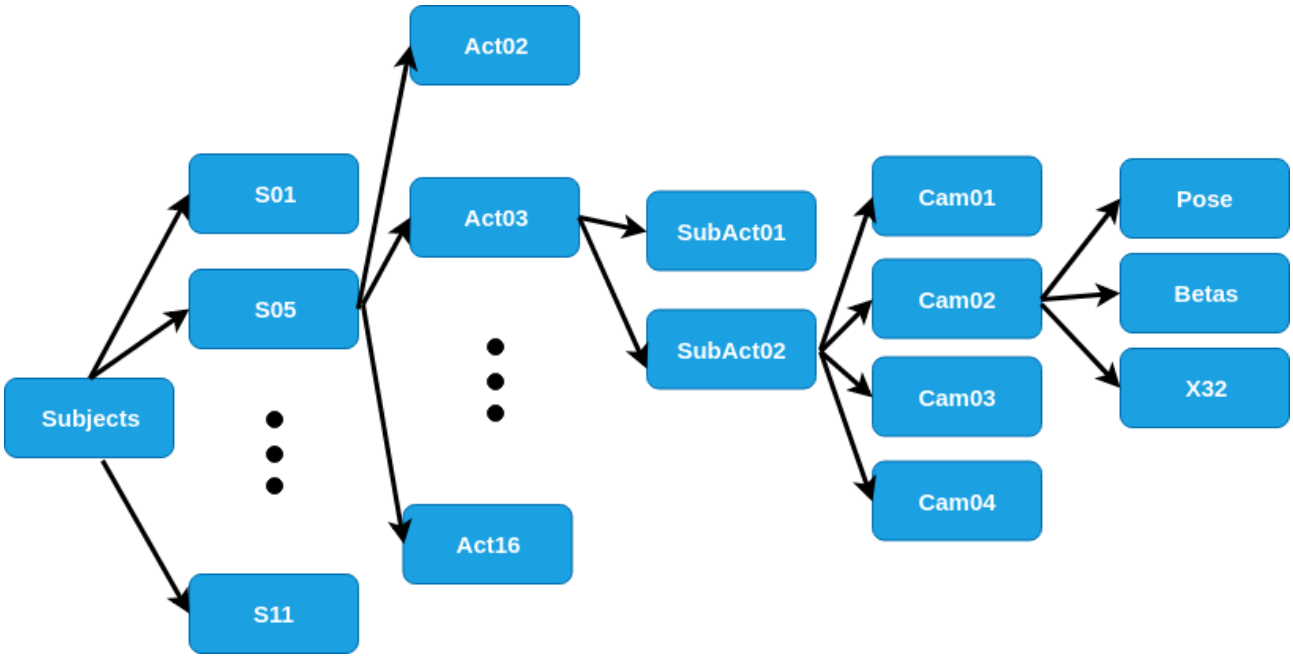


Figure 15: Hierarchical structure of the data inside the subjects. Pose is a vector of 72 elements. Beta is a shape vector of 10 elements. X32 is a matrix of 32x3

The data structure chosen to store the data of the experiments performed in this master thesis is the hdf5 data format [22]. This format was chosen because pytorch will access the data in each iteration of the training and the test part, so a high performance I/O is needed for this kind of problems.

To transform the data into the hdf5 format was necessary because the original data alone occupied around 750 Gb, and that is an unpractical amount of data. Without images and other variables which are not necessary for the SMPL model, we have stored all the necessary data for the training and validation part in just 1.5 Gigabytes.

For the store of the data in a hdf5 file we have read all the frames for each movement, from the 4 different cameras, and store the pose and beta parameters from each of them.

The data has been saved in a dictionary structure, with the index '000' of the dictionary referring to the 1^{st} SubMovement of the 1^{st} Movement of the 1^{st} Subject. The index '001' refers to the 2^{nd} SubMovement of the 1^{st} Movement of the 1^{st} Subject... and so on ...

We can observe those indexes on the table 1. We have stored first the subjects corresponding to the training and last the subject corresponding to the validation and testing.

Index Start	Index End	Subject
000	029	1
030	059	6
060	089	7
090	119	8
120	149	9
150	175	11
186	205	5

Table 1: Correspondence between our hdf5 file and the human3.6m subjects

Also, in the table 2, we have established the equivalences between Action numbers of our dataset and the

movements withing the human3.6m dataset.

Action Number	Action Human3.6m
Act02	Directions
Act03	Discussion
Act04	Eating
Act05	Greeting
Act06	Talking on the phone
Act07	Posing
Act08	Making purchases
Act09	Sitting on chair
Act10	Activities while seated
Act11	Smoking
Act12	Taking Photo
Act13	Waiting
Act14	Walking
Act15	Walking Dog
Act16	Walking Together

Table 2: Correspondence between our dataset and the human3.6m dataset

4.2.1.3 Data Normalization Once we have obtained the shape and pose parameters of the dataset, they have to be normalized so the deep learning networks can work properly. The reason behind this is that the implementation of deep learning models suffers from numeric problems in computers which have a fixed and relatively small precision in the representation of numbers.

The way we have done in our case is to calculate the mean and the standard deviation of all the dataset, that is, training and validation. Normally, you only take the statistics over the training set, completely ignoring the test set.

However, in the protocol of the baseline paper [1] they use the same normalization for both the training and the test and we have done the same.

To normalize, for each sample in the dataset, we subtract the mean and divide by the standard deviation of all the samples. The formula is the following, given a sample s of the dataset:

$$new(s) = \frac{s - \text{mean(s)}}{std(s)}$$

After applying the formula, the samples are standardized to have a mean of zero and a standard deviation of 1.

In the next section we will give some of the implementation details for the dataset and the deep learning network for our framework.

4.3 Implementation Details of the Dataset

4.3.1 Dataset

The default framework of Pytorch for deep learning [20] was made for the most common datasets such as CIFAR10, COCO and MNIST. To aid the development of new proyects, the Pytorch framework provides default readers that handle common datasets such as those.

The dataset that is used in the experiments of this master thesis is customized and has its own preprocessing. So the code for loading the dataset had to be adapted almost completely. We can do this with the custom datasets loader of Pytorch.

The code, which is the file *src/datahuman36m.py* that has been implemented for reading the data from our custom dataset can be divided into two general parts:

- Samples Indexing
- Get sample with index: Random Cropping & SubSampling

4.3.1.1 Sample Indexing

This part of the implementation is related with the organization of the data in our framework. In order to follow the protocol of the data in [1], we need to divide our samples between training, validation and testing. Subjects 1,6,7,8,9 and 11 for training and subject 5 for validation and testing.

In our framework, we have 3 .txt files in *datasets/h36m* called: train_ids, val_ids and text_ids. Inside these files, we must write the indexes of the sample movements that we want to treat as training, validation and testing respectively. So for the training part, we write the indexes corresponding to the subjects 1,6,7,8,9 and 11 in the train_ids file. In this case, that would be the indexes from 000 to 175 included, as the table 2 shows. For the validation, the indexes would be from 176 to 205 included.

4.3.1.2 Get sample with index: Random Cropping & SubSampling

Sample Retrieving As we mentioned before, our data has to be in a dictionary structure, with its keys corresponding to different movements as shown in table 1. The reason of why this kind of data, is because, in Pytorch, the pre-built function which reads data samples when training or testing is the function: *__getitem__(index)*. This function receives as an argument the index of the dictionary which stores the sample which is going to be retrieved. Without the index, Pytorch would not be able to identify each sample.

The Pytorch DataLoader for custom datasets calls the __getitem__ function from the dataset and iterates it using the specified batch size. It will be inside this __getitem__ function where we will **random cropping** and **subSampling.**

Random Cropping Our sample movements will be of 4 seconds in total. 2 will be fed as an input to the networks and the other 2 will be predicted, the same as in [1] so our experiments can be equivalent.

The videos of the human3.6m dataset have a frame rate of **50 fps** (frames per second). In order to predict 4 seconds we would need a total of 50 frames $= 25$ fps $\times 4$ s .

One thing that we have realised, is that, choosing always the same sequence of frames per movement does not give reliable results because the frames chosen correspond always to the same part of the movement.

In order to add statistical robustness to the results of our experiments we do not always choose the same sequence of frames, but a random sequence of successive frames of the sample movement. This is what is called **Random Cropping**.

Subsampling We can even add more statistical robustness by not picking all the frames successively. For instance, we could take only the next 100 frames which are even or odd. This procedure is called **subsampling**.

We can even go one step further with a **subSampling of window n**. This procedure consists on the following: for each n frames in a sequence, pick up only one of them randomly and ignore the rest. This procedure is shown for window 2 and window 3 in the figures 16 and 17 respectively.

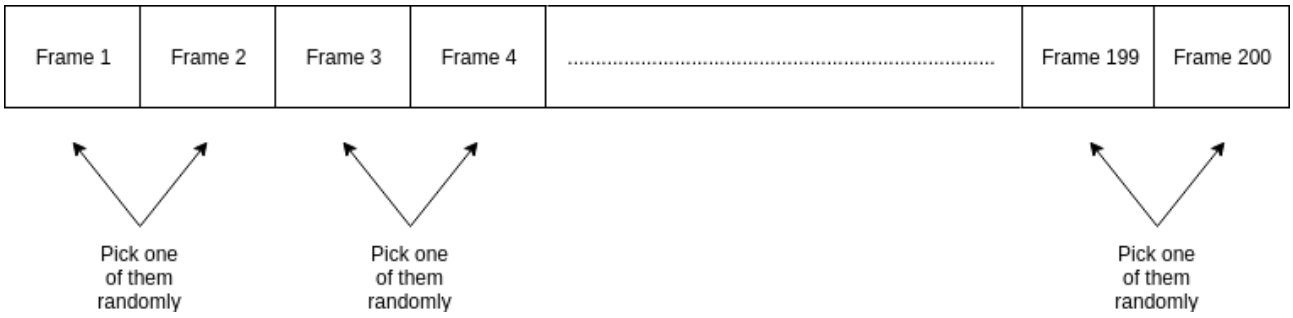


Figure 16: Subsampling of 200 frames with a window of size 2

Given for instance 200 frames of a movement, if we apply a window 2 subsampling we would end up with 100 frames that would be a random combination between even and odd frames. Subsampling of window 3 is a similar case, where from 300 frames we would end up with 100 frames. Each frame n of those 100 frames would be either be $n \mod 3 = 0$, $n \mod 3 = 1$ or $n \mod 3 = 2$.

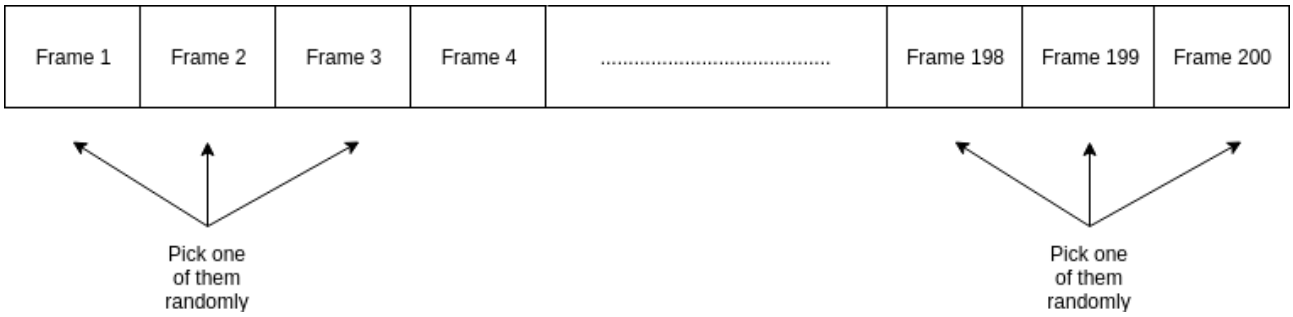
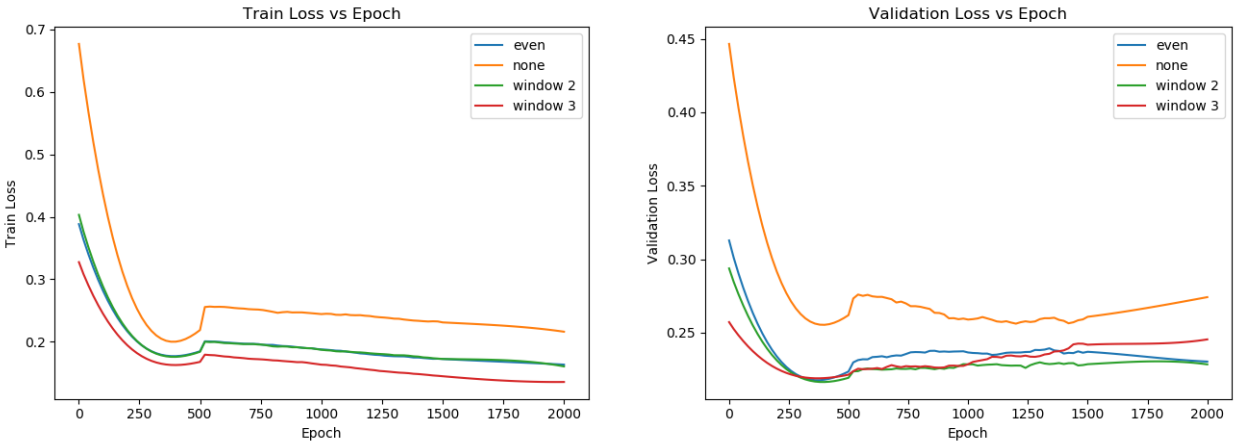


Figure 17: Subsampling of 200 frames with a window of size 3

There is the choice of which subsampling to apply. Either even frames, window 2, window 3 or no subsampling at all. We have calculated the training and validation loss for one of the networks with the same parameters but different type of subsampling. The results can be seen at figure 18.



(a) Training Loss

(b) Validation Loss

Figure 18: Different Subsampling with error calculated over time in an example for the network Martinez Simple

As we can observe, not applying subsampling at all gives worst results than applying it. Between the 3 other subsamplings, window 3 has the lowest training loss. However, in the validation loss, window 2 is has the lowest loss, followed by the even subsampling.

In the table 3 are stated the total number of frames needed for 4 seconds of movement with 50 fps for different types of subsampling.

Subsampling	Total Frames
None	200
Even	100
Window 2	100
Window 3	66

Table 3: Number of frames needed for a movement of 4 seconds of length with a frame rate of 50 fps with different types of subsampling applied

As we can observe, applying even or window 2 subsampling reduces by half the number of frames needed for the same movement, as one in each 2 frames is omitted. In the case of window 3, the total number of frames would be 66.

