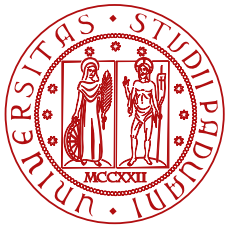


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Department of Information Engineering

Master Thesis in Computer Engineering

Enhancing Robot Collaborative Skills by Predicting Human Motion on Small Datasets through Deep Transfer Learning

Master Candidate

Marco Casarin

Student ID 2044727

Supervisor

Prof. Stefano Michieletto

University of Padova

Co-supervisor

Michael Vanuzzo

University of Padova

Academic Year
2022/2023
7th September 2023

Abstract

Collaborative Robotics is an emerging filed that combines the unique capabilities of humans and robots to create a flexible working environment that meets the evolving needs of industrial production. A key aspect of improving the collaborative capabilities of robots is their ability to accurately predict human movements, thereby facilitating smoother and more productive collaborations. Although the problem of Human Motion Prediction (HMP) has received considerable attention in recent years, it has not yet been studied in the specific context of Collaborative Robotics. There are some limitations in this regard, in particular the lack of comprehensive and specific datasets for training Deep Learning (DL) models, which are currently the state of the art solution for this problem. This thesis investigates Transfer Learning (TL) approaches for DL models to improve human motion prediction on smaller, domain-specific datasets, with the aim of enhancing the collaborative capabilities of robots. Several experiments were conducted to compare the performance of DL models initialised from scratch and those derived using TL approaches. The experiments were based on state of the art DL models and various domain-specific datasets of different sizes. The evaluation of individual DL models was based on the main metrics proposed in the literature for the HMP task. In addition, complementary metrics were used to assess the advantages of TL techniques. The results show that TL approaches have significant potential to improve accuracy for the HMP task in the case of domain-specific contexts. These benefits are particularly pronounced when dealing with datasets of limited size, especially when their human motion actions are similar to those found in the dataset used to pre-train these DL models. Furthermore, the importance of using a large and diverse dataset for model pre-training is highlighted. These observations pave the way for future research involving the replication of these experiments using a significantly larger and richer dataset for pre-training DL models. This will allow for the testing of much more complex DL models and TL techniques, thus providing a broader perspective on this type of approach.

Contents

List of Figures

List of Tables

List of Acronyms

CV Computer Vision

HMP Human Motion Prediction

HRC Human Robot Collaboration

ML Machine Learning

DL Deep Learning

TL Transfer Learning

DTL Deep Transfer Learning

DNN Deep Neural Networks

RNN Recurrent Neural Network

NN Neural Network

LSTM Long Short-Term Memory

GRU Gated Recurrent Unit

ERD Encoder-Recurrent-Decoder

DAE Dropout Autoencoder

PVRED Position-Velocity Recurrent Encoder-Decoder

GCN Graph Convolutional Network

CNN Convolutional Neural Network

DMGNN Dynamic Multi scale Graph Neural Network

GAN Generative Adversarial Network

NLP Natural Language Processing

LLM Large Language Model

H36M Human 3.6M

AMASS Achieve of Motion capture As Surface Shapes

3DPW 3D Poses in the Wild

fps frames per second

SMPL Skinned Multi-Person Linear Model

HRI History Repeat Itself

RED RNN Encoder-Decorder

MSE Mean Squared Error

QT Quaternion Transformation

DCT Discrete Cosine Transform

IDCT Inverse Discrete Cosine Transform

MPJPE Mean Per Joint Position Error

HAR Human Activity Recognition

PNN Progressive Neural Network

seq2seq sequence to sequence

IoU Intersection over Union

RL Reinforcement Learning

PCA Principal Component Analysis

JSP Jump-Start Performance

MEP Minimum Error Performance

TTT Time To Threshold

1

Introduction

Over the last few decades, the field of industrial robotics has undergone a remarkable transformation, significantly changing the manufacturing process and production automation. Industrial robots have been at the forefront of this revolution. They have undergone significant advances, leading to remarkable improvements in the speed and safety of production chains. The evolution of industrial robots, since their first applications in industry, has been significant and dynamic [9]. Initially, these machines lacked any form of intelligence and they were based on predefined movements, limiting their use to repetitive and very simple tasks. To adapt the same machine to a new task, the entire robot had to be reprogrammed, which made their use time demanding. Over time, advances in research fields such as Computer Vision (CV) and Machine Learning (ML), have paved the way for the introduction of intelligence into robots. By integrating various sensors, such as cameras and lidars, robots became able to perceive and understand their environment. By processing these data, the systems were able to adapt their behaviour to different situations, bringing to a new level of flexibility. This evolution has continued over time, leading to the current situation where machines have achieved a good level of self-adaptation. For example, by using a vision system, a robot can perform a pick-and-place task effectively even when dealing with different objects of different orientations. However, the evolving needs and goals of modern industry, which seeks to achieve customizable production and seamless integration between machines and humans, have given rise to a new frontier in robotics known as *Collaborative Robotics*. This emerging field aims to enable robots and humans to

work together productively and safely in close proximity, thus introducing a new era for industry and robotics. This chapter introduces the field of Collaborative Robotics and the problem of Human Motion Prediction (HMP) and provides an overview of the current state of the art.

1.1 Collaborative Robotics

Collaborative Robotics is an emerging filed of robotics, driven by the evolving needs of industries focused on a customizable production and a greater emphasis on creating a synergy between humans and machines [18]. Unlike traditional industrial robots that operate in closed and isolated environments, this new branch of robotics aims to develop and implement collaborative robots (commonly known as *cobots*) that can work alongside human operators in shared workspaces, bridging the gap between automation and human intelligence. Safety is a key aspect of Collaborative Robotics, as cobots are designed to coexist with humans. Equipped with force/torque sensors, vision systems and proximity detectors, cobots can sense human presence and understand their surroundings, ensuring a safe working environment. They can also contribute to human well-being by performing monotonous, repetitive, and dangerous tasks, thereby reducing the risk of workplace injuries. Their versatility extends beyond the industrial environment, with applications in sectors as diverse as healthcare, where they are used, for example, to assist doctors during surgical procedures [30]. Collaborative Robotics combines the strengths of both humans and robots. It allows humans to use their creativity, problem-solving skills and adaptability, while benefiting from the precision, speed and strength of robots. By assigning tasks based on each agent's unique capabilities, humans and cobots can work together to effectively achieve their common goals. They are also typically equipped with user-friendly programming interfaces, such as hand-guided teaching, which allow non-experts to easily train and command cobots. In addition, their adaptability to dynamic environments and seamless integration into existing workflows ensure minimal downtime and smooth operations. As a result, Collaborative Robotics holds great promise for improving industry and, in particular, the well-being and safety of the human operators.

Human Robot Collaboration (HRC) is an active and promising area of research that offers significant benefits in various applications, particularly in industrial environments. Despite its potential, the use of cobots still suffers

from certain limitations and research gaps [31]:

- *Human safety*: Safety is a critical issue due to the close proximity of humans and machines. While safety is essential, it can sometimes conflict with maximising performance, for example by reducing the cobot's speed to avoid potentially dangerous collisions. Finding the right balance between safety and performance is essential for efficient and safe collaboration.
- *Intuitiveness*: Human operators can find it challenging to work with cobots, mainly due to a lack of confidence in the robot's capabilities, which can hinder productive collaboration. Improving the intuitiveness and usability of such systems is essential to promote smooth collaboration.
- *Adaptability*: The ability of cobots to adapt to new environments and tasks is a another major research gap. Improvements in Collaborative Robotics technology should enable a single robot to perform different (not necessarily closely related) tasks in different environments without the need for major software or mechanical updates.

These research gaps and technological challenges can be mitigated by improving the overall intelligence of robotic systems, which can be identified in three main aspects: 1) the *robot perception*, i.e. everything related to the understanding of what is happening in the working environment, which includes the data acquisition and data processing. 2) the *prediction* of *human movements*, which allows the system to anticipate the operator's intentions, avoiding the need to constantly wait for the robot to perform the task in response to specific commands. This can improve the fluidity and safety of the collaboration, leading to increased productivity. 3) the *robot motion*, which includes aspects related to the handling of the machine according to the information acquired in the previous steps. It is essential to understand how to move the robot in a way that optimally increases the operator's confidence in collaborative interaction. This includes aspects such as improving the interpretability and comprehension of the robot's movements.

The work in this thesis focuses on improving the prediction of human motion in Collaborative Robotic environments. By understanding and anticipating human intentions, cobots can dynamically adapt and optimise their motion, improving coordination with the operator and the overall quality of the collaboration.

1.2 Human Motion Prediction

Anticipation is an innate and involuntary phenomenon that plays a crucial role in human interactions. In activities such as receiving an object from a person or lifting a heavy box with someone else, we naturally anticipate the other person's movements and adjust our actions accordingly, without explicit communication. This concept extends beyond physical interactions to situations such as driving, where we anticipate the movements of pedestrians or cyclists to avoid potential hazards. The ability to understand and anticipate human motion is a critical aspect for intelligent systems that coexist and collaborate with humans in shared workspaces [29]. The anticipation of human motion is a multidisciplinary task that focuses on predicting future human movements of individuals or groups. It involves interpreting human behaviour and translating it into predictive models capable of estimating future movements, sometimes even multiple plausible futures. The importance of predicting human motion spans multiple and diverse applications. In autonomous vehicles, predicting the intentions of pedestrian in real time is critical to avoid accidents. In sports and entertainment, motion prediction enhances virtual reality experiences, video game interactions and motion capture technologies. In addition, HMP has applications in healthcare, assisting in the analysis and rehabilitation of human movement disorders. Unfortunately, accurate predictions of body motion is a very difficult challenge, even for humans, due to the complexity and non-deterministic nature of human behaviour. Our movements are constantly influenced by internal and external stimuli, leading to sudden and unexpected changes in movement patterns [23]. For example, if any of us were to observe a person walking for a few seconds, we would most likely predict a walking motion for the next few seconds as well. A sudden event, such as a mobile phone call or a friend walking by, can significantly change the person's actual movement, making our prediction completely wrong. Nevertheless, the development of effective HMP algorithms is of great importance, given the wide range of applications that can benefit from this capability.

In Collaborative Robotics, the prediction of human motion is a critical aspect to ensure productivity and safety during human-machine collaboration tasks. However, predicting human motion in this context is extremely challenging due to several factors that are not present in traditional motion prediction scenarios. These additional aspects include the presence of objects in the shared workspace

(some static, some moving and some interacting) and the presence of the cobot itself. These elements significantly influence human behaviour and add complexity to the HMP problem. Deep Learning (DL) algorithms have recently shown great results in HMP, outperforming statistical methods and standard ML algorithms. However, most existing datasets, and consequently the DL models trained on them, are based on recordings of humans performing general activities in open spaces. To effectively predict human motion in Collaborative Robotics settings, it is essential to include more information about the environment, such as the robot's position, the location of objects in the workspace, and the semantic information about the human operator's actions.

Unfortunately, such comprehensive datasets are very limited, and creating new ones can be costly and time consuming. Therefore, this thesis aims to explore the technique of Transfer Learning (TL) to investigate the adaptability of existing DL trained models for HMP to Collaborative Robotics contexts. Transfer Learning is a general ML technique that allows the transfer of knowledge from pre-trained models on large and more generic datasets to downstream tasks, where data availability is limited. By applying Transfer Learning, this research aims to improve the predictive capabilities of existing models and open up new possibilities for effective Human Robot Collaboration.

1.3 HMP-knowledge transferability

To improve the prediction of human motion in the context of Collaborative Robotics, it is necessary to train DL models on very large and specific datasets, which are unfortunately not available at the time of writing. However, outside the context of Collaborative Robotics, many datasets are available for the more general problem of HMP (which will be discussed later). The work in this thesis aims to understand and investigate the transferability of the knowledge from DL models, trained on such datasets, to the specific context of Collaborative Robotics. In particular, the aim is to understand what can be achieved without the need to collect large datasets specific to the context of Collaborative Robotics. One of the most important aspects to understand is how well different DL models are able to actually learn about the human body and its motion dynamics, independently from the actions they are trained on. This property can be fundamental, as their knowledge of human body, gained from training on general but large datasets, can be transferred to more specific contexts. If

this is the case, by collecting a relatively small amount of training examples, the models can adapt their knowledge to well predict actions that are specific to Collaborative Robotics. Therefore, this work tries to answer the following questions: *How much can a DL model learn about the human skeleton and its motion dynamics, regardless the actions it sees during training? How much does the knowledge acquired by the model depend on the actions it sees? Is it possible to improve HMP in Collaborative Robotics settings without the need to collect large datasets?*

1.4 State of the Art review

This section provides an overview of the current state of the art approaches for the problem of HMP and the technique of Transfer Learning for Deep Neural Networks (DNN). It also provides an overview of the main datasets currently available for training DNN models for the task of HMP.

1.4.1 Human Motion Prediction techniques

The way in which human motion is represented, and therefore the type of data to be predicted, depends on the specific application; indeed, the problem of anticipating human motion has several facets. For example, in the work proposed by Lee et al. [19], the goal was to predict the overall position of football players within the field. In this specific application, each person can be modelled as a single point described by two coordinates in a 2D grid map representing the ground. Therefore, the prediction of a player's movement consists of a 2D occupancy grid over time. On the other hand, in the context of Collaborative Robotics, predicting human motion requires a more comprehensive approach, where the full 3D pose of the person must be determined. This involves not only capturing the absolute position in space, but also predicting the trajectories of individual joints that define the body pose. Indeed, a complete understanding of where the hands, legs, and all other parts of the human body are located in space is essential for successful collaboration. What these approaches have in common, however, is that they represent a movement of some duration through a sequence of poses, each of which contains the information required for the task of interest.

Human motion prediction algorithms can be broadly divided into two main groups [29, 23]: 1) models based on Deep Learning techniques and 2) models

based on statistical methods, pattern-based rules, and standard ML approaches. In recent years, end-to-end DL models have demonstrated superior performance compared to other methods for predicting human motion, especially for longer predictions. This is due to their remarkable ability to learn complex patterns from data. As a result, non-Deep Learning approaches have now become obsolete and the focus of the literature has shifted mainly to DL models. This section provides an overview of the most common DL approaches for 3D human motion prediction, which can be divided into three main categories according to the type of Neural Network (NN) architecture underlying the different proposed solutions [23]:

- *Recurrent Neural Networks (RNNs)*: RNNs are a type of NN specifically designed to handle sequential data, making them particularly well suited to time series tasks. The key concept behind their architecture is that information is represented by hidden states, which are then used recursively as input for subsequent steps. This recursive approach allows the hidden state of each step to encapsulate information from previous states, thereby encoding the entire context of the sequence up to that point. However, a notable limitation of standard RNNs is that they suffer from vanishing or exploding gradient during training, which limits their ability to effectively capture long-term dependencies in sequences. To mitigate this problem, several improved variants have been proposed, including Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) cells, which have shown significant improvements. For the HMP task, RNNs can be used to capture the temporal relationships between the poses along a motion sequence, and then use this information to generate a prediction sequence. Fragkiadaki et al. [8] proposed a Encoder-Recurrent-Decoder (ERD) approach based on a RNN architecture that incorporates non-linear encoder and decoder networks before and after recurrent layers. To reduce error accumulation over time, Ghosh et al. [10] proposed a 3-layer LSTM network followed by a Dropout Autoencoder (DAE) block that filters the predicted poses, reducing error accumulation. Wang et al. [37] proposed a Position-Velocity Recurrent Encoder-Decoder (PVRED) approach based on a RNN architecture that incorporates both the position and velocity information of human motion, improving long-term predictions.

- *Graph Convolutional Networks (GCNs)*: GCNs are a type of DL model de-

signed to process and analyse data represented in the form of a graph, where nodes (vertices) represent individual elements and the links (edges) between nodes represent relationships and interactions between these elements. The architecture of GCNs is based on convolutional operations, that aggregate information from adjacent nodes. In this way, they learn to effectively represent the relationships between different elements within the graph, and then use this knowledge to generate the output of interest. The underlying structure of GCNs is similar to that of classical Convolutional Neural Networks (CNNs), but instead of extracting features from images, they process information from graphs. This characteristic has led to their extensive use in scenarios where the data has an internal structure that can be effectively modelled as a graph. In particular, for the task of HMP, they can capture the relationships between the different joints of the skeletal representation of the human body. In [26], instead of representing the human pose as a kinematic tree, they used a graph where each pair of body joints is connected. Li et al. [20] proposed a Dynamic Multi scale Graph Neural Network (DMGNN) to model the internal relationships of the human body for feature learning at different scales. These features are then combined to generate the motion predictions.

- *Generative Adversarial Networks (GANs)*: GANs are a powerful class of DL models designed to generate realistic and high quality synthetic data by sampling from a learned data distribution. The GAN architecture consists of two basic components: the generator and the discriminator. The goal of the generator is to generate realistic data, while the discriminator is trained to distinguish between real and synthetic data (from the generator). During the training phase, the GAN strives to reach an equilibrium point where the generator becomes so good at producing synthetic data that it is indistinguishable from real data, making it difficult for the discriminator to distinguish between the two. The strength of GANs lies in their ability to learn complex data distributions and generate synthetic ones with a very high degree of fidelity. For the HMP task, this feature can be used to generate future predictions. However, it is important to recognise that GANs can present challenges during the training process and can generate unrealistic data. In [5], the authors proposed a GAN-based approach in which the input is represented as a probability distribution, which is

then used to generate multiple future sequences. In [16] they proposed a bidirectional GAN architecture to limit the mode-collapse problem (when the generator can only produce a small set of outputs). They also used the network to generate multiple futures.

The first two classes of architectures described above, RNNs and GCNs, represent deterministic models. The problem is formulated as a regression task, i.e. they always produce the same output for a given input. In contrast, GANs belongs to the class of probabilistic approaches that introduce an element of randomness into the network output during inference. As a result, GANs can produce different outputs for the same input. This feature can be used to produce multi-future human motion predictions.

The *attention mechanism* is another important aspect to consider in this review. Rather than being a distinct architecture, the attention mechanism is a versatile feature that can be integrated into various DL models. With the rapid advancement of technology, complex models with millions of parameters have been created with the ability to learn a significant amount of information. However, when dealing with sequential data, especially long sequences, it becomes difficult for these models to retain all the information from the beginning up to the current time, leading to information loss. The attention mechanism stands out as one of the most powerful techniques discovered in recent years. Given a sequence as input, this mechanism allows the model to focus on specific parts of the input data that are most relevant for predicting the current step. In practice, this is achieved by computing, at each time step, a representation of the previous input sequence by a weighted combination of hidden states from previous time steps. The weights, i.e. the importance to be given to each past segment, are learned by the model during the training phase. The attention mechanism has shown remarkable effectiveness in various fields, such as in Natural Language Processing (NLP) for machine translation, using the remarkable Transformers architecture [35]. In the context of HMP, attention can be seamlessly incorporated into NN architectures, allowing the models to focus on particular patterns of human body motion or sub-sequences within the input sequence. Mao et al. [25] proposed a DL model based on GCNs that implements the attention mechanism to capture similarities between the current and past motion sub-sequences. Aksan et al. [3] proposed a DL approach based on Transformers architecture that implements a spatial and temporal self-attention mechanism. The first relates

all the skeletal joints of the human body at a given time step, while the second relates a single joint to itself in time during the motion sequence.

