UNIVERSITAT POLITÈCNICA DE CATALUNYA

Doctoral Programme:

AUTOMÀTICA, ROBÒTICA I VISIÓ

PhD Thesis:

Visual understanding of human behavior: 3D pose, motion, actions and context

Alejandro José Hernández Ruiz

Advisor:

Francesc Moreno Noguer, Prof.

June 2022

Abstract

Visual understanding of human behavior is a very broad topic that, in the abstract, means understanding what a person or group of people is doing in an image or video. In practice, it can be broken down into a series of steps: detecting people in the image, estimating their posture and motion, recognizing objects in the environment, recognizing the action performed, predicting the subsequent motion, and predicting the next actions to be performed. Each of these steps is a computer vision task, and in this thesis we will focus on the following:

- 3D action recognition, identifying actions being performed based on the movements of people.
- 3D motion prediction, predicting the motion of a person based on a sequence of previous motion.
- Program generation, generate a program with an expected target that can adapt its behavior depending on the context.

For each of these tasks, the main contributions of this thesis are:

- A novel method for action recognition that uses a 3D CNN in conjunction with Euclidean Distance Matrices to analyze motion sequences.
- State of the art results for motion prediction using a generative adversarial network that can generate realistic sequences of up to 4 seconds.
- The creation of a new type of neural network, the Neural Cellular Automata Manifold, which can generate programs in the form of cellular automata whose behavior is learned from data.

In summary, understanding human behavior is central to many applications and is not trivial to solve. However, we propose methods for recognizing actions, predicting movements, and generating programs, and we have achieved very good results on each of these tasks.

Keywords action recognition, 3D CNN, motion prediction, generative adversarial networks, program generation, neural cellular automata.

Contents

Abstract **iii**

Contents **v**

1 Introduction **1**
1.1 Objectives . . . 3
1.2 Contributions . . . 3
1.2.1 3D action recognition . . . 3
1.2.2 Motion prediction . . . 4
1.2.3 Program generation . . . 4
1.3 Publications . . . 5
1.4 Thesis overview . . . 5

2 Definitions and Methods **7**
2.1 Computer Vision . . . 7
2.1.1 Image-like data . . . 7
2.1.2 Complex Behavior . . . 8
2.2 Relevant Computer Vision Tasks . . . 9
2.2.1 Pose Estimation . . . 9
2.2.2 Action Recognition . . . 10
2.2.3 Motion Prediction . . . 11
2.2.4 Program Generation . . . 11
2.3 Deep Learning Methods . . . 12
2.3.1 Classification Models . . . 12
2.3.2 Generative Models . . . 15

3 3D action recognition **21**
3.1 Related work . . . 23
3.1.1 Skeleton Sequence based Methods . . . 23
3.1.2 Image Sequence based Methods . . . 25
3.1.3 Other Methods . . . 25
3.2 Method . . . 26
3.2.1 EDMs over Transformed Skeletons . . . 27
3.2.2 3D CNNs over Distance Matrices . . . 28

3.2.3 Network Training . . . 29
3.2.4 Network Architecture Details . . . 29
3.3 Experiments . . . 29
3.3.1 Datasets . . . 30
3.3.2 Implementation and Training Details . . . 30
3.3.3 Comparison with State of the Art . . . 31
3.3.4 In-depth Analysis of DM-3DCNN . . . 34
3.4 Chapter summary . . . 36

4 3D motion prediction 37
4.1 Related work . . . 38
4.2 Problem Formulation . . . 40
4.3 Model . . . 40
4.3.1 STMI-GAN architecture . . . 40
4.3.2 Losses . . . 42
4.4 Metrics for motion prediction . . . 44
4.5 Implementation Details . . . 45
4.6 Training Details . . . 46
4.6.1 Data Augmentation . . . 47
4.6.2 Model Parameters . . . 47
4.6.3 Training Meta-Parameters . . . 48
4.7 Experiments . . . 48
4.7.1 Motion Prediction . . . 48
4.7.2 Occlusion Completion . . . 53
4.8 Chapter summary . . . 53

5 Program Generation 55
5.1 Related work . . . 57
5.2 Neural Cellular Automata Manifold . . . 58
5.2.1 Architecture Details . . . 59
5.2.2 DNA-encoding . . . 61
5.2.3 Dynamic Convolutions . . . 62
5.2.4 Neural Cellular Automata Architecture . . . 62
5.3 Experiments . . . 63
5.4 Chapter summary . . . 66

6 Concluding remarks 69
6.1 3D Action Recognition . . . 69
6.2 Motion Prediction . . . 69
6.3 Program Generation . . . 70

Bibliography 71

Chapter 1

Introduction

Understanding human behavior is one of the hottest topics in computer vision research today. It enables a wide range of applications in multimedia, gaming, augmented reality, video web search, automated surveillance, and human-robot interaction.

In the broadest sense, when we talk about understanding human behavior, we are referring to general understanding of actions and its context. This would require a general visual understanding of the world. The proposal for this ambitious task dates back to 1966, when Papert *et al.* [77] proposed the summer vision project, the goal of which was to develop a computer vision system capable of recognizing arbitrary objects in an image and matching them with a dictionary of known objects. And all of this was to be developed during the University summer break using 1966 computer equipment. More than 5 decades later, we are still struggling with the common task of object perception. Although recent breakthroughs made possible by large data sets and modern hardware have made us confident in the specific task of object classification in single images, we are still far from having a general method capable of understanding what these objects are for, how to interact with them, or what their properties or materials are.

In this work, we are interested in a visual understanding task with a narrower scope. In short, we want to be able to recognize what a person is doing in an image or video. When we think about how people perform this task, it is usually divided into a number of sub-tasks: Recognizing people in the picture and identifying their parts; observing the position of those parts, the environment, and the objects involved; and finally imagining their movement. Then we are able to recognize the action being performed.

As a thought experiment, we can analyze our own thinking in the following example: What action is performed in figure 1.1a? Any human would easily answer: it is a football player chasing the ball. We can also add that the player is very close to the goal, even if it is not visible in the image. Therefore, it is very likely that the next action will be shooting the ball. In the picture we can also see that the player of the opposing team is about to catch the ball. However, the other player in the

upper left corner of the picture cannot take part in the action because he is too far away. We can also see the spectators in the background very clearly and realize that they are not involved in the action. Even more, most people would recognize the identity of the player. Each of these steps is a relevant computer vision task, which we will define in sec. 2.1.2.

(a) Example of human action being performed:
football player running, pose by RT-CPM [11]

(b) Semantic segmentation and pose detection by the Mask R-CNN. [29]

Figure 1.1: Examples for motivation

Recognizing other people in a picture is an easy task for any human. It is also easy for a person to recognize the body parts and their relationship to each other, to infer movement from the pose and distance from the perspective. Also, in a known scenario, we can recognize what specific action is being performed and make a good estimate of the outcome of the action. For computers, each of these tasks is very complex. As we can see in Figure 1.1b, current methods are approaching human levels in human recognition and 2D pose estimation. The next task is then to recognize the action being performed. The current methods for action recognition require many assumptions or abstractions because the machines lack the necessary background knowledge. That is, action recognition methods often assume that there is a single person in the image, that the person is relatively close to the camera, and that the detection of body parts is reliable enough. Moreover, people are usually represented by a fixed skeletal shape that is perfectly symmetrical and assumed to have all limbs. We can argue that we are still far from a general method that can solve all these problems together and achieve human level performance.

We believe that it is crucial for action recognition to work with sequences rather than still images, because a sequence contains more information than the sum of its images. It is the relationship between the images through which we can uniquely identify an action performed. Therefore, videos are the ideal medium to learn this task. Even in a very short video, a complex action can be performed, and the interaction of humans with their environment is very complex.

Deep Learning models have recently shown excellent performance on complex visual tasks. Their inherent complexity and tailored architecture for visual understanding, along with huge amounts of data and computational power, are key to their success. These models are nowadays the most powerful tools for many of the topics related to our thesis, such as pose estimation and human action recognition. However, it is not a trivial task to assemble the various pieces into a general model for action recognition. To accomplish this, we will use our own understanding of how humans perform action recognition. This strategy follows the main idea of [27]: By incorporating prior knowledge into the design of our models, we can obtain clearer models that are easier to debug and improve, and that perform better in the long run.

1.1 Objectives

The main objective of this doctoral thesis is to perform visual understanding of human actions. Visual understanding means creating models that are capable of capturing the full complexity of human body pose and shape, as well as its evolution in space and time.

We can divide the main objective in the following specific objectives:

1. Pose-based action recognition: our objective is to classify pose sequences into different action categories, e.g., running, jumping, handshaking. We will extract pose sequences from videos using SOTA methods and propose new methods for their analysis.

2. Pose sequence prediction: after analyzing the pose, our next objective is to predict the pose in subsequent frames. This is a challenging problem due to free will and complex human behavior. Therefore, current methods can only make short-term predictions for sequences of up to one second. Our goal is to provide methods that outperform SOTA for predicting human motion by estimating coherent poses for up to four seconds in the future.

3. Context aware pose generation: With the final objective of generating plausible motions in a dynamic context, we aim to generate programs that can solve this challenging task.

1.2 Contributions

1.2.1 3D action recognition

From sequences of 3D skeletal data, we can recognize human actions. To this end, we combine a 3D Convolutional Neural Network with body representations based on Euclidean Distance Matrices (EDMs). However, an inherent limitation of EDMs is that they are defined up to a permutation of skeletal joints, i.e., random rearrangement of joints leads to many different representations. To address this problem, we

present a novel architecture that simultaneously and in an end-to-end manner, learns an optimal transformation of the joints while optimizing the other parameters of the convolutional network. The proposed approach achieves state-of-the-art results on 3 challenging benchmarks, where we outperform previous LSTM-based methods by more than 10 percentage points and also outperform other CNN-based methods while using almost 1000 times fewer parameters.

Publication:

- **3D CNNs on distance matrices for human action recognition**
  A. Hernandez Ruiz, L. Porzi, S. Rota Bulò, F. Moreno-Noguer.
  ACM on Multimedia Conference (ACMMM) 2017.

1.2.2 Motion prediction

Recent methods have shown promising results in predicting motion, but they are limited to predicting plausible motion over relatively short time periods (a few hundred milliseconds). They also typically ignore the absolute position of the skeleton with respect to the camera, limiting the range of applications to those that require only relative positions.

We propose a Generative Adversarial Network (GAN) to predict human 3D motion using a sequence of past 3D skeletal poses. Our system provides long-term predictions (one second or more) for both body pose and absolute position. Our approach relies on three main contributions. First, we represent the data using a spatio-temporal tensor of 3D skeletal coordinates, which allows us to formulate the prediction problem as an in-painting problem. Second, we design an architecture for learning the joint distribution of body pose and global motion, which is able to hypothesize large portions of the input 3D tensor with missing data. Finally, we propose two alternative metrics for motion prediction based on frequency distribution that can capture more realistic motion patterns. Extensive experiments show that our approach significantly improves the state of the art and also handles situations where previous observations are corrupted by occlusions, noise, and missing frames.

Publication:

- **Human motion prediction via spatio-temporal inpainting**
  A. Hernandez Ruiz, J. Gall, F. Moreno-Noguer.
  IEEE/CVF International Conference on Computer Vision (ICCV) 2019.

1.2.3 Program generation

The most challenging objective in this work is to generate programs that can predict human behavior given a dynamic context. The objective of such programs is to predict the correct pose for each state, taking into account the changing environment, to ultimately produce plausible behavior. The first step toward this ambitious

objective is to address the task of program generation.

Cellular Automata are a family of models of special interest because they offer the possibility of describing any algorithm without changing its underlying construction [13]. This is possible because each cell in a CA adjusts its state according to the context given a program. In the classical CA, the program is in fact a fixed set of rules that must be tested by hand, limiting the applicability of the model. In contrast, the Neural Cellular Automata [71] represents this program as an artificial neural network that can be trained to produce the desired behavior.

This model is key to our research because with proper adaptation, each step of the NCA could represent a state of a 3D skeleton and its context. The main limitation is that the NCA can generate a specific behavior for a single given context. To generalize the NCA model so that we can learn a variety of programs with a single model, we propose the Neural Cellular Automata Manifold model. Using Dynamic Neural Networks, we can generate NCAs that can adapt and perform complex behavior for any given context.

Publication:

- **Neural cellular automata manifold**
  A. Hernandez Ruiz, A. Vilalta Arias, F. Moreno-Noguer.
  IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) 2021.

1.3 Publications

- **3D CNNs on distance matrices for human action recognition**
  A. Hernandez Ruiz, L. Porzi, S. Rota Bulò, F. Moreno-Noguer.
  ACM on Multimedia Conference (ACMMM) 2017.
- **Human motion prediction via spatio-temporal inpainting**
  A. Hernandez Ruiz, J. Gall, F. Moreno-Noguer.
  IEEE/CVF International Conference on Computer Vision (ICCV) 2019.
- **Neural cellular automata manifold**
  A. Hernandez Ruiz, A. Vilalta Arias, F. Moreno-Noguer.
  IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) 2021.

1.4 Thesis overview

- **Chapter 2: Definitions and Methods.** In this chapter, we present the definitions that serve as a theoretical and conceptual framework. This is followed by an overview of the methods that form the basis for the methods used in this thesis. Note that this section does not discuss the state of the art

for each task.

- **Chapter 3: 3D action recognition.** In this chapter, we investigate the task of human action recognition from 3D skeletal data, also known as 3D action recognition. Here, we propose novel neural network models that use Euclidean Distance Matrices of the skeletal data and recognize the performed actions with high accuracy.
- **Chapter 4: 3D motion prediction.** In this chapter, we advance our understanding of human motion by predicting the motions of subjects longer than any previous work. Using Generative Adversarial Networks, we are able to learn a complex motion distribution that better represents reality.
- **Chapter 5: Program generation.** In this chapter we address the problem of program generation. We focus on creating a manifold of Neural Cellular Automata models, that allow us to generate programs in the form of neural networks.
- **Chapter 6: Concluding remarks.** Finally, we conclude our thesis with some remarks that compile our thoughts about the present and future work.

Chapter 2

Definitions and Methods

Long books and endless papers have been written on the subjects covered in this section, and those works explain in much greater depth and breadth the topics we will discuss here. Nevertheless, this section is important and necessary for two main reasons: 1) as a technical introduction for readers who are not experts on the topics covered in this thesis, with numerous references to the relevant original material. 2) to provide a starting point for the conversation with expert readers, allowing us to highlight relevant aspects of previous work while giving perspective to the contributions of this thesis.

2.1 Computer Vision

Computer vision is the "intelligent" processing of image-like data that enables machines to perform complex behavior. Let us first define what kind of data we will use, and then the definition of complex behavior.

2.1.1 Image-like data

Image-like data is a broad term, and the actual modality (or format) of the data depends on the sensors used to capture it, as well as the pre-processing techniques used. In the most common scenario, we are dealing with digital images that can be represented as 3D tensors. The first two dimensions of the tensor represent the plane of the image, while the third dimension encodes the color. By adding a fourth dimension that represents time, we effectively represent a video.

Digital images are the most abundant source of data for the purpose of this work, as the number of image capturing devices has exploded and huge datasets of this type of data are readily available today.

Another image-like data source of interest for our purposes is 3D skeletal data. This data is represented by a 5D tensor. In this tensor we represent the points of interest of reality. In our case, these are the joints of a 3D skeleton that encodes the

pose of a person. The first dimension represents the collection of joints. The next three dimensions represent the body in 3D space, with coordinates for height, width, and depth. And the last dimension encodes the sequence of time steps that we can use to represent the motion of the skeleton.

3D skeletal data can be obtained from many sources, but we mainly work with data from 3D point tracking systems, 3D skeletons from 2D + depth images (Kinect sensor), and 3D skeletons extracted from 2D images using pose estimation models. Recent wearable 3D body tracking systems are also compatible with the models proposed in this work, but they were not used.

In the Cellular Automata, we have a 2D grid representing a two-dimensional space. In this model, we treat each cell as an independent automaton. The state of the automaton is a tensor with N arbitrary values. This results in a tensor with 2D + N dimensions of a 2D world. As in the previous cases, we can add a dimension to represent evolution over time.

2.1.2 Complex Behavior

When we defined computer vision, we introduced the concept of complex behavior. In our understanding, complex behavior is any behavior that cannot be easily encoded in an algorithm or set of rules, but is nevertheless a behavior expected of living beings, such as humans, animals, or even unicellular organisms. Living beings gather information from their environment and process it in a way that allows them to take actions to maximize the likelihood of their survival. For example, while predicting the movement of a cat is not a trivial task for a machine, for a mouse it is a necessary skill to avoid being eaten. In our work, we explored several tasks, but we focused on action recognition, motion prediction, and program generation.

To analyze the images and sequences, computer vision relies on a variety of other fields such as geometry, physics, statistics, and machine learning. Machine learning is a field that focuses on methods that "learn" to produce complex behavior from data. Machine learning methods have proven to be very useful in solving computer vision tasks. In this thesis, we will focus exclusively on machine learning methods, and more specifically in deep learning. Deep Learning has the capability to create complex models that can generate the complex behavior we expect. We will explain this in more detail in Sec.2.3.

Machine learning methods allow us to solve many tasks that would otherwise be unsolvable for a computer. If we recall the example of the football player from the image 1.1a, in order to understand this image we would have to perform a variety of computer vision tasks, such as:

- **Action recognition**: recognizing the actions being performed in an image or video.
- **Image captioning**: providing the appropriate captions for an image or video.

- **Pose estimation**: estimating the position of the limbs of people in the image.
- **Image in-painting**: fill in missing parts of an image.
- **3D object localization**: estimate positions in 3D space from a 2D image or video.
- **Interaction analysis**: analyze interaction between humans and objects to correct previous estimations or help to predict possible outcomes of an action.
- **Subject re-identification**: identify previously seen subjects in an image.
- **Motion prediction**: predict the motion of a person given a initial motion sequence.
- **Action prediction**: predict the most likely action to be performed.

And after performing these tasks, we can use this information to produce behavior that would otherwise be impossible. For example, in the football player scenario, we could add a "virtual" commentator that says something along the lines of: *"Messi is quickly approaching the goal! He's going to shoot! He's under pressure, but he can do it! Common' messi don't hessitate!"*, which means that it understands the action, the actors, the possible outcomes and the interaction of all the elements in the scene. In this thesis we will focus only on some of these tasks, namely action recognition, motion prediction, and program generation.

2.2 Relevant Computer Vision Tasks

2.2.1 Pose Estimation

Pose estimation is about determining the posture of the subjects in an image. More specifically, we want to determine the position of the joints and the orientation of the limbs in 3D space. With this information, we can reconstruct a 3D skeleton that represents the person in a virtual space. Although pose estimation is not an objective of this thesis, we rely on it to process video data and obtain 3D skeletal data that we can use for our various goals. Cao *et al.* [11] propose a robust method that can perform pose detection in real time, and we use this method in Chapters 3 and 4.

Methods for pose detection are usually divided into steps for joint detection and pose reconstruction. The first step uses a keypoint detector, a model that can detect the points of interest for our application, i.e., the joints. Modern keypoint detectors are usually implemented with a convolutional neural network whose output is essentially a per-pixel classification of the different joints. We will discuss this in more detail in section 2.3. The second step is usually performed by a combination of optimization methods and heuristics that take the list of detected joints and link them in the most likely way, implicitly creating the 3D skeleton's limbs. In this second step, we can also refine the detected joints by removing unlikely detections, such as a foot detected above the head.

2.2.2 Action Recognition

Action recognition consists in classifying image sequences (videos) according to the behavior that the subjects exhibit. This behavior can be recognized based on the different positions and movements of the people involved. Although context can provide a significant amount of information, it is usually not the key to determining the action performed. For example, in a dataset for the action running, there may be sequences of people running in parks, on streets, football fields, etc.

There are interesting proposals that address this recognition task in an end-to-end manner. Simonyan *et al.* [84] propose to use a neural network that can extract relevant information directly from the image sequence to recognize the action. To this end, the network splits into two branches, where one branch takes a single image and the other branch takes the optical flow of the sequence. The idea is that one branch can analyze the contextual information like a normal image classifier and the other branch focuses on extracting the motion information. This was explicitly developed for small sequences of up to 25 frames. Recognizing actions in videos is a challenging task for several reasons:

- The subject appearance may vary during the sequence. The images may show the subject at different distances, they may show the whole subject, or it may be a close-up of a body part.
- The subject may be partially or completely occluded.
- There may be one or several subjects in the picture, and the action may be determined by the interaction between them.
- It may be difficult to distinguish the subject of interest from different subjects in the background (or foreground) that are performing different actions.
- Image based methods commonly used for this task rely more on texture detection than on understanding objects or subjects.
- The amount of data to be processed for each sample is large given the image resolution and video length, which make its processing computationally intensive.

A common strategy to overcome or mitigate these problems is to divide the task into two distinct sub-tasks: Pose estimation and action recognition from 3D skeleton data. Pose estimation works as described in the previous section by generating 3D skeletal data from image sequences. Action recognition from 3D skeleton data, also known as 3D action recognition, consists of classifying sequences of 3D coordinates representing joints of a skeleton in space. These sequences encode actions in terms of their motion, i.e., we can infer whether a skeleton is jumping, running, hugging, and many other actions just by observing its motion.

Processing action sequences in this way greatly reduces the amount of data to be processed at each step and allows us to use more complex and specific models for each task. In this context, the pose estimation model takes care of the problems related to the subject appearance, occlusion and background. The action recognition

model, on the other hand, can focus on dealing with multiple people and their interactions. This strategy also allows us to improve our results on the general action recognition task by implementing incremental improvements on each of the sub-tasks separately. In this work, we focus on improving of the 3D action recognition sub-task.

2.2.3 Motion Prediction

Sometimes it is not enough to recognize the action performed, we must also predict the motion of the subject. For example, when we drive a car, we need to anticipate the movements of other vehicles on the road in order to adjust our own behavior. From autonomous driving to any kind of human-machine interaction, motion prediction is critical not only for performance, but also for safety.

In general, motion prediction of living subjects is a very complex problem. Not only the state and intention of the subject, but also previous states and interactions with the environment must be considered. Of course, all this information is usually not available, but in certain cases it can be partially inferred from the motion sequence. Obviously, the time horizon of the motion prediction is a key factor for the complexity of the task.

For this reason, short-term prediction of 400ms or less is the most common case tackled so far [68]. At the time of this research, there is very little work that ventures into longer time horizons up to 1000ms [63], which is considered long-term prediction in the literature. And this is exactly the objective for this work, because the prediction of motion for more than 1 second is needed for many applications.

