POLITECNICO DI TORINO

**Master degree**
**in Data science and engineering**

Master thesis

# Audio-Visual Human Activity Recognition for Humanoid Robotics


**Supervisors**
Prof. Barbara Caputo
Prof. Tamim Asfour

**Candidate**
Lucia Innocenti

Academic year 2021-2022

*A fagiolino*

**Abstract**

Recent advances in Robotics research are pushing the limits of machines toward faster, smarter and more efficient devices. Notable results have been achieved under the hardware point of view, with humanoid devices that are able to consistently outperform human movements. Under a perception perspective, however, we are still far from being capable to match the human skills. The gap is even more consistent when we introduce constraints typical of on-board implementations, such as limited resources and real-time requirements, which leads to models that potentially works perfectly on a theoretical scenario, but are not deployable in a real application.

Among the others, one of the most important tasks that a humanoid robot should implement correctly to enable a fruitful human-robot interaction is an accurate human activity recognition, namely the online identification of the action that the human is performing, with the goal of triggering a consequent action, *e.g.* to support the task. State of the art approaches typically provide solutions that relies on large and complex neural models, which can hardly be deployed in the constrained hardware of a fully autonomous humanoid robot.

This thesis provides a solution to the problem of real time human activity recognition in the humanoid robotics; the model is built with an eye on the dimensionality, and it is provided a solution based on skeleton data and graph neural networks. The working scenario is a kitchen environment, with a single human and the robot being in the same room. The model focus on both audio and visual data. From the visual data point of view, a camera, hosted in the robot head, collects videos of the human that are sent to the main model, which, after preprocessing, perform a classification task. The proposed architecture demonstrated to be a convenient way for node level, edge level, and graph level prediction task. In this work the focus was on graph level prediction, by comparing different architectures for the updating and messages passing functions. The architecture achieves interesting results on the dataset used, and it is demonstrated to be efficient when deployed directly in the robot hardware. From the audio data point of view, a microphone records the audio in the environment; the human speech is analyzed using Natural Language Processing techniques and a classification model. The outputs of both the visual and the audio based models are the merged for obtaining the final prediction.

Future developments of this work will implement an Incremental Learning policy to expand the knowledge of the model to novel classes, and will consider the enrollment of sequence-based models such as Long short-term memory networks.

# Acknowledgements

-

# Contents

# List of Tables

# List of Figures

*Questa è acqua,*
*io sto cercando l'oceano*
[Disney, Soul]

# Chapter 1

# Introduction

## 1.1 Humanoid Robotics, Where Do We Stand?

Robotics is the intersection of science, engineering, and technology that produces machines, called robots, that replace (or replicate) human actions. Initially created for use in factories, robots have evolved over a long period of history. The spread of computer technology led to a new wave of robotics in the 2000s [1]: robots are not only anymore for industrial purposes, but also used for security, entertainment, health care, military and different other purposes.

As the need for robotics capability increases, robotics software need to evolve. A cobot, or collaborative robot, is a robot intended for direct human robot interaction within a shared space, or where humans and robots are in close proximity. The development of cognitive robots relies on artificial embodiment having complex and rich perceptual and motor capabilities. This leads to humanoid robots as the most suitable experimental platform. While simpler robotic systems might be more suitable for testing some theories in simplified environments, only humanoid robots can provide rich sensorial inputs and complex actions necessary to develop higher cognitive processes. The design of such robots which are capable of developing perceptual, behavioural and cognitive categories in a measurable way and are capable of communicating and sharing these with humans and other artificial agents is a challenging task. The target system is supposed to interact with humans in a reliable and safe manner. In particular, it is meant to be able to cooperate and to enter a dialogue with them.

Human–Robot interaction is the study of interactions between humans and robots. Multidisciplinary in nature, HRI draws on several disciplines including human–computer interaction, artificial intelligence, robotics, psychology, and natural language understanding. A crucial task for successful human–robot collaboration is Human activity recognition (HAR). Indeed, in a world where robots perform such a wide range of tasks, especially in human environments, it is fundamental for them to be able to deduce when and where their help is required, even if not explicitly requested. In a number of fields, from industry to healthcare, it is crucial for the cobot to be able to act autonomously in support of the person (or people) they are in contact with. Recognition of the performed action is

fundamental to achieve this, as it allows the robot to understand the current situation, to build a scenario from which to decide how to act. This is why the task is so well studied, although it still presents various difficulties. Indeed, the complexity of the problem is intrinsic: as a first obstacle, there is the difficulty of unambiguously defining the concept of activity, which can indicate concepts with different granularity. The segmentation between different actions in time evolution is also non-trivial: even for a human being, defining when one activity actually ends and another begins is not always simple or even possible. Then, problems such as visual occlusion, change of point of view, number of people present in the scene, impact and increase the complexity.

## 1.2 Humanoids at H2T

At H2T, the High Performance Humanoid Technologies research group in the Karlsruhe Institute for Technology, they created an environment made of different robots and the whole infrastructure for managing and working with them. With ArmarX, the event-

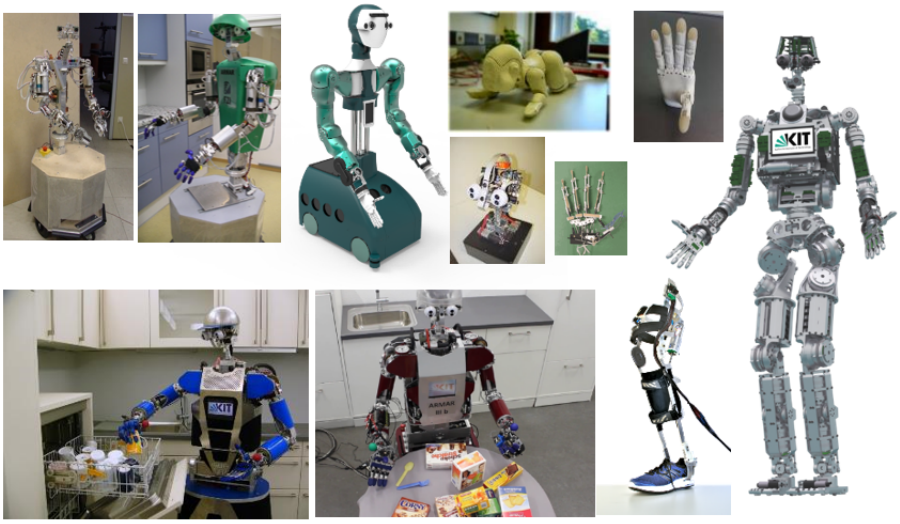



Figure 1.1: The ARMAR family [2]: they are different humanoid robots developed by the H2T team at KIT since 1999.

based software infrastructure, they introduce a robot programming environment that has been developed in order to ease the realization of higher level capabilities needed by complex robotic systems such as humanoid robots. The work presented in this thesis is built upon ArmarX (hereinafter named the infrastructure) and ARMAR-6 (hereinafter named the robot). ARMAR-6 is a humanoid assistance robot for industrial environments that interacts with humans and provides them with proactive support. Being able to understand what a human is doing is fundamental in the area of cobot and robot for industrial applications, in which they are supposed to support the human in their goals.

Starting from the video captured from a Microsoft Kinect posed in the robot head, and from the audio recorded from a microphone, the HAR model has been developed: it

is made of an algorithm based on GNNs and an NLP method, and they work together for recognizing the action based on changes in the human pose among different frames and on the analysis of the speech made by that person while performing the actions. The model was trained using a lower-granularity version of the Bimanual Manipulation Dataset from the H2T team itself. The videos in the dataset are captured by a camera in front of the human, simulating the position of the robot. In order to keep this configuration, in ARMAR-6, it has also been developed a system that makes the robot able to follow, rotating the head, the human along its movements; this can guarantee that the human pose is always centered in the camera frame. The dataset was originally made of 13 different classes, but because of their behaviours a subset of 7 was used. In fact, some of the classes in the dataset represent more actions that activities (like Close and Open a bottle), and so they were out of the scope of the thesis.The model achieves very good results, with an accuracy of over 80% in the test phase. In addition, it also achieves the goal of lightness and low impact on the use of resources, since it is built using only data that was already collected and analysed for other purposes, thus avoiding burdening the computational process of the robot.

## 1.3 Thesis organization

This thesis is organized as it follows: in Chapter 2, the robotics part is introduced, explaining the basic of robotics and the ArmarX framework that has been used for this work; in Chapter 3, the activity recognition problem is formalized, and the different approaches available in the literature are presented. The framework of graph neural networks, that are the main model used in the thesis, is illustrated in Chapter 4, where it is also available a review of basic graph theory for the reader that is not familiar with the topic. Chapter 5 is the core of the thesis, as it contains the proposed solution, the experiments and the obtained results; finally, Chapter 6 contains the conclusions and the discussion about the AVAR model, as well as the analysis of future possible works.

# Chapter 2

# Robotics

## 2.1 Generalities

A definition of Robot, by the ISO standard, is "actuated mechanism programmable in two or more axes with a degree of autonomy, moving within its environment, to perform intended tasks" [3]. Robotics, therefore, is a field that refers to the study of machines capable of replacing humans in the performance of tasks, whether these be physical work or decision-making. Humans have always looked for ways to automate their tasks, especially the heaviest and riskiest ones. For this reason, the idea of robotics and automation in general culture predates by far its actual technological introduction. Even the term Robot was coined by Karel Capek, a Czech writer, and used in one of his works (Rossumovi univerzální roboti).

Around the 1930s, the development of the first industrial robots began in the United States of America. At the same time, the science fiction writer Isaac Asimov formulated the laws of robotics, which represent the expectations and fears that society had of robots at that time:

- First Law: A robot may not injure a human being, or, through inaction, allow a human being to come to harm.
- Second Law: A robot must obey orders given it by human beings, except where such orders would conflict with the First Law.
- Third Law: A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

Even nowadays, since there is no robot or automatic machine to which any of them apply, they are considered fictitious. However, since the laws were created, great strides have been made in the development of robotics: around 1949, at the Burden Neurological Institute in Bristol, it was created the first electronic autonomous robots with complex behavior; in 1954, George Devol constructed the Unimate, the first digitally controlled and programmable robot. This later laid the foundations of the modern robotics industry; Stanford Research Institute developed Shakey, the first mobile robot capable of reasoning

about its environment, in 1970; in 2014 it was presented the Pepper, a humanoid robot by SoftBank Robotics, designed with the ability to read emotions. Thanks to the evolution of both hardware and software, especially with the rise of artificial intelligence systems, it is possible to have robots that can talk like us, walk like us and express a wide range of emotions.

### 2.1.1 Humanoid Robotics

Humanoids are robots whose appearance is similar to that of the human body. Some humanoid robots may replicate only specific parts, others may duplicate the entire body: it depends on the application. It can be used for interacting with people, for substituting humans in certain contexts, or for experimental purposes. Firstly created in the 70s as a field, several projects have been developed since then. In the development of humanoids, as in other robotic applications, there are both hardware and software challenges. An essential element in the development of a humanoid robot are sensors, which mimic the human senses and facilitate human-robot interaction. Vision, audio and tactile are the sense in which researchers most focused on. Moreover, internal sensors for position and velocity are usually present. Data received from these internal sensors are used for addressing the body motion. Combination of internal and external sensors data are fundamental for the manipulation task.

Given a robot, and an environment in which the robot is located, we need to describe point's positions in space. For doing that, it is essential to have a coordinate system and a framework to describe positions and orientations, so that we can also address change in them, by translations or rotations into the 6 degrees of freedom (DOF). Let's give some basic definitions. The joints of the robot connect adjacent links, that are rigid elements, and allow for relative motion between them. The number of DOF represents the number of parameters needed to specify a configuration. An essential concept in kinematic is the pose: it is the positioning and orientation of the robot in the global coordinate frame. Defined a fixed world reference coordinate frame **W**, a local coordinate frame **R** in the robot's reference point **P** in the space, then it is possible to describe the point **P** itself as a vector **r** representing the relative position of point **P** w.r.t. referent point of system **W**.

## 2.2 The ArmarX Framework and ARMAR Robots

The robot development environment (RDE) ArmarX aims at providing an infrastructure for developing a customized robot framework that enables the realization of distributed robot software components. For an RDE to be used with modern robotic platforms, it must meet certain requirements. Specifically, based on [4], it is required to provide:

- distributed processing; this refers to multiple computer systems using more than one computer (or processor) to run an application. A parallel processing method uses more than one CPU to execute programs on a single computer. In robotics, a hardware platform is typically composed of several sub-systems, each of which serves a specific function. Therefore, this requires a distributed system.

- interoperability, which is the ability for different systems to coordinate their communications and collaborate without requiring the end user to make any consistent effort. It is imperative that different hardware platforms and operating systems are integrated, and the RDE should provide an interface that takes care of this.
- open source model, that enables researchers and developers to collaborate on improving the model, which ultimately leads to greater improvements in research.

ArmarX meets these requirements. The software has a GPL open-source license; shared memory and Ethernet are used for communications, and it runs on a variety of operating systems and supports a variety of programming languages (C++, Java, and Python are just a few examples).

From a structural viewpoint, ArmarX is comprised of three layers. A representation can be found in the Figure 2.1, in which it is possible to see the Middleware Layer, the Robot Framework Layer and the Application Layer.

The **Middleware Layer** implements all necessary components to realize distributed applications by providing the basic building blocks for implementing the architectures. It is based on the Ice platform, but also includes a shared memory channel for efficient data transfers. It abstracts the communication mechanisms, provides basic building blocks of the distributed application, and provides entry points for visualization and debugging. Here there is the ArmarXCore, that contains the base functionality for all other packages. With ArmarX RT a bridge to real time components can be established, as it is needed for accessing low level robot control modules.

**ArmarXCore** It is basically the application programming interface provided by the ArmarX Middleware Layer; made of four different layers, it is described in Figure 2.2.The Inter-object communication layer provides mechanisms for communication between objects in the system. The Sensor-Actor Units offer a generic interface for robot components, such as kinematics or camera. The ArmarX APIs follow an event-driven approach; the Observers layer is crucial because it monitors sensor values and generates events based on the data that is provided by the sensors. The events generated by Observers are sent to the Statecharts, which process the events and send the resulting control commands to the sensor-actor units that control the robot.

In the **Robot Framework Layer**, there are structures for providing a uniform interface among perception modules and/or memory structures. It consists of generic APIs that can be parametric and tailored to fit the scope of each project. Among the notable projects in the Robot framework layer are memory (MemoryX), robot and world model, perception (VisionX), and the robot API, required for basic Middleware layer building blocks.

**MemoryX** It is the ArmarX layer that manages the memory; it incorporates the building blocks for memory structures, both for system memory and storage of databases. The architecture is made up of working memory and long-term memory, both of which are accessible to applications. As well as this, there is the prior knowledge memory that allows already known data to be added to entities. MemoryX provides network-transparent access facilities for updating and querying memory.

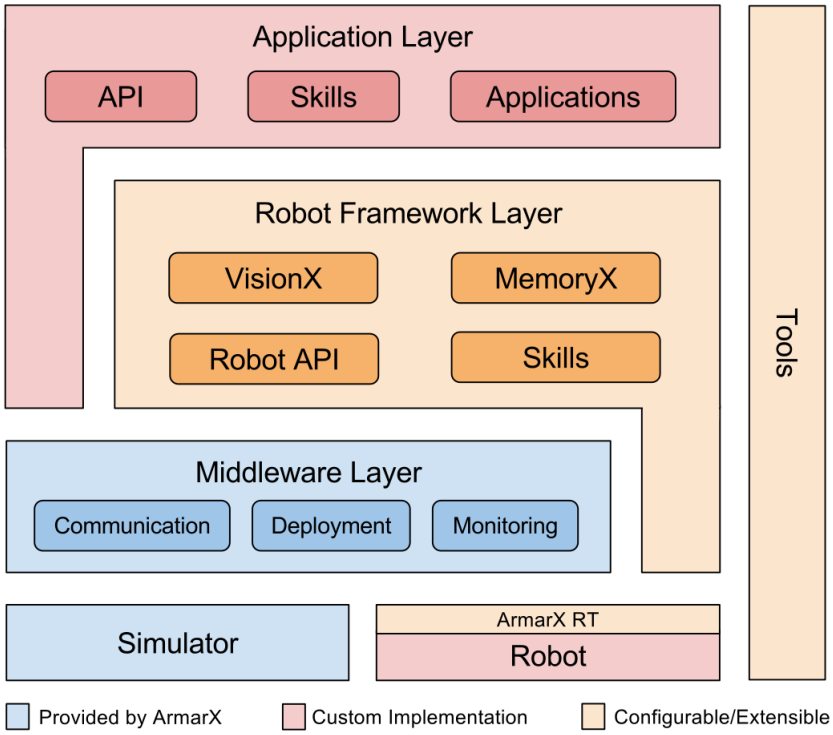



Figure 2.1: ArmarX organization structure from [4]: "ArmarX is organized in three layers. The Middleware Layer provides all core facilities to implement distributed applications as well as basic building blocks for robot software architectures. Based on these building blocks, the Robot Framework Layer provides a robot API implementing more complex functionality like kinematics, memory, and perception. Robot specific APIs can be implemented by extending the provided generic robot API modules. Robot programs are realized in the Application Layer. They are implemented as distributed applications, making use of the generic and specific robot APIs and statecharts. The ArmarX Tools comprise a plugin-based GUI that can communicate with the components and visualize their content. Specialized components can interact with the ArmarX Simulator or the robot hardware via ArmarX RT."

**VisionX** It is the ArmarX layer that provides tools and facilities for include and process images recorded from the cameras. There are two main components: the image provider and the image processor; the image provide that care of sending the visual data as a data stream, using shared memory or Ethernet technology; different image processors can be implemented, for different purposes like object detection, pose estimation, scene perception.

The final robot program, utilizing both generic and specific robot APIs is implemented as a distributed application in the **Application Layer**.

Humanoid robots have been developed by KIT since 1999; the goal is to have humanoids capable of undertaking a variety of tasks. The ARMAR robot series have been

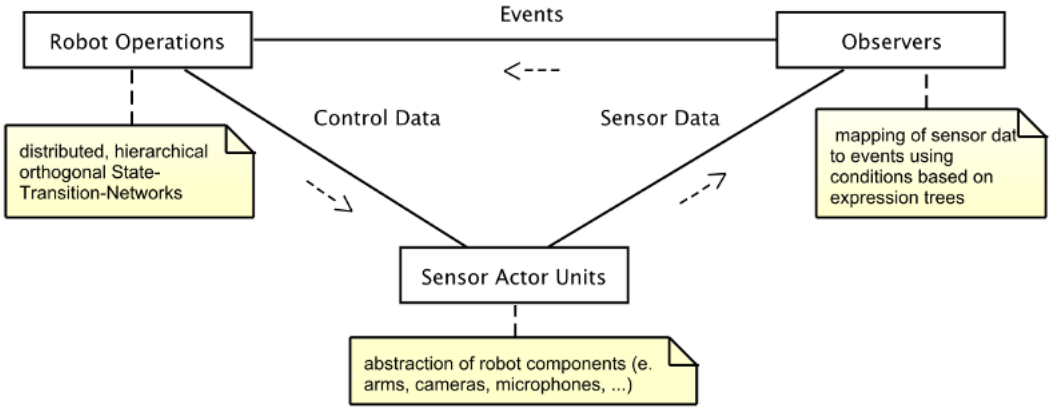



Figure 2.2: ArmarXCore layers and their interactions, from [4]: "the application programming interface provided by the ArmarX Middleware Layer comprises four different elements. The Sensor-Actor Units serve as abstraction of robot components, the Observers generate events from the continuous sensory data stream resulting in transitions between Operations. Operations are organized as hierarchical state-transition networks. These elements are connected by the communication mechanisms (arrows in the figure)."

designed for grasping and dexterous manipulation, and for learning through human observation and natural human-robot interaction. They are all based on the ArmarX infrastructure. In 2018 the ARMAR-6 [5], represented in Figure 2.3, was presented. It is a collaborative robot with highly integrated hardware and software; main features are robustness, reliability and modularity. It actuates 27 DOF: 8 actuated joints for each

Figure 2.3: The ARMAR-6 humanoid robot, from [5]

arm, 2 motors for the hands, 2 joints in the neck, 1 in the torso and 4 Mecanum wheels in the base (ARMAR-6 doesn't have legs, instead motion is possible thanks to a mobile base) From a software point of view, ARMAR-6 contains four identical computers. Each of them consists of a Mini-ITX motherboard with an Intel Core-i7 CPU, 32GB of RAM,

500GB of SSD storage, and a dual Gigabit Ethernet link. The four PCs serve different purposes: one is for real time control; a second one is used for vision related processes; the third one for motion planning and the last one for NLP purposes. ARMAR-6 is able to communicate with, act with and react to humans and their environment. Because of its constant interaction with humans, it is necessary for the robot to be able to perform some complicated tasks, like grasping known and unknown objects, bimanual mobile manipulation, activity recognition or recognition for the need of help.