1.4.2 Transfer Learning

DL models require large datasets for effective training, which typically involves large investments in both data collection and model training. Particularly in the filed of Collaborative Robotics, this can be challenging due to data scarcity. TL is emerging as a valuable machine learning technique to overcome this problem. By using models pre-trained on large datasets for similar tasks, TL allows knowledge transfer to a task-specific model that can be adapted to perform well on a smaller dataset. This approach actually improves the generalisation performance of the model when limited data is available. TL has shown remarkable effectiveness in several contexts, including NLP and CV [33]. In NLP, Large Language Models (LLMs) have gained significant popularity, exemplified by chatbots such as *ChatGPT* and *Google BARD*. These LLMs are trained on large corpora of text from different sources, such as *Wikipedia* and *Google Books*, to build up a large knowledge about the language, and then used for downstream tasks using TL techniques. Mozafari et al. [28] proposed a TL approach based on the BERT LLM [7] to address the lack of labelled data for detecting hateful content in online social media. Similarly, Tida et al. [34] used BERT for real-time spam email detection. In CV, TL is widely used for object detection tasks, where models pre-trained on millions of images are fine-tuned for more specific applications. For example, the pre-trained object detector YOLOv5 [17] can be adapted trough fine-tuning techniques to serve as a high-performance object detector for specific classes of targets [21, 1].

To the best of our knowledge, no work has been published in the context of HMP that specifically investigates the TL technique for DL algorithms. However, there are some papers in the literature that have applied these techniques to related tasks. Gupta et al. [13] proposed TL approaches for the task of Human Activity Recognition (HAR) via video surveillance. The TL approach has been studied to deal with the need to re-train these models when the activity patterns of new users differ from those seen by the model during training. Gui et al. [12] proposed an approach based on *meta-learning* to train a predictor with improved generalisation performance. In particular, the goal was to train a model that could later be quickly adapted to task-specific unseen actions in a few training

sequences.

Given the promising results of TL in various domains, this thesis aims to explore its application in the context of collaborative robotics, investigating the transferability of knowledge for HMP models. In doing so, it seeks to address the challenge of limited data availability and open up new possibilities for improving Human Robot Collaboration.

1.4.3 Datasets

The type and the amount of data available influences the architecture of the NN, e.g. the size of certain layers (input and output) or the complexity required by the network: large amounts of data require a larger model to achieve good performance. Datasets play a key role in the design and training DL models, so it is important to make an in-deep analysis of the available datasets before starting to train the networks. In the following an overview of the main datasets available for the task of HMP: Human 3.6M (H36M) [15], Achieve of Motion capture As Surface Shapes (AMASS) [24], 3D Poses in the Wild (3DPW) [36], and CMU [11]:

- *Human 3.6M (H36M)*: This dataset contains 3D human poses recorded from 11 different subjects (5 females and 6 males), each performing 15 different typical activities such as walking, smoking, discussing, or eating . The data was captured by using a vicon motion capture system at 50 frames per second (fps), recording two different sub-actions for each activity, giving a total of 30 motion captures per subject. Each human movement is represented by a sequence of poses, where each pose is described by joint angle rotations of 32 keypoints. H36M is one of the first large datasets for HMP to be published, and it has been the standard for training and evaluating DL models for this task in recent years. However, given recent advances in computing technology, this dataset is now relatively small. Furthermore, the limited number of actions recorded introduces a strong bias in the data, which can lead to poor generalisation performance for models trained on this dataset.

- *Achieve of Motion capture As Surface Shapes (AMASS)*: This is one of the larger dataset available at the time of writing, containing recordings from 500 different subjects for a total of more than 60 hours of recordings.

AMASS aggregates motion captures from many different small datasets, standardising the representation using the Skinned Multi-Person Linear Model (SMPL) [22]. Each human pose is represented by 24 keypoints rotations at different fps, depending on the specific sub-dataset, ranging from 25 to 120.

- *3D Poses in the Wild*: 3DPW is a dataset of people performing actions outdoors, hence the name "in the wild". The motion sequences were recorded using a mobile phone camera, which allowed flexibility in capturing everyday life scenes, including playing sports, hugging, riding a bus, and taking selfies. It consists of a total of 60 videos and 7 actors, recorded at 30 fps.

- *CMU*: CMU is a dataset published by Carnegie Mellon University. It consists of a total of 144 different subjects, recorded from 41 markers placed on the human body. Similar to previous datasets, it contains recordings of common actions such as sports activities, walking, and directing traffic.

The problem with these datasets is that they relate to the movements of people in general contexts. In a Collaborative Robotics environment, however, it is important to include other information, such as the position of the cobot itself in space, or the position of objects, knowing which are interactive and which are static. Since the acquisition of datasets is quite expensive, it is important to understand how much knowledge can be transferred from models trained on these generic datasets to smaller ones, that are more specific to collaborative environments.

2

Deep Learning approaches for Human Motion Prediction

This Chapter introduces the representation of human motion in a machine-readable form and a detailed description of the Deep Learning (DL) models that will be later the subject of the experiments for the Deep Transfer Learning (DTL) technique. These models where chosen to be representative of the main state of the art architectures for the problem of Human Motion Prediction (HMP) reviewed in Section 1.4. In particular, two models are considered: 1) Position-Velocity Recurrent Encoder-Decoder (PVRED) [37], an architecture based on a Recurrent Neural Network (RNN) and 2) History Repeat Itself (HRI) [25], based on Graph Convolutional Network (GCN) and the attention mechanism.

2.1 Human Motion Representation

This Section discusses the dominant approach to represent human motion in DL models, which require input data in a machine-readable form, i.e. numbers. The standard methodology is to represent the human body through a skeletal structure, so that each pose can be described by the angles of rotation of individual joints, or by using three-dimensional coordinated for each joint position in the space. The transition between these representations is possible by the use of both forward and inverse kinematics. However, due to the non-uniqueness of solutions in the latter approach, an angular representation of the skeletal structure is generally preferred. Regarding the angular representation, multiple

notations can be used:

- *Euler angles*: Euler angles provide a human-friendly and intuitive method of expressing 3D rotations using a set of three parameters. However, the presence of singularities and the associated gimbal-lock problem, combined with the existence of 12 different Euler angle sequences, introduces complexity and ambiguity into the representation of rotations.

- *Quaternions*: Quaternions offer a solution to the main drawbacks of the Euler angle notation. They successfully mitigate the gimbal-lock problem offering a singularity-free representation which can be easly interpolated. However, it is important to notice that quaternions are more complicated to implement, they require normalization to maintain the mathematical properties, and they are specified by means of four parameters.

- *Exponential map/angle-axis*: The exponential map or angle-axis representation introduces a simpler notation that eliminates the need for normalization, which distinguishes it from quaternions. In addition, it mitigates the singularity problem of Euler angles, while maintaining a concise three-parameter representation. However, exponential maps are less intuitive and can be hard to interpret.

- *Rotation matrices*: Rotation matrices provide a complete and singularity-free representation of 3D rotations, which are commonly used in kinematics computation. Their main drawback lies in the representation redundancy , as it requires the specification of nine parameters for each rotation. This redundancy can potentially lead to numerical issues, especially when dealing with large datasets and models.

Various works have implemented training models using different representations in an attempt to identify potential performance differences between them. However, even if in some cases specific notations brought to better results, a single representation that consistently outperforms all the others remains to be identified. Similarly, the number of joints (or keypoints) used to represent a human pose depends on the level of detail with which the various parts of the human body need to be described. Generally, in the context of HMP, the main articulations of the body are considered, with the head, hands, and feet represented as a single keypoint, thus making no distinction between the fingers.

The choice of the representation depends on the specific requirements of the application at hand. Figure 2.1 shows an example of a human body skeletal representation based on the Skinned Multi-Person Linear Model (SMPL) model.

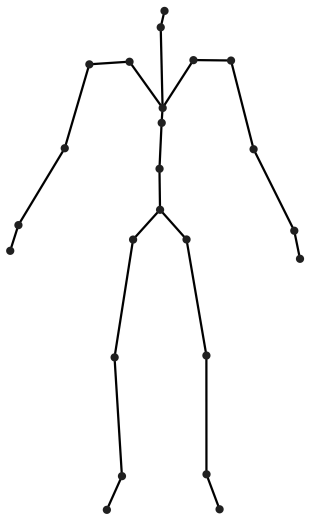


Figure 2.1: An example of a human body skeleton representation

Given a choice of notation, a human pose $\boldsymbol{x}$ can be represented as follows:

$$\boldsymbol{x} = (\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_J) \in \mathbb{R}^{J \times D} \tag{2.1}$$

where J is the number of skeleton joints and $\boldsymbol{e}_j \in \mathbb{R}^D$ is the representation of the j_{th} joint angle or position with D parameters, $1 \leq j \leq J$. For example, in the case of quaternions representation $D = 4$, while for rotation matrices $D = 9$. This notation can be easily extended to describe a whole body movement $\boldsymbol{X}_{1:N}$, represented by a sequence of N poses over time at a certain frequency:

$$\boldsymbol{X}_{1:N} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_N) \in \mathbb{R}^{J \times D \times N} \tag{2.2}$$

where $\boldsymbol{x}_i$ represents the human body pose at time stamp i, $1 \leq i \leq N$.
Therefore, given an input sequence $\boldsymbol{X}_{1:N}$ of length N (*seed sequence*), the problem of predicting the future M human poses $\hat{\boldsymbol{X}}_{N+1:N+M}$ (*target sequence*) can be formulated as follows:

$$\hat{\boldsymbol{X}}_{N+1:N+M} = \arg \max_{\boldsymbol{X}_{N+1:N+M}} P(\boldsymbol{X}_{N+1:N+M} | \boldsymbol{X}_{1:N}) \tag{2.3}$$

where $P(\boldsymbol{X}_{N+1:N+M}|\boldsymbol{X}_{1:N})$ is the probability for a target sequence to be the future for a given seed sequence. HMP models can be interpreted as estimators of the probability P, so given a seed sequence as input they output the most probable future. The length N of the seed sequence and the length M of the target sequence represent model parameters, which are also correlated with the number of frames per second (fps) of the motion sequence. Predictions up to $400ms$ are defined as *short-term*, while predictions up to $1000ms$ are defined as *long-term*.

2.2 Position-Velocity Recurrent Encoder-Decoder

PVRED is a DL model for HMP based on a RNN architecture [37]. Standard RNN approaches, whose input is only the sequence of human poses, tend to converge to a static pose after a few prediction frames or fail to generate natural looking sequences. To mitigate this problem, thus to improve long-term predictions, the authors proposed a novel architecture that also incorporates pose velocity information and a temporal embedding of the motion sequence frames. The latter, inspired by the positional word encoding applied in Natural Language Processing (NLP), is used to give to the model the information about the temporal relationships between the different frames of the sequence. The PVRED model is based on the RNN Encoder-Decorder (RED) architecture, to which it adds the velocity and positional frames embedding. PVRED is based on an exponential map representation for the joint angle rotations of the body skeleton, and it is designed to work with human motion sequences at 25 fps. In the following, the general structure of the RED architecture is described and then the details of the PVRED model are presented.

2.2.1 RNN Encoder-Decoder architecture

The basic architecture of PVRED is the standard RNN Encoder-Decoder (RED) model, which is used for sequence to sequence (seq2seq) tasks. It consists of two blocks: the encoder, which operates on an input sequence and gathers the relevant information through its hidden states, and the decoder, which exploits the information from the encoder and autoregressively generates the output sequence. Both the encoder and the decoder blocks use hidden states, which are vectors that embed the information about the past sequence up to the

current time stamp. The goal of the encoder is to encapsulate all the relevant information about the input sequence in its last hidden state, which is the only past information available to the decoder to predict the future sequence. Following the human motion representation as in Section 2.1, given the input seed sequence $\boldsymbol{X}_{1:N} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_N)$ of length N, the encoder hidden state $\boldsymbol{h}_i$ at time stamp i is updated by:

$$\boldsymbol{h}_i = f(\boldsymbol{h}_{i-1}, \boldsymbol{x}_i) \tag{2.4}$$

where f is a non-linear function, $\boldsymbol{x}_i$ is the input human pose at time stamp i, and the first hidden state $\boldsymbol{h}_0$ is initialised accordingly to some criteria, e.g. randomly. The decoder hidden states $\tilde{\boldsymbol{h}}_j$ follow the same structure:

$$\tilde{\boldsymbol{h}}_j = g(\tilde{\boldsymbol{h}}_{j-1}, \hat{\boldsymbol{x}}_{N+j}) \tag{2.5}$$

where g is again a non-linear function, $\hat{\boldsymbol{x}}_{N+j}$ is the predicted human pose at time stamp j, and the first hidden state $\tilde{\boldsymbol{h}}_0$ is initialised as the last encoder hidden state $\boldsymbol{h}_N$. The key idea is that the encoder contains all the information about the input sequence in its last hidden state, which is then passed to the decoder through the initialisation of its first hidden state. Both encoder and decoder hidden states have a fixed size, which is a hyperparameter of the Neural Network (NN) architecture.

The information contained in the hidden states of the decoder can then be applied to predict future poses using linear regression. Formally, for each prediction time stamp $j \in 1, ..., M$, the future j_{th} pose can be computed as:

$$\hat{\boldsymbol{x}}_{N+j} = \boldsymbol{W}\tilde{\boldsymbol{h}}_{j-1} + \boldsymbol{b} \tag{2.6}$$

where $\boldsymbol{W}$ represents the weights and $\boldsymbol{b}$ the bias, both learnable parameters. For the training phase, the Mean Squared Error (MSE) loss can be used which is defined as follows:

$$L = \frac{1}{M}\sum_{j=1}^{M} \|\hat{\boldsymbol{x}}_{N+j} - \boldsymbol{x}_{N+j}\| \tag{2.7}$$

where $\hat{\boldsymbol{x}}_{N+j}$ and $\boldsymbol{x}_{N+j}$ represent the predicted and the ground truth poses, respectively, at prediction time stamp j.

2.2.2 Position-Velocity RED architecture

PVRED improves the architecture of RED by feeding the model not only with the information about the past poses, but also with the pose velocities and their positional embedding. Moreover, instead of estimating directly the future pose as in 2.6, the decoder first predicts the human pose velocities and then, through a residual connection, the future pose. In addition, a Quaternion Transformation (QT) layer is added before computing the loss function. A description of the positional embedding and the overall structure of the NN architecture is presented below.

Positional embedding

The addition of positional embedding information is inspired by the field of NLP, which allows the model to understand about the relationships between poses at different time stamps. For example, if we think of a movement where a person sits down and then stands up from a chair, the individual poses between sitting and standing will be very similar, because the movement is basically the same but performed in the opposite order. By adding positional information, the model becomes able to distinguish between two poses that are almost equivalent, but one involves sitting and the other involves getting up from a chair. Positional encoding consists of representing the temporal position of a given frame trough by a real-valued vector. A simple approach could be to use a one-hot representation, which consists of encoding each frame in a vector of size equal to the sequence length, composed of all zeros except for a one corresponding to the current frame. However, this allows only fixed-length sequences to be handled. In PVRED, the positional encoding is implemented through sine and cosine functions. Each frame t is embedded by a vector $\boldsymbol{p}_t \in \mathbb{R}^{d^p}$, where d^p is the fixed size of the positional embedding. To allow operations such as linear combinations between the different input data, the size of positional embeddings d_p is set equal to the size of the human poses and their velocity representations. Assuming a seed sequence and a target sequence of length N and M respectively, for a given time stamp $t \in 1, ..., N, ..., N + M$, its positional embedding

$\boldsymbol{p}_t$ is then computed as follows:

$$\boldsymbol{p}_t(i) = \begin{cases} sin(t/10000)^{2i/d^p}, & \text{if } i \text{ is even} \\ cos(t/10000)^{2i/d^p}, & \text{if } i \text{ is odd} \end{cases} \tag{2.8}$$

where i represents the index of the positional embedding vector. This sinusoidal embedding allows the model to handle sequences of arbitrary length and to capture the temporal relationships between different frames helping in predicting natural looking poses.

Position-Velocity RNN

To improve long-term predictions, the RNN architecture of PVRED makes use of Gated Recurrent Unit (GRU) cells which are fed at each time step t with the human pose $\boldsymbol{x}_t$, the pose velocity $\boldsymbol{v}_t$, and the positional embedding $\boldsymbol{p}_t$. The pose velocity $\boldsymbol{v}_t$ is computed as the discrete time derivative of $\boldsymbol{x}_t$. The mathematical formulation of each GRU cell of the encoder at a given time stamp t is then as follows:

$$\begin{aligned} \boldsymbol{z}_t &= \sigma(\boldsymbol{U}_x^z \boldsymbol{x}_t + \boldsymbol{U}_v^z \boldsymbol{v}_t + \boldsymbol{U}_p^z \boldsymbol{p}_t + \boldsymbol{W}^z \boldsymbol{h}_{t-1}) \\ \boldsymbol{r}_t &= \sigma(\boldsymbol{U}_x^r \boldsymbol{x}_t + \boldsymbol{U}_v^r \boldsymbol{v}_t + \boldsymbol{U}_p^r \boldsymbol{p}_t + \boldsymbol{W}^r \boldsymbol{h}_{t-1}) \\ \boldsymbol{h}'_t &= tanh(\boldsymbol{U}_x^h \boldsymbol{x}_t + \boldsymbol{U}_v^h \boldsymbol{v}_t + \boldsymbol{U}_p^h \boldsymbol{p}_t + \boldsymbol{W}^h(\boldsymbol{r}_t \circ \boldsymbol{h}_{t-1})) \\ \boldsymbol{h}_t &= (1 - \boldsymbol{z}_t) \circ \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \circ \boldsymbol{h}'_t \end{aligned} \tag{2.9}$$

where $\boldsymbol{r}_t$ represents the recurrent gate, $\boldsymbol{z}_t$ the update gate, $\boldsymbol{h}_t$ the hidden state, and $\boldsymbol{U}$,$\boldsymbol{W}$ the variables and weights matrices respectively. The same mathematical formulation holds also for the decoder hidden state $\tilde{\boldsymbol{h}}_t$.

To predict the future human pose at time stamp $j \in 1, ..., M$, first the discrete time velocity is predicted and then it is added to the previous time stamp pose:

$$\begin{aligned} \boldsymbol{v}_{N+j-1} &= \boldsymbol{W}\boldsymbol{h}_{N+j-1} + \boldsymbol{b} \\ \boldsymbol{x}_{N+j} &= \boldsymbol{x}_{N+j-1} + \boldsymbol{v}_{N+j-1} \end{aligned} \tag{2.10}$$

where $\boldsymbol{W}$ and $\boldsymbol{b}$ represents the weights and the bias learnable parameters respectively. This operation is performed by a linear layer stacked on top of the GRU layer(s).

Quaternion Transformation

Since the exponential map notation for joint rotations suffers from singularities and discontinuities, PVRED integrates a QT layer to convert from exponential map to quaternion notation. This QT layer is integrated withing the end-to-end architecture and thus included in the overall training process. Given a three-dimensional vector in exponential map representation $\boldsymbol{e}$, the QT layer transforms it into a four-dimensional quaternion representation $\boldsymbol{q}$:

$$\boldsymbol{q}(i) = \begin{cases} cos(0.5\|\boldsymbol{e}\|_2) & i = 1 \\ \frac{sin(0.5\|\boldsymbol{e}\|_2)}{\|\boldsymbol{e}\|_2} \cdot \boldsymbol{e}(i-1) & i \geq 2 \end{cases} \tag{2.11}$$

where i represent the i_{th} element of $\boldsymbol{q}$. In addition, during the training phase, the derivative of the loss (based on the quaternions $\boldsymbol{q}$ and defined below) is computed with respect to $\boldsymbol{e}$ to allow backpropagation of the error and the optimisation of the parameters of the NN.