2.2.4 Program Generation

The results we obtained in motion prediction were a significant improvement over SOTA. However, we wanted to improve the performance of the model for the same task, we wanted to increase the time horizon of the predictions, and we wanted to include a dynamic context. In addition, we wanted to be able to predict not only the motion, but also the actions to be performed, i.e., the model should have an implicit understanding of what the action being performed is and how it relates to other behaviors.

After extensive research, we have concluded that the best approach to improving all of these goals is to learn to generate programs that can reproduce the complex behavior of the subject while also taking context into account. We approach this challenge using dynamic neural networks to learn to generate programs encoded in the form of a neural network that operates in the cell grid of a Cellular Automata model.

This line of research is pushing the envelope in current research, not only in the task of motion prediction, but in the field of Deep Learning in general.

2.3 Deep Learning Methods

Deep Learning is a family of machine learning methods that use Artificial Neural Networks (ANNs) in combination with representation learning techniques. It is commonly used to tackle tasks that would be difficult or impossible for a machine using a classical algorithm. The term Deep Learning was coined by Rina Dechter in 1986, and the adjective "deep" refers to the use of a ANN with many layers. These layers allow us to model increasingly complex functions that can be trained on data.

The Multilayer Perceptron (MLP) is perhaps the oldest model in the Deep Learning family. As can be seen in the figure 2.1, it consists of multiple layers of linear transformations, each followed by a non-linear activation function. The classical non-linear activation functions were the sigmoid: $\boldsymbol{sigmoid}(x) = (1 + \boldsymbol{e}^{-x})^{-1}$, and the hyperbolic tangent: $\boldsymbol{tanh}(x) = (\boldsymbol{e}^{2x} - 1)/(\boldsymbol{e}^{2x} + 1)$. The system was trained with the backpropagation algorithm, which uses the gradients of the errors to optimize the parameters of the network. The different components of the MLP can be viewed as a framework that has served to develop most current Deep Learning models. However, the model was initially very limited. One of the main problems of early deep models was the vanishing gradient problem. In classical MLP, the norm of the gradients decreased with each successive layer. This meant that the model could only grow so far before the gradients could no longer flow from output to input during backpropagation. This limited the size and complexity of the MLP. Because of this and other shortcomings, the machine learning community preferred other methods over the MLP for decades until the success of Convolutional Neural Networks revitalized the field.

In machine learning, there are two broad classes of models based on their output: Classification and Regression. Classification models predict a label given an example (e.g., dog, car), while regression models predict a continuous value relevant to our application (e.g., 15 degrees). We can think of generative models as a special case of regression models whose output is a set of variables that have a complex relationship that the model must take into account in its predictions. These models can also be understood as multivariate regression models, but with a high degree of complexity and non-linearity that is unattainable by classical statistical models.

2.3.1 Classification Models

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of ANN, which rely on the mathematical operator of convolution to perform their computations. CNNs have been

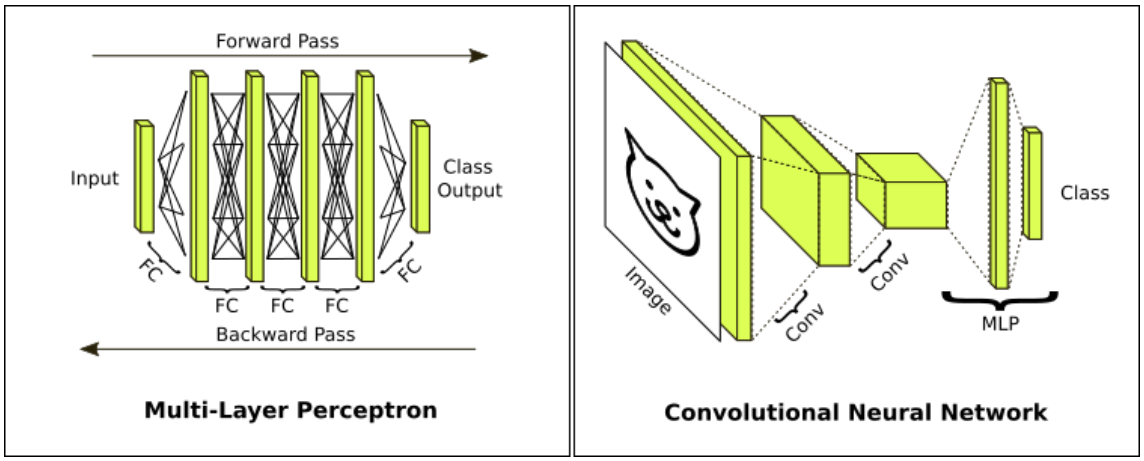

Figure 2.1: In the left size we can observe a diagram of the MLP. This particular MLP consists of four hidden layers that would convert an input tensor into a class output. During training, the MLP propagates the information in a forward pass from input to output, computes a prediction error, and then propagates the gradients of the error backward from output to input. The hidden layers of the MLP are linear transformations followed by nonlinearity. Since the weight matrix contains entries to connect each neuron in one layer to each neuron in the next layer, they are also called fully connected (FC) layers. On the right side, we can see the diagram of the CNN. Starting with an input image, it performs two convolutions that result in tensors that are smaller in height and width but larger in depth. The depth in this case represents the number of features extracted by each layer. Finally, the CNN is connected to an MLP that serves as a classifier.

applied with great success to a variety of problems, but were originally developed for image classification. LeNet [61] is one of the first successful examples of this type of network and was used to classify the handwritten digits of the Modified National Institute of Standards and Technology (MNIST) database. A CNN is formed by the succession of many convolutional layers which usually end up connecting to an MLP for classification. Figure 2.1 shows a visual representation of a CNN. Each convolutional layer is defined as follows: $\boldsymbol{Y}_n^I = \sum_m \boldsymbol{\kappa}_{mn} \star \boldsymbol{X}_m^I$, where $\star$ is the convolution operator, $\boldsymbol{\kappa}_{mn}$ is the convolutional kernel, and (m, n) are the input and output channels.

In 2012, AlexNet [59] won the ImageNet LSVRC [81] challenge and made a big leap in accuracy over previous methods. The success of AlexNet can be largely attributed to its size, which was the largest neural network model at the time of its release. The size of AlexNet was made possible by the use of Rectified Linear Unit (ReLU) activations: $\boldsymbol{ReLU}(x) = \boldsymbol{max}(x, 0)$, which mitigate the vanishing gradient problem.

Other important developments that helped CNNs and the field of Deep Learning in general make breakthroughs were the normalization layers, which enabled deeper networks and better models in general. In this thesis, we use two types of normalization layers: The batch normalization [43] layer, which tracks the statistics of each batch of samples during training and normalizes the samples according to those statistics. And the instance normalization [93] layer, which normalizes each sample based on the statistics of each channel independently.

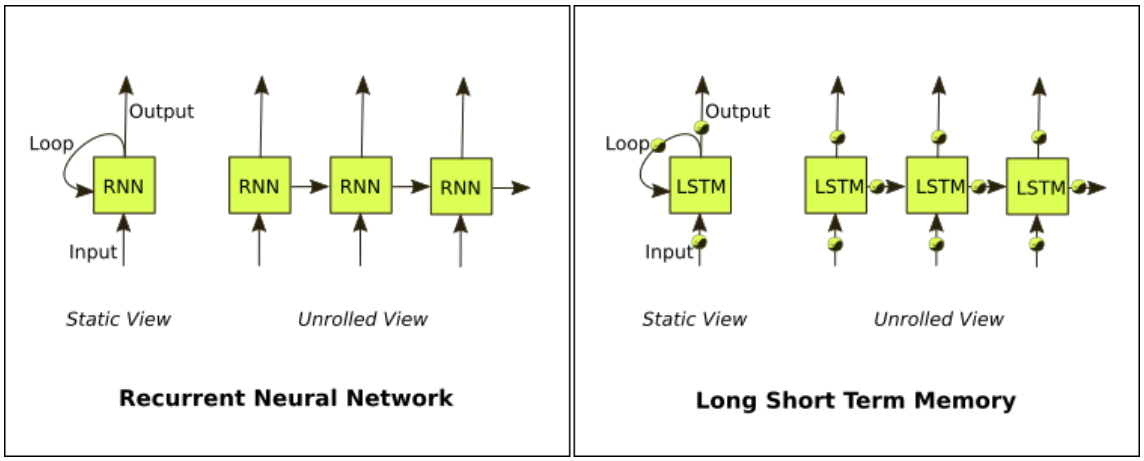

Figure 2.2: On the left, we can see a diagram of an RNN cell. An RNN cell is a self-contained network that, in the simplest case, consists of a single layer of an MLP. The RNN cell can be viewed as a static network with loop connections, or if it is "unrolled" on the sequence, we can view it as a network that is cloned and processes the inputs for each time step. The main difference with an MLP is that the RNN has a connection to itself in time, i.e. it receives two inputs, the input tensor from a sequence of inputs and the previous output. On the right hand side, the LSTM is very similar to a vanilla RNN, but it has gates that control the flow of information in and out of the cell and even the information that is stored within the cell.

Residual Neural Networks

Residual Neural Network (ResNet) [31] is an important addition to CNN architectures. It introduces the concept of the residual connection, which allows the network to grow much larger than previously thought without the vanishing gradient problem.

The residual connection is essentially a linear transformation from input to output computed in parallel to a block or section of the network. The output of the transformation is then added directly to the output of the block. Intuitively, we can think of this as splitting the problem into two distinct components, a linear component and a non-linear component, each learned by one side of the network. The principles introduced in the ResNet architecture are found in most current ANNs, and we use them extensively in this work as well.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a subfamily of neural networks with loop connections in their layers that allow them to process information at each step, taking into account the output of the previous state to properly adjust their function. See figure 2.2 for a visual representation. The sequential nature of RNNs makes them ideal for representing and processing text and other sequential data, but they were originally designed for processing temporal sequences. Long-Short Term Memory (LSTM) networks [37] are an improvement over simple RNNs; they introduce gates that control the flow of information within each layer. LSTMs have been successfully applied to various tasks, including NLP tasks [51]. They have been shown to be a

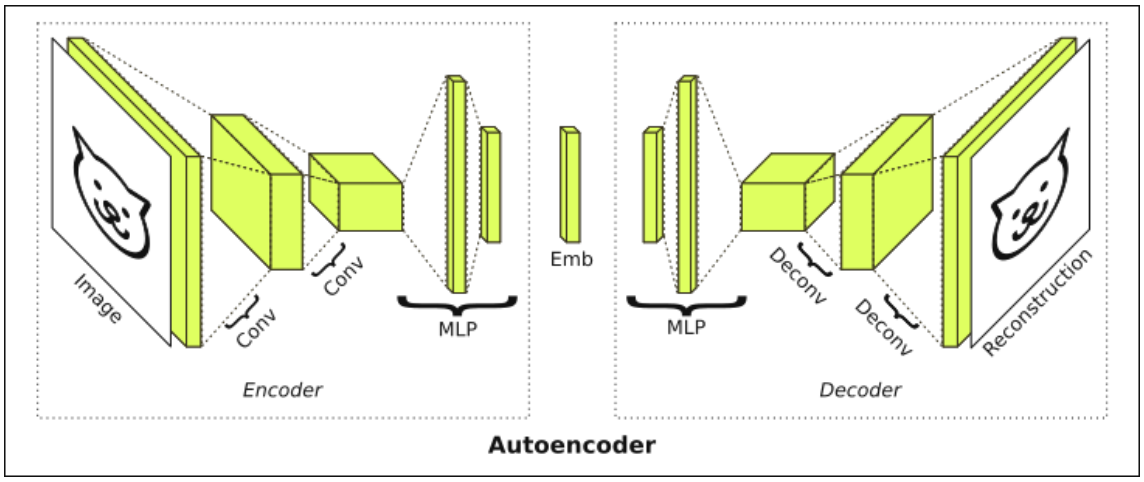

Figure 2.3: The Autoencoder combines an encoder network and a decoder network. The encoder converts an input image into an embedding tensor, and the decoder reconstructs the image from that embedding tensor. The encoder and decoder are essentially CNNs that mirror each other. The decoder contains transposed convolutions, also called deconvolutions, instead of convolutions. As the name implies, a transposed convolution is a convolution in which the inputs and outputs are swapped so that the image increases in height and width after each layer.

robust model, outperformed by more recent models, but still a good baseline model for many tasks such as action recognition.

An interesting property of RNNs is that they have been shown to be equivalent to Turing machines [42]. Neural Turing Machines (NTMs) [25] are a special type of RNN that has been proposed for program learning in general. This is of interest to us in the area of program generation, however NTMs can only learn a single program at a time.

2.3.2 Generative Models

Autoencoders

Autoencoders (AE) [36] consist of two subnetworks, an encoder network and a decoder network. The encoder network extracts features similar to a CNN trained for classification. These features are then passed to a decoder network, which reconstructs the image passed in the input. The encoder usually has a contractive architecture that reduces the spatial dimension while increasing the number of features. The decoder is usually the mirror image of the encoder. It increases the spatial dimension while decreasing the number of features. We can observe this architecture in figure 2.3.

Originally, it was thought that autoencoders must have an hourglass architecture to learn a dense feature space, but in fact sparse autoencoders [75] have also proven useful. Tying the weights of the network to learn symmetric features for compression and decompression was also initially thought to be a good idea, but later proved to be not useful and even counterproductive in the general case. These architectural

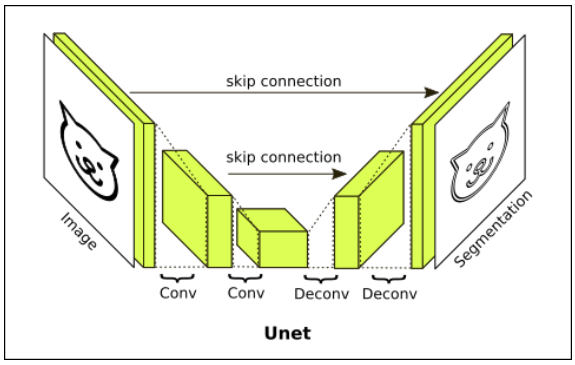

Figure 2.4: The Unet is a fully convolutional autoencoder, i.e., it lacks the MLPs found at the center of a regular autoencoder. The Unet has skip connections that pass information from the first part of the network to layers in the second part of the network that have the same spatial dimensions, skipping the main path of connections.

choices, which were an initial exploration of the Deep Learning field, showed that the behavior of these models is very complex and in many cases counterintuitive, and that sometimes the exact opposite of what is expected can improve the results.

The UNET [80] is an autoencoder architecture with skip connections between layers of the same resolution. This is a connection similar to the residual connection, but in this case without any kind of transformation. The skip connection, much like the residual connection, was designed to allow the network to preserve information across its layers. However, in this case, the goal is to maintain the spatial structure as much as possible so that the network can use the contextual spatial information at each step. This architecture was applied to semantic segmentation and achieved SOTA results when published.

Autoencoders can be trained to denoise data. This is done by adding random noise to the input and minimizing the error with respect to the clean data. When trained in this way, they are called denoising autoencoders. To complete the information masked by the noise, denoising autoencoders must learn the intrinsic properties of the data. Therefore, they are forced to learn a manifold [98]. Denoising autoencoders are considered generative models, but sampling their latent space remains a problem, as we can see in figure 2.5. A recent development, the Implicit Rank Minimizing AutoEncoder (IRMAE) [49], shows that adding some linear transformations to the latent space of the autoencoder actually improves it, mitigating the sampling problem. However, more complex models such as Variational Autoencoders or Generative Adversarial Networks are needed to produce images that people consider to be qualitatively good.

Variational Autoencoders

Variational Autoencoders (VAEs) [56] tackle the problem of generating images using an autoencoder, and more specifically they aim at solving the sampling problem of

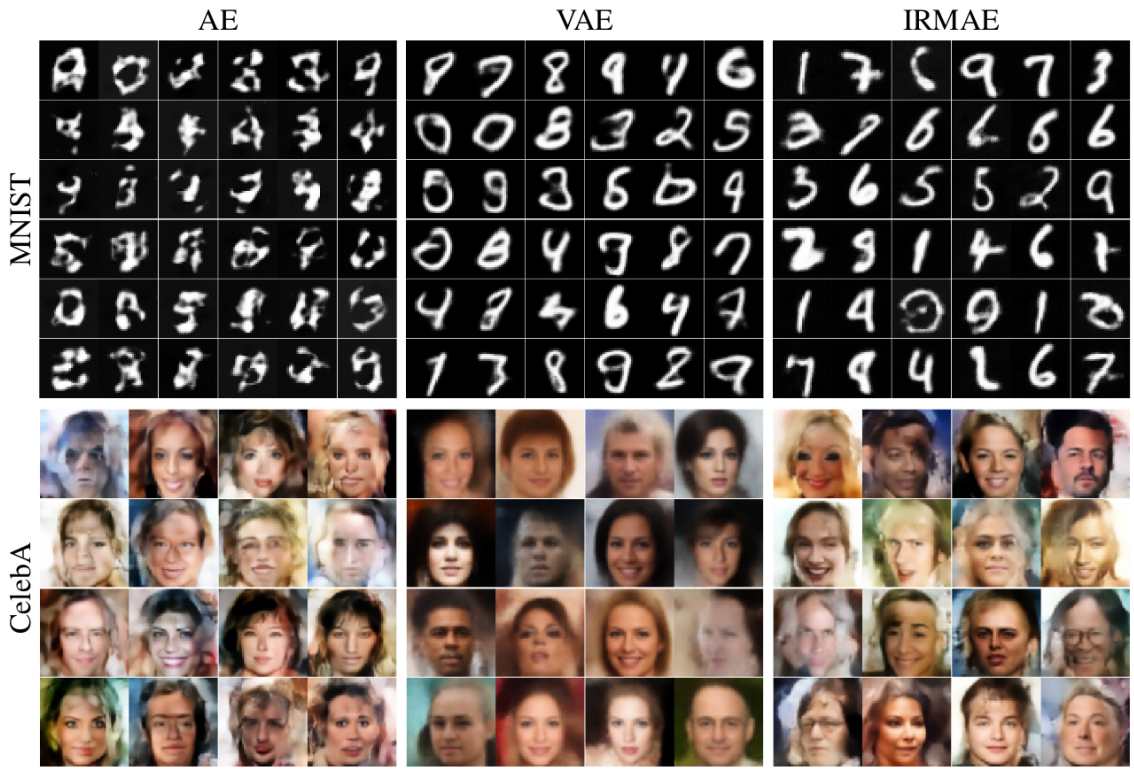

Figure 2.5: Image taken from [49]. Original caption: "MNIST/CelebA images samples from Multivariate Gaussian with covariance estimated from training set. From left to right are images generated from an unregularized AE, a VAE, and an IRMAE, respectively". We can observe that the embedding space of regular AE cannot be easily sampled to generate new images. In addition, we can state that both the VAE and the IRMAE are an improvement over the original. Finally, we can note that the images generated by the VAE have a certain smooth appearance.

the autoencoders by constraining the model to learn an easy to sample Gaussian distribution in its latent space. The VAEs encoder is very similar to that of the original Autoencoder, but its output, or latent vector, is divided in two parts, these two parts represent the mean and variance of a Gaussian distribution. In the next step, a sample from an multivariate Gaussian distribution with diagonal covariances is taken, and this sample is multiplied by the variance and added with the mean values taken from the encoder output. The resulting value is passed on to the decoder which will generate an image that probably resembles the input. VAE is trained end-to-end, with a reconstruction loss and an additional KL-divergence loss, which is imposed on the latent space, to avoid it from deviating too much from the original Gaussian distribution.

The VAE model main strength is that it is very easy to generate coherent images by sampling its latent space. The downside is that the resulting images tend to be very soft looking, and lose many details. This can be attributed to the Gaussian space being too harsh a constrain, but it is also due to the fact that the reconstruction loss used is an l2 loss which minimizes at the mean value, and this might be a less

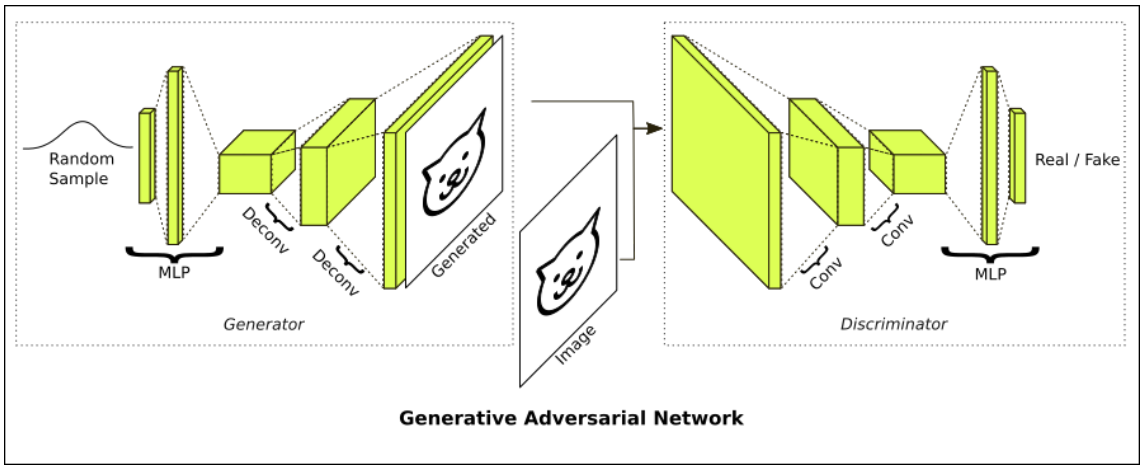

Figure 2.6: In GANs, we have a generator subnetwork and a discriminator subnetwork. These two networks are adversarials in their roles. The objective of the discriminator is to determine whether the image is a real image or a generated image. The objective of the generator is to generate images that fool the discriminator.

than ideal target for many cases.

Generative Adversarial Networks

Generative Adversarial Networks (GANs) [22] turn the Autoencoder framework on its head, by placing the decoder in the beginning of the network and then using the encoder as a classifier, we can train both networks in an adversarial manner. The network is trained by taking turns. In the encoder turn, it will learn to classify real images as real, and images generated by the decoder as fake. Because of their roles, in GAN jargon, the encoder is called a discriminator and the decoder is called the generator. In the decoder turn, it will learn to generate images trying to fool the discriminator in classifying them as real. To generate images, we get a random sample from a Gaussian distribution, and pass it to the generator as if it was the latent tensor of an autoencoder. By repeating this alternating optimization, we can train the two networks in an end-to-end manner.