With what we have seen from the graphics from above, we have decided to apply a window 2 subsampling type for the experiments. The first reason is that in the protocol of [1], they use an even subsampling, and they also have the same Frames Per Second quality in their videos so we would be predicting the same amount of time for the movements.

The second reason is that window 2 has an slight advantage over even subsampling from the graphics. It is also more robust statistically speaking, as apart from the random cropping, we will not always pick the even or the odd frames, but a random combination of odd and even frames for that sample.

Those are the reasons why we have chosen the window 2 subsampling as the subsampling method for the experiments.

4.4 Implementation Details of the Networks & Models

In this section we will talk about the practical implementations of the deep learning networks used for the experiments. First we will address the two input types that we will feed into the deep learning models.

After that, we will introduce the metrics and losses that will be used during the experiments to check the performance of the networks. Finally, we will give the implementation details of the deep learning networks used for the experiments.

4.4.1 Input Types

We have tried two different types of inputs for the networks in this master thesis. The first input type, called **Input Hidden Frames**, is to set the frames that we want to predict to 0, and let the model learn the pose base of the next frame on their own. This input type can be seen at figure 19.

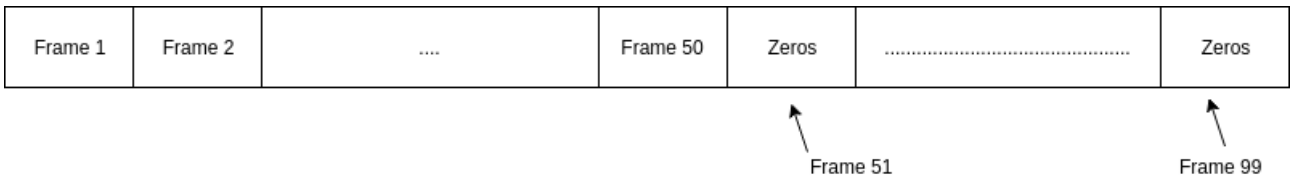


Figure 19: Input type hidden frames. From the frame 51^{th} until the end of the sequence is set to zero.

The second type of input, called the **Input Repeated Frame**, is to copy, from the frame that we want to predict until the end of the sequence this same frame. So for instance if we predict from the 50^{th} then we will make the frame 51^{th}, 52^{th} ... until the end of the sequence, have the same value as the frame 50^{th}. We can see a representation of this input at figure 20.

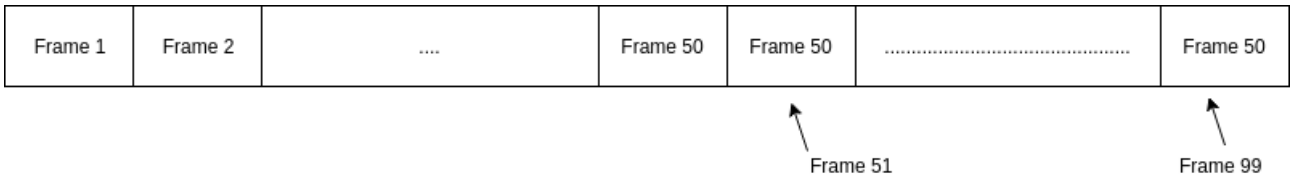


Figure 20: Input type repeated frame 51^{th}. From the frame 51^{th} until the end of the sequence is set to the frame 51^{th}.

In the code, this is done in the file *src/models/martinez.py* at the function *set_input*. In section 5, we will perform experiments with these two different input types to see how they affect the performance of the models. Now we will describe the implementation for the networks that will be used during the experiments.

4.4.2 Metrics & Loss

The metrics & loss used for the training, validation and testing of the deep learning experiments will be discussed in this section. The metrics are defined at the file *src/models/martinez.py* in the *compute_metric*, *compute_metric_test* and *compute_movement* functions. Moreover, the loss is obtained from the *forward* function.

4.4.2.1 Obtaining the Loss The loss of the network is obtained in the *forward* function. Here the loss between the estimated movement and the ground truth movement is calculated. This loss is the criterion that measures the **mean squared error (squared L2 norm)** between each element in the input x and target y. The reason why we chose this criterion is because we are dealing with a **regression problem**, as of a group of 50 frames, we have to obtain another 50 frame values.

Also in this function is where we set the root orientation to 0. This is because in the protocol followed in the baseline paper [1], they set their first six parameters of their data to 0.

However for the SMPL model, only the 3 first parameters correspond to the root orientation, that is why we set them to 0. With the three first parameters set to 0, we calculate the error, and after that, return to their original values for the representation of the movement.

This way the loss of the the models is obtained and is how the model learn during the training. For each epoch, it will try to minimize this error between the 100 frames of ground truth movement and the 100 frames of the predicted movement (50 of them are ground truth as well).

The returned loss of the models each epoch is the same for all the networks except for the network in [9]. This is because, as it can be observed in the figure 7, this network returns 2 losses. So for this network, in order to calculate the total loss we just sum the two obtained networks. This can be seen in *src/models/srivastava.py* in the *forward* function.

4.4.2.2 Obtaining the metrics Once the loss has been obtained, we will calculate different metrics for the evaluation of the experiments. These metrics for us are a measure of how well do the networks perform. For the metrics calculation we have decided to use the **Euclidean** distance because in the protocol of the paper [1] they use this distance for calculating the metrics.

We have 5 different metrics in total:

- Exponential Maps Normalized metric
- Euclidean metric
- Euclidean Predicted metric
- Exponential Maps metric
- Exponential Maps Predicted metric

Please note that for all the metrics mentioned here except the Exponential Maps Normalized metric, the data is first **unnormalized**, as it was previously normalize when feeding the networks.

Moreover, for some of the metric we have calculated the error of only the predicted frames, in our case 50 frames. As for us this is a necessary metric in order to see how well do the models predict human motion. The metrics are obtained in the function *compute_metric* of the file *src/models/martinez.py*.

Exponential Maps Normalized metric This metric is equivalent to the obtained loss from the previous section. For calculating this metric, the Mean Squared Error distance is applied between the obtained movement and the ground truth movement for the whole 100 frames of the sequence. In this case the data remains with the normalization applied, as the unnormalization is necessary only to represent the movements with the SMPL renderer.

Euclidean metric This metric calculates the error measured in euclidean distance between the input sequence, and the predicted sequence. The sequences in this case are of 100 frames of length, composed by the first 50 frames acting as ground truth and the last 50 frames acting as the predicted frames. The format of the input sequence is given in **Euler angles**. Also the unnormalization process is applied tom comply with the protocol in [1]. This metric takes into account both the frames that were predicted and the ground truth frames.

Euclidean Predicted metric This metric calculates the error measured in euclidean distance between the input sequence, and the predicted sequence. The sequences in this case are of 50 frames of length, composed by the last 50 frames acting of the sequence as the predicted frames. The format of the input sequence is given in **Euler angles**. Also the unnormalization process is applied to comply with the protocol in [1]. This metric takes into account only the predicted frames.

Exponential Maps metric This metric calculates the error measured in euclidean distance between the input sequence, and the predicted sequence. The sequences in this case are of of 100 frames of length , composed by the first 50 frames acting as ground truth and the last 50 frames acting as the predicted frames. The format of the input sequence is given in **Exponential map angles**. Also the unnormalization process is applied. This metric takes into account both the frames that were predicted and the ground truth frames.

Exponential Maps Predicted metric This metric calculates the error measured in euclidean distance between the input sequence, and the predicted sequence. The sequences in this case are of 50 frames of length, composed by the last 50 frames acting of the sequence as the predicted frames. The format of the input sequence is given in **Exponential map angles**. Also the unnormalization process is applied. This metric takes into account only the predicted frames.

4.4.2.3 Obtaining the Movement Representation

This is done in the function ***compute_movement*** of the file *src/models/martinez.py*. It is similar to the ***compute_metric*** function. The only difference is that in this function we reconstruct the first 3 parameters of the pose which we set to 0 before. Thus we restore their original values in order to display the movement. Moreover, the data has to be unnormalized as it was normalized when it was read from the hdf5 file. This is done by multiplying by the standard deviation and adding the mean of the dataset.

The movement is represented by using the renderer of the SMPL model [7]. In our code, this is located at *src/train.py* in the *_display_shape* function. This function receives the ground truth and predicted movement that we want to represent as an argument. It also receives the shape parameter which uniquely identifies each individual of the human3.6m dataset.

With the shape parameter, we can identify whether the subject is a male or a female. If the subject is a female we will use the SMPL model for females whereas if the subject is a male we will use the SMPL model for males. Then the function, using the rendering method of the SMPL model, represents each of the 100 frames of the ground truth and predicted movements side by side and stores them in a .gif file. We can observe an example of this in the figure 21.

Figure 21: Example of Ground Truth Movement vs Estimated Movement with the use of the SMPL model [7]

Now we will enter in the implementation details of the networks used for the experiments of this master thesis.

4.4.3 Tested Networks

4.4.3.1 Martinez Network The deep learning model from [1], as seen it was seen in figure 6 from section 2. This model consists on an encoder network, followed by a decoder network that feeds its own predictions.The decoder also has a residual connection, which forces the RNN to internally model angle variables.

The interesting part about this network it skips frames connections between frames on Recurrent Neural Network. This type of connection was not provided within the official implementation of Pytorch as is not standard. So we have had to implement this type of connection manually with the use of RNN Cells.

To implement this network, as shown in 6, first we have to implement the encoder network. This encoder receives as an input the first n frames (ground truth) of the movement, as well as the hidden state initialized with zeros.

Regarding the skip frame connection to the next frame part is not a standard procedure. The way we have implemented this connection is with the use of the RNN Cells. For each of the layers of the RNN network, we save its corresponding hidden states. And starting from frame the first frame to predict the 51^{th} until the end of the sequence, we apply a the RNN Cell for each layer l of the network.

The RNN Cell has as an input the output frame, and the hidden states of the corresponding layer l of the RNN network. with its corresponding hidden states. Finally, the output frame will be the sum between the output frame n of the network plus the input frame n, thus resulting in a skipping frames connection between layers.

Then the residual connection is applied by adding the output of the RNN network with the input of the sequence. At the end, predicted frames are concatenated with the ground truth frames and this is what is returned from the forward function.

4.4.3.2 Simple Residual This network is the first network that we have implemented. The network is similar to the Martinez network [1]. Its architecture can be seen in the left part of the figure 11. In this network, first we initialize the hidden states to zero. After that the Recurrent Neural Network, in our case, a Gated Recurrent Unit (GRU) or a Long Short Term Memory (LSTM) is applied. Pytorch makes this relatively simple with the commands *nn.LSTM* and *nn.GRU*, which apply a LSTM or GRU RNN to the given input sequence.

After applying the RNN, we use a fully connected layer to transform the output of the network from the number of hidden units to the dimension of the output. The dimension of the output in our case is the pose vector parameter of dimension 1×72 per each frame of the input sequence, thus in total we will have an output of 100×72.

4.4.3.3 Srivastava Now we will analyze the network in [9] which can be seen at figure 7.

To implement this network, first we implement the encoder network. To do this, we simply initialize the hidden states to 0, and pass the ground truth of the input sequence, that is the first 50 frames as an input. The hidden states obtained with the encoder network will be the hidden states that we apply to the future prediction network and the input reconstruction network.

Now for implementing the future prediction network, we just call the *nn.LSTM* command with the 100 frames of input sequence as an input, and the hidden states of the network are the hidden states obtained with the encoder network. And similarly to the other networks, for each frame of the sequence obtained, we apply a fully connected layer to decode the input and adapt it to the dimensions desired.

In order to implement the input reconstruction network, we first flip the ground truth input sequence frames so the 1^{st} is the 50^{th}, the 2^{nd} is the 49^{th} and so on ... Then we pass to the LSTM the flipped ground truth sequences as an input and the hidden states of the network are the hidden states obtained with the encoder network. Finally, we apply a fully connected layer to decode the input and adapt it to the dimensions desired.

Please note that this model returns two losses instead of one. The loss of the reconstruction of the input sequence and the loss of the predicted sequence. In the model part of the code, when dealing with this model, we simply sum both of the losses to calculate the error. This way when minimizing the error, we take into account both the reconstruction of the input and the future prediction part.

4.4.3.4 Encurrent Recurrent Decoder (ERD) This network has been implemented following the paper of [4]. The architecture of this network can be seen at the figure 8.

The architecture of this network is relatively simple, with first an encoder part, then a RNN and at the end a decoder. The implementation has been done in the following way.

For the encoder part, we first declare a Pytorch Sequential structure in which two linear neural networks are applied. The first one has the input dimension of the input of the networks and as output dimension the number of hidden units of the RNN network. Then for adding non-linearity we apply a LeakyRelu function activation. At the end. we apply another linear neural network with dimensions equal to the number of hidden units of the RNN network.

For the Recurrent Neural Network, either a LSTM or GRU is applied. Finally, the decoder part, is symmetric to the encoder and a Leaky Relu activator is applied as well.

4.4.3.5 Encurrent Recurrent Decoder (ERD) Residual This network is from our own implementations. It is based on the ERD decoder from [4]. What we add with respect to the original ERD network is a **Residual Connection**. This residual connection is applied between the input sequence and the last layer of the Recurrent Neural Network. Its implementation is similar to the ERD, but instead of returning just the output of the ERD, we sum the input of the sequence to make the residual connection effective.

After describing the implementation of the networks and the practical framework in general, we will proceed to describe the experiments performed during this master thesis and its results.

5 Experiments

Here we will cover some of the experiments that were performed trying to emulate the results on the papers that we have seen in the section 2 and our proposed models in section 3.

The dataset which contains the data is a preprocessing of the human3.6m dataset [6]. The preprocessing adapted the data to the format of the SMPL model, obtaining the pose and shape parameters per frame.

Please note that in the images with plots, a smoothing has been applied in order to obtain a better visualization of the plots. This filter is the ***savgol_filter*** from the *scipy* library. The metrics that are used in the experiments were introduced in section 4.

In this section we will first take a look at the hyperparameters used in the networks of the experiments. After, we will explain the validation results obtained with different configurations of the networks described in section 4. Then, we will make a comparison between the different networks to determine which one gives the best results for the human motion prediction problem. Finally, we will show some examples of prediction with videos on the wild. That is, working with videos which are not from the human 3.6m dataset and not knowing the pose and shape parameters of each frame of the movement.