# Chapter 3

# Activity Recognition

## 3.1 Problem Formulation

Human activity recognition (HAR) is a classification task that attempts to determine the activity performed by an individual. Due to its inherent complexity, HAR systems are often designed to be context-aware in order to reduce complexity. Some characteristics that HAR systems are usually required to have are being non-intrusive and non-collaborative. In other words, the task should be performed when the individual is completely free to act as he or she wishes. The problem is relevant and has different applications [6]. Let us analyze some of them.

**Smart homes** It is referred to a home automation system that monitors and controls home attributes in order to facilitate daily life; in this scenario, having data about the occupant's habits is crucial, because it allows to adapt the system to the specific behaviours of the people using it.

**Healthcare** Regular monitoring and recognition of physical activity can potentially assist to manage and reduce the risk of many diseases; moreover, as the average age of the population continues to rise, it is becoming increasingly useful to decouple the care of the elderly from the constant physical presence of human carers. Automating these processes would create benefits for society.

**Security** It is becoming increasingly important to use video surveillance to monitor security, production, and deter predatory and theft behaviors. For example, having an automatic system that in real-time processed video data in order to detect unusual human behaviours during an event, could help in detecting and preventing security-related problems.

**Cobot** Robots capable of performing domestic tasks, or industrial tasks by collaborating with human workers, are required being able to recognize human actions in the analyzed scenario.

In most instances, the problem is approached as part of the field of Computer Vision, using video sequences or images as inputs. A significant difficulty is the fact that there are multiple elements that may lead to classification errors, including partial occlusion, point of view changes, lighting and appearance. For these reasons, HAR is one of the most studied subjects in computer vision and it still remains a hot topic. Because of this complexity, HAR is usually addressed using advanced Machine Learning and Deep Learning techniques.

It is important, while addressing the HAR task, to keep in mind the distinction between action and activities. The term action, also called gesture or primitive, is often used to refer to a single movement (raising harm, closing hand etc.). When talking about activity, usually, the focus is on something more complex, that can be also seen as a union of different actions, which can involve objects and that is longer in time. This distinction is not unique in literature. A different, but still related, taxonomic approach is presented in [7]: here the activities are identified as sequences of motions that achieves several goals to accomplish one overarching purpose, while actions (also called functional movements) are motions that achieve a few goals to achieve a single purpose.

### 3.1.1 Action Recognition

A human body is a complex structure that is capable of performing an infinite number of actions. Our everyday lives are characterized by continuous actions, with no clear and defined distinction between one and another from a temporal and logical point of view: different actions can be performed in parallel, or as a stream, even without being the human able to explicitly segmenting them. It is this continuous nature of our world that makes it difficult to address the issue of automatic recognition of an action performed by a human. Additionally, each part of the human body is capable of performing different functions, either independent or connected to a singular purpose. As an example of the first situation, let's say that one hand is used to open a door while the other is used to hold something. An example of the second situation is when both hands are used to uncork a bottle.

The nature of algorithms and computer systems is not suitable for continuous problems; therefore, when considering the development of a model for action recognition, it is necessary to be able to discretize the performance of actions over time. This implies that different actions can be extracted from each sequence of variable length. For the analysis, the focus should not be on the entire environment, but rather on different parts of the body. In traditional approaches for action recognition, methods from the field of signal processing were employed: these models sought to extract domain-specific, rule-based features from the original data and apply statistical and machine learning models to these features. The drawback of this approach is that it requires expertise in signal processing and domain knowledge to analyze the raw data and engineer the feature set that will allow a model to be fitted to the data. This expertise is expensive and not scalable. Due to the expansion and improvement of deep learning techniques in recent years, action recognition has also improved considerably. In the literature we can find different points of view to this problem: an important work focus on video segmentation in order to separate different actions and then apply a classification algorithm on some

extracted features [8]; a different approach involves the study of motion flux to perform both action recognition and classification [9]. Another context-based approach studies the relative position of objects and body parts to make inferences about what the human is doing at that moment [10].

### 3.1.2 Activity Recognition

With an Action Recognition concept at its core, it can be realized that granularity of analysis can be reduced to a level that does not focus so much on the movements of individual body parts but on the purpose of an action series. Here, we are moving from the action level to the activity level. Activity is a complex concept that is difficult to define and standardise. It includes different states that can be statics, dynamics and/or transitions related. Also, the general Activity Recognition task can be faced from different perspectives. One of the most popular is activity classification from a video, which entails categorizing a segmented video in a supervised learning environment. There are many available datasets tailored for this purpose, like UCF101 [11], HMDB [12], Kinects [13], with videos taken from different sources like security cameras, movies, and others. Regularly, multiple activities can take place at the same time. Because of that, in its purest form, one could refer to the activity recognition problem as a multi-label classification problem. The analysis can be simplified by making assumptions, which in turn reduces the problem to a simpler multi-class classification problem, for which a finite vocabulary of actions can be defined. A different approach is activity detection, which is also known as activity localization. This task involves identifying the person performing a certain activity either temporarily or spatially in continuous video recordings. The logical next step to Activity Recognition in this development is activity prediction or forecasting, which involves predicting future activities based on current data. In this thesis, the focus is on a Human Activity Recognition problem in a supervised learning framework, considering mutually-exclusive classification.

## 3.2 Approaches

In machine learning, the choice of data to be used is crucial. Using one data source rather than another can significantly vary both the choice of model to be applied and the final performance. In the context of HAR, it is necessary to use data that enable the perception of activities by automatically extracting features that can be used in classification. For this, motion capture techniques are often used to generate the necessary information from the real world for the task. In motion-capture (also known as mo-cap or MoCap for short), objects or people are digitally recorded as they move. Given the fundamental nature of data collection as a whole, it makes sense to study the HAR literature on the basis of how the motion capture is carried out. According to the [14], there are three primary types of approaches to MoCap: vision-based, sensor-based, and RF-signal-based.

In the radio-frequency (RF) signal-based approach, we use radio signals to detect human pose or posture from an image or a video. Based on RF technology, movements can be tracked by detecting radio signals. Generally, the system will consist of stationary

sensor receivers, one (or more) transmitter marker tags, at least one stationary reference tag transmitter, and a processing system for analyzing the signals received. With this approach, generated signals are used to produce heat maps, which are then transferred to encoders for feature extraction. Pose decoding is then performed on these encoded features to extract key points, from which confidence maps can be generated for RF signals. Classification is performed by passing these confidence maps to a network that was trained using available data sets and that was already equipped with confidence maps for key points for visual inputs. The RF approach suffers from echo and noise; moreover, external nodes are required, limiting the user freedom of movements, and the estimate of the posture of the user is qualitative [15].

Afterward, the two other approaches will be examined: vision-based and wearable-sensor-based. It is also possible to categorize these two approaches differently, namely, as marker-based and marker-less approaches. This is due to the fact that video-based approaches do not require the user to wear anything, and that the analysis may also be conducted based on available sources and not on purpose recorded videos. The marker-based approach, on the other hand, requires the human to wear some tracking devices.

### 3.2.1 Marker-based

In the marker-based approach, MoCap is often achieved by analyzing the signals generated by sensors attached to the body. A body-worn inertial sensor is the most commonly used among the various kinds of sensor available, including force-based sensors, electromyography sensors, and others. The main reason is that they permit direct measurement of the body's movements. It is likely that accelerometers and gyroscopes will be widely utilized in this type of system; let us recall that the accelerometer measures the change in velocity and the gyroscope measures the 3-D orientation of the body.

Sensors generated signals are passed through the pre-processing layer in order to reduce any noise; then, a feature extraction pipeline is applied to the data, that are prepared to be used for the classification. In literature [16] is discussed that, for the HAR system to be truly useful in a real-life situation, it needs to not interfere with the subject's habits. Marker suits which require heavy or large sensor setups or require users to deal with uncomfortable devices are thus not suitable. In selecting which data sources an HAR model should utilize, it has to be taken into consideration that the more available data, the better function the HAR system will provide. While combining these two aspects is not trivial, the sensors industry has achieved remarkable advances, for instance with sports watches and other unobtrusive devices. In a marker-based system, due to the rapid emergence of wireless sensor networks, we can normally find, in addition to the wearable-marker data, object and environmental sensors providing additional information. Object sensors are attached to a particular object in the scene [17], and they are used in HAR as an indirect way for inferring human activity by detecting specific objects' movements. Typically, they are utilized less often than wearable sensors due to high costs and setup challenges. Environmental sensors are usually placed in the surroundings to sense accurate data on fundamental environmental parameters that can occur when physical activities are performed. They are extremely useful in certain applications (e.g. HAR for smart homes), in which having sensors such as microphones, light sensors, thermometers allows

to better track the human activity in a specific-activity-related context. Because of that, the choice of sensors and of the setup is highly dependent on the kind of actions that have to be captured [18].

Each type of sensor provides benefits and disadvantages. When different sources are used, it is necessary to combine them into a single model. A well-suited fusion step may also allow some of the problems related to each signal type to be solved. The fusion process becomes more difficult as the number of sensors increases. In general, sensor fusion techniques are typically based on Bayesian estimation, Kalman filters, and particle filters.

The strength of marker-based systems is found in the adaptability of the marker model. The precise and marker-point-based behavior of this model is extremely useful for tracking objects or specific body parts. One disadvantage of marker-based systems is the setup time, as well as the impact of skin movement and human error. Moreover, using sensors is problematic because markers tend to move or fall off and the repeatability is poor.

### 3.2.2 Marker-less

In the field of marker-less motion capture, full-body motion capture is achieved without the use of markers or special suits which have to be worn. Ideally, motion capture would only use one set of cameras from one angle, similar to human vision. A number of advantages can be derived from this approach, mainly related to its flexibility. The first advantage of video models is that they can be applied to heterogeneous data, not necessarily collected for the purpose of being used with a specific model (it is not unusual to find trained models among YouTube videos, for example); the second advantage is that software improvements are more readily available and cost less than hardware upgrades, which facilitates a faster improvement of the technology. Also, software upgrades enable re-use of data that has already been collected. Furthermore, the absence of setup times, such as those involved in installing sensor technology on the human body, allows for more data to be collected. In the literature, [19] it is possible to find a review of marker-less models divided on the type of representation each method uses; namely, it is possible to identify four categories:

- **Space-Time methods**: these methods leverage features with a space-time aspect; from this viewpoint, an activity is a 3D structure, which is basically a concatenation of 2D structures in time, or the third dimension. Local space-time features are the results of combining spatial and temporal data that represent the evolution of a phenomenon at a given location and time. As an example of how such features might be represented, see Figure 3.1. They have been found to be a successful representation of data for activity recognition. Different approaches to data extraction have been tested, from the Harris detector [20] to Gabor Filters [21] and Hessian matrices [22]. These methods serve as feature extractors, following which any algorithm can be applied, from KNN over SVM to Neural Networks. As a drawback of the space-time method, one disadvantage is that complex activities are more difficult to recognize, particularly when they have to do with small movements. Furthermore, the model tends to be point-of-view specific.

  Another important example in [24]: in this paper, the authors faced the problem

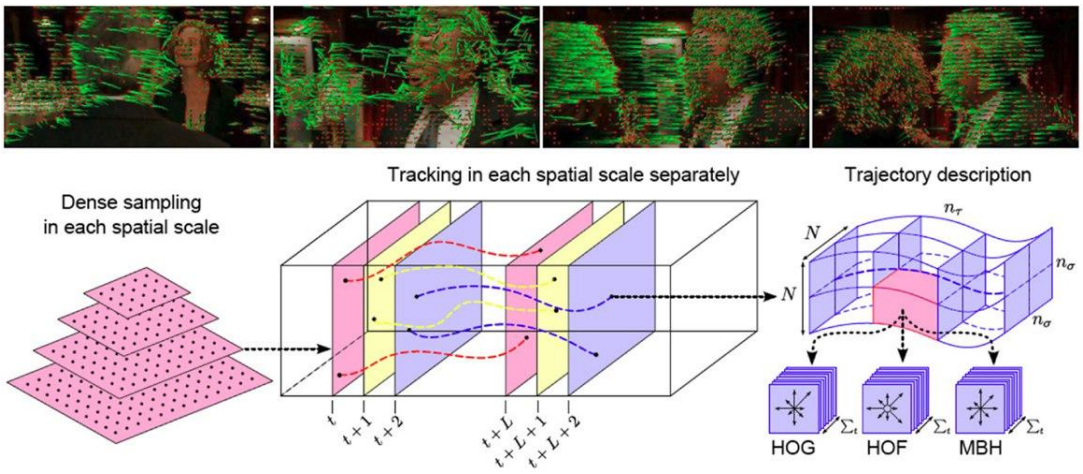



Figure 3.1: Visualization of human actions with dense trajectories (top row). Example of a typical human space-time method based on dense trajectories (bottom row) [23].

of view-invariant fine-granularity action recognition, focused on the upper part of the human body. As the list of primitives and the 3D samples for each of them are given, a motion context is created for each primitive. This context captures the information about the amount of motion, the direction of movement, and the location using a Histogram of Optical Flow representation. An approximation to spherical harmonics is used to construct a view-invariant representation. It is then proposed to approach the problem of classifying a test sample in two steps: recognition of motion primitives and recognition of gestures. Given the test sequence of frames and the list of primitives, each frame is processed by looking for the corresponding primitive. The set of them is retrieved as an array and, by using the Edit distance method, a gesture is predicted from this list.

- **Stochastic Methods**: the entity to be recognized may be considered as a stochastically predictable sequence of states. Models such as Markov models [25] and hidden conditional random fields [26] have been used to model human behavior as a stochastic sequence of actions. It is possible to describe each action using a feature vector, which encompasses information about the position, velocity, and local characteristics. Using these features, a stochastic model like HMM can be utilized to encode human actions, whereas recognition can be accomplished by identifying image features that are representative of human actions. Stochastic methods perform well but they are more complicated than non-parametric models.

- **Rule-Based Methods**: Essentially, a rule-based system applies human-made rules to store, sort, and manipulate data. To work, rule-based systems require a set of facts or sources of data and a set of rules for manipulating that data. Each activity is considered as a set of primitive rules/attributes, which enables the construction of a descriptive model for human activity recognition [27], [28]. Complex human activities cannot be recognized directly from rule-based approaches. Thus, decomposition into

simpler atomic actions is applied, and then the combination of individual actions is employed for the recognition of complex or simultaneously occurring activities

- **Shape-Based Methods**: Algorithms based on human silhouettes have recently gained considerable popularity in literature. From the dictionary, the silhouette is "the dark shape and outline of someone or something visible in restricted light against a brighter background". The silhouette may be defined at different levels of precision: it may be represented as a unique bounding box [29] or may represent the union of various bounding boxes [30] each representing a particular part of the body. For a more accurate representation, a 3D representation can be used, and in these approaches, it is also possible to use a single 3D box or a more advanced model. A very popular and advanced approach for defining a silhouette is the skeleton based one, which represents the silhouette as a bone structure constructed of joints and links [31]. A more detailed discussion of skeleton-based models is available in Section 3.3.

Even if it is now clear that different approaches for defining a silhouette can be used, it is possible to define a general pipeline, as presented in Figure 3.2. Let us analyze it. After the video has been loaded, a frames selection is applied, which selects from the entire video the frames that will be used for analysis. The model may, for instance, choose only to select a sample of the entire video, or ignore frames where no changes have occurred from the previous frames. Then, a background subtraction is applied. This is a problem of image segmentation, and it has been well studied in the literature. There are many available methods in literature that can be applied, and the results vary according to the context: the chosen method must be tailored to the data. Then, based upon the shape, it is necessary to determine the silhouette itself, that is, the bounding boxes, the joints, and so forth. After determining the silhouettes across frames, we need to prepare the data for the recognition process: it is common in this case to use velocity, position, angles, and accelerations as features; these can easily be derived from the evolution of the silhouettes in time. Lastly, the classification model is applied and inferences are drawn.

Even though it is difficult and arduous to define, the shape-based approach has various advantages: it allows for the creation of a view-invariant model, it can be applied to any video, and, by utilizing normalization techniques, it can also be subject-invariant.

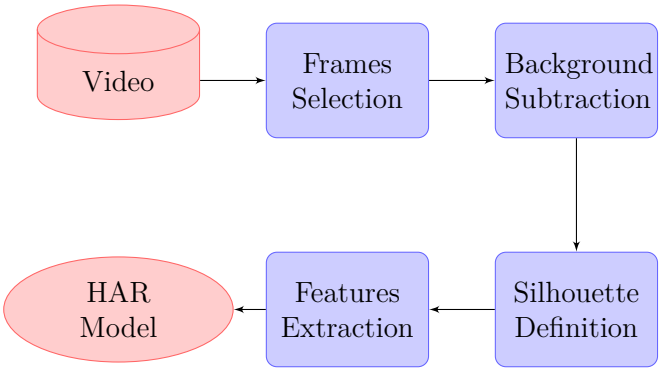



Figure 3.2: General pipeline describing HAR systems based on silhouettes representation

## 3.3 Skeleton-Based Methods

Earlier in this thesis, we discussed marker-less models and specifically shape-based models. The kinematic skeleton definition is one of the most widely explored approaches in the literature. It is also known as human pose estimation, and it can be addressed in a 2D or 3D environment. Human pose estimation is primarily concerned with extracting, from an image, the coordinates for a fixed set of points and links, each of which represents a particular part of the human body. We must first specify the exact joint set that we desire to identify, and also in this case different levels of precision may be required depending on the specific application: for certain tasks, identifying the hand as a single joint is sufficient, for others, multiple joints, one for each finger, may be essential.

As already said, the focus can be on either 2D or 3D points estimation. A difference between the two final results, taken from [32], is available in Image 3.3.

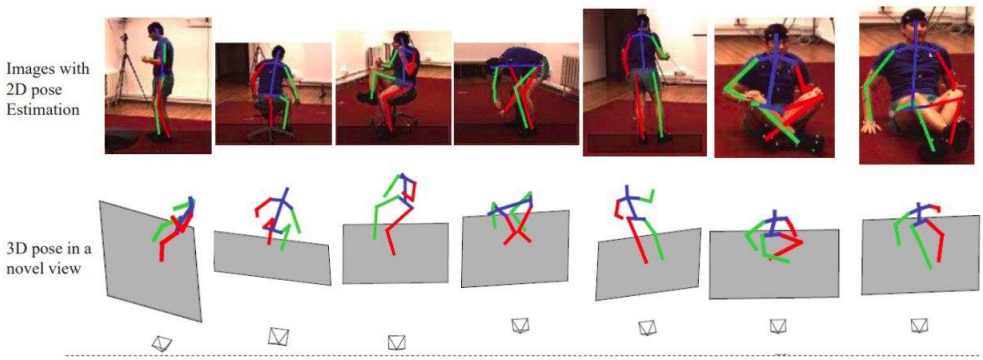



Figure 3.3: Differences among the results in estimating a 2D pose or a 3D pose [32]

Although 2D points estimation is a simpler problem than 3D, it still poses some challenges: for instance, point occlusion, which is the estimation of the position of a non-visible joint based on the location of visible joints. The estimation of a 3D pose requires the determination of the accurate spatial position of each joint in the third dimension. Beginning

with 2D data, many different ways can be taken to accomplish this. A possible approach to the problem is to view it as a regression problem in which one estimates the depth of a point based on its 2D representation. Alternatively, the task can be interpreted as a matching problem; given a dictionary of 2D $\rightarrow$ 3D poses, and a 2D pose $P$ for whom the 3D representation is necessary, the goal is to select the element from the dict that maximizes the similarity between $P$ and the 2D representation in the dictionary.