Loss function

The loss function L used during the training phase of the model is defined as follows:

$$L = \frac{1}{M} \sum_{j=1}^{M} \|g(\hat{\boldsymbol{x}}_{N+j}) - g(\boldsymbol{x}_{N+j})\| \tag{2.12}$$

where g represents the QT for both the prediction and the target which are originally expressed in exponential map notation. Since the only purpose of the QT layer is to compute the loss function, it is discarded at inference time. An overview of the PVRED architecture is shown in Figure 2.2.

Position-Velocity Recurrent Encoder-Decoder is a fairly simple NN architecture, being based on the RED model. However, the use of velocities and positional embeddings makes this model very competitive in the research landscape, as it performs very well compared to many other state of the art NN models. Several variants of the network can be created by changing the hyperparameters of the network, such as the size of the hidden states or the embeddings. In addition, higher complexity can be achieved by stacking multiple GRU layers. In the original paper, the authors obtained the best results by using 2 GRU layers.

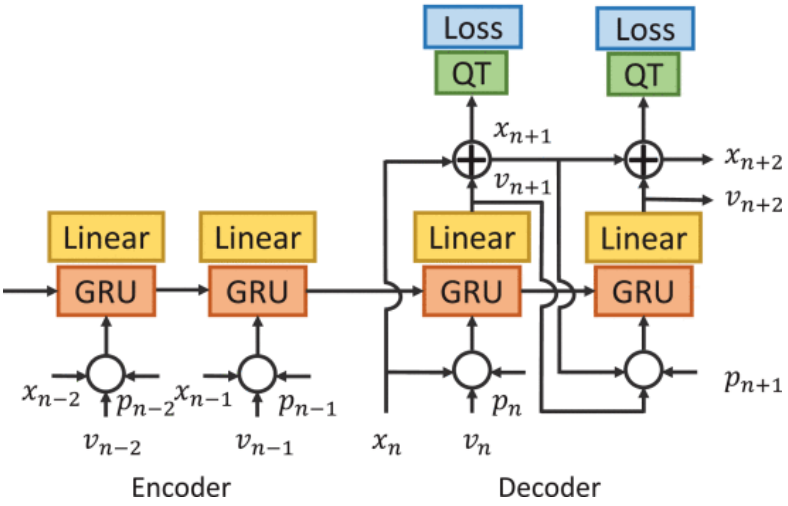


Figure 2.2: The architecture of Position-Velocity Recurrent Encoder-Decoder, taken from the original paper [37]. At time stamp t GRU cells are fed with human pose $\boldsymbol{x}_t$, pose velocity $\boldsymbol{v}_t$, and positional embedding $\boldsymbol{p}_t$. The output is computed as the sum of the previous predicted pose $\boldsymbol{x}_{t-1}$ and the current predicted velocity $\boldsymbol{v}_t$. A Quaternion Transformation layer is used to compute the loss in the quaternion joint angle representation.

2.3 History Repeat Itself

HRI is a DL model that introduces a new concept in the context of HMP. The key idea behind its architecture is that human movement tends to *repeat itself*, hence the name of the model. This is most obvious in periodic actions, such as walking or running, where the same motion patterns are continuously repeated at a certain frequency. However, this interpretation is also true in more complex movements, such as cooking or different sports activities. While in the first examples the individual sub-movements are very few and repeated frequently, in more complex actions the individual sub-movements are much more numerous and generally repeated at greater intervals in time. In order to find relevant historical information, HRI implements a motion attention mechanism that aims to find these repeated sub-actions in different movements. This information, combined with the past motion sequence, is processed by a GCN to learn the relationships between the different joints of the human skeleton, which is then used to predict future poses. In the following, it is presented the motion attention mechanism and the most relevant aspects of the HRI architecture.

2.3.1 Motion attention mechanism

One of the major drawbacks of RNN approaches is their limited ability to deal with long input sequences due to information loss. This is mainly because the information available to the decoder to predict future poses is only the one encoded in the last hidden layer of the decoder. Since it is fixed in size, this requires the NN to embed all the information in a very limited space, which inevitably leads to a loss of information. The general concept of the attention mechanism instead allows the model to create a dynamic embedding of the input sequence for each prediction time stamp, i.e. it is computed as a weighted combination of all the encoder hidden states. The weights, also called *attention*, allow the model to extract the aspects of the input sequence that are the most relevant for predicting the current time stamp pose.

In the context of HMP, some previous approaches have introduced a frame-wise attention mechanism, which aims to find relationships between individual frames. However, these approaches lead to ambiguous motion because the model is unable to distinguish between two similar poses that appear in completely different contexts. To solve this problem, HRI introduces a *motion-attention mechanism* that aims to find relationships between sub-sequences rather than individual frames. This allows the model to discriminate between similar poses that appear in completely different contexts.

With reference to the human motion representation described in Section 2.1, the motion attention model implemented in HRI is based on a decomposition of the input motion sequence $\boldsymbol{X}_{1:N}$ into $N - M - T + 1$ sub-sequences of length $M + T$. The motion prediction block of the model architecture, which will be described later, exploits past M frames to predict future T frames. The motion attention mechanism is based on the same formalism used in the Transformer architecture, namely *key-value* pairs and a *query*. Each key-value pair is associated with a motion sub-sequence $\boldsymbol{X}_{i:i+M+T-1}$, $i \in 1, 2, ..., N - N - T + 1$, where the key corresponds to first M frames, while the value to the whole sub-sequence. The query, on the other hand, is associated to the latest sub-sequence $\boldsymbol{X}_{N-M+1:N}$ of length M which will then represent the input of the predictor together with the dynamic context of the seed sequence. In order to avoid predicting too high frequency motion which do not represent natural movements of the human body, HRI introduces a Discrete Cosine Transform (DCT) on the temporal dimension of the values so that to truncate high frequencies. Therefore, each value

$\boldsymbol{X}_{i:i+M+T-1}$ is mapped into $\boldsymbol{V}_i \in \mathbb{R}^{K\times(M+T)} = \mathbb{R}^{(J\times D)\times(M+T)}$, where each row contains the DCT coefficients of one joint coordinate sequence. At each prediction step, a dynamic context of the past motion sequence is obtained as a weighted sum of the values, where the weights (i.e. the attention scores) are computed from the keys and the query. The latter are mapped into vectors of equal size by two functions f_q, $f_k : \mathbb{R}^{K\times M} \rightarrow \mathbb{R}^d$ respectively, implemented through NN layers:

$$\begin{aligned} \boldsymbol{q} &= f_q(\boldsymbol{X}_{N-M+1:N}) \\ \boldsymbol{k}_i &= f_k(\boldsymbol{X}_{i:i+M-1}) \end{aligned} \tag{2.13}$$

where $\boldsymbol{q}, \boldsymbol{k}_i \in \mathbb{R}^d$ for $i \in \{1, 2, ..., N-M-T+1\}$. Then, for each key the attention score is computed as follows:

$$a_i = \frac{\boldsymbol{q}\boldsymbol{k}_i^T}{\sum_{i=1}^{N-M-T+1} \boldsymbol{q}\boldsymbol{k}_i^T} \tag{2.14}$$

To avoid getting negative attention scores which would cause numerical issues, the NN layers implementing f_q and f_k have *ReLU* as their activation function. The dynamic context U for the current prediction time stamp is computed as a weighted sum of the values:

$$\boldsymbol{U} = \sum_{i=1}^{N-M-T+1} a_i \boldsymbol{V}_i \tag{2.15}$$

where $\boldsymbol{U} \in \mathbb{R}^{K\times(M+T)}$. The output of the attention model, combined with the last sub-sequence of size M (i.e. the key), is then given as input to the predictor to estimate the future T poses, that is the sequence $\hat{\boldsymbol{X}}_{N+1:N+T}$.

2.3.2 Prediction model

To predict future poses, the DCT representation described above is used to represent the temporal information of the past human poses. The latter, combined with the dynamic context vector, is processed by a stack of GCN layers to capture the spatial dependencies between the joint coordinates or angles used to represent skeletal poses. Therefore, the predictor is divided into two separate blocks, the temporal and the spatial encodings:

- *Temporal encoding*: To predict the future sequence $\boldsymbol{X}_{N+1:N+T}$, the key $\boldsymbol{X}_{N-M+1:N}$

is extended to $M+T$ frames by replicating the last one, i.e. $\boldsymbol{x}_N$, for T times. Then, it is computed its DCT encoding, $\boldsymbol{D} \in \mathbb{R}^{K\times(M+T)}$, which is concatenated with the dynamic context vector $\boldsymbol{U}$, and then passed to the GCN layers. This encodes the information about the entire sequence history, focusing on the last M frames.

- *Spatial encoding*: To capture the relationships between the links of the human body, the skeleton is modeled through a fully connected graph of K nodes. The input of a GCN layer p consists of a matrix $\boldsymbol{H}^p \in \mathbb{R}^{K\times F}$, where each row is represented by a feature vector of size F. For example, the first GCN layer takes as input the concatenation of the DCT representation of the last sub-sequence $\boldsymbol{D}$ and the dynamic context vector $\boldsymbol{U}$, therefore $F = 2(M+T)$. The output of a GCN layer is then computed as follows:

$$\boldsymbol{H}^{p+1} = \sigma(\boldsymbol{A}^p \boldsymbol{H}^p \boldsymbol{W}^p) \in \mathbb{R}^{K\times\hat{F}} \tag{2.16}$$

where $\boldsymbol{A}^p \in \mathbb{R}^{K\times K}$ is the trainable adjacency matrix representing the degree of correlation between the nodes in the graph, and $\boldsymbol{W}^p \in \mathbb{R}^{F\times\hat{K}}$ is a set of trainable weights; $\sigma()$ is the activation function. A forward pass on multiple GCN layers stacked on top of each other gives the output of the model, from which the coordinates or rotation angles of each joint for the predicted sequence $\hat{\boldsymbol{X}}_{N-M+1:N+T}$ can be obtained by applying the Inverse Discrete Cosine Transform (IDCT) transform.

Loss function

HRI has been designed to work with both 3D coordinated and joint angles in exponential map notation, so two separate loss functions are defined depending on which representation is used during training. For the 3D joint coordinate representation, the Mean Per Joint Position Error (MPJPE) is used, which is defined for each training sample as follows:

$$L_{3D} = \frac{1}{J(M+T)} \sum_{t=1}^{M+T} \sum_{j=1}^{J} \|\hat{\boldsymbol{p}}_{t,j} - \boldsymbol{p}_{t,j}\|^2 \tag{2.17}$$

where $\hat{\boldsymbol{p}}_{t,j} \in \mathbb{R}^3$ represents the 3D coordinates of the j_{th} joint of the t_{th} frame for the predicted sequence. The same notation also holds for the target sequence

$\boldsymbol{p}_{t,j}$. In angle notation, the loss instead corresponds to the average l_1 distance defined as follows:

$$L_{ang} = \frac{1}{K(M+T)} \sum_{t=1}^{M+T} \sum_{k=1}^{K} |\hat{\boldsymbol{x}}_{t,k} - \boldsymbol{x}_{t,k}| \tag{2.18}$$

where $\hat{\boldsymbol{x}}_{t,k}$ is the predicted k_{th} angle for the t_{th} frame. Instead, $\boldsymbol{x}_{t,k}$ represents the ground truth. A representation of the overall structure of the HRI network is shown in Figure 2.3.

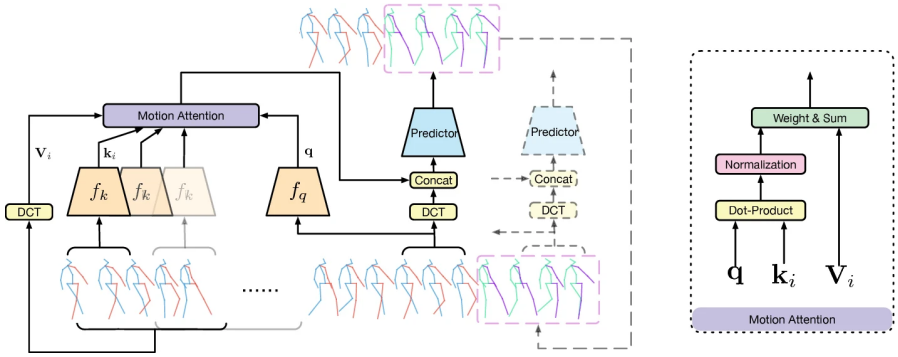


Figure 2.3: The NN architecture of History Repeat Itself, taken from the original paper [25]. The attention mechanism tries to find relationships between past motion sub-sequences to computed a dynamic context vector. This information, together with the last motion sub-sequence, is processed by the predictor to generate the next future sub-sequence.

In practice, the two functions f_q and f_k for the attention mechanism are implemented by two 1D convolutional layers, each followed by a ReLU activation function. The total number of stacked GCN layers is 12. Compared to the Position-Velocity Recurrent Encoder-Decoder model, History Repeat Itself represents a more complex architecture due to the motion attention mechanism and the stacking of several GCN layers, which highers the total number of NN parameters. In particular, the motion attention mechanism allows the model to learn about common patterns of the human body movements, and then use this information to generate future sequences. However, this attention mechanism may itself be one of the factors causing the model to perform poorly in terms of generalisation. If the model is used to make predictions about movements other than those seen during training, the attention mechanism may be strongly biased towards the actions in the training dataset.

3

Deep Transfer Learning

In recent years, Deep Learning (DL) models have shown significant improvements in solving very diverse problems in many fields. This is mainly because, compared to more standard Machine Learning (ML) algorithms or statistical approaches, these Deep Neural Networks (DNN) are extremely capable of modelling and learning from data with non-linear relationships. This characteristic is a consequence of the structure and mathematical complexity of these models, which are built by stacking multiple Neural Network (NN) layers, each with its own set of parameters and activation function. However, although the complexity of these models is their strength, it also highlights one of their main limitations. In fact, DL models are trained by updating their parameters to obtain the best prediction through back-propagation and optimisation algorithms. Since the number of parameters of DL models is usually in the millions or even more, the amount of data required to train them optimally is huge. In the context of supervised learning, the data must also be labelled, a process that is typically done manually, making it a very time-consuming and expensive task. In addition, the training process itself is very time consuming and requires a lot of computing power, which is often not affordable for small research groups. The technique of Transfer Learning (TL) can be used to mitigate these problems by reducing the amount of domain-specific data and the training time required to achieve good performance for the task at hand. This Chapter introduces the general concept of TL and then focuses on its application in the case of DL models. It also describes the main techniques used for Deep Transfer Learning (DTL) and presents their main advantages and criticisms.

3.1 Transfer Learning

TL is a technique used in the field of ML which consists in transferring the knowledge acquired by a pre-trained model on a large dataset to a more specific downstream task. More specifically, this technique is used when the availability of data for a specific task is low and it is not possible or too costly or difficult to collect more data. If a ML model is trained on a dataset that is too small, the amount of data is insufficient for the model to gain enough knowledge to achieve good generalisation performance. This problem is called *overfitting*, which in practice occurs when the model becomes very good at making predictions on the training set, but its performance on the test set deteriorates rapidly. This usually happens when the training set is too small, so that it is easy to get good results on the latter, but not large enough for the model to acquire general knowledge about the type of data it is dealing with.

The idea behind TL is to use a pre-trained model that has been trained on a large dataset as a starting point for building a new model to solve a specific task (called the target domain). The basic concept of this approach is that the pre-trained model has already learned a lot of knowledge from a large dataset (called the source domain), which is then used to improve performance for the target task. This knowledge is then incorporated into the new model, which is tuned on the small dataset available for the specific task. Conversely, attempting to build a model from scratch when a small dataset is available would hinder the acquisition of domain knowledge, resulting in strong biases and poor generalisation performance. As pointed out by Yosinski et al. [39], the transferability of knowledge from a pre-trained model is more effective when the source and target domains have similar characteristics. However, TL approaches can also be applied when the two domains are not strictly related, and such techniques are generally still more effective than randomly initialising the weights of the NN.

In recent years, TL has become so common in various ML and data mining applications that many different techniques have been developed depending on the type of data and the type of model or algorithm used [38]. This thesis focuses on the technique of TL applied to DL models for the task of Human Motion Prediction (HMP).

3.2 Deep Transfer Learning

While the TL technique is a versatile approach applicable to standard ML algorithms, its significant advances are particularly notable within the context of DL models, which take the name DTL. This importance is primarily due to the fact that DL models require large amounts of data that may not be available, especially for very specific tasks. Here, TL approaches play a key role in improving model generalisation and mitigating the overfitting problem. The basic idea remains the same as that of traditional TL: instead of initialising a fresh model with random parameters for the downstream task, they can first be taken from a pre-trained model and then optimised through training iterations. Several techniques have been proposed in recent years, each offering different strategies. Some focus only on parameter optimisation, maintaining the pre-trained NN architecture. Others introduce the concept of freezing certain layers or adding new ones within the NN. This Section provides an overview of the main techniques currently used for DTL.

3.2.1 Deep Transfer Learning approaches

A systematic categorisation of different DTL approaches can be made based on the relationship between the source domain and the task domain [14]. Two categories can be identified: *homogeneous* and *heterogeneous*. The first refers to scenarios where the source and task domains are of the same nature. The latter, on the other hand, refers to cases in which the two domains differ in their characteristics. It is important to note that the categorisation of domains as homogeneous or heterogeneous is subjective and can take on different facets depending on the specific application context. Another categorisation can be based on the labelling context of the source and target datasets. Three distinct categories can be identified: *transductive*, where only labelled source data are involved; *inductive*, where both source and target data are labelled; and *unsupervised*, where neither source nor target domains are labelled. However, in the context of this thesis, a more interesting classification is certainly based on the type of DTL approaches that can be used. In this context, four different categories of approaches are identified as pointed out by Iman et al. [14]: 1) *instance-based*, 2) *feature-based*, 3) *model-based*, and 4) *relational/adversarial-based*. Below is a description of each of these categories:

- *Instance-based*: The Instance-based approach is a solution that can be applied in the case of homogeneous domains. It involves assigning weights to samples in the source domain and then using these weighted samples in the target domain. This approach makes two main contribution: first, it helps to increase the size of the target dataset, which also improves its distribution. Secondly, the weights help the new model to focus only on the relevant information from the source domain.

- *Feature-based*: The category of feature-based TL approaches focuses on mapping instances or features from both the source and the target domains into a homogeneous space in order to create a more uniform data representation. Within this category, two distinct ones can be identified: *Asymmetric* and *Symmetric* approaches. The first consists in mapping the features of the source domain to the corresponding features of the target domain. The latter, on the other hand, maps both source and target features in a common latent space. Overall, this feature-based approach helps to reduce the lack of information in the target domain by accumulating knowledge from the source domain.

- *Model-based*: Model-based approaches focus on knowledge transfer at the model level, achieved by sharing parameters between a pre-trained model and a newly initialised model. The basic concept behind these approaches is that a pre-trained model, having been exposed to a large dataset, has learned significant information about the data. This knowledge can then be transferred to downstream task models by sharing the parameters of the layers that build the NN architecture, thus improving their performance.

- *Relational/Adversarial-based*: Relational or adversarial approaches focus on extracting features, common patterns, and correlations between the source and the target domains. Relational-based approaches use rules or logical relationships, while adversarial approaches are inspired by Generative Adversarial Network (GAN).

An overview of the classification of DTL approaches described above is shown in Figure 3.1. Currently, the most common class of approaches in DTL is represented by the model-based ones as they allow to easily perform a domain adaptation from the source to the target domains by adjusting the parameters of the network.