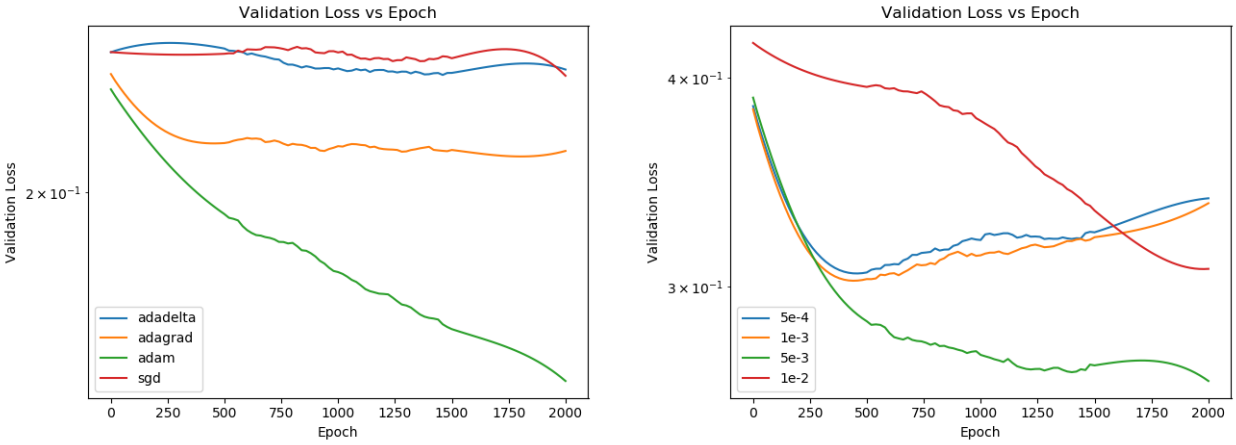
Let's start talking about the tested hyperparameters of the network.

5.1 Hyperparameters

5.1.1 Learning Rate & Optimizer

We have tested several learning rates for all the networks used in these experiments. One learning rate that has achieved good results from all the ones we have tested is $5e^{-3}$. With this learning rate, there is a quick decay at first in the loss, with a big slope, and after that it stabilises, this can be seen at figure 22 part (a). However any value between $1e^{-3}$ and $5e^{-3}$ is a good value for the learning rate.

Because we use the **adam** optimizer, it does not require as much hyper-parameter tuning as with other optimizers. We chose the adam optimizer because it gives better results in most of the cases that when using other optimizers and we do not have to tune the hyper parameters that much as we would have to do with other optimizers. The results can be seen at figure 22 part (b).



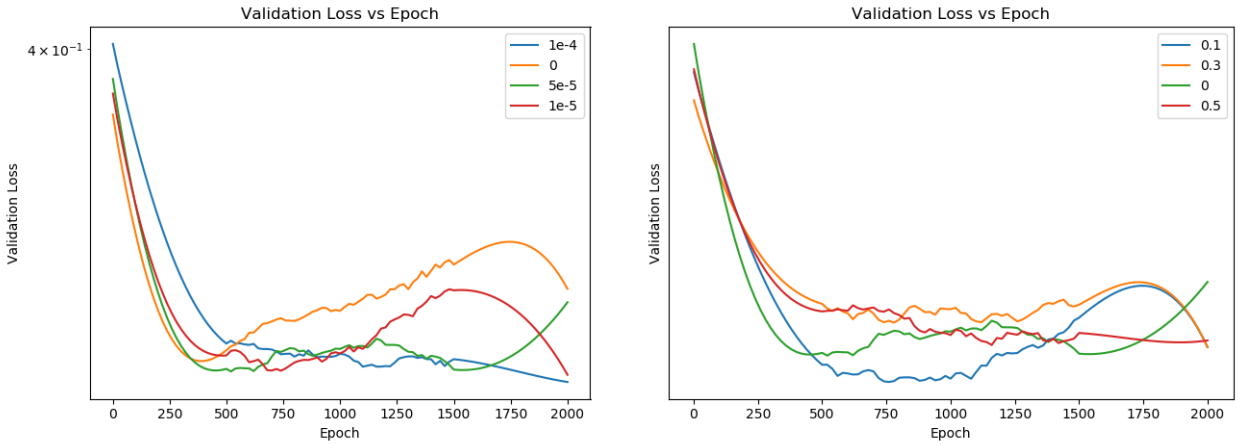
(a) Optimizers

(b) Learning Rates

Figure 22: Different Tested Learning Rates & Optimizers of our Simple Martinez model

5.1.2 Regularization & Dropout

Now for the hyperparameters: the **regularization L2** (parameter weight_decay in Pytorch) and the **dropout**, we concluded after several test, that one of the best values for the weight decay was between $5e^{-5}$ and $1e^{-5}$. Moreover, the dropout was not significant in this case. We can observe these results at the figure 23. We haven chosen the dropout to 0, as there was not significant difference in the results.



(a) L2 Regularization (b) Dropout

Figure 23: Different Tested L2 Regularizations & Dropouts of our Simple Martinez model

5.1.3 Number of layers and Hidden Dimension

Regarding the number of layers and hidden dimension of the RNN networks, we have tested several configurations between 2 and 4 layers, and 32 to 2048. We did not observe big differences between using one configuration or another although for all the tested networks, having a number of layers of 2 or 3 and hidden units of the RNN between 512 and 1024 gave the best results.

This is because the architectures of the tested networks are relatively similar to one another. All of them use RNNs for the human-motion prediction, and most of them have residual connections so the same configuration of parameters can be applied to all of them.

5.1.4 Number of Epochs

Regarding the number of epochs, the experiments have shown that **2000 epochs** is enough to show whether the deep learning model is learning or not. From those 2000 epochs, the **first 100 epochs are with no decay** and the other **1900 epochs have a linear decay**. That means applying the following formula, the learning rate lr at the epoch e is:

$$lr_e = lr_{e-1} - \frac{lr_{e-1}}{\text{number of decay epochs}}$$

So at the epoch 2000, the learning rate will be 0. This is done in order to make the optimizer learn steps smaller. With this we also try to find a better minimum solution as well as accelerating the convergence of the solution.

5.1.5 Used Hyperparameters

After doing some test with the Hyperparameters we have decided that the hyperparameters which we will execute the experiments with the different networks are:

Optimizer	Learning Rate	Regularization L2	Epochs No Decay
adam	1e-3	5e-5	100

Epochs Decay	Subsampling	Layers RNN	Hidden Units RNN
1900	window200	3	512

Sequence length	Starting Frame Prediction	Distance
99	51	euclidean

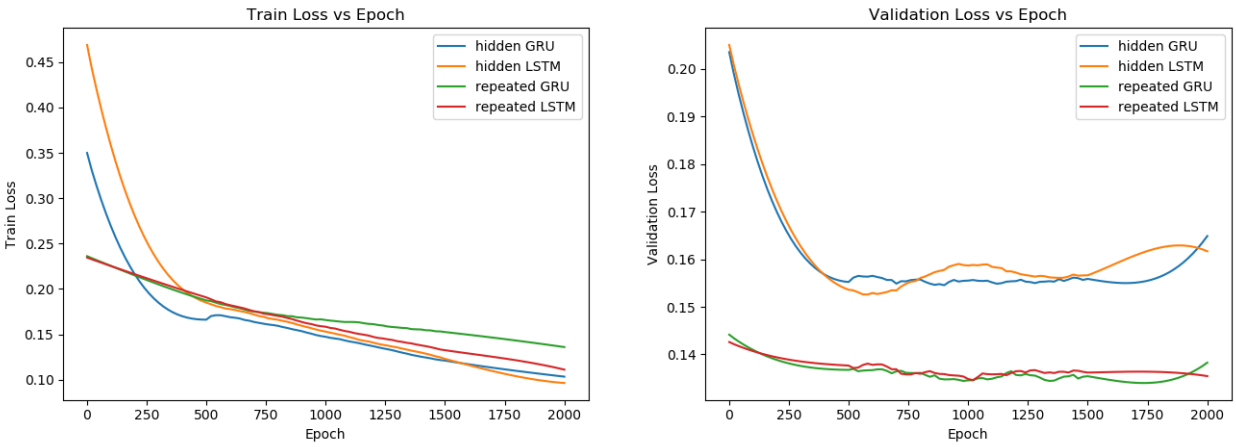
Table 4: Hyperparameters used in the implemented networks

The reasoning behind this is that the networks that we have tried, have a similar architecture in the sense that all of the use Recurrent Neural networks and the data the received as an input is similar for all of them. So the most important parameter is picking up a learning rate that adapts well to the learning of all such as the learning rate $1e^{-3}$. And the same optimizer which in this case is Adam so other hyperparameters do not require much hyperparameter tuning.

5.2 Simple Residual

For this network, the two different types of inputs are used, both the input hidden frames and the input repeated 51^{th} frames. Moreover we use two types of Recurrent Neural Networks (RNNs). The Long Short Term Memory (LSTM) and the Gated Recurrent Unit (GRU). The GRU differs from the LSTM in the sense that the GRU unit controls the flow of information like the LSTM unit, but without having to use a memory unit. Performance wise the GRU is computationally more efficient because its structure is simpler than the LSTM structure.

First, if we take a look at the training loss vs validation loss from the figure 24. It seems that the networks with input type repeated frame 51^{th} gives better results that the experiments with the input hidden frames.



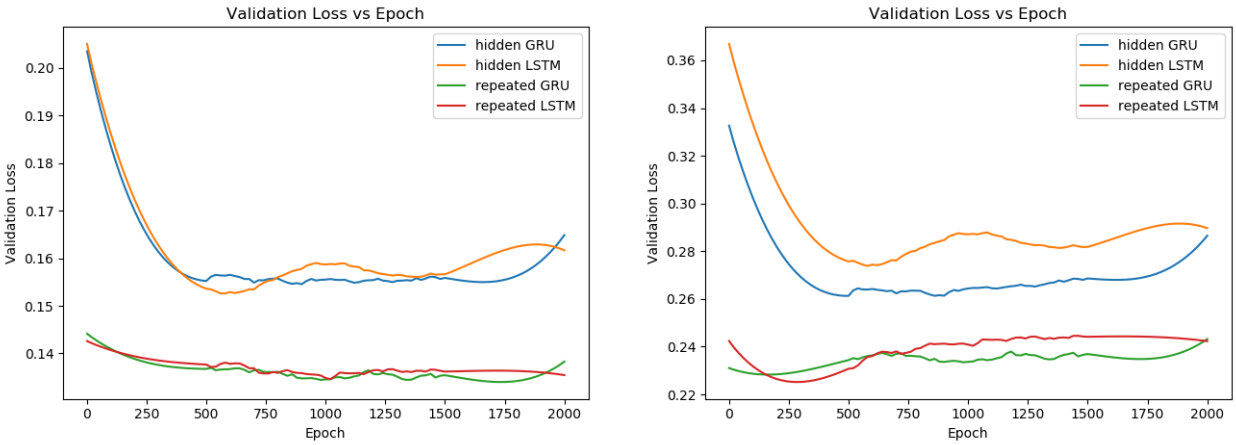
(a) Training Loss

(b) Validation Loss

Figure 24: Simple Residual model Training vs Validation

Now if we center just in the validation part, we can observe, from the figure 25, that the GRU gives slightly better results than the LSTM whenever the input type is the same. Also, the repeated frame 51^{th} gives better results than the input hidden frames.

We part from the premise that for each frame, we want to learn the difference in movement between the frame n and the frame $n+1$. If we already have the frame n then it will be much easier to calculate the movement.



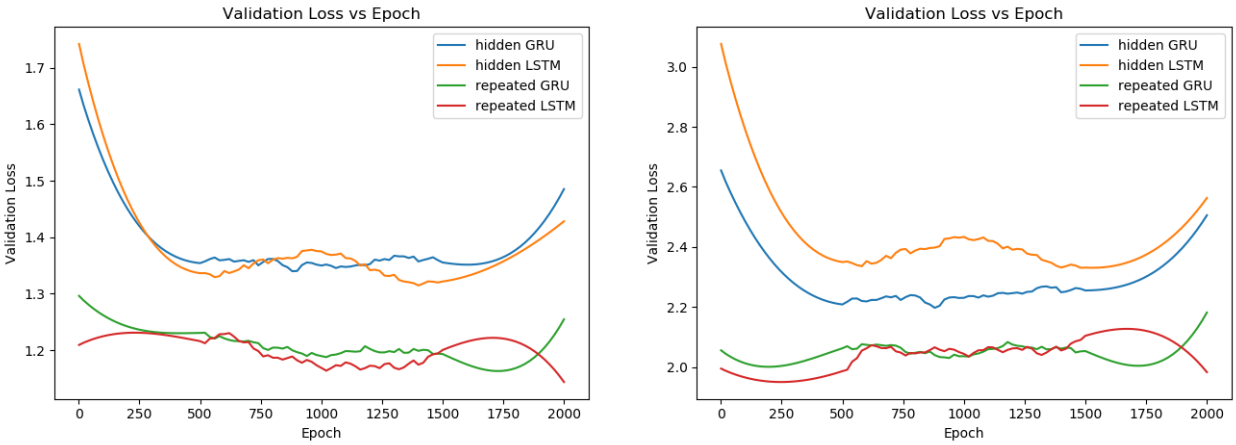
(a) Validation Expmap Loss

(b) Validation Expmap Predicted Loss

Figure 25: Simple Residual model Expmap vs Expmap Predicted

This is because when the base pose frame n is not given for the frames to predict, it takes longer to the model to predict the movement, as it has no reference of the base pose frame n. On the contrary case, when feeding the base pose frame n to the network, the network does not have to predict this, and will produce a more similar result as the ground truth movement, as long as the movement is fluent and gradual.

Regarding the Euler Predicted Loss, it behaves similarly as when using exponential maps as angles. Please note that considering only the predicted frames gives a higher error than considering the sequence as a whole. This is because, when the sequence movement is whole, the 50 first frames acts as a ground truth, as we feed the model 2 seconds (50 frames) and the rest is predicted. This can be seen at figure 26.



(a) Validation Euler Loss

(b) Validation Euler Loss

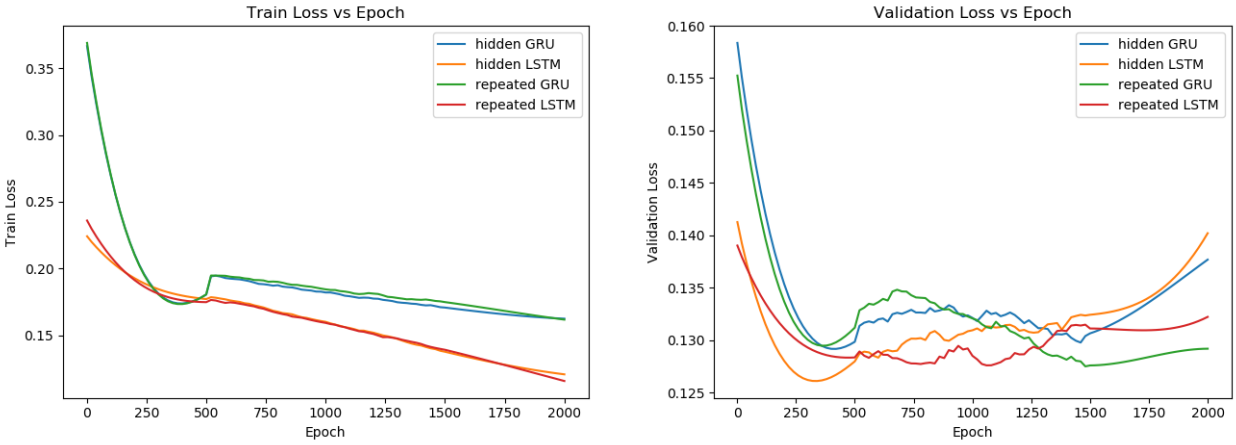
Figure 26: Martinez Simple model Euler vs Euler Predicted

For the comparison experiments between different network architectures in section 5.6 we have chosen, for the tested configurations of the network Simple Residual, the network with input type repeated frame 51^{th} and Recurrent Neural Network of LSTM, as is one of the best in performance from these 4 networks.