Various libraries have been developed for estimating human poses in the literature. Among these, OpenPose [33] is a popular and highly effective library. This technology was first developed in 2018 for the COCO Challenge. It had a number of important features such as real-time 2D and 3D key point detection as well as a calibration toolbox for estimating the camera's distortion parameters. The model is presented as a pipeline, in which an image is input and forwarded to a CNN with two branches, which jointly predict confidence maps. Then, part affinity fields (a set of 2D vectors encoding location and orientation) are derived. Using these data, the model creates sets of bipartite matching that are used to associate body part candidates. The last step is assembling full-body poses for each member of the group. A visual representation is available in Figure 3.4.

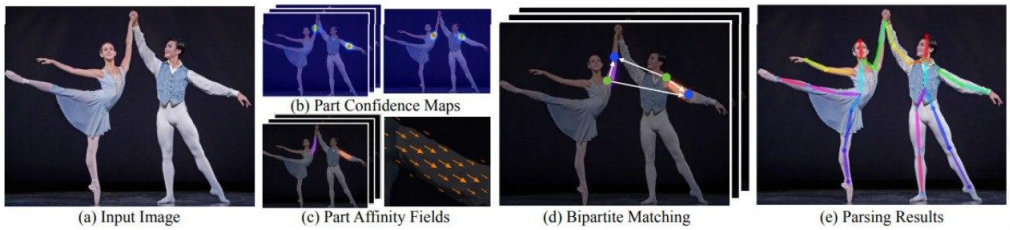



Figure 3.4: Pipeline of the OpenPose human pose detection starting from an image [33]: "Overall pipeline. (a) Our method takes the entire image as the input for a CNN to jointly predict (b) confidence maps for body part detection and (c) PAFs for part association. (d) The parsing step performs a set of bipartite matchings to associate body part candidates. (e) We finally assemble them into full body poses for all people in the image."

The 3D estimation is conducted not only from a software perspective, but also from the standpoint of the hardware department. Depth cameras have been suggested as an important solution. This type of camera contains a sensor which is capable of detecting both depth and color. The Microsoft Kinect, which is a component of the Xbox kit, is one of the most popular depth cameras on the market. Three hardware components are critical to the success of the Kinect: an RGB color VGA video camera, a depth sensor, and a multi-array microphone array. The video has a resolution of 640x480 pixels and a frame rate of 30 frames per second. A depth additional dataset derived from such a camera can greatly improve the 3D Human Pose Estimation algorithm [34].

### 3.3.1 Examples of skeleton-based HAR

**Pose Refinement Graph Convolutional Network for Skeleton-Based Action-Recognition** [35]: the authors proposed a model for skeleton-based action recognition that gradually fuses position and motion information. The pose refinement module reduces

the impact of pose estimation errors and further improves the accuracy at a small increase of computational cost. The network corrects pose estimation errors of the input sequence, and then uses an action flow branch and a temporal aggregation branch to generate a final prediction. An example of the pipeline is available in Figure 3.5

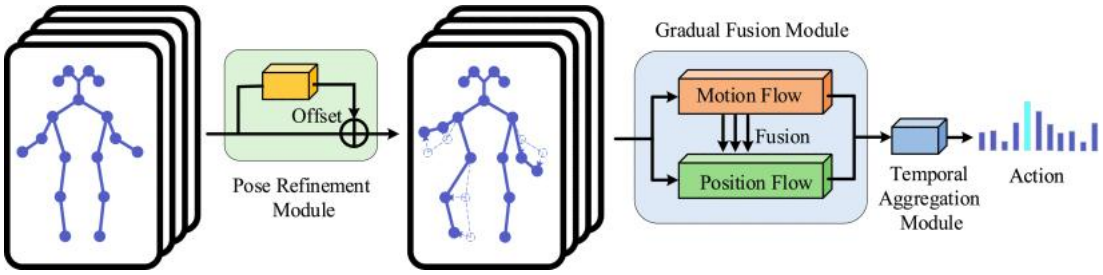



Figure 3.5: Pipeline of the Pose Refinement Graph Convolutional Network [35]: "Pose Refinement Graph Convolutional Network. The input skeleton sequences are first passed through a pose refinement module to reduce the impact of errors in the skeleton data. Then the refined skeleton sequences are fed into the gradual fusion module consisting of a motion-flow-branch and a position-flow-branch for fusing position and motion information. The position flow aggregates spatial information of skeleton joints at each time step whereas the motion flow captures the long-range temporal dependencies. Finally, the temporal aggregation module aggregates the information over time and predicts the action class probabilities."

**An Attention Enhanced Graph Convolutional LSTM Network for Skeleton-Based Action Recognition** [36]: the proposed AGC-LSTM network uses a shared LSTM to dispel scale variance between spatial and temporal features. It also uses attention mechanisms to enhance the features of key nodes at each time step, which can promote AGC-LSTM to learn more discriminative features. The model is based on a combination of graph and recurrent neural network, and it can be used to detect and recognize human-object interactions in images and videos.

**A Human Activity Recognition System Using Skeleton Data from RGBD Sensors** [37]: The proposed activity recognition algorithm starts with a set of skeleton joints and computes a vector of features for each activity. The authors find that many features are able to be extracted from the skeleton data representing the input to the algorithm. These features can be extracted by joint locations or from their distances, considering spatial information. A set of N feature vectors is created with a human activity constituted by N frames. In order to achieve the activity recognition goal, it is used a clustering algorithm that groups the feature vectors into clusters and gives the centroids of each cluster. Refer to Figure 3.6 for a scheme of the pipeline applied. The clustering algorithm used in this work selects the most informative postures from each sequence, instead of finding the nearest neighbor key pose for each frame constituting the sequence.

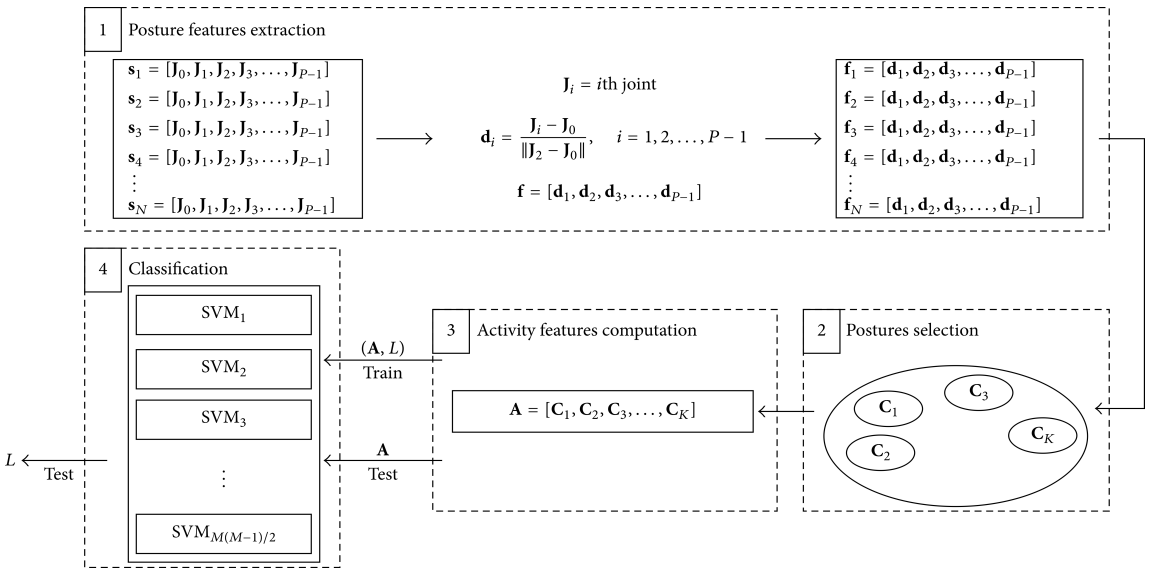



Figure 3.6: Scheme of HAR [37]: "The general scheme of the activity recognition algorithm is composed of 4 steps. In the first step, the posture feature vectors are computed for each skeleton frame; then the postures are selected and an activity features vector is created. Finally, a multi-class SVM is exploited for classification."

**Action Recognition Using Attention-Joints Graph Convolutional Neural Networks** [38]: the model faced the task of action recognition from skeleton images using attention-joints and a graph convolutional neural network. The process, represented in Figure 3.7, is the following. A residual attention network emphasizes the most relevant regions in an RGB-image, minimizing the number of network parameters. The human skeleton can be represented as a single feature vector or a spatial-temporal graph. According to the connectivity of the human body structure, joints in one frame are connected via edges to form an undirected graph. For each node, a set of features is defined: the nodes labeling procedure includes three types of features, dCoG, dAN, and flow features, namely OFF. dCoG is defined as the distances from the center of the body to the attention-joints, and dAN is defined as distances between neighboring attention-joints. The joint-flow (OFF) of an attention-node is computed in three levels: firstly for joints between two consecutive frames, then for joints between current frame and third-last frame, and lastly for joints between current frame and fifth-last frame. The classification is then achieved by using a geometric graph-based convolutional neural network; the joints within a single frame are represented as an adjacency matrix and self-connections as an identity matrix, and these data are passed through the network.

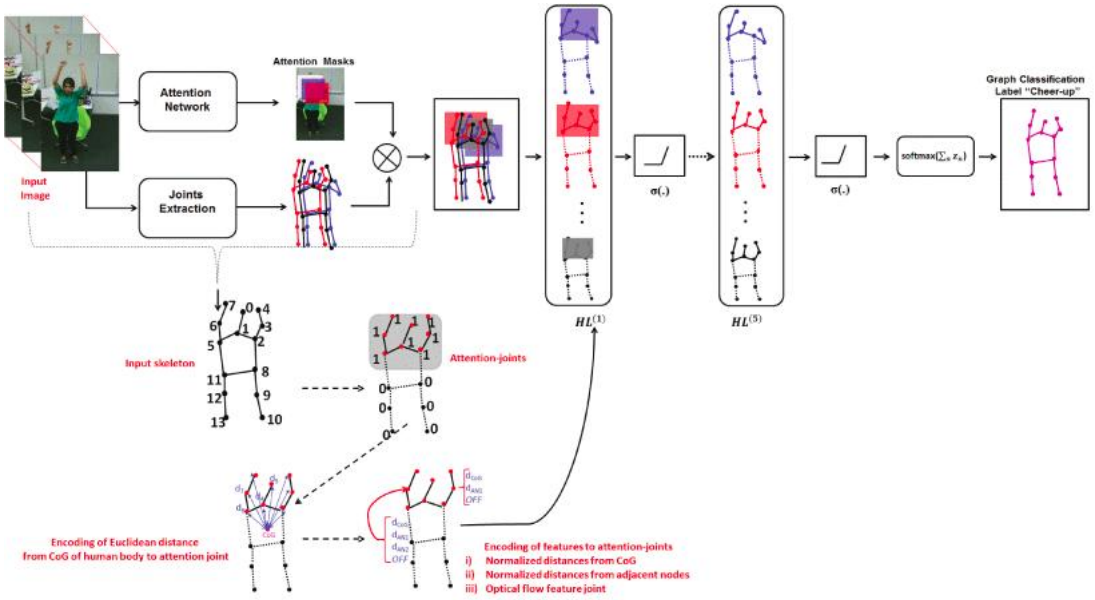



Figure 3.7: Schematic representation of the proposed network in [38]: "Block diagram of our proposed attention-joints graph convolutional neural network. Attention network is utilized to extract attention-joints of input skeleton. Then only the features associated with attention-joints are fed into graph convolutional network for classification."

# Chapter 4

# Graph Neural Networks

## 4.1 Graph Theory

To model the interconnection pattern in a network, mathematicians invented the concept of a graph. Let us emphasize three key aspects that are captured by the concept of graph:

- set of nodes $\mathcal{V}$, representing the units that constitute the network. The node is the base element of a graph, and it is a simple unit that can represent complex structures. For example, nodes can represent people, locations, and every kind of entities. In certain cases, it is possible to associate to each node some features, and to have another matrix that links each node to its related features.

- set of edges $\mathcal{E}$, pairwise connections among nodes where each $e = (i, j)$ is an ordered pair indicating an existing link going from node $i$ to node $j$, having $i, j \in \mathcal{V}$

- an eventual weights matrix $\mathcal{W}$, where each element is a positive scalar which represents the weight of a specific link, the strength of the connection. If $\mathcal{W}$ contains all ones, the graph is named unweighted and the definition of $\mathcal{W}$ can be omitted.

A graph can be represented by using the Adjacency Matrix: in this matrix, the nodes of the graph have been represented as row and column indices and are characterized by the fact that the elements $(i, j)$ of the matrix can only assume value $W_{ij}$ or 0; if the graph does not contain self-loops, the elements of the main diagonal are always zeros. From this matrix it is easy to retrieve the degree *Deg* of a node $i$; it is the number of edges connected to the node and, in terms of the adjacency matrix $\mathbf{A}$, it corresponds to $Deg(i) = \sum_j \mathbf{A_{i,j}}$, with

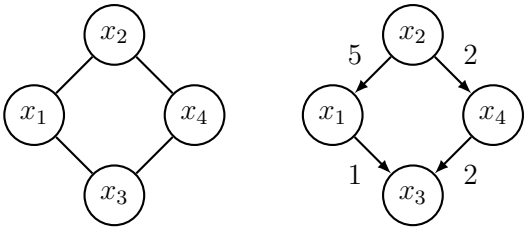
$$\mathbf{A}_{ij} = \begin{cases} 1 & \forall i \neq j | (i, j) \in \mathcal{E}, \\ 0 & \forall i = j \vee i \neq j | (i, j) \notin \mathcal{E}. \end{cases} \tag{4.1}$$

Another important and common representation is the node-link incidence matrix $\mathbf{B}$ of a graph $\mathcal{G}$; it is defined as in Eq.4.2 and represents, for each edge in the graph, its starting

and ending point.

$$\mathbf{B}_{ie} = \begin{cases} +1 & e = (i,j) \text{ for some } j \neq i, \\ -1 & e = (j,i) \text{ for some } j \neq i, \\ 0 & e = (i,i) \text{ or } e = (j,k) \text{ for some } j \neq i, k \neq i. \end{cases} \tag{4.2}$$

Even if the concept of an edge is intrinsically ordered, there are applications in which this concept loses its utility. It is the situation in which every kind of relationship is automatically bidirectional, having the same strength (or weight) in both directions. In this case, we have that link $(i,j)$ and link $(j,i)$ are always present with $W_{i,j}$ always equal to $W_{j,i}$. In this case, the graph is called undirected. When a graph is unweighted and undirected, it is called a simple graph.




(a) Undirected Graph (b) Directed Graph

Figure 4.1: *[Left]* Simple graph. *[Right]* Directed, weighted graph

Let's define, for a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, the notions of neighborhood, defined as the set of nodes $\mathcal{N}_i$ that are directly linked to node $i$, as defined in Eq.4.3. The concept of neighborhood can also be split, for directed graphs, into in-neighborhood $\mathcal{N}_i^+$(for links from other nodes to $i$) and out-neighborhood $\mathcal{N}_i^-$(vice-versa)

$$\mathcal{N}_i = j \in \mathcal{V} | (i,j) \in \mathcal{E} \tag{4.3}$$

In a graph it is possible to define, given two nodes, the concept of reachability; let's define the walk from node $i$ to node $j$ as a finite or infinite string of nodes such that there is a link between every to consecutive nodes and the string starts in $i$ and finishes in $j$. Then, a node $i$ is said to be reachable from $j$ is there exist a sequence among them.

The concept of centrality is very important in graph analysis for identifying the importance of nodes within a graph. Basically, the goal is to determine how important a node is in the context of the graph in which it is embedded; since centrality is a relative concept, it is not surprising that there are a variety of definitions for centrality. Various approaches or metrics can be used to define centrality; each one defines a node's importance from a different point of view and offers relevant analytical information about the graph. The simplest notion is the degree centrality, that defines the centrality $C_i$ of node $i$ as expressed in Eq.4.4

$$C_i = \sum_j 1 \; \forall j \in \mathcal{N}_i^+ \tag{4.4}$$

Reachability and centrality of nodes are fundamental in the study of flow evolution in graphs. In general, arcs in a network can be interpreted as channels through which goods flow. Goods can be, based on the applications, the number of cars on a road, or the amount of oil flowing in a pipeline and, in general, any measurable quantity. They can represent absolute values or relative values. There is a large number of real problems, in a wide variety of domains, which are effectively modelled as flow problems, from production scheduling to opinion dynamics. Studying how the flow evolves and is transmitted along the graph is a field of graph theory.

## 4.2 GNNs

In the real world, objects are often defined in terms of their mutual relationships. The connections between objects, and the objects themselves, can be expressed naturally as graphs. When used as a representation model for complex structures, graphs have the advantage of taking into account only relative information between the objects the represent. The graph network representation allows modelling topologically complex structures, of arbitrary size and eventually having a dynamic. A graph is inherently invariant with respect to changes in the absolute position of nodes: there is not a fixed node ordering or reference point, as it is possible to highlight in Figure 4.2 in which (a) and (b) represents an isomorphism of graphs. From there it is possible to notice that it is impossible to determine unambiguously a representation of a graph as a features vectors, because all the three proposal in the Figure 4.2(c) are equivalent from the graph perspective, but are not equivalent if for example we want to apply a KNN algorithm.

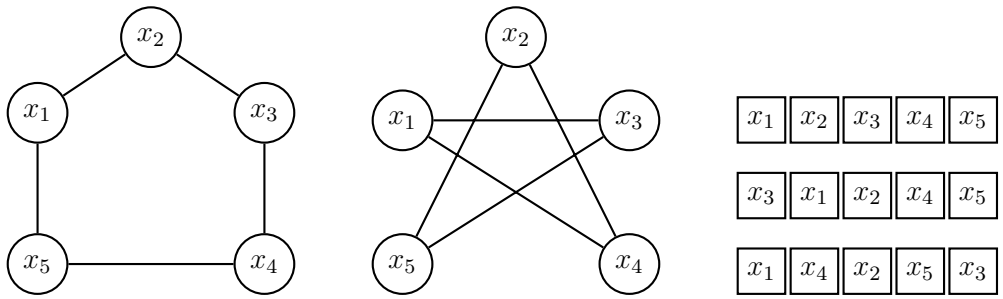



Figure 4.2: Two isomorphic graphs (a), (b), that are two graphs having the sane number of vertices, edges and also the same connectivity but different structures. In (c) there are three possible vector representation of nodes of graph (a)

Because of that particular structure, classical models from Machine and Deep Learning can't be applied to graph networks. Let us analyze in more details the reason: first, classical machine learning and deep learning rely on a concept of order that is not applicable here. Second, classical models need to have as input a fixed-sized representation of the image, but this is not applicable when we talk about graph-structured data. Third, in

graphs generally there is nothing such as a grid structure, that is for example the basic requirement for Convolutional Neural Networks to work. Given such a described structure for representing entities, we can be interested in performing different kind of tasks: node predictions, edge predictions, graph classification are examples of them. Starting from these points, it is clear that we need a new suitable way for analyzing graph data; this is called representation learning. Such problem has been approached in 2008, when the Graph Neural Network framework [39] has been proposed for the first time. Using all the information from the graph, including the node features and the connections, the GNN outputs new representations, or encoding, for each node and edge of the graph itself. In order to do so, one aims to map the nodes to a d-dimensional embedding space (low-dimensional space rather than the actual dimension of the graph), whereby similar nodes in the graph have similar embedding vectors in the new space. Embeddings contain information about other nodes in the graph in the form of structural and functional information. It encodes information about other node's features, connections, and context within the graph. The embeddings are then used to perform predictions. Each node, at each step, is influenced by the whole neighborhood $\mathcal{N}_i$.