GANs were originally proposed as a way to train robust networks, but they have proven to be a good way to generate images [54], motion [33] and many other data modalities.

Dynamic Neural Networks

Dynamic Neural Networks (DNNs) are neural networks that generate other neural networks. There are many motivations for creating a DNN, such as compressing the size of parameters [28, 104, 110], or improving models and thus performance for a task, such as weather prediction [57], action recognition [7], and video prediction [47]. But DNNs also enable entirely new tasks, such as asking the network questions about the objects in an image [18]. We are more interested in this last line of thought

because DNNs allow us to generate networks, and networks can define programs [25], so they are used in this thesis for just that purpose.

Chapter 3

3D action recognition

In recent years, 3D sensing technologies, and in particular RGBD cameras have become increasingly cheap and widely available. Devices such as the Kinect or Leap Motion, and the associated software libraries, allow for accurate 3D tracking of human body parts with minimal effort. Because of this, human action recognition algorithms working directly with 3D skeletal data have gained substantial popularity in the research community. As in many other fields of research, remarkable results have recently been obtained in this task by employing deep learning-based approaches [64, 83, 96, 114, 116] exploiting large-scale datasets [83].

Human action recognition from 3D skeletal data is inherently a sequence-based problem, which can be naturally tackled in the context of deep learning using recurrent networks. Indeed, many works [64, 83, 114, 116] propose Long Short-Term Memory (LSTM) networks working directly on the 3D coordinates of the body joints [64, 116] or hand-crafted geometric features derived from these [114]. An alternative, more recent approach is that of projecting and color coding the joint trajectories in image space [65, 103], obtaining a compact representation of the whole skeleton sequence that can be processed using standard convolutional networks. This allows the authors of [65, 103] to reuse well-tested CNN architectures (*i.e.* AlexNet [59]) and to exploit large scale image datasets (*i.e.* ILSVRC2012 [81]) to pre-train their networks, obtaining the current state of the art on a challenging large-scale action recognition dataset [83].

Following from the success of [65, 103], we propose a novel action recognition CNN based on the ResNeXt architecture of Xie *et al.* [107]. Differently from these works, however, we explicitly take the temporal nature of our problem into account, by constructing a network composed of *spatio-temporal* convolution and pooling operators. Furthermore, instead of the indirect, image-based representation employed in [65, 103], we encode the input skeletons as sequences of Euclidean Distance Matrices (EDM) computed over their joints. EDMs are rigid transformation-invariant representations of sets of points, which can be effectively processed by convolutional networks, as recently shown in [72]. Intuitively, as depicted in Fig. 3.1, convolutional neurons can learn to respond to local EDM structures which encode the spatial configurations

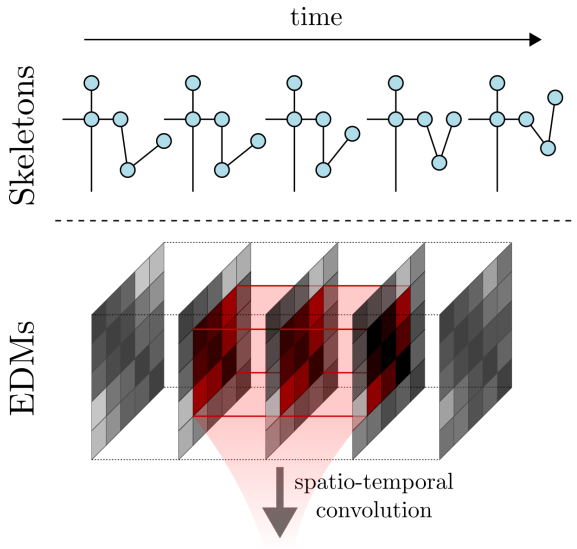

Figure 3.1: We encode sequences of skeletons in 3D space as stacked Euclidean Distance Matrices. By performing convolution both along the spatial and temporal dimensions our network learns to respond to the spatio-temporal dynamics of the input data.

of groups of 3D joints. By performing convolution also along the *time* dimension, we allow the network to learn to respond to the spatio-temporal dynamics of our data.

Euclidean Distance Matrices, however, are defined up to a permutation of the points they represent. By changing the order of the joints when computing an EDM, the region of the skeleton associated with each local neighborhood of the matrix would also change. This would lead the same convolutional neurons to respond in a radically different way to the same skeleton. To sidestep this problem, we augment our EDM representation with a learned combination of the input points. Thus, we endow our network with the capability of selecting the most advantageous distance matrix configuration for the purpose of classifying actions. This only adds a negligible amount of parameters to the overall model, which can be trained end-to-end together with the rest of the network.

In section3.3 We evaluate our network, named DM-3DCNN, on three benchmark datasets, including the challenging and recently released NTU RGB-D [83], obtaining competitive results w.r.t. other recent methods. On this particular dataset, we obtain the new state of the art, surpassing previous LSTM-based approaches by an average of 10 % accuracy and the CNN-based approach in [65] by 3 %, while using about 1000 times fewer parameters and operations.

Contributions. To summarize, this chapter includes three main contributions. *First*, we present an approach to human action recognition from 3D skeletal data represented as sequences of Euclidean Distance Matrices. To overcome the permutation ambiguity inherent in encoding the skeletons as EDMs, we compute the distance matrices from a learned combination of the joints which is general enough to include all possible

permutations. *Second*, we propose a ResNeXt-inspired [107] network architecture built from spatio-temporal convolution and pooling operators and taking sequences of EDMs as input. *Third*, we evaluate our approach on three benchmarks, obtaining the new state of the art on the large-scale NTU RGB-D dataset and surpassing recent CNN-based approaches [65, 103] while employing three orders of magnitude fewer parameters and operations.

3.1 Related work

Human action recognition is one of the most interesting topics of computer vision, and it has many use cases within the academy, robotics, surveillance, games and entertainment multimedia; because of this, the quantity and diversity of works is impressive, and impossible to cover in a single work. We will focus then in reviewing the works we consider relevant to this particular problem and to our approach.

The diversity of works and datasets also imply diversity in the representation of the actions. We will consider two main representations: skeletal data sequences and RGBD sequences.

3.1.1 Skeleton Sequence based Methods

Before the advent of the deep learning methods, most works focused in designing representation of the movements that could be learned or hand crafted, and allowed a simple model to classify or make inference on them.

The first type of methods are based in Support Vector Machines (SVMs), which are simple yet powerful classifiers that can easily learn to discriminate the sequences, when the features are able to convey the relevant information. To accomplish that, these features need to be both invariant to changes in the viewpoint and length of the sequence. In [50] the authors encode the sequence using temporal and view invariant representations based on different distance measures between the joints in a skeleton, computed in time and space dimensions. Similarly, [41] encodes the sequence using covariance features of the joint locations in a small time window. In [106], sequences were represented by Histograms of the location of 3D joints (HOJ3D), and in [76], by spatio-temporal Histogram of Gradients (HoGs) in the joint angles. Another way to abstract the complexity of the movements is to learn dictionaries to group and classify them easily [66, 115]. A very abstract representation can be found in [97], where the sequences are transformed into lie groups, mapped to its lie algebra, and passed through Dynamic Time Warping (DTW) for aligning and applied pyramidal fourier analysis. A different approach was proposed in [86], were a Gaussian Mixture Model (GMM) was used to learn a compact representation of the sequence and a hierarchical Hidden Markov Model (HMM).

Alternatively, instead of classifying the whole sequence at once, we can classify the frames of such sequence, and then combine this classification in a global predic-

tion. The idea is to be able to identify certain important frames or instances, that unequivocally identify the sequences. Roughly following this idea we find a family of methods called Multiple Instance Learning [109, 113].

More recently, a number of works leverage on the deep learning methods that are the state of the art in many pattern recognition problems. These works focus on creating models with deep architectures, that are capable to learn to classify the sequences from a very simple encoding or even raw data. We can find two main groups of architectures: those based in Recurrent Neural Networks (RNN) and variants like the Long Short Term Memory [37] (LSTM) network; and on the other hand the Convolutional Neural Networks (CNN) based methods.

The following models inspire their architectures by taking into account the separation of the body into parts of interest (e.g. left and right arms/legs, and the central part composed by torso, neck and head). The Hierarchical-RNN [17] takes each body part as the input to a separate RNN and combines them into a unique output which is used for classification. [116] proposes Co-occurrence LSTM, a modification of the LSTM cell that seeks to capture the correlation between the different parts involved in a movement. With similar inspiration, [85] proposed an LSTM with attention model that reweighs the relevance of the joints in the input at each time frame. And also we have the Part-aware LSTM [83], that modifies the LSTM cell to have the input to hidden state gates explicitly separated by each body part.

One of the most important factors to measure the information in a frame is the amount of movement. Some models build upon this idea. For instance, [96] proposes a modified LSTM model which incorporates the magnitude of the difference with respect to previous frame into the gating mechanisms of the cell. [64], introduces the ST-LSTM, a modified LSTM model that scans the input in a path following fashion over a graph, and such graph is constructed by unrolling the time component of the input; ST-LSTM also incorporates a trust gate to the cell, that basically takes an inverse distance measurement that reweighs the input to the empirically estimated trust over such input.

Regarding the CNN based methods, one of the first approaches was [16], which converted the sequences to images by representing the skeleton as a vector of pixels, and concatenating these vectors along the temporal dimension to create a single RGB image; this image was then fed to a CNN to perform action classification. More recently, a group of models leverage on widely known CNN architectures like AlexNet [59] to perform human action recognition. In these models some transformations and projections of the sequence are performed, yielding a single color image that serves as input to the network: in [103] the 3D skeleton is projected as a 2D image by framing it in the point of view of a camera along the X, Y or Z axis; the temporal dimension is converted into a color code, and finally the 2D image used as input is the superposition of all frames. It is worth to note also, that reusing known architectures is very convenient, as it allows for efficient training, and also allows

for building ensemble models, that usually perform better than the single network models; in [103] the output of three networks are combined to produce an ensemble classifier. [65] follows a very similar approach, the main difference being that the projection is done over a transformed coordinate system instead of the Cartesian system, and that the ensemble model combines ten networks instead of three.

3.1.2 Image Sequence based Methods

Differently to the skeleton based methods, image sequence methods do not explicitly model the human body. Instead they work with raw pixels, and try to find meaningful patterns in the distributions of pixels in each frame and its transformations over the time.

To apply convolutions over a sequence of images we can either represent the input sequence as a 3D volume of information over which we can apply 3D convolutions, or we can take the time dimension in the input as different channels and perform 2D convolutions. The first approach in this category was [46] which performed 3D convolutions over the video, in a similar manner as the 2D convolutions were applied to a single image. A number of convolutional layers were used to extract features and the network output was produced by a dense layer and a softmax classifier. This relatively straightforward approach is however limited however by a high computational cost. For this reason, other types of models using 2D CNNs were proposed: the family of models comprised by stream networks [84, 101], take as input two different representations, a single image and the optical flow of the sequence; these two representations are passed through a parallel inference architecture, that fuses the extracted features in its topmost layer. Another option is to represent the whole video as a single image, by transforming its representation to a sort of histogram over the movements in the video [7, 102]. Also worthwhile of mention models that take mixed approaches. For instance, in [40] 2D CNNs are used to extract features, and 3D CNNs over the computed features fuse the spatial and temporal information.

3.1.3 Other Methods

There are a few other methods that influence this work, and are important to mention. In the related problem of human pose estimation, we can find [72], where the authors show that the use of CNNs over EDMs as view invariant representation of the skeletons produces good results. Also, we consider the ResNeXt [107] architecture, as a reference of a top performing modern CNN architecture.

Drawing inspiration, ideas and design choices from all the works mentioned in this section, we propose a novel method that mixes them in a sensible way and produces excellent results for this problem.

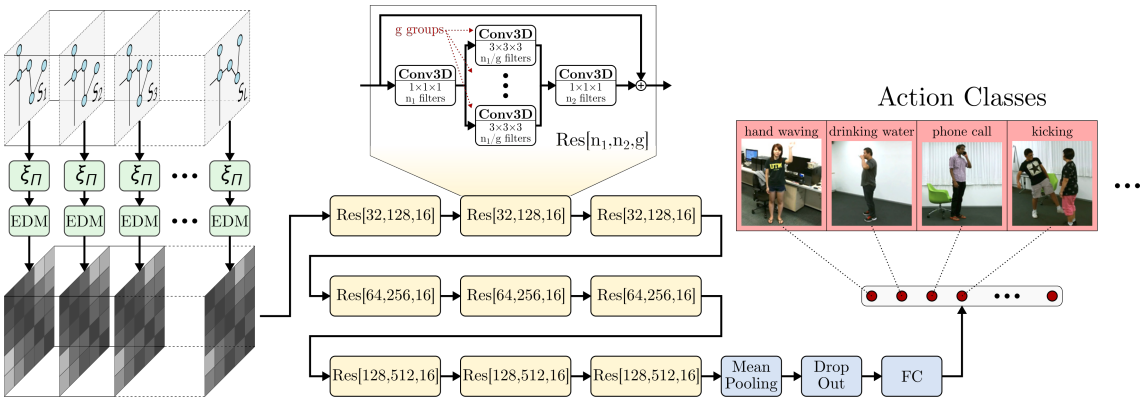

Figure 3.2: Overview of the proposed network architecture. Skeletons from the input sequence are transformed using a learned linear function ξ_Π and the output is used to compute a sequence of Euclidean Distance Matrices (green blocks). The matrices are stacked to form a 3D tensor, which is fed to a fully-convolutional network built from ResNeXt units [107] with 3D convolutions (yellow blocks). Finally, a classifier composed of global mean pooling, dropout and a fully connected layer with soft-max (blue blocks) is used to compute the predicted probability distribution over action classes (red blocks). Note that we are not showing the case of multiple skeletons per frame for the sake of simplicity.

3.2 Method

Our goal is to recognize human actions or interactions from temporal skeleton data, *i.e.* sequences of human body joints encoded as 3D points varying over time. In general, the sequences might comprise multiple human bodies, but we restrict our focus to interactions that involve up to two actors, since this is the setting commonly encountered in the main benchmark datasets (*i.e.* NTU RGB-D dataset [83] and SBU Interaction dataset [113]).

The problem setting is given as follows. Let J be the number of skeleton joints and let $\mathcal{S} \subset R^{3\times\mathsf{J}}$ be the set of all possible configurations of joints for a single body skeleton at some fixed time. For convenience we simply call skeletons the elements of $\mathcal{S}$. Joints are given as 3D points and each skeleton forms a $3 \times \mathsf{J}$ matrix with joints as columns. The input space for our action (or interaction) recognition problem is given by $\mathcal{X} = (\mathcal{S} \times \mathcal{S})^{\mathsf{L}}$, *i.e.* sequences of L pairs of skeleton configurations, where L is a fixed time-window length. We consider pairs of body skeleton in the sequence to account for up to two actors in the scene. In case only a single actor is in the scene, this will be repeated twice to form a pair. The set of possible actions (or interactions) to be predicted is denoted by $\mathcal{Y} = \{1, \ldots, \mathsf{K}\}$ and the action recognizer is a function $f_\theta : \mathcal{X} \to \mathcal{Y}$ parametrized by θ that assigns action labels in $\mathcal{Y}$ to sequences of pairs of skeletons in $\mathcal{X}$. The set of feasible parameterizations is denoted by Θ.

The state-of-the-art methods for human action recognition from skeleton data follow two main approaches: i) implementing the action recognition function f_θ as a deep recurrent neural network; ii) rendering the skeleton sequences as a single image and implementing f_θ as a deep CNN taking these images as input. Our solution is

closer in spirit to the latter type of approaches, but instead of encoding the sequence of skeletons as a single image, we retain one additional spatial dimension for representing skeletons and introduce by construction invariance to rigid transformations. This is indeed our *first contribution* that we implement by using distance matrices defined over a learned transformation of the skeleton data. Distance matrices are in fact invariant to rigid transformations by nature. Our *second contribution* consists in adopting a deep neural network with spatio-temporal convolutional operators applied to distance matrices extracted from skeleton data. This contrasts with the approaches in the literature that typically rely on standard CNNs, or recurrent neural networks.

3.2.1 EDMs over Transformed Skeletons

Many approaches in the literature [17, 86, 116] do not feed the action recognizer with the original skeleton data, but try manipulate the input to enforce some form of invariance to rigid transformations. This is indeed empirically shown to be beneficial to the human action recognition task. For instance, in [17] the skeletons are put into a canonical reference system through a change of coordinates. The solution we pursue to achieve invariance to rigid transformations is different and consists in representing skeletons in terms of distance matrices, which are inherently invariant to rigid transformations.

Given a $3 \times \mathsf{M}$ matrix Z of 3D points we define the corresponding Euclidean distance matrix $\mathsf{D} = \texttt{edm}(\mathsf{Z}) \in \mathbb{R}_+^{\mathsf{M}\times\mathsf{M}}$ as the nonnegative, symmetric $\mathsf{M} \times \mathsf{M}$ matrix with (i, j)th entry given by the squared Euclidean distance between the ith and jth columns of Z. That is, $\mathsf{D}_{ij} = \|\mathsf{Z}_i - \mathsf{Z}_j\|^2$, where $\|\cdot\|$ is the Euclidean norm, and $\mathsf{Z}_i \in \mathbb{R}^3$ is the ith column of matrix Z. Distance matrices are invariant to rigid transformations (*e.g.* translations, rotations, reflections) applied to the original points and thus suit well our purpose of having a representation for skeletons that is invariant to such transformations.

However, distance matrices are not invariant to permutation. This means that the EDM computed from a skeleton is sensitive to the ordering of the joints. In general, permutations might exist that have a negative impact on the final performance of the action recognition task (see Sec. 3.3.4), in particular if we aim to exploit local structures of the distance matrix via convolutional neural networks (see next subsection). To overcome this issue, we propose to learn a transformation of the skeleton that is sufficiently general to represent all possible permutations, and compute the distance matrices of transformed skeletons. By having this component embedded into the neural network, we give the classifier the freedom of emphasizing the importance of some joints and potentially discovering an optimal permutation of the joints that enhances local structures in the distance matrices. At the same time we preserve an invariance to rigid transformations in all layers that follow the distance matrix computation. In this chapter we keep the skeleton transformation

simple by considering a linear operator

$$\xi_{\Pi}(\mathsf{S}) = \mathsf{S}\Pi\,,$$

acting on $\mathcal{S}$, where $\Pi \in \mathbb{R}^{\mathsf{J}\times\mathsf{J}}$ is a $\mathsf{J}\times\mathsf{J}$ real matrix to be learned. This transformation encompasses permutations of joints as a special case.

Next we deal with the problem of encoding pairs of skeletons in terms of EDMs, since our problem setting assumes up to two body skeletons in the scene. Accordingly, let $\mathsf{S}, \hat{\mathsf{S}} \in \mathcal{S}$ be two skeletons and remind that $\mathsf{S} = \hat{\mathsf{S}}$ if a single skeleton is present. There are different ways in which the two skeletons can be encoded using an EDM representation. We consider the following two approaches:

- **Decoupled encoding.** The first approach simply encodes S and $\hat{\mathsf{S}}$ independently, after undergoing the transformation ξ_{Π}, into $\mathtt{edm}(\xi_{\Pi}(\mathsf{S}))$ and $\mathtt{edm}(\xi_{\Pi}(\hat{\mathsf{S}}))$, and stacks the two representations as they were two separate feature channels. As a result we obtain a $\mathsf{J}\times\mathsf{J}\times 2$ tensor representing the two skeletons.
- **Coupled encoding.** The second approach concatenates the two skeletons into a single matrix of points that we denote as $\mathsf{S}|\hat{\mathsf{S}} \in \mathbb{R}^{3\times 2\mathsf{J}}$ and uses the distance matrix $\mathtt{edm}(\xi_{\Pi}(\mathsf{S}|\hat{\mathsf{S}}))$ as encoding. This yields a $2\mathsf{J}\times 2\mathsf{J}$ matrix representation for the two skeletons.

The encoding of skeleton data that we have detailed for a pair of skeletons is actually applied to the entire sequence of skeletons. This adds also the temporal dimension to the representations mentioned above, yielding a $\mathsf{L}\times J\times J\times 2$ tensor if we opt for the independent encoding scheme, and a $\mathsf{L}\times 2\mathsf{J}\times 2\mathsf{J}$ if we apply the joint encoding instead, where L is the temporal window length.

3.2.2 3D CNNs over Distance Matrices

The application of CNNs to distance matrices has recently proven effective to tackle the problem of human pose regression from skeleton data [72]. Indeed, distance matrices exhibit rich local structures (up to permutations), which can be effectively learned by convolutional filters. In this work, we extend the ideas in [72] by considering *time* as an additional spatial dimension when performing convolution. This results in a 3D spatio-temporal convolution operator which allows our network to capture the temporal evolution of the local structures encoded by the EDMs.

Formally, given a tensor $\mathsf{Z} \in \mathbb{R}^{\mathsf{T}\times\mathsf{H}\times\mathsf{W}\times\mathsf{C}}$ and a convolutional filter $\mathsf{w} \in \mathbb{R}^{\mathsf{t}\times\mathsf{h}\times\mathsf{w}\times\mathsf{C}}$, we define 3D convolution $\star$ as

$$(\mathsf{w}\star\mathsf{Z})_{i,j,k} = \sum_{i'=1}^{\mathsf{t}}\sum_{j'=1}^{\mathsf{h}}\sum_{k'=1}^{\mathsf{w}}\sum_{c=1}^{\mathsf{C}} \mathsf{w}_{i',j',k',c}\mathsf{Z}_{i+i',j+j',k+k',c}\,,$$

where the first three dimensions of Z and w are interpreted as spatial dimensions, while C are the feature channels. In practice, 3D convolution can be used as a drop-in replacement for 2D convolution in most networks, providing us with great flexibility

when defining our architecture.

Building from these spatio-temporal convolution operators, we propose a network architecture inspired by the recent ResNeXt of Xie *et al.* [107] (see Fig. 3.2). In particular, we adapt the configuration employed in [107] for the CIFAR-10 experiments, replacing each convolution with a spatio-temporal convolution and performing the final global average pooling both over the spatial and time dimensions. To partially compensate for the increased number of parameters in our kernels compared to the ones in [107], we reduce the number of filters in each layer by a factor 2. Furthermore, differently from [107], we perform dropout on the inputs of the final fully-connected layer. For additional details refer to Appendix 3.2.4.