5.3 Martinez

Now let's have a look at the experiments from the network in [1]. Again, we first compare the training loss against the total validation loss. As we can observe from the the figure 27, the two GRU networks and the two LSTM networks have a really similar training loss, being lower in the case of the LSTM. However, the netwotk

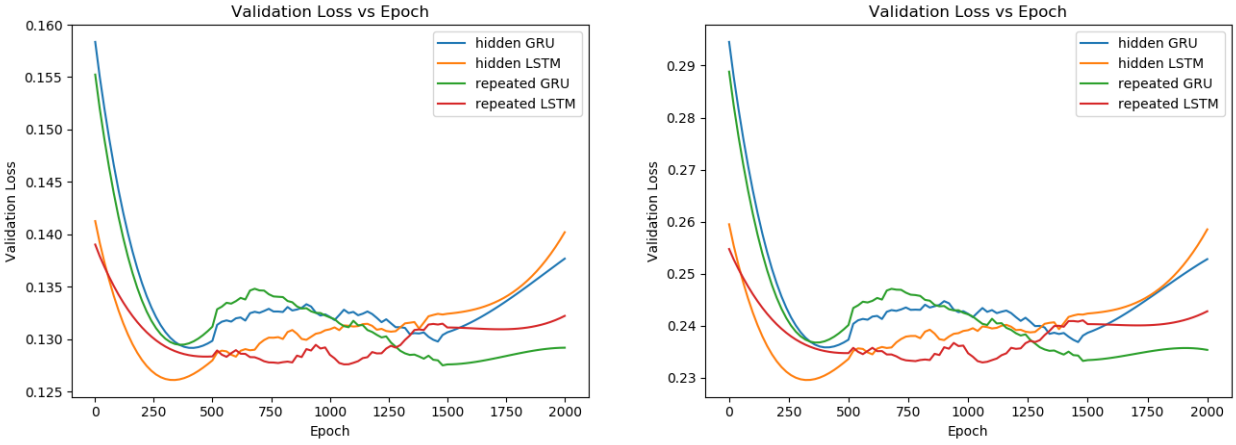
with less total validation loss is the repeated GRU network, even though its training error is higher than in the LSTM networks.



(a) Training Loss (b) Validation Loss

Figure 27: Martinez model Training vs Validation

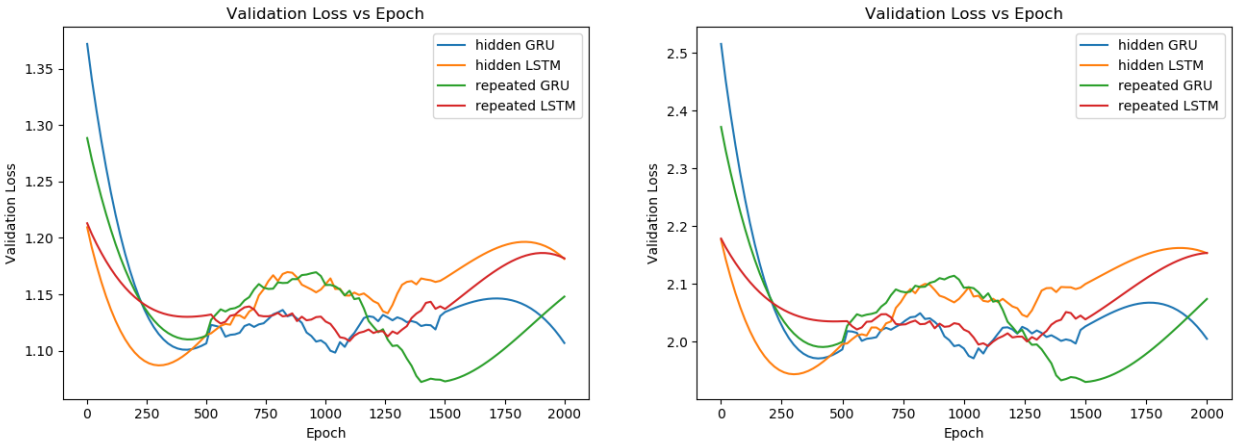
Regarding the validation losses in exponential map format we can see that the network with input repeated frames and Gated Recurrent Unit (GRU) as RNN seems to perform better than the other configurations.



(a) Validation Expmap Loss (b) Validation Expmap Predicted Loss

Figure 28: Martinez model Expmap vs Expmap Predicted

However if we take a look at the results of the validation loss using Euler angles instead of angles in exponential map format we can see how the network with input hidden frames and GRU seems to perform slightly better than the network with repeated frame 50^{th} and GRU. What we can clearly see is that these two clearly perform better than their LSTM counterparts.



(a) Validation Euler Loss

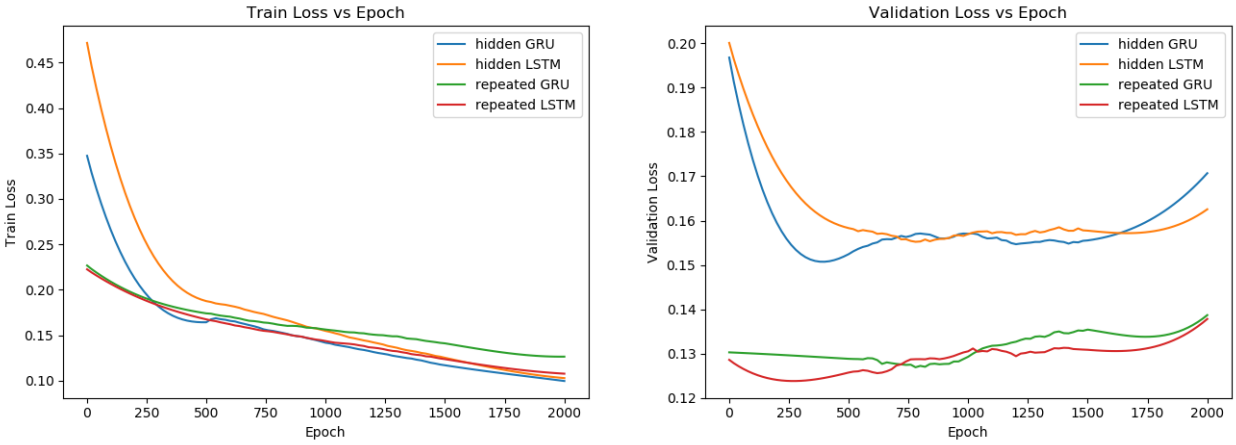
(b) Validation Euler Loss

Figure 29: Martinez model Euler vs Euler Predicted

For comparison experiments between different network architectures we have chosen the network repeated GRU, as is one of the best in performance from these 4 networks and is the same implementation as the network in [1] (although not exactly the same preprocessing and protocol as previously mentioned).

5.4 Srivastava

The experiments performed in the implementation of the paper [9], are covered in this section. If we compare the training vs the validation error at figure 30 we do not see a big difference in the training error whereas in the validation error whenever we use the repeated frame 50^{th} input, the error is considerably less than when using the hidden frame input.

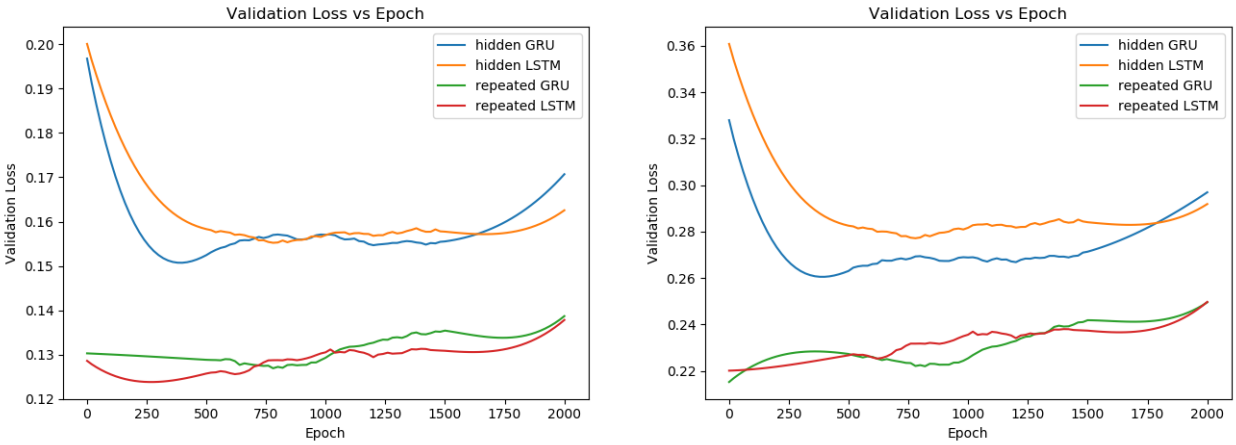


(a) Training Loss

(b) Validation Loss

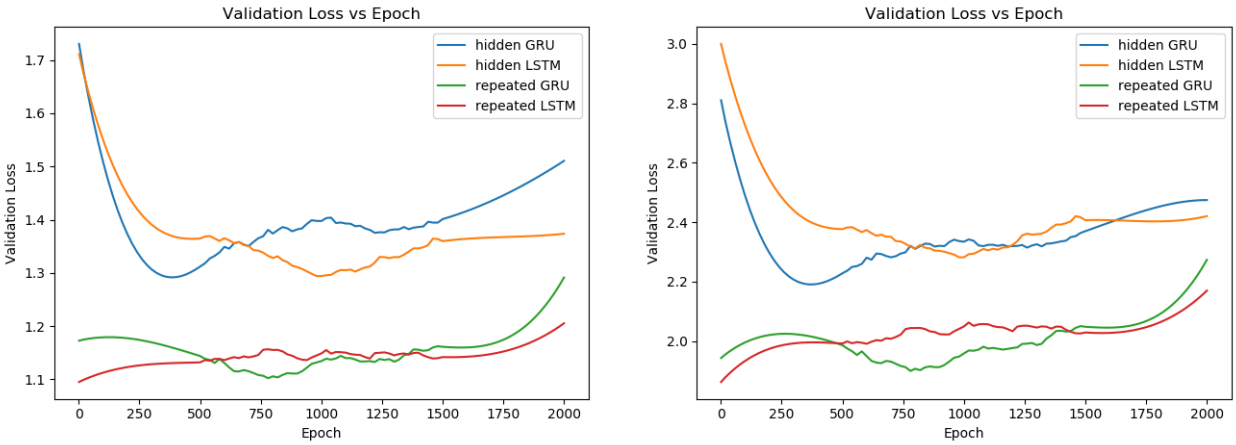
Figure 30: Srivastava model Training vs Validation

If we take a look at the graphics with exponential maps and Euler angles at figures 31 and 32 respectively, we can see that the difference is similar.



(a) Validation Expmap Loss (b) Validation Expmap Predicted Loss

Figure 31: Srivastava model Expmap vs Expmap Predicted



(a) Validation Euler Loss (b) Validation Euler Loss

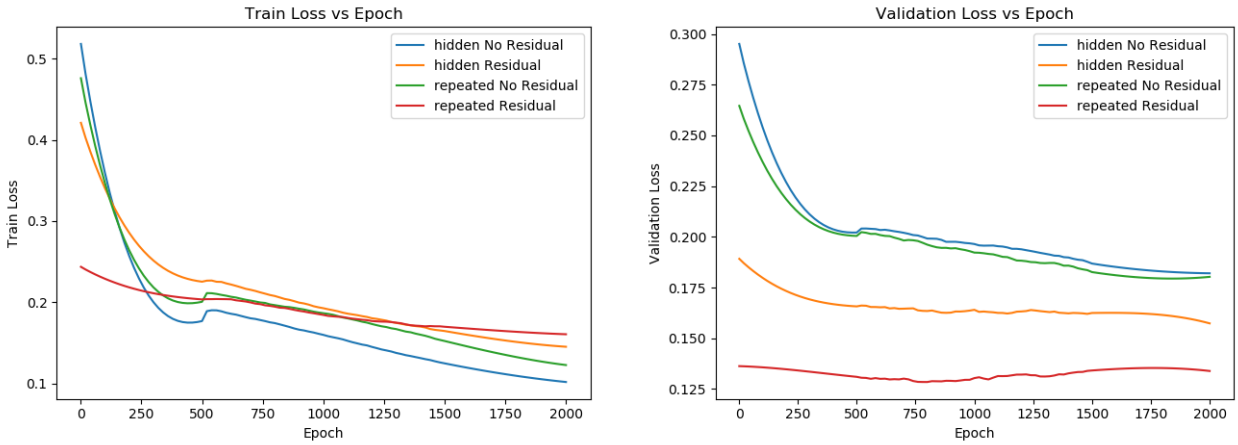
Figure 32: Srivastava model Euler vs Euler Predicted

The reason that repeating the frame 50^{th} as an input gives better results than passing zeros is because the encoder present in this network (see figure 7). If we feed the encoder a sequence containing zeros in the frames, the network will reconstruct that the input is mostly composed of zeros, and therefore it will learn worse than if it learnt the frame 50^{th} of the sequence as it has a base pose to start learning with.

5.5 ERD & ERD Residual

Now we perform similar test for the Encoder Recurrent Decoder network based on the paper from [4] that can be shown in the left of the figure 8. We also perform experiments for our network ERD Residual. What is new about this network is that we have added a **residual connection**, similarly to the network from [1]. A Leaky ReLU activator function was also used instead of the normal ReLU that is used in that [4] .In the experiments of this section, we will center in how adding or not a residual connection, affects the performance of the network.

As we can observe from the figure 33, although the training error of the **repeated input Residual** network is not the lowest, it has has the lowest of the validation error.