Looking at Figure 4.3, taken from [40], we have a representation of a convolutional layer in a structured entity: the convolution on images is about striding a learnable kernel on a regular structure, which extracts the most important information. But the concept

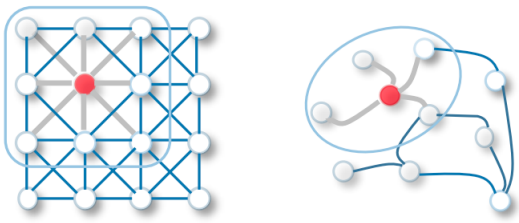


Figure 4.3: *[Left]*: Convolution on an image. *[Right]*: Convolution on a graph

of convolution can also be seen as combining all the information in a local area by using neighborhood data. In fact, on the right of Figure 4.3, it is shown a convolution on graphs; the idea of neighborhood is stressed, and we can see how the embedding is a function of the $\mathcal{N}_i$. The embedding process is applied simultaneously for all nodes at each step. This sharing mechanism is also called message passing, because the states can be seen as many messages passing backward and forward between the adjacent nodes.

Using the GNN, the number of layers is what determines how many neighbor hops we perform: this number is a hyper-parameter based on the graph structure. Small graphs can be learned using smaller GNNs, while bigger graphs can require more hops to include in the embedding all the relevant information. As in CNNs, just stacking as many layers as possible is not a good idea: in fact, it can lead to over-smoothing, which is the condition in which all the nodes are converging to the same value.

### 4.2.1 The message passing layer

The message passing layers are the core building blocks of GNNs and are those responsible for combining the node and edge information, together with graph structural information, in order to create the embedded values.

In order to do that, it is necessary to define a function that describes how relationships in the embedded space are related to relationships in the original features space. In other words, we need to learn a function $f\colon \mathcal{S} \to \mathcal{S}'$, as represented in Figure 4.4, that takes the graph and gives embeddings of individual nodes. Looking at those two embeddings, and having a similarity function $sim(a, b)$, that can be for example the euclidean distance, we want to define the embedding function such that $sim(x_1, x_2) \approx sim(y_1, y_2)$

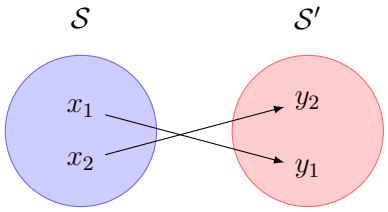



Figure 4.4: Mapping function from the original graph space $\mathcal{S}$ to the embedding space $\mathcal{S}'$

The embedding is obtained by combining an updating step and an aggregating one, as it is shown is Eq.4.5. These two steps needs to be differentiable functions, in order to allow error back-propagation that lead the network to learn the desired function. The pipeline for the graph encoding procedure corresponds to the Eq.4.5. Each node creates its own message starting from the embedding obtained at the previous layer (in the first step, it corresponds to the embedding); then, a computation graph is generated for the node in analysis by taking all the neighbors; for each of these, the "message" is prepared, i.e. a transformation is applied to their feature vectors. Then it is performed the aggregation of the messages of all the neighbours. Finally, the updating function is called to revise the current state of the node; this is done by using together the current node state vector and the result of the aggregation of the neighbours' messages. The basic aggregating function consists in averaging information from the neighbors, so that the function $\texttt{AGGREGATE}^{(k)}$ in Eq.4.5 became as described in Eq.4.7, where $h_u^k$ is the previous layer's embeddings. For the message creation point of view, instead, the basic $\texttt{MSG}^{(k)}$ function is a linear layer with its own weights, and it is described in Eq.4.6.

$$h_u^{(k+1)} = \texttt{UPDATE}^{(k)}(h_u^{(k)}, \texttt{AGGREGATE}^{(k)}(\texttt{MSG}(h_v^{(k)}), \forall v \in \mathcal{N}(u))) \tag{4.5}$$

$$\texttt{MSG}^{(k)}(h_v^{(k)}) = W^{(k-1)} * h_v^{(k-1)} \tag{4.6}$$

$$\texttt{AGGREGATE}^{(k)}(\mathcal{N}(u)) = \sum_{v \in \mathcal{N}_u} \frac{h_v^k}{|\mathcal{N}(u)|} \tag{4.7}$$

The $\text{UPDATE}^{(k)}$ function, instead, can be any kind of layer, from a simple linear layer to a more complex CNN. It takes as input the embedding vector of a node and updated it. So it takes as input the whole set of nodes $\mathcal{V}$ and returns an updated version $\mathcal{V}'$. When it is expressed as a linear layer, the updating function takes the form in Eq.4.8, in which $B_k$, is a trainable weight matrix and $\sigma(\cdot)$ is a non-linear function.

$$\texttt{UPDATE}^{(k)}(u^{k+1}) = \sigma(h_u^{(k)} + \mathbf{B}_k * h_v^{(k)}) \tag{4.8}$$

A GNN can be trained in either a supervised, an unsupervised or a semi-supervised environment.

For the Unsupervised Learning approach, only the graph structure is used, and the training phase assumes similar nodes will have similar embeddings. As a loss function, losses similar to those used for clustering in classical machine learning can be used: node proximity in the graph, or random walks.

The Supervised Learning task can be applied to either Node, Edge or Graph classification. It relies on losses like Cross-Entropy and similar. This approach requires to have labeled data.

In the Semi-Supervised Learning approach we have a mix of the two. Let's focus on Node Classification: in this case, part of the nodes in a graph are labelled, and part aren't, and we want to use all of them for the training phase.

The general idea of how a Graph Neural Network works has been clarified, but in order to have a model that is actually usable we need to switch from the just presented form to the matrix form. Let's define a matrix containing all the embedding layers at a certain step $k$, as $\mathbf{H}^{(k)} = [h_1^{(k}, ..., h_{|V|^{(k)}}]^T$. Then Eq.4.5 can be expressed in a general, matrix multiplication based form as is it in Eq.4.9. In this equation, $\mathbf{A}_u$ is the adjacency matrix as defined in Eq.4.1.

$$\sum_{v \in \mathcal{N}(u)} \frac{h_v^k}{|\mathcal{N}(u)|} = \mathbf{A}_u \mathbf{H}^{(k)} \tag{4.9}$$

Let $D$ and $D^{-1}$ be two diagonal matrices, one the inverse of the other, as described in Eq.4.10 representing the degree of each node.

$$\begin{aligned} \mathbf{D}_{u,u} &= Deg(u) = |\mathcal{N}(u)| \\ \mathbf{D}_{u,u}^{-1} &= \frac{1}{|\mathcal{N}(u)|} \end{aligned} \tag{4.10}$$

This allows to express the message passing model as a matrix multiplication, by the formula in Eq.4.11.

$$\mathbf{H}^{(k+1)} = \mathbf{D}^{-1}\mathbf{A}\mathbf{H}^{(k)} \tag{4.11}$$

So, by using the elements defined above, we can rewrite the Eq.4.5 as it is in Eq.4.12, where we can highlight two distinct terms: the first term represent neighborhood aggregation, the second one is the self transformation.

$$\mathbf{H}^{(k+1)} = \sigma(\tilde{\mathbf{A}}\mathbf{H}^{(k)}\mathbf{W}_k^T + \mathbf{H}^{(k)}\mathbf{B}_k^T) \tag{4.12}$$

### 4.2.2 Examples of GNN

GNNs have been extensively studied in the literature. It is possible to create new and different layers, independent of the general one presented here, through the combination of different message passing algorithms and other aggregation algorithms. Those layers can concentrate on a variety of aspects and address a wide range of graph network problems. In this section, let's present some famous examples of GNNs:

**Graph Convolutional Networks** : GCNs [41] were introduced as a method for applying neural networks to graph-structured data. It generalize the operation of convolution from grid data to graph data. The message-aggregation framework defined is the following:

$$h_u^{(k)} = \sigma(\sum_{v \in \mathcal{N}(u)} \mathbf{W}^{(k)} \frac{h_v^{(k-1)}}{|\mathcal{N}(u)|}) \tag{4.13}$$

Where:

- The message computation is written as $\texttt{MSG}_u^{(k)} = \mathbf{W}^{(k)} \frac{h_u^{(k-1)}}{|\mathcal{N}(v)|}$, where $\mathbf{W}^{(k)}$ is the matrix that contains the learnable parameters for the training step
- A sum over neighbors is used as $\texttt{AGGREGATE}^{(k)}$
- $\sigma(\cdot)$ is a non-linear function used to add some non-linearity to the model, with the same purpose as in CNNs.

**GraphSage** Theorised in [42], the GraphSage method addresses the situation where a graph is not fixed, but takes on a dynamic attitude that leads it to change its structure, or the situation in which the graph is dense and each node has a very high degree, that makes the computation as faced in GCNs too much expensive. In GraphSage the aggregation function is modified and it introduces the concept of neighbor sampling. The layer of GraphSage is built upon the GCN one, but it allows for different aggregation functions, not only the sum. Moreover, it also update the aggregate messages from neighbors by also concatenating the hidden representation of the node itself. The aggregation, shown in Eq.4.14, can be done by using a pooling layer and by applying $\gamma$ as the element-wise mean/max. In this representation, $\mathbf{Q}$ is transformation matrix, and it acts as a convolution operation that corresponds to the convolution in GCNs.

$$\texttt{AGGREGATE}^{(k)}(\mathcal{N}(u)) = \gamma(\mathbf{Q}h_v^{(k}, \forall v \in \mathcal{N}_u) \tag{4.14}$$

**Graph Attention Networks** The GAT layer [43] expands the basic aggregation function of the GCN layer by adding the concept of attention. It basically means assigning different importance to each edge through the attention coefficients, based on the importance we want it to have for the underlined task. It allow the network to learn some weights $\alpha_{ij}^k$ for each step $k$ and each edge $(i, j) \in \mathcal{E}$ that are used to weight the factors in the aggregation phase. The step-by-step process of how the attention weights are trained and used is described here. First, the layer embedding is linearly transformed as shown

in Eq.4.15. Then, a pair-wise un-normalized score is computed between each pair of two neighbors; here, the $z_i^{(k)}$ refers to the embedding of the $i$th node at $k$th step. A softmax normalization is applied in order to allow reading the result of Eq.4.17 as a probability distribution. Finally, Eq.4.18 corresponds to the GCN aggregation and provided the updated layer.

$$z_i^{(k)} = \mathbf{W}^{(k)} h_i^{(k)} \tag{4.15}$$

$$e_{ij} = \sigma(a(k)^T (z_i^{(k)} || z_j^{(k)}) \tag{4.16}$$

$$\alpha_{ij}^{(k)} = \frac{\exp(e_{ij}^{(k)})}{\sum_{l \in \mathcal{N}_i} \exp(e_{il}^{(k)})} \tag{4.17}$$

$$h_i^{(k+1)} = \sigma(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^{(k)} z_j^{(k)}) \tag{4.18}$$

### 4.2.3 Graph Nets from DeepMind

In [44], a group of researchers from DeepMind presented a new library, called Graph Nets, to address graph tailored deep learning. The basic element on which the library is built is the *graph networks* (GN) block. It is a graph-to-graph module that takes a graph $\mathcal{G}$ as input and, after a set of operations that can be user-defined, retrieves the updated graph $\mathcal{G}'$. Main entities are graph's nodes, edges and global attributes. The whole framework was developed having in mind three key principles: flexible representations, configurable within-block structure and composable multi-block architectures.

**Flexible representations** : both in terms of the representation of attributes and the structure of the graph. With regards to the attributes, it means that there are no limitations in the data structure used: DeepLearning typically relies on vectors and tensors, so these are clearly supported. But other data structures are also supported. Even from the viewpoint of output, the data retrieved by the graph block can be customized to meet the individual user's needs. Accordingly, when discussing the flexibility of the graph structure, it is intended that the graph block can handle both graphs with explicit relational structures, as well as those for which the structure must be inferred from the data itself.

**Configurable within-block structure** : Graph Networks can be configured in many different ways, allowing both inputs and outputs to be adapted to the needs of the task. The functions used for entity updating must refer to a function $f(\cdot)$ that determines the form of the input and output. Functions of varying complexity can be chosen, from the simple linear layer to very complex networks such as neural networks with recurrent structures. On the basis of the type of goal, it is also possible to define different types of interaction between the various entities. For example, one of the proposed solutions is the "Independent recurrent block", whereby each entity is updated independently taking into account only the current and previous state. Another example are the "Deep sets", which,

being sets, do not foresee the presence of arcs, and therefore the block is constructed in such a way as to manage the interaction only between nodes and globals.

**Composable multi-block architectures** : The graph network block, understood as the union of the updating and aggregation functions, can be interpreted as a layer in a convolutional network. In fact, once the input and output parameters have been defined, as long as they remain coherent on various levels, and exactly as in CNNs, it is possible to create increasingly complex networks through the use of several levels, communicating with each other. The union of the various layers can be sequential, thus creating a concatenation in which the output of the layer $k$ becomes the input of the layer $k+1$, simulating the evolution of the process of message-passing along the computational graph of the single nodes. The layers can be all the same or different, creating for example an encoder-decoder type structure.

A GN block is made of update functions $\phi$ and aggregate functions $\rho$, one for each entity; the expression is available in Eq.4.19, where $\mathbf{u}$ is the global attribute, $\mathbf{v}_i$ is the node's attribute and $\mathbf{e}_k$ represents edge's attribute. Each edge is identified by its attribute and by the pair $(r_k, s_k)$, that are respectively the receiver (the edge's arrive node) and the sender (the edge's starting node), . We also have that $V = \{\mathbf{v}_i\}_{i=1:|V|}$ is the set of nodes, $E = \{(\mathbf{e}_k, r_k, s_k)\}_{k=1:|E|}$ is the set of edges.

$$\begin{aligned} \mathbf{e}'_k &= \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}) & & \bar{\mathbf{e}}'_i = \rho^{e\to v}(E'_i) \\ \mathbf{v}'_i &= \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}) & \leftrightarrow \quad & \bar{\mathbf{e}}' = \rho^{e\to u}(E') \\ \mathbf{u}' &= \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}) & & \bar{\mathbf{v}}' = \rho^{v\to u}(V') \end{aligned} \tag{4.19}$$

When a graph $\mathcal{G}$ is provided as input to the GN block, the updating process is called and the model follows the operations described in Algorithm 1, as described in [44]. Starting from the edge embeddings, the updating function is applied to all the edges, allowing to retrieve the updated edges. Then, the new edges are used for the aggregation needed for updating the nodes. This allows to have the new node attributes, obtained after $N$ steps that corresponds to $N$ hops in the graph. Having the updated attributes for both the nodes and the edges, it is now possible to apply the updating function also to the global attributes. The new graph $\mathcal{G}'$ is then constructed by using updated nodes, edges and globals.

**Algorithm 1** Steps of computation in a full GN block

**function** GRAPHNETWORK((E,V, $\mathbf{u}$))
  **for** $i \in \{1...|E|\}$ **do**
    $\mathbf{e'_k} \leftarrow \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u})$
  **end for**
  **for** $i \in \{1...N\}$ **do**
    **let** $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,k=1:|E|}$
    $\bar{\mathbf{e}}' \leftarrow \rho^{e\rightarrow v}(E'_i)$
    $\mathbf{v}'_i \leftarrow \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u})$
  **end for**
  **let** $V' = \{\mathbf{v}'\}_{i=1:|V|}$
  **let** $E' = \{(\mathbf{e}_k, r_k, s_k)\}_{k=1:|E|}$
  $\bar{\mathbf{e}}' \leftarrow \rho^{e\rightarrow u}(E')$
  $\bar{\mathbf{v}}' \leftarrow \rho^{v\rightarrow u}(V')$
  $\mathbf{u}' \leftarrow \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u})$
  **return** $(E', V', \mathbf{u}')$
**end function**

# Chapter 5

# The AVAR Model

Starting from what has been defined in the previous chapters, let us outline the context in which the project was carried out. Given the intrinsic complexities of the task of activity recognition, some assumptions were made to simplify the problem, but these do not detract from the results obtained as generalisation to a more elastic model is still possible. First of all, it was chosen to work in a domestic environment, specifically inside a kitchen, in order to exploit the KIT Bimanual Manipulation Dataset created within the team itself; moreover, given the problems of tracking a single person in a context where several individuals are present, it was chosen to assume that only the robot and another person are present in the environment at any given time. As mentioned, this assumption is not particularly limiting since, especially in the context of robots used for elderly care, this is the most common scenario. Moreover, by applying facial recognition techniques that allow the assisted subject to be tracked even within a group of people, the hypothesis is easily removable. Another hypothesis, which is very reasonable given the context of application of the project, is that human beings and robots collaborate with each other. This hypothesis, especially in the analysis of audio data, implies that the human being does not try to confuse the robot, does not give him deliberately wrong directions and is aware of the robot's capabilities and limitations.

The aim of the the Audio Visual Activity Recognition (AVAR) project, as already mentioned, is to build an activity recognition system that acts in real time and can therefore be used directly by the robot. For this reason, the development of the project was carried out in two strands: on the one hand there is the creation and training of the offline model, with the tuning of the hyperparameters and the evaluation of the results on the basis of quantitative metrics. On the other hand, the creation of a pipeline that takes in input, separately, the audio and video data, analyses them in real time using the previously trained models and finally merges the results of the two in order to have a single final classification. This chapter will deal with the development of the different blocks of this pipeline, addressing both its development and offline training and its use in real time.

## 5.1 AVAR Complete Architecture

The data used for real-time HAR comes from two sources, which are the Microsoft Kinect for video data and microphones for audio data. These sensors are mounted on the head of the robot. Audio and video sensors operate in different ways. Since the camera records continuously, it provides a frame for every instant of time; each frame corresponds to the video recording at that specific moment. Differently, the analysis of audio data is event-based: input audio to the AVAR model is only sent when the speaker is speaking; this eliminates unnecessary data processing during those times when no valuable information can be extracted from the analysis. A temporal representation of how this data is collected and analysed can be found in Figure 5.1.

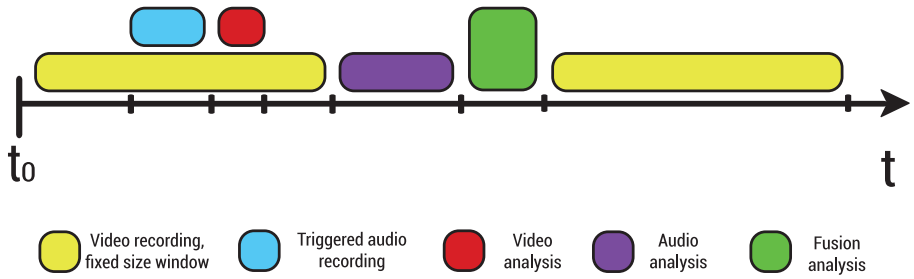



Figure 5.1: A temporal representation of how the AVAR model works. It is possible to distinguish two timelines: the audio line and the video line. In the video line it is possible to distinguish the fixed-sized recording window and the analysis time for the Vision Model. The audio line, being event-based, is triggered whenever the human being speaks; here, audio data are collected and passed to the Audio Model. Once the analysis of the visual data finished, if there is also audio data, the two results are merged in the Fusion Model and the result is the final classification.