3.2.3 Network Training

Given a training set $\mathcal{T} \subset \mathcal{X} \times \mathcal{Y}$ we estimate an action recognition function $f_\theta : \mathcal{X} \to \mathcal{Y}$ by minimizing the regularized empirical risk

$$R(\theta; \mathcal{T}) = \frac{1}{|\mathcal{T}|} \sum_{(X,y)\in\mathcal{T}}^{\mathsf{N}} \ell(f_\theta(X), y) + \lambda\Omega(\theta)\,,$$

over Θ, where $\ell : \mathcal{Y} \times \mathcal{Y}$ is a loss function penalizing wrong predictions and $\Omega : \Theta \to \mathbb{R}$ is a regulariser. In our experiments, ℓ coincides with the standard log-loss and Ω with the ℓ_2 norm.

The minimization of the empirical risk is performed using stochastic gradient descent. Details about the hyperparameters of the optimizer are provided in the experimental section.

3.2.4 Network Architecture Details

Following the terminology employed in [107], our network is composed of three stages of 3 ResNeXt blocks each. Each block is obtained from the bottleneck template $\begin{bmatrix} 1 \times 1 \times 1, 32 \\ 3 \times 3 \times 3, 32 \\ 1 \times 1 \times 1, 128 \end{bmatrix}$, with cardinality $C = 16$ and pre-activation structure [32]. The network starts with a $3 \times 3 \times 3$ convolution with 32 filters. In the second and third block we halve the spatial resolution of the feature maps by applying a stride of $2 \times 2 \times 2$ on the first ResNeXt block. Correspondingly, we increase the depth of the feature maps by a factor 2. The third stage is followed by global 3D average pooling and a fully connected layer producing the final prediction.

3.3 Experiments

In the following Sec. 3.3.3 we study the performance of the proposed action recognition method by conducting an extensive evaluation on three benchmark datasets (see

Sec. 3.3.1), including the recent large-scale NTU RGB-D [83]. Furthermore, in Sec. 3.3.4 we perform an in-depth ablation study to evaluate the effects of learning the shuffle matrix under different sets of constraints. Additional details about our network architecture and training procedure are reported in Sec. 3.3.2.

3.3.1 Datasets

In this chapter we consider three benchmark datasets:

NTU RGB-D. The NTU RGB-D dataset [83] is, to the best of our knowledge, the largest-scale publicly available action recognition dataset. It contains over 56 thousand sequences, captured with multiple Kinect 2 sensors, of 40 actors performing 60 different actions in 17 different setups. Each action is repeated 2 times for each actor / setup pair and recorded from three different views at the same time. For each sequence, both RGB-D videos and 3D skeletons with 25 joints, automatically extracted using the Kinect 2 software, are made available. Depending on the action class, one or two actors can be present in the same sequence at the same time. Following the experimental protocol in [83], we consider both a cross-view and a cross-subject setting, splitting training and testing data on the basis of, respectively, the view from which the action is recorded or the actor performing it.

MSRC12 Gestures dataset. The MSRC12 Gesture dataset [19] contains 594 video sequences, captured with a Kinect, of 30 actors performing 12 actions. Each sequence contains several repetitions of the action of interest, for a total of 6244 action instances. As in [65, 103], we follow a cross-view evaluation protocol. Differently from NTU RGB-D, the skeleton detections provided with this dataset are computed using the Kinect v1 software, and contain skeletons with 20 joints.

SBU Interaction dataset. The SBU Interaction dataset [113] focuses solely on actions involving two interacting actors. It contains ≈ 300 sequences, subdivided in 21 sets, each containing one or two repetitions of each of 8 action classes, performed by a different pairing of subjects from a set of 7. Skeleton detections with 15 joints, extracted using the PrimeSense software, are provided together with the original RGB-D video sequences. In our experiments we follow the 5-fold cross-validation protocol also adopted in [113].

3.3.2 Implementation and Training Details

We train our DM-3DCNN network by stochastic gradient descent using the Adam [55] algorithm, with a batch size of 32 and parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. When considering the NTU RGB-D and MSRC12 datasets, we adopt the following training schedule: we start with a learning rate of 10^{-3}, reducing it by a factor 10 after 40 epochs and again after 60 epochs, training for a total of 80 epochs. The network parameters are initialized following the method from [30]. For SBU, given its considerably small size, we train by fine-tuning from the network trained on NTU

RGB-D: we initialize all convolutional filters from the values learned on NTU RGB-D, while learning the final fully connected classifier from scratch, and train for 300 iterations with an exponential learning rate decay from 10^{-4} to 10^{-5}. In all cases, the regularization factor (*i.e.* weight decay) is set to $\lambda = 5 \times 10^{-4}$. We implement our networks using the TensorFlow [1] framework and run our experiments on a single Nvidia GTX 1080 GPU[1].

Following the experimental setting in [83], in all datasets we down-sample the input sequences along the temporal dimension by subdividing it into 20 equally spaced sections and randomly selecting a frame from each. During training a different random sampling is considered each time a sequence is fed to the network, as we observed that this provides a useful regularizing effect. As mentioned in Sec.3.2.1, depending on the dataset, we consider up to two input skeletons at each time step. Since the datasets considered in our experimental evaluation do not define any explicit semantic about the ordering of the skeletons, during training we randomly select which one is interpreted as S and $\hat{S}$ each time a sequence is loaded. The same sampling procedure, both for sequence down-sampling and skeleton swapping, is also performed at test time, and all results in the following are reported as the average of the accuracy over 10 independent runs.

3.3.3 Comparison with State of the Art

Before performing our main evaluation, we conduct a set of preliminary cross-validation experiments on held-out training data from NTU RGB-D and SBU, in order to select which EDM encoding to use (see Sec.3.2.1). Interestingly, we observe that for NTU RGB-D the *decoupled* encoding exhibits the best performance, while for SBU the *coupled* encoding is favored. This is not surprising: the SBU dataset is mostly focused on interactions, thus the cross-skeleton distances encoded in the coupled EDM contain valuable information for our network. On the other hand, interaction classes are a strict minority in NTU RGB-D, thus making the more compact representation of the decoupled encoding a better fit for this dataset. Note that we do not need to choose which encoding to use in the MSRC12 case, as it only contains sequences with one skeleton.

NTU RGB-D. Compared to most previous datasets [19, 100, 113], NTU RGB-D contains 1-2 orders of magnitude more data, captured with the improved Kinect 2 sensor. Nonetheless, we note that the skeleton detections provided with the dataset still contain a substantial amount of noise, to the point that, even for a human observer, it can be hard to recognize the actions just by looking at the skeletons.

In a first set of experiments, we compare the performance of our method against recent LSTM-based and CNN-based approaches to human action recognition from skeleton data. In particular, we consider: the part-aware LSTM in [83]; the spatio-

[1]The source code is available at: `https://github.com/magnux/DMNN`

Method	C/Subject	C/View
LSTM-based methods		
Deep LSTM [83]	60.7 %	67.3 %
Part-aware LSTM [83]	62.9 %	70.3 %
ST-LSTM [64]	69.2 %	77.7 %
Multilayer LSTM [114]	64.9 %	79.7 %
CNN-based methods		
Liu *et al.* [65] single	73.5 %	84.0 %
Liu *et al.* [65] ensemble	80.0 %	87.2 %
DM-3DCNN single	**82.0** %	**89.5** %

Table 3.1: Results on the NTU RGB-D dataset.

temporal LSTM (ST-LSTM) in [83]; the multi-layer LSTM with geometric features in [114]; the ensemble of CNNs approach in [65]. For the method in [65] we report both the performance of the ensemble and that of the best single network in the ensemble. Finally, we also consider a plain 3-units LSTM baseline, as reported in [83].

Table 3.1 summarizes our results, highlight the advantages of our method when dealing with this large and challenging dataset. Compared to the best LSTM-based approach we observe an absolute increase in accuracy of $\approx 8\,\%$ in the cross-subject and $\approx 6\,\%$ in the cross-view setting. Similarly, we obtain a $\approx 2\,\%$ improvement over the CNN-based approach in [65] in both settings under exam. It is worth noting that the networks used in [65] have a considerably larger number of parameters than DM-3DCNN, *i.e.* $\approx 6 \times 10^7$ parameters for each network in the ensemble [59] and $\approx 6 \times 10^8$ overall, compared to 6.1×10^5 parameters in DM-3DCNN. This suggests that our EDM-based encoding is indeed more effective at representing sequences of skeletal data, compared to the image-based one adopted in [65], as our network is able to exploit it to obtain superior performance while using ≈ 1000 times less parameters.

MSRC12. In the next set of experiments, we focus our attention on the MSRC12 dataset. Here we consider two traditional approaches based on hand-crafted features, *i.e.* LC-KSVD [115] and Cov3DJ [41]; and three CNN-based approaches, *i.e.* Du *et al.* [16], Wang *et al.* [103] and Liu *et al.* [65]. The results are reported in Tab.3.2. It is clear that the sequences in this dataset are considerably less challenging than those in NTU RGB-D, as most methods under exam are able to achieve greater than 90 % accuracy. Among the non-ensemble models, DM-3DCNN obtains the highest accuracy, also surpassing the ensemble of CNNs in [103]. Interestingly, DM-3DCNN performs better than the single best CNN of [65], while being slightly surpassed by their ensemble, at the cost of employing ≈ 1000 times more parameters and, consequently, ≈ 1000 times more operations. Furthermore, differently from DM-3DCNN, the networks in [65] also exploits a vast amount of additional data, as they are pre-trained on the ILSVRC2012 data [81].

Method	Accuracy
Hand-crafted features	
LC-KSVD [115]	90.2 %
Cov3DJ [41]	91.7 %
CNN-based methods	
Du *et al.* [16]	84.5 %
Wang *et al.* [103]	93.1 %
Liu *et al.* [65] single	93.2 %
Liu *et al.* [65] ensemble	**96.6** %
DM-3DCNN single	95.8 %
DM-3DCNN ensemble	**96.6** %

Table 3.2: Results on the MSRC12 dataset.

Given the results of Liu *et al.* [65], both in NTU RGB-D (Tab. 3.1) and MSRC-12 (Tab. 3.2), it appears that an ensemble of networks, each trained on a different representation of the skeleton sequences, can significantly outperform the single models. This concept can easily be extended to our approach, *e.g.* by feeding a different permutation of the joints to each network in the ensemble. To explore this idea, we train five independent instances of DM-3DCNN, using the default joint ordering and four additional permutations[2], and average their output probabilities to obtain the final predictions. The results are reported in Tab.3.2 in the "DM-3DCNN ensemble of 5" row. Using our ensemble we are able to fill the performance gap with the method of Liu *et al.* [65], while still using ≈ 200 times fewer parameters. Note, however, that in our case the relative improvement going from the single model to the ensemble is smaller than in [65], further validating the effectiveness of our network.

Method	Accuracy
Other LSTM-based methods	
HBRNN [17]	80.4 %
Deep LSTM [116]	86.0 %
Co-occurrence LSTM [116]	90.4 %
ST-LSTM [64]	93.3 %
DM-3DCNN	**93.7** %

Table 3.3: Results on the SBU Interaction dataset.

SBU Interaction. In our final comparison with state of the art methods, we consider the interaction-focused SBU dataset. Table 3.3 reports the results obtained with

[2]The permutations are selected by separating the joints in six subsets corresponding to left/right arm, left/right leg, head and torso and randomly shuffling the subsets while keeping the order of the joints in each subset fixed.

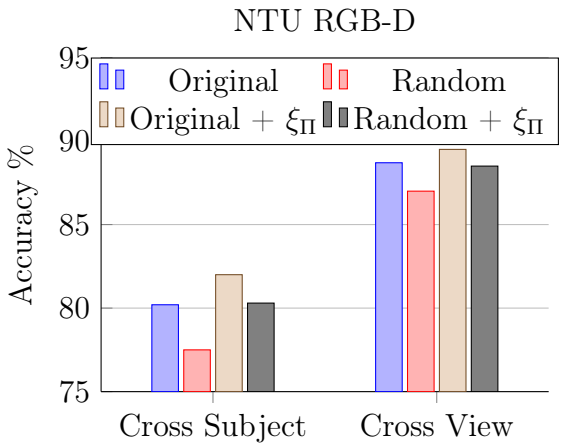

Figure 3.3: Ablation study on the NTU RGB-D dataset, comparing four different settings of DM-3DCNN: without joint transformation, using the original permutation (Original) or a random one (Random); with joint transformation, using the original permutation (Original + ξ_Π) or a random one (Random + ξ_Π).

DM-3DCNN and four RNN-based approaches: the hierarchical recurrent network of [17], the co-occurrence LSTM of [116] and the spatio-temporal LSTM of [64]. We also include in the comparison a plain LSTM model as reported in [116]. DM-3DCNN shows the highest accuracy, surpassing ST-LSTM by 0.4 %, a considerably lower advantage when compared to that obtained in the NTU RGB-D dataset. A possible explanation of this difference lies in the relative size of the two datasets. In fact, SBU contains about 200 times less sequences than NTU RGB-D, suggesting that our DM-3DCNN can be more effective at exploiting large-scale datasets compared to the LSTM-based approach in [116].

3.3.4 In-depth Analysis of DM-3DCNN

As noted in Sec.3.2.1, the joint permutation considered when forming the EDMs to be fed to the network can have a substantial impact on classification accuracy. To compensate for this, we propose to calculate the EDMs on a learned linear combination ξ_Π of the joints, which encompasses all possible permutations as special cases. In order to validate this approach, we perform an ablation study on the NTU RGB-D dataset and collect the results in Fig.3.3. In particular, we consider variations of DM-3DCNN trained with EDMs computed on the original joints (Original) or their transformation with ξ_Π (Original + ξ_Π). Learning ξ_Π produces an observable increase in accuracy, both in the cross-subject and cross-view settings.

While ξ_Π is in principle able to produce any permutation, we still expect the initial ordering of the input joints to play a role, as we are learning Π by minimizing an highly non-convex function. To test this effect, we re-run the experiments above, this time considering a different, randomly selected permutation of the joints instead of the original one given in the dataset. The resulting accuracies, visualized in

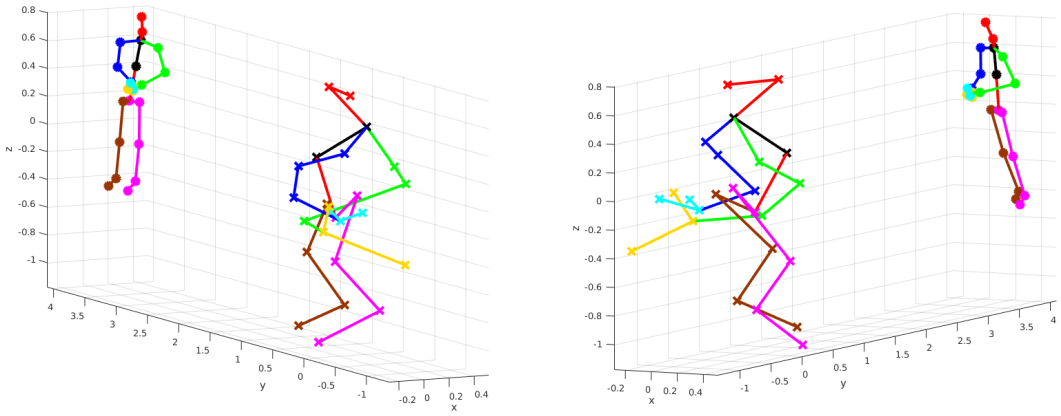

Figure 3.4: Original skeleton points from NTU RGB-D (circular markers) end transformed points using ξ_Π (cross markers). Two views are shown to better convey the 3D shape of the data. (Image best viewed on screen)

Fig. 3.3 as Random and Random + ξ_Π, are noticeably lower than those obtained with Original and Original + ξ_Π. This can be easily explained by observing that the original permutation is not random, but instead (loosely) follows the structure of the skeleton, keeping joints from distinct body parts close together and thus, intuitively, producing more informative local structures in the EDMs. Interestingly, however, when learning ξ_Π our network is able to overcome the disadvantage imposed by the random shuffling and reach the same accuracy obtained with the hand-picked permutation.

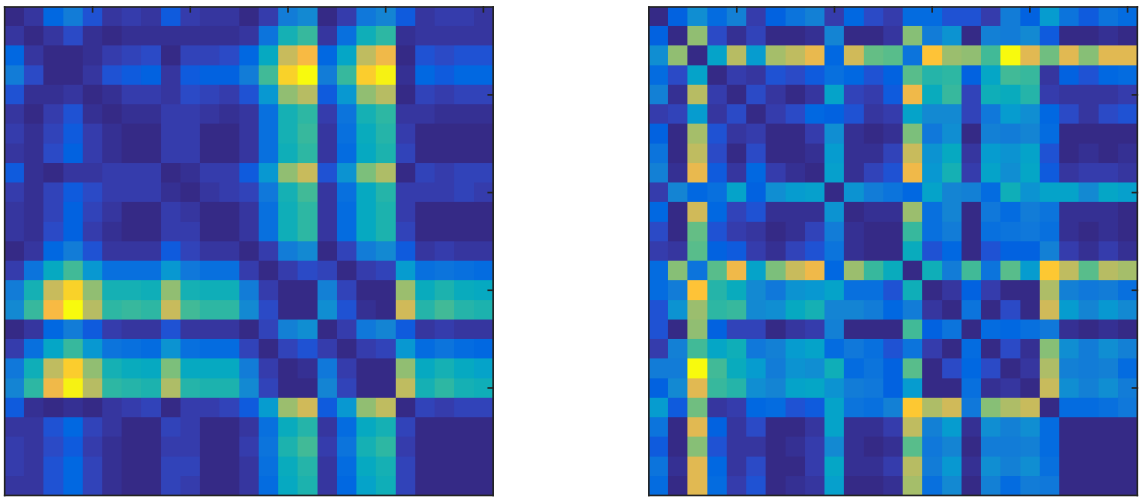

Figure 3.5: Distance matrices corresponding to the points in Fig.3.4. Left: original points. Right: transformed points.

Since we are not applying any constraint on the matrix Π, in general ξ_Π will transform the skeletons in complex, if useful, ways. In order to gain some more insights about the action of ξ_Π on the points, in Fig.3.4 we plot an example of original

and transformed skeletons from NTU RGB-D. Two main phenomena are immediately apparent: i) the transformed points are shifted towards the origin of the coordinates system; ii) the limbs appear to be stretched, while the torso becomes comparatively more compressed. (ii) can be explained as the network giving more importance to the joints in the arm and legs, which, intuitively, can be more discriminative for the task of recognizing actions. For another perspective on the effect of ξ_{Π}, in Fig.3.5 we plot the distance matrices corresponding to the points in Fig.3.4. Here, the patterns visible in the EDM of the transformed points appear to be more contrasted than those in the original one, with stronger edges and corners.

3.4 Chapter summary

In this chapter, we are interested in human action recognition from sequences of 3D skeletal data. To this end, we combine a 3D Convolutional Neural Network with body representations based on Euclidean Distance Matrices (EDMs). These have recently been shown to be very effective in capturing the geometric structure of the human pose. However, an inherent limitation of EDMs is that they are defined up to a permutation of the skeletal joints, i.e., a random rearrangement of the joints leads to many different representations. To address this problem, we present a novel architecture that simultaneously and in an end-to-end manner, learns an optimal transformation of the joints while optimizing the other parameters of the convolutional network. The proposed approach achieves top results on 3 benchmarks, including the recent NTU RGB -D dataset, where we outperform previous LSTM-based methods by more than 10 percentage points and also outperform other CNN-based methods while using almost 1000 times fewer parameters.

Chapter 4

3D motion prediction

Recent advances in motion capture technologies, combined with large scale datasets such as Human3.6M [44], have spurred the interest for new deep learning algorithms able to forecast 3D human motion from past skeleton data. State-of-the-art approaches formulate the problem as a sequence generation task, and solve it using Recurrent Neural Networks (RNNs) [20, 45], sequence-to-sequence models [68] or encoder-decoder predictors [10, 26].

While they show promising results, these works suffer from three fundamental limitations. First, they address a simplified version of the problem in which global body positioning is disregarded, either by parameterizing 3D body joints using position agnostic angles [20, 45, 68] or body centered coordinates [10]. Second, current methods require additional supervision in terms of action labels during training and inference, which limits their generalization capabilities. And third, most approaches aim to minimize the L2 distance between the ground truth and generated motions. The L2 distance, however, is known to be an inaccurate metric, specially to compare long motion sequences.

In particular, the use of this metric to train a deep network favors motion predictions that converge to a static mean pose. Even though this issue has been raised in [26, 45] and is partially solved during training using other metrics (e.g. geodesic loss), the L2 distance is still being used as a common practice when benchmarking different methodologies. To our understanding, this practice compromises the progress in this field.

In this chapter we tackle all three issues. Specifically, we design a novel GAN architecture that is conditioned on past observations and is able to jointly forecast non-rigid body pose and its absolute position in space. For this purpose we represent the observed skeleton poses (expressed in the camera reference frame) using a spatio-temporal tensor and formulate the prediction problem as an inpainting task, in which a part of the spatio-temporal volume needs to be regressed. A GAN architecture consisting of a fully convolutional generator specially designed to preserve the temporal coherence, and three independent discriminators that enforce

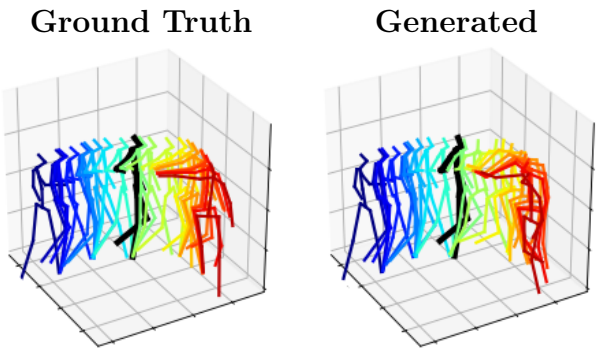

Figure 4.1: **Example result.** Our approach is the first in generating full body pose, including skeleton motion and absolute position in space. The predicted sequence starts from the skeleton marked in black. Note that the generated motion is somewhat different but semantically indistinguishable from the ground truth.

anthropomorphism of the generated skeleton and its motion, makes it possible to render highly realistic long-term predictions (of 2 seconds or more). Interestingly, the L2 loss is only enforced over the reconstructed past observations and not over the hypothesized future predictions. This way, the generation of future frames is fully controlled by the discriminators. In fact, our model does not require ground truth annotations of the generated frames nor explicit information about the action being performed.