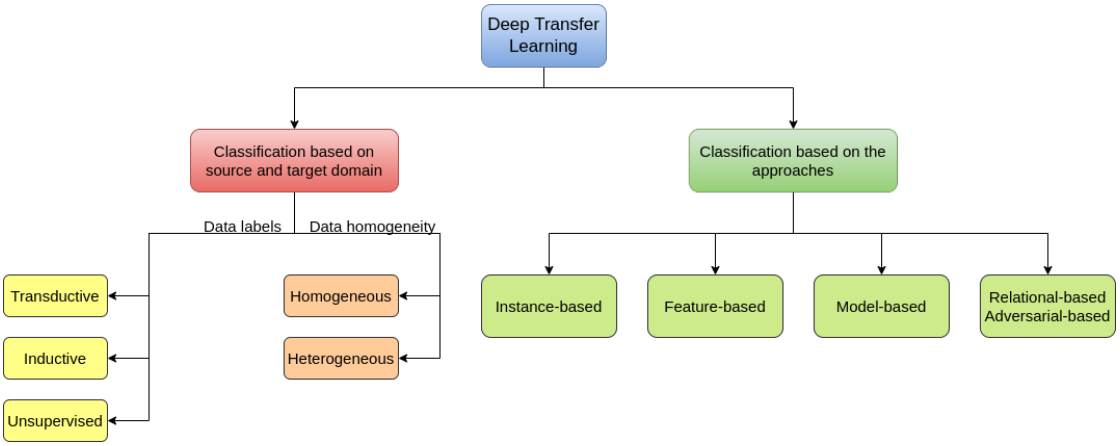


Figure 3.1: An overview of the classification of the main Deep Transfer Learning techniques, based on the relationship between the source and target domains, and on the different classes of approaches.

3.2.2 Model-based Deep Transfer Learning

Traditional DL approaches typically involve initialising the parameters of the NN at random or according to other protocols, followed by iterative optimisation to achieve the best performance on the target task. Model-based approaches for DTL allow the use of a pre-trained model on a similar task as a starting point to build the new model specific to the target task. However, two problems can arise: *catastrophic forgetting* and a *biased pre-trained model*. The first refers to the situation where the knowledge accumulated in the pre-trained model is completely lost during the adaptation to the target task, thus completely loosing the advantages of DTL. The biased pre-trained model, on the other hand, occurs when the pre-trained model has a strong bias towards the source domain data it encountered during training. This bias greatly reduces the ability of the model to effectively adapt to novel data within the target domain. Therefore, when applying DTL techniques, it is important to mitigate these two issues in order to maximise the effectiveness of the model adaptation.

Numerous techniques have been proposed for model-based DTL, generally involving a combination of *pre-training* a model on a source domain, selectively *freezing* and/or *fine-tuning* specific layers of the pre-trained model, and possibly *adding fresh layers* to enhance the model's capabilities. There are still no standardised techniques and the choice between these methods depends heavily on the specific application and the type of architecture of the NN to which TL is applied. The following paragraphs describe each of these strategies, highlighting

what they consist of and their main advantages and challenges.

Fine-tuning

Rather than randomly initialising the parameters of the NN, the fine-tuning involves initialising them using the weights from a pre-trained model, which requires the two NN to have the same architecture. Later, these values are adjusted to tune the model for the specific downstream task. However, this approach can potentially lead to catastrophic forgetting because of the pre-trained weights that are changed too much. In practice, several strategies can be used to mitigate this problem. These generally consist of using a lower learning rate to allow smaller changes in the parameters, and fine-tuning only specific layers of the network. Fine-tuning is a very common approach used in practice that allows a direct way of transferring knowledge and it has a relatively simple implementation process.

Freezing

In DNN, the first layers are typically responsible for extracting high-level data features. Subsequently, middle layers process these features to extract finer, lower-level details, which are finally processed by the last layers to solve the task at hand, i.e. to make predictions or classify data. In the context of DTL, another important strategy consist of freezing some layers, which corresponds to avoid their updating during the training phase. This is usually done on the first layers, which helps to preserve the main features that the pre-trained model has gathered about the source data, thus reducing the risk of catastrophic forgetting. However, this strategy can introduce a significant bias from the pre-trained model, which has the potential to hinder the tuned model for a downstream task to learn novel low-level features. This issue can be potentially enhanced if the source and target domain are not homogeneous. This strategy is more complex to apply than fine-tuning, as it requires in-depth knowledge of the NN architecture to identify which layers are important to freeze and which are not.

Progressive learning

Progressive Neural Network (PNN) mimics the human learning process, which involves building on previously learned skills. This strategy lies between

fine-tuning and freezing, and is an effective technique for mitigating both catastrophic forgetting and the strong bias of the pre-trained model. This strategy consists of freezing the entire pre-trained NN while stacking additional layers on top of it. This allows the reuse of the features acquired by the pre-trained model and their subsequent elaboration to address the specific task. Progressive learning facilitates the preservation of the existing knowledge and helps to use its features in the context of the new task. Clearly, understanding how many and which layers to add to the network and where to position them is a challenging task that requires a lot of experience to apply Progressive Learning optimally.

There is no specific DTL technique that can be applied generally, as it depends on many factors, including the NN architecture, its total number of parameters, and the relationship between the source and the target domains. However, some general considerations can be made. Firstly, a large and diverse source dataset is crucial to avoid having a strong bias or insufficient knowledge from the pre-trained model. DTL is generally more effective when the source and the target domains are homogeneous, which helps with knowledge transfer. Similarly, the reuse of pre-trained features leads to optimal results in cases where the datasets are similar between source and target. Furthermore, the potential of DTL methods is enhanced when the target dataset is very limited in size, as training a fresh new model is likely to lead to poor generalisation performance due to data scarcity. Therefore, the main advantages of DTL are the improved model performance when data is limited, but also the reduction of training epochs. In fact, when TL techniques are applied, DL models tend to converge to the optimal configuration faster then when a fresh new model is initialised. This helps to reduce the training time and cost, making the training of complex NN feasible even when computational resources are limited.

However, it is important to note that finding the best configuration of the parameters for training DNN, including the context of DTL, requires a lot of expertise in the field. Nevertheless, good results can also be obtained experimentally by testing different configurations of the various techniques.

4 Metrics

In order to quantitatively evaluate the results of the experiments described in Chapter 5, both metrics for assessing the effectiveness of Transfer Learning (TL) techniques and for evaluating the performance of Human Motion Prediction (HMP) models are introduced and described in this Chapter. It also introduces the *zero-velocity* model, a baseline that is used to benchmark HMP models.

4.1 Human Motion Prediction metrics

To measure the quality of the motion prediction, it is necessary to compare the entire predicted sequence with the target one, which represents the ground truth. Given the complexity of the human body structure, the metrics should take into account all frames and all joint rotation angles or coordinates describing a single pose in the predicted sequence. The metrics that are considered in this work, which are described below, are inspired by two papers proposed by Aksan et al. [2, 3]. Their formulation is based on the mathematical description of the human motion representation introduced in Section 2.1. For each metric, its definition and computational details are provided for a given frame t and for a given test batch $\boldsymbol{X}_{test}$.

4.1.1 Euler Error

The Euler Error is one of the most commonly used metrics at the time of writing to evaluate and compare different models for the accuracy of joint angle

predictions. For a given time stamp t and a test batch $\mathbf{X}_{test}$, the Euler Error L_{eul} can be computed as follows:

$$L_{eul}(t) = \frac{1}{|\mathbf{X}_{test}|} \sum_{X_t \in \mathbf{X}_{test}} \sum_{j=1}^{J} \|\hat{\boldsymbol{\alpha}}_j - \boldsymbol{\alpha}_j\|_2 \tag{4.1}$$

where $\hat{\boldsymbol{\alpha}}_j$ and $\boldsymbol{\alpha}_j$ represents the rotation of the j_{th} joint expressed in Euler angle notation for the predicted and target sequences, respectively, at frame t. The Euler Error evaluates the average Euclidean distance between the predicted and the target sequences. To compute an overall metric for the entire predicted sequence rather than a single frame t, the average over all the prediction frames can be computed. The Euler Error metric has a total of 9 variants, based on the different Euler angle sequences.

4.1.2 Joint Angle Difference

The Joint Angle Difference is based on quantifying the angular rotation required to align the predicted joint with the direction of the ground truth. Unlike the Euler Error, this metric does not depend on the specific parameterisation of rotation, as it is based on rotation matrices. The Joint Angle Difference metric L_{angle} is therefore defined as follows:

$$L_{angle}(t) = \frac{1}{|\mathbf{X}_{test}|} \sum_{X_t \in \mathbf{X}_{test}} \frac{1}{J} \sum_{j=1}^{J} \|\log(\tilde{\mathbf{R}}_j)\|_2 \tag{4.2}$$

where $\tilde{\mathbf{R}}_j$ is the rotation matrix needed to align the predicted direction of the j_{th} joint with the target joint. Similarly as before, an overall metric for the entire predicted sequence can be computed by averaging over all predicted frames in the sequence.

4.1.3 Positional Error

Positional Error measures the accuracy of body keypoint positions in the 3D space. Instead of measuring the error based on the rotation angles, this metric is based on a comparison between the 3D coordinates of the body keypoints of the predicted and target sequences. To obtain the 3D coordinates of the keypoints, the forward kinematics can be computed by moving along the kinematic tree

that describes the human body skeleton. Therefore, the Positional Error L_{pos} is computed as the Euclidean distance between the 3D coordinates of paired keypoints between the predicted and target sequences. For a given time stamp t, the Positional Error can be computed as follows:

$$L_{pos}(t) = \frac{1}{|\mathbf{X}_{test}|} \sum_{X_t \in \mathbf{X}_{test}} \frac{1}{J} \sum_{j=1}^{J} \|\tilde{\boldsymbol{p}}_j - \boldsymbol{p}_j\|_2 \tag{4.3}$$

where $\tilde{\boldsymbol{p}}_j$ represents the 3D coordinates of the j_{th} keypoint for the predicted sequence, while $\boldsymbol{p}_j$ refers to the target sequence. By averaging over all prediction frames, the total error of the predicted sequence can be computed. Unlike the other two metrics, Positional Error somehow weights the impact of each keypoint in the overall error. In fact, forward kinematics depends on the length of the links, so given the same angular error for two links of different lengths, the Positional Error will be much higher for a long link, rather than for a short one. This also reflects our perception when we make a visual evaluation. In fact, a shoulder rotation error of a given angle is certainly worse than a wrist rotation error of the same angle.

4.1.4 Qualitative evaluation

Another approach that can be used to assess the quality of predicted sequences, is a qualitative assessment based on graphical visualisation. Although this is not an objective metric, it is often very effective because it is easy to interpret and can highlight common patterns of error that would otherwise be almost impossible to detect using quantitative metrics alone. For example, if a model makes very good predictions for the upper part of the body but it is completely wrong for the lower part, the metrics described above would not be able to detect this problem because they take into account all the joints in the skeleton. Instead, by visualising different predicted sequences, this problem would be easily detected. Visualising predicted sequences can also be very useful when overlapped with the corresponding target sequences to visually highlight which joints have the greatest error. Of course, similar considerations could be made by computing the metrics independently for each joint and comparing them; however, this approach would make the evaluation too complex, so a graphical visualisation is preferable.

4.2 Zero-Velocity baseline

Zero-Velocity is a naïve HMP model introduced as a baseline to assess the predictive capabilities of state of the art models. As the name suggests, this model produces a prediction with Zero-Velocity, i.e. the human skeleton stands still at the position of the last frame of the seed sequence. In other words, the last pose of the input sequence is replicated as many times as there are prediction frames. Although this model may seem useless because it makes no predictions, when it was introduced it outperformed most existing state of the art models [27]. The prediction of human motion is a very difficult task given the complexity of the human body structure. Predicting that a person will move is generally riskier than predicting that the person will remain stationary. This is because if the predicted motion is in the opposite direction to the ground truth, the error is double that of a stationary prediction. In the context of HMP, making a prediction in the opposite direction to the true one is quite common, given the ambiguity and the many factors that can influence a person's movement. In fact, it is not uncommon for even a human being to make such mistakes. Therefore, a comparison with the Zero-Velocity model may be of interest to understand when its predictions are better than a static one, so as to evaluate its potential.

4.3 Transfer Learning metrics

The TL technique has now become a very common approach in the field of Deep Learning (DL). In general, to evaluate the effectiveness of this approach with respect to training a new model from scratch, one compares the performance of the two models on the task of interest. For example, for an object detection task, the Intersection over Union (IoU) score of the two models can be compared and the effectiveness of TL can be associated with the improvement of this metric. Similarly, in the context of HMP, this comparison can be based on the metrics described in Section 4.1, which can be computed by using the best model obtained during the training phase. For example, the average Euler Error or Positional Error in the test set along the prediction frames can be compared between a model trained from scratch and another model initialised with pre-trained weights. However, this simple evaluation does not really highlight the benefits of TL. In particular, as suggested by Zhu et al. [40], it is important

to consider two main aspects when evaluating a TL approach:

- *Mastery*: Mastery refers to how well the learned model, starting from a pre-trained one, performs in the target domain.
- *Generalisation*: Generalisation refers to the ability of the pre-trained model to quickly adapt to the target domain.

In this regard, the authors have reviewed the main metrics that have been proposed in recent years to assess these two aspects of TL, mastery and generalisation. It is important to note that their work focused on the context of Reinforcement Learning (RL) rather than DL, which is a different area of Machine Learning (ML). However, although some of these metrics are specific to RL, others can be directly applied or suitably adapted to the context of DL. Therefore, following Zhu et al. [40], the metrics that will be used to evaluate the effectiveness of TL approaches for HMP are *Jumpstart Performance*, *Minimal Error Performance*, and *Time To Threshold*.

All of these metrics are based on how the error of the model evolves over the different epochs of the training phase. For a given model, say A, trained for N_A epochs, the error during training can be represented by a vector $\boldsymbol{e}_A$ of length N_A, where $\boldsymbol{e}_A(i)$ corresponds to the error at the training epoch $i \in 1, 2, ..., N_A$. There is no specific type of error or score that must be used, rather these metrics can be generalised with appropriate adjustments when necessary. In the specific context of the work of this thesis, the aim is to compute these TL metrics to compare the performance of a DL model trained from scratch with a model obtained by using some TL technique. However, their description and mathematical formulation below is generalised for a comparison between two generic models, A and B.

Jumpstart Performance

Jump-Start Performance (JSP) compares the performance of the two models after the very first training epoch, and therefore assess the goodness of the predictions of the two when initialised. When comparing a pre-trained model with another model starting from scratch, JSP is an index of how much knowledge the pre-trained model has from the source dataset. Mathematically, Jump-Start Performance JSP can be computed as:

$$JSP = \boldsymbol{e}_A(1) - \boldsymbol{e}_B(1) \tag{4.4}$$

where $\boldsymbol{e}_A(1)$ and $\boldsymbol{e}_B(1)$ represent the error after the first training epoch for the two models A and B respectively. In this particular case, a positive JSP means that model A performs worse than model B.

Minimal Error Performance

Minimum Error Performance (MEP) corresponds to the difference between the minimum error achieved by the two models during training. The latter can be computed separately for each model as follows:

$$\begin{aligned} MEP_A &= \min\{\boldsymbol{e}_A(i) | i \in 1, 2, ..., N_A\} \\ MEP_B &= \min\{\boldsymbol{e}_B(i) | i \in 1, 2, ..., N_B\} \end{aligned} \tag{4.5}$$

Therefore, the MEP can be computed as:

$$MEP = MEP_A - MEP_B \tag{4.6}$$

A positive value for this metrics means that model A reaches an overall minimum error which is lower compared to model B.

Time To Threshold

Time To Threshold (TTT) is a metric related to the number of epochs needed for a given model to reach a given error threshold. In this particular case, given two models A and B, the threshold T is defined as the maximum between the minimum error of the two models:

$$T = \max\{MEP_A, MEP_B\} \tag{4.7}$$

This definition allows to have a threshold that is always reached by both the models, thus avoiding numerical problems in the computation of the metric. Then, the number of epochs t_A needed by model A to reach the error T can be defined as the epoch i at which the error goes below T for the first time; an analogous definition for model B:

$$\begin{aligned} t_A &= \min_{i \in (1,2,...,N_A)} \{\boldsymbol{e}_A(i) \leq T\} \\ t_B &= \min_{i \in (1,2,...,N_B)} \{\boldsymbol{e}_B(i) \leq T\} \end{aligned} \tag{4.8}$$

Finally, the Time To Threshold TTT is defined as:

$$TTT = t_A - t_B \tag{4.9}$$

Similar to JSP, positive values for TTT mean that model A performs worse than model B as it requires more training epochs to reach the threshold.

In some other cases, the threshold T can be defined as the minimum error of one of the two models, rather than the maximum between the two. For example, assuming $T = MEP_B$, three different cases can be identified for TTT: 1) $TTT > 0$, so the minimum error of A is lower than that of B, and model A reaches T in fewer training epochs. 2) $TTT < 0$, meaning that the minimum error of A is still lower than the minimum error of B, but model B reaches T faster. 3) TTT is undefined, which happens when model B reaches a minimum error lower than that of A. This definition of T can be useful to include the MEP, and thus to compute a more comprehensive metric.

These metrics described above can be computed separately for the training error, validation error, test error, or even the training loss. They can also be computed for the different HMP metrics described in Section 4.1, i.e. Euler Error, Joint-Angle Difference, and Positional Error. Obviously, computing all possible combinations would be too confusing, so it is important during the experiments to select the ones that are most relevant for drawing conclusions.

5

Experimental setup

In order to investigate the transferability of knowledge from Deep Learning (DL) models trained for the task of Human Motion Prediction (HMP), numerous experiments have been carried out to understand whether Transfer Learning (TL) techniques can bring benefits to the quality of the prediction when limited datasets are available. This Chapter motivates the choice of the state of the art DL models and the datasets among those available used for the experiments, and describes some of the main preprocessing steps required to perform the various tests. In addition, a complete list of the experiments carried out is presented, with technical details, together with a description of the resources available and the selected development environment.

5.1 Deep Learning Models and Datasets

The choice of DL models and datasets used in the experiments evolved in parallel, as it was important to find models that had been trained, or at least had been designed to be trained, on common datasets in order to allow for a fair comparison between them. Regarding the choice of DL models, it was important to choose them to be representative of the current state of the art algorithms for HMP, as it would not have been possible to run experiments on all available models due to limited resources. The choice of datasets, on the other hand, was guided by two main aspects. It was important to identify those that had already been used to train the models, so that they were already set up for that

type of data, and others that were closer to a Collaborative Robotics context. The first would have been used as the source domain, while the latter as target domains. This Section provides a detailed description of the motivations behind the choice of models and datasets, describing their main characteristics useful for the objectives of this thesis.

5.1.1 Models selection

As described in the state of the art review in Section 1.4, DL approaches for the task of HMP can be divided into three main classes based on the type of Neural Network (NN) underlying their architecture, which can be identified as Recurrent Neural Networks (RNNs), Graph Convolutional Networks (GCNs) and Generative Adversarial Networks (GANs). The latter category, i.e. GANs, were not considered for the experiments, mainly because of their non-deterministic behaviour, which can lead to unreliable results. Given that the experiments were characterised by target datasets of small size, differences in predictions due to their non-determinism would lead to significant differences in the results for the same input, making their analysis unreliable. Furthermore, these type of approaches are generally used to predict multi-future sequences, which is outside the scope of this thesis. Therefore, the choice of models fell on the other two categories, i.e. RNNs and GCNs.