Let us take a closer look at the timeline of video data, that corresponds to the lower segment in Figure 5.1. The idea of segmenting time by identifying when one activity ends and another begins is not a trivial task, often even for a human. Because of that, and since the entire project was designed with the lightness and simplicity of the model in mind, it was decided to pursue a simpler solution that still assures the goal to be achieved, even if with possibly less accurate results. Keeping in mind that the primary goal of the project is activity recognition and not primitive recognition, our interest is in a long time window for each instance: in fact, an activity takes place in the order of seconds. Therefore, the loss of a few milliseconds of video does not imply a significant loss of information that could invalidate the analysis. The point can be illustrated using an example of "sweeping" as the activity to be recognized: on a video of several seconds, even if the first milliseconds are lost, it is still possible to detect the performed activity from the rest of the video. Given these assumptions, the approach chosen for the segmentation of a single stream of

video data into various actions is the following: given a value $D_v$ which corresponds to the temporal dimension of the analysis window, we record up until the time of $t_0 + D_v$; then, the recording stops and the data are sent for analysis to AVAR. After AVAR returns the classification result, the process that collects the data restarts recording, which, in turn, goes from $t_0 + D_v + t_v$ to $t_0 + 2D_v + t_v$, where $t_v$ is the processing time for the visual model and corresponds approximately to some milliseconds (further analysis on that are available in Table 6.1. In Figure 5.1, the $D_v$ corresponds to the yellow segment and the $t_v$ to the red one. For simplicity, in this thesis the model that analyzed the visual data, namely the OpenPose skeleton, will be referenced as Visual Model and, correspondingly, the output from this model will be called Visual Output, but the reader should keep in mind that the output is a probability vector. The same approach will be followed also for the model that analyzes the audio data, called Audio Model and its output Audio Output.

When we examine the audio data, which is represented by the top segment in the figure, we can see that its arrival is not synchronous. Again, however, we have a recording time, which in this case is variable and corresponds to $D_a$, and a processing time for the audio model, $t_a$. In Figure 5.1, $D_a$ in the represented in blue and $t_a$ in orange.

Once the results of the visual model and, if any, the audio model are obtained, the two results are greased by the Fusion model, which applies a linear transformation to the sum of the two models by applying learned weights. In order to use the model in the way described, we require the three models (audio, video, and fusion) to be offline trained and then tailored for be used online. We will analyze separately, in the following paragraphs, the way in which each of the three models has been built and trained.

## 5.2 AVAR Visual Model

### 5.2.1 The Visual Dataset

To perform GNN training on visual data, the KIT Bimanual Manipulation Dataset [45] was used. The dataset was recorded to address the problem of recognizing actions performed by hands simultaneously. At this juncture, 2 different subjects performing 12 different activities while standing behind a table were considered. The dataset is designed for a finer granularity than task recognition, but nevertheless also provides indications of tasks performed at a higher granularity, such as sweeping, pouring and the like. For each record, therefore, both activity classification and sub-classification into actions are available. The 12 recorded actions are not present in the dataset in a balanced number. Below is the distribution of the number of instances across the various classes: *Scoop*: 256, *Pour*: 193, *Transfer*: 154, *Sweep*: 143, *Cut*: 134, *Wipe*: 128, *Stir*: 126, *Close*: 78, *Roll*: 67, *Mix*: 42, *Peel*: 42, *Open*: 42. Furthermore, it should be noted that some classes, such as Close, Open or Scoop do not fit completely into the defined scenario: in fact, they are carried out mainly using the hands, without involving the whole body as would be necessary for an action. For this reason, the 25-joint representation is not very suitable to classify them: the network remains confused by the stationary joints and is unable to generalize. This cannot be seen as a problem of the network, but as a poor fit of the dataset to the task. For this reason, it was decided in the design phase to reduce the

dataset to those classes that can actually be considered tasks, or at least involve a larger number of body parts. The selection of the classes to be kept was made in a qualitative way by analyzing which classes the network tended to confuse the most and noting that they were indeed those with the characteristics described so far. In the end a dataset having `num_classes` = 7 was kept; the selected classes are the following: *Sweep, Wipe, Cut, Stir, Roll, Mix, Peel.*

Even in the reduced dataset there is still an imbalance of classes. To solve this problem, a data augmentation model was developed, which creates new instances from existing ones. Let us analyze how this process worked.

**Data Augmentation** First of all, for each class is estimated the normal distribution of the position of the joint corresponding to the nose. From this, is obtained a list of `num_classes` distribution probabilities such that $\mathcal{N}_j(\mu_j, \sigma_j), \forall j \in (1{,}25)$. Secondly, a list of thresholds $T$ is defined, such that $t_i, \forall i \in (0, \texttt{num\_classes}$. The threshold was defined to be proportional to the number of elements present in the class itself. Thirdly, the data augmentation is performed for each element in the dataset as follows:

- given the element $k$, a probability $p_k$ is sampled from a uniform distribution.
- from the list of thresholds $T$, the $t_i$ corresponding to the class of the elements $k$ is taken, namely $t_i(k)$.
- if $p_k \geq t_i(k)$ then the new instance is generate by translating the human body pose in the camera frame of a value $\Delta$ along each axis. The value $\Delta = [\Delta_x, \Delta_y, \Delta_z]$ is defined as the distance, on each axis, between the coordinates of the nose in $k$ and the coordinates of the nose in an element extracted from the normal distribution defined above.

### 5.2.2 The Visual Pipeline: An Overview

The part of AVAR that deals with the analysis of visual data can be represented by means of a pipeline that, roughly speaking, consists of three blocks: pose refinement, feature extraction and classification using GNNs. Thus, the input data to this pipeline are the frames containing the skeleton data, while the output is a probability vector of size `num_classes`, where each element represents the probability that the corresponding activity is performed in those frames. Let us analyse each of the three blocks in detail.

### 5.2.3 Pose refinement

Given the frames recorded by the camera, the first step is to extract the human skeleton of the subject to be tracked. The Kinect records at a rate of 30 frames per second. Given a recording time window $D_v$ sec, this means that for each analyzed window we would have in input to the AVAR model $D_v * 30$ frames. From each of these, using OpenPose, we obtain for each frame a skeleton like the one in Figure 5.1, with 25 joints. It is obvious that, although OpenPose is a very accurate model and it is the state of the art, it is not yet perfect. In addition to some instability in the temporal evolution, another issue is that

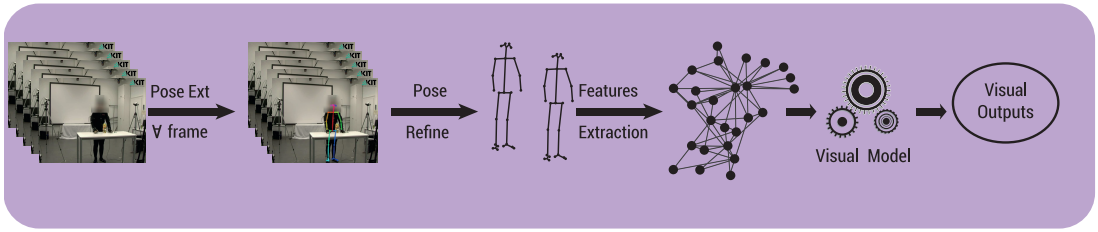
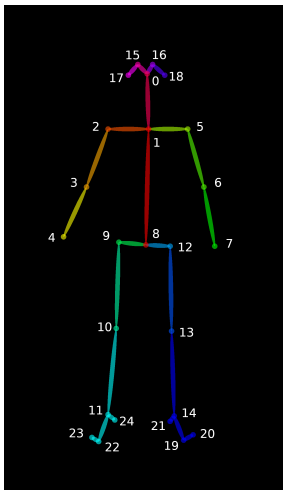



Figure 5.2: A diagram illustrating the visual data analysis pipeline. This model takes as input the frames corresponding to the video; first, OpenPose is applied to each of these frames to extract the human poses. Then, it performs a post-processing step of filtering and smoothing. Next, the data is embedded in a graph and sent to a model that returns a classification result.




| Node | Name |
|---|---|
| 0 | Nose |
| 1 | Mid shoulder |
| 2, 5 | Shoulder |
| 3, 6 | Elbow |
| 4, 7 | Hand |
| 8 | Mid Hip |
| 9, 12 | Hip |
| 10, 13 | Knee |
| 11, 14 | Ankle |
| 21, 24 | Heel |
| 22, 19 | Big toe |
| 23, 20 | Small toe |
| 15, 16 | Eye |
| 17, 18 | Ear |
| 25 | Background |

Table 5.1: Locations of 25 body key-points extracted from OpenPose and the used mapping between numbers in the image and body parts names

it often detects non-existent skeletons, although with low confidence, in random parts of the image. In order to address this problem, a post-processing technique is applied to the OpenPose result. This technique, through the application of a filter and a smoothing function, enhances the stability of the extracted poses. Here is a more detailed explanation of the two functions.

**Filtering** The problem of OpenPose detecting not-existing skeletons in the image is represented in Figure 5.3. The filtering method is intended to determine, from the possible various skeletons that OpenPose finds in the input image, the one that actually belongs

to the subject of interest. In order to accomplish this, the skeleton with the maximum confidence value of `min_conf` is taken from each frame. `min_conf` is defined as the minimum confidence of all 25 joints. The function, described in Algorithm 2, takes as input a list of poses and returns the selected single pose.

**Algorithm 2** Filtering on the OpenPose results

```
function FILTERING(S)
    selected ← None
    conf_selected ← 0
    for s ∈ S do                          ▷ S: set of extracted skeletons
        min_conf ← 0
        for j ∈ J_s do                    ▷ J_s: set of 25-joints of skeleton s
            if conf(j) ≤ min_conf then
                min_conf ← conf(j)
            end if
        end for
        if min_conf ≥ conf_selected then
            selected ← j
            conf_selected ← min_conf
        end if
    end for
    return selected
end function
```

**Smoothing** Smoothing prevents stuttering during pose transitions between adjacent frames. In fact even with an unaided eye it is apparent that joints can jump from one position to another between one frame and the next. For this reason, we smoothed the position using the algorithm described in the Algorithm 3. The smoothing is made over a sliding window of $N$ frames, and it returns, for each joint, the average position and confidence among the window. The model also addressed those situations in which a joint is missing for a specific frame, that can be because of OpenPose mistakes, and it downs its confidence by averaging for the total number of frames $N$.

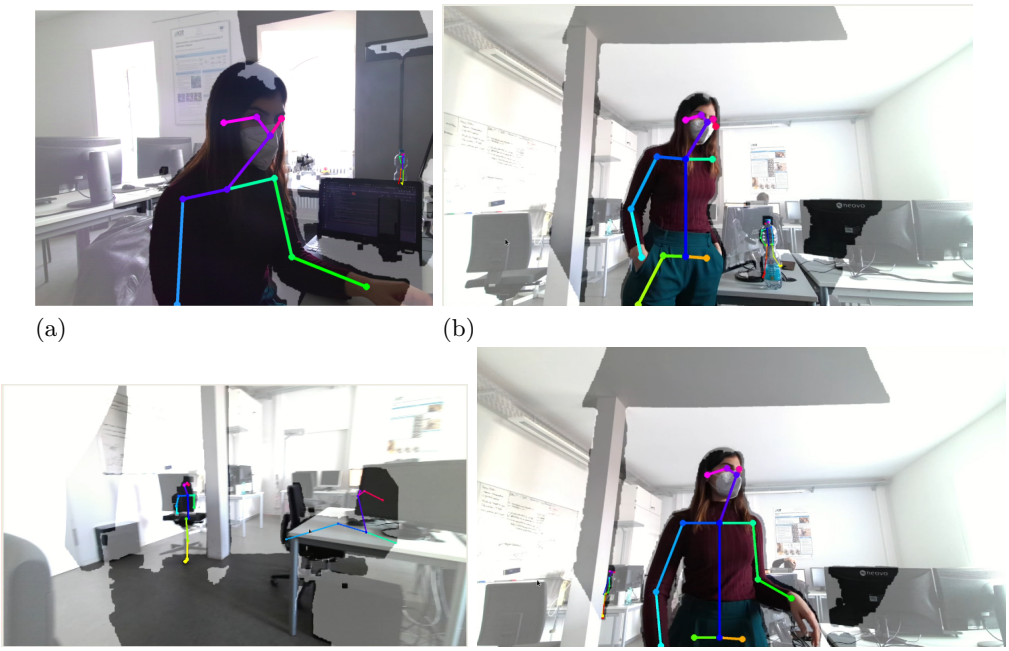
### 5.2.4 Features extraction

In the training phase of the visual model, one needs the shape of the data to be consistent with that of real time use. For this reason, the same post-processing techniques explained for the OpenPose outputs in real time were also applied to the training dataset.

Given then, for each instance, the set of filtered and smoothed poses, we need to apply feature engineering techniques in order to create the data that will then be passed to the classification model. For each set of filtered and smoothed pose data, we will apply feature engineering techniques in order to produce the data that will be passed to the classification model. Recall that for this project we chose to use GNNs, which, as discussed in Chapter

**Algorithm 3** Smoothing on the OpenPose results

**function** SMOOTHING($s_1, ...s_N$) ▷ $s_i$: $i$-th skeleton in a window of $N$ frames
  `smoothed` ← `None`
  `conf_smoothed` ← 0
  **for** $j \leftarrow 0$ to 25 **do** ▷ $J_s$: set of 25-joints of skeleton $s$
    `new_pos` ← 0; `new_conf` ← 0;
    `cnt_joint` ← 0
    **for** $i \leftarrow 1$ to $N$ **do**
      **if** `exists`($s_i[j]$) **then**
        `cnt_joint` ← `cnt_joint` + 1
        `new_pos` ← `new_pos` + $s_i[j]$.`pos`
        `new_conf` ← `new_conf` + $s_i[j]$.`conf`
      **end if**
    **end for**
    `smoothed`$[j] \leftarrow \frac{\texttt{new\_pos}}{\texttt{cnt\_joint}}$
    `conf_smoothed`$[j] \leftarrow \frac{\texttt{new\_conf}}{N}$
  **end for**
  **return** `smoothed, conf_smoothed`
**end function**



(a) (b)

(c) (d)

Figure 5.3: Four examples of real-time use of OpenPose in which there are fake skeletons in random part of the image. In Figure (a) it is possible to see a skleton in the bottle on the right. In Figure (b) a fake skeleton in the the bottle on the table. In Figure (c) two fake skeleton are detected, one in the chair and the other in the table. In Figure (d) a fake skeleton in drawn in the wall pill in the left.

4, are well suited to represent human pose information. It is essential that both spatial and temporal information be included in the graph used as input to the network. In fact, it is necessary to consider both these two elements while classifying the performed action: not only the instantaneous pose, but also how it evolves over time. The data is therefore represented as a graph structure composed of nodes, edges and global values. Detailed discussion of the three entities will follow.

**Nodes** A graph is composed primarily of nodes. Each pose joint corresponds to a node of the graph. A vector of features is associated with each node, which will be used by the model to distinguish among the various activities. A node from OpenPose is associated with a timestamp and $x, y, z$ coordinates corresponding to its position in the frame of the camera. From the instantaneous position and time, it is quite natural to derive the concept of velocity, which we remember being defined as the rate at which the position changes. In Eq.5.1, we can see that for a given timestamp, the velocity can be computed as function of the changes in point position from the previous stamp to the actual one.

$$\mathbf{v}(t_1) = \frac{\mathbf{x}(t_1) - \mathbf{x}(t_0)}{t_1 - t_0} \tag{5.1}$$

Based on these data, we define a vector of features, represented in Figure 5.4, associated with each node.

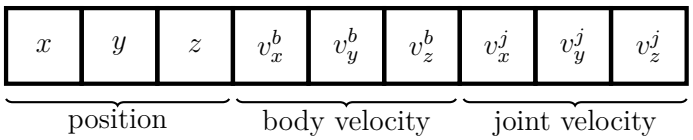



Figure 5.4: Representation of the features vector that characterizes each node in the graph structure used for embedding the video data

As clear from the figure, it is possible to split the features into three groups. Let us analyze them:

- **Position**: these three elements are the coordinates obtained via OpenPose in the camera frame.
- **Body velocity**: it represents the change in position in the body with respect to its central axis, intended as the line dividing the body vertically into equal right and left haves. The aim is to identify those movements, such as walking, that generate a velocity on all joints of the body.
- **Joint velocity**: is the difference between the total velocity of a joint and the velocity of the body in the same direction. It identifies single movements of the joint, such as lifting a hand to demand attention.

**Edges** : they are used to describe relationships between nodes. Within this context, there are two types of relationships: temporal and spatial. As there is no need to include other types of information within the edges, it was decided to incorporate them as a binary channel. This would guide the learning process by indicating the type of relation existing between two nodes. Therefore, the existing edges are: from a temporal perspective, the connection between the same joint at the previous and subsequent time snapshots of the analysis; from a spatial perspective, the physical links between various joints seen in Figure 5.1.

**Globals** : considering activity recognition is a classification at graph level, the globals that we are using should reflect the ground truth of our analysis. Therefore, for each graph it will be initialised as a vector of size `num_classes`, which represents the one-hot encoding of the activity actually performed.

### 5.2.5 Classification

The graph constituted as described so far is used as input to a classification model, which aims to learn to distinguish between one class and another. The model used to achieve this goal, represented in Figure 5.5, includes three components. Let's analyze each of them.

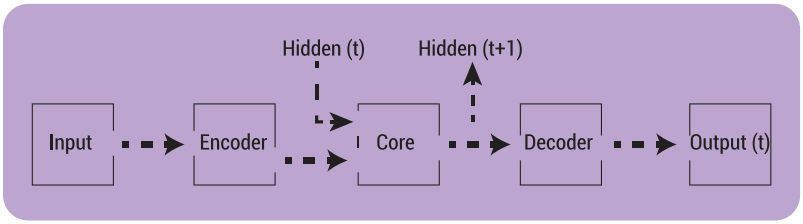



Figure 5.5: The GNN used for analyzing the visual data. It is an encode-process-decode model: it takes as input a graph, maps it into a latent space by using the encoder module, then applies $m$ message-passing steps and finally return the output by decoding the graph at the $m$-th step.

**Encoder** An encoder graph network, which independently encodes edge, node and global attributes. Independently in the sense that each of the three encoding is created independently without considering the relationships between elements. The encoder is based on the `GraphIndipendent` model from GraphNets [44], that acts as follows:

$$\begin{aligned} \texttt{encoding(e)} &= f^e(\mathbf{e}) \\ \texttt{encoding(v)} &= f^v(\mathbf{v}) \\ \texttt{encoding(u)} &= f^u(\mathbf{u}) \end{aligned} \tag{5.2}$$

The Encoder function is applied to nodes and globals in order to expand the feature space by the $f : \mathbb{R}^d \to \mathbb{R}^{d'}$. As represented in Figure 5.4, in fact, the space of the nodes is $d_n = 9$; by encoding it, we want to map these into a latent space of dimension $d'_n > d_n$. The global space, on the other hand, is $d_g = \texttt{num\_classes}$; this too is mapped to $d'_g > d_g$.

For edges, however, the situation is different. As previously explained, the edges in the graph under analysis can only take on two values, namely (0,1) if the given relation is spatial and (1,0) if it is temporal. Precisely because of this intrinsic characteristic of binary channels, during the entire process it is necessary that their value never changes: a temporal (or spatial) edge must always and only indicate this specific characteristic. So in this project the function $f^e$ is not used. The specific functions used for the node and globals encoding are discussed in Section 5.5.

**Core** A Core network of graphs, which performs $N$ rounds of processing (message-passing) steps. The input of the Core is the concatenation of the output of the Encoder and the previous output of the Core (labelled hereafter as `Hidden(t)`, where $t$ is the processing step). This is the real fulcrum of the GNN, as the output of the Core will be dependent not only on the input, but above all, on the inter-current relations among the different entities.

As mentioned, the Core function is applied several times: the $m$ hyper-parameter defining the number of steps is strongly related to the structure of the graph. It represents how far each node will "look" in updating its features: a value of $m = 1$ indicates that

each node is only affected by its direct neighbours, while higher values add jumps in the graph.