We also introduce a novel metric for estimating the similarity between the generated and the ground truth sequences. Instead of seeking to get a perfect match for all joints across all frames (as done when using the L2 distance), the metric we propose aims to estimate the similarity between distributions over the human motion manifold.

In the experimental section we show that our approach, besides yielding full body pose, orientation and position, is also robust to challenging artifacts including missing frames and occluded joints on the past skeleton observations. Fig. 4.1 shows an example result of our approach.

4.1 Related work

Early approaches to modeling human motion. The inherent high-dimensionality and non-linearity of human body motion makes building statistical models a major challenge. Traditional approaches have addressed this task with latent variable models such as Hidden Markov Models [8], bilinear spatio-temporal basis models [3], Gaussian processes [94, 95, 99], linear dynamic systems [78], and random forests [62]. However, these models are based on relatively simple dynamics that apply only to short-term predictions and are highly specialized to specific types of motions. Conditional Restricted Boltzmann Machines [87, 90, 91] better capture the non-linearities

of human motion, although these systems are more complex to train and require random sampling for inference.

Deep Learning for motion prediction. Most recent Deep Learning approaches build on the problem formulation proposed by [91], where input motion sequences are represented by 3D body joint angles in a kinematic tree. Motivated by their success in machine translation problems [12, 51, 88], RNNs are then used to predict motion sequences of body joint angles. Fragkiadaki *et al.* [20], for example, introduces an Encoder-Recurrent-Decoder (ERD) in combination with a Long Short-Term Memory (LSTM) for this purpose. Jain *et al.* [45] introduced structural RNNs, an approach that exploits the structural hierarchy of human body parts. Martinez *et al.* [68] developed a sequence-to-sequence architecture with a residual connection that incorporates information about action classes via one-hot vectors. While these approaches work well for the specific motion for which they were trained, they do not generalize to other actions. More importantly, these models are only effective for the short-term and medium-term ranges and are often outperformed by a simple zero-velocity baseline model. This is due in part to the use of the L2 metric for both training and evaluation. More recent approaches have different strategies to address this issue.

Li *et al.* [63], propose a model with an auto-regressive CNN generator and combine the L2 loss with an adversarial loss. Gui *et al.* [26] completely eliminate the L2 loss during training and use a ERD model with a combination of an adversarial and geodesic losses. However, these works still perform evaluation using the L2 metric, which does not capture the semantics of motion, especially for long-term predictions. Furthermore, since motion is parameterized in terms of joint angles, the rotation and translation of the body in space are not estimated.

Sequence completion and image inpainting. Completing missing data within a sequence has traditionally been approached using low-rank matrix factorization [2, 105]. Deep Learning approaches have also been used for this purpose, e.g., by RNNs [60, 67]. However, these works are not designed for future prediction.

Image inpainting is a very related problem. In the era of Deep Learning, Denoising AEs [5] and Variational AEs [56] have become popular for denoising and completing missing data and for image inpainting. However, these basic systems cannot handle large portions of missing structured data. The state of the art has been significantly advanced by GANs conditioned on partial or corrupted images [79, 111, 112]. As we will see, our approach draws inspiration from this idea.

Metrics for evaluating human motion prediction. The fact that L2 is not suitable for measuring similarity between human motion sequences has been recently discussed and addressed in several papers. Coskun *et al.* [14] uses deep metric learning and a contrastive loss to learn the metric directly from the data. This is arguably the best way to semantically compare motion sequences. However, the problem with this approach is that once the metric is trained, it is difficult to apply it to other models because the metric was trained with a specific setting. An alternative for sequences

that does not require training would be to use frequency-based metrics. In [24], a metric based on the power spectrum is proposed. This metric shows interesting properties and seems to be suitable for comparing actions with periodic movements such as walking. The main drawback of this approach is that it compares sequence by sequence, which in our view is not desirable. We would like to compare distributions of sequences instead.

For image generation, there is recent work that proposes to measure the fitness of models based on properties of the distribution of the generated data. The Inception Score [82] measures the entropy of the label outputs of the inception network on the generated images. The Frechet Inception Distance [35] (FID) instead proposes to fit two multivariate Gaussians to the activations of the inception network for the real and generated samples, respectively. Then, the FID is determined by measuring the distance between the Gaussian models. Following this work, we propose new metrics based on the distribution of the frequencies of the generated samples. These metrics have the advantage of being easy to implement and replicate, and they measure the overall fitness of a model by considering a distribution of sequences.

4.2 Problem Formulation

We represent the human pose with a J-joint skeleton, where each joint consists of its 3D Cartesian coordinates expressed in the camera reference frame. Rotational and translational transformations are inherently encoded in these coordinates. A motion sequence is a concatenation of F skeletons, which we will represent by a tensor $\mathbf{S} \in \mathbb{R}^{\mathrm{F}\times\mathrm{J}\times 3}$. Let us define an occlusion mask as a binary matrix $\mathbf{M} \in \mathbb{B}^{\mathrm{F}\times\mathrm{J}\times 3}$ that determines the part of the sequence that is not observed, and which is applied to the sequence by performing the element-wise dot product $\mathbf{S} \circ \mathbf{M} \equiv \mathbf{S}^{\mathrm{m}}$. Our goal is then to estimate the 3D coordinates of the masked joints. Note that we can define different subproblems depending on the pattern used to generate the occlusion mask $\mathbf{M}$. For example, if we mask the last frames of the sequence, we can represent a prediction problem. If we mask certain intermediate joints instead, we represent random joint occlusions, structured occlusions, or missing images. Our model can solve any combination of these sub-problems.

4.3 Model

4.3.1 STMI-GAN architecture

Fig. 4.2 shows an overview of the GAN we propose, in which we pose the human motion prediction problem as an inpainting task in the spatio-temporal domain. We denote our network as STMI-GAN (Spatio-Temporal Motion Inpainting GAN). We next describe its main components.

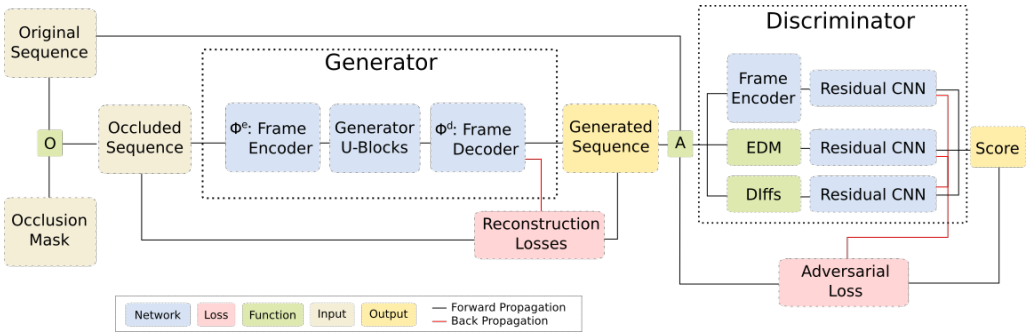

Figure 4.2: Overview of our architecture. An input masked sequence of 3D joint coordinates is fed into a fully convolutional and time preserving generator. The output sequence is controlled by a number of geometric constraints, including losses applied to the generator output and adversarial losses of three independent discriminators.

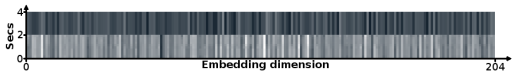

Figure 4.3: **Motion embedding.** A motion sequence of the H3.6M dataset passed through the frame encoder. The sequence is occluded from the half onwards, with the goal of motion prediction.

Generator. The rationale for the design holds in that convolutional GANs have been successful in image inpainting problems, which is similar to ours. A masked human motion sequence $\mathbf{S}^{\mathrm{m}}$, however, cannot be directly processed by a convolutional network because the dimension corresponding to the joints (J) does not have a spatial meaning, in contrast to the temporal (F) and Cartesian coordinates dimensions. That is, neighboring joints along this dimension do not correspond to neighboring joints in 3D space[1]. To alleviate this lack of spatial continuity problem, the generator is placed in-between a frame autoencoder, namely a frame encoder Φ^e and a frame decoder Φ^d, which are symmetrical networks. The frame encoder, operates over the J-dimension and projects each frame $\mathbf{S}^{\mathrm{m}}_{i::} \in \mathbb{R}^{\mathrm{J}\times 3}$ of the sequence to a one-dimensional vector $\mathbf{S}^{\mathrm{e}}_{i::} = \Phi^e(\mathbf{S}^{\mathrm{m}}_{i::}) \in \mathbb{R}^{\mathrm{H}\times 1}$, where H is the dimension of the space for the pose embedding[2]. To project each frame, the encoder does not use information from neighboring frames, being thus time invariant. We denote the encoded sequence as $\mathbf{S}^e \in \mathbb{R}^{\mathrm{F}\times\mathrm{H}\times 1}$.

As we can observe in Fig.4.3, the frame encoder learns to represent the sequence as a 2D matrix, in an arbitrary space. Although the learnt space has no clear interpretation, we can observe from the sample that it retains certain properties, such as the temporal ordering, and a constant "zero" value for the occluded frames. This encoded sequence is then passed through a series of generator blocks Φ^g, which produces a new sequence $\mathbf{S}^g \in \mathbb{R}^{\mathrm{F}\times\mathrm{H}\times 1}$ in the embedded space. The blocks of the generators are CNNs that process the sequence in both temporal and spatial dimensions. Further details are explained in Section 4.5. Finally the decoder network Φ^d

[1]For instance, the joint #0 is normally the hip, and its neighbors in the body graph are the joints #1 (left hip), #5 (right hip) and #9 (spine).

[2]$\mathbf{S}_{i::}$ denotes the i-th element of $\mathbf{S}$ along the first axis.

maps back the the transformed sequence $\mathbf{S}^g$ to the output sequence in the original shape $\mathbf{S}^{out} \in \mathbb{R}^{\mathrm{F}\times\mathrm{J}\times 3}$.

Discriminator. To capture the complexity of the human motion distribution we split the discriminator into three branches, capturing different aspects of the generated sequence. Each discriminator is a Residual CNN classifier that serves as a feature extractor. These features are linearly combined to obtain a probability of the sequence of being real. We next describe the main blocks of our model. Details of the underlying architectures are detailed later in Sect. 4.5.

Base discriminator. The same architecture of the frame encoder Φ^e, with independent parameters, is used to process the generated sequence $\mathbf{S}^{out}$. The reason to reuse such architecture is that we want a discriminator to be applied directly on the non Euclidean representation used by the CNN blocks of the generator, in order to boost its performance.

EDM discriminator. We introduce a geometric discriminator that evaluates the anthropomorphism of the generated sequence $\mathbf{S}^{out}$ via the analysis of its Euclidean Distance Matrix, computed as $\mathrm{EDM}(\mathbf{S}^{out}) \equiv \mathbf{D} \in \mathbb{R}^{\mathrm{J}\times\mathrm{J}}$, where $\mathbf{D}_{ij}$ is the Euclidean distance between joints i and j of $\mathbf{S}^{out}$. This is a rotation and translation invariant representation [34, 73], allowing to focus the attention of the discriminator into the shape of the skeleton.

Motion discriminator. the Base discriminator sees the sequences as absolute coordinates of joints in the space, and the EDM discriminator sees them as relative coordinates w.r.t the other joints. But these discriminators are missing the joint correlations between the absolute motion and their relative (articulated) counterpart. Thus, we consider a third discriminator that operates over the concatenation of both, the temporal differences of absolute coordinates $\|\mathbf{S}^{out}(t) - \mathbf{S}^{out}(t-1)\|_1$ and the temporal differences of EDM representations $\|\mathrm{EDM}(\mathbf{S}^{out}(t)) - \mathrm{EDM}(\mathbf{S}^{out}(t-1))\|_1$, where $\mathbf{S}^{out}(t)$ indicates generated the sequence at time t.

4.3.2 Losses

To train our network we use two main losses: 1) The reconstruction losses, that encourage the generator to preserve the information from the visible part of the sequence; 2) The GAN loss, which guides the generator to inpaint the sequences by learning and reproducing the motion in the dataset. For all the following formulae, let $\mathbf{S}$ be the input motion sequence, $\mathbf{M}$ the occlusion mask, $\mathbf{S}^{out}$ the generated sequence, F number of frames and J number of joints.

Reconstruction Loss. Our default reconstruction loss computes the L2 norm over

the generated sequence w.r.t. the visible portion of the ground truth.

$$\mathcal{L}_{rec} = \|(\mathbf{S}^{out} \circ \mathbf{M}) - (\mathbf{S} \circ \mathbf{M})\|_2 \tag{4.1}$$

This loss is only applied over the visible part of the original and generated sequences. By doing this, we penalize deviations from the visible part of the sequences, while avoiding to penalize the different possible completions of the sequence.

Limb Distances Loss. [34] showed that most common actions can be recognized from just the relative distance between the extremities, *i.e.* hands, feet and head. We therefore add a loss that explicitly enforces the correct distance between these semantically important joints.
Since this loss looks at the relative distance, instead of the absolute position, it provides different gradients to the reconstruction loss, and encourages the network to learn a more precise location for the limbs. Formally, if we denote by $\mathcal{E} = \{i, j\}$ the set of limb pairs , the loss $\mathcal{L}_{limb}$ is computed as:

$$\sum_{f=1}^{F} \sum_{\{i,j\} \in \mathcal{E}} \| \|\mathbf{S}^m_{fi:} - \mathbf{S}^m_{fj:}\|_2 - \|\mathbf{S}^{m,out}_{fi:} - \mathbf{S}^{m,out}_{fj:}\|_2 \|_2 \tag{4.2}$$

where $\mathbf{S}^{m,out} = \mathbf{S}^{out} \circ \mathbf{M}$, denoting again that this loss is only computed over the visible part of the original sequence.

Bone Length Loss. We also enforce constant bone length of the whole generated sequence. Its main goal is to discourage the generator to explore solutions where the skeleton is not well formed. If we denote by $\overline{\mathcal{B}} = \{\bar{l}_1, \ldots, \bar{l}_B\}$ the mean length of the B body bones computed over the visible part of the sequence, and by $\mathcal{B}_f = \{l_{f1}, \ldots, l_{f_B}\}$ the length of the bones at frame f, this loss is computed as

$$\mathcal{L}_{bone} = \sum_{f=1}^{F} \sum_{b=1}^{B} \|\bar{l}_b - l_{fb}\|_2 \tag{4.3}$$

Regularized Adversarial Loss. Our adversarial loss is based on the original GAN loss [22], with the R1 regularization described in [69]. Let G_θ be the generator network, parameterized by the variable θ, D_ψ be the discriminator network, parameterized by the variable ψ, and $\mathbb{P}_o$ the distribution of input motion sequences. We can then write the Discriminator Loss as:

$$\begin{aligned} \mathcal{L}_D =& \mathbb{E}_{\mathbf{S}^{out} \sim \mathbb{P}_o} \left[\log(1 - D_\psi(G_\theta(\mathbf{S} \circ \mathbf{M})))\right] \\ &+ \mathbb{E}_{\mathbf{S} \sim \mathbb{P}_o} \left[\log(D_\psi(x))\right] + \frac{\gamma}{2} \mathbb{E}_{\mathbf{S} \sim \mathbb{P}_o} \left[\|\nabla D_\psi(x)\|^2\right] \end{aligned} \tag{4.4}$$

The Generator Loss is as follows:

$$\mathcal{L}_G(\theta, \psi) = \mathbb{E}_{\mathbf{S}^{out} \sim \mathbb{P}_o} \left[\log(D_\psi(G_\theta(\mathbf{S} \circ \mathbf{M})))\right] \tag{4.5}$$

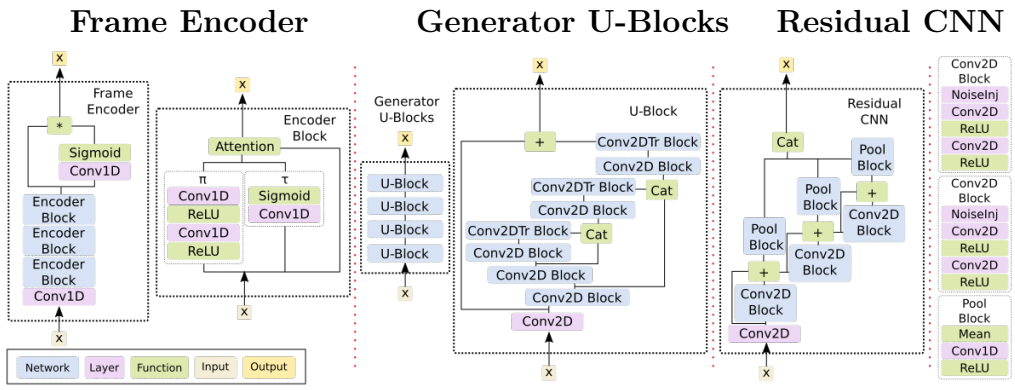

Figure 4.4: **Details of the architecture.** From left to right: Frame Encoder; Generator U-Blocks; Residual CNN. In each case we plot the general view of the block (left) and the fine detail of the structural elements (right). The Attention function is defined as: $att(x, \pi, \tau) = \pi\tau + x(1 - \tau)$. The Conv2DTr denotes Convolution 2D Transpose, also called deconvolution.

Full Loss. The full loss $\mathcal{L}$ consists of a linear combination of all previous partial losses:

$$\mathcal{L} = \lambda_r \mathcal{L}_{rec} + \lambda_l \mathcal{L}_{limb} + \lambda_b \mathcal{L}_{bone} + \lambda_D \mathcal{L}_D + \lambda_G \mathcal{L}_G \tag{4.6}$$

where λ_r, λ_l, λ_b, λ_D and λ_G are the hyper-parameters that control the relative importance of every loss term. Finally, we can define the following minimax problem:

$$G^\star = \arg\min_G \max_{D \in \mathcal{D}} \mathcal{L} \ , \tag{4.7}$$

where $G^\star$ draws samples from the data distribution.

4.4 Metrics for motion prediction

Our goal then is to analyze the distribution generated by our model, for which we propose metrics that are analogue to the Inception Score [82] and the Frechet Inception Distance [35]. With this in mind we propose the following metrics:

PSEnt measures the entropy in the power spectrum of a dataset. This metric can give us a rough estimate of the fitness of the model. First we compute the power spectrum of the dataset independently per each joint and axis. Each joint-axis combination is considered a distinct feature of a sequence. Formally, the power spectrum of a feature is computed as: $PS(s_f) = \|FFT(s_f)\|^2$. We can then compute the Power Spectrum Entropy over a dataset:

$$PSEnt(D) = \frac{1}{S} \sum_{s \in D} \frac{1}{F} \sum_{f=1}^{F} - \sum_{e=1}^{E} \|PS(s_f)\| * log(\|PS(s_f)\|) \tag{4.8}$$

where D is a dataset, s is a sequence, f is a feature, and e is frequency.

A common characteristic of the generative models trained with the L2 loss is that they have the tendency to regress to the mean, lowering the entropy of the generated sequences. An entropy value lower than the expected is a telltale sign of a biased model, whereas a higher entropy value points to a rather noisy and maybe inaccurate or unstable model.

PSKL measures the distance (in terms of the KL divergence) between the ground truth and generated datasets:

$$PSKL(C, D) = \sum_{e=1}^{E} \|PS(C)\| * log(\frac{\|PS(C)\|}{\|PS(D)\|}) \tag{4.9}$$

where C and D are datasets, s is a sequence, f is a feature, and e is frequency. The KL divergence is asymmetric, so we compute both directions PSKL(GT, Gen) and PSKL(Gen, GT) to have the complete picture of the divergence. If both directions are roughly equal, it would mean that the datasets are different but equally complex. On the other hand if the divergences are considerably different, it would mean that one of the datasets has a biased distribution.

L2 based metrics. We also measure the distance between the ground truth sequence s_{gt} and the generated sequence s_{gen}, considering each joint (j) as an independent feature vector.

$$L2(s_{gt}, s_{gen}) = \frac{1}{J} \sum_{f=1}^{J} \|s_{gt} - s_{gen}\|_2 \tag{4.10}$$

In [20, 68] s is represented in Euler angles, but in our work s is in coordinates, making it readable in millimeters. The mean is used to obtain a measure for the complete dataset.

4.5 Implementation Details

We next describe the blocks of our architecture.[3]

Frame Autoencoder. The Frame Encoder (Fig. 4.4-left) is a fully connected network with identical sequential blocks. Each block contains two consecutive fully connected layers, and an attention mechanism [4] at the end. The fully connected layers can also be seen as 1D convolutions with kernel size 1 along the time dimension. The attention is performed by applying a mask over the output of the block. The mask is a linear transformation of the input to the block, followed by a sigmoid activation. After the blocks, a final linear transformation and an attention are applied. The architecture is similar to a VAE [56], but without the Gaussian constrains over the output of the encoder. The decoder network is a symmetrical network to

[3]Code Available at: https://github.com/magnux/MotionGAN

the encoder, with the same number of blocks and layers, but independent parameters.

Generator U-Blocks. The goal of the generator is to produce an output that should be indistinguishable from an unmasked input, while preserving the shape of the sequence. To accomplish this, we use U-blocks [80] (see Fig. 4.4-center) with convolutions that halve the spatial resolution of the input in each layer, until it reaches a small representation. Then a transposed convolution is used to double the resolution until it reaches again the same dimensions as the input. A key component in this architecture are the skip connections that connect the output of the convolutional layers to the input of the deconvolutional layers. We can think of this architecture as an iterative refinement, in which the output of a block is refined by the next block to produce a better final output. Following [52], we also incorporate a noise injection layer into our convolutional blocks which makes the model prediction non-deterministic and enriches it.

Residual CNN. We designed the architecture of our discriminator inspired on ResNet [31] and DenseNet [39]. Our network (Fig. 4.2) branches in three discriminators, and each discriminator has a classifier, with the same architecture but separate parameters. Their architecture (Fig. 4.4-right) consists of several consecutive blocks with two convolutional layers and additive residual connections, similar to ResNet. The outputs of each block are also transformed and then concatenated into a tensor. The final output is the concatenation of the outputs of all blocks. This output is finally passed onto a fully connected network that assigns a score.

Spatial Alignment. Since we are working over an absolute coordinate system, the sequences have a wide range of values (from mm to m). To improve the robustness of the generator we subtract the position of the hip joint in the first frame to all the joints in the sequence. Then the skeleton is rotated to always face in the same direction. The alignment is performed by a custom layer in the network before the frame encoder, and is reversed just after the frame decoder.

4.6 Training Details

Alternating Training. We used the standard GAN alternating training: one batch to train the discriminator without updating the generator. The next batch to train the generator without updating the discriminator.