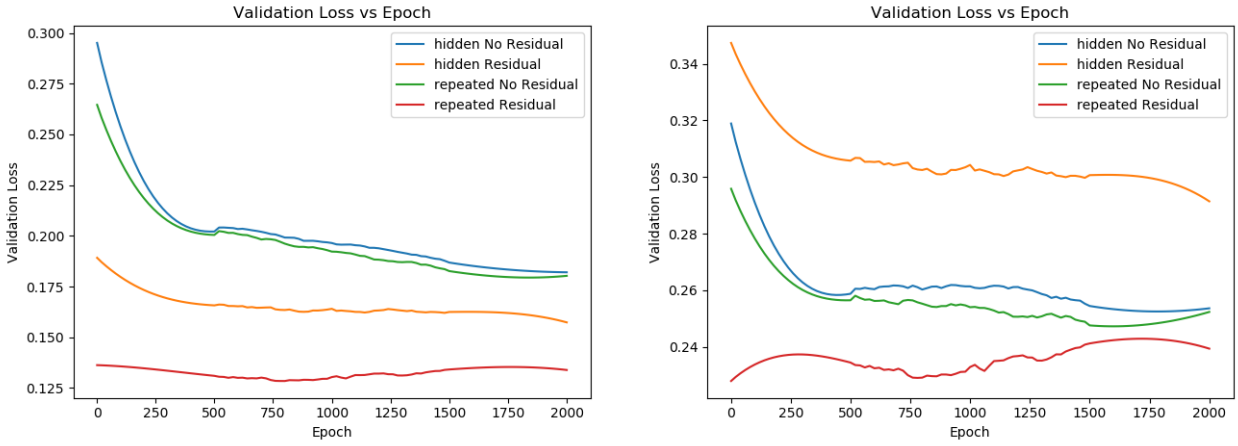


(a) Training Loss (b) Validation Loss

Figure 33: ERD model Training vs Validation

In figure 34 we can see how having an input of **repeated frames** type and a residual connections clearly outperform the other network types. This is because of the structure of the network. Moreover, for the predicted frames plot (b), the **residual input hidden type** network performs worse than having an input of **repeated frames**. This is because in a residual connection, the input frames are really important for the prediction of the future. So, if we feed the residual connection a zero as an input frame, it will have a pretty big disadvantage with respect to feeding the 50^{th} frame.

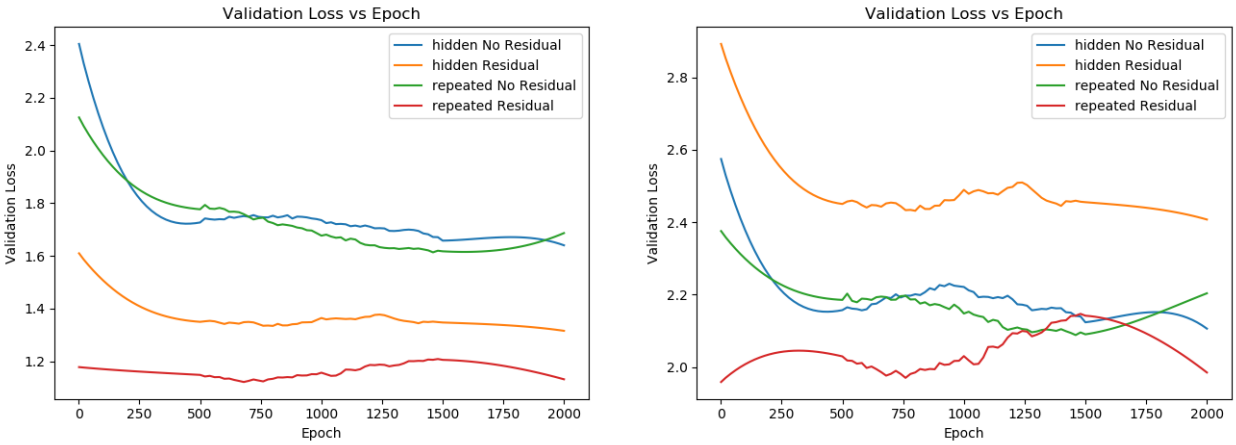
In order words, networks with residual connections tend to go more towards the mean of the estimated frames because they have information regarding the base pose of the movement, so they will be better at predicting fluent and smooth movements (such as walking) as the frames all more similar to each other than for instance having a discussion, where abrupt movements can be made.



(a) Validation Expmap Loss (b) Validation Expmap Predicted Loss

Figure 34: ERD model Expmap vs Expmap Predicted

Now if we take a look at the figure 35, we can see that there is a gap between the total loss of the networks hidden Residual (yellow) and repeated Residual (red). This is produced because, the prediction of the hidden Residual is much worse than the other three, as is a residual connection but we are adding zeros at first to it. So it take this network much more effort to learn than with any of the other approaches.



(a) Validation Euler Loss

(b) Validation Euler Loss

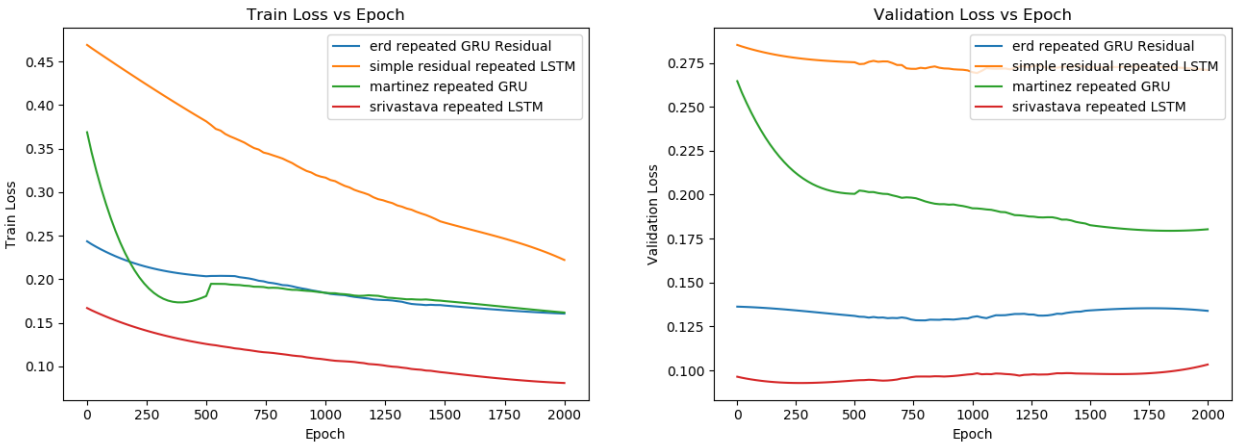
Figure 35: ERD model Euler vs Euler Predicted

For comparison experiments between different network architectures we have chosen the network ERD repeated residual with GRU as Recurrent Neural network, as it performs better that its ERD hidden residual. Moreover, it performs considerably better that the network from he original ERD paper [4] as the figures have shown.

5.6 Comparison between different networks

5.6.1 General View

Now let's compare different networks between each other to see which one gives the best results in the testing part. If we take a look at figure 36 we can see the training and validation loss of this 4 networks. Simple Residual and Martinez have a similar Training Loss, Srivastava has the lowest training loss and ERD with residual connection has the highest loss.

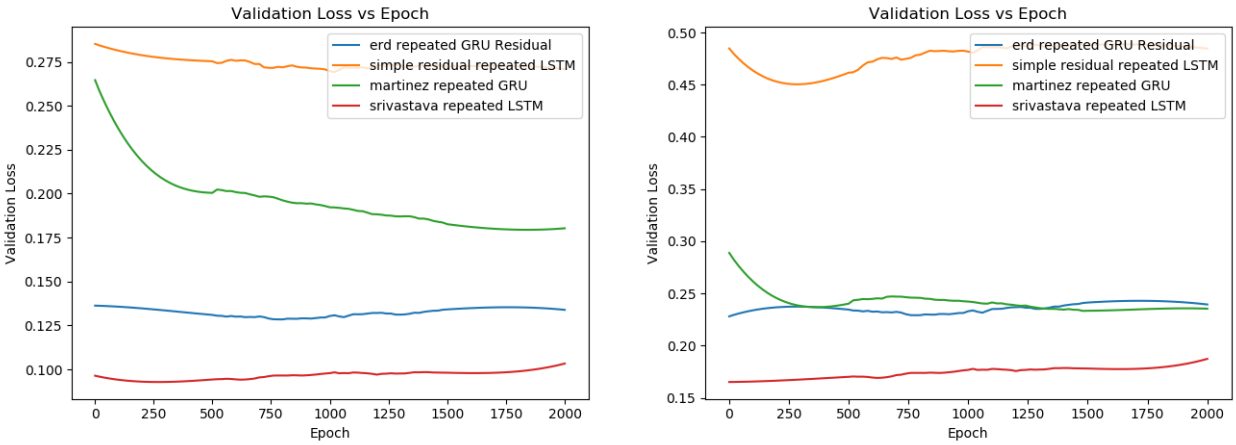


(a) Training Loss

(b) Validation Loss

Figure 36: Comparison models Training vs Validation

In the figure 37 is observed that although Martinez has a higher Validation Expmap loss than Residual Simple network, if we only take the predicted 50 frames and calculate the loss, the losses of those two networks are similar.

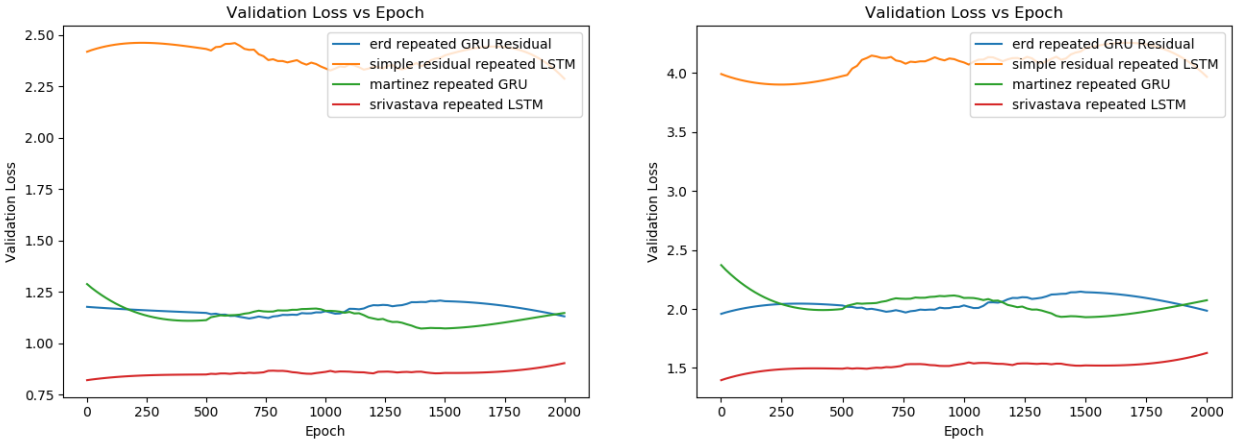


(a) Validation Expmap Loss

(b) Validation Expmap Predicted Loss

Figure 37: Comparison models Expmap vs Expmap Predicted

Regarding the Euler loss 38 is a very similar situation. With the higher predicted error for ERD residual, Martinez and Simple Residual a similar loss and Srivastava the lowest validation error. This can be deceiving because the network Srivastava, based in the network of the paper [9] has two outputs as an error: the future predicted and the input reconstruction. For the metrics we only have considered the future predicted output as this metrics tells how well the motion has been predicted. That is the reason why the Srivastava error plot is lower than the other two networks when its performance in human-motion-prediction is similar as we will see later.



(a) Validation Euler Loss

(b) Validation Euler Loss

Figure 38: Comparison models Euler vs Euler Predicted

Remembering that one of the objectives of this master thesis is, to try different networks for human motion prediction with the SMPL [7] model format. The performance of the networks is measured used the paper [1] as a baseline.

To do the testing of the networks, although we try to use a similar preprocessing of the human3.6m dataset as in [1], we willingly **do not** follow exactly the same protocol. The reason is that in their protocol they only do the tests with 8 samples per action. That is, in our opinion a not high enough number of samples for the experiments to be **statistically robust** when performing the experiments. In our experiments, we use 1000 samples per action to test the networks.

Our preprocessing is also different from their preprocessing. First of all, they have 32 joints whereas we have only 23 joints. Moreover, we explained in section 4 that the dataset in which we perform the experiments was

extracted from the original human3.6m dataset [6] by using the method in the paper [8] to extract the pose and the shape parameter. This method adds some noise to the data produced by small error between the predicted shape and pose and the real shape and pose of the person.

Although not having obtained exactly the same result as the paper (table 5 [1], we have achieved similar number in Euclidean Euler angles format and even in some movements, such smoking, better results. Those are the reasons why we will use as a baseline the implemented version of the paper, rather than the results that they obtained with their own preprocessing and protocol of data. In our opinion this is acceptable because we are working with different data, as we use the SMPL model format while they use data directly extracted from the human 3.6 m dataset.

	Walking				Eating				Smoking				Discussion			
miliseconds	80	160	320	400	80	160	320	400	80	160	320	400	80	160	320	400
Martinez Repeated GRU	0.37	0.64	1.04	1.17	0.54	0.84	1.3	1.5	0.27	0.45	0.73	0.85	0.43	0.69	1.07	1.2
Martinez Original	0.27	0.47	0.70	0.78	0.25	0.43	0.71	0.87	0.33	0.61	1.04	1.19	0.31	0.69	1.03	1.12

Table 5: Martinez Original vs Implemented [1]

We can represent the mean error per predicted frame in Euler angles to see how much does the error increase with each of the networks. In general, in human-motion prediction, the models developed for prediction can be measured in short-term prediction and long-term prediction. Short-term here is considered 10 frames or10 frames /25 frames per second $= 0.4$ seconds prediction and long-term longer than 0.4 seconds. As we are predicting 2 seconds, short-term will be from 1 to 10 predicted frames and long-term will be from 11 to 50 (50 frames or 50 frames /25 frames per second $= 2$ seconds) of predicted frames.