The block used for the core is a modified version of the Full GN Block presented in Figure 5.6. As can be seen, the original block applies all the functions $\phi$ and $\rho$ described in Eq.4.19. Since, as described for the encoder, for this project it has been decided to

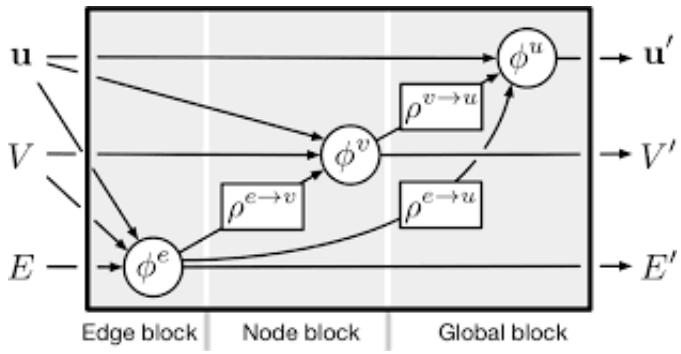



Figure 5.6: The full GN block as described in [44]: is shows how both the updating and the aggregating functions are performed among the three entities nodes, edges and globals.

keep the features of the edges fixed, this means that the function $\phi^e$ is a linear function that does not depend neither on **v** nor on **u**. The updating functions may be of various types; aggregation, on the other hand, is used by nodes and globals to aggregate nodes and edges. In *Graph Nets*, two functions are suggested: a sum function and an average function. Further details about the results and the impact the choice has to the model performances are available in Section 5.5.

**Decoder** A network of decoder graphs, which independently decodes the edge, node and global attributes (it does not calculate the relations, etc.), at each message step. Again, as with encoding, the three entities are analysed separately and mapped in the output space. The applied functions are consistent with those in Eq.5.2, but in this case the dimension of $d_n$ and $d_g$ depends on the output dimension of the Core layer. Decoding is applied to each of the $m$ steps, but only the last of these will actually be passed to the final dense layer to extract the probabilities to be used in the classification. The same functions can be used for the decoding function as for encoding, as long as they are followed by a final layer that remaps everything in the final result space. In the case of the project under analysis, since it is a graph classification problem, the final space will be a vector of globals of dimension `num_classes`.

### 5.2.6 Implementation Details

Both the encoding of the encoder block, the encoding of the decoder block and also the embedding function of the central block require features to be mapped from one space to another. Thus, although different in size, the same type of network can be used as an implementation for both. Several combinations of networks have been explored in this

thesis; we define the basic structures in this section, then in the 5.5 section we will study the results obtained with each of the combinations. The two models that were utilized in this thesis were a Normalized Multilayer Perceptron and a Convolutional Neural Network for tabular data. Let us examine them in more detail.

### The Normalized Multilayer Perceptron (NMLP)

Multilayer Perceptrons, represented in Figure 5.7 are a class of feed-forward artificial neural networks that consist of more than one perceptron. The feed-forward neural network is an artificial neural network constructed without internal recurrence, that is, where information travels directly from the input layer to the output layer. The perceptron is a

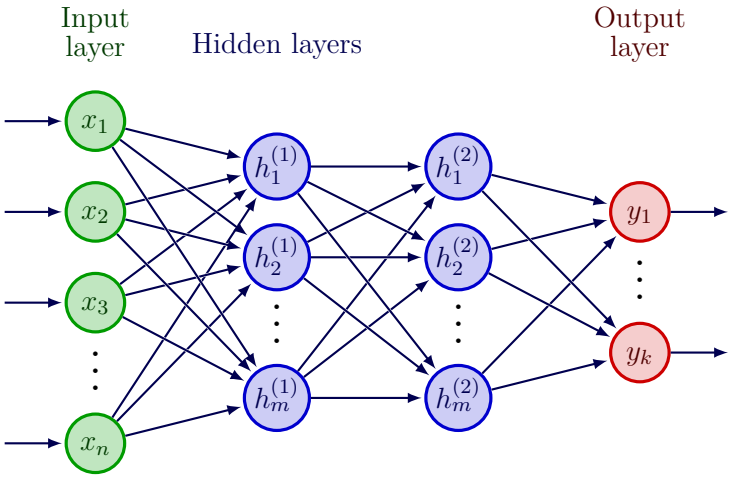



Figure 5.7: A representation of Mulilayer Perceptron; it has $n$ input units and $k$ output units and the learning process in made by using 2 layers and $m$ neurons

basic unit for a neural net. It produces a weighted linear combination of input, weights and biases, though the function defined in Eq.5.3. The aim is to obtain a separation of hyper planes that divides the classes through the learning of opportune weights and biases. Therefore, by combining different perceptrons and using activation functions in order to enable learning nonlinear models, the MLP is obtained.

$$f(x) = \sigma(\langle w, x \rangle + b) \tag{5.3}$$

The common activation function historically has been a sigmoid, but in recent developments of deep learning the rectifier linear unit (ReLU) is more frequently used, and it is also the function used in this thesis. It is defined as $\texttt{ReLu}(x) = x^+ = \max(x,0)$. In order to construct the MLP, you need to specify both the number of layers and the number of neurons per layer. Additionally, in the version implemented for the thesis, a normalization function is applied after each layer. The chosen function is a technique known as batch normalization [46]. Using the distribution of inputs to each neuron, a mean and variance are calculated, which are then used to normalize the input to that neuron on each training trial.

### The CNN for Tabular Data (TabCNN)

Many Computer vision problems are nowadays most commonly addressed with CNNs [47]. The effectiveness of CNNs on image and video data arises from the fact that they consider the spatial structure of the data, capturing spatially local input patterns. Due to the way convolution occurs, the image can be analyzed by extracting features that will then be combined with others in subsequent layers, exploiting local connectivity and spatial locality. Because spatial locality determines that the points where the convolution kernels are applied are highly correlated, processing them together permits meaningful feature representations to be extracted. Local connectivity means that each kernel is connected to a small region of the input image when convolution is performed. In this, CNNs have demonstrated an incredible learning ability.

However, when it comes to tabular data, which is a very common type of data, this concept of spatial locality and local connectivity is missing. The use of this powerful structure, however, is tempting, and some researchers have recently proposed solutions to circumvent the problem, such as transforming tabular data into something like an image, so that the convolution [48] can be applied. The work done in this thesis is similar. The

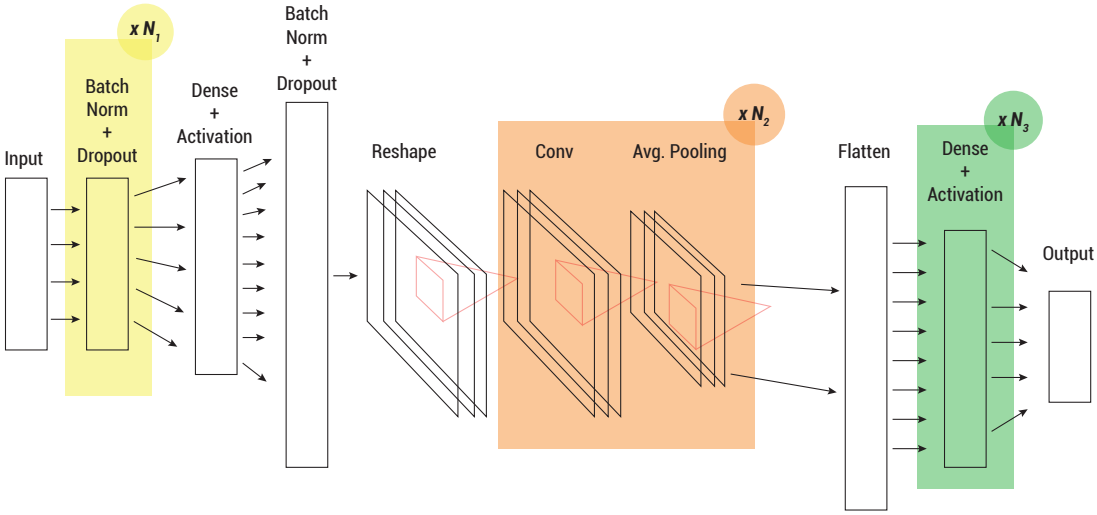



Figure 5.8: A representation of how the Convolutional Neural Network for Tabular Data works. The input is passed as vector, it is expanded by using $N_1$ Dense layers, each followed by its activation function. Then, the obtained vector is reshaped and passed through $N_2$ convolutional layer followed by pooling layers. The result is flatted and, by using $N_3$ Dense + Activation function layers, it is reduced to the needed output dimension.

model is presented in Figure 5.8, and it shows how we break the problem down into two parts: starting from the tabular data, an expansion of the features is created through a concatenation of dense layers. These layers are then reorganised to take a matrix form. The matrix is then treated as a normal input to a CNN. The internal CNN constructed

for this task is quite simple: 3 convolutions, interspersed with normalization, activation function and dropout.

## 5.3 AVAR Audio Model

### 5.3.1 NLP Introduction

The aim of natural language processing (NLP) is to provide computers with the ability to understand text and spoken words in the same way as humans. A non-exhaustive list of NLP applications is as follows: sentiment analysis, text categorisation, machine translation, question answering, topic modelling, text summarization. Texts can be analysed at various granularity, depending on the basic entity chosen (characters, n-grams, words, sentences, and so on). Usually, NLP projects consist of several steps. If the data is provided in the form of audio, the first thing that is done is the transcription into text. After that, pre-processing is applied to clean the data. Among the most common pre-processing steps are the removal of stop words (very common words that do not carry any information), the tokenization step (which divides raw-text into sub-units) stemming or lemmatization (which aim to return each word to its basic form) and Part-Of-Speech tagging, which annotates the test by indicating for each unit its role in the sentence.

Once the clean text has been created, several techniques have been developed to achieve the set task. One of these involves the creation of lexical chains that analyse the relationships between textual units; in this way, a text can be represented by a hierarchical structure [49] [50]. Another involves the creation of embedding vectors for each unit; using this approach, each word can be represented numerically, allowing it to be analysed by a machine.

### 5.3.2 The Audio Pipeline: An Overview

The audio data stream, unlike the video data stream, is not continuous. The approach chosen is an event-based model that triggers the sending of data only when it is actually present. Given the purpose of the project, working directly with the audio data to carry out the classification would not be productive; it was therefore preferred to carry out an intermediate step of transcribing the audio data into text. This allows the classification problem to be tackled using Natural Language Processing (NLP) techniques. In the NLP field, the present problem turns out to be text categorization: it consists in assigning a label to a text, it is a supervised process and can be solved both with deep NLP techniques and with NLP rules. A representation of the steps in the audio analysis pipeline can be found in the Figure 5.9. There are two blocks representing the textual analysis: "Root Extraction" and "Classification".

### 5.3.3 Root Extraction

In the context of this project, it can be assumed that the sentences arriving as audio input are short, simple and direct sentences. We can assume that it is not in the context of this thesis that the human make use of complex, misleading or contradictory sentences.

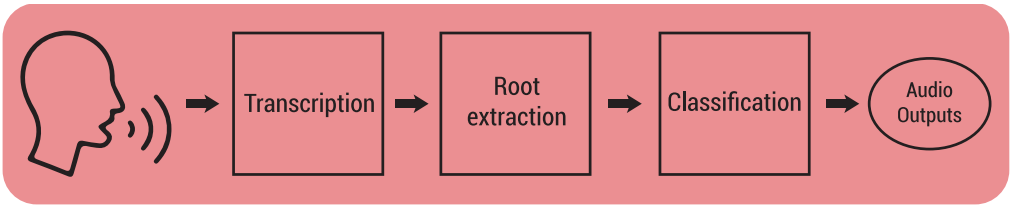



Figure 5.9: A diagram illustrating the audio data analysis pipeline. This model takes as input the recorded audio; first, a speech-to-test model is applied that transcribes the audio data in a textual form. Then, from the dependency tagging it is extracted the root of the sentence. Finally, the root is used for the classification and the model provides as output the probabilities for each class.

Based on this, each sentence is expected to consist of a main verb, subject and object. This means that it is plausible to expect a sentence like *"I am going to sweep the kitchen"*, whereas sentences like *"Yesterday my daughter found a ring while sweeping the kitchen"* would not be in the context.

A computationally simple way of analysing sentences of the type specified involves creating the structure of hierarchical dependencies between the various words. The root of this structure will be the main verb in the sentence, that is, the heart of the sentence itself. The creation of the hierarchy is called dependency parsing, in NLP. The task of dependency parsing involves extracting a dependency parse from a sentence that represents its grammatical structure and identifies the relationship between "head" words and words that modify those head words. A dependency parser analyzes the grammatical structure of a sentence based on the inter dependencies between the words in that sentence and assigns to every word a dependency tag. Each tag signifies a specific relationship between two words in a sentence. For each sentence spoken by the subject, therefore, it is tran-

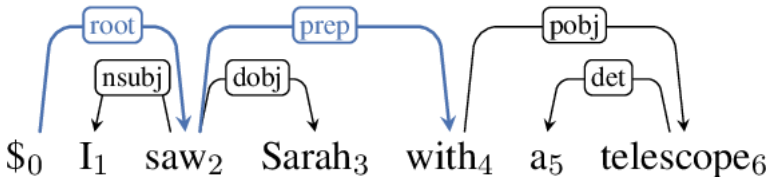



Figure 5.10: Dependency tagging tree from [50]: "An example full dependency tree. In the case of partial annotation, only some (not all) dependencies are annotated, for example, the two thick (blue) arcs."

scribed as text and passed as input to a pretrained dependency parsing model, which will create a structure like the one in the Figure 5.10. From here, the lemmatized version of the root will be extracted. In the example in figure, the result of this step would be the word *"see"*.

### 5.3.4 Classification

**Algorithm 4** Classification of audio data

```
function ClassEmbeddings(C)
    num_classes ← len(C)
    for i ∈ range(num_classes) do
        embedding_classes[i] = fastText.embedding(verb(c_i))
    end for
    return embedding_classes[i]
end function
function Classifications(r, C)
    num_classes ← len(C)
    embedding_classes ← ClassEmbeddings(C)
    embedded_r = fastText.embedding(r)
    max_sim = 0
    selected = None
    similarities = [0] * num_classes
    for i ∈ range(num_classes) do          ▷ Compute similarity as defined in Eq.5.4
        similarities[i] = cos_sim(embedded_r, embedding_classes[i])
        if similarities[i] > max_sim then
            max_sim ← similarities[i]
            selected ← C[i]
        end if
    end for
    similarities = softmax(similarities)
    return selected, similarities
end function
```

The input for this step, therefore, is the root verb extracted in the previous step. As mentioned above, computers are not able to analyse words as such. They need to convert them into a space of real numbers. This process, in NLP, is called Word Embedding, and is based on the distributional hypothesis, i.e. the idea that each word tends to be represented by the surrounding words. The general idea is to create, for each word in a dictionary, a vector of dimension $d_e$ which represents its syntactic and semantic characteristics. The mapping must be done so that similar words have similar vectors. There are different types of word-embedding: from "word-level" ones, which therefore analyse each word separately, without taking into account the context in which it is inserted, to "context-based" systems which provide different embeddings depending on the sentence in which the word is used. An example of the first type is Word2Vec [51], an example of the second type is Elmo [52].

This thesis made use of a pretrained model that would be able to embed words that were never encountered during the training process. FastText [53] is a word embedding model that matches these features: by saving an embedding dictionary of words and n-grams, it allows to obtain the embedding for each word through the combination of

the sub-elements. The used model was trained using CBOW with position-weights, in dimension 300, with character n-grams of length 5, a window of size 5 and 10 negatives.

The classification of the audio data is then performed as described in Alg. 4. Let us analyze it: for each class $c_i \in C$, an embedding vector of the corresponding verb has been defined using FastText: it is $w_{c_i} =$ `fastText.embedding(verb(`$c_i$`))` . Each time an audio input data is received, first the root extraction is performed as explained in previous subsection, obtaining $r$. Then, the embedding vector $w_i =$ `fastText.embedding(`$r$`)` is created using FastText and then the class is assigned for which the cosine similarity as defined in Eq.5.4 between the embedding of the root and the embedding of the class is maximum.

In the inner product space, the cosine similarity is the measure of the similarity between two vectors. By measuring the cosine of the angle between two vectors, this measure indicates whether two vectors point roughly in the same direction. An example of how this measure works is available in Figure 5.11.

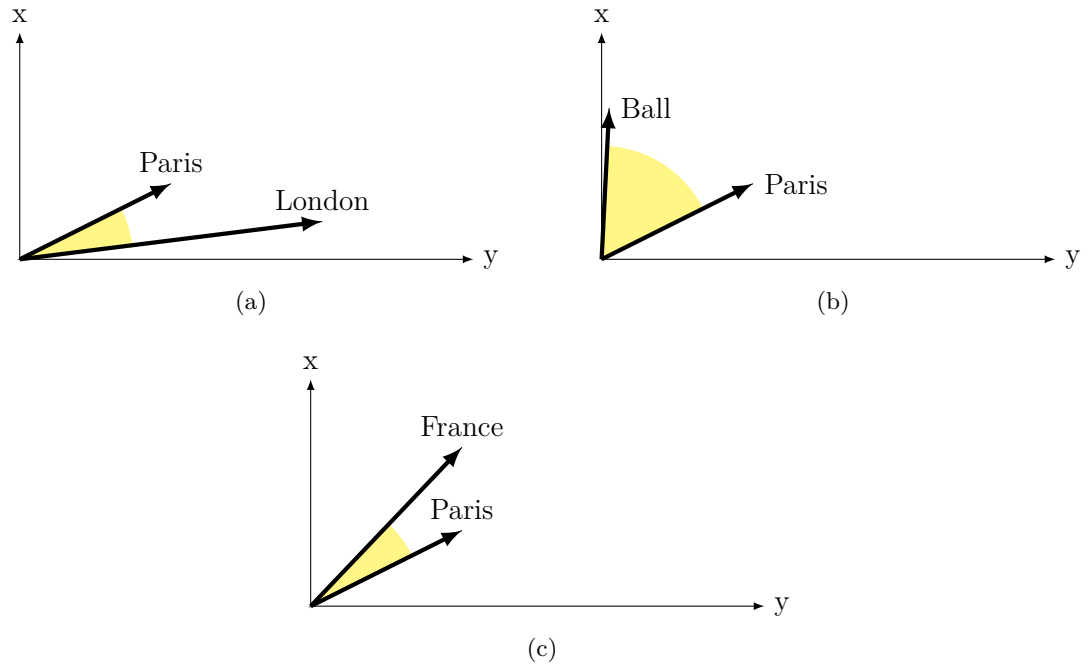



Figure 5.11: Three qualitative examples of how the cosine similarity on word embedding vectors works. From the example is clear how the similarity is higher in (a) and (c) and lower in (b)

The model retrieves both the assigned classes, for evaluating the Audio Model as itself, and the classification probabilities for each class, obtained by applying a softmax operation on the cosine similarity vector. This probability vector is used by the Fusion

Model described in Section 5.4.

$$\texttt{cos\_sim}(a, b) = \frac{\sum_i a_i * b_i}{\sqrt{\sum_i a_i^2} * \sqrt{\sum_i b_i^2}} \tag{5.4}$$

## 5.4 AVAR Fusion Model

We have seen so far how to analyze audio and visual data, by means of two models that take as input the necessary data and return, both, two probability vectors of dimension `num_classes` representing the classification of the model for that specific instance. We

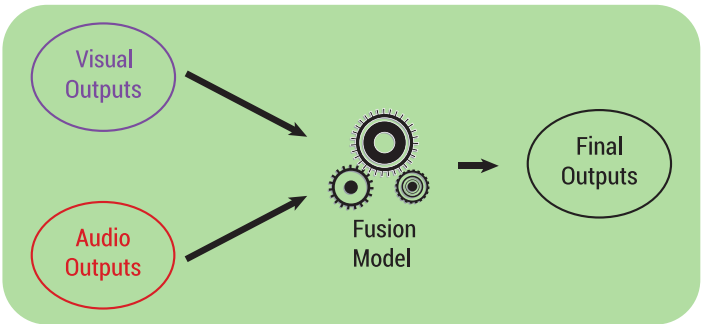



Figure 5.12: A

now analyze how the two outputs are merged to obtain a single final classification that takes into account both data sources. The chosen model learns weights for each class for both the audio ($w_a$) and the visual ($w_v$) part. Given the probabilities estimated from the audio model, $p_a$, and the probabilities estimated from the vision model, $p_v$, the fusion model acts as follows:

$$p_f = \texttt{softmax}(p_a * w_a + p_v * w_v)$$

where $p_f$ is the final probability vector. All the weights are initialized as ones and then the best value is learnt during the training.