Mask Generation. The occlusion masks for training are randomly generated for each batch. We have the following rules for this generation:

- The binary masks are drawn from a binomial distribution with the shapes that correspond to future prediction, random joints occlusions, structured occlusions or missing frames.
- Even batches are trained for future prediction, odd batches are trained in one of the other mask modes randomly selected.

- For each batch, the occlusion probability p is randomly drawn from the uniform distribution $p \in [0.25, 0.75]$.
- Some datasets contain real occlusion masks M_R which are inherent in the data. We combine these masks with our generated masks by multiplying them M, $M = M \circ M_R$.

4.6.1 Data Augmentation

In order to train a robust network, especially on small datasets, we need to augment our data. We perform a number of transformations on the sequences, both in the temporal dimension (random cropping and random sub-sampling) and in the spatial dimension (height jittering and side flipping)

Random Cropping. If the dataset contains sequences with an average duration of over 4 seconds, for each batch we select a window of f consecutive frames starting at a random point in the sequence. In practice, the length of the window corresponds to 4 seconds. This algorithm is useful when the dataset contains very long sequences,e.g. Human3.6M dataset.

Random Sub-Sampling. Similar to other approaches [68], we resample the data at a lower sampling rate. However, our sub-sampling is not deterministic; instead, we sample randomly within fixed windows. More specifically: The sequence S is divided into n identical windows, a sample is drawn from each window at random with uniform probability. For example: the original dataset has a frame rate of 50hz, the sequence length is 200, we split the sequence into $n = 20$ windows and draw a random sample from each window. The resulting sequence is 20 frames long and its sampling rate is approximately equal to 5 Hz.

Height Jittering. In small datasets, there are a limited number of subjects, leading to significant interpersonal differences in shape and motion. To make the network robust to different shapes, we augment the sequences by perturbing the height of the skeleton in the training sequences. Formally, we draw a factor $c = U(0.7, 1.3)$, then multiply the z-axis (or spatial dimension of height) by this factor.

Side Flipping. Another important source of interpersonal differences in motion is the handedness of the subject. In small datasets, it is common for most (or all) subjects to be right-handed. For this reason, we randomly flip the sides of the skeleton with equal probability. We do this by inverting the y-axis (or one of the horizontal dimensions).

4.6.2 Model Parameters

Our full model contains 4,812,716 parameters, split into 2,338,268 for the generator and 2,474,448 for the discriminator. The generator has a total of 228 layers and the

discriminator has a total of 173 layers. We should note that most layers have no parameters, but perform functions such as activation or arithmetic operations.

4.6.3 Training Meta-Parameters

The model was trained using the Nadam [15] optimizer, with an initial learning rate of 1e-3. The learning rate was stair-stepped by a factor of 10. LR was decreased twice, at 1/3 and 2/3 of the scheduled epochs. The batch size was 128. The number of scheduled epochs was 128, and each epoch contained 256 augmented batches.

4.7 Experiments

Datasets. In the experimental section we mainly use the Human3.6M [44] dataset. We follow the same split used in [20, 68].

4.7.1 Motion Prediction

In this section we compare our approach to [68], one of the baseline works in the state of the art. From this work, we are using the residual supervised model, which is a sequence-to-sequence model with residual connections and uses the labels as part of the inputs. Since our model is based on Cartesian coordinates, we compute the joint angle representation equivalent to that used by the Residual supervised (Res.sup.) [68] model. This transformation allows us for a consistent comparison between the two.

We next run an ablation study, using always the same generator network, but training it with different discriminator networks. As we argued in previous sections, we aim to capture the distribution of the Ground Truth (GT) data, and each component in the architecture was designed with this purpose. Our hypothesis is that an adversarial loss is better than L2 and geometric losses to train a generative network. Even more, we argue that the complexity of the discriminator network should be correlated with a better result in the generated sequences.

Our tested models are: NoGAN: generator network trained with the reconstruction losses over the whole sequence, and no adversarial loss. Base disc: is the same network, but using the encoder discriminator as loss for the generated part. +EDM disc: the network is trained using both the base and the EDM discriminators. +Motion disc: the network is trained using the base and motion discriminator. STMI-GAN: is the full network, trained with all joint discriminators.

Every discriminator seems to be adding information to the generated distribution. We can observe this qualitatively, and we can confirm it with the proposed metrics. **Entropy Analysis and KL distance Analysis**. We should first note that the PSEnt in the original distribution is almost identical in every one second window, at

Model	PSEnt	PSKL(GT,Gen)	PSKL(Gen,GT)
0 to 1 second			
Org.Data. (Val vs Train)	0.67990	0.00590	0.00572
Res.sup. [68]	0.37492	0.03293	0.04524
NoGAN	0.44363	0.03729	0.05040
Base disc	0.73626	0.01198	0.01149
+EDM disc	0.57045	0.01801	0.02131
+Motion disc	0.72617	0.01220	0.01141
STMI-GAN	**0.68099**	**0.01090**	**0.01125**
1 to 2 seconds			
Org.Data. (Val vs Train)	0.67749	0.00628	0.00611
Res.sup. [68]	0.20975	0.10188	0.17004
NoGAN	0.27969	0.07989	0.13743
Base disc	0.60450	0.01559	0.01766
+EDM disc	0.49198	0.02546	0.03315
+Motion disc	0.72963	0.01223	0.01129
STMI-GAN	**0.68328**	**0.01041**	**0.01010**
2 to 3 seconds			
Org.Data. (Val vs Train)	0.67391	0.00640	0.00620
Res.sup. [68]	0.12752	0.17402	0.33566
NoGAN	0.34717	0.06099	0.09562
Base disc	0.60804	0.01396	0.01611
+EDM disc	0.45627	0.03368	0.04596
+Motion disc	0.72368	0.01312	0.01201
STMI-GAN	**0.71778**	**0.01306**	**0.01213**
3 to 4 seconds			
Org.Data. (Val vs Train)	0.67891	0.00590	0.00566
Res.sup. [68]	0.09333	0.18692	0.37605
NoGAN	0.26750	0.08672	0.15567
Base disc	0.50224	0.02646	0.03460
+EDM disc	0.41653	0.04516	0.06541
+Motion disc	0.76111	0.01436	0.01275
STMI-GAN	**0.70985**	**0.01108**	**0.01024**
0 to 4 seconds			
Org.Data. (Val vs Train)	1.65373	0.01225	0.01227
Res.sup. [68]	0.85732	0.13320	0.15644
NoGAN	1.07468	0.10245	0.12508
Base disc	1.58270	0.02197	0.02274
+EDM disc	1.22901	0.08416	0.09894
+Motion disc	1.77806	0.02434	0.02270
STMI-GAN	**1.69147**	**0.01888**	**0.01801**

Table 4.1: **Ablation Study.** Power Spectrum based metrics for different configurations of our model.

approximately 0.678. This number represents the entropy of the uniform distribution, which means that the short term frequencies are fairly uniform. The PSEnt raises to 1.65, when we consider a 4 second time window. Such raise means that the long term motion has a biased, and more complex frequency distribution, which is not uniform but denser in some parts of the spectrum.

We can observe in Tab.4.1 that the Res.sup. [68] and NoGAN baselines decay in entropy as the seconds pass. Also we see that the KL divergence grows rapidly for the baselines and is around an order of magnitude higher than any of the GAN models.

The GAN models all seem to have a good behavior, with PSEnt values close to the GT distribution. The Base disc model is already stable, but has some decay in entropy towards the end of the sequence. The Motion disc model seems to be pretty stable but it consistently overshoots on the entropy. This may be interpreted as the model overemphazising in moving. The EDM disc model seems to be harming at a first glance, since it considerably lowers the PSEnt, but the main point of adding this discriminator is to prevent unexpected poses from happening. It is a regularizer by design.

When we combine the three discriminators in the STMI-GAN model, the network approximates closely to the expected distribution. The STMI-GAN is stable both in PSEnt and PSKL and its performance does not decay as time passes. Indeed, it has a PSEnt close to the GT and a low PSKL, meaning that it is not only producing the same amount of motion but also the same kind of motion. We should note that the Human3.6M dataset has a considerable difference between the validation and train splits when following the standard protocol [68]. The PSKL between the validation and train splits for the whole sequence is around 0.012, almost symmetrical, and the PSKL between the generated distribution of STMI-GAN and validation is around 0.018, also symmetrical. This means that the distribution produced by the GAN is almost as close as the training and validation sets are.

L2 metric experiments. To demonstrate the point that L2 metric is not correlated with a realistic generation, we use a subset of the 120 test sequences of [20, 68], concretely the #8, #26, #27, #88. Fig. 4.5 shows the results of our approach and Res.sup. [68] on these sequences. Note that Res.sup. has a tendency to converge to the same pose (see the red skeleton in the center column), which is very close to the mean pose in the dataset. There is also the tendency to produce very small motions (see black and red at the bottom center frame). Indeed, [68] shows that the zero velocity baseline is often better than their model, specially for the class 'discussion' which has high uncertainty (see last row in Fig.4.5).

When computing the L2 metric over angles as in [68] we obtain the following results: L2 Res.sup. $\rightarrow$ (0.69, 0.36, 0.64, 0.25); L2 STMI-GAN $\rightarrow$ (1.09, 0.74, 1.33, 0.96). Despite the baseline model has a lower L2, the sequences generated by our approach seem more diverse and realistic. These effects derive from the objectives

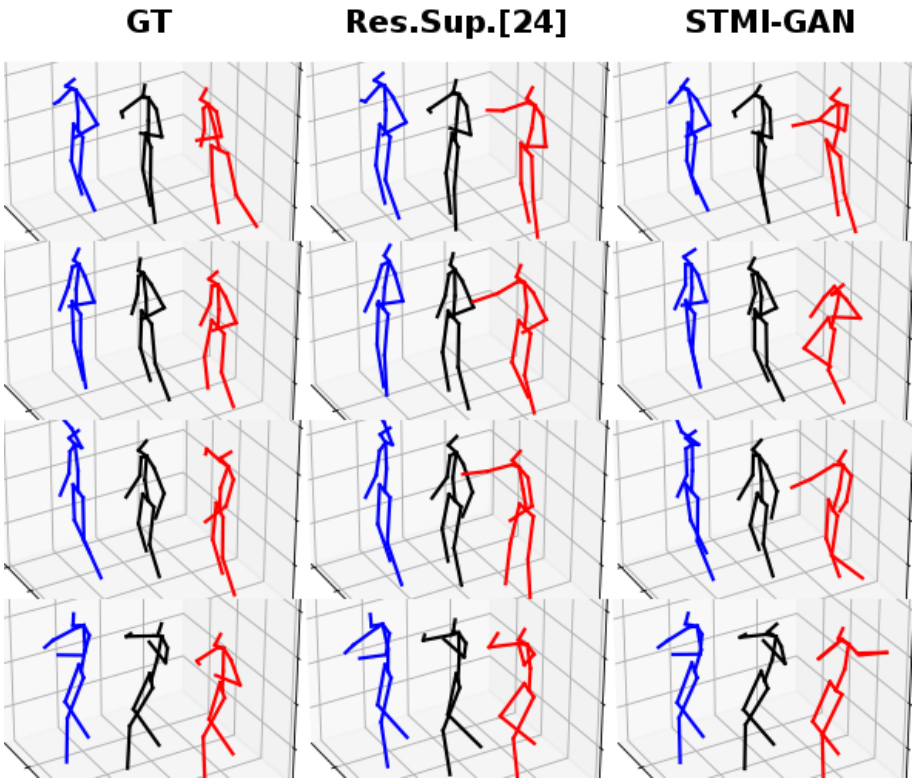

Figure 4.5: **H3.6 Examples.** Blue is first frame, black is first predicted frame, red is last predicted frame. Total length is 4 secs, 2 seed + 2 predicted.

used to train each model. The baseline models aim to minimize the spatial distance through the L2 loss. In our work we seek to reproduce the the distribution of human motion, and we use a GAN for this purpose. These objectives are not always aligned, and the L2 metric often fails to grasp the complexity of realistic human motions.

Noise Injection. It may seem that the noise injection would cause big differences in the output of the network, but actually the expected difference in the prediction is around $0.81mm$ per joint. This means that when called with the same seed sequence, the network produces almost identical sequences, only tweaking minor aspects of the sequence. This result confirms the effects of the noise reported in [52]. It is also interesting to note that the difference increases with the length of the prediction, meaning that indeed the injected noise is solving some level of uncertainty, but the maximum difference that we have measured predicting 4 seconds is $3.02mm$ per joint.

Qualitative Evaluation. We conducted evaluation with 15 person and four distinct surveys, all of them following the same scheme: a prediction model vs the ground truth. In the first two surveys we tested the baseline Res.sup. and STMI-GAN, to perform relative motion prediction. In the case of our model we removed the translation from the prediction to make it comparable. The last two surveys assess

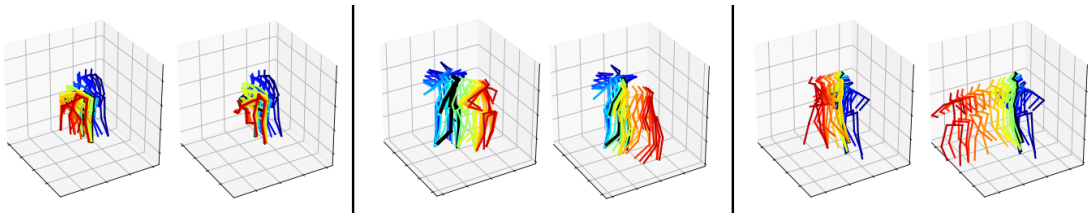

Figure 4.6: **Example result.** Three examples of the predicted motions (left: ground truth, right: predicted). The bluish colors are the part of the sequence that is observed. The prediction starts after the black skeleton and corresponds to the yellow-reddish colors.

the absolute motion prediction. We use our NoGAN model as baseline vs our STMI-GAN model. In these surveys our goal is to obtain a 50% chance of being classified as real. More than 50% would mean that the model is "more realistic" than the ground truth. As we can observe in Tab. 4.2 the results have a wide range of values. This is due to the fact that the survey was sent to a diverse audience.

We can see that the baselines perform in a similar range of average values as our model, our model being bit better. However, the baselines perform very poorly to the trained eye, as the min score tells us. It is worth to note that while the relative motion prediction is an easier problem compared to the absolute motion prediction, the average score of our STMI-GAN is lower in the relative setting. This suggests that the relative motion generation is both harder for machine learning models and for humans. Also, the highest score on the surveys is in the max of the STMI-GAN in absolute prediction.

Model	Motion	Min	Avg	Max
Res.sup [68]	Rel.	6.25%	31.88%	40.63%
STMI-GAN	Rel.	25.00%	33.54%	40.63%
NoGAN	Abs.	9.38%	31.46%	62.50%
STMI-GAN	Abs.	15.63%	**38.39**%	62.50%

Table 4.2: **Human Evaluation.** Percentage of times that a human evaluator thought a generated sequence was real. The min is the score of the "hardest" evaluator, who was fooled the less by the generative models. The max is the score of the "easiest" evaluator, who was confused more often by the model.

Problem	Linear Int	LR-Kalman [9]	NoGAN	STMI-GAN
Joint Occl.	232.06	329.23	**96.52**	108.99
Limb Occl.	209.45	312.40	**123.07**	189.09
Missing Frames	**50.42**	123.05	72.67	102.03
Noisy Transm.	**94.54**	308.98	98.53	110.29

Table 4.3: **Occlusion Completion.** We test different types of occlusion, concretely: **Joint Occlusions:** joints occluded at random in each frame. **Limb Occlusions:** joint chains representing limbs are occluded at random in each frame. **Missing Frames:** entire frames are occluded at random. **Missing transmission:** data points in any dimension are occluded at random. The table reports the L2 metric over coordinates(See 4.4).

Fig.4.6 shows three examples. The sequence on the left is easy to predict, just

continuing the motion of picking up things off the floor. The sequence on the center is a bit harder, as it is easy to guess that the person will continue walking, but the person halts after a couple of steps, and this is unexpected. The sequence on the right is challenging, because just before the generator begins its forecast, the person stops. This makes it hard to predict as the uncertainty rises and many options are plausible. We can see that the generator in fact guessed the action (walking) and the correct direction, but the speed of the motion is not accurate. We consider it however a good guess given the input.

4.7.2 Occlusion Completion

Finally, in Table 4.3 we report the robustness to different types of occlusion. In every test we used 80% of occlusion, *i.e.* we are trying to recover a sequence conditioning only on 20% of the data. The generator model is particularly robust to structured occlusions, it produces good results even without the GAN, we hypothesize that this is because it was trained to produce anthropomorphic guesses. When the occlusions happen at random, linear interpolation is also a good approach, but depending on the nature of the occlusion we may need a more robust model.

4.8 Chapter summary

In this chapter, we propose a Generative Adversarial Network (GAN) to predict human motion given a sequence of past 3D skeleton poses. Although recent GANs have shown promising results, they can only predict plausible motion over relatively short time periods (a few hundred milliseconds) and generally ignore the absolute position of the skeleton with respect to the camera. Our system provides long-term predictions (two seconds or more) for both posture and absolute position. Our approach relies on three main contributions. First, we represent the data using a spatio-temporal tensor of 3D skeleton coordinates, which allows us to formulate the prediction problem as an inpainting problem for which GANs are particularly well suited. Second, we design an architecture for learning the joint distribution of postures and global motions that is capable of hypothesizing large portions of the input 3D tensor with missing data. Finally, we argue that the L2 metric, previously considered the standard by most approaches, fails to capture the true distribution of long-term human motion. We propose two alternative metrics based on the distribution of frequencies that can capture more realistic movement patterns. Extensive experiments show that our approach significantly improves the state of the art and also handles situations where previous observations are corrupted by occlusions, noise, and missing images.

Chapter 5

Program Generation

As a first step toward program generation for motion prediction, we explore the field of dynamic neural networks in the context of cellular automata. Cellular Automata is a model based on a grid representation of the world. Each cell of the grid is an automaton or program which perceives the environment (i.e., its state and the state of neighboring cells) and updates its state according to a fixed rule, generally a mathematical function. The rule is the same for each cell and does not change over time. CA models can be universal [13], meaning that they can be programmed to emulate any other system without changing its underlying construction. Despite their expressive power, the main drawback of CAs is that, given a particular rule, it is impossible to predict its behavior. This means that each rule must be tried with multiple initial states to understand its behavior, and its performance must be observed over many iterations.

Neural Cellular Automata (NCA), recently proposed by [71], is a class of Cellular Automata that use an artificial neural network as the update function so that the NN parameters can be learned to obtain a desired behavior. Even more, the specific behavior is learned without specifying the intermediate states, but by minimizing a loss toward a target pattern, i.e., an image. The NCA approach is conceptually different from other image generation techniques in that it aims to create a complex program that is itself capable of generating the final image. Accordingly, its properties are different to those of a classical ANN, e.g., it can recover damaged patterns.

Although the NCA is an interesting model, its practical application is hampered by the fact that each program must be learned individually. This is not ideal for us, because we would like to have a model that can adapt to many different situations and generate programs even for unseen situations. We have found a solution to this problem by using dynamic neural networks (DNNs). As explained in section 2.3, DNNs are neural networks that generate other neural networks, which would allow us to model a manifold of NCA. This is consistent with our goal, as we did not want to model a single behavior, but a manifold of behaviors. In this context, we can think of behavior as a movement performed for a particular action and context. Thus, by learning a manifold, each program or network in the manifold could represent a

behavior.

The original NCA has another inconvenient feature, which is that it uses the alpha or "visibility" channel in images to encode a "alive" state. While this is an interesting feature for CA, we see it as a limitation for a model that is intended to be applicable to many types of data. In this work, we also address the requirement of an alpha channel of the original NCA [71], which allows for more general applicability to any type of data in tensor form.

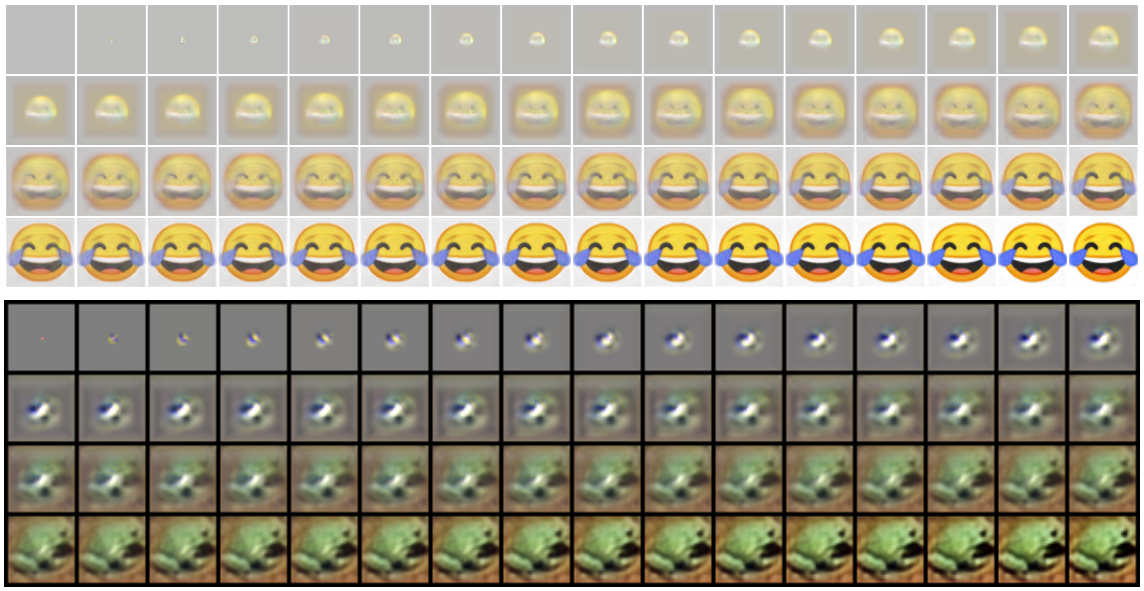

Figure 5.1: **Growth process** step by step from pixel seed image. Top, sample from the full `NotoColorEmoji` dataset. Bottom, sample from the full `CIFAR-10` dataset.

Our proposed model is also inspired by biology. CA models are often used to model biological processes such as tumor growth processes [48] and infection dynamics [21]. This is no coincidence, as the CA model was originally developed to model "living" organisms. Even more, the main inspiration of the NCA was to model the processes of cellular differentiation in multi-cellular organisms. These processes lead to the emergence of a whole body from a single cell. Complex organisms must also generate different types of somatic cells and arrange them spatially to form the different tissues while maintaining temporal stability. These three aspects, cellular differentiation, morphogenesis, and cell growth control, are the pillars of developmental biology. Computational methods are an essential part of developmental biology research, so much so that the term ”morphogene” itself was coined by Turing [92] decades before it was empirically demonstrated. In the figure 5.1, we can observe how images can ”grow” starting from a single pixel, much like an organism starts from a single cell.