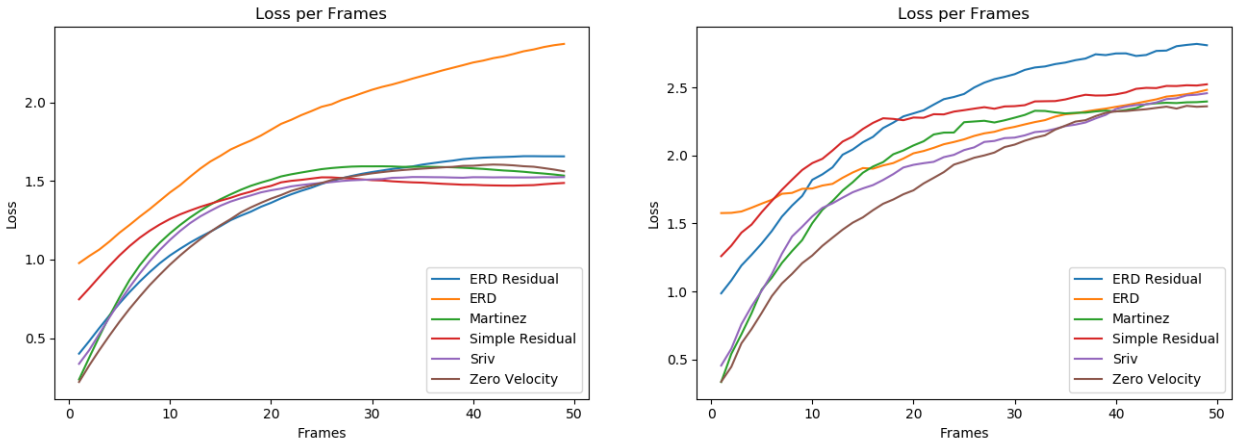
5.6.2 Long-Term Prediction Results

For this section we have added a baseline called **Zero velocity** that just copies the last known frame (50^{th} in our case) as the predicted frames and calculate the error. In other words, we set an input of repeated frame 50^{th} and do not estimate for this baseline but just calculate the error with respect to the ground truth movement while maintaining the pose in the frame 50^{th} for all the predicted frames. This is done in order to see how do the models tend towards the mean pose in the long run.

The results of the human motion prediction for the movements walking, eating, smoking and discussion for the long-term prediction are shown in figure 39. We have chosen these 4 movements because in the baseline [1] they are chosen as well. In this figure, the Walking error is represented on the left and the testing error on the right. In case of walking, the ERD network [4] with no residual connection is considerably worse in the long-term than the rest. However if we add a residual connection the error is considerably lower. This shows the importance of the residual connections for movements that are smooth and fluid such as walking. Because by adding the input to the output, for the deep learning network will be easier to predict the movement.

Now let's take a look at the eating part of figure 39. The eating movement is not a fluid and smooth movement such as walking. It involves more complex movements, such as for instance cutting a fillet or eating a bowl of soup. In this case we notice that networks with residual connections such as ERD Residual obtain a bigger error than the ERD network with no residual connection in the long-term.

The rest of the networks behave similarly although the test error for eating is considerably higher than the error for walking. In general, human-motion predictors will have it easier to predict one-way (non starting to walk suddenly backwards or walking in the other direction), smooth movements, This is because the human-motion predictors for the long-term prediction tends to converge to the mean pose, as is shown in papers [4] and [1].

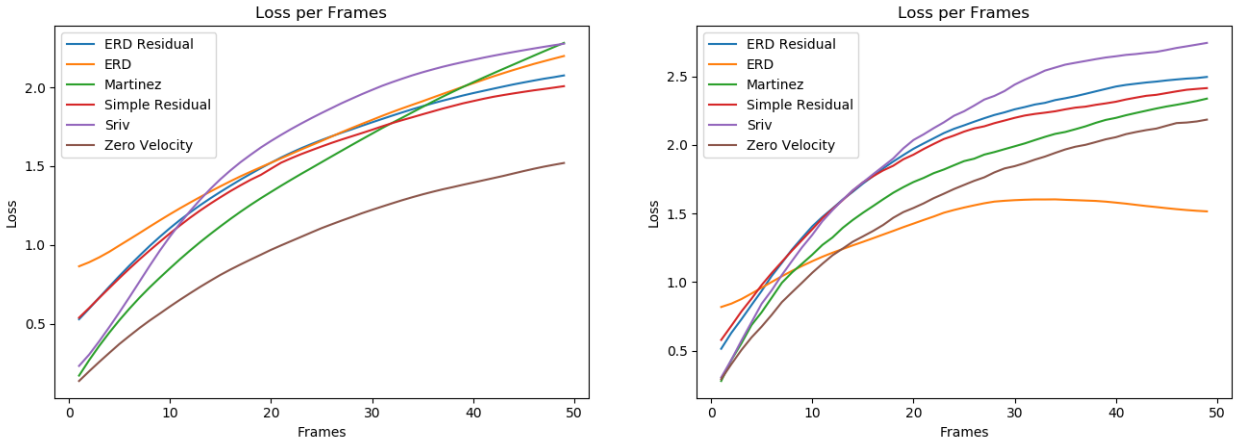


(a) Testing Loss for Walking

(b) Testing Loss for Eating

Figure 39: Long-Term Testing Loss per number of frames of different networks: Walking & Eating

By looking at figure 40 we can see how does the error of the prediction increases when predicting more frames for the movements smoking and discussion. Again, the network ERD from [4] is the best for the long-term prediction in the discussion movement. Once the 20 frames of prediction have been reached it is clearly the best in the figure. Regarding the other networks, the Martinez network outperforms the Srivastava networks in the long-term prediction.



(a) Testing Loss for Smoking

(b) Testing Loss for Discussion

Figure 40: Long-Term Testing Loss per number of frames of different networks: Smoking & Discussion

We can also observe, that in those 4 movements, the zero velocity, or in other words, passing the last known frame as the predicted sequence is the best in the short-term, run. As in the space of a few frames the ground truth frames will not differ too much from the repeated frame $50^{[th}$ as long as the movements are not extremely abrupt and faltering. However, for the long-term prediction, its error increases as more frames are predicted with respect to the other networks. This makes sense because after 2 seconds the movement will most likely not have the same pose as before unless is an almost static movement such as sitting. In fact, in the case of the movement smoking, this is a movement which is pretty static, where the subject stands still not nearly moving while they take a puff of a cigarette. This is way in this case the difference between the zero velocity and the networks is bigger than with the other more dynamic movements.

For the long-term prediction, we have seen that for the network ERD [4] although for the movement walking gives the worst results, for the movement discussion it gives the best results. The ERD network is the only network for all the considered networks that do not have a residual connection. So for smooth continuous

movements such as walking it gives worst results as in this movement feeding the input to the output of the network gives good information. On the contrary, when the movement is somewhat abrupt as in discussion where aggressive hand gestures are made, feeding the input to the output of the network is not that good as the pose of the frames can suddenly change greatly from one frame to another.

What we have seen for the results on the long-term prediction is that the networks tend to predict to the mean pose. We have to take into account that the human-motion prediction is a problem with a high-dimensionality. In our data, we dispose of 23 joints, but each of those joints has a degree of freedom in angles from 0 to 2π to rotate between one or more axis when performing a movement. Because the human movement has an stochastic nature, its prediction has a high complexity, and that is why it is not obvious to measure how well does a network performs.

5.6.3 Short-Term Prediction Results

Now we will focus on the results of the tests for short-term predictions, that is predicting from 1 frame to 10 frames. The results for the obtained mean error for predicting 2,4,8 and 10 frames for the movements: Walking, Eating, Smoking and Discussing can be seen at the table 6. We have chosen these 4 movements because in the baseline [1] they are chosen as well.

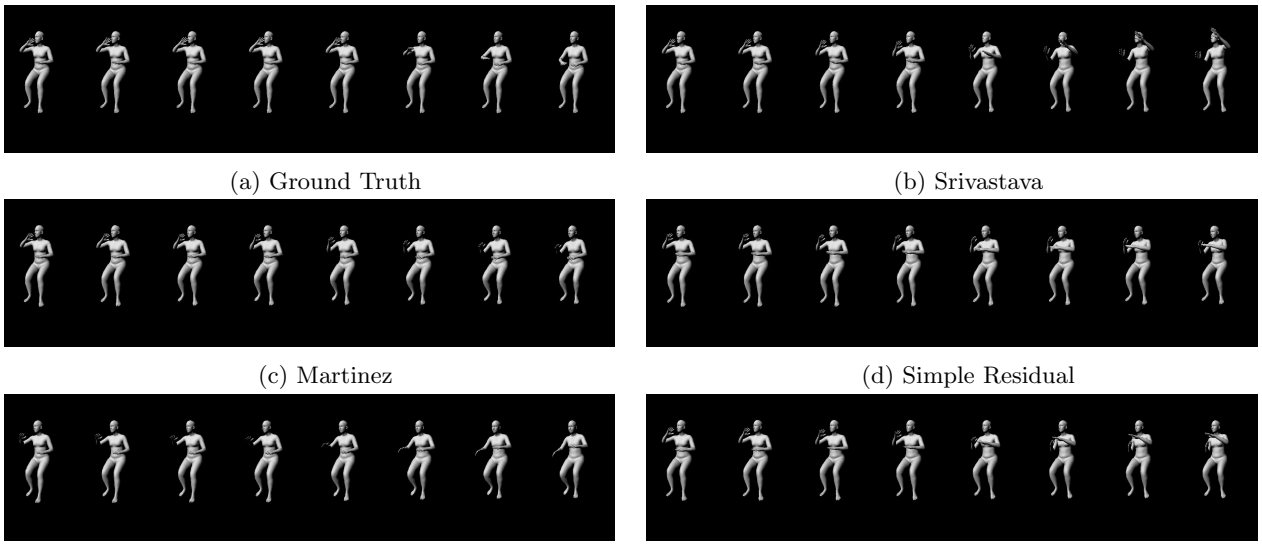
As we can observe, the Martinez network in [1] gets the less error than all the other networks in 11 out of the 16 cells in the table. More specifically, it is the best performing network in both eating and smoking movements. And it performs the best in most of the discussion move predicted frames error. The only movement where it does not perform the best is in walking, where ERD Residual and Srivastava have better performances than Martinez. In fact, it makes sense that ERD Residual performs the best in Walking, because the movement is smooth and fluid so if the input is feed to the output of the network the model will predict the movement better.

	Walking				Eating				Smoking				Discussion			
predicted frames	2	4	8	10	2	4	8	10	2	4	8	10	2	4	8	10
ERD Residual Repeated GRU	0.48	0.64	0.92	1.03	1.08	1.27	1.63	1.82	0.60	0.74	0.99	1.11	0.63	0.83	1.24	1.42
ERD No Residual Repeated GRU	1.02	1.11	1.32	1.43	1.58	1.62	1.72	1.76	0.89	0.96	1.12	1.20	0.84	0.92	1.08	1.15
Martinez Repeated GRU	0.37	0.64	1.04	1.17	0.54	0.84	1.3	1.5	0.27	0.45	0.73	0.85	0.43	0.69	1.07	1.2
Simple Residual Repeated LSTM	0.82	0.96	1.18	1.26	1.34	1.49	1.82	1.94	0.6	0.73	0.97	1.07	0.68	0.88	1.23	1.39
Srivastava Repeated LSTM	0.42	0.63	0.99	1.12	0.58	0.89	1.4	1.55	0.31	0.48	0.87	1.05	0.43	0.7	1.15	1.34

Table 6: Detailed results for motion prediction, measured in mean angle error for walking, eating, smoking and discussion activities of the Human 3.6M dataset with the SMPL model Format.

Now let's check how the representation of the predicted frames looks for each one of those 4 models. The 4 poses starting from the left correspond to the prediction of 2,4,8 and 10 respectively, or in other words, the short-term prediction. The next 4 poses correspond to the long-term prediction by predicting 20, 30, 40 and 50 frames respectively. For all the networks we represent exactly the same movement, they all start at frame n and finish at frame $n + 100$.

We will start with eating. As we can observe from the figure 41, ERD is the network which performs worst in the short term (e) as the right hand is more separated from the mouth than in the other networks. We also can observe how Martinez (c) and Srivastava (b) are the networks with the predicted right hand closer to the mouth as in eating. Please note that this particular sequence of frames from this movement in particular does not require of motion with the legs, so the pose estimation will be better. For the long-term prediction, in Srivastava after 50 frames is one of the worst to predict, as well as ERD Residual, for this particular example movement.



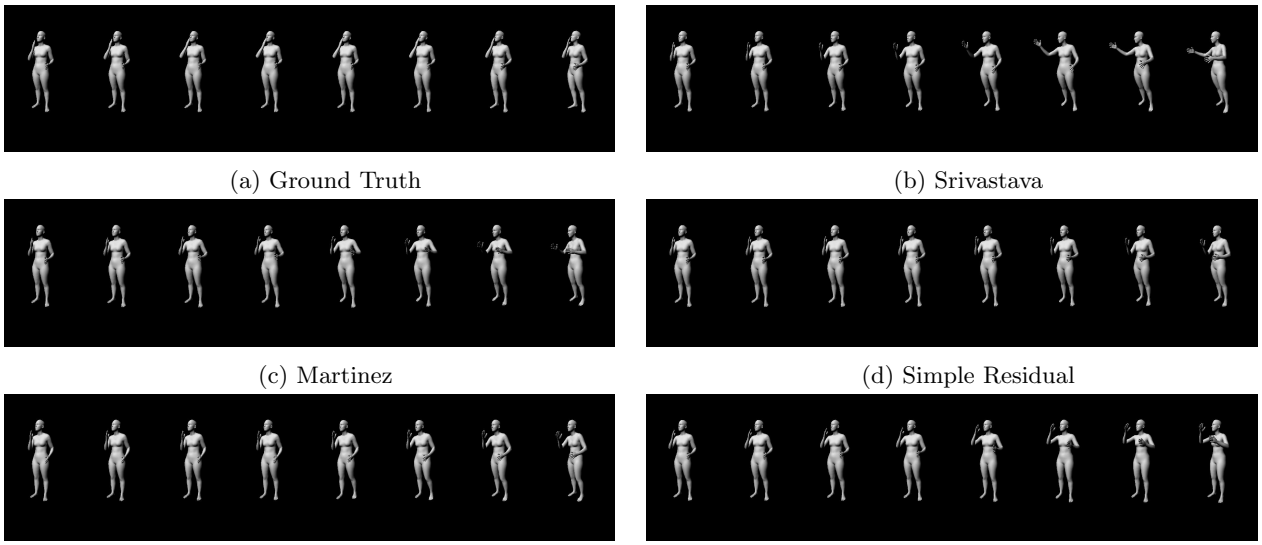
(a) Ground Truth (b) Srivastava

(c) Martinez (d) Simple Residual

(e) ERD (f) ERD Residual

Figure 41: Predicted frames: 2,4,8,10,20,30,40 and 50 of different networks for eating

By taking a look at the figure 42, for the short-prediction, we can observe that they all behave similar for this particular example. Is just in the case of Srivastava (b) and ERD Residual (f) where the right hand is more separated from the mouth of the subject and with slightly different rotation. For the long-term prediction we can see how Simple Residual (d) is the network with a predicted 50 frames closer to the ground truth.