## 5.5 Experiments

The AVAR model, as has been explained, consists of several parts, which need separate training and experiments to assess their effectiveness. In this section there is the presentation of which experiments were carried out, and in the next section results will be presented. The AVAR training process was performed using a 12GB NVIDIA Titan X GPU.

### 5.5.1 Evaluate Pose Refinement

In 5.2.3 it was presented the Pose refinement framework used on the OpenPose data. The purpose of this function, as mentioned, is to solve some of OpenPose's problems such as the

fact that it sees skeletons where they are not present, or that the representation stutters from one frame to another. In order to assess whether the function actually improves the data, two analyses were carried out: a qualitative one, which shows that the skeleton obtained from the refinement is more stable and the representation no longer suffers from "ghost skeletons" corresponding to objects with human features (such as a coat hanging on a hanger) or simple analysis errors by OpenPose.In order to assess whether the function actually improves the data, two analyses were carried out: the first is qualitative, showing that the skeleton obtained from the refinement is more stable and the representation no longer suffers from "phantom skeletons" corresponding to objects with human features (such as a coat hanging on a hanger) or simple analysis errors on the part of OpenPose. The second one is quantitative and measures the mean squared error (`MSE`) between the position estimated with OpenPose and the value of the joints obtained through a marker based model. The `MSE` formula is defined as follows, where $y_{true}$ has to be intended as the true coordinate value of each joint and $y_{pred}$ has to be inteded as the estimated coordinate value using OpenPose:

$$\texttt{MSE}(y_{true}, y_{pred}) = \sqrt{(y_{true}^2 - y_{pred}^2)}.$$

The marker based model used in this context is the master motor map (MMM) [54], a reference kinematics and dynamics model of the human body. The set of markers used are 56 markers that are derived from specific anatomical landmarks of the human body, see Figure 5.13 for reference. It is possible to see that there is not a direct correspondence between OpenPose joints and MMM markers. Because of that, it is not possible to

Figure 5.13: [54]: "Reference marker set used for whole-body human motion capture as proposed in the MMM framework."

directly apply the error computation from pairs of joint of different models. In order to have comparable models, a subset of joints from both representations was takes, selecting those that are more similar. From this analysis, it is clear that the refinement applied to the pose improves the performances. Results are available in 5.6.1

### 5.5.2 Training Visual Model

The central part of the Visual Model for AVAR is the graph neural network. The training of the network takes place through the backpropagation of the loss. Basically, the loss is a measure of how well a particular algorithm models a given data set. Loss function would result in a very large number if predictions deviated too much from actual results. Loss function allows to learn how to reduce the prediction error over time with the help of optimization functions. The aim of the training is to find the best combination of factors and hyper-parameters that guarantees the best final result. In order to define what the "best result" is, the evaluation metrics must first be defined. In classification problems, some standard metrics are accuracy, recall, precision, $F_1$ and confusion matrix. For simplicity, let us analyze a binary classification problem (positive class VS negative class), but the definitions easily extend to multi-class classification problem. A true positive $TP$ occurs if the model correctly predicts the positive class. A true negative $TN$ occurs if the model correctly predicts the negative class. There is, instead, a false positive $FP$ if the model incorrectly predicts the positive class, and a false negative $FN$ if the model incorrectly predicts the negative class. Starting from these definitions, the metrics are mathematically formulated as:

- Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$
- Precision = $\frac{TP}{TP+FP}$
- Recall = $\frac{TP}{TP+FN}$
- $F_1 = \frac{2*Precision*Recall}{Precision+Recall} = \frac{2*TP}{2*TP+FP+FN}$
- Confusion Matrix = it is a squared matrix in the number of classes, in which in the diagonal there are the $TP$ and in the other cells $(r, c)$ the number of element of class $r$, the row-class, predicted as elements of class $c$.

In this thesis the used evaluation metrics are accuracy and confusion matrix. Additionally, a relaxed version of the accuracy has been defined: instead of counting the $TP$, it counts the almost $TP$: basically, it takes the final output probabilities from the Visual Model and check if the True Class is in the top-3 of higher probabilities.

As previously said, two important aspects of the training process are the loss and the optimizer. Different methods and combinations have been tested before choosing the best one.

**Loss Functions** The Cross Entropy Loss (`CE`) is a loss for classification problems. Its formulation is:

$$\texttt{CE} = \sum_{c}^{C} y_c \log(\sigma(x_c))$$

where $C$ is the number of classes, $y_i$ is the ground truth, $\sigma$ is the activation function and $s_i$ is the result value obtained from the model.

A variation of the CE is the Binary Cross Entropy (BCE) Loss, that for this project is defined as follows:

$$\texttt{BCE} = \sum_{c}^{C} -w_c[y_c \log \sigma(x_c) + (1 - y_c) \log(1 - \sigma(x_c))]$$

where $y_c$ is the one-hot-encoded version of the ground truth, $x_c$ is the result value and $\sigma$ the activation function. In the thesis there is the distinction among weightedBCE, that uses the vale $w_c$ from the previous equation as weights for compensating the class imbalance, and the BCE that doesn't use the. The values of $w_c$ are set such that they are higher for the classes with fewer elements; in this way, elements from that classes result in having a higher loss and the model can focus more on learning them.

An alternative to CE for binary classification problems is the focal loss (FL); it applies a modulating term to the CE in order to focus learning on hard misclassified examples. The definition is:

$$\texttt{FL} = -(1 - \sigma(x_c))^{\gamma} \log(x_x)$$

**Optimizers** In order to reduce losses, optimization provides information about how to modify the weights and learning rates of neural networks. Optimizing is carried out by using the gradient and additional information, which are dependent on the specific optimization function chosen for the task at hand. The optimizer on which are based most of the others is the Gradient Descent (GD). The idea of GD is optimizing a function by finding a local minimum of a differentiable loss function. It looks for the direction of steepest descent and then takes repeated steps in that direction. The step direction is found at each step as being the opposite direction to the loss gradient at the current point. In this thesis, two modified and more modern versions of the classical SG were tested.

Stochastic Gradient Descent (SGD) updates all the parameters for each instance individually, instead of computing the gradient with respect to the whole training set. The equation used for updating is:

$$\theta = \theta - \mu \nabla_{\theta} J(\Theta; x, y)$$

where $\theta$ are the parameters to update (*e.g.* the weights), $\mu$ is the learning rate (*i.e.* the step dimension) and $\nabla$ is the gradient of $J$, the objective function.

Adaptive Moment Estimation (Adam) is an evolution of SGD. It uses momentum and adaptive learning rate, being momentum a method that accelerates the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients, and adaptive learning rate changes the value of the learning rate over time. The mathematical formulation is the following:

$$\theta = \theta - \frac{\mu}{\sqrt{\epsilon + \sum_{\tau=1}^{t} (\nabla(\theta_{\tau,i}))^2}} * \nabla J(\theta_{\tau,i})$$

**Training configurations** Starting from all that has been stated in this thesis, let us resume here the different variations in the GNN architecture configuration, in the training

parameters and in the hyperparameters, that has been tested. From the GNN point of view, the configurations used for the performed analysis are described in Table 5.2; the scope was measuring the impact that each of those function has on the final results. It has been noticed during the training that using sum or averaging has aggregating function has not noticeable impact on the final result, and so the analysis was carried out using the average function. On the other hand, Table 5.3 shows the values tested

| **Conf. ID** | **Encoder** | **Core** | | **Decoder** |
|---|---|---|---|---|
| | Encoding | Embedding | Aggregate | Decoding |
| **Conf1** | NMLP | NMLP | Avg | NMLP |
| **Conf2** | NMLP | NMLP | Sum | NMLP |
| **Conf3** | NMLP | TabCNN | Avg | NMLP |
| **Conf4** | TabCNN | TabCNN | Avg | TabCNN |

Table 5.2: The Encoder-Decoder GNN is defined from those four functions. Encoding, Embedding and Decoding functions can be either the Normalized Multilayer Perceptron (NMLP) or the CNN for Tabular Data (TabCNN). The Aggregate function can be either a Sum or an Average function.

for the hyperparameters of the model, understood as those parameters that define the architecture of the model; this process of finding the ideal architecture of the model is referred to as hyperparameter tuning. Regarding the `num_epochs` parameter, the final choice has been to use the higher value in Table 5.3 together with an implemented early stopping mechanism. In fact, training for too many epochs can lead to overfitting, but the use of early stopping addresses this problem. The basic idea under early stopping is to interrupt the training if the validation loss is not decreasing anymore (clear sign that the network is not learning). The training was carried out by dividing the dataset into 3 parts: training, evaluation and test splits. The first part, which contains 70% of the total elements, is used during the training phase. The second one, made of 20%, during the tuning phase to evaluate the effectiveness of the hyperparameters and finally the final results are evaluated on the third part, the 10% of elements left for the test set. The division was done in a stratified way, in order to reproduce in each subset the distribution of the classes of the complete set.

### 5.5.3 Training Fusion Model

The Fusion Model is a much simpler and smaller model of the GNN. In the implementation used for AVAR, it contains only $2 * \texttt{num\_classes}$ parameters, which correspond to the weights for the probability vector of the Audio Model and for those of the Visual Model. As already described for the training of the Visual Model, even for the Fusion Model it has been necessary to define the loss function and the optimizer. In this case, a `CE` loss was used, and an ADAM-type optimizer. For the hyperparameters point of view, a learning rate of 1e-3 has proven to be the best in this case.

As the two training, Visual Model and Fusion Model, were carried out at the same time, the same division into training, validation and test splits as in the Visual Model was

| Structure | Implementation | Hyperparameter | Value |
|---|---|---|---|
| GNN | | | |
| | general | n_frames | 10, 15 |
| | | n_processing_steps | 2, 3, 5 |
| | NMLP | n_neurons | 64, 128, 256 |
| | | n_layers | 1, 2, 3 |
| | | dropout | 0, 0.1, 0.2 |
| | TabCNN | dropout | 0, 0.1, 0.2 |
| Loss | | | |
| | general | regularization | 1e-1,1e-2, 1e-3 |
| Optimizer | | | |
| | general | lr | 1e-3, 1e-4, 1e-5 |
| | | num_epochs | 500, 1500, 2500, 5000 |
| | ADAM | $\beta_1$ | 0.85, 0.9 |
| | | $\beta_2$ | 0.9, 0.99 |

Table 5.3: Overview of tested hyperparameters for each element of the Visual Model during the training phase.

also used here. A difference lies in the number of epochs for which the model is trained. As already mentioned, the Fusion Model is very simple. Training it for a very large number of epochs leads to overfitting. For this reason, the training phase of the Fusion Model lasts only 100 epochs instead of 5000 (with possible early stopping) as defined for the Visual Model.

The inputs for the fusion model were, at each training step, the output probabilities of the Visual Model and the output probabilities of the Audio Model. As for the audio data, not having a specific dataset for this purpose, what was done was to manually create a sample set. For each activity in the dataset several instances were generated, using as possible verbs a variety of synonyms and different forms of the verb associated with the class, representing the actual variety of a real use situation.

## 5.6 Results

### 5.6.1 Results Pose Refinement

The selected joints for this experiment from the OpenPose representation and from the MMM representation, and the correspondent mapping, are available in Table 5.4. Given the imperfect correspondence between the joints in the two different representations, an analysis of the error on the individual joint rather than on the total model was preferred. In this way, it is possible to focus on the error difference between the OpenPose and the Refined pose instead that focusing on the magnitude of the total error. There are cases in which the joints from MMM and those from OpenPose almost overlap (*e.g.* CLAV and Neck). The table shows how, in those cases, the pose data is not completely reliable. But from these comparison is clear that the refinement, in most of the cases, improves the

| **OpenPose** | **MMM** | MSE(MMM,OpenPose) | MSE(MMM,Refined) | **DeltaError** |
|---|---|---|---|---|
| CLAV | Neck | 12.703 | 12.783 | −0.080 |
| RSHO | RShoulder | 10.211 | 10.326 | −0.115 |
| LSHO | LShoulder | 114.125 | 113.040 | 1.086 |
| LAOL | LElbow | 39.588 | 39.578 | 0.010 |
| RAOL | RElbow | 43.090 | 43.464 | −0.374 |
| LWTS | LWrist | 27.582 | 26.844 | 0.739 |
| RWTS | RWrist | 25.508 | 25.724 | −0.216 |
| RHIP | RHip | 18.565 | 18.041 | 0.524 |
| LHIP | LHip | 210.054 | 209.686 | 0.368 |
| RKNE | RKnee | 81.613 | 79.499 | 2.113 |
| LKNE | LKnee | 75.080 | 74.593 | 0.488 |
| L3 | MidHip | 43.000 | 42.956 | 0.044 |
| LHEE | LHeel | 9.851 | 9.932 | −0.081 |
| RHEE | RHeel | 11.828 | 11.499 | 0.329 |

Table 5.4: The first two columns represent the mapping among selected joints from the MMM representation and selected joints in the OpenPose representation. The third column represents the error obtained comparing the OpenPose result with the MMM values. The forth column represents the error obtained comparing the refined pose from OpenPose (Refined) with the MMM values. The last column is the delta among the OpenPose pose and the refined one

accuracy of the pose estimation. In those cases in which the **Delta Error** is negative, indicating an MSE(MMM,Refined) greater than an MSE(MMM,OpenPose), we can see that the values are very small, indicating that probably that point was already stable in the OpenPose prediction.

### 5.6.2 Results Visual Model

In this section, the results of the Visual Model experiments discussed above will be presented. The best parameter configurations turned out to be the following: for the network structure, the **Conf**4 from Table 5.2, that uses TabCNN for encoders and decoders, TabCNN as core embedding and average as aggregation function. The TabCNN used a dropout value of 5e-2. Moreover, 2 steps of message passing were performed. From the optimizer point of view, the used $lr$ was 1e-5 with an Adam optimizer, the loss was the CE with $L2$ regularization. Let's analyze the results obtained with this configuration. In Figure 5.14 it is shown a plot of the trend of the losses and accuracies during training, while in Figure 5.15 there is the confusion matrix of the Visual Model on the test test. We can clearly see that the Visual Model performs very well on all classes, with a high peak performance on the "Sweeping" class. Compared to the other activities represented in the dataset, the "Sweeping" involves more the lower body in the movement, so it makes sense that the networks better recognize it. Taking into account the relaxed accuracy introduced earlier, we notice that it is very high right from the start, which means that

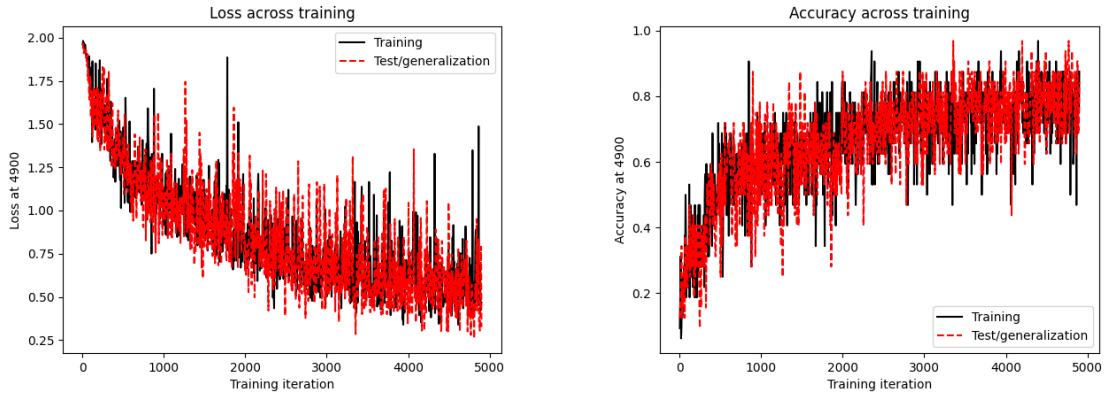



Figure 5.14: The loss and accuracy on the training and validation set for the Visual Model along the epochs of the training process.

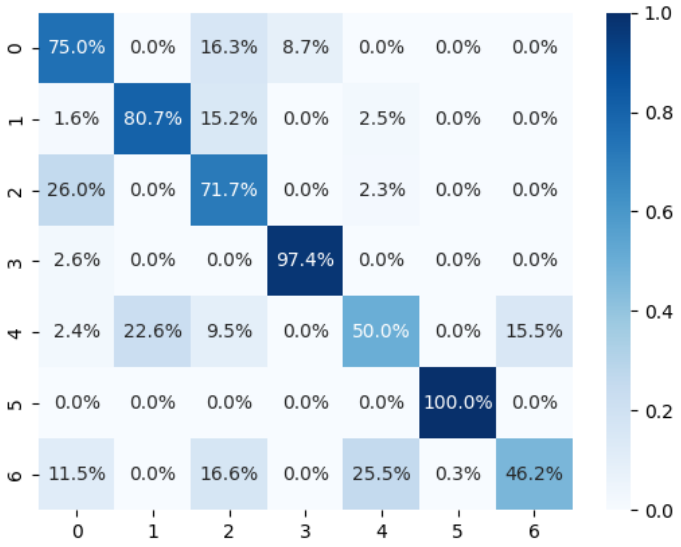



Figure 5.15: The confusion matrix for the Visual Model on the test set.

the model tends to muddle through on a few classes but you still manage to assign a high probability, though perhaps not always the highest, to the correct class. Another very interesting visualization to do is t-distributed stochastic neighbour embedding (t-SNE); this is a statistical method for visualizing high-density data that maps each point in a two- or three-dimensional space while preserving the proportional relative distance between points. Displaying this representation on the data output from the decoder allows us to verify that the model is actually learning parameters that correctly separate the

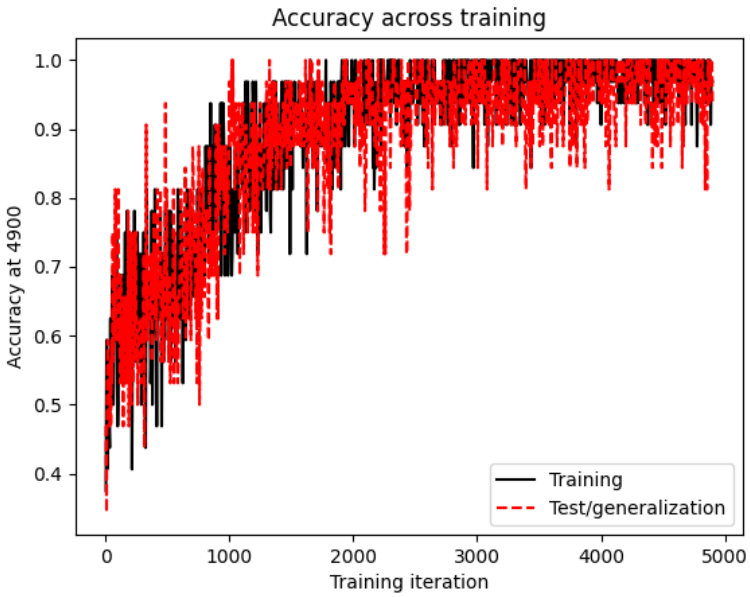



Figure 5.16: The relaxed accuracy on the training and validation set for the Visual Model along the epochs of the training process.

various classes. Since the model has been trained with 2 steps of message passing, we

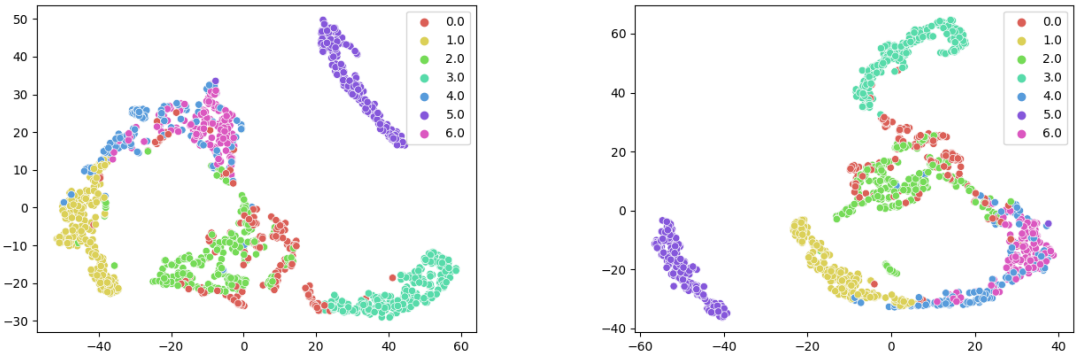



Figure 5.17: The t-SNE visualization of the output of the decoder model at each message passing step.

have two calls to the decoder and thus the possibility of displaying features at each message passing step. The plots are available in Figure 5.17. It is easy to highlight how the classes that are better separate from the others are the ones having better results in the confusion matrix in Figure 5.15. This makes perfectly sense and confirm that the network

is actually learning a correct representation for the data.