It was then important to consider models for which the code used by the authors to build, train, and test the model was available. This was essential not only to avoid having to build the whole model from scratch, but more importantly to have the guarantee of using the same model that was described in the paper. In fact, it was fundamental to use pre-trained models that reflected the state of the art. Often papers omit a lot of details to provide a more high-level description of the architecture, so it would be almost impossible to fully replicate the NN. In addition, models for which code is available generally provide training checkpoints from which the authors have derived the results shown in the corresponding papers. This save training time, as the trained model in the source domain is already available.

As a result of the above considerations, the models chosen are those described in Sections 2.2 and 2.3, i.e. Position-Velocity Recurrent Encoder-Decoder (PVRED) and History Repeat Itself (HRI) respectively. The first is representative of the class of approaches based on RNN architecture, while the latter on GCN

and the attention mechanism. Both models were trained on motion sequences at 25 frames per second (fps) and tested on predicting sequences of 25 frames, which corresponds to a prediction of 1 second. The set of hyperparameters proposed by the authors was used, except for some modification that will be described below in the experiments Section. Some additional details on the configurations of these models are given in the following paragraphs.

PVRED

PVRED was pre-trained on Human 3.6M (H36M) as the source dataset to predict sequences of 25 frames, which is the same as when testing. The model uses the rotation angle representation of the human body skeleton, in particular the exponential map notation. Two different configurations of this model's architecture were used for the experiments, corresponding to NN with one and two RNN layers.

HRI

HRI was also pre-trained on H36M as the source dataset to predict sequences of 10 frames instead of 25. This is because of its network architecture and the motion attention mechanism. In fact, as described in Section 2.3, HRI is designed to predict window sequences of M frames in a single step, which is then autoregressively fed as input to the model to obtain longer sequences. For windows of 10 frames, the prediction step is repeated three times in order to obtain futures of 25 frames. The model was set up to work with both 3D coordinates and rotation angles to represent the body skeleton; for the experiments, exponential map notation was used.

5.1.2 Datasets selection

The choice of the datasets was closely related to the choice of the models, as it was important to identify suitable source and target datasets that could be used for the TL experiments. With regard to the source domain, it was necessary to identify a sufficiently large dataset on which both the models had already been trained. This would have led to common characteristics between the pre-trained models and therefore the possibility to make a fair comparison between the different NN architectures based on the results obtained during the

experiments. In this regard, as mentioned above, H36M dataset was chosen. As it is still the main benchmark for testing HMP algorithms, the majority of the state of the art DL models have been trained on it, making it perfect for obtaining pre-trained configurations. In addition, it still represents one of the largest and richest datasets available today in terms of the variability of the actions it contains.

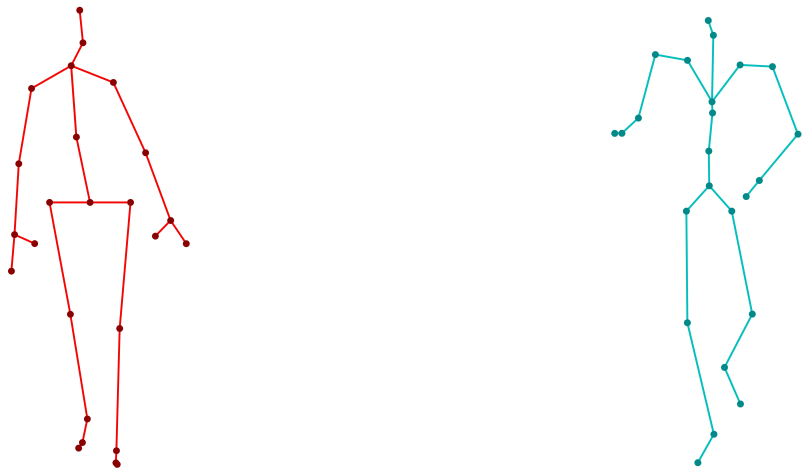
With regard to the target domain, it was important to identify a dataset that was sufficiently different from the source domain in order to test the knowledge transferability of the models during the experiments. Given the Collaborative Robotics filed in which this thesis is set, the ideal case would have been to use target datasets specific to that context. However, since the latter are not available at the time of writing, it was important to identify some that are close in terms of actions to the Collaborative Robotics context. Nevertheless, it was also interesting to study the transferability of the knowledge for this type of data from a more general perspective, therefore also considering target domains similar to the source domain, in order to compare the benefits of TL in different cases. As a result of these considerations, Achieve of Motion capture As Surface Shapes (AMASS) dataset was chosen as the target domain. It is not a single dataset, but a collection of smaller standardised datasets, each with its own set of actions. This makes it really suitable for the purposes of this thesis, as different datasets can be extracted as the target domain depending on the the specific goals of the experiments.

However, the two datasets have significant differences, mainly in the representation of the human skeleton, which are highlighted in the following paragraphs.

H36M dataset

H36M [15] contains recordings of 15 different actions performed by a total of 11 subjects recorded at 50 fps, but only 7 of them are commonly used for training and testing DL models. This dataset follows a very specific structure, in fact for each subject there are two separate recordings for each action, giving a total of 30 motion captures for each subject. The full list of actions is as follows: *walking*, *eating*, *smoking*, *discussing*, *giving directions*, *greeting*, *phoning*, *posing*, *purchasing*, *sitting*, *sitting down*, *taking photo*, *waiting*, *walking dog*, and *walking together*. A human body pose is represented by the angular rotation of a 32-link skeleton

plus the global rotation in the 3D space expressed in exponential map notation, giving a total of 99 parameters describing each pose. An example of the skeletal representation for a general walking pose is shown in Figure 5.1a.



(a) H36M skeleton

(b) AMASS skeleton

Figure 5.1: An example of the skeleton representation of the two datasets H36M and AMASS for a walking pose. The main differences between the two are in the shoulder area and on the upper part of the legs.

As can be seen from the visual representation of the skeleton, both hands and feet are represented by more than one keypoint, as the skeleton representation also considers a simplified scheme for their articulations. However, it is common to discard these and some other joints when predicting the human motion, thus simplifying the overall representation of the body skeleton with a total of 21 joints. The standard approach when using this dataset is to downsample the motion sequences at 25 fps and to use the actions of subjects 1,6,7,8, and 9 as the training set, subject 11 as the validation set and subject 5 as the test set.

AMASS DATASET

AMASS [24] is the largest dataset available at the time of writing and it consists of a collection of smaller datasets. For this reason there is no common frame rate, as each datasets has its own, ranging from 30 fps to 120 fps. Similarly, the set of actions depends on the specific dataset, so there is no predefined internal structure, which is instead a characteristic of H36M. However, one of the most interesting aspects of AMASS is that the skeletal representation of all

the datasets which it contains is standardised according to the Skinned Multi-Person Linear Model (SMPL) [22] body model. Each pose is specified by the rotation angles of 24 links with the additional global rotation in the 3D space in exponential map notation. An example of the AMASS skeleton representation is shown in Figure 5.1b.

Regarding the choice of the target AMASS sub-datasets to be used for the TL experiments, three were selected: ACCAD, GRAB, and DanceDB. The following paragraphs give a description of each of the three datasets and explain why they were chosen.

ACCAD

ACCAD [4] is a small dataset containing actions that are quite similar to the source domain represented by H36M. This dataset was chosen to assess the benefits of TL techniques when the source and target domains are quite similar, so that a comparison can be made with the results on a target dataset with a different distribution of actions compared to H36M.

GRAB

As the name suggests, GRAB [32] is a dataset containing actions of a subject grasping various objects from a static workstation, resulting in motion sequences with a static lower body. Of the datasets available in AMASS, this is probably the closest to a Collaborative Robotics context. For example, in industrial contexts, the operator may be assembling products while remaining stationary at a particular workstation.

DanceDB

DanceDB [6] is a dataset consisting of rather long motion sequences about dancing subjects. Although not directly related to the context of Collaborative Robotics, this dataset was chosen to study the effectiveness of TL in the case of target domains that are very different from the source domain, since H36M does not contain actions similar to the dancing ones.

In summary, H36M has been chosen as the source dataset on which to pre-train the DL models selected for the experiments, as it represents the most

common benchmark and all the latter are designed to deal with its data. On the other hand, three smaller datasets were selected from AMASS to represent the target domain in the TL experiments. As ACCAD was the smallest of the three, only part of GRAB and DanceDB were included in order to balance their size and to be able to make a fair analysis of the results obtained. Table 5.1 gives an overview of the main characteristics of the datasets that have been considered.

	H36M	ACCAD	GRAB	DanceDB
Total time [min]	175.87	26.75	24.06	24.29
Frames per second	50	120	120	120

Table 5.1: Summary of the datasets selected for the experiments, showing total duration and frames per second.

However, given the differences between all the selected datasets in terms of the number of fps and the two different skeletal representations of the human body, some preprocessing steps were required in order to standardise the data to be given as input to the DL models.

5.2 Preprocessing

Before proceeding with the experiments that will be described below, some preliminary steps are required in order to prepare the datasets and the models. In particular, they refer to a transformation from AMASS to H36M to match the two skeletal representations of the human body, a downsampling approach to uniform the fps of the different motion sequences, and a procedure to split the datasets into training, test, and validation sets. This Section describes these preprocessing steps and presents a graphical tool developed to visualise motion sequences.

5.2.1 AMASS to H36M transformation

The two datasets H36M and AMASS use a different skeletal representation, so giving a motion sequence from AMASS as input to a pre-trained model on H36M without changing the NN architecture would not only cause numerical problems, but also the actual poses would not make sense if interpreted with a different kinematic tree. As shown in Figure 5.1, the two skeletons are charac-

terised by strong differences, especially in the shoulder area and in the upper part of the legs. In addition, the hands and feet in AMASS are modelled in a simpler way than in H36M. Therefore, when taking any motion sequence from AMASS based on the SMPL body model, it is necessary to compute a projection to obtain the same actual poses from the H36M skeleton perspective before giving them as input to the models.

Mathematically, this procedure can be represented by a function $f : \mathbb{R}^{J_A \times D} \rightarrow \mathbb{R}^{J_H \times D}$, which maps individual poses from AMASS to their corresponding representation with the skeletal model of H36M; J_A and J_H represent the number of joints used in the AMASS and H36M skeleton, respectively. In other words, given a body pose $\boldsymbol{x}_A$ from AMASS, the goal is to compute $f(\boldsymbol{x}_A) = \boldsymbol{x}_H$ which can be interpreted with the H36M skeleton representation so that the two are visually equivalent, i.e. they correspond to the same pose but based on two different skeleton models. Since each pose is described by the rotation angles of each joint, the problem can be broken down into the individual joints as follows: given a joint $i, i \in 0, 1, ..., J_A$ from the AMASS skeleton and its corresponding joint $j, j \in 0, 1, ..., J_H$ from the H36M skeleton, the transformation can be expressed as:

$$\boldsymbol{e}_j^H = \boldsymbol{e}_i^A \times \boldsymbol{R}_i \tag{5.1}$$

where $\boldsymbol{e}_i^H$ and $\boldsymbol{e}_j^A$ are the rotation angles in rotation matrix notation for H36M and AMASS respectively, and $\boldsymbol{R}_i$ is the rotation matrix representing the transformation step. The problem can therefore be seen as computing $\boldsymbol{R}_i, i \in 1, 2, ..., j_A$, which are called *aligning rotation matrices* because they represent the rotation required to project one representation into the other. The pairs (i, j) of corresponding joints were obtained experimentally by visually comparing the two skeletons.

Computing aligning rotation matrices

Each body pose is described by the set of the joint rotation angles. However, another fundamental aspect of each body model is given by the offsets, which represent not only the length of each joint of the skeleton, but also its default orientation in 3D space when the corresponding rotation angle is zero; in practice, they are described by three coordinates representing a vector in the 3D space. Therefore, if the rotations of all the joints of both the H36M and AMASS skeletons are set to zero, the resulting poses are called *default pose* and

they do not match each other. Thus, based on the two default poses, the aligning rotation matrices $\boldsymbol{R}_i$ can be derived by rotating the joints of AMASS to match those of H36M. This could be done experimentally by manually trying to find the correct rotations, but it is better to compute them analytically to improve the overall accuracy of the transformation.

The offsets, and therefore the links, are relative, so each part of the body can be represented by a vector connecting the end of the preceding link to the beginning of the next. Therefore, given two paired links l_i^A and l_j^H from AMASS and H36M skeletons respectively, the aligning rotation matrix $\boldsymbol{R}_i$ corresponds to the rotation to be applied to the vector representing l_i^A to align it with the vector representing l_j^H; this can be computed using standard linear algebra techniques. However, since the offsets are relative, it is important to move along the kinematic tree of the body skeleton and consider the progressive rotations of previous links when computing each aligning rotation matrix.

Therefore, given two paired links l_i^A and l_j^H, the relative offset $\tilde{l}_i^A$, which account for the previous aligning rotations along the kinematic tree, can be computed as follows:

$$\tilde{l}_i^A = l_i^A \times \boldsymbol{R}_{chain}(i) \tag{5.2}$$

where $\boldsymbol{R}_{chain}(i)$ corresponds to the multiplication of all aligning rotation matrices for all links preceding l_i^A along the kinematic chain. Then the corresponding aligning rotation matrix $\boldsymbol{R}_i$ can be computed as the rotation matrix needed to align $\tilde{l}_i^A$ with l_j^H. Finally, assuming i' as the next link after l_i^A along the kinematic chain, $\boldsymbol{R}_{chain}(i')$ is updated as follows:

$$\boldsymbol{R}_{chain}(i') = \boldsymbol{R}_i \times \boldsymbol{R}_{chain}(i) \tag{5.3}$$

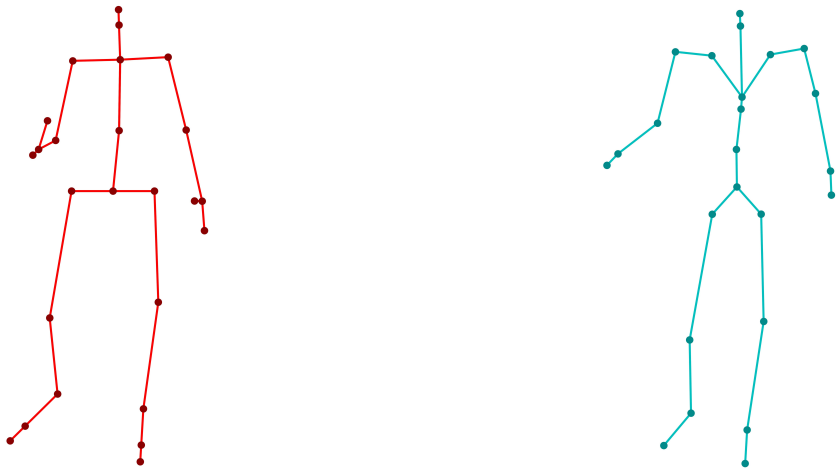
Therefore, a relative rotation for the i_{th} link of the AMASS representation can be transformed into the H36M representation as follows:

$$\boldsymbol{e}_j^H = \boldsymbol{e}_i^A \times \boldsymbol{R}_i \tag{5.4}$$

where the two link rotations $\boldsymbol{e}_j^H$ and $\boldsymbol{e}_i^A$ are expressed as rotation matrices.

Transforming poses and sequences

Once the aligning rotation matrices have been computed for each joint, a complete transformation of an AMASS body pose $\boldsymbol{x}_A$ into a H36M one $\boldsymbol{x}_H$ can be computed by iteratively applying Equation 5.4 to all the joints of the skeleton. Since the H36M skeleton representation consists of more joints than AMASS, for those having no correspondence, their rotation is set to zero angle. This is not a problem as these joints are all discarded when making predictions. Similarly, by applying a pose transformation to each frame of a given sequence, it is possible to transform a complete motion sequence. An example of a body pose transformation is shown in Figure 5.2.



(a) H36M skeleton (b) AMASS skeleton

Figure 5.2: An example of body pose transformation from AMASS to H36M skeletal representation. The two poses are almost equivalent except for some parts, mainly around the shoulders and the upper part of the legs.

It is important to note that the description given above concerns the general procedure for transforming each link rotation, whereas for some specific links it was necessary to apply some additional minor steps which are not described for the sake of simplicity. Moreover, as can be seen in Figure 5.2, this transformation leads to very similar but not exactly equal poses. Some differences between the two, such as the length of the links, are mainly due to the position of the keypoints in the body skeleton. One could try to mitigate these discrepancies by manually adjusting some offsets or rotation angles, however they do not really pose a problem. In fact, a visual assessment of the transformation of complete motion sequences shows that they are characterised by natural body

movements that are, on the whole, very similar to the original. Moreover, since these sequences are fed to the DL models during the experiments, the discrepancies will be consistent in both the seed and the target sequence, thus cancelling out the differences.

5.2.2 Downsampling motion sequences

Both the PVRED and HRI models work with 25 fps motion sequences, so it was necessary to unify all the different datasets to this frame rate. For the H36M motion sequences, originally at 50 fps, a simple decimation approach based on skipping one frame every two was used. However, this is only possible if the source sequence fps is a multiple of the target sequence fps. In the case of AMASS this was not possible because many motion sequences are at 30 or 60 fps. To downsample such motion sequences, each joint rotation was treated as a discrete signal over time. A polynomial-based interpolation method was then used to obtain an approximate continuous signal, which was then sampled at the frequency of interest to obtain the desired fps for the downsampled motion sequence. The motion sequences were also filtered to remove high-frequency movements that are not characteristic of the human body. When visually comparing the original and downsampled sequences, the differences are very minimal and barely noticeable. Using this interpolation approach, all motion sequences were downsampled to 25 fps.

5.2.3 Splitting the datasets

After transforming and downsampling to 25 fps the three target datasets from AMASS to H36M skeletal representation, it was necessary to split them into training, test, and validation sets before proceeding with the experiments. Unfortunately, in the case of AMASS there is no standard splitting procedure based on subject ids as there is for the H36M dataset. A simple approach could be to randomly assign each motion sequence in the dataset to training, test, or validation set accordingly to a probability distribution. However, given the limited size of the target datasets, and the fact that human motion is a data type that is strongly biased by the type of action, this approach would result in very unbalanced splits. There is a high probability that some type of actions would only appear in the test dataset, thus greatly reducing the generalisation performance of the DL models. Ideally, each motion sequence should have a

label describing its action type on which to base a balanced split; however, this type of information is not available.

A clustering-based approach was used to synthetically generate such labels. First, instead of splitting the full motion sequences directly, they were broken down into separate windows of fixed length, which were collected into a set S, a procedure that would be done later anyway. This is because the different sequences in the dataset vary greatly in their length and therefore do not contain the same amount of information. The aim was then to try to assign a label to each window so that those with the same value were characterised by the same type of action. To do this, a *k-means* clustering algorithm was used, based on embeddings $\boldsymbol{e}_s \in \mathbb{R}^d$ for each window $s \in S$. The embeddings were created by concatenating the vector representation of all the poses in the window and then reducing their size to d using the Principal Component Analysis (PCA) technique. After running the clustering algorithm with K clusters, the labels were used to create balanced splits of the target datasets.

The size of the embeddings d and the number of clusters K were set to 200 and 15 respectively, based on experimental tests.

5.2.4 Skeleton visualisation

The graphical visualisation of body poses or whole motion sequences can be useful for a qualitative evaluation of the predicted sequences and also as a general debugging tool. Therefore, a human pose visualisation tool has been developed based on 3D plots where each link of the human body pose is represented as a segment in the space connecting the two extreme keypoints. The 3D coordinates of the keypoints are computed by applying forward kinematics along the kinematic tree representing the human skeleton, based on the specific rotation angles of the links of the given pose. Instead, in order to visualise entire motion sequences, the plot is updated with the pose corresponding to the current frame at the correct frequency. The visualiser implements several features, including the ability to overlap multiple skeletons to compare two or more motion sequences, which can be extremely useful when evaluating the quality of the predicted sequence. This makes it possible to clearly visualise the differences with respect to the ground truth and possibly identify the main weaknesses of the prediction algorithm. The images of the human skeleton shown in the different Chapters have been rendered with the developed visualiser.