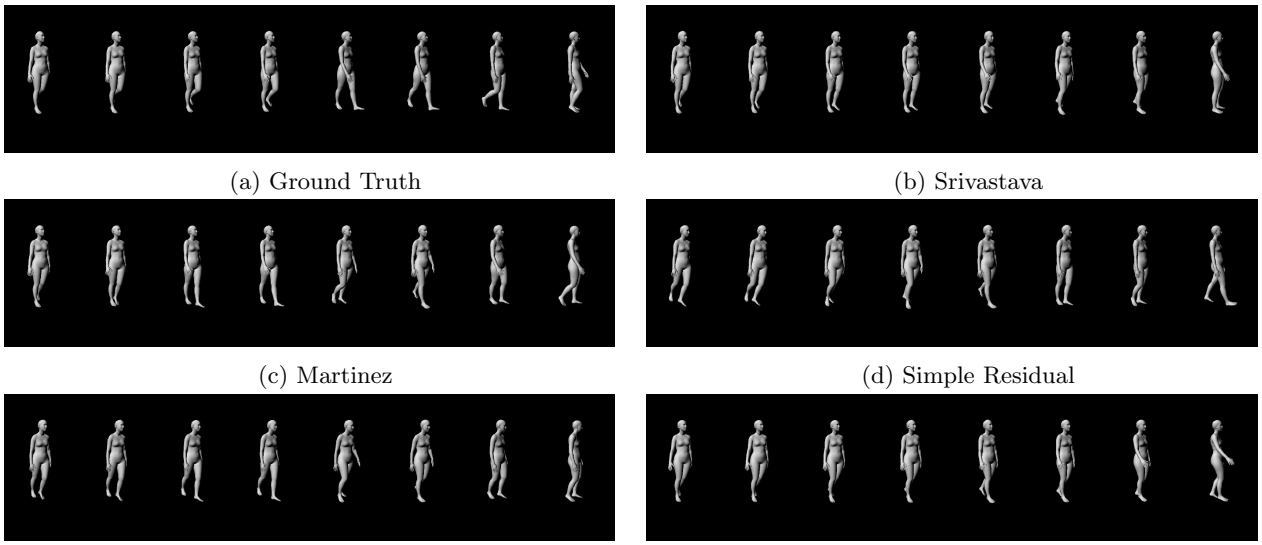
(a) Ground Truth (b) Srivastava

(c) Martinez (d) Simple Residual

(e) ERD (f) ERD Residual

Figure 42: Predicted frames: 2,4,8,10,20,30,40 and 50 of different networks for smoking

In the case of the movement walking, we can see the representation of 2 seconds of prediction in figure 43. As this movement is smooth and in only one direction (either in straight line or in circles) almost all the networks in this case have a good prediction of the predicted frames. For the short-term prediction ERD (e) performs the worst, as in the predicted frames 2,4,8 and 10 the feet and legs of the woman are roughly in the same place and the subject has moved so this walking representation of ERD is not realistic.



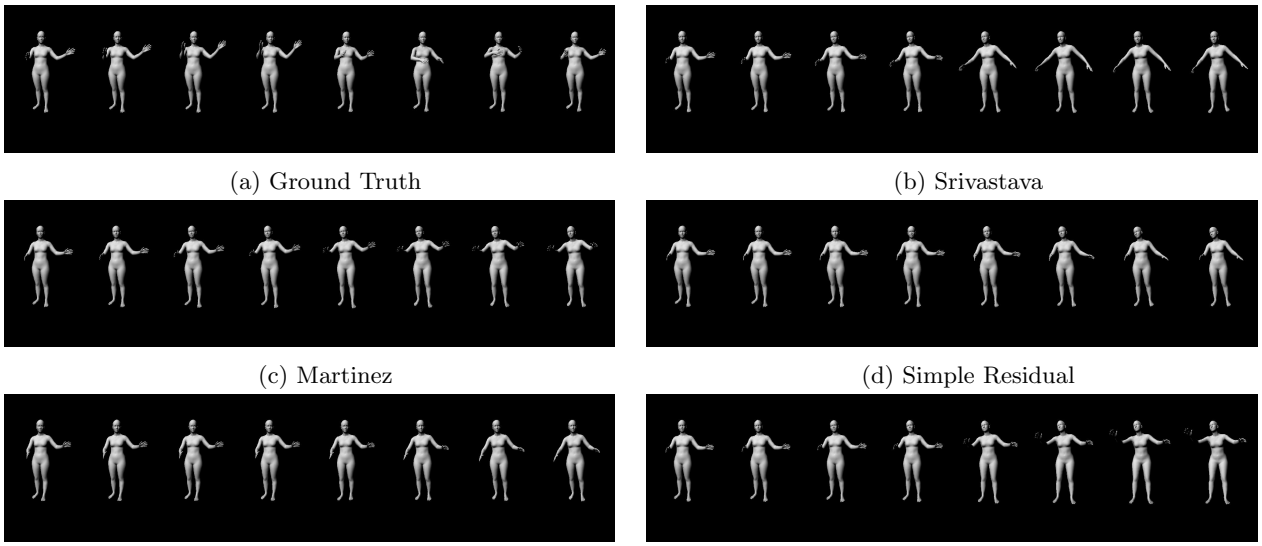
(a) Ground Truth (b) Srivastava

(c) Martinez (d) Simple Residual

(e) ERD (f) ERD Residual

Figure 43: Predicted frames: 2,4,8,10,20,30,40 and 50 of different networks for walking

Finally let's take a look at the figure 44, which represents the discussion movement. In this case, for the short-term prediction, Martinez (c), Srivastava (b) and ERD Residual (f) perform similarly, as in the third and fourth pose they are in a very similar pose. For the long-term prediction, Martinez (c) and ERD Residual (f) are the closest to the ground truth as in both the left arm of the subject is raised.



(a) Ground Truth (b) Srivastava

(c) Martinez (d) Simple Residual

(e) ERD (f) ERD Residual

Figure 44: Predicted frames: 2,4,8,10,20,30,40 and 50 of different networks for discussion

After seeing the predicted frames, we can take a look at the other 11 movements, plus the average of the movements results. The results are at tables 7 and 8. In these tables, we can observe how in average, the Martinez network performs better than the other networks. Let's put the average in a rank from 1 to 5 1 being the best performance and 5 being the worst in table 9. In this table, we have that the Martinez network comes at first, followed by the Srivastava mode, the ERD Residual, the Simple Residual and the ERD No Residual. It is interesting to see how adding a residual connection to ERD network drastically improves the performance of the network for the experiments.

	Directions				Greeting				Phoning				Posing				Purchasing				Sitting			
predicted frames	2	4	8	10	2	4	8	10	2	4	8	10	2	4	8	10	2	4	8	10	2	4	8	10
ERD Residual Repeated GRU	0.59	0.71	0.95	1.06	1.12	1.42	1.91	2.27	0.57	1.06	1.46	1.58	0.49	0.67	1.08	1.25	0.96	1.24	1.7	1.92	0.42	0.55	0.81	0.93
ERD No Residual Repeated GRU	1.15	1.18	1.36	1.44	1.54	1.67	1.86	1.99	1.5	1.61	1.87	1.89	1.33	1.41	1.55	1.68	1.55	1.73	1.91	1.96	0.99	1.04	1.18	1.24
Martinez Repeated GRU	0.25	0.53	0.83	0.95	0.66	0.95	1.47	1.66	0.41	0.66	1.12	1.3	0.34	0.68	1.27	1.46	0.62	0.99	1.4	1.55	0.27	0.45	0.75	0.89
Simple Residual Repeated LSTM	0.56	0.7	1.0	1.12	0.98	1.18	1.46	1.59	1.34	1.6	1.77	1.87	0.85	1.05	1.53	1.75	0.99	1.3	1.6	1.75	0.47	0.63	1.03	1.2
Srivastava Repeated LSTM	0.42	0.6	0.98	1.17	0.52	0.97	1.37	1.55	0.49	0.66	1.04	1.25	0.69	0.89	1.34	1.61	0.54	0.84	1.34	1.56	0.38	0.58	0.99	1.16

Table 7: Detailed results for motion prediction, measured in mean angle error for Directions, Greeting, Phoning, Posing, Purchasing and Sitting movements of the Human 3.6M dataset with the SMPL model Format

	Sitting Activities				Taking Photo				Waiting				Walking Dog				Walking Together				Average			
predicted frames	2	4	8	10	2	4	8	10	2	4	8	10	2	4	8	10	2	4	8	10	2	4	8	10
ERD Residual Repeated GRU	0.5	0.66	0.98	1.11	0.37	0.46	0.67	0.78	0.92	1.04	1.33	1.44	0.6	0.77	1.09	1.27	0.44	0.59	0.87	0.99	0.65	0.84	1.18	1.33
ERD No Residual Repeated GRU	1.28	1.32	1.46	1.53	0.89	0.93	0.99	1.04	1.66	1.82	2.18	2.17	0.82	0.96	1.09	1.28	0.76	0.82	0.98	1.05	1.19	1.27	1.44	1.52
Martinez Hidden GRU	0.36	0.64	1.08	1.26	0.28	0.51	0.95	1.09	0.44	0.76	1.17	1.47	0.41	0.57	1.00	1.19	0.32	0.52	0.83	0.95	0.4	0.66	1.07	1.23
Simple Residual Repeated LSTM	0.56	0.81	1.05	1.18	0.9	1.08	1.38	1.49	0.91	1.07	1.45	1.67	0.66	0.87	1.27	1.44	0.58	0.73	1.0	1.11	0.82	1.0	1.32	1.46
Srivastava Repeated LSTM	0.43	0.66	1.03	1.25	0.44	0.67	1.02	1.14	1.18	1.59	2.12	2.46	0.46	0.58	0.97	1.15	0.31	0.5	0.89	1.05	0.51	0.75	1.17	1.36

Table 8: Detailed results for motion prediction, measured in mean angle error for Sitting Activities, Taking Photo, Waiting, Walking Dog, Walking Together movements & average of the Human 3.6M dataset with the SMPL model Format

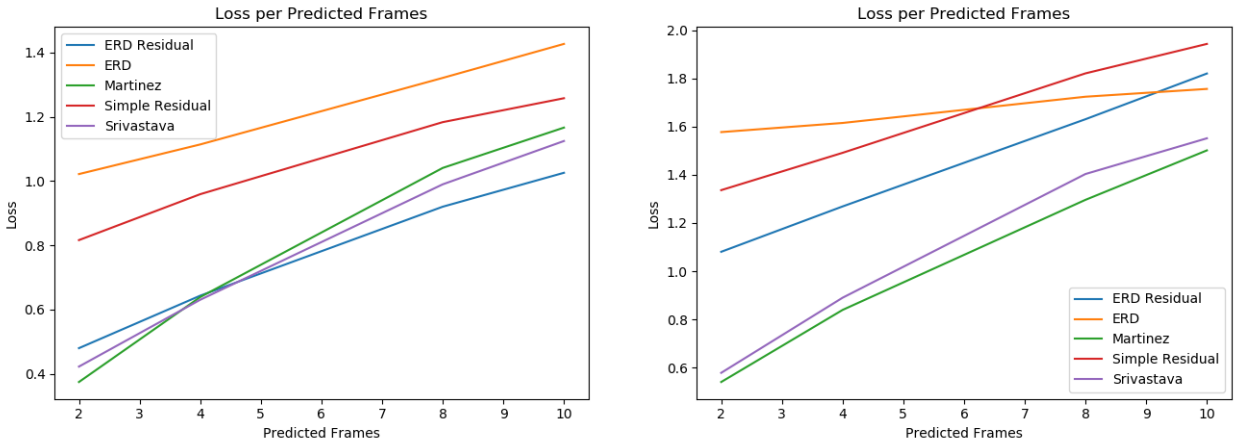
We also take into account, for the other 11 movements from the human 3.6m dataset which are not either walking, smoking, eating or discussion, which network is the best for a set number of predicted frames. Only three of the five networks are the best one at some point: Martinez, Srivastava and ERD Residual.

It is interesting to see how Martinez is better when the number of predicted frames is 2, where is the best at 8 of the movements. When the number of predicted frames is 4, Martinez is the best at 7 of the movements. However, when the predicted number of frames is 8 or 10, Martinez is not the best network as Srivastava wins when the number of predicted frames is 8, and Martinez is tied with ERD Residual when the number of predicted frames is 10. This means that Martinez gives better results when the number of predicted frames is 4 or less. But when is 8 or 10 Srivastava and ERD Residual have similar performances as Martinez.

	Average Rank				Nº Best Network			
predicted frames	2	4	8	10	2	4	8	10
ERD Residual Repeated GRU	3	3	3	2	0	2	2	4
ERD No Residual Repeated GRU	5	5	5	5	0	0	0	0
Martinez Repeated GRU	1	1	1	1	8	7	4	4
Simple Residual Repeated LSTM	4	4	4	4	0	0	0	0
Srivastava Repeated LSTM	2	2	2	3	3	2	5	3

Table 9: Rank of the Average Performance of the Networks per short-term prediction & Number of Times best networks among rest of the movements

For more detailing of the scores obtained by the previous tables, at figure 45 we can observe how the networks perform within the short-term, between 2 and 10 frames, for the Walking and Eating movements. ERD Residual network performs better than the other networks in walking for predicted frames 8 and 10, followed closely by Srivastava and Martinez networks. ERD network does not perform too well for the short-term prediction both in walking and eating, The ERD Residual performs better than the Simple residual in walking, performing similarly for the Walking movement to the Srivastava and Martinez networks.

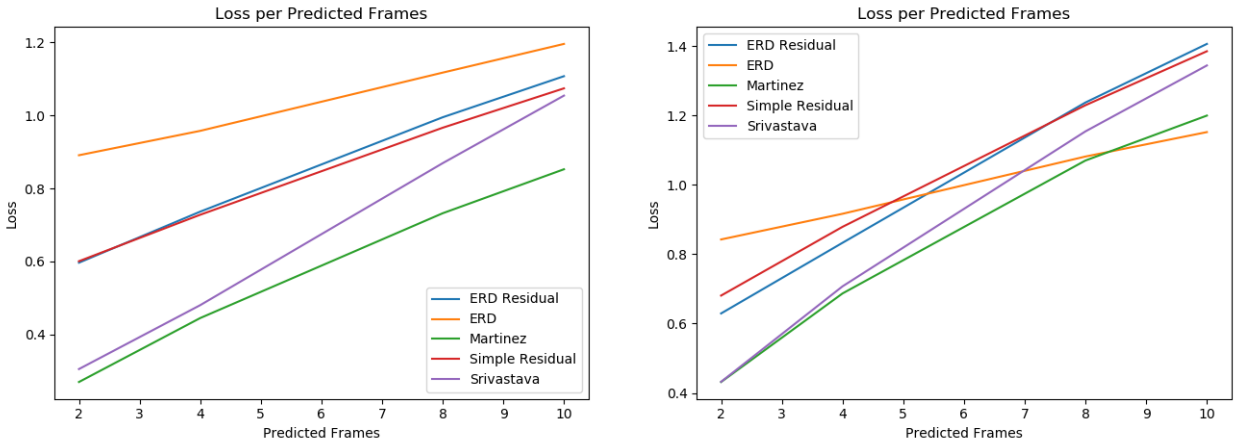


(a) Testing Loss for Walking

(b) Testing Loss for Eating

Figure 45: Short-Term Testing Loss per number of frames of different networks: Walking & Eating

At figure 46 we can observe how the networks perform within the short-term, between 2 and 10 frames, for the Smoking and Discussion movements. Again Martinez network performs better than the other networks for smoking and discussion movements, followed closely by Srivastava network. Also ERD network does not perform too well for the short-term prediction for smoking, being the network with the worst results of all. The ERD Residual performs slightly better than the Simple residual in the discussion movement, while for the smoking movement they perform really similar.