With these ideas from biology in mind, it was only natural to draw an analogy between our model and living cells. Having addressed the process of morphogenesis,

we turned our attention to the previous step in the system, DNA coding and decoding. In most complex organisms, DNA is protected inside a nucleus. DNA expression outside the nucleus is carried out by the cellular machinery and regulated by transcription factors (TFs). Many TFs are involved in defining the spatial arrangement of an organism. Most of them are morphogens, soluble molecules that can diffuse and transmit signals via concentration gradients. This biological model also inspired our network architecture (see Fig. 5.2). In our network, we create a vector encoding that stores information common to all cells, as in DNA. For this reason, we also propose a way to transform our vector encoding, which was originally a conventional continuous tensor encoding, into a DNA-like categorical encoding. It is important to point out that this is not a necessary ingredient for our model to work, but rather a proof of concept that allows us to test the limits of biological analogy.

The main contributions in this chapter are:
- Introducing a new type of model that can learn a space of programs in the form of Cellular Automata, which are capable of producing desired target patterns.(see Sec.5.2)
- Showing that the learned program's space has generalization capabilities on the results the programs produce.(see genetic engineering experiments in Sec.5.3)
- Showing that the encoding space is capable of learning and representing up to 50.000 different programs simultaneously with only 512 real-valued dimensions.(see CIFAR experiments in Sec.5.3)
- Introducing a new fully dynamic network architecture, which generates CA's parameters from NN output.(see Sec. 5.2.3)
- The architecture proposed is trainable end-to-end, without need of pre-training any part, for datasets of different sizes and characteristics.
- Extending the capabilities of the original NCA [71] from RGBA to RGB images, and potentially to any type of vectorial data.
- Demonstrate how to build a categorical encoding space similar to DNA, capable of encoding the same information that a continuous encoding space while achieving high robustness to random mutations.

5.1 Related work

Conditioning Models. Conditioning models use information either from the same network (e.g. squeeze and excitation block [38]), from a joint network (e.g. style network [53, 54]) or from a completely independent source (e.g. language embedding [18]), to scale the internal representation channel-wise. The key part is to inject into the network the all the information it needs to perform its task. The conditioning is not always considered an input in the standard manner, but rather transformation on it. We could have used the conditioning approach to embed the NCA into our model, but this would have changed the original architecture by adding the conditioning layer to the CA. Such layer would also add an external source of information at each step, which would change the definition of a CA. Because our goal is to preserve

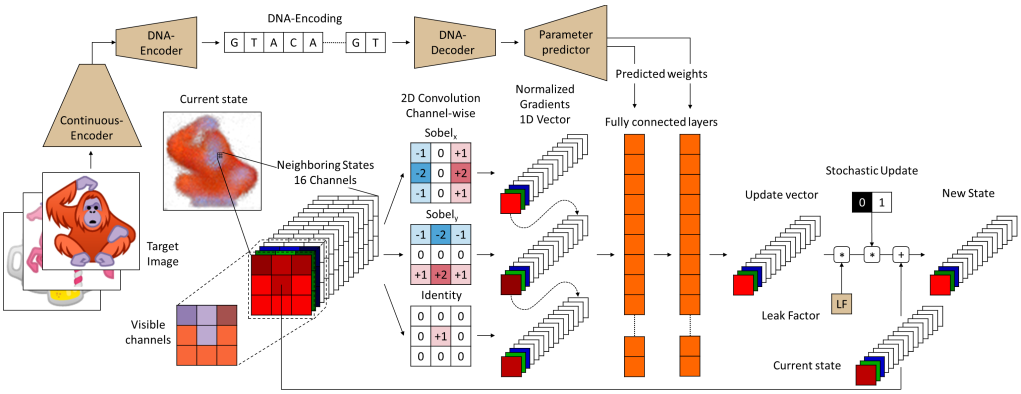

Figure 5.2: **Model overview.** Beige elements contain trainable parameters while orange layers use only predicted parameters. See Fig.5.3 for details of the architecture.

as much as possible the architecture of a CA model, this approach was not ideal to us.

Dynamic Convolutions. Instead of using conditioning to inject the information, we opted for the dynamic convolutions, in which the weights of the convolutional kernel are specifically computed for each sample. In previous works that use dynamic convolutions [47, 57], the architecture forks in two paths from the input image: one path computes the kernels while the other process the input through some fixed convolutions before feeding applying the previously computed dynamic kernels. In contrast, in our approach, the kernel weights are generated from an encoding which is completely independent (and different) from the image to be processed by them. Indeed, the image to be processed by our dynamic convolutions is the pixel seed, which is the same for all targets.

Dynamic Networks. The formulation of our architecture makes the decoder a fully dynamic network, in which the weights are computed on the fly for each sample. In this perspective, it is related to the deep fried transform [110] one shot learners [6] and the HyperLSTM [28]. The goal in these works was to classify images or text, which is a very different task from ours. To the best of our knowledge, our model is the only using a fully convolutional and fully dynamic network architecture in a generative setting.

5.2 Neural Cellular Automata Manifold

We already described in the introduction the biological inspiration of our model. Next, we move forward to its detailed formulation. The Neural Cellular Automata Manifold (NCAM) uses dynamic convolutions to model a space of NCAs. This space is learned through an Autoencoder framework. Formally, our model is defined by

the following equations:

$$\begin{aligned}
\boldsymbol{I}^t &= \{\boldsymbol{C}_{ij}^t\} \quad \forall i,j \in \boldsymbol{I} \\
\boldsymbol{C}_{ij}^t &= f(\boldsymbol{C}_{ij}^{t-1}, \boldsymbol{M}_{kl}^{t-1}, \kappa(\boldsymbol{e}^I, \boldsymbol{\theta}), \theta_{LF}) \quad \forall (k,l) \in \boldsymbol{\epsilon}_{ij} \\
\boldsymbol{M}_{ij}^t &= g(\boldsymbol{C}_{ij}^{t-1}, \boldsymbol{M}_{kl}^{t-1}, \kappa(\boldsymbol{e}^I, \boldsymbol{\theta}), \theta_{LF}) \quad \forall (k,l) \in \boldsymbol{\epsilon}_{ij} \\
\kappa(\boldsymbol{e}^I, \boldsymbol{\theta}) &= \mathcal{P}(\mathcal{D}(\boldsymbol{e}^I, \boldsymbol{\theta}_{\mathcal{D}}), \boldsymbol{\theta}_{\mathcal{P}}) \\
\boldsymbol{\theta} &= \{\boldsymbol{\theta}_{\mathcal{P}}, \boldsymbol{\theta}_{\mathcal{D}}\} \\
\boldsymbol{\epsilon}_{ij} &= (\{i-n_x, \ldots, i+n_x\}, \{j-n_y, \ldots, j+n_y\})
\end{aligned} \tag{5.1}$$

where $\boldsymbol{I}^t$ is the image generated at step t, $\boldsymbol{C}_{ij}^t$ is the color vector (RGB or RGB-Alpha) of the pixel in position (i,j), $\boldsymbol{M}_{ij}^t$ is the corresponding vector of "Morphogenes" (i.e. invisible channels in the grid), $\boldsymbol{\epsilon}_{ij}$ are indices of the neighborhood of the cell (i,j) which extend n_x, n_y positions to each side in x and y axis, and $\boldsymbol{e}^I$ is the vector encoding the image. $f(\cdot)$ and $g(\cdot)$ are the functions implemented as an NCA to predict the colors and "Morphogenes", respectively. $\kappa(\boldsymbol{e}^I, \boldsymbol{\theta})$ is the function that predicts the weights of the NCA from the encoding $\boldsymbol{e}^I$ and its learned parameters $\boldsymbol{\theta}$, which is the composition of the functions learned by the DNA-decoder $\mathcal{D}(\cdot)$ and the Parameter Predictor $\mathcal{P}(\cdot)$. The learned parameters are $\boldsymbol{\theta}_{\mathcal{P}}$, $\boldsymbol{\theta}_{\mathcal{D}}$ and θ_{LF}, the Leak Factor (see Sec. 5.2.1).

In order to train this model, we could simply feed it with arbitrary codes, compute the reconstruction error for corresponding target images, and back-propagate the error to learn the parameters. However, we found it more sensible to learn latent codes that can exploit similarities between images. We, therefore, decided to use an Auto-Encoder architecture [61] at the macro level, which learns to map the set of inputs to a latent space and reconstructs them back. The Autoencoder consists of an Encoder and a Decoder (see Fig. 5.3). The Encoder is composed of two main components: a continuous encoder, that maps the input to a continuous variables encoding; and a DNA-encoder, that transforms this encoding to a categorical variables encoding. The Decoder's structure is symmetrical, mapping first the DNA-encoding to a continuous encoding which, in turn, feeds the parameter predictor block. From an auto-encoder perspective, the NCA should be considered the third part of the decoder since it is the responsible for finally providing the reconstructed images. From a NN perspective, we can see the NCA as a stack of Residual CNN blocks sharing weights or a "Recurrent Residual CNN".

5.2.1 Architecture Details

The architecture of the net is based on different types of residual blocks [31]. The smallest building blocks are of 3 types: Convolutional Block 3x3 (CB3), Convolutional Block 1x1 (CB1) and Fully Connected Block (FCB) (see details in Fig. 5.3). Unlike most of previous works [31, 89, 108], the blocks do not modify input/output representation dimensionality, neither in space resolution or number of filters. While these characteristics can be useful for classification problems, where information

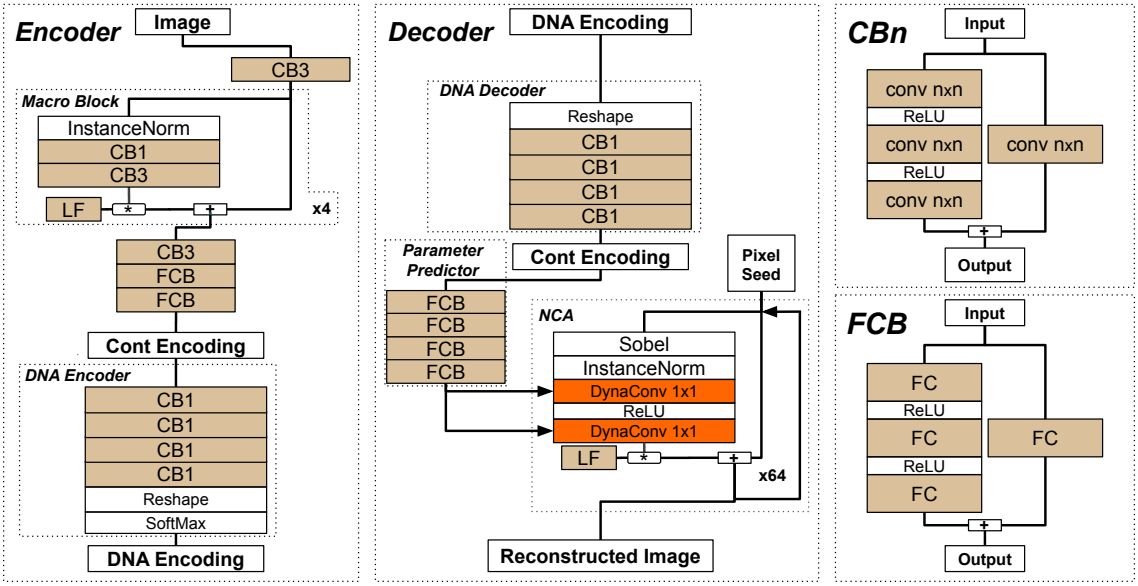

Figure 5.3: **Architecture Details.** Beige elements contain trainable parameters while orange layers use only predicted parameters. CBn blocks can be CB1 or CB3. All blocks share the same number of filters except the last blocks whose output matches the encoding or parameters dimensionality.

needs to be condensed through processing steps, our intuition is that this is not desirable in a reconstruction problem since detail information is lost.

A significant peculiarity of CB1 and CB3 is the expansion of the number features in the internal layer by a factor of 4 and 2, respectively. We consider that this increase of dimensionality of the encoding space allows for a better disentanglement of the data manifold. Notice that CB1 needs to be used in combination with CB3 to introduce neighbourhood information. This detail is opposed to previous architectures [31, 89, 108] that reduce the number of filters in the inner layer of the block, while increasing dimensionality in the short-cut path.

A specific detail of the architecture is the use of a Leak Factor (LF) as an aid for training stability. LF is a single learnable parameter that regulates how much each successive step contributes to the output. A low value of LF encourages the network to retain most of the knowledge from previous steps, avoiding to distort the image too abruptly at any given step. After the network has learnt the basics of the problem this scenario is less likely, thus the network can learn to let more and more information to leak through each step. LF is constrained between 10^{-3} and 10^{3}, initialized at 10^{-1}.

Unlike many architectures that only take the spatial mean of the convolutional output tensor to feed FC layers [31, 89, 108], we use 3 additional slices. Being $(c * h * w)$ the dimensions of the output tensor, a spatial mean over h, w provides a (c) dimensional encoding of the image. A mean over h yields a $(c * w)$ dimensional vector, and doing similarly over the other 2 dimensions (c and w), we obtain the input to our FCB of $dimension = c + (c * h) + (c * w) + (h * w)$.

5.2.2 DNA-encoding

The vector encoding, or latent encoding can be interpreted as the source code that dictates the behavior on the cells of each of the NCAs. This definition inspired us to think of a possible alternative representation of such source code: each value in the latent vector can be encoded into a categorical base, making our encoding compatible to that of the DNA. Notice that a simple quantization on the continuous variable would provide an ordinal discrete variable instead of a DNA-like categorical one. This encoding is dimensioned to handle the same 32 bits of information of the corresponding float variable, thus no additional bottleneck or expansion is introduced here.

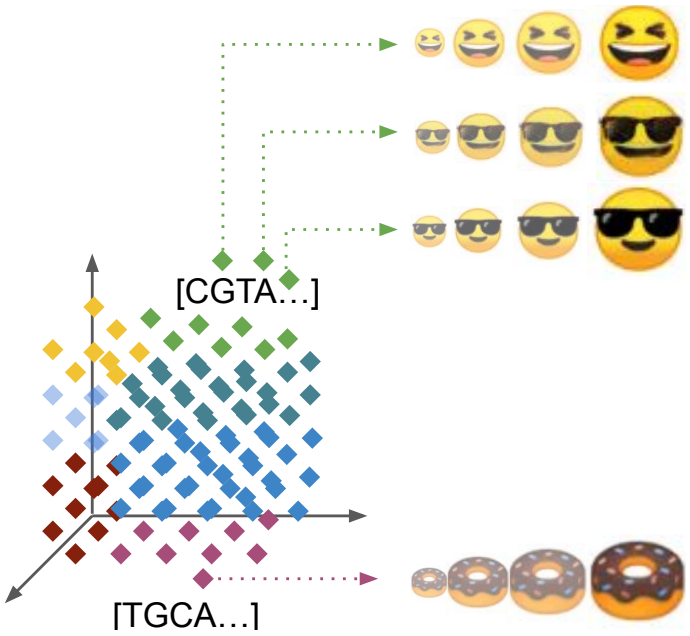

Figure 5.4: **A manifold of Neural Cellular Automata.** Our model encodes a manifold of programs in a DNA-like encoding. The encodings are transformed into NCAs parameters, whose behavior is able to reconstruct (or "grow") a desired target image from a pixel seed.

As we can see in Fig. 5.3, the DNA-encoder is composed of 4 successive CB1 feeding a softmax layer to obtain the 4-categories DNA-encoding. The DNA-decoder follows a symmetrical structure, mapping back each "gene" to a continuous feature. These CB1 are all 1D convolutions, independently expanding each feature of the continuous encoding to a 16-features "gene". This independent processing of different variables makes no assumption about the "meaning" of each category for one variable in relation to its "meaning" in other variables, so the actual "meaning" depends only on latter interaction between corresponding continuous variables.

In order to train our encoding, we add a biologically plausible noise replacing half of the letters by randomly drawn letters. Our intent is not to precisely mimic the mutation rate of DNA ($2.5 \times 10^{-8}/generation$ [74]) but to enforce a high redundancy in the encoding.

This DNA encoding has also the property that it is amenable to recombination. In our "genetic engineering" experiments (Sec.5.3) we show that it is possible to produce sensible new images from CA defined by new codes based on existing samples in the dataset. For the sake of similarity with the biological model, we also included an encoding conceptually similar to that of DNA, i.e. a vector encoding on a base of four possible categorical values similar to our DNA's cytosine, guanine, adenine, and thymine bases (the well-known "CGAT" sequences).

5.2.3 Dynamic Convolutions

Similarly to [57], for an image sample I, being $\boldsymbol{X}^I$ the input tensor of the convolution and $\boldsymbol{Y}^I$ the output tensor, we define the dynamic convolution as $\boldsymbol{Y}^I_n = \sum_m \kappa(\boldsymbol{e}^I)_{mn} \star \boldsymbol{X}^I_m$, where $\kappa(\boldsymbol{e}^I)_{mn}$ is the convolution kernel dynamically computed from the encoding for I, and (m, n) are the input and output channels of the convolution. Although dynamic convolutions are quite a novel idea, in our case they are the simplest way to achieve our goal, keeping the original NCA architecture unchanged but encapsulated in a meta-learning system.

5.2.4 Neural Cellular Automata Architecture

The last component of the decoder is an NCA. To reconstruct an image, the NCA starts from a pixel seed image (typically a blank image with a different pixel). Then, each cell in the grid recurrently perceives the environment and computes an update. After a fixed number of steps, the output is evaluated and the reconstruction error back-propagated. In [71], NCA's kernel weights are directly updated, but in our case the error is back-propagated further, through the NCA and to the parameter predictor, updating its weights instead of the NCA's. We can divide the NCA architecture in two main blocks, perception and update.

The **Perception Layer** was designed to propagate the gradients across the grid in the 16 channels composing it (the first four corresponding to the visible RGBA). This is achieved with manually defined Sobel filters (see Fig. 5.2), similarly to [71]. Since our network's architecture has only *ReLU* activation functions, the activations have an unbounded positive value. Given the recurrent architecture of the NCA, during the training process, an exponential reinforcement can occur, leading activations and weights to ∞ and degenerate states. To avoid this problem, we apply instance normalization [93] on top of the Sobel filters.

To compute the **Update**, we use the NCA architecture, consisting on 2 pixel-wise dense layers (implemented as 1x1 convolutions). The update is modulated by the Leak Factor (the only trainable parameter in our NCA), analogous to the LF used in the Continuous Encoder. Finally, the update is stochastically applied on the current state with an independent probability of $p = 0.5$ for each cell.

Figure 5.5: **Growth results.** First row is a random set of images from the full `NotoColorEmoji` dataset. Following rows are generated by different variants of NCAM codified as: CE: Continuous encoding, DNA: DNA-encoding; STO: Stochastic update, SYN: Synchronous update; 16-512: number of channels in NCA (16,32) - Dimensionality of the continuous encoding (256, 512, 1024) (DNA dimensionality is 16 times that of continuous).

5.3 Experiments

Datasets. `NotoColorEmoji` dataset [23]: 2924 images of synthetic emojis. This kind of images are very simple and with sharp edges, therefore it is relatively easy to visually assess the quality of the produced images. The original images are 128x128 pixels but in our experiments we downscaled them to 64x64 to reduce the computational requirements. `CIFAR-10` dataset [58]: 50.000 real 32x32 images from 10 categories: plane, car, bird, cat, deer, dog, frog, horse, ship, truck. The number of images and its variability make it challenging.

Continuous vs. DNA encoding. Generation results for these two sets of experiments are very similar, both visually (see Fig. 5.5, rows 1-2) and numerically in MSE (CE: 0.01591, DNA: 0.01633). Results show no clear reduction or increase on performance with the extra processing involved in the DNA encoding and decoding process. We consider that the proposed methodology achieves the desired objective of obtaining a categorical encoding equivalent to the continuous encoding commonly used in DNNs. Moreover, we applied a 50% chance of category error during the tests to showcase the remarkable robustness achieved by this encoding.

Stochastic vs. synchronous update. We consider that the stochastic update

Figure 5.6: **Growth results on reduced datasets.** For each subset of `NotoColorEmoji`, first row are random images while the second are generated using DNA-encoding and stochastic update.

feature of the NCA, while being relevant in the biological analogy, may suppose an extra difficulty in the learning process. In these experiments we remove the stochastic update ($p = 0.5$) making it synchronous at every time step ($p = 1.0$). Notice that this change implies, on average, doubling the number of steps on the NCA processing. It also makes more reliable the expected state of neighboring cells. This modification reduced the error significantly in both scenarios, continuous and DNA encoding. Note that images generated with this approach succeed in reconstructing even the finest details (see Fig.5.5, rows 3-4). MSE is one order of magnitude below stochastic approaches (MSE: CE: 0.00199, DNA: 0.00262). With either kind of update, it is remarkable the absence of artifacts in the background of the emojis, given that we removed the alive masking mechanism used in [71], proving that our design is able to learn by itself the limits of the figure.

Effect of encoding dimensionality. We next evaluate our method under significant changes in the encoding dimensionality setting it to half (256) and double (1024) size. Notice that these dimensionalities refer to the continuous encoding, the actual DNA-encoding dimensions are 16 times larger. The experiments show (see Fig. 5.5, rows 5-6) that there is a significant quality degradation when the dimensionality is reduced to half while there is no appreciable improvement when it is doubled (MSE: 256: 0.02302, 1024: 0.01584). Therefore, we consider that the encoding size is not a bottleneck for the problem at hand.