5.3 Experiments

The aim of this thesis is to study the problem of HMP in the context of Collaborative Robotics and to investigate the effectiveness of the technique of Deep Transfer Learning (DTL) to improve the performance of the models when limited datasets are available. Therefore, the experiments carried out were aimed at answering the questions posed by this thesis. The first set of experiments involved a comparison between a model initialised from scratch and the fine-tuning technique on different partitions of the target datasets with different sizes. The aim was to understand if this TL approach can improve the model performance and how they change as the number of data varies. Subsequently, further experiments were conducted to test different TL techniques to understand if they can bring improvements compared to a more commonly used fine-tuning approach. This Section presents an overview of the different partitions of the target datasets used and a complete list of all the experiments conducted with their motivations.

5.3.1 Progressive Dataset Partitioning

In the case of practical applications, it would be useful to study how much the benefits of TL approaches vary depending on the size of the datasets, in order to understand the amount of data that needs to be collected to achieve good performance for the HMP task. Therefore, each of the three selected target datasets, i.e. ACCAD, GRAB, and DanceDB, was partitioned based on the following approach: the test and validation sets are fixed for all partitions and correspond to 10% of the dataset, while the training set ranges from 10% to 80% with progressive increases of 10%. For example, the smaller ACCAD partition will have training, test, and validation sets corresponding to 10%, 10%, and 10% of its total size, with the 70% discarded. On the other hand, the larger partition will have the same test and validation sets, but the training set will correspond to the 80% of the dataset; in this case, the entire dataset is used. Using different partitions makes it possible to understand how results change when different amounts of data are available for training, but the same set is used to evaluate the models. Table 5.2 shows the number of windows contained in each of the different dataset partitions; the splitting procedure was the one described in 5.2.3.

The number of windows between the different target datasets is quite similar,

Number of windows extracted for each target dataset partitions									
Partition	ACCAD			GRAB			DanceDB		
	Train	Test	Valid	Train	Test	Valid	Train	Test	Valid
10,10,10	21	21	21	24	24	24	22	22	22
20,10,10	42	21	21	48	24	24	44	22	22
30,10,10	63	21	21	72	24	24	66	22	22
40,10,10	84	21	21	96	24	24	88	22	22
50,10,10	105	21	21	120	24	24	110	22	22
60,10,10	126	21	21	144	24	24	132	22	22
70,10,10	147	21	21	168	24	24	154	22	22
80,10,10	168	21	21	192	24	24	176	22	22

Table 5.2: Number of motion sequence windows extracted for each partition of the three target datasets. Partition x,y,z represents the percentage size of the training, test, and validation sets, respectively.

which allows for balanced experiments. If this were not the case, the results could be strongly influenced by the size of the partitions, making a comparison between them unreliable.

5.3.2 Transfer Learning by fine-tuning

The first set of experiments conducted aimed to compare the performance of a DL model initialised from scratch with a fine-tuning approach on the different partitions of the target datasets. Specifically, in both cases the NN architecture is the same, but for the scratch model its parameters are randomly initialised, while in the case of a fine-tuned model they are taken from a checkpoint pre-trained on the source dataset. In the specific case of these experiments, when referring to the fine-tuning approach, this means that all model parameters are updated during training. Symmetric experiments were performed for both DL models described in Section 5.1, and in particular two different NN architectures for PVRED with one and two RNN layers respectively were considered. This was done to compare the effectiveness of the fine-tuning approach as the model architecture becomes more complex.

The hyperparameters used for the different models, such as the size of the NN layers or the optimiser, are those specified by the authors in the respective papers. This was done to avoid the risk that changes in the NN architectures or other parameter changes would strongly influence the results of the experiments,

rather than being a consequence of the TL technique itself. Furthermore, if these techniques were to be deployed in practical contexts, the pre-trained model would probably be left as it is, or only a few minor changes would be made. However, the learning rate was reduced by a factor of 10 only for the fine-tuned model, which, as described in Chapter 3, helps to reduce the risk of catastrophic forgetting. With regard to the number of epochs, some preliminary tests were carried out with a very high value for this parameter in order to understand when the models start to overfit on average and therefore to stop training a few epochs ahead. Table 5.3 shows the number of epochs and the learning rate used for training from scratch or for fine-tuning the different DL models.

Different configurations of some model parameters						
Parameter	PVRED				HRI	
	1 RNN layer		2 RNN layers			
	scratch	tuned	scratch	tuned	scratch	tuned
Learning rate	0.0001	0.00001	0.0001	0.00001	0.0005	0.00005
Epochs	1000	1000	1000	1000	100	100

Table 5.3: Different configurations of the parameters of PVRED with one and two RNN layers and HRI models.

The same model configurations were used to run the experiments on all partitions of the target datasets. During training, the models were evaluated using the metrics described in Section 4.1 and the results were stored for further analysis. These initial experiments have already made it possible to identify common features between the different experiments performed on the different partitions, and thus to draw some preliminary conclusions about the benefits of the fine-tuning technique.

5.3.3 Other Transfer Learning techniques

The second set of experiments focused on applying different TL techniques to understand if the results of the fine-tuning approach can be improved. These techniques can be divided into two main classes:

- Freezing the entire pre-trained network and replacing only the last layer.
- Freezing the entire pre-trained network and stacking fresh-new layers on top of it.

Both classes of approaches aim to reduce the problem of catastrophic forgetting by using the features learned by the pre-trained model. In particular, the first class is generally less powerful than the second, which increases its potential to process and learn about data by adding new layers. These experiments focused only on the ACCAD dataset, which from the fine-tuning experiments appeared to be the dataset where TL is more beneficial. Moreover, only its 30% training partition was considered as it is neither too small nor too large, and according to the results of previous experiments it is representative of the different partitions.

Regarding PVRED, only the 1 RNN layer configuration was used as the fine-tuning approach performed best on it. The experiments consisted of 1) freezing the entire NN and replacing the last linear layer, which is the only one trained, and 2) freezing the entire NN and stacking a fresh new RNN layer, which is trained together with the last linear layer. Similar experiments were carried out for HRI. They consisted of 1) freezing the whole NN architecture and training only the last prediction layer, and 2) freezing the entire NN architecture and stacking several GCN layers trained with the last prediction layer. In particular, two different setups with two and four additional GCN layers were tested. The number of epochs and the learning rate were the same as those used in the fine-tuning experiments.

5.4 Resources and development setup

Training DL models can be costly in terms of time and resources. To mitigate this, we conducted heavier experiments on a computing cluster while minor processes on a desktop computer. The specifications for each device are presented in Table 5.4.

	Desktop Computer	Computing Cluster
GPU	NVIDIA GeForce RTX 2080 Ti	3x NVIDIA Tesla V100 S
GPU memory	11 GB	3x 32 GB
CPU	Intel Core i9-9900K	Intel Xeon CPU E5-2609 V3
RAM	32 GB	64 GB
Disk	250 GB	30 TB

Table 5.4: Specifications of Desktop Computer and Computing Cluster.

DL models were developed in Python 3.6.10, based on the PyTorch library.

6

Results and discussion

This Chapter presents and discusses the main results of the experiments described in Section 5.3, which aimed to investigate whether Deep Transfer Learning (DTL) approaches can be beneficial for the task of Human Motion Prediction (HMP) when limited amount of data is available. The first part of the experiments is related to the fine-tuning approach, while the second part applies other Transfer Learning (TL) techniques based on freezing part of the Neural Network (NN) and adding fresh new layers. The setup of the experiments, as well as the settings of the different model hyperparameters, refer to what is described in Chapter 5. The results show the pattern of the validation error along the training epochs and the computation of the TL metrics based on the latter. Furthermore, the best configurations of the different models are evaluated on the test set by computing the average error along the 25 prediction frames.

6.1 Fine-tuning experiments

This Section presents the results of the comparison between the scratch and fine-tuned models. In both cases the NN architecture is the same, but the initialisation of the parameters is different: in the scratch model they are randomly initialised, while in the fine-tuned model they are taken from the pre-trained checkpoint. During training, all parameters are updated, with a 10 times lower learning rate in the case of fine-tuning. The experiments were carried out on the two Position-Velocity Recurrent Encoder-Decoder (PVRED) configurations,

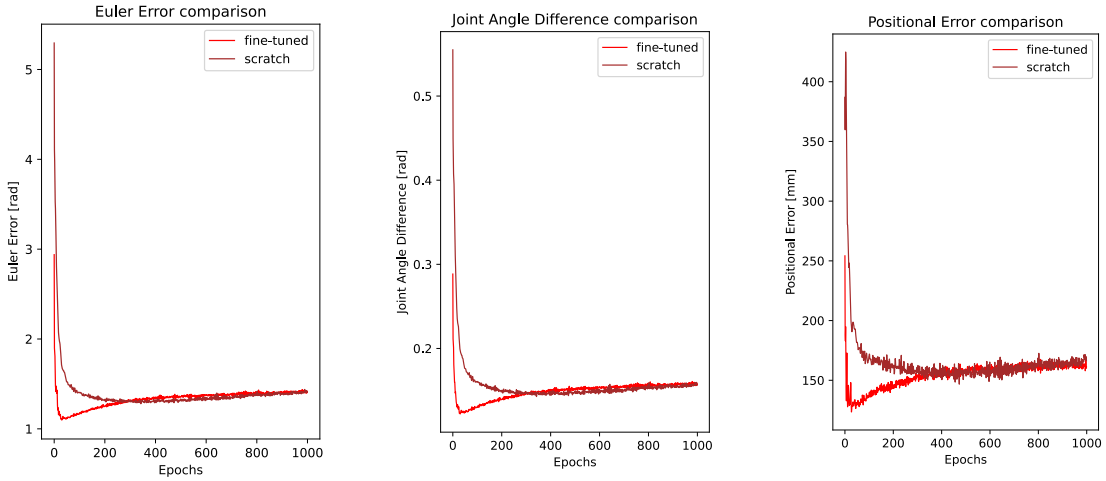
with one and two Recurrent Neural Network (RNN) layers, and on History Repeat Itself (HRI), with progressively larger training partitions.

6.1.1 PVRED with one RNN layer

The first experiments involved the simpler NN architecture, i.e. PVRED with a RNN layer, on the smallest ACCAD partition for which all HMP metrics are shown. Given the similar pattern for the latter, only the Euler Error is shown in the results for the following experiments.

Preliminary experiments on ACCAD

Both the scratch and fine-tuned models based on the PVRED architecture with one RNN layer were trained for 1000 epochs. The results of all HMP metrics computed on the validation set are shown in Figure 6.1.



(a) Euler Error Comparison

(b) Joint Angle Difference Comparison

(c) Positional Error Comparison

Figure 6.1: Comparison of HMP metrics between a model initialised from scratch and a fine-tuned model using the ACCAD dataset (10% training set partition).

As can be seen from the plots, the fine-tuned model performs better than the scratch model: not only is the minimum error lower, but the convergence is also much faster in terms of epochs. In addition, there is a strong improvement in the initial performance of the model after the very first epoch, implying that the pre-trained model carries knowledge from the source domain.

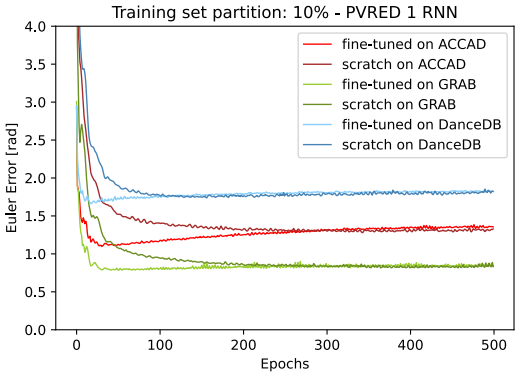
Since the error trend is very similar for all three HMP metrics, only the Euler Error is shown in the following results, as this is also the most common metric used for evaluation purposes at the state of the art. This helps to keep the notation simple. If there are cases where significant differences between the diverse metrics are present, these will be reported and detailed for each specific experiment. Furthermore, to help reading the graphs shown in this Chapter, some simplifications were adopted based on two observations. First, the gap between the initial and the minimum error is quite large, resulting in a very stretched y axis. Second, the error pattern is very flat after the first half of the training epochs. Since the most interesting part of comparing different models is the convergence region, in the following plots both the x and y axes are clipped at $[0, 500]$ and $[0, 4]$ respectively. In very few cases the convergence of the scratch model (which has a more non-increasing error pattern) occurs beyond 500 epochs. However, this behaviour and the initial error gap are captured by the TL metrics.

These first results are very promising as the fine-tuning approach clearly improves the performance of the model significantly. However, this first partition tested is relatively small, so it will be interesting to understand what happens when the size of the training sets increases.

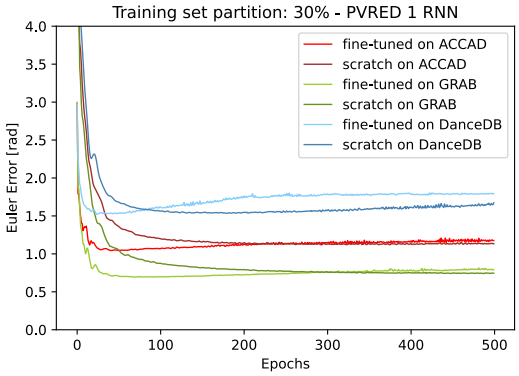
Experiments on all target datasets partitions

As mentioned above, the subsequent experiments included all partitions of the three target datasets to assess how the results differ with different training set sizes and action distributions. As a common pattern emerged as the amount of data increased (which will be discussed later), in order to keep the notation more compact, only the results for the 10%, 30%, 50%, and 80% training partitions are shown in Figure 6.2.

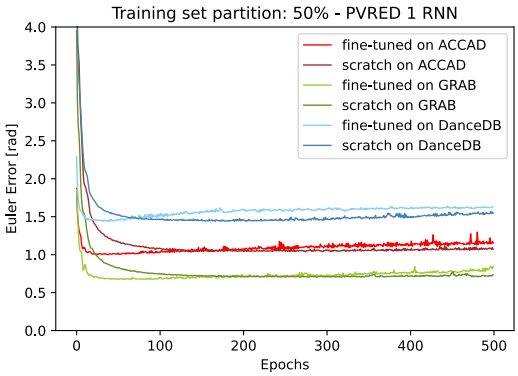
In general the fine-tuned approach improves the performance of the model compared to scratch training. As mentioned above, the main improvements are in terms of an overall lower minimum error plus fewer training epochs are needed to reach this point. However, some interesting observations can be made. When comparing the three target datasets, the one where the fine-tuning approach works best is ACCAD, since the gap between the minimum error with the scratch model is the largest and the epochs required to converge are comparable to the other datasets. This is probably because ACCAD is the



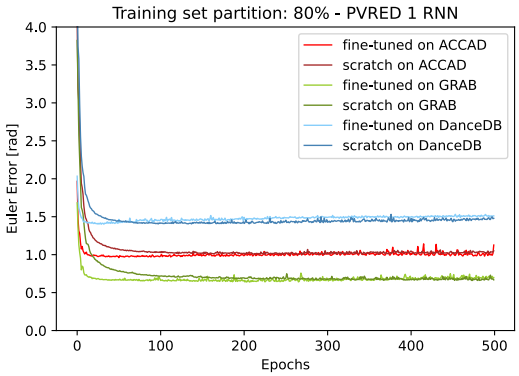
(a) Training set partition 10%



(b) Training set partition 30%



(c) Training set partition 50%



(d) Training set partition 80%

Figure 6.2: The validation Euler Error during the training epochs of PVRED with one RNN layer is shown in the graphs, which illustrate the comparison between a model trained from scratch and a fine-tuned model, using different partitions of the target datasets.

most similar dataset to the source domain of the three, so the transferability of pre-trained knowledge is more beneficial. On DanceDB there is a similar behaviour to ACCAD, although less pronounced, while on GRAB the minimum errors are quite similar, but the convergence of the fine-tuned model is much faster. Considering that GRAB includes actions with the lower part of the body at rest, the overall error is lower, which is probably why the improvements of the fine-tuned model are less noticeable.

Regarding the different sizes of the partitions, the fine-tuned model seems to give the highest benefits compared to the scratch model when the training partition is the smallest. This is an expected behaviour, probably because the randomly initialised model does not have enough information to properly tune its parameter when trained on small partitions. On the other hand, the fine-

tuned model can use the knowledge from the source domain and improve its predictions. With 50% and 80% training partitions, the minimum error is almost the same, but there is still a huge improvement in the number of epochs needed to converge, i.e. the fine-tuned model is much faster.

Some additional considerations can be made based on the total Euler Error between the three source datasets. GRAB is the one with the lowest error, and this is probably due to its actions: the lower part of the body is stationary, which makes it easier to predict and therefore the overall error decreases. On the other hand, DanceDB is the target dataset with the highest error, which can also be explained by the nature of its actions: dancing very often involves very wide body movements, which are extremely unpredictable unless you already know the dance. This makes its predictions very difficult, considering also that there are no similar movements in the source domain, i.e. Human 3.6M (H36M). The results for the other partitions of the datasets not shown, i.e. 20%, 40%, 60%, and 70%, reflect the trend shown in Figure 6.2: the lower the partition, the higher the improvements given by the TL approach.

However, further analysis can be done by quantitatively evaluating the results shown above by computing the TL metrics, which are reported in Table 6.1.

TL Metrics for PVRED with 1 RNN Layer						
TL Metric	Training set partition 10%			Training set partition 30%		
	ACCAD	GRAB	DanceDB	ACCAD	GRAB	DanceDB
JSP	2.356	2.573	4.047	3.059	3.134	3.970
MEP	0.182	0.030	0.076	0.074	0.046	0.014
TTT	329	337	147	278	496	156
TL Metric	Training set partition 50%			Training set partition 80%		
	ACCAD	GRAB	DanceDB	ACCAD	GRAB	DanceDB
JSP	4.049	2.094	2.711	2.101	2.139	2.488
MEP	0.039	0.029	0.002	0.036	0.012	0.003
TTT	262	320	157	964	419	72

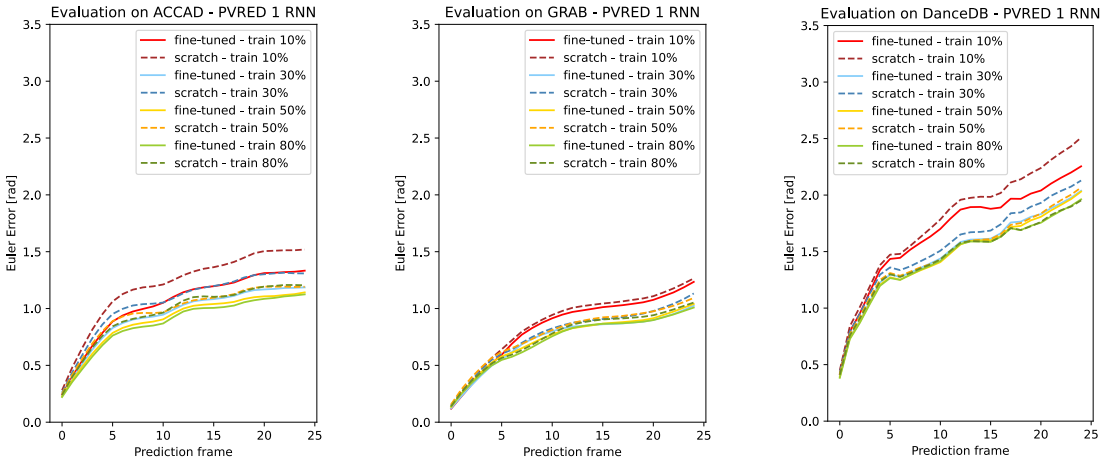
Table 6.1: Transfer Learning metrics computed for different training set partitions, comparing the performance of a model trained from scratch with a fine-tuned model based on PVRED with one RNN architecture. Positive values indicate better performance for the fine-tuned model.