(a) Testing Loss for Smoking

(b) Testing Loss for Discussion

Figure 46: Short-Term Testing Loss per number of frames of different networks: Smoking & Discussion

Hence, from what we have seen in this section. We can say that the network in Martinez [1] is the network which has given us the best results in average for the short-term prediction. This dominance is more explicit whenever the number of predicted frames is less than 4. When the number of predicted frames is 8 or 10, the Srivastava [9] and our ERD Residual network also give similar results to the Martinez, although Martinez still has an slight advantage.

5.7 3D pose estimation and prediction with videos in the wild

We have also used the method in [8] to extract the 3D pose parameters of given videos of human performing some movements. To do this we have used some of the videos from the endlessReference dataset [23]. Featuring different actors and actresses doing some movements like walking, talking on the phone, walking drunk . . . From these videos, we have selected 100 frames of each test video randomly, extract the pose and beta parameters from the frames, and predict the movement by applying our models. An example of this pose extraction can be see at figure 47.

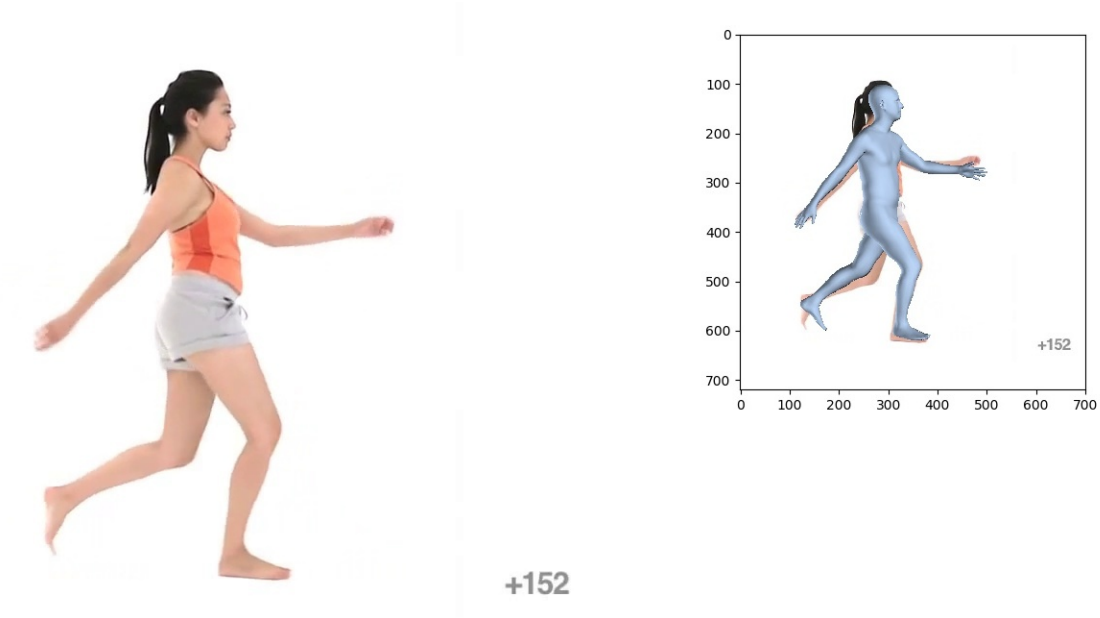


Figure 47: Left: Original Frame of [23] video. Right: Estimated SMPL Pose using [8]

Examples from these videos will be shown during the presentation of this master thesis to see an End-to-end human shape motion predictor with 3D pose estimation in the Wild. Which that means, is that there is no information about the shape or the pose, only the video is given. Also there are some examples of human-motion prediction from estimated pose in the Wild in the following github: `https://github.com/finuxs/Shape-Motion-Predictor` [21], in the folder *wildEstimation*

6 Conclusions & Future Work

6.1 Conclusions

During the development of this master thesis we have seen that human motion prediction is a highly complex problem because of the high dimensionality of the prediction and the stochastic nature of the human movement. The problem is interesting for us, but is a complex problem to pose.

For the human-motion prediction, we have seen the previous work and the theoretical concepts of different networks whose common point is that they use Recurrent Neural Networks for the human motion prediction. The implementation of these networks is based on the papers [1], [4] and [9]. We have also proposed two networks of our own which are variations of the networks cited on those papers.

For the pose and shape representation of human-motion we have introduced the SMPL model [7] a realistic representation in 3D format. The pose parameter of the SMPL model represents the relative position of the joints of the kinematic tree obtained from the human3.6m [6] dataset.

We have also introduced our Pytorch practical framework developed in order to perform the experiments. As the dataset we use is not the original human3.6m dataset, we explained the preprocessing of the data that was followed to obtain the SMPL pose and shape parameters for the human-motion representation. This preprocessing was done by using the method in the paper [8] to extract the SMPL parameters.

Within this practical framework, we also have explained the random cropping and subsampling method to add statistical robustness to our data by making the input of the networks not being the same each time the experiments were performed. Moreover, the two different input types for the predicted frames: Input Hidden Frames and Input Repeated frame were introduced.

Also the metrics that we used for the experiments were explained, distinguishing between the format of the angles between the joints, either in Exponential Map format or Euler angles. The transformation between these two types of angles was also reviewed by the method in [15]. The implementation of the networks from the previous work and our proposed networks was explained, such as the addition of residual connections or skip frames connections in some of the networks.

Regarding the experiments part, we first have tried each of the networks separately, testing with different input type (hidden or repeated frame) and RNN architecture (Gated Recurrent Unit or Long-Short Term Memory). We have seen that for all the networks, it gives better results repeating the last known frame to the predicted frames instead of setting the predicted frames to 0. This is because feeding to the networks the last known frame helps from the transition from the ground truth frame to the predicted frames and make the movement smoother. Otherwise this transition will be abrupt, as in the frame 51^{th} it has to learn from zeros. For the type of RNN architecture, we have seen that there is not a big difference in loss between using a GRU or a LSTM.

For short-term prediction, the network in Martinez [1] is the network which has given us the best results in average for the short-term prediction. The Martinez network excels whenever the number of predicted frames is between 2 and 4. When this number is 8 or 10, the performance of the Martinez network is similar to the Srivastava network [9] and our ERD Residual network.

Regarding the long-term prediction, we have seen that the networks tend to predict to the mean pose. As the human movement has an stochastic nature, its prediction has a high complexity. Moreover, the dimension of the prediction is high, of 69 parameters in our case, with values ranging from 0 to 2π. That is one of the reasons why measuring how well does a network predicts motion has certain complexity, as there are many joint errors to try to minimize in each epoch of the models.

We have seen the fact that not having a residual connection such as in the network ERD [4], the results change greatly depending of the type of the movement. For smooth continuous movements such as walking, the results are worst when compared to the other networks with residual connections. For more abrupt movements such as discussing, where aggressive hand gestures are made, feeding the input to the output of the network is not that good as the pose of the frames can suddenly change greatly from one frame to another. And is better to not add a residual connection in that case.

For the paper [1] that we used as a baseline we got the expected best results of all the tried networks. However, the results that are written on their paper are not statistically robust, because they only do testing with 8 samples. In our opinion, 8 is not nearly a high enough number to get consistent results. The results of the testing in this master thesis has been done with 1000 samples each, a much bigger number than 8.

Finally, we have used the method in [8] to extract the shape and pose SMPL parameters of a given video

featuring a person doing a certain movement. Once we have the parameters of SMPL, we predict the motion from this video with out trained networks. And that is the special point about this master thesis. Our networks were trained with a dataset of humans performing some movements. But the pose and shape of the subjects were not present in those videos, we had extracted it first. This can be defined as a more **generic** approach to the human-motion prediction. In all the papers of the previous work that we have mentioned, all of them were trained with given pose parameter, it was inherent to the dataset. We could define the work in this master thesis as an end-to-end RGB video to pose prediction with shape.

In what our approach contributes is that ability to generalize, as long as we can extract the SMPL parameters of the given video, human-motion can be predicted. It also allow us to represent human-motion with a realistic human shape with SMPL.

6.2 Future Work

The problem of the human-motion prediction that has been covered in this master thesis is interesting but it also would be interesting to see the human-pose classification problem. This problem would be, in our opinion easier to solve. This is because the dimensionality of the problem of the classification would not be as high as with the human-motion prediction. Also the metrics of the classification problem would be simpler. For this type of problem the networks would be different from the ones in this master thesis because we use RNNs for the prediction problem. Convolutional Neural Networks could be a good choice for the classification problem.

Another thing that would be interesting regarding human-motion prediction is to use other type of networks such as Generative Adversarial Network (GANs). Also more configurations RNN architectures can be used, changing the method of choosing the input (random cropping, subsampling...), the input type and the hyperparameters.

Regarding the 3D pose and shape estimation from in the Wild videos such as in [8]. We could try with other methods in order to make it more robust of computationally faster. The method could be obtained from [24], a neural network for direct inference of volumetric body shape from a single image.

References

[1] Julieta Martinez, Michael J. Black, and Javier Romero. "On human motion prediction using recurrent neural networks". In: *CoRR* abs/1705.02445 (2017). arXiv: 1705.02445. URL: http://arxiv.org/abs/1705.02445.

[2] *The AI boom*. https://www.theverge.com/2018/12/12/18136929/artificial-intelligence-ai-index-report-2018-machine-learning-global-progress-research. Online; accessed 16-April-2019.

[3] *Self-Driving Car Fatalities*. https://en.wikipedia.org/wiki/List_of_self-driving_car_fatalities. Online; accessed 16-April-2019.

[4] Katerina Fragkiadaki, Sergey Levine, and Jitendra Malik. "Recurrent Network Models for Human Dynamics". In: *CoRR* abs/1508.00271 (2015). arXiv: 1508.00271. URL: http://arxiv.org/abs/1508.00271.

[5] Ashesh Jain et al. "Structural-RNN: Deep Learning on Spatio-Temporal Graphs". In: *CoRR* abs/1511.05298 (2015). arXiv: 1511.05298. URL: http://arxiv.org/abs/1511.05298.

[6] Catalin Ionescu et al. "Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 36.7 (July 2014), pp. 1325–1339. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2013.248. URL: https://doi.org/10.1109/TPAMI.2013.248.

[7] Matthew Loper et al. "SMPL: A Skinned Multi-Person Linear Model". In: *ACM Trans. Graphics (Proc. SIGGRAPH Asia)* 34.6 (Oct. 2015), 248:1–248:16.

[8] Angjoo Kanazawa et al. "End-to-end Recovery of Human Shape and Pose". In: *CoRR* abs/1712.06584 (2017). arXiv: 1712.06584. URL: http://arxiv.org/abs/1712.06584.

[9] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. "Unsupervised Learning of Video Representations using LSTMs". In: *CoRR* abs/1502.04681 (2015). arXiv: 1502.04681. URL: http://arxiv.org/abs/1502.04681.

[10] Mykhaylo Andriluka et al. "2D Human Pose Estimation: New Benchmark and State of the Art Analysis". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2014.

[11] Kathleen Robinette, Hein Daanen, and Eric Paquet. "The CAESAR project: a 3-D surface anthropometry survey". In: vol. 380. Feb. 1999, pp. 380–386. ISBN: 0-7695-0062-5. DOI: 10.1109/IM.1999.805368.

[12] D. Hirshberg et al. "Coregistration: Simultaneous alignment and modeling of articulated 3D shape". In: *European Conf. on Computer Vision (ECCV)*. LNCS 7577, Part IV. Springer-Verlag, Oct. 2012, pp. 242–255.

[13] F Sebastian Grassia. "Practical Parametrization of Rotations Using the Exponential Map". In: *Journal of Graphics Tools* 3 (Jan. 1998). DOI: 10.1080/10867651.1998.10487493.

[14] Graham W. Taylor, Geoffrey E Hinton, and Sam T. Roweis. "Modeling Human Motion Using Binary Latent Variables". In: *Advances in Neural Information Processing Systems 19*. Ed. by B. Schölkopf, J. C. Platt, and T. Hoffman. MIT Press, 2007, pp. 1345–1352. URL: http://papers.nips.cc/paper/3078-modeling-human-motion-using-binary-latent-variables.pdf.

[15] *Exponential map to Rotation matrix*. https://github.com/asheshjain399/RNNexp/blob/srnn/structural_rnn/CRFProblems/H3.6m/mhmublv/Motion/expmap2rotmat.m. Online; accessed 16-April-2019.

[16] Gregory G. Slabaugh. *Computing Euler angles from a rotation matrix*.

[17] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Apr. 2011, pp. 315–323. URL: http://proceedings.mlr.press/v15/glorot11a.html.

[18] *Deep Learning Framework Power Scores 2018*. https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a. Online; accessed 16-April-2019.

[19] *Institut de Robòtica i Informàtica industrial-Universitat Politècnica de Catalunya*. https://www.iri.upc.edu/. Online; accessed 16-April-2019.

[20] *Pytorch Deep Learning Framework of Institut de Robòtica i Informàtica industrial: IRI*. https://github.com/albertpumarola/IRI-DL. Online; accessed 16-April-2019.

[21] *A Human Pose & Shape Motion Predictor*. https://github.com/finuxs/Shape-Motion-Predictor. Online; accessed 16-April-2019.

[22] *High Performance Data Store hdf5*. https://www.hdfgroup.org/solutions/hdf5/. Online; accessed 16-April-2019.

[23] *Endless Reference Dataset*. https://endlessreference.com/. Online; accessed 16-April-2019.

[24] Gül Varol et al. "BodyNet: Volumetric Inference of 3D Human Body Shapes". In: *CoRR* abs/1804.04875 (2018). arXiv: 1804.04875. URL: http://arxiv.org/abs/1804.04875.