### 5.6.3 Results Fusion Model

As mentioned, the Fusion model takes as input the results of the Audio Model and the results (if any) of the Visual Model to provide a single final result. The training phase was carried out obtaining the results for loss and accuracy shown in Figure 5.18. Let us note that the two curves are very similar to those obtained from the Visual Model, as they are constructed from the results obtained with it and are therefore closely linked to it. From the results it is immediately evident that using this system improves the final

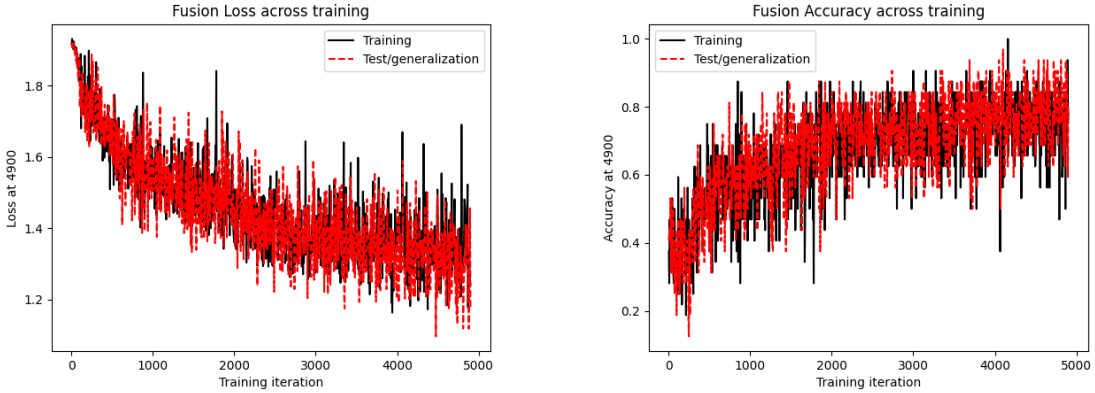



Figure 5.18: The training loss and training accuracy for the Fusion Model along the epochs.

results, especially for classes for which the Visual Model alone cannot produce very good results.

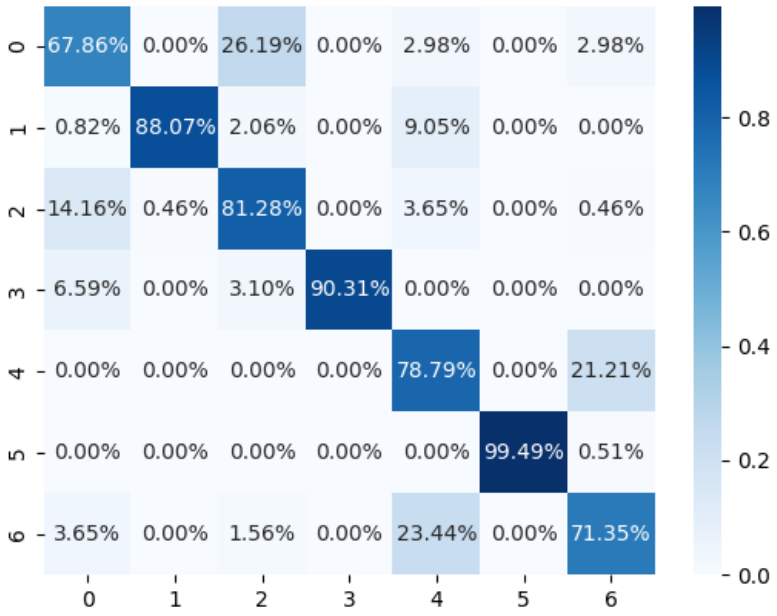



Figure 5.19: The confusion matrix for the Fusion Model on the test set.

# Chapter 6

# Discussion and Conclusions

## 6.1 Discussion

The AVAR model is fast and lightweight, in addition to offering excellent performance in terms of accuracy. AVAR actually does not require any input data that is not already available in the robot. Indeed, the OpenPose data in the ARMAR-6 robot are required for other tasks, which means, no matter whether or not AVAR is employed, it would have been produced regardless. Consequently, the model developed provides a human activity recognition system without burdening the robot's hardware too much further.

As explained above, various configurations of the Graph Neural Network used for the visual part of the network were tested. So far the results have been analysed in terms of inference. However, as the aim of this thesis is also to build an efficient model, we also analyse this aspect. Let us recall that **Conf1** corresponds to the configuration having NMLP for encoder, decoder and embedding model; **Conf3** refers to the configuration having NMLP for encoder and decoder and TabCNN as embedding function and finally **Conf4** has TabCNN for all the three. Table 6.1 shows the training time, inference time and weight for the three configurations. Training time is define as the time needed for the whole training process on the entire training set and validation set. Inference time, instead, is the average time needed for making the inference on a single instance of the test test. The variable weight refers to the MB occupied in the GPU memory for using the whole Vision Model.

| Model | Training Time | Inference Time | Weight |
|---|---|---|---|
| **Conf1** | 1h20min | 0.0026s | 3250MB |
| **Conf3** | 2h10min | 0.0029s | 3250MB |
| **Conf4** | 1h55min | 0.0031s | 3250MB |

Table 6.1: Performances comparison among the three versions of Graph Neural Networks used as Visual Model for the analysis of video data in AVAR. The Table shows the whole training time, the inference time and the model's weight in the memory for each configuration.

The use of audio data in combination with visual data has proven its effectiveness. Above all, it was noted that for some classes the use of audio data is more important than for others. In fact, some activities are quite similar to each other in terms of movements (Wipe and Mix, for example). In this case, the relaxed accuracy (calculated taking into account the first 3 higher probabilities) of the visual model, introduced in 5.5.2, is very high, but the model is not able to define the correct result. The use of audio data helps the model to resolve this confusion by selecting the actual activity performed.

## 6.2 Conclusions

In this thesis, the problem of Human Activity Recognition using video and audio data was addressed. The AVAR model consists of a visual data analysis model, an audio data analysis model and a fusion linear model for obtaining the final results. The Visual Model is based on a Graph Neural Network and has been trained on the KIT Bimanual Manipulation Dataset [45]; the Audio Model uses natural language processing techniques; the fusion model is a linear model that weighs the probabilities obtained from the two previous models.

The project provides results that fully meet the objectives. In addition to correctly classifying the activities seen, the model also proved to be light on real-time use on ARMAR-6. As mentioned above, the dataset used has some limitations due to the type of classes present: the focus on the whole body for activities that focus mainly on the upper body could in fact be a limitation. An interesting analysis could be to add attention mechanisms that allow to give more weight, during the analysis, to the joints more involved in the movement. This would make it easier to classify both total-body activities, such as walking, sweeping and the like, and activities more focused on one part of the body.

Another very interesting aspect for robotics could be developing an incremental learning scenario, i.e. making the robot able to learn new tasks without having to be trained again on all those already seen. This could be done by first recognizing a task as an unknown task, then by saving the relevant data and accessing data sources that can create a dataset to train the model on.

# Bibliography


[1] International Federation of Robotics. The significance of software in the evolution of robotics, 2021. https://ifr.org/post/the-significance-of-software-in-the-evolution-of-robotics.

[2] MS Windows NT h2t research - robots. https://h2t.anthropomatik.kit.edu/english/106.php. Accessed: 2022-03-27.

[3] ISO/IEC 2382:2015(E). Languages used in information technology. Standard, International Organization for Standardization, Geneva, CH, May 2015.

[4] Nikolaus Vahrenkamp, Mirko Wächter, Manfred Kröhnert, Kai Welke, and Tamim Asfour. The robot software framework armarx. *it - Information Technology*, 57(2): 99–111, 2015. doi: doi:10.1515/itit-2014-1066. URL https://doi.org/10.1515/itit-2014-1066.

[5] Tamim Asfour, Lukas Kaul, Mirko Wächter, Simon Ottenhaus, Pascal Weiner, Samuel Rader, Raphael Grimm, You Zhou, Markus Grotz, Fabian Paus, Dmitriy Shingarey, and Hans Haubert. Armar-6: A collaborative humanoid robot for industrial environments. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 447–454, 2018. doi: 10.1109/HUMANOIDS.2018.8624966.

[6] Suneth Ranasinghe, Fadi Al Machot, and Heinrich C Mayr. A review on applications of activity recognition systems with regard to performance and evaluation. *International Journal of Distributed Sensor Networks*, 12(8):1550147716665520, 2016. doi: 10.1177/1550147716665520. URL https://doi.org/10.1177/1550147716665520.

[7] Heidi M. Schambra, Avinash Parnandi, Natasha G. Pandit, Jasim Uddin, Audre Wirtanen, and Dawn M. Nilsen. A taxonomy of functional upper extremity motion. *Frontiers in Neurology*, 10, 2019. ISSN 1664-2295. doi: 10.3389/fneur.2019.00857. URL https://www.frontiersin.org/article/10.3389/fneur.2019.00857.

[8] Jiasen Lu, Ran Xu, and Jason J. Corso. Human action segmentation with hierarchical supervoxel consistency. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3762–3771, 2015. doi: 10.1109/CVPR.2015.7299000.

[9] Marta Sanzari, Valsamis Ntouskos, Simone Grazioso, Francesco Puja, and F. Pirri. Human motion primitive discovery and recognition. *ArXiv*, abs/1709.10494, 2017.

[10] C. R. G. Dreher, Mirko Wächter, and Tamim Asfour. Learning object-action relations from bimanual human demonstration using graph networks. *IEEE Robotics and Automation Letters*, 5(1):187–194, 2020. doi: 10.1109/LRA.2019.2949221.

[11] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012. URL http://arxiv.org/abs/1212.0402.

[12] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. Hmdb: A large video database for human motion recognition. In *2011 International Conference on Computer Vision*, pages 2556–2563, 2011. doi: 10.1109/ICCV.2011.6126543.

[13] Allison Janoch, Sergey Karayev, Yangqing Jia, Jonathan T. Barron, Mario Fritz, Kate Saenko, and Trevor Darrell. *A Category-Level 3D Object Dataset: Putting the Kinect to Work*, pages 141–165. Springer London, London, 2013. ISBN 978-1-4471-4640-7. doi: 10.1007/978-1-4471-4640-7_8. URL https://doi.org/10.1007/978-1-4471-4640-7_8.

[14] Tina, Anmol Kumar Sharma, Siddharth Tomar, and Kapil O. Gupta. Various approaches of human activity recognition: A review. *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, pages 1668–1676, 2021.

[15] Matteo Giuberti and Gianluigi Ferrari. *Motion Capture: From Radio Signals to Inertial Signals*, pages 791–812. 01 2015. doi: 10.1007/978-3-319-12817-7_34.

[16] Oscar D. Lara and Miguel A. Labrador. A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys Tutorials*, 15(3):1192–1209, 2013. doi: 10.1109/SURV.2012.110112.00192.

[17] Wenjie Ruan, Quan Z. Sheng, Lina Yao, Xue Li, Nickolas J.G. Falkner, and Lei Yang. Device-free human localization and tracking with uhf passive rfid tags: A data-driven approach. *Journal of Network and Computer Applications*, 104:78–96, 2018. ISSN 1084-8045. doi: https://doi.org/10.1016/j.jnca.2017.12.010. URL https://www.sciencedirect.com/science/article/pii/S1084804517304228.

[18] Florenc Demrozi, Graziano Pravadelli, Azra Bihorac, and Parisa Rashidi. Human activity recognition using inertial, physiological and environmental sensors: A comprehensive survey. *IEEE Access*, 8:210816–210836, 2020. doi: 10.1109/ACCESS.2020.3037715.

[19] Michalis Vrigkas, Christophoros Nikou, and Ioannis A. Kakadiaris. A review of human activity recognition methods. *Frontiers in Robotics and AI*, 2, 2015. ISSN 2296-9144. doi: 10.3389/frobt.2015.00028. URL https://www.frontiersin.org/article/10.3389/frobt.2015.00028.

[20] Ivan Laptev. On space-time interest points. *International Journal of Computer Vision*, 64:107–123, 2005.

[21] P. Dollar, V. Rabaud, G. Cottrell, and S. Belongie. Behavior recognition via sparse spatio-temporal features. In *2005 IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, pages 65–72, 2005. doi: 10.1109/VSPETS.2005.1570899.

[22] Geert Willems, Tinne Tuytelaars, and Luc Gool. An efficient dense and scale-invariant spatio-temporal interest point detector. In *Proceedings of the 10th European Conference on Computer Vision: Part II*, ECCV '08, page 650–663, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 9783540886853. doi: 10.1007/978-3-540-88688-4_48. URL https://doi.org/10.1007/978-3-540-88688-4_48.

[23] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Dense trajectories and motion boundary descriptors for action recognition. *International journal of computer vision*, 103(1):60–79, 2013.

[24] M.B. Holte, T.B. Moeslund, and P. Fihl. View-invariant gesture recognition using 3d optical flow and harmonic motion context. *Computer Vision and Image Understanding*, 114(12):1353–1361, 2010. ISSN 1077-3142. doi: https://doi.org/10.1016/j.cviu.2010.07.012. URL https://www.sciencedirect.com/science/article/pii/S1077314210001748. Special issue on Time-of-Flight Camera Based Computer Vision.

[25] Yong-Joong Kim, Bong-Nam Kang, and Daijin Kim. Hidden markov model ensemble for activity recognition using tri-axis accelerometer. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*, pages 3036–3041, 2015. doi: 10.1109/SMC.2015.528.

[26] Douglas L. Vail, Manuela M. Veloso, and John D. Lafferty. Conditional random fields for activity recognition. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9788190426275. doi: 10.1145/1329125.1329409. URL https://doi.org/10.1145/1329125.1329409.

[27] Maki Sugimoto, Thi Thi Zin, Takashi Toriu, and Shigeyoshi Nakajima. Robust rule-based method for human activity recognition. 2011.

[28] Holger Storf, Martin Becker, and Martin Riedl. Rule-based activity recognition framework: Challenges, technique and learning. In *2009 3rd International Conference on Pervasive Computing Technologies for Healthcare*, pages 1–7, 2009. doi: 10.4108/ICST.PERVASIVEHEALTH2009.6108.

[29] F. Niu and M. Abdel-Mottaleb. View-invariant human activity recognition based on shape and motion features. In *IEEE Sixth International Symposium on Multimedia Software Engineering*, pages 546–556, 2004. doi: 10.1109/MMSE.2004.88.

[30] S.X. Ju, M.J. Black, and Y. Yacoob. Cardboard people: a parameterized model of articulated image motion. In *Proceedings of the Second International Conference on Automatic Face and Gesture Recognition*, pages 38–44, 1996. doi: 10.1109/AFGR.1996.557241.

[31] Fabrizio Natola, Valsamis Ntouskos, Marta Sanzari, and F. Pirri. Bayesian non-parametric inference for manifold based mocap representation. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4606–4614, 2015.

[32] Ching-Hang Chen and Deva Ramanan. 3d human pose estimation = 2d pose estimation + matching. *CoRR*, abs/1612.06524, 2016. URL http://arxiv.org/abs/1612.06524.

[33] Zhe Cao, Gines Martinez, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP:1–1, 07 2019. doi: 10.1109/TPAMI.2019.2929257.

[34] Andrea Cherubini, Osama Mazhar, sofiane ramdani, Benjamin Navarro, and Robin Passama. Towards real-time physical human-robot interaction using skeleton information and hand gestures. 10 2018. doi: 10.1109/IROS.2018.8594385.

[35] Shijie Li, Jinhui Yi, Yazan Abu Farha, and Juergen Gall. Pose refinement graph convolutional network for skeleton-based action recognition. *IEEE Robotics and Automation Letters*, 6(2):1028–1035, 2021. doi: 10.1109/LRA.2021.3056361.

[36] Chenyang Si, Wentao Chen, Wei Wang, Liang Wang, and Tieniu Tan. An attention enhanced graph convolutional lstm network for skeleton-based action recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[37] Enea Cippitelli, Samuele Gasparrini, E. Gambi, and Susanna Spinsante. A human activity recognition system using skeleton data from rgbd sensors. *Computational Intelligence and Neuroscience*, 2016:1–14, 03 2016. doi: 10.1155/2016/4351435.

[38] Tasweer Ahmad, Huiyun Mao, Luojun Lin, and Guozhi Tang. Action recognition using attention-joints graph convolutional neural networks. *IEEE Access*, 8:305–313, 2020. doi: 10.1109/ACCESS.2019.2961770.

[39] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

[40] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32:4–24, 2019.

[41] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. 12 2013.

[42] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf.

[43] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[44] Peter Battaglia, Jessica Blake Chandler Hamrick, Victor Bapst, Alvaro Sanchez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andy Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Jayne Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv*, 2018. URL https://arxiv.org/pdf/1806.01261.pdf.

[45] Franziska Krebs, Andre Meixner, Isabel Patzer, and Tamim Asfour. The kit bimanual manipulation dataset. In *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, pages 499–506, 2021. doi: 10.1109/HUMANOIDS47582.2021.9555788.

[46] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[47] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience*, 2018:1–13, 02 2018. doi: 10.1155/2018/7068349.

[48] Yitan Zhu, Thomas Brettin, Fangfang Xia, Alexander Partin, Maulik Shukla, Hyunseung Yoo, Yvonne Evrard, James Doroshow, and Rick Stevens. Converting tabular data into images for deep learning with convolutional neural networks. *Scientific Reports*, 11, 05 2021. doi: 10.1038/s41598-021-90923-y.

[49] Daniel Jurafsky and James Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, volume 2. 02 2008.

[50] Xinyu Wang and Kewei Tu. Second-order neural dependency parsing with message passing and end-to-end training. *arXiv preprint arXiv:2010.05003*, 2020.

[51] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[52] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. 02 2018.

[53] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. ISSN 2307-387X.

[54] Christian Mandery, Oemer Terlemez, Martin Do, Nikolaus Vahrenkamp, and Tamim Asfour. Unifying representations and large-scale whole-body motion databases for studying human motion. *IEEE Transactions on Robotics*, 32:796–809, 08 2016. doi: 10.1109/TRO.2016.2572685.