These metrics allow to further analyse and compare the performance of the fine-tuned and scratch models, especially for some aspects that are not visible from the Euler Error plots in 6.2. First, the Jump-Start Performance (JSP) highlights the gap between the two models after the very first training epoch where, as can be seen from the Table, it is always positive, meaning that the Euler Error of the pre-trained model is lower. The gap is quite large and there is not really a clear pattern along the increasing size of the training partitions. This is probably because the initial performance of the scratch model depends on the initialisation of its weights, which is random.

The Minimum Error Performance (MEP) and Time To Threshold (TTT) reflect the considerations made above when looking at the Euler Error plots: the fine-tuned model reaches a lower minimum error in much fewer training epochs, which is more pronounced in the case of ACCAD and for smaller partitions of the training set. Some strange values for the TTT can be observed in the case of ACCAD and GRAB 80% partition. This is because, as mentioned above, the scratch model tends to have a non-increasing Euler Error pattern, especially for larger training sets, which makes it reach the minimum error much further. On the other hand, the fine-tuned model reaches the threshold much earlier, resulting in a very large difference in the number of epochs. This does not happen for smaller partitions, as the scratch model tends to overfit the data in the first half of the training epochs.

Nevertheless, in general the improvements are always significant. Also comparing this metric with the Euler Error curves, it is clear that the TL approach significantly improves the results when only a few training epochs are available. In the specific case of these experiments, the training time was not too high due to the small size of the datasets. However, when considering scaling up the models to contexts where both the source and target domains are much larger, a significant reduction in the number of epochs may be extremely important to make the training procedure feasible. Another interesting comparison can be made by evaluating the performance of the different models on the test set by computing the average Euler Error along the 25 prediction frames. Following standard Machine Learning (ML) approaches, the training checkpoint chosen for each model corresponds to the one with the lowest validation Euler Error. The results are shown in Figure 6.3.

The evaluation results on the test set are consistent with the above considerations based on the validation error. The fine-tuned model always performs



(a) Test on ACCAD (b) Test on GRAB (c) Test on DanceDB

Figure 6.3: Evaluation of the 25 prediction frames for the different scratch and fine-tuned models based on PVRED with one RNN layer architecture.

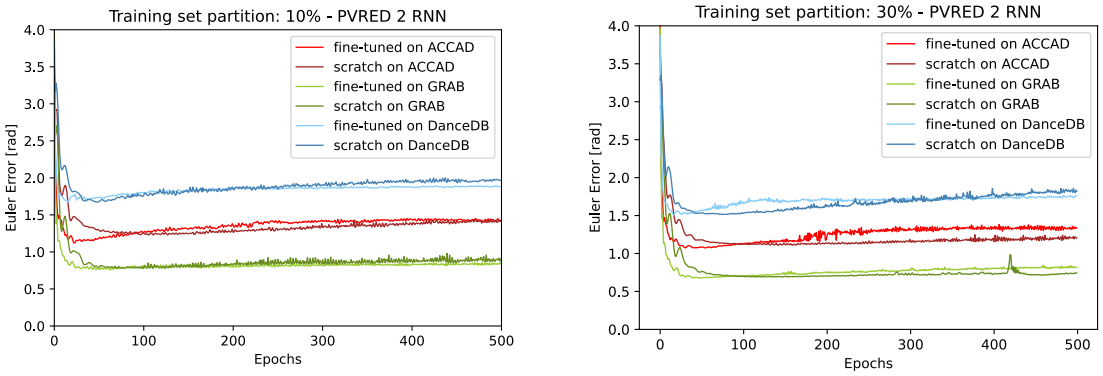
better than the scratch model, especially for small dataset partitions and in the case of ACCAD. Some additional considerations can be made to reason about the amount of data required to train those models. Based on ACCAD, the fine-tuned model on the 30% partition performs even better than the scratch model trained on the 80% partition. This result is very significant as it suggests that if the pre-trained model brings enough information from the source domain, not only the training epochs and the minimum error are much better, but also the size of the dataset required is much smaller. This can be very beneficial in practical applications of Collaborative Robotics, where, if a HMP model already has sufficient pre-trained information, only a small number of samples are needed to fine-tune the model effectively. Unfortunately, the same behaviour is not that stronger for the other two datasets, i.e. GRAB and DanceDB, suggesting that the pre-trained knowledge is probably not rich enough to lead to good generalisation performances for datasets that are quite different from the source domain.

Overall, the error has an exponential increasing trend along the 25 prediction frames, which is caused by the accumulation of error over time.

6.1.2 PVRED with two RNN layers

The second set of experiments replicates the same performed in 6.1.1 but using the PVRED architecture with two RNN layers. The aim is to understand

how the results change when using the same type of NN with two different levels of complexity. Increasing the number of parameters generally improves the capabilities of the NN, but it also requires much more data to optimise it properly. As before, the results are based on the Euler Error metric, given the similarity of the three HMP metrics on the pattern. The results of the comparison between the scratch and fine-tuned models based on the validation Euler Error are shown in Figure 6.4.



(a) Training set partition 10%

(b) Training set partition 30%

(c) Training set partition 50%

(d) Training set partition 80%

Figure 6.4: The validation Euler Error during the training epochs of PVRED with two RNN is shown in the graphs, which illustrate the comparison between a model trained from scratch and a fine-tuned model, using different partitions of the target datasets.

Again, it is clear that the fine-tuned model still generally performs better than the scratch model. However, the advantages of the TL approach are significantly less pronounced compared to those performed with PVRED with one RNN layer. In particular, a first visual comparison shows that the gap between the minimum Euler Error of the two models is significantly reduced and that the scratch model

now converges much faster. A more detailed comparison between the two fine-tuned models and the two scratch models in the two configurations shows that the error curves of the first two are almost equivalent, showing that there is definitely no improvement. On the other hand, the scratch model with two RNN layers shows a strong improvement with respect to the NN architecture with one RNN layer.

Overall, similar considerations as before can be drawn: the target dataset where the TL approach brings the greatest benefits is ACCAD, as the size of the training partitions increases, the performance of the fine-tuned and scratch models are almost equivalent. To better understand the differences in the minimum Euler Error and the number of epochs needed to converge, the TL metrics are shown in Table 6.2.

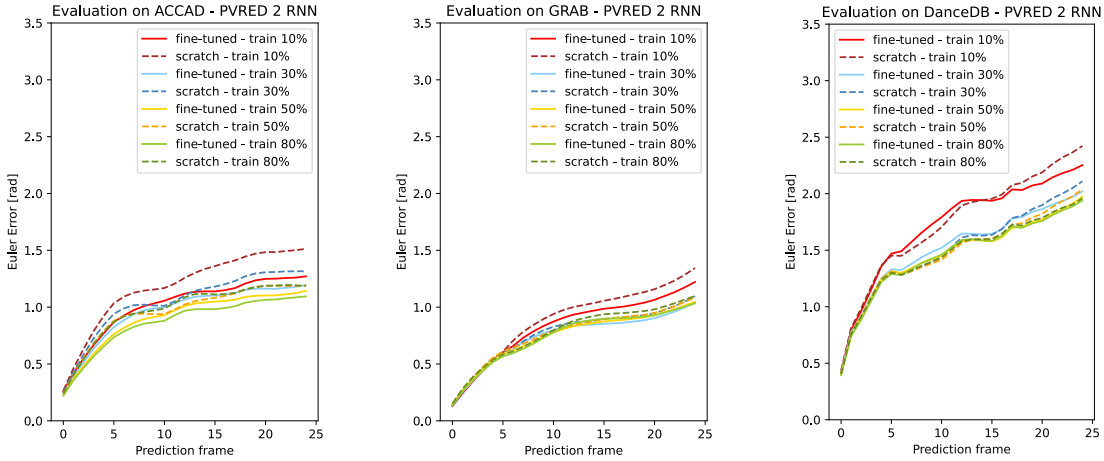
TL Metrics for PVRED with 2 RNN Layer						
TL Metric	Training set partition 10%			Training set partition 30%		
	ACCAD	GRAB	DanceDB	ACCAD	GRAB	DanceDB
JSP	-0.437	-0.643	-0.334	-0.288	-1.042	-0.579
MEP	0.108	0.019	-0.007	0.033	0.012	0.008
TTT	99	69	/	118	116	58
TL Metric	Training set partition 50%			Training set partition 80%		
	ACCAD	GRAB	DanceDB	ACCAD	GRAB	DanceDB
JSP	0.689	0.512	1.055	1.219	1.29	1.185
MEP	0.058	0.013	-0.009	0.043	0.008	0.013
TTT	105	108	/	685	567	60

Table 6.2: Transfer Learning metrics computed for different training set partitions, comparing the performance of a model trained from scratch with a fine-tuned model based on PVRED with two RNN architectures. Positive values indicate better performance for the fine-tuned model. Empty values for the TTT metrics mean that the fine-tuned model never reaches the minimum error of the scratch model.

The computation of the TL metrics clearly shows the differences compared to the NN architecture with only one RNN layer. For the ACCAD and GRAB datasets, the behaviour is quite similar, although for the first the benefits of fine-tuning are significant. This is particularly noticeable in the TTT metric, as the scratch model is now much faster to converge. For the DanceDB dataset, on the other hand, the performances of the fine-tuned and scratch models are now

much closer, even for small training partitions. In particular, the scratch model is even able to achieve a lower minimum Euler Error than the fine-tuned model, albeit only slightly.

In terms of the JSP metric, this is negative for the 10% and 30% partitions, meaning that the performance of the fine-tuned model is worse than that of the scratch model after the very first epoch. This may indicate that the knowledge of the pre-trained model is not broad enough to generalise well to more complex models. The scratch model, which has a higher learning rate, is able to tune its parameters more quickly. Figure 6.5 shows the results of the evaluation on the test set.



(a) Test on ACCAD (b) Test on GRAB (c) Test on DanceDB

Figure 6.5: Evaluation of the 25 prediction frames for the different scratch and fine-tuned models based on PVRED with two RNN layers architecture.

In the case of the evaluation on the test set, the results are much more comparable between the two PVRED architectures with one and two RNN layers. In fact, comparing the computation of the TL metrics, the main differences are in the TTT and JSP, which are not reflected in the evaluation, since the selection of the model checkpoints is based only on the MEP. The latter shows the main differences in the DanceDB dataset, in fact comparing Figure 6.3c and 6.5c it is more evident that the scratch models reaches better performance in the case of two RNN layers configuration. On the other hand, in the case of ACCAD and GRAB, the evaluation curves are almost equivalent between the two NN architectures.

In the case of ACCAD, the fine-tuned model trained on the 30% partition again performs slightly better than the scratch model trained on the 80% parti-

tion. However, this does not hold for the other two datasets, suggesting that the TL approach works best when the source and target domains are similar.

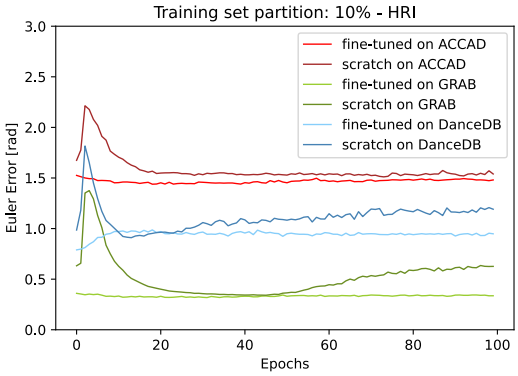
After these first experiments on PVRED, some conclusions can be drawn. In general, the fine-tuning approach performs better than training a model from scratch, and this is much more evident in the case of the simpler architecture, i.e. with one RNN layer. This is probably because the source dataset, i.e. H36M, and the type of actions it consists of are not rich enough to provide sufficient pre-trained information to achieve a good grade of generalisation and so better performance in more complex NN. Furthermore, as pointed out in the review of DTL in Chapter 3, the fine-tuning approach works better when the source and target domains are similar. In the specific case of these experiments, this is represented by the ACCAD dataset. This suggests that the knowledge acquired by the pre-trained model is strongly related to the specific type of actions it sees during training. Therefore, it is necessary to have large and very rich source datasets in order to be able to successfully apply the TL technique in different target domains for the task of HMP.

6.1.3 HRI

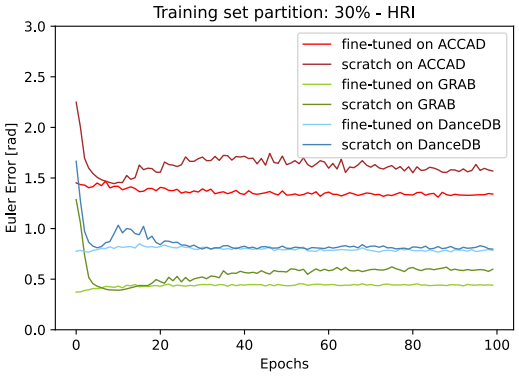
In order to understand if fine-tuning is an approach that can be successfully applied to different Deep Learning (DL) models for the task of HMP, other experiments were performed as for PVRED in the case of the HRI model. Similarly, they consist of a comparison between a fine-tuned model and a scratch model on the different partitions of the three target datasets. As the error pattern is again similar for all three HMP metrics, the results are based on the Euler Error metric only. The different models were trained for 100 epochs and the validation error results are shown in Figure 6.6.

From these first results it is clear that HRI has a completely different behaviour compared to PVRED. Both the fine-tuned and scratch models start with a much lower Euler Error, and the improvement over the training epochs is much smoother. Overall, the minimum error of the fine-tuned model tends to be lower compared to the scratch model, especially in the case of the ACCAD dataset. On the other hand, the scratch model seems to converge faster than the fine-tuned model, which has a smoother error pattern.

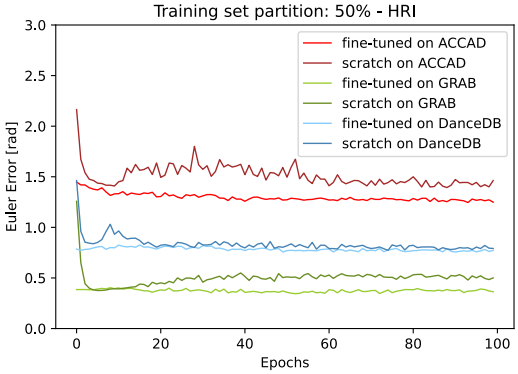
In addition, a strange behaviour can be observed in the case of the 10% partition, where the scratch model has a strong initial peak, while the fine-



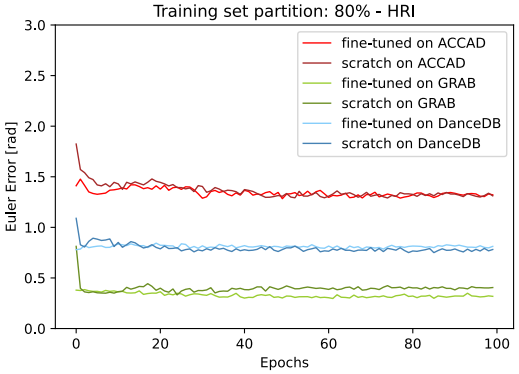
(a) Training set partition 10%



(b) Training set partition 30%



(c) Training set partition 50%



(d) Training set partition 80%

Figure 6.6: The validation Euler Error during the training epochs of HRI is shown in the plots, which illustrate the comparison between a model trained from scratch and a fine-tuned model, using different partitions of the target datasets.

tuned model immediately starts to deteriorate. This suggests that the amount of data available in the training set is not large enough for this type of NN architecture, which is more complex compared to PVRED and consists of many more parameters. In addition, the fine-tuned model is generally more difficult to improve, and this can be caused by a strong bias from the pre-trained model. In particular, the motion attention mechanism is likely to be strongly biased by the actions the model has seen during training on the source dataset. However, similarly to PVRED, increasing the partition size makes the performance of the fine-tuned and scratch models very similar. Table 6.3 shows the results of the computation of the TL metrics.

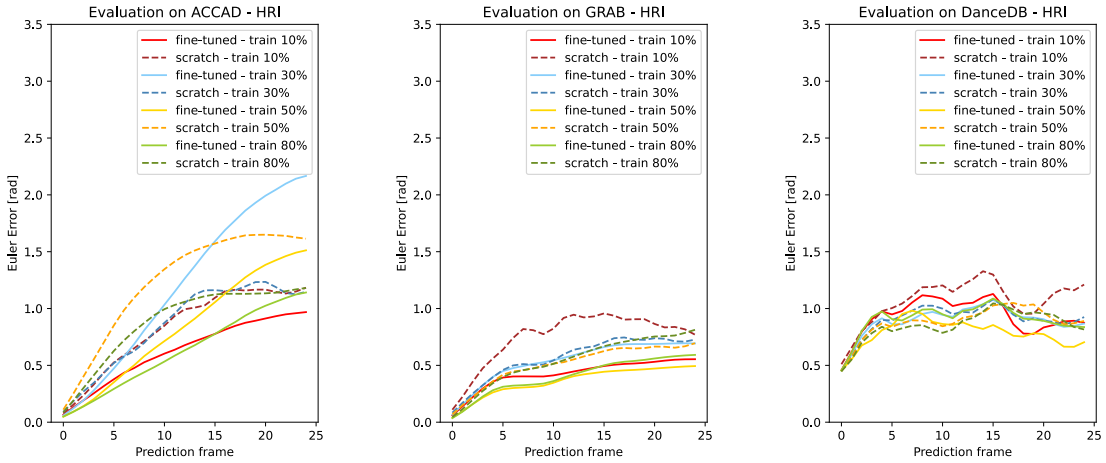
In general, the fine-tuning approach achieves a lower minimum error and converges faster compared to the scratch model. In the case of the 50% partition

TL Metrics for HRI						
TL Metric	Training set partition 10%			Training set partition 30%		
	ACCAD	GRAB	DanceDB	ACCAD	GRAB	DanceDB
JSP	0.150	0.272	0.196	0.797	0.914	0.888
MEP	0.072	0.023	0.12	0.134	0.018	0.028
TTT	71	38	13	8	10	55
TL Metric	Training set partition 50%			Training set partition 80%		
	ACCAD	GRAB	DanceDB	ACCAD	GRAB	DanceDB
JSP	0.716	0.872	0.678	0.412	0.433	0.310
MEP	0.148	0.031	0.017	0.007	0.036	-0.029
TTT	85	-11	73	44	3	/

Table 6.3: Transfer Learning metrics computed for different training set partitions, comparing the performance of a model trained from scratch with a fine-tuned model based on the HRI architecture. Positive values indicate better performance for the fine-tuned model. Empty values for the TTT metrics indicate that the fine-tuned model never reaches the minimum error of the scratch model.

on GRAB, the scratch model converges 11 epochs earlier, while in the case of the 80% partition on DanceDB, the latter reaches a lower error. However, there is no clear behaviour along the different partitions. In the tests performed on PVRED, it was clearly noticeable that the MEP and TTT decreased as the size of the training set increased. In the case of HRI, however, this is only noticeable for ACCAD, while there seems to be no correlation for the other datasets. This could be due to the NN architecture itself, which is much more complex for HRI and probably requires a larger amount of data to be trained properly. Figure 6.7 shows the results of the evaluation of the different models on the target test sets.