Smaller datasets. In order to assess the challenge that the size of the dataset and its visual complexity poses on the proposed method, we experiment on 4 different subsets of the `NotoColorEmoji` dataset, classified by the authors according to their visual appearance. `Chars` (96 images): emojis containing characters of the Latin

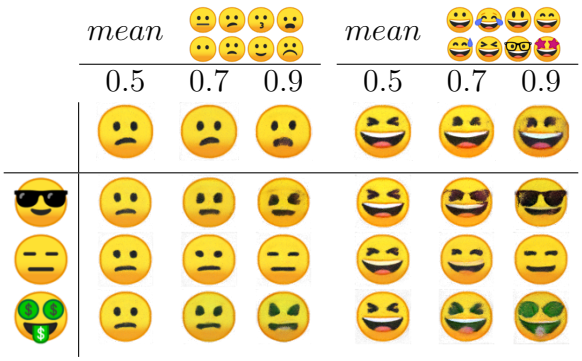

Figure 5.7: **"Genetic engineering" results.** On top: source images for each of two mean encodings. Just below: mean images generated by different thresholds (0.5, 0.7, 0.9): the higher the threshold, the more common need to be the "genes" in order to be injected in the target encoding. Three bottom rows: On the left: 3 original target images; On the right: different images generated when the thresholded mean "genes" are injected into targets.

set and a few symbols. They are simple shapes, usually with straight lines and few colors. `Emos` (103 images): round yellow faces showing emotions. `Heads` (858 images): images of person heads, typically showing different professions. All theses images include the full set of versions according to skin tone. `Variety` (684 images): images not present in other sets that are also visually different among them. It mainly includes animals, objects and food.

Results show that, as expected, problems have a growing level of difficulty in terms of MSE: `Chars`: 0.00976 < `Emos`: 0.01799 < `Heads`: 0.02439 < `Variety`: 0.05035 which can also be visually assessed on Fig. 5.6. The primary factor is the visual complexity or variety, not the size of the dataset. Linear simple shapes seem easier to generate than more rounded or intricate ones.

Ablation Study. A small ablation study using the `variety` dataset gives us the following results: Our proposed architecture achieves a MSE of 0.0504. w/o leak factor: 0.0597. w/o normalization 0.0627. w/o slices: 0.0670. These numbers could be further improved by training longer and tuning the training parameters. In our preliminary experiments we observe that the tendency of the error does not change, thus the ordering of the different options for the model remains the same over time.

Comparison to NCA. The method defined in [71] is a qualitative reference but it can not be a baseline since the original NCA model was designed to tackle a different problem, and the learned CA can only produce a single image. Nevertheless, using the code[1] provided by the authors to train a 64×64px CA, we obtain a MSE in the order of 0.001, comparable to our results with continuous encoding, but on a single image.

"Genetic engineering". We next play a little of "genetic engineering", injecting

[1] `https://colab.research.google.com/github/google-research/self-organising-systems/blob/master/notebooks/growing_ca.ipynb`

Figure 5.8: **CIFAR-10 results** using the best methods: Continuous and DNA Encoding with Syncronous update and baseline dimensionalities.

part of the DNA-encoding from some images to others. To do so, we first generate a mean encoding of a group of source images that share a visual trait. We compute the mean of the 4 discrete features over the samples (i.e. DNA categories) and take the ones with values over a defined threshold. Notice that since they are produced by a softmax we can not have two values over 0.5. If none of the values is over the threshold we would have new category "none" which the DNA-decoder will ignore. With this encoding we generate the mean image. The parts of the mean encoding that are not "none" will substitute corresponding parts of target image encoding.

Results in Fig. 5.7 show some interesting properties of the encoding. The lack of several features in the mean encoding causes no perturbation in the common traits and usually also provides a reasonable solution for the uncommon. The transfer process does not disrupt other common traits in target images. If the traits transferred collide with existing traits, they produce mixed shapes. We can observe that some traits that are not common in the source images are also transferred, suggesting some kind of trait dominance.

CIFAR-10 results. Finally, we report results on `CIFAR-10` dataset [58]. Note that in this case, our NCAM model is capable to generate up to 50K different images. In Fig. 5.1 we show an example of image generation for this case. On this dataset we only experimented with the synchronous update (best solution) since it is difficult to appreciate the level of detail required to visually evaluate the quality of the results (see Fig. 5.8). However, MSE values obtained are CE: 0.00717 DNA: 0.00720, perfectly comparable with those of `NotoColorEmoji`, which implies that the NCAM model also has enough capacity for this dataset.

5.4 Chapter summary

In this chapter, we show that the neural network (NN) architecture of the NCA can be encapsulated in a larger NN. This allows us to propose a new model that encodes a manifold of NCA, each of which is capable of generating a distinct image. In this way,

we effectively learn an encoding space of CA. We achieve this by introducing dynamic convolutions into an auto-encoder architecture that, for the first time, combines two different sources of information: the encoding and information from the cell's environment. Despite the limited time we have spent exploring the behavior of the model, it has already shown good generalization capabilities. More importantly, our work introduces a general-purpose network that can be used for a variety of problems beyond image generation. We lay the foundations needed for the general problem of program generation.

Chapter 6

Concluding remarks

6.1 3D Action Recognition

In this chapter, we presented a novel CNN architecture for human action recognition from 3D skeletal data. Our approach is based on two main ideas: (i) representing sequences of skeletons as sequences of the Euclidean distance matrix (EDM) via a learned transformation of the skeletons' joints; (ii) processing these sequences with a 3D convolutional neural network. We validated our method on three benchmark datasets and obtained SOTA results. In particular, on the large NTU RGB -D dataset, we achieved an improvement in accuracy of $\approx 2\,\%$ over the previous SOTA, while requiring almost 1000 times fewer parameters and operations.

We believe that a natural extension of this work is to combine 3D skeletal data obtained from 3D pose estimation systems with semantic image embedding data obtained from image-based action recognition systems. In this way, we can use the visual information and help distinguish actions where the 3D motion is very similar, e.g., "eating" and "drinking." In addition, multiple modalities would also be very useful to understand a more complex taxonomy of actions and detect subtle differences that may have important semantics, e.g., instead of "talking" we could detect "arguing" and "chatting", which are impossible to distinguish without fine-grained information in the image. Although we have not attempted to explore this line further, we believe it is promising and has a wide range of applications.

6.2 Motion Prediction

We presented a novel GAN architecture for predicting 3D human motion from historical 3D skeletal data. We extended existing work by also predicting the absolute position of the body. We formulated our problem as an inpainting task in spatio-temporal volumes. To capture the essence and semantics of human motion, the training of our network was mainly guided by three independent discriminators. They promote the generation of motion sequences with a frequency distribution similar to that of the original dataset. Since L2 is known to be insufficient to compare

the generated sequences, we also propose new metrics that estimate the frequency distribution of the datasets and capture the concept of multiple possible futures. Experimental results with Human3.6M demonstrate the effectiveness of our model in generating highly realistic predictions of human motion.

There are many possible lines to continue this work. One of them, developing a more general model, was taken up in the chapter on program generation. But we have left many other possibilities unexplored, such as incorporating context by conditioning the model with visual information, creating an efficient end-to-end system for human-robot interaction, optimizing the model to make it compatible with embedded systems and autonomous vehicles. We hope that other researchers can use the information in this thesis and related publications as a basis for further development of these topics.

6.3 Program Generation

This chapter is a first step in creating encoding spaces that represent not only static information but also behavioral patterns, i.e., programs. Moreover, our experiments generate programs for multiple agents interacting with the environment to achieve the desired collective outcome. Since our model is capable of handling any type of tensor data, we see no theoretical limitation to learning complex programs that perform all types of tasks, based only on data and an approximate loss function. Moreover, thanks to the NCAM encoding space, we can generate programs that produce new, meaningful behaviors that are not seen in the data.

It is important to note that the properties of the manifold learned by the proposed autoencoder depend on the loss function used for training. In our work, we use the MSE loss to learn to reproduce the original images with high fidelity. If we were to use an adversarial loss [22], we would likely obtain visually plausible images of the same class, but not an exact replica. We believe that applying a reinforcement learning loss [70] could allow us to produce a model driven by a fitness metric, such a model would then be similar to genetic evolution. These simple adaptations open up immeasurable applications for the NCAM.

We are particularly interested in applying NCAM to generate programs that simulate human behavior, incorporating semantic and contextual information into the program generation and agent environment. For example, we could generate a program for an action such as running, where the agent should adapt to follow a particular path while performing the expected motion. We did not achieve this goal in this thesis, mainly due to time constraints, but it is future work that we believe can be very valuable not only for motion prediction, but also for applications such as controlling robots and autonomous cars, etc.

Bibliography

[1] TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Antonio Agudo and Francesc Moreno-Noguer. Dust: Dual union of spatio-temporal subspaces for monocular multiple object 3d reconstruction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, volume 1, page 2, 2017.

[3] Ijaz Akhter, Tomas Simon, Sohaib Khan, Iain Matthews, and Yaser Sheikh. Bilinear spatiotemporal basis models. *ACM Transactions on Graphics*, 31(2):17, 2012.

[4] Antonio Valerio Miceli Barone. Low-rank passthrough neural networks. *arXiv preprint arXiv:1603.03116*, 2016.

[5] Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems*, pages 899–907, 2013.

[6] Luca Bertinetto, João F Henriques, Jack Valmadre, Philip Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. In *Advances in neural information processing systems*, pages 523–531, 2016.

[7] Hakan Bilen, Basura Fernando, Efstratios Gavves, Andrea Vedaldi, and Stephen Gould. Dynamic image networks for action recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3034–3042, 2016.

[8] Matthew Brand and Aaron Hertzmann. Style machines. In *ACM Conference on Computer graphics and interactive techniques*, pages 183–192. ACM Press/Addison-Wesley Publishing Co., 2000.

[9] M Burke and Joan Lasenby. Estimating missing marker positions using low dimensional kalman smoothing. *Journal of biomechanics*, 49(9):1854–1858, 2016.

[10] Judith Bütepage, Hedvig Kjellström, and Danica Kragic. Anticipating many futures: Online human motion prediction and generation for human-robot interaction. In *IEEE International Conference on Robotics and Automation*, pages 1–9. IEEE, 2018.

[11] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. *arXiv preprint arXiv:1611.08050*, 2016.

[12] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[13] Matthew Cook. Universality in elementary cellular automata. *Complex systems*, 15(1):1–40, 2004.

[14] Huseyin Coskun, David Joseph Tan, Sailesh Conjeti, Nassir Navab, and Federico Tombari. Human motion analysis with deep metric learning. *arXiv preprint arXiv:1807.11176*, 2018.

[15] Timothy Dozat. Incorporating nesterov momentum into adam. *International Conference on Learning Representations Workshop*, 2016.

[16] Yong Du, Yun Fu, and Liang Wang. Skeleton based action recognition with convolutional neural network. In *IAPR Asian Conference on Pattern Recognition*, pages 579–583. IEEE, 2015.

[17] Yong Du, Wei Wang, and Liang Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1110–1118, 2015.

[18] Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. Feature-wise transformations. *Distill*, 2018. https://distill.pub/2018/feature-wise-transformations.

[19] Simon Fothergill, Helena Mentis, Pushmeet Kohli, and Sebastian Nowozin. Instructing people for training gestural interactive systems. In *SIGCHI Conference on Human Factors in Computing Systems*, pages 1737–1746. ACM, 2012.

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

[21] Ramón ER González, Sérgio Coutinho, Rita Maria Zorzenon dos Santos, and Pedro Hugo de Figueirêdo. Dynamics of the hiv infection under antiretroviral therapy: A cellular automata approach. *Physica A: Statistical Mechanics and its Applications*, 392(19):4701–4716, 2013.

[22] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[23] Google. Noto emoji distribution, 2020.

[24] Anand Gopalakrishnan, Ankur Mali, Dan Kifer, C Lee Giles, and Alexander G Ororbia. A neural temporal model for human motion prediction. *arXiv preprint arXiv:1809.03036*, 2018.

[25] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[26] Liang-Yan Gui, Yu-Xiong Wang, Xiaodan Liang, and José MF Moura. Adversarial geometry-aware human motion prediction. In *European Conference on Computer Vision*, pages 823–842, 2018.

[27] Çalar Gülçehre and Yoshua Bengio. Knowledge matters: Importance of prior information for optimization. *Journal of Machine Learning Research*, 17(8):1–32, 2016.

[28] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[29] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *IEEE international conference on computer vision*, pages 2961–2969, 2017.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[33] Alejandro Hernandez, Jurgen Gall, and Francesc Moreno-Noguer. Human motion prediction via spatio-temporal inpainting. In *IEEE International Conference on Computer Vision*, pages 7134–7143, 2019.

[34] Alejandro Hernandez Ruiz, Lorenzo Porzi, Samuel Rota Bulò, and Francesc Moreno-Noguer. 3d cnns on distance matrices for human action recognition. In *ACM on Multimedia Conference*, pages 1087–1095. ACM, 2017.

[35] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[36] Geoffrey E Hinton and Richard Zemel. Autoencoders, minimum description length and helmholtz free energy. *Advances in neural information processing systems*, 6, 1993.

[37] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[38] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018.

[39] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, volume 1, page 3, 2017.

[40] Farzad Husain, Babette Dellen, and Carme Torras. Action recognition based on efficient deep feature learning in the spatio-temporal domain. *IEEE Robotics and Automation Letters*, 1(2):984–991, 2016.

[41] Mohamed E Hussein, Marwan Torki, Mohammad Abdelaziz Gowayyed, and Motaz El-Saban. Human action recognition using a temporal hierarchy of covariance descriptors on 3d joint locations. In *International Joint Conference on Artificial Intelligence*, volume 13, pages 2466–2472, 2013.

[42] Heikki Hyötyniemi. Turing machines are recurrent neural networks. *Proceedings of step*, 96, 1996.

[43] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[44] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1325–1339, jul 2014.

[45] Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5308–5317, 2016.

[46] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2013.

[47] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, pages 667–675, 2016.

[48] Yang Jiao and Salvatore Torquato. Emergent behaviors from a cellular automaton model for invasive tumor growth in heterogeneous microenvironments. *PLoS computational biology*, 7(12), 2011.

[49] Li Jing, Jure Zbontar, et al. Implicit rank-minimizing autoencoder. *Advances in Neural Information Processing Systems*, 33:14736–14746, 2020.

[50] Imran N Junejo, Emilie Dexter, Ivan Laptev, and Patrick Perez. View-independent action recognition from temporal self-similarities. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):172–185, 2011.

[51] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[52] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *arXiv preprint arXiv:1812.04948*, 2018.

[53] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[54] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[55] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[56] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[57] Benjamin Klein, Lior Wolf, and Yehuda Afek. A dynamic convolutional layer for short range weather prediction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4840–4848, 2015.

[58] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Technical Report*, 2009.

[59] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems Conference*, pages 1097–1105, 2012.

[60] Taras Kucherenko, Jonas Beskow, and Hedvig Kjellström. A neural network approach to missing marker reconstruction in human motion capture. *arXiv preprint arXiv:1803.02665*, 2018.

[61] Yann LeCun. Modeles connexionnistes de l'apprentissage (connectionist learning models). *Doctoral dissertation*, 1987.

[62] Andreas M Lehrmann, Peter V Gehler, and Sebastian Nowozin. Efficient nonlinear markov models for human motion. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1314–1321, 2014.

[63] Chen Li, Zhen Zhang, Wee Sun Lee, and Gim Hee Lee. Convolutional sequence to sequence model for human dynamics. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5226–5234, 2018.

[64] Jun Liu, Amir Shahroudy, Dong Xu, and Gang Wang. Spatio-temporal lstm with trust gates for 3d human action recognition. In *European Conference on Computer Vision*, pages 816–833. Springer, 2016.

[65] Mengyuan Liu, Hong Liu, and Chen Chen. Enhanced skeleton visualization for view invariant human action recognition. *Pattern Recognition*, 2017.

[66] Jiajia Luo, Wei Wang, and Hairong Qi. Group sparsity and geometry constrained dictionary learning for action recognition from depth maps. In *IEEE International Conference on Computer Vision*, pages 1809–1816, 2013.

[67] Utkarsh Mall, G Roshan Lal, Siddhartha Chaudhuri, and Parag Chaudhuri. A deep recurrent framework for cleaning motion capture data. *arXiv preprint arXiv:1712.03380*, 2017.

[68] Julieta Martinez, Michael J Black, and Javier Romero. On human motion prediction using recurrent neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4674–4683. IEEE, 2017.

[69] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *International Conference on Machine Learning*, pages 3478–3487, 2018.

[70] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[71] Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. Growing neural cellular automata. *Distill*, 2020. https://distill.pub/2020/growing-ca.

[72] Francesc Moreno-Noguer. 3d human pose estimation from a single image via distance matrix regression. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017.

[73] Francesc Moreno-Noguer. 3d human pose estimation from a single image via distance matrix regression. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017.

[74] Michael W Nachman and Susan L Crowell. Estimate of the mutation rate per nucleotide in humans. *Genetics*, 156(1):297–304, 2000.

[75] Andrew Ng. Sparse autoencoder. CS294A Lecture notes vol. 72, 2011.

[76] Eshed Ohn-Bar and Mohan Trivedi. Joint angles similarities and hog2 for action recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 465–470, 2013.

[77] Seymour A Papert. The summer vision project. 1966.

[78] Vladimir Pavlovic, James M Rehg, and John MacCormick. Learning switching linear models of human motion. In *Advances in neural information processing systems*, pages 981–987, 2001.

[79] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[80] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[81] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[82] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[83] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. Ntu rgb+ d: A large scale dataset for 3d human activity analysis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1010–1019, 2016.

[84] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Neural Information Processing Systems Conference*, pages 568–576, 2014.

[85] Sijie Song, Cuiling Lan, Junliang Xing, Wenjun Zeng, and Jiaying Liu. An end-to-end spatio-temporal attention model for human action recognition from skeleton data. *arXiv preprint arXiv:1611.06067*, 2016.

[86] Jaeyong Sung, Colin Ponce, Bart Selman, and Ashutosh Saxena. Unstructured human activity detection from rgbd images. In *IEEE International Conference on Robotics and Automation*, pages 842–849. IEEE, 2012.

[87] Ilya Sutskever, Geoffrey E Hinton, and Graham W Taylor. The recurrent temporal restricted boltzmann machine. In *Advances in neural information processing systems*, pages 1601–1608, 2009.

[88] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[89] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[90] Graham W Taylor and Geoffrey E Hinton. Factored conditional restricted boltzmann machines for modeling motion style. In *International Conference on Machine Learning*, pages 1025–1032. ACM, 2009.

[91] Graham W Taylor, Geoffrey E Hinton, and Sam T Roweis. Modeling human motion using binary latent variables. In *Advances in neural information processing systems*, pages 1345–1352, 2007.

[92] Alan Turing. The chemical basis of morphogenesis. *Sciences*, 237(641):37–72, 1952.

[93] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[94] Raquel Urtasun, David J Fleet, and Pascal Fua. 3d people tracking with gaussian process dynamical models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, volume 1, pages 238–245. IEEE, 2006.

[95] Raquel Urtasun, David J Fleet, Andreas Geiger, Jovan Popović, Trevor J Darrell, and Neil D Lawrence. Topologically-constrained latent variable models. In *International Conference on Machine learning*, pages 1080–1087. ACM, 2008.

[96] Vivek Veeriah, Naifan Zhuang, and Guo-Jun Qi. Differential recurrent neural networks for action recognition. In *International Conference on Computer Vision*, pages 4041–4049, 2015.

[97] Raviteja Vemulapalli, Felipe Arrate, and Rama Chellappa. Human action recognition by representing 3d skeletons as points in a lie group. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 588–595, 2014.

[98] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine learning*, pages 1096–1103, 2008.

[99] Jack M Wang, David J Fleet, and Aaron Hertzmann. Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence*, 30(2):283–298, 2008.

[100] Jiang Wang, Xiaohan Nie, Yin Xia, Ying Wu, and Song-Chun Zhu. Cross-view action modeling, learning and recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2649–2656, 2014.

[101] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: towards good practices for deep action recognition. In *European Conference on Computer Vision*, pages 20–36. Springer, 2016.

[102] Pichao Wang, Wanqing Li, Zhimin Gao, Yuyao Zhang, Chang Tang, and Philip Ogunbona. Scene flow to action map: A new representation for rgb-d based action recognition with convolutional neural networks. *arXiv preprint arXiv:1702.08652*, 2017.

[103] Pichao Wang, Zhaoyang Li, Yonghong Hou, and Wanqing Li. Action recognition based on joint trajectory maps using convolutional neural networks. In *ACM Conference on Multimedia*, pages 102–106. ACM, 2016.

[104] Moritz Wolter, Shaohui Lin, and Angela Yao. Neural network compression via learnable wavelet transforms. In *Artificial Neural Networks and Machine Learning–ICANN 2020: 29th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 15–18, 2020, Proceedings, Part II*, pages 39–51, 2020.

[105] Guiyu Xia, Huaijiang Sun, Beijia Chen, Qingshan Liu, Lei Feng, Guoqing Zhang, and Renlong Hang. Nonlinear low-rank matrix completion for human motion recovery. *IEEE Transactions on Image Processing*, 27(6):3011–3024, 2018.

[106] Lu Xia, Chia-Chih Chen, and JK Aggarwal. View invariant human action recognition using histograms of 3d joints. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 20–27. IEEE, 2012.

[107] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016.

[108] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1492–1500, 2017.

[109] Yanhua Yang, Cheng Deng, Shangqian Gao, Wei Liu, Dapeng Tao, and Xinbo Gao. Discriminative multi-instance multitask learning for 3d action recognition. *IEEE Transactions on Multimedia*, 19(3):519–529, 2017.

[110] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets. In *IEEE International Conference on Computer Vision*, pages 1476–1483, 2015.

[111] Raymond A Yeh, Chen Chen, Teck-Yian Lim, Alexander G Schwing, Mark Hasegawa-Johnson, and Minh N Do. Semantic image inpainting with deep generative models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, volume 2, page 4, 2017.

[112] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Generative image inpainting with contextual attention. *arXiv preprint*, 2018.

[113] Kiwon Yun, Jean Honorio, Debaleena Chattopadhyay, Tamara L Berg, and Dimitris Samaras. Two-person interaction detection using body-pose features and multiple instance learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 28–35. IEEE, 2012.

[114] Songyang Zhang, Xiaoming Liu, and Jun Xiao. On geometric features for skeleton-based action recognition using multilayer lstm networks. In *IEEE Winter Conference on Applications of Computer Vision*, 2017.

[115] Lijuan Zhou, Wanqing Li, Yuyao Zhang, Philip Ogunbona, Duc Thanh Nguyen, and Hanling Zhang. Discriminative key pose extraction using extended lc-ksvd for action recognition. In *International Conference on Digital Image Computing: Techniques and Applications*, pages 1–8. IEEE, 2014.

[116] Wentao Zhu, Cuiling Lan, Junliang Xing, Wenjun Zeng, Yanghao Li, Li Shen, and Xiaohui Xie. Co-occurrence feature learning for skeleton based action recognition using regularized deep lstm networks. *arXiv preprint arXiv:1603.07772*, 2016.