In the case of ACCAD and GRAB, the fine-tuned models generally performs better than the scratch model. Moreover, in some cases, especially for ACCAD, the fine-tuned models trained on smaller partitions perform significantly better than the scratch model trained on much larger partitions. However, these results must be taken with caution, as some inconsistent results can be identified with respect to the validation Euler Error. For example, looking at the validation error for the 30% partition on ACCAD in Figure 6.6b, it is evident that the fine-tuned model achieves a minimum error that is significantly lower than the scratch model. However, when compared with the evaluation on the test set in Figure



(a) Test on ACCAD (b) Test on GRAB (c) Test on DanceDB

Figure 6.7: Evaluation of the 25 prediction frames for the different scratch and fine-tuned models based on HRI.

6.7a, the fine-tuned model performs much worse, especially for the second part of the predicted sequence. Another example is that the fine-tuned model on the 80% partition performs worse than the fine-tuned model on the 50% partition for both GRAB and DanceDB, which can be seen in Figure 6.7b and 6.7c respectively. These inconsistencies and strange results in the comparison of performance on the validation and test sets may be a consequence of both the NN architecture itself and the limited amount of data in the target datasets. In fact, HRI relies on the attention mechanism to make predictions, attempting to repeat common motion patterns that it has already seen in the source dataset during training. This makes the model strongly biased towards the type of actions in the training set. In the specific case of this experimental setup, even if the target datasets were split using a clustering-based procedure, due to their limited size, it is almost impossible to have the same distribution of actions between the training, test, and validation sets. Therefore, considering that the best model checkpoints are selected based on the minimum error on the validation set, this is very often not reflected on the test set. To obtain more reliable results, it is necessary to repeat the experiments with much larger source and target datasets.

6.2 Other TL techniques

This Section presents the experiments related to TL techniques other than fine-tuning. The aim of these experiments is to understand whether other approaches for DTL can lead to improvements over standard fine-tuning, and whether they can mitigate the main TL problems, i.e. catastrophic forgetting and a biased pre-trained model. In particular, since ACCAD seems to be the target dataset where TL is more effective, the results shown in this Section focus only on the latter. Furthermore, to keep the notation simple, only the 30% partition is used, as it is neither the smallest nor the largest, and can therefore be representative of all the other partitions. Furthermore, since the fine-tuning performed much better on PVRED with one RNN layer, the experiments focus on this configuration of NN and on the HRI model.

6.2.1 PVRED

Some other TL techniques were tested on the PVRED architecture with one RNN. In particular, one experiment focused on freezing the entire pre-trained weights of NN, and replacing only the last linear layer, which is then tuned. Another experiment instead followed a similar approach, stacking an additional RNN layer. Figure 6.8 shows the results of the validation Euler Error along the training epochs.

As can be seen from the results, the best performing approach is the fine-tuning one, which achieves the lowest error with the fastest convergence. It is interesting to note that the two other TL techniques achieve a lower Euler Error compared to the scratch model, but only after many training epochs. The frozen pre-trained model, without stacking an additional RNN layer, improves faster in the first epochs than the one with the additional layer. This is because it is a simpler architecture that is easier to optimise. The fact that the fine-tuning approach is the one that achieves the best performance was expected, as it is a relatively simple NN architecture overall.

6.2.2 HRI

Similarly to PVRED, the experiments tested other TL approaches consisting of freezing and/or stacking other Graph Convolutional Network (GCN) lay-

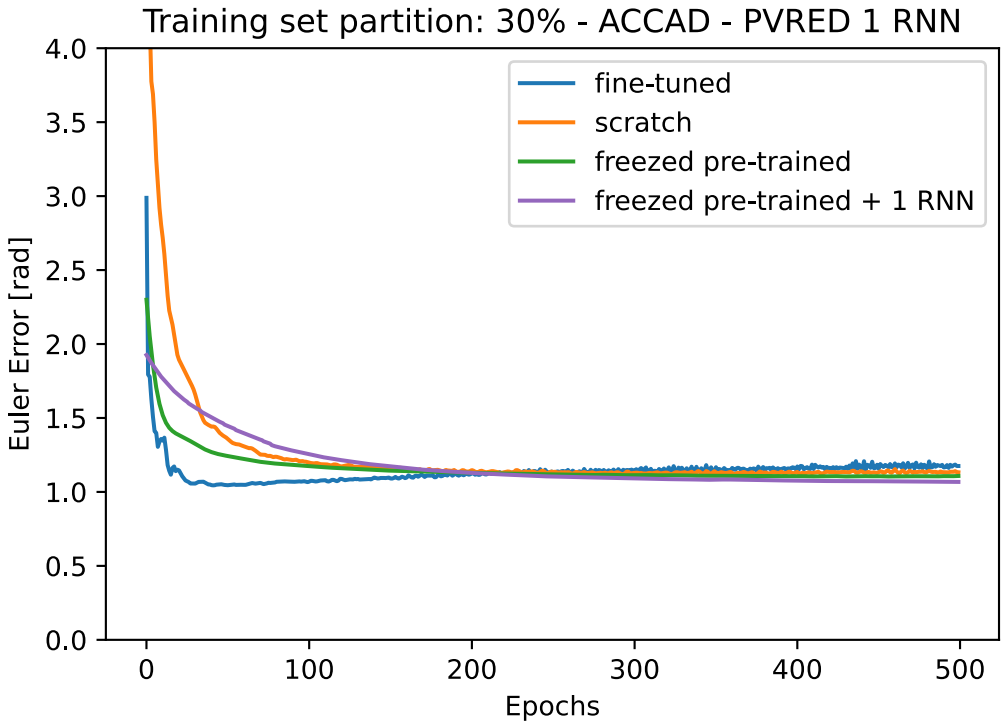


Figure 6.8: A comparison between different TL approaches for PVRED based on the 30% partition of the ACCAD target dataset.

ers. The first experiment consisted of freezing the entire pre-trained NN and replacing the last layer, which is the only one trained. Other experiments focused on freezing the pre-trained network and stacking several GCN layers. The idea is that in this way the model can use the pre-trained features learned from the source dataset and elaborate them with the fresh new layer to make the predictions. Several combinations were tested, but only the one with two additional GCN is shown in the results, as different combinations led to very similar behaviour. Figure 6.9 shows the validation Euler Error along the training epochs.

In this case, all the TL approaches perform very similarly, although the fine-tuned one is slightly better than the others. Additional approaches, not reported here for the sake of simplicity, were tested, but the fine-tuned results were never improved. Notably, the tested TL approaches included unfreezing the layers that implement the motion attention mechanism. Probably, unlike PVRED where the fine-tuning works best because of the simpler NN architecture, in the case of HRI this is due to the small amount of data used to pre-train the model. Given also the above considerations about HRI, it would be necessary to repeat all

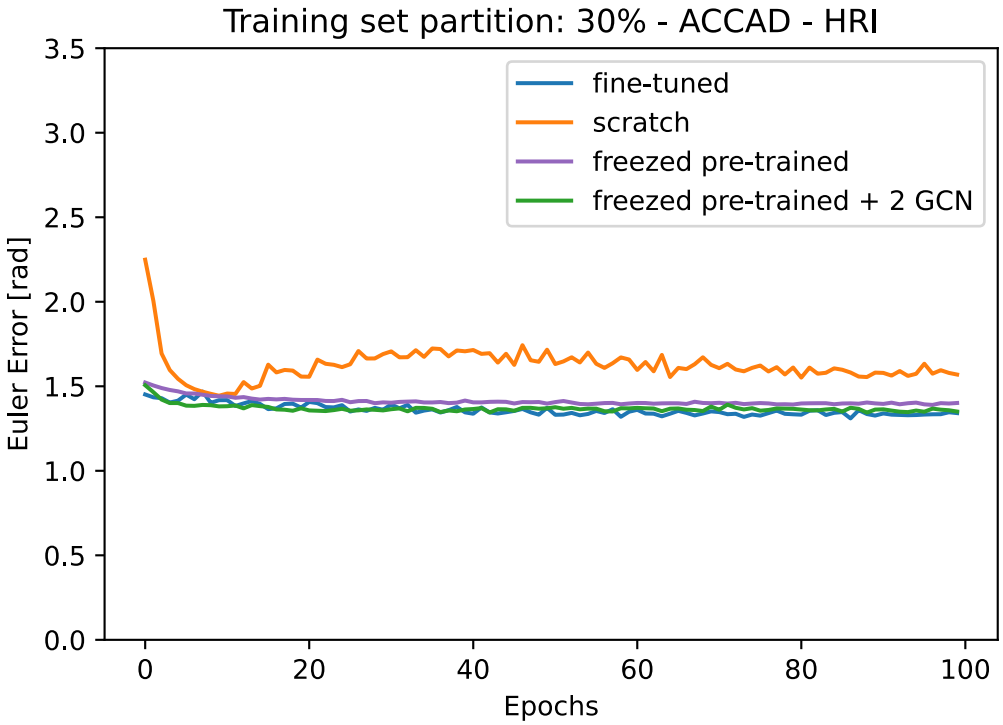


Figure 6.9: A comparison between different TL approaches for HRI based on the 30% partition of the ACCAD target dataset.

the experiments with a much larger source dataset in order to extract the true potential of this model.

7

Conclusions

The study of this thesis explored Transfer Learning (TL) techniques in the context of Deep Learning (DL) models applied to the task of Human Motion Prediction (HMP). The aim was to determine whether the use of human body knowledge embedded in pre-trained models could improve the accuracy of motion predictions, when dealing with small, domain-specific datasets. To achieve this goal, different experiments were conducted using state of the art DL models designed for the problem of HMP. In the case of TL approaches, these models were first pre-trained on one of the largest datasets available, i.e. Human 3.6M (H36M), and then fine-tuned to predict motion sequences within the target datasets. In this regard, three distinct target datasets were selected: 1) ACCAD, which closely resembles the source dataset, 2) GRAB, consisting of actions similar to a Collaborative Robotics context, and 3) DanceDB, a dataset significantly distinct from the source dataset. In terms of DL model selection, two were chosen, namely Position-Velocity Recurrent Encoder-Decoder (PVRED) and History Repeat Itself (HRI), which represent the primarily DL approaches currently used for HMP. PVRED is based on a Recurrent Neural Network (RNN) architecture, while HRI uses a motion attention mechanism combined with a Graph Convolutional Network (GCN) architecture.

Several TL techniques have been tested and compared with respect to training a model from scratch. This aimed to determine whether these strategies can bring to performance improvements when dealing with a limited amount of data, as is the case in Collaborative Robotics. Furthermore, different partitions of the target datasets were used to analyse the experimental results in terms of

dataset dimensionality. The evaluation of DL models performance was based on HMP metrics from the literature, while additional TL-specific metrics were used to assess the effectiveness of the TL approaches.

The results show improvements of the fine-tuning approach compared to training a model from scratch in the case of PVRED, which is the simpler Neural Network (NN) architecture. In this case, fine-tuning showed significant improvements both in the decrease of the minimum error and in the reduction of the number of training epochs required to converge. Moreover, the NN configuration of PVRED with one RNN show the grates benefits from fine-tuning. Regarding the dataset, the best improvements of fine-tuning were obtained in the case of ACCAD, which is the closer to the source dataset. Conversely, when examining the results for HRI, the fine-tuning approach generally showed improved performance compared to training from scratch. However, these improvements were less pronounces and in some cases the scratch model performed better. Similarly to PVRED, the fine-tuning approach demonstrated its effectiveness primarily on the ACCAD dataset. In addition, other TL techniques were tested for both the DL architectures. However, none of these alternative approaches were able to improve the gains achieved by fine-tuning.

Several notable conclusions can be drawn from these results. First, it is evident that the fine-tuning approach generally outperformed the training of models from scratch. This effect was particularly pronounced for the simpler NN architecture, i.e. PVRED, and for smaller partitions on the ACCAD dataset. This observations suggest that DL models may not be able to acquire an understanding of human motion inherently, independent of the actions they see during training. In particular, TL demonstrated its effectiveness when the target dataset closely resembled the source dataset. The results also showed that pre-trained knowledge derived from the source domain may not be sufficient for more complex NN architectures such as HRI. The source dataset, H36M, despite being one of the largest and most used datasets, is limited in size and contains a relatively small number of actions, which introduces significant bias into the pre-trained models. However, the results for PVRED suggest that if the pre-trained knowledge is sufficiently rich, TL approaches can significantly improve performance while reducing the need for extensive data collection to fine-tune the models. These observations are of particular importance in the field of Collaborative Robotics, where collecting new data can be costly and time expensive.

In terms of future work, an interesting direction to explore is the use of a much larger source dataset, such as Achieve of Motion capture As Surface Shapes (AMASS). This would serve to increase the richness of the pre-trained knowledge, while reducing the bias introduced by the limited set of different actions in H36M. Such direction has the potential to enhance the effectiveness of TL, particularly in the context of more complex architectures such as HRI, and to investigate the conclusions of this study in a broader perspective. It is also worth noting that the field of HMP is developing rapidly. Consequently, in the upcoming years more sophisticated DL models and much larger datasets will become available, offering opportunities for improved performance and broader applicability in the near future.

References

[1] Lalainne Anne J Abel, Toni Ceciro N Oconer, and Jennifer C Dela Cruz. "Realtime object detection of pantry objects using yolov5 transfer learning in varying lighting and orientation". In: *2022 2nd International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET)*. IEEE. 2022, pp. 1–7.

[2] Emre Aksan, Manuel Kaufmann, and Otmar Hilliges. "Structured prediction helps 3d human motion modelling". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 7144–7153.

[3] Emre Aksan et al. "A spatio-temporal transformer for 3d human motion prediction". In: *2021 International Conference on 3D Vision (3DV)*. IEEE. 2021, pp. 565–574.

[4] Advanced Computing Center for the Arts and Design. *ACCAD*. URL: `https://accad.osu.edu/research/motion-lab/mocap-system-and-data` (visited on 08/21/2023).

[5] Emad Barsoum, John Kender, and Zicheng Liu. "Hp-gan: Probabilistic 3d human motion prediction via gan". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2018, pp. 1418–1427.

[6] University of Cyprus Computer Graphics Lab. *DanceDB*. URL: `https://dancedb.eu/` (visited on 03/25/2023).

[7] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[8] Katerina Fragkiadaki et al. "Recurrent network models for human dynamics". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4346–4354.

[9] Alessandro Gasparetto, Lorenzo Scalera, et al. "A brief history of industrial robotics in the 20th century". In: *Advances in Historical Studies* 8 (2019), pp. 24–35.

[10] Partha Ghosh et al. "Learning human motion models for long-term predictions". In: *2017 International Conference on 3D Vision (3DV)*. IEEE. 2017, pp. 458–466.

[11] *Graphics lab motion capture database*. URL: http://mocap.cs.cmu.edu/ (visited on 09/02/2023).

[12] Liang-Yan Gui et al. "Few-shot human motion prediction via meta-learning". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 432–450.

[13] Saurabh Gupta and Rajendra Prasad Mahapatra. "Deep Custom Transfer Learning Models for Recognizing Human Activities via Video Surveillance". In: (2023).

[14] Mohammadreza Iman, Hamid Reza Arabnia, and Khaled Rasheed. "A review of deep transfer learning and recent advancements". In: *Technologies* 11.2 (2023), p. 40.

[15] Catalin Ionescu et al. "Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.7 (2014), pp. 1325–1339. DOI: 10.1109/TPAMI.2013.248.

[16] Deepak Kumar Jain et al. "GAN-Poser: an improvised bidirectional GAN model for human motion prediction". In: *Neural Computing and Applications* 32.18 (2020), pp. 14579–14591.

[17] Glenn Jocher. *YOLOv5 by Ultralytics*. Version 7.0. May 2020. DOI: 10.5281/zenodo.3908559. URL: https://github.com/ultralytics/yolov5.

[18] Ali Keshvarparast et al. "Collaborative robots in manufacturing and assembly systems: literature review and future research agenda". In: *Journal of Intelligent Manufacturing* (2023), pp. 1–54.

[19] Namhoon Lee and Kris M Kitani. "Predicting wide receiver trajectories in american football". In: *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2016, pp. 1–9.

[20] Maosen Li et al. "Dynamic multiscale graph neural networks for 3d skeleton based human motion prediction". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 214–223.

[21] Wei Liu, Karoll Quijano, and Melba M Crawford. "YOLOv5-Tassel: Detecting tassels in RGB UAV imagery with improved YOLOv5 based on transfer learning". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 15 (2022), pp. 8085–8094.

[22] Matthew Loper et al. "SMPL: A skinned multi-person linear model". In: *ACM transactions on graphics (TOG)* 34.6 (2015), pp. 1–16.

[23] Kedi Lyu et al. "3D human motion prediction: A survey". In: *Neurocomputing* 489 (2022), pp. 345–365.

[24] Naureen Mahmood et al. "AMASS: Archive of motion capture as surface shapes". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 5442–5451.

[25] Wei Mao, Miaomiao Liu, and Mathieu Salzmann. "History repeats itself: Human motion prediction via motion attention". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*. Springer. 2020, pp. 474–489.

[26] Wei Mao et al. "Learning trajectory dependencies for human motion prediction". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 9489–9497.

[27] Julieta Martinez, Michael J Black, and Javier Romero. "On human motion prediction using recurrent neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2891–2900.

[28] Marzieh Mozafari, Reza Farahbakhsh, and Noel Crespi. "A BERT-based transfer learning approach for hate speech detection in online social media". In: *Complex Networks and Their Applications VIII: Volume 1 Proceedings of the Eighth International Conference on Complex Networks and Their Applications COMPLEX NETWORKS 2019 8*. Springer. 2020, pp. 928–940.

[29] Andrey Rudenko et al. "Human motion trajectory prediction: A survey". In: *The International Journal of Robotics Research* 39.8 (2020), pp. 895–935.

[30] XIONG Ruo-chu and CHEN Xiao-lei. "Progress in the application of neurosurgical collaborative robot systems." In: *Chinese Journal of Contemporary Neurology & Neurosurgery* 23.1 (2023).

[31] Fahad Sherwani, Muhammad Mujtaba Asad, and Babul Salam Kader K Ibrahim. "Collaborative robots and industrial revolution 4.0 (ir 4.0)". In: *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*. IEEE. 2020, pp. 1–5.

[32] Omid Taheri et al. "GRAB: A dataset of whole-body human grasping of objects". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*. Springer. 2020, pp. 581–600.

[33] Chuanqi Tan et al. "A survey on deep transfer learning". In: *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III 27*. Springer. 2018, pp. 270–279.

[34] Vijay Srinivas Tida and Sonya Hsu. "Universal spam detection using transfer learning of BERT model". In: *arXiv preprint arXiv:2202.03480* (2022).

[35] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[36] Timo Von Marcard et al. "Recovering accurate 3d human pose in the wild using imus and a moving camera". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 601–617.

[37] Hongsong Wang et al. "PVRED: A position-velocity recurrent encoder-decoder for human motion prediction". In: *IEEE Transactions on Image Processing* 30 (2021), pp. 6096–6106.

[38] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. "A survey of transfer learning". In: *Journal of Big data* 3.1 (2016), pp. 1–40.

[39] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *Advances in neural information processing systems* 27 (2014).

[40] Zhuangdi Zhu et al. "Transfer learning in deep reinforcement learning: A survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023).